中国移动
China Mobile

磐维数据库（PanWeiDB）V2.0-S3.0.1_B01

开发者指南

中移（动）信息技术有限公司

2024 年 9 月

【服务声明】

本文档意在向客户介绍中移信息全部或部分产品、服务的当时的整体概况，部分产品、服务的内容可能有所调整。您所购买的产品、服务的种类、服务标准等应由您与中移信息之间的商业合同约定，除非双方另有约定，否则，中移信息对本文档内容不做任何明示或模式的承诺或保证。

修订历史记录

序号	修订描述	修订日期	修订者
1	创建文档	2023-10-13	
2	合入新特性	2024-03-15	
3	合入新特性	2024-05-20	
4	合入新特性	2024-07-13	
5	文本修订	2024-07-16	
6	修复文档缺陷	2024-07-25	
7	修复文档缺陷	2024-08-15	
8	合入新特性	2024-09-25	

目 录

1. 应用开发

1.1.开发规范

1.1.1.数据库设计

对象命名规范

约束

数据库对象命名需要满足如下约束：

- 长度不超过 128 字符。
- 以字母或下划线开头，中间字符可以是字母、数字、下划线、$、#。
- 对象名使用双引号""，则可以包括合法字符的任意组合，例如："123abc"等。
- 对于 Oracle 兼容模式，对象名不区分大小写，使用双引号""包围可以区分大小写。
- 对于 MySQL 兼容模式，对象名区分大小写，可以通过 GUC 参数 lower_case_table_names 配置该项行为。

建议

- 避免使用保留或者非保留关键字命名数据库对象。可以使用 select * from pg_get_keywords()查询 PanWeiDB 的关键字。
- 避免使用双引号括起来的字符串来定义数据库对象名称。
- 数据库对象命名风格务必保持统一。
  - 增量开发的业务系统或进行业务迁移的系统，建议遵守历史的命名风格。
  - 建议使用多个单词组成，以下划线分割。
  - 数据库对象名称建议能够望文知意，尽量避免使用自定义缩写（可以使用通用的术语缩写进行命名）。例如，在命名中可以使用具有实际业务含义的英文词汇或汉语拼音，但规则应该在数据库实例范围内保持一致。
  - 变量名的关键是要具有描述性，即变量名称要有一定的意义，变量名要有前缀标明该变量的类型。

- 表对象的命名应该可以表征该表的重要特征。例如，在表对象命名时区分该表是普通表、临时表还是非日志表。
  - 普通表名按照数据集的业务含义命名。
  - 临时表以“tmp_+后缀”命名。
  - 非日志表以“ul_+后缀”命名。
  - 外表以“f_+后缀”命名。
  - 不创建以 redis_为前缀的数据库对象。
  - 不创建以 mlog_和以 matviewmap_为前缀的数据库对象。

【注意】
表对象的命名长度以不超过 15 个字符为宜(避免超过 20)。如果超过 128 字节，内核会对表名进行截断，从而造成和设置值不一致的现象。且在不同字符集下，可能造成字符被截断，出现预期外的字符。

创建数据库

规则

在实际业务中，根据需要创建新的 Database，不建议直接使用数据库实例默认的数据库。

建议

- 一个数据库实例内，用户自定义的 Database 数量建议不超过 3 个。
- 为了适应全球化的需求，使数据库编码能够存储与表示绝大多数的字符，建议创建 Database 的时候使用 UTF-8 编码。
- 创建 Database 时，需要重点关注字符集编码(ENCODING)配置项。

【注意】
Database 的 owner 默认拥有该 Database 下所有对象的所有权限，包括删除权限。删除权限影响较大，请谨慎使用。

创建用户

通过 CREATE USER 创建的用户，默认具有 LOGIN 权限。并且建用户的同时，系统会在执行该命令的数据库中，为该用户创建一个同名的 SCHEMA。

系统管理员在普通用户同名 schema 下创建的对象，所有者为 schema 的同名用户（非系统管理员）。

用户名称取值范围：字符串，要符合标识符的命名规范。且最大长度不超过 20 个字符。

用户登录密码规则如下：

- 取值范围：字符串。
- 密码默认不少于 8 个字符。
- 不能与用户名及用户名倒序相同。
- 至少包含大写字母（A-Z），小写字母（a-z），数字（0-9），非字母数字字符（限定为~!@#$%^&*()-_=+|[{}];:,<.>/?）四类字符中的三类字符。
- 密码也可以是符合格式要求的密文字符串，这种情况主要用于用户数据导入场景，不推荐用户直接使用。如果直接使用密文密码，用户需要知道密文密码对应的明文，并且保证明文密码复杂度，数据库不会校验密文密码复杂度，直接使用密文密码的安全性由用户保证。
- 创建用户时，应当使用双引号或单引号将用户密码括起来。

权限相关

数据库对象创建后，进行对象创建的用户就是该对象的所有者。数据库安装后的默认情况下，未开启三权分立，数据库系统管理员具有与对象所有者相同的权限。也就是说对象创建后，默认只有对象所有者或者系统管理员可以查询、修改和销毁对象，以及通过 GRANT 将对象的权限授予其他用户。

为使其他用户能够使用对象，必须向用户或包含该用户的角色授予必要的权限。PanWeiDB 支持以下的权限：SELECT、INSERT、UPDATE、DELETE、TRUNCATE、REFERENCES、CREATE、CONNECT、EXECUTE、USAGE、ALTER、DROP、COMMENT、INDEX 和 VACUUM。不同的权限与不同的对象类型关联。

要撤销已经授予的权限，可以使用 REVOKE。对象所有者的权限（例如 ALTER、 DROP、COMMENT、INDEX、VACUUM、GRANT 和 REVOKE）是隐式拥有的，即只要拥有对象就可以执行对象所有者的这些隐式权限。对象所有者可以撤销自己的普通权限，例如，使表对自己以及其他人只读，系统管理员用户除外。

标识了需要系统管理员权限的系统表和系统视图只有系统管理员可以查询。其他系统表和系统视图对所有用户可见。

示例

创建用户 kfst，并对其授予 schema 为 mytest 下表 t 的访问权限：

```
create user kfst identified by 'Kfst@2022';
grant usage on mytest to kfst;
grant select on mytest.t to kfst
```

创建表

总体上讲，良好的表设计需要遵守的规则：

- 规划好表结构设计，避免添加字段、修改字段类型或长度，表字段命名不超过 63 个字符。
- 在设计时尽量包含两个日期字段:created(创建日期)，modified(修改日期)且非空,对表的记录进行更新的时候，必须包含对 modified 字段的更新。
- 尽可能使用简单数据类型，不要使用类似数组或者嵌套表这种复杂类型。
- 必须要有主键，且尽量不要使用有实际意义的字段做主键。
- 需要 join 的字段，数据类型保持绝对一致（否则无法使用索引）。
- 当字段的类型为枚举型或布尔型时，建议使用 char(1)类型。
- 频繁更新使用的表应该单独放在存储性能好的表空间。
- 数据量超过亿级或占用磁盘超过 10GB 的表，建议考虑分区。
- 必须为表添加 comment 注释信息。
- 同一个模块的表尽可能使用相同的前缀，表名尽可能表达含义，临时或备份的数据库对象名，如 table,建议添加日期。
- 减少需要扫描的数据量。通过分区表的剪枝机制可以大幅减少数据的扫描量。
- 尽量减少随机 I/O。通过聚簇/局部聚簇可以实现热数据的连续存储，将随机 I/O 转换为连续 I/O，从而减少扫描的 I/O 代价。

存储方案的选择

表的存储类型是表定义设计的第一步，客户业务类型是决定表的存储类型的主要因素，表存储类型的选择依据请参考下表：

存储类型	适用场景
行存	❖ 点查询（返回记录少，基于索引的简单查询）。

	❖ 增、删、改操作较多的场景。
列存	❖ 统计分析类查询（关联、分组较多的场景）。 ❖ 即时查询（查询条件不确定，行存表扫描难以使用索引）。

分区方案的选择

当表中的数据量很大时，应当对表进行分区，一般需要遵循以下原则：

❖ 使用具有明显区间性的字段进行分区，比如日期、区域等字段上建立分区。

❖ 分区名称应当体现分区的数据特征。例如，关键字+区间特征。

❖ 将分区上边界的分区值定义为 MAXVALUE，以防止可能出现的数据溢出。

表的分区方式及使用场景如下：

分区方式	描述
range	通过范围进行分区。
list	通过指定列，按照具体的值进行分区。
interval	通过范围进行分区，超出范围的会自动根据间隔创建新的分区。
hash	通过 hash 散列的方式进行分区。

典型的分区表定义如下：

❖ 创建 range 分区表。

```
CREATE TABLE staffS_p1
(
 staff_ID       NUMBER(6) not null,
 FIRST_NAME     VARCHAR2(20),
 LAST_NAME      VARCHAR2(25),
 EMAIL          VARCHAR2(25),
 PHONE_NUMBER   VARCHAR2(20),
 HIRE_DATE      DATE,
 employment_ID  VARCHAR2(10),
 SALARY         NUMBER(8,2),
 COMMISSION_PCT NUMBER(4,2),
 MANAGER_ID     NUMBER(6),
 section_ID     NUMBER(4)
)
PARTITION BY RANGE (HIRE_DATE)
(
  PARTITION HIRE_19950501 VALUES LESS THAN ('1995-05-01 00:00:00'),
```

```
  PARTITION HIRE_19950502 VALUES LESS THAN ('1995-05-02 00:00:00'),
  PARTITION HIRE_maxvalue VALUES LESS THAN (MAXVALUE)
);
```

❖ 创建 Interval 分区表，初始两个分区，插入分区范围外的数据会自动新增分区。

```
CREATE TABLE sales
(prod_id NUMBER(6),
 cust_id NUMBER,
 time_id DATE,
 channel_id CHAR(1),
 promo_id NUMBER(6),
 quantity_sold NUMBER(3),
 amount_sold NUMBER(10,2)
)
PARTITION BY RANGE (time_id)
INTERVAL('1 day')
( PARTITION p1 VALUES LESS THAN ('2019-02-01 00:00:00'),
  PARTITION p2 VALUES LESS THAN ('2019-02-02 00:00:00')
);
```

❖ 创建 List 分区表。

```
CREATE TABLE test_list (col1 int, col2 int)
partition by list(col1)
(
partition p1 values (2000),
partition p2 values (3000),
partition p3 values (4000),
partition p4 values (5000)
);
```

❖ 创建 Hash 分区表。

```
CREATE TABLE test_hash (col1 int, col2 int)
partition by hash(col1)
(
partition p1,
partition p2
```

```
);
```

创建索引

创建索引遵循如下规则：

- 频繁 DML 操作的表索引数量不建议超过 5 个。
- 复合索引的字段数不建议超过 3 个。
- 复合索引第一个字段是常用检索条件。
- 复合索引第一个字段不应存在单字段索引。
- 真正创建索引前可以使用虚拟索引确定索引的有效性。
- 分区表索引分为 LOCAL 索引与 GLOBAL 索引，一个 LOCAL 索引对应一个具体分区，而 GLOBAL 索引则对应整个分区表。

一般来说，应该在这些列上创建索引：

- 在经常需要搜索查询的列上创建索引，可以加快搜索的速度。
- 经常用作表连接的字段上，建立索引。
- 在经常需要根据范围进行搜索的列上创建索引，因为索引已经排序，其指定的范围是连续的。
- 在经常需要排序的列上创建索引，因为索引已经排序，这样查询可以利用索引的排序，加快排序查询时间。
- 在经常使用 WHERE 子句的列上创建索引，加快条件的判断速度。
- 为经常出现在关键字 ORDER BY、GROUP BY、DISTINCT 后面的字段建立索引。

不应该创建索引的这些列具有下列特点：

- 对于那些在查询中很少使用或者参考的列不应该创建索引。既然这些列很少使用到，因此有索引或者无索引并不能提高查询速度。相反，由于增加了索引，反而降低了系统的维护速度和增大了空间需求。
- 对于那些只有很少数据值的列也不应该增加索引。由于这些列的取值很少，在查询的结果中，结果集的数据行占了表中数据行的很大比例，即需要在表中搜索的数据行的比例很大。这种场景下增加索引并不能明显加快检索速度。
- 对于那些定义为 text, image 和 bit 数据类型的列不应该增加索引。这些列的数据量要么相当大，要么取值很少。

- 当修改性能远远大于检索性能时，不应该创建索引。修改性能和检索性能是互相矛盾的。当增加索引时，会提高检索性能，但是会降低修改性能。当减少索引时，会提高修改性能，降低检索性能。

1.1.2.SQL 开发规范

DDL 规范

- 在代码中不允许出现任何 DDL 语句。
- DDL 语句一律由开发 DBA 编写并交由运维 DBA 统一执行，例：CREATE、ALTER、DROP。
- 所有生产 DDL 操作必须经过开发测试环境验证。
- 在批处理作业中尽量避免 DDL 操作。
- 避免大量并发事务对性能的影响。
- 索引字段的总长度不超过 50 字节。否则，索引大小会膨胀比较严重，带来较大的存储开销，同时索引性能也会下降。

DML 规范

- 更新数据的 SQL 语句禁止出现 where 1=1。
- 清空表中的数据时，应使用 truncate。
- 对于风险性较高的操作，应该显式地开启事务，确认无误后再提交，例：

```
begin;
update ...
commit;
end;
```

- 事务中 SQL 逻辑尽量简单，操作执行完后要及时提交，避免 idle in transaction 状态。

DQL 规范

1、请不要写 select *这样的代码，指定需要的字段名。

- 错误写法：

```
select * from test a;
```

- 正确的写法：

```
select a.id,a.name from test a;
```

2、写 SQL 的时候一定要使用绑定变量。对于极少数情况下不使用绑定变量提高性能，使用之前一定要和 DBA 沟通

3、写 SQL 的时候一定要给每个字段指定表名做前缀。带来性能的提升同时可以避免因 schema 切换导致访问到非预期的表。

❖ 错误写法：

```
select id,name from test a;
```

❖ 正确的写法：

```
select a.id,a.name from test a;
```

4、避免在 where 子句中对字段施加函数。

❖ 不允许在字段上添加函数或者表达式，这样将导致索引失效。

➢ 错误的写法：

```
select *
from iw_account_log
where to_char(trans_dt, 'yyyy-mm-dd') = '2007-04-04';
select qty from product where p_id + 12 = 168;
```

➢ 正确的写法：

```
select *
from iw_account_log
where trans_dt>= to_date('2007-04-04', 'yyyy-mm-dd')
and trans_dt<to_date('2007-04-05', 'yyyy-mm-dd');
select qty from product where p_id = 168 - 12;
```

❖ 如果是业务要求，但需要在编写时咨询 DBA。

❖ 当表连接时，用于连接的两个表的字段如果数据类型不一致，则必须在一边加上类型转换的函数。

➢ 错误的写法（a.id 是 number 类型，而 b.operator_number 是 char 类型）：

```
select count
 from adm_user a, adm_action_log b
where a.id = b.operator_number
  and a.username = '小钗';
```

➢ 正确的写法：

```
select count
from adm_usera, adm_action_logb
```

```
where to_char(a.id) = b.operator_number
and a.username = '小钗';

select count
from adm_usera, adm_action_logb
where a.id = to_number(b.operator_number)
and a.username = '小钗';
```

5、限制 join 的数量，不建议超过 3 个。

超过 3 张表或视图进行关联（特别是 FULL JOIN）时，执行代价难以估算。建议使用 WITH AS 语句创建中间临时表的方式增加 SQL 语句的可读性。例：

```
WITH t1 AS (
select ...
), t2 AS (
select ...
)
SELECT * FROM t1,t2 ...;
```

6、需要统计表中所有记录数时，不要使用 count(col)来替代 count(*)。count(*)会统计 NULL 值（真实行数），而 count(col)不会统计，可能造成数据失真。

7、全模糊查询无法使用 INDEX，应当尽可能避免。例如：

```
select * from table where name like '%jacky%';
```

8、严格要求使用正确类型的变量，杜绝数据库做隐式类型转换的情况。推荐在 sqlmap 的变量中指定变量的数据类型。例如：

```
select * from iw_user where iw_user_id = #userid:VARCHAR#
```

对于变量数据类型错误导致 SQL 严重性能问题的，按严重的编码错误 Bug 处理。

1.1.3.应用程序开发规范

如果用户在 APP 的开发中，使用了连接池机制，那么需要遵循如下规范：

- 如果在连接中设置了 GUC 参数，那么在将连接归还连接池之前，必须使用如下命令，将连接的状态清空。

```
SET SESSION AUTHORIZATION DEFAULT;
```

```
RESET ALL;
```

- 如果使用了临时表，那么在将连接归还连接池之前，必须将临时表删除。否则，连接池里面的连接就是有状态的，会对用户后续使用连接池进行操作的正确性带来影响。

2. 字符集

2.1.字符集列表

下表显示了可以在 PanWeiDB 中支持使用的字符集：

名称	描述	语言	服务端	字节-字符	别名
BIG5	Big Five	繁体中文	否	1-2	WIN950, Windows950
EUC_CN	扩展 UNIX 编码-中国	简体中文	是	1-3	
EUC_JP	扩展 UNIX 编码-日本	日文	是	1-3	
EUC_JIS_2004	扩展 UNIX 编码-日本, JIS X 0213	日文	是	1-3	
EUC_KR	扩展 UNIX 编码-韩国	韩文	是	1-3	
EUC_TW	扩展 UNIX 编码-台湾	繁体中文，台湾话	是	1-3	
GB18030	国家标准	中文	是	1-4	
GB18030-2022	国家标准	中文	是	1-4	
GBK	扩展国家标准	简体中文	是	1-2	WIN936, Windows936

名称	描述	语言	服务端	字节-字符	别名
ISO_8859_5	ISO 8859-5, ECMA 113	拉丁语/西里尔语	是	1	
ISO_8859_6	ISO 8859-6, ECMA 114	拉丁语/阿拉伯语	是	1	
ISO_8859_7	ISO 8859-7, ECMA 118	拉丁语/希腊语	是	1	
ISO_8859_8	ISO 8859-8, ECMA 121	拉丁语/希伯来语	是	1	
JOHAB	JOHAB	韩语	否	1-3	
KOI8R	KOI8-R	西里尔语（俄语）	是	1	KOI8
KOI8U	KOI8-U	西里尔语（乌克兰语）	是	1	
LATIN1	ISO 8859-1, ECMA 94	西欧	是	1	ISO88591
LATIN2	ISO 8859-2, ECMA 94	中欧	是	1	ISO88592
LATIN3	ISO 8859-3, ECMA 94	南欧	是	1	ISO88593
LATIN4	ISO 8859-4, ECMA 94	北欧	是	1	ISO88594
LATIN5	ISO 8859-9, ECMA 128	土耳其语	是	1	ISO88599
LATIN6	ISO 8859-10, ECMA 144	日耳曼语	是	1	ISO885910

名称	描述	语言	服务端	字节-字符	别名
LATIN7	ISO 8859-13	波罗的海	是	1	ISO885913
LATIN8	ISO 8859-14	凯尔特语	是	1	ISO885914
LATIN9	ISO 8859-15	带欧罗巴和口音的LATIN1	是	1	ISO885915
LATIN10	ISO 8859-16, ASRO SR 14111	罗马尼亚语	是	1	ISO885916
MULE_INTERNAL	Mule 内部编码	多语种编辑器	是	1-4	
SJIS	Shift JIS	日语	否	1-2	Mskanji, ShiftJIS, WIN932, Windows932
SHIFT_JIS_2004	Shift JIS, JIS X 0213	日语	否	1-2	
SQL_ASCII	未指定（见下文）	任意	是	1	
UHC	统一韩语编码	韩语	否	1-2	WIN949, Windows949
UTF8	Unicode, 8-bit	所有	是	1-4	Unicode
WIN866	Windows CP866	西里尔语	是	1	ALT
WIN874	Windows	泰语	是	1	

名称	描述	语言	服务端	字节-字符	别名
	CP874				
WIN1250250	Windows CP1250	中欧	是	1	
WIN1251	Windows CP1251	西里尔语	是	1	WIN
WIN1252	Windows CP1252	西欧	是	1	
WIN1253	Windows CP1253	希腊语	是	1	
WIN1254	Windows CP1254	土耳其语	是	1	
WIN1255	Windows CP1255	希伯来语	是	1	
WIN1256	Windows CP1256	阿拉伯语	是	1	
WIN1257	Windows CP1257	波罗的海	是	1	
WIN1258	Windows CP1258	越南语	是	1	ABC, TCVN, TCVN5712, VSCII

【说明】
需要注意并非所有的客户端 API 都支持上面列出的字符集。

如果服务器字符集是 SQL_ASCII，服务器把字节值 0-127 根据 ASCII 标准解释，而字节值 128-255 则当作无法解析的字符。如果设置为 SQL_ASCII，

就不会有编码转换。因此，这个设置基本不是用来声明所使用的指定编码，因为这个声明会忽略编码。在大多数情况下，如果你使用了任何非 ASCII 数据，那么不要使用 SQL_ASCII，因为 PanWeiDB 将无法转换或者校验非 ASCII 字符。

2.2.字符集设置

用户可以在数据库创建时指定非缺省编码，但是指定的编码必须与所选的区域相兼容：

```
createdb -E GB18030 -T template0 --lc-collate=zh_CN.gb18030 --lc-ctype=zh_CN.gb18030 testdb
```

将创建一个使用 GB18030 字符集以及 zh_CN 区域的 testdb 数据库。另外一种实现方法是使用 SQL 命令：

```
CREATE DATABASE testdb WITH ENCODING 'GB18030' LC_COLLATE='zh_CN.gb18030' LC_CTYPE='zh_CN.gb18030' TEMPLATE=template0;
```

注意上述命令声明拷贝 template0 数据库。当拷贝任何其他数据库时，来自源数据库的编码和区域设置不能被改变，因为可能导致数据损坏。

数据库的编码存储在 pg_database 系统表中。可以用 gsql 的-l 选项或\l 命令列出这些编码。

3. Schema

Schema 又称作模式。通过管理 Schema，允许多个用户使用同一数据库而不相互干扰，可以将数据库对象组织成易于管理的逻辑组，同时便于将第三方应用添加到相应的 Schema 下而不引起冲突。

每个数据库包含一个或多个 Schema。数据库中的每个 Schema 包含表和其他类型的对象。数据库创建初始，默认具有一个名为 public 的 Schema，且所有用户都拥有此 Schema 的 usage 权限，只有系统管理员和初始化用户可以在 public Schema 下创建函数、存储过程和同义词对象，其他用户即使赋予 create 权限后也不可以创建上述三种对象。可以通过 Schema 分组数据库对象。Schema 类似于操作系统目录，但 Schema 不能嵌套。默认只有初始化用户可以在 pg_catalog 模式下创建对象。

相同的数据库对象名称可以应用在同一数据库的不同 Schema 中。例如，a_schema 和 b_schema 都可以包含名为 mytable 的表。具有所需权限的用户可以访问数据库的多个 Schema 中的对象。

CREATE USER 创建用户的同时，系统会在执行该命令的数据库中，为该用户创建一个同名的 SCHEMA。

数据库对象是创建在数据库搜索路径中的第一个 Schema 内的。有关默认情况下的第一个 Schema 情况及如何变更 Schema 顺序等更多信息，请参见"搜索路径"。

创建、修改和删除 Schema

- 要创建 Schema，请使用 CREATE SCHEMA（参考：SQL 语法参考->SQL 语法->CREATE SCHEMA 章节）。默认初始用户和系统管理员可以创建 Schema，其他用户需要具备数据库的 CREATE 权限才可以在该数据库中创建 Schema，赋权方式请参考：SQL 语法参考->SQL 语法->GRANT 章节中将数据库的访问权限赋予指定的用户或角色中的语法。
- 要更改 Schema 名称或者所有者，请使用 ALTER SCHEMA（参考：SQL 语法参考->SQL 语法->ALTER SCHEMA 章节）。Schema 所有者可以更改 Schema。

- 要删除 Schema 及其对象，请使用 DROP SCHEMA（参考：SQL 语法参考->SQL 语法->DROP SCHEMA 章节）。Schema 所有者可以删除 Schema。
- 要在 Schema 内创建表，请以 schema_name.table_name 格式创建表。不指定 schema_name 时，对象默认创建到"搜索路径"中的第一个 Schema 内。
- 要查看 Schema 所有者，请对系统表 PG_NAMESPACE 和 PG_USER 执行如下关联查询。语句中的 schema_name 请替换为实际要查找的 Schema 名称。

```
panweidb=# SELECT s.nspname,u.usename AS nspowner FROM pg_namespace s,
pg_user u WHERE nspname='schema_name' AND s.nspowner = u.usesysid;
```

- 要查看所有 Schema 的列表，请查询 PG_NAMESPACE 系统表。

```
panweidb=#  SELECT * FROM pg_namespace;
```

- 要查看属于某 Schema 下的表列表，请查询系统视图 PG_TABLES。例如，以下查询会返回 Schema PG_CATALOG 中的表列表。

```
panweidb=#  SELECT distinct(tablename),schemaname from pg_tables where sche
maname = 'pg_catalog';
```

搜索路径

搜索路径定义在 search_path 参数中（参考：GUC 参数说明->客户端连接缺省设置->语句行为章节），参数取值形式为采用逗号分隔的 Schema 名称列表。如果创建对象时未指定目标 Schema，则该对象会被添加到搜索路径中列出的第一个 Schema 中。当不同 Schema 中存在同名的对象时，查询对象未指定 Schema 的情况下，将从搜索路径中包含该对象的第一个 Schema 中返回对象。

- 要查看当前搜索路径，请使用 SHOW（参考：SQL 语法参考->SQL 语法->ALTER SCHEMA 章节）。

```
panweidb=#  SHOW SEARCH_PATH;
search_path
----------------
"$user",public
(1 row)
```

search_path 参数的默认值为："$user",public。$user 表示与当前会话用户名同名的 Schema 名，如果这样的模式不存在，$user 将被忽略。所以默认情况下，用户连接数据库后，如果数据库下存在同名 Schema，则对象会添加到同名 Schema 下，否则对象被添加到 Public Schema 下。

- 要更改当前会话的默认 Schema，请使用 SET 命令。
  执行如下命令将搜索路径设置为 myschema、public，首先搜索 myschema。

```
panweidb=# SET SEARCH_PATH TO myschema, public;
```

3.1.WDR Snapshot Schema

WDR Snasphot 在启动后（打开参数 enable_wdr_snapshot 参考：GUC 参数说明->系统性能快照），会在用户表空间"pg_default"、数据库"postgres"下新建 schema "snapshot"，用于持久化 WDR 快照数据。默认初始化用户或 monadmin 用户可以访问 Snapshot Schema。

根据参数 wdr_snapshot_retention_days（参考：GUC 参数说明->系统性能快照）来自动管理快照的生命周期。

WDR Snapshot 生成性能报告

功能描述

基于 WDR Snapshot 数据表汇总、统计并生成性能报告，默认初始化用户或监控管理员用户可以生成报告。

操作步骤

1、执行如下命令新建报告文件。

```
touch /home/panweidb/wdrTestNode.html
```

2、连接数据库。

3、执行如下命令查询已经生成的快照，以获取快照的 snapshot_id。

```
select * from snapshot.snapshot;
```

4、执行如下命令手动创建快照。数据库中只有一个快照或者需要查看在当前时间段数据库的监控数据，可以选择手动执行快照操作，该命令需要用户具有 sysadmin 权限。（可选）

```
select create_wdr_snapshot();
```

执行“cm_ctl query -Cdvi”，回显中“Central Coordinator State”下显示的信息即为 CCN 信息。

5、执行如下命令，在本地生成 HTML 格式的 WDR 报告。

（1）执行如下命令设置报告格式。\a: 不显示表行列符号， \t: 不显示列名，\o: 指定输出文件。

```
\a
\t
\o /home/panweidb/wdrTestNode.html
```

（2）执行如下命令，生成 HTML 格式的 WDR 报告。

```
select generate_wdr_report(begin_snap_id Oid, end_snap_id Oid, int report_type, i
nt report_scope, int node_name );
```

示例一：生成集群级别的报告：

```
select generate_wdr_report(1, 2, 'all', 'cluster',null);
```

示例二：生成某个节点的报告：

```
select generate_wdr_report(1, 2, 'all', 'node', pgxc_node_str()::cstring);
```

（3）执行如下命令关闭输出选项及格式化输出命令。

```
\o \a \t
```

6、在/home/panweidb/下根据需要查看 WDR 报告。

generate_wdr_report 函数参数说明如下表所示：

参数	说明	取值范围
begin_snap_id	查询时间段开始的 snapshot 的 id（表 snapshot.snapshot 中的 snapshot_id）。	-
end_snap_id	查询时间段结束 snapshot 的 id。默认	-

参数	说明	取值范围
	end_snap_id 大于 begin_snap_id（表 snapshot.snapshot 中的 snapshot_id）。	
report_type	指定生成 report 的类型。例如，summary/detail/all。	summary：汇总数据。detail：明细数据。all：包含 summary 和 detail。
report_scope	指定生成 report 的范围，可以为 cluster 或者 node。	cluster：数据库级别的信息。node：节点级别的信息。
node_name	在 report_scope 指定为 node 时，需要把该参数指定为对应节点的名称。（节点名称可以执行 select * from pg_node_env;查询）。在 report_scope 为 cluster 时，该值可以省略或者指定为空或 NULL。	node：PanWeiDB 中的节点名称。cluster：省略/空/NULL。

注意事项

- 需进入系统库查看快照信息。
- WDR Snapshot 启动（即参数 enable_wdr_snapshot 为 on 时），且快照数量大于等于 2。

示例

1、修改参数。

```
alter system set enable_wdr_snapshot=on;
```

2、创建报告文件。

```
touch  /home/panweidb/wdrTestNode.html
```

3、登录数据库后切换至系统库。

```
\c postgres
```

4、查询已经生成的快照。

```
select * from snapshot.snapshot;
```

结果显示如下：

```
snapshot_id |        start_ts          |          end_ts
------------+--------------------------------+--------------------------------
        432 | 2022-07-05 10:34:25.477173+08 | 2022-07-05 10:34:40.441504+08
        435 | 2022-07-05 13:20:23.230771+08 | 2022-07-05 13:20:38.65221+08
```

5、生成格式化性能报告 wdrTestNode.html。

```
\a \t \o /home/om/wdrTestNode.html
```

6、查询 node 节点名称

```
select * from pg_node_env;
```

结果如下所示，node_name 即为节点名称。

```
 node_name |  host   | process | port |       installpath        |        datapath         | lo
g_directory
-----------+-----------+---------+------+------------------------------+--------------
--------------+--------------
 node1     | localhost |  148191 | 5432 | /home/panweidb/local/panweidb | /home/p
anweidb/data/panweidb | pg_log
(1 row)
```

7、向性能报告 wdrTestNode.html 中写入数据。

```
select generate_wdr_report(432, 435, 'all', 'node', 'node1');
```

8、关闭性能报告 wdrTestNode.html。

```
postgres=# \o
```

9、生成格式化性能报告 wdrTestCluster.html。

```
postgres=# \o /home/om/wdrTestCluster.html
```

10、向格式化性能报告 wdrTestCluster.html 中写入数据。

```
select generate_wdr_report(432, 435, 'all', 'cluster');
```

11、关闭性能报告 wdrTestCluster.html。

```
postgres=# \o \a \t
```

3.1.1.WDR Snapshot 原信息表

须知

用户应该禁止对 Snapshot schema 下的表进行增删改等操作，人为对这些表的修改或破坏可能会导致 WDR 各种异常情况甚至 WDR 不可用。

3.1.1.1.SNAPSHOT

SNAPSHOT 表记录当前系统中存储的 WDR 快照数据的索引信息、开始时间和结束时间。只能在系统库中查询到结果，在用户库中无法查询。

表 1 SNAPSHOT 表属性

名称	类型	描述	示例
snapshot_id	bigint	WDR 快照序号。	1
start_ts	timestamp	WDR 快照的开始时间。	2019-12-28 17:11:27.423742+08
end_ts	timestamp	WDR 快照的结束时间。	2019-12-28 17:11:43.67726+08

3.1.1.2.SNAPSHOT.TABLES_SNAP_TIMESTAMP

TABLES_SNAP_TIMESTAMP 表记录所有存储的 WDR snapshot 中数据库、表对象、以及数据采集的开始和结束时间。

表 1 TABLES_SNAP_TIMESTAMP 表属性

名称	类型	描述	示例
snapshot_id	bigint	WDR 快照序号。	1
db_name	text	WDR snapshot 对应的 database。	tpcc1000
tablename	text	WDR snapshot 对应的 table。	snap_xc_statio_all_indexes
start_ts	timestamp	WDR 快照的开始时间。	2019-12-28

名称	类型	描述	示例
			17:11:27.425849+08
end_ts	timestamp	WDR 快照的结束时间。	2019-12-28 17:11:27.707398+08

3.1.2.WDR Snapshot 数据表

WDR Snapshot 数据表命名原则：snap_{源数据表}。

WDR Snapshot 数据表来源为 DBE_PERF Schema（参考：系统表和系统视图->schema->DBE_PERF Schema 章节）下的视图。

3.1.3.查看 WDR 报告

WDR 报表主要内容如下表所示：

表 1 WDR 报表主要内容

项目	描述
Database Stat	❖ 数数据库维度性能统计信息：事务，读写，行活动，写冲突，死锁等。 ❖ 数据库范围报表，仅 cluster 模式下可查看此报表。
Load Profile	❖ 数据库维度的性能统计信息：CPU 时间，DB 时间，逻辑读/物理读，IO 性能，登入登出，负载强度，负载性能表现等。 ❖ 数据库范围报表，仅 cluster 模式下可查看此报表。
Instance Efficiency Percentages	❖ 数据库级或者节点缓冲命中率。 ❖ 数据库、节点范围报表，cluster 模式和 node 模式下均可查看此报表。
Top 10 Events by Total Wait Time	❖ 最消耗时间的事件。 ❖ 节点范围报表，仅 node 模式下可查看此报表。
Wait Classes by Total Wait Time	❖ 最消耗时间的等待时间分类。 ❖ 节点范围报表，仅 node 模式下可查看此报表。

项目	描述
Host CPU	❖ 主机 CPU 消耗。 ❖ 节点范围报表，仅 node 模式下可查看此报表。
IO Profile	❖ 数据库或者节点维度的 IO 的使用情况。 ❖ 数据库、节点范围报表，cluster 模式和 node 模式下均可查看此报表。
Memory Statistics	❖ 内核内存使用分布。 ❖ 节点范围报表，仅 node 模式下可查看此报表。
Time Model	❖ 节点范围的语句的时间分布信息。 ❖ 节点范围报表，仅 node 模式下可查看此报表。
SQL Statistics	❖ SQL 语句各个维度性能统计，按以下维度排序展示：总时间、平均时间、CPU 耗时、返回的行数、扫描的行数、执行次数、逻辑读、物理读。 ❖ 数据库、节点范围报表，cluster 模式和 node 模式下均可查看此报表。
Wait Events	❖ 节点级别的等待事件的统计信息。 ❖ 节点范围报表，仅 node 模式下可查看此报表。
Cache IO Stats	❖ 用户的表、索引的 IO 的统计信息。 ❖ 数据库、节点范围报表，cluster 模式和 node 模式下均可查看此报表。
Utility status	❖ 复制槽和后台 checkpoint 的状态信息。 ❖ 节点范围报表，仅 node 模式下可查看此报表。
Object stats	❖ 表、索引维度的性能统计信息。 ❖ 数据库、节点范围报表，cluster 模式和 node 模式下均可查看此报表。
Configuration settings	❖ 节点配置。 ❖ 节点范围报表，仅 node 模式下可查看此报表。
SQL Detail	❖ SQL 语句文本详情。 ❖ 数据库、节点范围报表，cluster 模式和 node 模式下均

项目	描述
	可查看此报表。

3.1.3.1.Database Stat

Database Stat 列名称及描述如下表所示。

表 1 Database Stat 报表主要内容

列名称	描述
DB Name	数据库名称。
Backends	连接到该数据库的后端数。
Xact Commit	此数据库中已经提交的事务数。
Xact Rollback	此数据库中已经回滚的事务数。
Blks Read	在这个数据库中读取的磁盘块的数量。
Blks Hit	高速缓存中已经发现的磁盘块的次数。
Tuple Returned	顺序扫描的行数。
Tuple Fetched	随机扫描的行数。
Tuple Inserted	通过数据库查询插入的行数。
Tuple Updated	通过数据库查询更新的行数。
Tup Deleted	通过数据库查询删除的行数。
Conflicts	由于数据库恢复冲突取消的查询数量。
Temp Files	通过数据库查询创建的临时文件数量。
Temp Bytes	通过数据库查询写入临时文件的数据总量。
Deadlocks	在该数据库中检索的死锁数。
Blk Read Time	通过数据库后端读取数据文件块花费的时间，以毫秒计算。
Blk Write Time	通过数据库后端写入数据文件块花费的时间，以毫秒计算。
Stats Reset	重置当前状态统计的时间。

3.1.3.2.Load Profile

Load Profile 指标名称及描述如下表所示。

表 1 Load Profile 报表主要内容

指标名称	描述
DB Time(us)	作业运行的 elapse time 总和。

指标名称	描述
CPU Time(us)	作业运行的 CPU 时间总和。
Redo size(blocks)	产生的 WAL 的大小（块数）。
Logical read (blocks)	表或者索引文件的逻辑读（块数）。
Physical read (blocks)	表或者索引的物理读（块数）。
Physical write (blocks)	表或者索引的物理写（块数）。
Read IO requests	表或者索引的读次数。
Write IO requests	表或者索引的写次数。
Read IO (MB)	表或者索引的读大小（MB）。
Write IO (MB)	表或者索引的写大小（MB）。
Logons	登录次数。
Executes (SQL)	SQL 执行次数。
Rollbacks	回滚事务数。
Transactions	事务数。
SQL response time P95(us)	95%的 SQL 的响应时间。
SQL response time P80(us)	80%的 SQL 的响应时间。

3.1.3.3.Instance Efficiency Percentages

Instance Efficiency Percentages 指标名称及描述如下表所示。

表 1 Instance Efficiency Percentages 报表主要内容

指标名称	描述
Buffer Hit %	Buffer Pool 命中率。
Effective CPU %	CPU time 占 DB time 的比例。
WalWrite NoWait %	访问 WAL Buffer 的 event 次数占总 wait event 的比例。
Soft Parse %	软解析的次数占总的解析次数的比例。
Non-Parse CPU %	非 parse 的时间占执行总时间的比例。

3.1.3.4.Top 10 Events by Total Wait Time

Top 10 Events by Total Wait Time 报表记录了数据库最消耗时间的事件。该报表的列名称及描述如下表所示：

表 1 Top 10 Events by Total Wait Time 报表主要内容

列名称	描述
Event	Wait Event 名称。
Waits	wait 次数。
Total Wait Time(us)	总 wait 时间（微秒）。
Avg Wait Time(us)	平均 wait 时间（微秒）。
Type	Wait Event 类别： ❖ STATUS ❖ LWLOCK_EVENT ❖ LOCK_EVENT ❖ IO_EVENT

3.1.3.5.Wait Classes by Total Wait Time

Wait Classes by Total Wait Time 报表记录了数据库最消耗时间的等待事件分类。该报表的列名称及描述如下表所示：

表 1 Wait Classes by Total Wait Time 报表主要内容

列名称	描述
Type	Wait Event 类别。 ❖ STATUS ❖ LWLOCK_EVENT ❖ LOCK_EVENT ❖ IO_EVENT
Waits	wait 次数。
Total Wait Time(us)	总 Wait 时间（微秒）。
Avg Wait Time(us)	平均 Wait 时间（微秒）。

3.1.3.6.Host CPU

Host CPU 报表记录了主机 CPU 消耗的相关内容。该报表的列名称及描述如下表所示：

表 1 Host CPU 报表主要内容

列名称	描述

列名称	描述
Cpus	CPU 数量。
Cores	CPU 核数。
Sockets	CPU Sockets 数量。
Load Average Begin	开始 snapshot 的 Load Average 值。
Load Average End	结束 snapshot 的 Load Average 值。
%User	用户态在 CPU 时间上的占比。
%System	内核态在 CPU 时间上的占比。
%WIO	Wait IO 在 CPU 时间上的占比。
%Idle	空闲时间在 CPU 时间上的占比。

3.1.3.7.IO Profile

IO Profile 指标名称及描述如下表所示。

表 1 IO Profile 指标表主要内容

指标名称	描述
Database requests	Database IO 次数。
Database (MB)	Database IO 数据量。
Database (blocks)	Database IO 数据块。
Redo requests	Redo IO 次数。
Redo (MB)	Redo IO 量。

3.1.3.8.Memory Statistics

Memory Statistics 报表记录了内存的使用情况分布。通过该报表用户可以了解两次快照之间的数据库内存的变化情况。该报表的列名称及描述如下表所示：

表 1 Memory Statistics 报表主要内容

列名称	描述
Memory Type	内存的类型： ❖ shared_used_memory：已经使用共享内存大小（MB）。

列名称	描述
	❖ max_shared_memory：最大共享内存（MB）。 ❖ process_used_memory：进程已经使用内存（MB）。 ❖ max_process_memory：最大进程内存（MB）。
Begin(MB)	内存起始大小。
End(MB)	内存终止大小。

3.1.3.9.Time Model

Time Model 报表记录了数据库各种状态所消耗的时间。该报表的列名称及描述如下表所示：

表 1 Time Model 报表主要内容

列名称	描述
Stat Name	状态名称： ❖ DB_TIME：所有线程端到端的墙上时间（WALL TIME）消耗总和（单位：微秒）。 ❖ EXECUTION_TIME：消耗在执行器上的时间总和（单位：微秒)。 ❖ PL_EXECUTION_TIME：消耗在 PL/SQL 执行上的时间总和（单位：微秒）。 ❖ PLAN_TIME：消耗在执行计划生成上的时间总和（单位：微秒）。 ❖ PARSE_TIME：消耗在 SQL 解析上的时间总和（单位：微秒）。 ❖ PL_COMPILATION_TIME：消耗在 SQL 编译上的时间总和（单位：微秒）。 ❖ NET_SEND_TIME：消耗在网络发送上的时间总和（单位：微秒）。 ❖ REWRITE_TIME：消耗在查询重写上的时间总和（单位：微秒）。 ❖ DATA_IO_TIME：消耗在数据读写上的时间总和（单位：微秒）。 ❖ CPU_TIME： 所有线程 CPU 时间消耗总和（单位：微秒）。
Value(us)	消耗的时间总和（单位：微秒）。

3.1.3.10.SQL Statistics

SQL Statistics 列名称及描述如下表所示。

表 1 SQL Statistics 报表主要内容

列名称	描述
Unique SQL Id	归一化的 SQL ID。

列名称	描述
Node Name	节点名称。
User Name	用户名称。
Tuples Read	访问的元组数量。
Calls	调用次数。
Min Elapse Time(us)	最小执行时间（us）。
Max Elapse Time(us)	最大执行时间（us）。
Total Elapse Time(us)	总执行时间（us）。
Avg Elapse Time(us)	平均执行时间（us）。
Returned Rows	SELECT 返回行数。
Tuples Affected	Insert/Update/Delete 行数。
Logical Read	Buffer 逻辑读次数。
Physical Read	Buffer 物理读次数。
CPU Time(us)	CPU 时间（us）。
Data IO Time(us)	IO 上的时间花费（us）。
Sort Count	排序执行的次数。
Sort Time(us)	排序执行的时间（us）。
Sort Mem Used(KB)	排序过程中使用的 work memory 大小（KB）。
Sort Spill Count	排序过程中，若发生落盘，写文件的次数。
Sort Spill Size(KB)	排序过程中，若发生落盘，使用的文件大小（KB）。
Hash Count	hash 执行的次数。
Hash Time(us)	hash 执行的时间（us）。
Hash Mem Used(KB)	hash 过程中使用的 work memory 大小（KB）。
Hash Spill Count	hash 过程中，若发生落盘，写文件的次数。
Hash Spill Size(KB)	hash 过程中，若发生落盘，使用的文件大小（KB）。
SQL Text	归一化 SQL 字符串。

3.1.3.11.Wait Events

Wait Events 报表记录了节点级别的等待事件的统计信。该报表的列名称及描述如下表所示：

表 1 Wait Events 报表主要内容

列名称	描述
Type	Wait Event 类别名称：

列名称	描述
	❖ STATUS。 ❖ LWLOCK_EVENT。 ❖ LOCK_EVENT。 ❖ IO_EVENT。
Event	Wait Event 名称。
Total Wait Time (us)	总 Wait 时间（单位：微秒）。
Waits	总 Wait 次数。
Failed Waits	Wait 失败次数。
Avg Wait Time (us)	平均 Wait 时间（单位：微秒）。
Max Wait Time (us)	最大 Wait 时间（单位：微秒）。

3.1.3.12.Cache IO Stats

Cache IO Stats 包含 User table 和 User index 两张表，列名称及描述如下所示。

User table IO activity ordered by heap blks hit ratio

表 1 User table IO activity ordered by heap blks hit ratio 报表主要内容

列名称	描述
DB Name	Database 名称。
Schema Name	Schema 名称。
Table Name	Table 名称。
%Heap Blks Hit Ratio	此表的 Buffer Pool 命中率。
Heap Blks Read	该表中读取的磁盘块数。
Heap Blks Hit	此表缓存命中数。
Idx Blks Read	表中所有索引读取的磁盘块数。
Idx Blks Hit	表中所有索引命中缓存数。
Toast Blks Read	此表的 TOAST 表读取的磁盘块数（如果存在）。
Toast Blks Hit	此表的 TOAST 表命中缓冲区数（如果存在）。

列名称	描述
Tidx Blks Read	此表的 TOAST 表索引读取的磁盘块数（如果存在）。
Tidx Blks Hit	此表的 TOAST 表索引命中缓冲区数（如果存在）。

User index IO activity ordered by idx blks hit ratio

表 2 User index IO activity ordered by idx blks hit ratio 报表主要内容

列名称	描述
DB Name	Database 名称。
Schema Name	Schema 名称。
Table Name	Table 名称。
Index Name	Index 名称。
%Idx Blks Hit Ratio	Index 的命中率。
Idx Blks Read	所有索引读取的磁盘块数。
Idx Blks Hit	所有索引命中缓存数。

3.1.3.13.Utility status

Utility status 记录了复制槽和后台 checkpoint 的状态信息。包含 Replication slot 和 Replication stat 两张表，列名称及描述如下所示。

Replication slot

表 1 Replication slot 报表主要内容

列名称	描述
Slot Name	复制节点名。
Slot Type	复制节点类型。
DB Name	复制节点数据库名称。
Active	复制节点状态。
Xmin	复制节点事务标识。
Restart Lsn	复制节点的 Xlog 文件信息。
Dummy Standby	复制节点假备。

Replication stat

表 2 Replication stat 报表主要内容

列名称	描述
Thread Id	线程的 PID。
Usesys Id	用户系统 ID。
Usename	用户名称。
Application Name	应用程序。
Client Addr	客户端地址。
Client Hostname	客户端主机名。
Client Port	客户端端口。
Backend Start	程序起始时间。
State	日志复制状态。
Sender Sent Location	发送端发送日志位置。
Receiver Write Location	接收端 write 日志位置。
Receiver Flush Location	接收端 flush 日志位置。
Receiver Replay Location	接收端 replay 日志位置。
Sync Priority	同步优先级。
Sync State	同步状态。

3.1.3.14.Object stats

Object stats 包含 User Tables stats、User index stats 和 Bad lock stats 三张表，列名称及描述如下所示。

❖ **User Tables stats**

表 1 User Tables stats 报表主要内容

列名称	描述
DB Name	Database 名称。
Schema	Schema 名称。
Relname	Relation 名称。
Seq Scan	此表发起的顺序扫描数。
Seq Tup Read	顺序扫描抓取的活跃行数。
Index Scan	此表发起的索引扫描数。
Index Tup Fetch	索引扫描抓取的活跃行数。
Tuple Insert	插入行数。
Tuple Update	更新行数。

列名称	描述
Tuple Delete	删除行数。
Tuple Hot Update	HOT 更新行数（即没有更新所需的单独索引）。
Live Tuple	估计活跃行数。
Dead Tuple	估计死行数。
Last Vacuum	最后一次此表是手动清理的（不计算 VACUUM FULL）时间。
Last Autovacuum	上次被 autovacuum 守护进程清理的时间。
Last Analyze	上次手动分析这个表的时间。
Last Autoanalyze	上次被 autovacuum 守护进程分析的时间。
Vacuum Count	这个表被手动清理的次数（不计算 VACUUM FULL）。
Autovacuum Count	这个表被 autovacuum 清理的次数。
Analyze Count	这个表被手动分析的次数。
Autoanalyze Count	这个表被 autovacuum 守护进程分析的次数。

❖ **User index stats**

表 2 User index stats 报表主要内容

列名称	描述
DB Name	Database 名称。
Schema	Schema 名称。
Relname	Relation 名称。
Index Relname	Index 名称。
Index Scan	索引上开始的索引扫描数。
Index Tuple Read	通过索引上扫描返回的索引项数。
Index Tuple Fetch	通过使用索引的简单索引扫描抓取的表行数。

❖ **Bad lock stats**

表 3 Bad lock stats 报表主要内容

列名称	描述
DB Id	数据库的 OID。
Tablespace Id	表空间的 OID。
Relfilenode	文件对象 ID。
Fork Number	文件类型。

列名称	描述
Error Count	失败计数。
First Time	第一次发生时间。
Last Time	最近一次发生时间。

3.1.3.15.Configuration settings

Configuration settings 记录了数据库节点参数配置的相关信息。列名称及描述如下所示：

表 1 Configuration settings 报表主要内容

列名称	描述
Name	GUC 参数名称。
Abstract	GUC 参数描述。
Type	数据类型。
Curent Value	当前值。
Min Value	合法最小值。
Max Value	合法最大值。
Category	GUC 类别。
Enum Values	如果是枚举值，列举所有枚举值。
Default Value	数据库启动时参数默认值。
Reset Value	数据库重置时参数默认值。

3.1.3.16.SQL Detail

SQL Detail 列名称及描述如下表所示。

表 1 SQL Detail 报表主要内容

列名称	描述
Unique SQL Id	归一化 SQL ID。
User Name	用户名称。
Node Name	节点名称。Node 模式下不显示该字段。
SQL Text	归一化 SQL 文本。

3.2.DBE_PERF Schema

DBE_PERF Schema 内视图主要用来诊断性能问题，也是 WDR Snapshot 的数据来源。数据库安装后，默认只有初始用户具有模式 dbe_perf 的权限。若

是由旧版本升级而来，为保持权限的前向兼容，模式 dbe_perf 的权限与旧版本保持一致。从 OS、Instance、Memory 等多个维度划分组织视图，并且符合如下命名规范：

- GLOBAL_开头的视图，代表从数据库节点请求数据，并将数据追加对外返回，不会处理数据。
- SUMMARY_开头的视图，代表是将 PanWeiDB 内的数据概述，多数情况下是返回数据库节点（有时只有数据库主节点的）的数据，会对数据进行加工和汇聚。
- 非这两者开头的视图，一般代表本地视图，不会向其他数据库节点请求数据。

3.2.1.OS

3.2.1.1.OS_RUNTIME

显示当前操作系统运行的状态信息。

表 1 OS_RUNTIME 字段

名称	类型	描述
id	integer	编号。
name	text	操作系统运行状态名称。
value	numeric	操作系统运行状态值。
comments	text	操作系统运行状态注释。
cumulative	boolean	操作系统运行状态的值是否为累加值。

3.2.1.2.GLOBAL_OS_RUNTIME

提供 PanWeiDB 中所有正常节点下的操作系统运行状态信息。

表 1 GLOBAL_OS_RUNTIME 字段

名称	类型	描述
node_name	name	数据库进程名称。
id	integer	编号。
name	text	操作系统运行状态名称。
value	numeric	操作系统运行状态值。

名称	类型	描述
comments	text	操作系统运行状态注释。
cumulative	boolean	操作系统运行状态的值是否为累加值。

3.2.1.3.OS_THREADS

提供当前节点下所有线程的状态信息。

表 1 OS_THREADS 字段

名称	类型	描述
node_name	text	数据库进程名称。
pid	bigint	数据库进程中正在运行的线程号。
lwpid	integer	与 pid 对应的轻量级线程号。
thread_name	text	与 pid 对应的线程名称。
creation_time	timestamp with time zone	与 pid 对应的线程创建的时间。

3.2.1.4.GLOBAL_OS_THREADS

提供 PanWeiDB 中所有正常节点下的线程状态信息。

表 1 GLOBAL_OS_THREADS 字段

名称	类型	描述
node_name	text	数据库进程名称。
pid	bigint	当前节点进程中正在运行的线程号。
lwpid	integer	与 pid 对应的轻量级线程号。
thread_name	text	与 pid 对应的线程名称。
creation_time	timestamp with time zone	与 pid 对应的线程创建的时间。

3.2.2.Instance

3.2.2.1.INSTANCE_TIME

提供当前节点下的各种时间消耗信息，主要分为以下类型：

- DB_TIME：作业在多核下的有效时间花销。
- CPU_TIME：CPU 的时间花销。
- EXECUTION_TIME：执行器内的时间花销。

- PARSE_TIME：SQL 解析的时间花销。
- PLAN_TIME：生成 Plan 的时间花销。
- REWRITE_TIME：SQL 重写的时间花销。
- PL_EXECUTION_TIME：plpgsql（存储过程）执行的时间花销。
- PL_COMPILATION_TIME：plpgsql（存储过程）编译的时间花销。
- NET_SEND_TIME：网络上的时间花销。
- DATA_IO_TIME：IO 上的时间花销。

表 1 INSTANCE_TIME 字段

名称	类型	描述
stat_id	integer	统计编号。
stat_name	text	类型名称。
value	bigint	时间值（单位：微秒）。

3.2.2.2.GLOBAL_INSTANCE_TIME

提供 PanWeiDB 中所有正常节点下的各种时间消耗信息（时间类型见 instance_time 视图）。

表 1 GLOBAL_INSTANCE_TIME 字段

名称	类型	描述
node_name	name	数据库进程的名称。
stat_id	integer	统计编号。
stat_name	text	类型名称。
value	bigint	时间值（单位：微秒）。

3.2.3.Memory

3.2.3.1.MEMORY_NODE_DETAIL

显示某个数据库节点内存使用情况。

表 1 MEMORY_NODE_DETAIL 字段

名称	类型	描述
nodename	text	数据库进程名称。

名称	类型	描述
memorytype	text	内存使用的名称。 ❖ max_process_memory：PanWeiDB 实例所占用的内存大小。 ❖ process_used_memory：进程所使用的内存大小。 ❖ max_dynamic_memory：最大动态内存。 ❖ dynamic_used_memory：已使用的动态内存。 ❖ dynamic_peak_memory：内存的动态峰值。 ❖ dynamic_used_shrctx：最大动态共享内存上下文。 ❖ dynamic_peak_shrctx：共享内存上下文的动态峰值。 ❖ max_shared_memory：最大共享内存。 ❖ shared_used_memory：已使用的共享内存。 ❖ max_cstore_memory：列存所允许使用的最大内存。 ❖ cstore_used_memory：列存已使用的内存大小。 ❖ max_sctpcomm_memory：sctp 通信所允许使用的最大内存。 ❖ sctpcomm_used_memory：sctp 通信已使用的内存大小。 ❖ sctpcomm_peak_memory：sctp 通信的内存峰值。 ❖ other_used_memory：其他已使用的内存大小。 ❖ gpu_max_dynamic_memory：GPU 最大动态内存。 ❖ gpu_dynamic_used_memory：GPU 已使用的动态内存。 ❖ gpu_dynamic_peak_memory：GPU 内存的动态峰值。 ❖ pooler_conn_memory：链接池申请内存计数。 ❖ pooler_freeconn_memory：链接池空闲连接的内存计数。 ❖ storage_compress_memory：存储模块压缩使用的内存大小。 ❖ udf_reserved_memory：UDF 预留的内存大小。
memorymbytes	integer	内存使用的大小，单位为 MB。

3.2.3.2.GLOBAL_MEMORY_NODE_DETAIL

显示当前 PanWeiDB 中所有正常节点下的内存使用情况。

当 GUC 参数 enable_memory_limit=off 时，本视图不可使用。

表 1 GLOBAL_MEMORY_NODE_DETAIL 字段

名称	类型	描述
nodename	text	数据库进程名称。
memorytype	text	内存使用的名称。 ❖ max_process_memory：PanWeiDB 实例所占用的内存大小。 ❖ process_used_memory：进程所使用的内存大小。 ❖ max_dynamic_memory：最大动态内存。 ❖ dynamic_used_memory：已使用的动态内存。 ❖ dynamic_peak_memory：内存的动态峰值。 ❖ dynamic_used_shrctx：最大动态共享内存上下文。 ❖ dynamic_peak_shrctx：共享内存上下文的动态峰值。 ❖ max_shared_memory：最大共享内存。 shared_used_memory：已使用的共享内存。 ❖ max_cstore_memory：列存所允许使用的最大内存。 ❖ cstore_used_memory：列存已使用的内存大小。 ❖ max_sctpcomm_memory：sctp 通信所允许使用的最大内存。 ❖ sctpcomm_used_memory：sctp 通信已使用的内存大小。 ❖ sctpcomm_peak_memory：sctp 通信的内存峰值。 ❖ other_used_memory：其他已使用的内存大小。 ❖ gpu_max_dynamic_memory：GPU 最大动态内存。 ❖ gpu_dynamic_used_memory：GPU 已使用的动态内存。 ❖ gpu_dynamic_peak_memory：GPU 内存的动态峰值。

名称	类型	描述
		❖ pooler_conn_memory：链接池申请内存计数。 ❖ pooler_freeconn_memory：链接池空闲连接的内存计数。 ❖ storage_compress_memory：存储模块压缩使用的内存大小。 ❖ udf_reserved_memory：UDF 预留的内存大小。 ❖ min_dynamic_memory：最小动态内存。 ❖ max_backend_memory：后端进程可用最大内存。 backend_used_memory：后端进程使用的内存大小。
memorymbytes	integer	内存使用的大小，单位为 MB。

3.2.3.3.GS_SHARED_MEMORY_DETAIL

查询当前节点所有已产生的共享内存上下文的使用信息。

表 1 GS_SHARED_MEMORY_DETAIL 字段

名称	类型	描述
contextname	text	内存上下文的名称。
level	smallint	内存上下文的级别。
parent	text	上级内存上下文。
totalsize	bigint	共享内存总大小（单位：字节）。
freesize	bigint	共享内存剩余大小（单位：字节）。
usedsize	bigint	共享内存使用大小（单位：字节）。

3.2.3.4.GLOBAL_SHARED_MEMORY_DETAIL

查询 PanWeiDB 中所有正常节点下的共享内存上下文的使用信息。

表 1 GLOBAL_SHARED_MEMORY_DETAIL 字段

名称	类型	描述
node_name	name	数据库进程名称。
contextname	text	内存上下文的名称。
level	smallint	内存上下文的级别。
parent	text	上级内存上下文。
totalsize	bigint	共享内存总大小（单位：字节）。

名称	类型	描述
freesize	bigint	共享内存剩余大小（单位：字节）。
usedsize	bigint	共享内存使用大小（单位：字节）。

3.2.4.File

3.2.4.1.FILE_IOSTAT

通过对数据文件 IO 的统计，反映数据的 IO 性能，用以发现 IO 操作异常等性能问题。

表 1 FILE_IOSTAT 字段

名称	类型	描述
filenum	oid	文件标识。
dbid	oid	数据库标识。
spcid	oid	表空间标识。
phyrds	bigint	读物理文件的数目。
phywrts	bigint	写物理文件的数目。
phyblkrd	bigint	读物理文件块的数目。
phyblkwrt	bigint	写物理文件块的数目。
readtim	bigint	读文件的总时长（单位：微秒）。
writetim	bigint	写文件的总时长（单位：微秒）。
avgiotim	bigint	读写文件的平均时长（单位：微秒）。
lstiotim	bigint	最后一次读文件时长（单位：微秒）。
miniotim	bigint	读写文件的最小时长（单位：微秒）。
maxiowtm	bigint	读写文件的最大时长（单位：微秒）。

3.2.4.2.SUMMARY_FILE_IOSTAT

通过 PanWeiDB 内对数据文件汇聚 IO 的统计，反映数据的 IO 性能，用以发现 IO 操作异常等性能问题。

表 1 SUMMARY_FILE_IOSTAT 字段

名称	类型	描述
filenum	oid	文件标识。
dbid	oid	数据库标识。

名称	类型	描述
spcid	oid	表空间标识。
phyrds	numeric	读物理文件的数目。
phywrts	numeric	写物理文件的数目。
phyblkrd	numeric	读物理文件块的数目。
phyblkwrt	numeric	写物理文件块的数目。
readtim	numeric	读文件的总时长（单位：微秒）。
writetim	numeric	写文件的总时长（单位：微秒）。
avgiotim	bigint	读写文件的平均时长（单位：微秒）。
lstiotim	bigint	最后一次读文件时长（单位：微秒）。
miniotim	bigint	读写文件的最小时长（单位：微秒）。
maxiowtm	bigint	读写文件的最大时长（单位：微秒）。

3.2.4.3.GLOBAL_FILE_IOSTAT

得到所有节点上的数据文件 IO 统计信息。

表 1 GLOBAL_FILE_IOSTAT 字段

名称	类型	描述
node_name	name	数据库进程名称。
filenum	oid	文件标识。
dbid	oid	数据库标识。
spcid	oid	表空间标识。
phyrds	bigint	读物理文件的数目。
phywrts	bigint	写物理文件的数目。
phyblkrd	bigint	读物理文件块的数目。
phyblkwrt	bigint	写物理文件块的数目。
readtim	bigint	读文件的总时长（单位：微秒）。
writetim	bigint	写文件的总时长（单位：微秒）。
avgiotim	bigint	读写文件的平均时长（单位：微秒）。
lstiotim	bigint	最后一次读文件时长（单位：微秒）。
miniotim	bigint	读写文件的最小时长（单位：微秒）。
maxiowtm	bigint	读写文件的最大时长（单位：微秒）。

3.2.4.4.FILE_REDO_IOSTAT

本节点 Redo（WAL）相关的统计信息。

表 1 FILE_REDO_IOSTAT 字段

名称	类型	描述
phywrts	bigint	向 wal buffer 中写的次数。
phyblkwrt	bigint	向 wal buffer 中写的 block 的块数。
writetim	bigint	向 xlog 文件中写操作的时间（单位：微秒）。
avgiotim	bigint	平均写 xlog 的时间（writetim/phywrts）（单位：微秒）。
lstiotim	bigint	最后一次写 xlog 的时间（单位：微秒）。
miniotim	bigint	最小的写 xlog 时间（单位：微秒）。
maxiowtm	bigint	最大的写 xlog 时间（单位：微秒）。

3.2.4.5.SUMMARY_FILE_REDO_IOSTAT

PanWeiDB 内汇聚所有的 Redo（WAL）相关的统计信息。

表 1 SUMMARY_FILE_REDO_IOSTAT 字段

名称	类型	描述
phywrts	numeric	向 wal buffer 中写的次数。
phyblkwrt	numeric	向 wal buffer 中写的 block 的块数。
writetim	numeric	向 xlog 文件中写操作的时间（单位：微秒）。
avgiotim	bigint	平均写 xlog 的时间（writetim/phywrts）（单位：微秒）。
lstiotim	bigint	最后一次写 xlog 的时间（单位：微秒）。
miniotim	bigint	最小的写 xlog 时间（单位：微秒）。
maxiowtm	bigint	最大的写 xlog 时间（单位：微秒）。

3.2.4.6.GLOBAL_FILE_REDO_IOSTAT

得到 PanWeiDB 内各节点的 Redo（WAL）相关统计信息。

表 1 GLOBALXC_FILE_REDO_IOSTAT 字段

名称	类型	描述
node_name	name	数据库进程名称。

名称	类型	描述
phywrts	bigint	向 wal buffer 中写的次数。
phyblkwrt	bigint	向 wal buffer 中写的 block 的块数。
writetim	bigint	向 xlog 文件中写操作的时间（单位：微秒）。
avgiotim	bigint	平均写 xlog 的时间（writetim/phywrts）（单位：微秒）。
lstiotim	bigint	最后一次写 xlog 的时间（单位：微秒）。
miniotim	bigint	最小的写 xlog 时间（单位：微秒）。
maxiowtm	bigint	最大的写 xlog 时间（单位：微秒）。

3.2.4.7.LOCAL_REL_IOSTAT

获取当前节点中数据文件 IO 状态的累计值，显示为所有数据文件 IO 状态的总和。

表 1 LOCAL_REL_IOSTAT 字段

名称	类型	描述
phyrds	bigint	读物理文件的数目。
phywrts	bigint	写物理文件的数目。
phyblkrd	bigint	读物理文件的块的数目。
phyblkwrt	bigint	写物理文件的块的数目。

3.2.4.8.GLOBAL_REL_IOSTAT

获取所有节点上的数据文件 IO 统计信息。

表 1 GLOBAL_REL_IOSTAT 字段

名称	类型	描述
node_name	name	数据库进程名称。
phyrds	bigint	读物理文件的数目。
phywrts	bigint	写物理文件的数目。
phyblkrd	bigint	读物理文件块的数目。
phyblkwrt	bigint	写物理文件块的数目。

3.2.4.9.SUMMARY_REL_IOSTAT

获取所有节点上的数据文件 IO 统计信息。

表 1 SUMMARY_REL_IOSTAT 字段

名称	类型	描述
phyrds	numeric	读物理文件的数目。
phywrts	numeric	写物理文件的数目。
phyblkrd	numeric	读物理文件的块的数目。
phyblkwrt	numeric	写物理文件的块的数目。

3.2.5.Transaction

3.2.5.1.TRANSACTIONS_PREPARED_XACTS

显示当前准备好进行两阶段提交的事务的信息。

表 1 TRANSACTIONS_PREPARED_XACTS 字段

名称	类型	描述
transaction	xid	预备事务的数字事务标识。
gid	text	赋予该事务的全局事务标识。
prepared	timestamp with time zone	事务准备好提交的时间。
owner	name	执行该事务的用户的名称。
database	name	执行该事务所在的数据库名。

3.2.5.2.SUMMARY_TRANSACTIONS_PREPARED_XACTS

显示 PanWeiDB 中数据库主节点当前准备好进行两阶段提交的事务的信息。

表 1 SUMMARY_TRANSACTIONS_PREPARED_XACTS 字段

名称	类型	描述
transaction	xid	预备事务的数字事务标识。
gid	text	赋予该事务的全局事务标识。
prepared	timestamp with time zone	事务准备好提交的时间。
owner	name	执行该事务的用户的名称。
database	name	执行该事务所在的数据库名。

3.2.5.3.GLOBAL_TRANSACTIONS_PREPARED_XACTS

显示各节点当前准备好进行两阶段提交的事务的信息。

表 1 GLOBAL_TRANSACTIONS_PREPARED_XACTS 字段

名称	类型	描述
transaction	xid	预备事务的数字事务标识。
gid	text	赋予该事务的全局事务标识。
prepared	timestamp with time zone	事务准备好提交的时间。
owner	name	执行该事务的用户的名称。
database	name	执行该事务所在的数据库名。

3.2.5.4.TRANSACTIONS_RUNNING_XACTS

显示当前节点运行事务的信息。

表 1 TRANSACTIONS_RUNNING_XACTS 字段

名称	类型	描述
handle	integer	事务对应的事务管理器中的槽位句柄，该值恒为-1。
gxid	xid	事务 id 号。
state	tinyint	事务状态（3：prepared 或者 0：starting）。
node	text	节点名称。
xmin	xid	节点上当前数据涉及的最小事务号 xmin。
vacuum	boolean	标志当前事务是否是 lazy vacuum 事务。
timeline	bigint	标志数据库重启次数。
prepare_xid	xid	处于 prepared 状态的事务的 id 号，若不在 prepared 状态，值为 0。
pid	bigint	事务对应的线程 id。
next_xid	xid	其余节点发送给当前节点的事务 id，该值恒为 0。

3.2.5.5.SUMMARY_TRANSACTIONS_RUNNING_XACTS

显示集群中各个节点运行事务的信息，字段内容和 transactions_running_xacts 一致。

表 1 SUMMARY_TRANSACTIONS_RUNNING_XACTS 字段

名称	类型	描述
handle	integer	事务对应的事务管理器中的槽位句柄，该值恒为-1。
gxid	xid	事务 id 号。

名称	类型	描述
state	tinyint	事务状态（3：prepared 或者 0：starting）。
node	text	节点名称。
xmin	xid	节点上当前数据涉及的最小事务号 xmin。
vacuum	boolean	标志当前事务是否是 lazy vacuum 事务。
timeline	bigint	标志数据库重启次数。
prepare_xid	xid	处于 prepared 状态的事务的 id 号，若不在 prepared 状态，值为 0。
pid	bigint	事务对应的线程 id。
next_xid	xid	其余节点发送给当前节点的事务 id，该值恒为 0。

3.2.5.6.GLOBAL_TRANSACTIONS_RUNNING_XACTS

显示集群中各个节点运行事务的信息。

表 1 GLOBAL_TRANSACTIONS_RUNNING_XACTS 字段

名称	类型	描述
handle	integer	事务对应的事务管理器中的槽位句柄，该值恒为-1
gxid	xid	事务 id 号。
state	tinyint	事务状态（3：prepared 或者 0：starting）。
node	text	节点名称。
xmin	xid	节点上当前数据涉及的最小事务号 xmin。
vacuum	boolean	标志当前事务是否是 lazy vacuum 事务。
timeline	bigint	标志数据库重启次数。
prepare_xid	xid	处于 prepared 状态的事务的 id 号，若不在 prepared 状态，值为 0。
pid	bigint	事务对应的线程 id。
next_xid	xid	其余节点发送给当前节点的事务 id，该值恒为 0。

3.2.6.Session/Thread

3.2.6.1.GLOBAL_SESSION_STAT_ACTIVITY

显示 PanWeiDB 内各节点上正在运行的线程相关的信息。

表 1 GLOBAL_SESSION_STAT_ACTIVITY 字段

名称	类型	描述
coorname	text	数据库进程名称。
datid	oid	用户会话在后台连接到的数据库 OID。
datname	text	用户会话在后台连接到的数据库名称。
pid	bigint	后台线程 ID。
usesysid	oid	登录该后台的用户 OID。
usename	text	登录该后台的用户名。
application_name	text	连接到该后台的应用名。
client_addr	inet	连接到该后台的客户端的 IP 地址。如果此字段是 null，它表明通过服务器机器上 UNIX 套接字连接客户端或者这是内部进程，如 autovacuum。
client_hostname	text	客户端的主机名，这个字段是通过 client_addr 的反向 DNS 查找得到。这个字段只有在启动 log_hostname 且使用 IP 连接时才非空。
client_port	integer	客户端用于与后台通讯的 TCP 端口号，如果使用 Unix 套接字，则为-1。
backend_start	timestamp with time zone	该过程开始的时间，即当客户端连接服务器时间。
xact_start	timestamp with time zone	启动当前事务的时间，如果没有事务是活跃的，则为 null。如果当前查询是首个事务，则这列等同于 query_start 列。
query_start	timestamp with time zone	开始当前活跃查询的时间，如果 state 的值不是 active，则这个值是上一个查询的开始时间。

名称	类型	描述
state_change	timestamp with time zone	上次状态改变的时间。
waiting	boolean	如果后台当前正等待锁则为 true。
enqueue	text	该字段不支持。
state	text	该后台当前总体状态。可能值是： ❖ active：后台正在执行一个查询。 ❖ idle：后台正在等待一个新的客户端命令。 ❖ idle in transaction：后台在事务中，但是目前无法执行查询。 ❖ idle in transaction (aborted)：这个状态除说明事务中有某个语句导致了错误外，类似于 idle in transaction ❖ fastpath function call：后台正在执行一个 fast-path 函数。 ❖ disabled：如果后台禁用 track_activities，则报告这个状态。 说明：普通用户只能查看到自己帐户所对应的会话状态。即其他帐户的 state 信息为空。例如以 judy 用户连接数据库后，在 pg_stat_activity 中查看到的普通用户 joe 及初始用户 panweidb 的 state 信息为空。 panweidb=# SELECT datname, usename, usesysid,state,pid FROM pg_stat_activity; datname | usename | usesysid | state | pid ----+---+----+---+------postgres | panweidb | 10 | |139968752121616 postgres | panweidb | 10 | |13996

名称	类型	描述
		8903116560 db_tpcds | judy | 16398 | active |139968391403280 postgres | panweidb | 10 | |139968643069712 postgres | panweidb | 10 | |139968680818448 postgres | joe | 16390 | |139968563377936 (6 rows)
resource_pool	name	用户使用的资源池。
query_id	bigint	查询语句的 ID。
query	text	该后台的最新查询。如果 state 状态是 active（活跃的），此字段显示当前正在执行的查询。所有其他情况表示上一个查询。
unique_sql_id	bigint	语句的 unique sql id。
trace_id	text	驱动传入的 trace id，与应用的一次请求相关联。

3.2.6.2.SESSION_STAT_ACTIVITY

显示当前节点上正在运行的线程相关的信息。

表 1 SESSION_STAT_ACTIVITY 字段

名称	类型	描述
datid	oid	用户会话在后台连接到的数据库 OID。
datname	name	用户会话在后台连接到的数据库名称。
pid	bigint	后台线程 ID。

名称	类型	描述
usesysid	oid	登录该后台的用户 OID。
usename	name	登录该后台的用户名。
application_name	text	连接到该后台的应用名。
client_addr	inet	连接到该后台的客户端的 IP 地址。如果此字段是 null，它表明通过服务器机器上 UNIX 套接字连接客户端或者这是内部进程，如 autovacuum。
client_hostname	text	客户端的主机名，这个字段是通过 client_addr 的反向 DNS 查找得到。这个字段只有在启动 log_hostname 且使用 IP 连接时才非空。
client_port	integer	客户端用于与后台通讯的 TCP 端口号，如果使用 Unix 套接字，则为-1。
backend_start	timestamp with time zone	该过程开始的时间，即当客户端连接服务器时间。
xact_start	timestamp with time zone	启动当前事务的时间，如果没有事务是活跃的，则为 null。如果当前查询是首个事务，则这列等同于 query_start 列。
query_start	timestamp with time zone	开始当前活跃查询的时间，如果 state 的值不是 active，则这个值是上一个查询的开始时间。
state_change	timestamp with time zone	上次状态改变的时间。
waiting	boolean	如果后台当前正等待锁则为 true。
enqueue	text	该字段不支持。
state	text	该后台当前总体状态。可能值是：

名称	类型	描述
		❖ active：后台正在执行一个查询。 ❖ idle：后台正在等待一个新的客户端命令。 ❖ idle in transaction：后台在事务中，但是目前无法执行查询。 ❖ idle in transaction (aborted)：这个状态除说明事务中有某个语句导致了错误外，类似于 idle in transaction ❖ fastpath function call：后台正在执行一个 fast-path 函数。 ❖ disabled：如果后台禁用 track_activities，则报告这个状态。 说明：disabled：如果后台禁用 track_activities，则报告这个状态。 panweidb=# SELECT datname, usename, usesysid,state,pid FROM pg_stat_activity; datname | usename | usesysid | state | pid ----+----+----+----+------postgres | panweidb | 10 | |139968752121616 postgres | panweidb | 10 | |139968903116560 db_tpcds | judy | 16398 | active |139968391403280 postgres | panweidb | 10 | |139968643069712 postgres | panweidb | 10 | |139968680818448 postgres | joe | 16390 | |139968563377936 (6 rows)

名称	类型	描述
resource_pool	name	用户使用的资源池。
query_id	bigint	查询语句的 ID。
query	text	该后台的最新查询。如果 state 状态是 active（活跃的），此字段显示当前正在执行的查询。所有其他情况表示上一个查询。
unique_sql_id	bigint	语句的 unique sql id。
trace_id	text	驱动传入的 trace id，与应用的一次请求相关联。

3.2.6.3.GLOBAL_THREADPOOL_STATUS

GLOBAL_THREADPOOL_STATUS 视图显示在所有节点上的线程池中工作线程及会话的状态信息。具体的字段详见表 1。（章节 LOCAL_THREADPOOL_STATUS）。

3.2.6.4.LOCAL_THREADPOOL_STATUS

LOCAL_THREADPOOL_STATUS 视图显示线程池下工作线程及会话的状态信息。该视图仅在线程池开启（enable_thread_pool = on）时生效。

表 1 LOCAL_THREADPOOL_STATUS 字段

名称	类型	描述
node_name	text	数据库进程名称。
group_id	integer	线程池组 ID。
bind_numa_id	integer	该线程池组绑定的 NUMA ID。
bind_cpu_number	integer	该线程池组绑定的 CPU 信息。如果未绑定 CPU，该值为 NULL。

名称	类型	描述
listener	integer	该线程池组的 Listener 线程数量。
worker_info	text	线程池中线程相关信息，包括以下信息： ❖ default：该线程池组中的初始线程数量。 ❖ new：该线程池组中新增线程的数量。 ❖ expect：该线程池组中预期线程的数量。 ❖ actual：该线程池组中实际线程的数量。 ❖ idle：该线程池组中空闲线程的数量。 ❖ pending：该线程池组中等待线程的数量。
session_info	text	线程池中会话相关信息，包括以下信息： ❖ total：该线程池组中所有的会话数量。 ❖ waiting：该线程池组中等待调度的会话数量。 ❖ running：该线程池中正在执行的会话数量。 ❖ idle：该线程池组中空闲的会话数量。
stream_info	text	stream 池相关信息，包含以下信息： ❖ total：该 stream 池组中所有的线程数量。 ❖ running：该 stream 池中正在执行的线程数量。 ❖ idle：该 stream 池组中空闲的线程数量。

3.2.6.5.SESSION_STAT

当前节点以会话线程或 AutoVacuum 线程为单位，统计会话状态信息。

表 1 SESSION_STAT 字段

名称	类型	描述
sessid	text	线程启动时间+线程标识。
statid	integer	统计编号。
statname	text	统计会话名称。
statunit	text	统计会话单位。

名称	类型	描述
value	bigint	统计会话值。

3.2.6.6.GLOBAL_SESSION_STAT

各节点上以会话线程或 AutoVacuum 线程为单位，统计会话状态信息。

表 1 GLOBAL_SESSION_STAT 字段

名称	类型	描述
node_name	name	数据库进程名称。
sessid	text	线程启动时间+线程标识。
statid	integer	统计编号。
statname	text	统计会话名称。
statunit	text	统计会话单位。
value	bigint	统计会话值。

3.2.6.7.SESSION_TIME

用于统计当前节点会话线程的运行时间信息，及各执行阶段所消耗时间。

表 1 SESSION_TIME 字段

名称	类型	描述
sessid	text	线程启动时间+线程标识。
stat_id	integer	统计编号。
stat_name	text	会话类型名称。
value	bigint	会话值。

3.2.6.8.GLOBAL_SESSION_TIME

用于统计各节点会话线程的运行时间信息，及各执行阶段所消耗时间。

表 1 GLOBAL_SESSION_TIME 字段

名称	类型	描述
node_name	name	数据库进程名称。
sessid	text	线程启动时间+线程标识。
stat_id	integer	统计编号。
stat_name	text	会话类型名称。

名称	类型	描述
value	bigint	会话值。

3.2.6.9.SESSION_MEMORY

统计 Session 级别的内存使用情况，包含执行作业在数据节点上 PanWeiDB 线程和 Stream 线程分配的所有内存，单位为 MB。

表 1 SESSION_MEMORY 字段

名称	类型	描述
sessid	text	线程启动时间+线程标识。
init_mem	integer	当前正在执行作业进入执行器前已分配的内存。
used_mem	integer	当前正在执行作业已分配的内存。
peak_mem	integer	当前正在执行作业已分配的内存峰值。

3.2.6.10.GLOBAL_SESSION_MEMORY

统计各节点的 Session 级别的内存使用情况，包含执行作业在数据节点上 PanWeiDB 线程和 Stream 线程分配的所有内存，单位为 MB。

当 GUC 参数 enable_memory_limit=off 时，本视图不可使用。

表 1 GLOBAL_SESSION_MEMORY 字段

名称	类型	描述
node_name	name	数据库进程名称。
sessid	text	线程启动时间+线程标识。
init_mem	integer	当前正在执行作业进入执行器前已分配的内存。
used_mem	integer	当前正在执行作业已分配的内存。
peak_mem	integer	当前正在执行作业已分配的内存峰值。

3.2.6.11.SESSION_MEMORY_DETAIL

统计线程的内存使用情况，以 MemoryContext 节点来统计。

表 1 SESSION_MEMORY_DETAIL 字段

名称	类型	描述
sessid	text	线程启动时间+线程标识。

名称	类型	描述
sesstype	text	线程名称。
contextname	text	内存上下文名称。
level	smallint	内存上下文的重要级别。
parent	text	父级内存上下文名称。
totalsize	bigint	总申请内存大小（单位：字节）。
freesize	bigint	空闲内存大小（单位：字节）。
usedsize	bigint	使用内存大小（单位：字节）。

3.2.6.12.GLOBAL_SESSION_MEMORY_DETAIL

统计各节点的线程的内存使用情况，以 MemoryContext 节点来统计。

表 1 GLOBAL_SESSION_MEMORY_DETAIL 字段

名称	类型	描述
node_name	name	数据库进程名称。
sessid	text	线程启动时间+线程标识。
sesstype	text	线程名称。
contextname	text	内存上下文名称。
level	smallint	内存上下文的重要级别。
parent	text	父级内存上下文名称。
totalsize	bigint	总申请内存大小（单位：字节）。
freesize	bigint	空闲内存大小（单位：字节）。
usedsize	bigint	使用内存大小（单位：字节）。

3.2.6.13.THREAD_WAIT_STATUS

通过该视图可以检测当前实例中工作线程（backend thread）以及辅助线程（auxiliary thread）的阻塞等待情况，具体事件信息请参考：系统表和系统视图->系统视图->PG_THREAD_WAIT_STATUS 章节中表 2 等待状态列表、表 3 轻量级锁等待事件列表、表 4 IO 等待事件列表和表 5 事务锁等待事件列表。

表 1 THREAD_WAIT_STATUS 字段

名称	类型	描述
node_name	text	数据库进程名称。
db_name	text	数据库名称。

名称	类型	描述
thread_name	text	线程名称。
query_id	bigint	查询 ID，对应 debug_query_id。
tid	bigint	当前线程的线程号。
sessionid	bigint	session 的 ID。
lwtid	integer	当前线程的轻量级线程号。
psessionid	bigint	streaming 线程的父线程。
tlevel	integer	streaming 线程的层级。
smpid	integer	并行线程的 ID。
wait_status	text	当前线程的等待状态。
wait_event	text	如果 wait_status 是 acquire lock、acquire lwlock、wait io 三种类型，此列描述具体的锁、轻量级锁、IO 的信息。否则为空。

3.2.6.14.GLOBAL_THREAD_WAIT_STATUS

通过该视图可以检测所有节点上工作线程（backend thread）以及辅助线程（auxiliary thread）的阻塞等待情况。具体事件信息请参考：系统表和系统视图->系统视图->PG_THREAD_WAIT_STATUS 章节中表 2 等待状态列表、表 3 轻量级锁等待事件列表、表 4 IO 等待事件列表和表 5 事务锁等待事件列表。

通过 GLOBAL_THREAD_WAIT_STATUS 视图，可以查看 PanWeiDB 全局各个节点上所有 SQL 语句产生的线程之间的调用层次关系，以及各个线程的阻塞等待状态，从而更容易定位 hang 以及类似现象的原因。

GLOBAL_THREAD_WAIT_STATUS 视图和 THREAD_WAIT_STATUS 视图列定义完全相同，这是由于 GLOBAL_THREAD_WAIT_STATUS 视图本质是到 PanWeiDB 中各个节点上查询 THREAD_WAIT_STATUS 视图汇总的结果。

表 1 GLOBAL_THREAD_WAIT_STATUS 字段

名称	类型	描述
node_name	text	数据库进程名称。
db_name	text	数据库名称。
thread_name	text	线程名称。
query_id	bigint	查询 ID，对应 debug_query_id。
tid	bigint	当前线程的线程号。

名称	类型	描述
sessionid	bigint	session 的 ID。
lwtid	integer	当前线程的轻量级线程号。
psessionid	bigint	streaming 线程的父线程。
tlevel	integer	streaming 线程的层级。
smpid	integer	并行线程的 ID。
wait_status	text	当前线程的等待状态。
wait_event	text	如果 wait_status 是 acquire lock、acquire lwlock、wait io 三种类型，此列描述具体的锁、轻量级锁、IO 的信息。否则是空。

3.2.6.15.SESSION_CPU_RUNTIME

SESSION_CPU_RUNTIME 视图显示和当前用户执行复杂作业（正在运行）时的负载管理 CPU 使用的信息。

表 1 SESSION_CPU_RUNTIME 字段

名称	类型	描述
datid	oid	连接后端的数据库 OID。
usename	name	登录到该后端的用户名。
pid	bigint	后端线程 ID。
start_time	timestamp with time zone	语句执行的开始时间。
min_cpu_time	bigint	语句在数据库节点上的最小 CPU 时间，单位为 ms。
max_cpu_time	bigint	语句在数据库节点上的最大 CPU 时间，单位为 ms。
total_cpu_time	bigint	语句在数据库节点上的 CPU 总时间，单位为 ms。
query	text	正在执行的语句。
node_group	text	语句所属用户对应的逻辑 PanWeiDB。
top_cpu_dn	text	cpu 使用量 topN 信息。

3.2.6.16.SESSION_MEMORY_RUNTIME

SESSION_MEMORY_RUNTIME 视图显示和当前用户执行复杂作业正在运行时的负载管理内存使用的信息。

表 1 SESSION_MEMORY_RUNTIME 字段

名称	类型	描述
datid	oid	连接后端的数据库 OID。
usename	name	登录到该后端的用户名。
pid	bigint	后端线程 ID。
start_time	timestamp with time zone	语句执行的开始时间。
min_cpu_time	bigint	语句在数据库节点上的最小 CPU 时间，单位为 ms。
max_cpu_time	bigint	语句在数据库节点上的最大 CPU 时间，单位为 ms。
total_cpu_time	bigint	语句在数据库节点上的 CPU 总时间，单位为 ms。
query	text	正在执行的语句。
node_group	text	语句所属用户对应的逻辑 PanWeiDB。
top_cpu_dn	text	cpu 使用量 topN 信息。

3.2.6.17.STATEMENT_IOSTAT_COMPLEX_RUNTIME

STATEMENT_IOSTAT_COMPLEX_RUNTIME 视图显示当前用户执行作业正在运行时的 IO 负载管理相关信息。以下涉及到 iops，对于行存，均以万次/s 为单位，对于列存，均以次/s 为单位。

表 1 STATEMENT_IOSTAT_COMPLEX_RUNTIME 字段

名称	类型	描述
query_id	bigint	作业 id。
mincurriops	integer	该作业当前 io 在各数据库节点中的最小值。
maxcurriops	integer	该作业当前 io 在各数据库节点中的最大值。
minpeakiops	integer	在作业运行时，作业 io 峰值中，各数据库节点的最小值。
maxpeakiops	integer	在作业运行时，作业 io 峰值中，各数据库节点的最大值。
io_limits	integer	该作业所设 GUC 参数 io_limits。

名称	类型	描述
io_priority	text	该作业所设 GUC 参数 io_priority。
query	text	作业。
node_group	text	作业所属用户对应的逻辑 PanWeiDB。
curr_io_limits	integer	使用 io_priority 管控 io 时的实时 io_limits 值。

3.2.6.18.LOCAL_ACTIVE_SESSION

LOCAL_ACTIVE_SESSION 视图显示本节点上的 ACTIVE SESSION PROFILE 内存中的样本。

表 1 LOCAL_ACTIVE_SESSION 字段

名称	类型	描述
sampleid	bigint	采样 ID。
sample_time	timestamp with time zone	采样的时间。
need_flush_sample	boolean	该样本是否需要刷新的磁盘。
databaseid	oid	数据库 ID。
thread_id	bigint	线程的 ID。
sessionid	bigint	会话的 ID。
start_time	timestamp with time zone	会话的启动时间。
event	text	具体的事件名称。
lwtid	integer	当前线程的轻量级线程号。
psessionid	bigint	streaming 线程的父线程。
tlevel	integer	streaming 线程的层级。与执行计划的层级（ID）相对应。
smpid	integer	smp 执行模式下并行线程的并行编号。
userid	oid	session 用户的 ID。
application_name	text	应用的名称。
client_addr	inet	client 端的地址。
client_hostname	text	client 端的名称。
client_port	integer	客户端用于与后端通讯的 TCP 端口号。
query_id	bigint	debug query id。
unique_query_id	bigint	unique query id。

名称	类型	描述
user_id	oid	unique query 的 key 中的 user_id。
cn_id	integer	cn id，在 DN 上表示下发该 unique sql 的节点 id，unique query 的 key 中的 cn_id。
unique_query	text	规范化后的 UniqueSQL 文本串。
locktag	text	会话等待锁信息，可通过 locktag_decode 解析。
lockmode	text	会话等待锁模式。
block_sessionid	bigint	如果会话正在等待锁，阻塞该会话获取锁的会话标识。
final_block_sessionid	bigint	表示源头阻塞会话 ID。
wait_status	text	描述 event 列的更多详细信息。
global_sessionid	text	全局会话 ID。

3.2.7.Query

3.2.7.1.STATEMENT_COMPLEX_HISTORY

STATEMENT_COMPLEX_HISTORY 视图显示在数据库主节点上执行作业结束后的负载管理记录。

名称	类型	描述
datid	oid	连接后端的数据库 OID。
dbname	text	连接后端的数据库名称。
schemaname	text	模式的名称。
nodename	text	数据库进程名称
username	text	连接到后端的用户名。
application_name	text	连接到后端的应用名。
client_addr	inet	连接到后端的客户端的 IP 地址。 如果此字段是 null，它表明通过服务器机器上 UNIX 套接字连接客户端或者这是内部进程，如 autovacuum。
client_hostname	text	客户端的主机名，这个字段是通过 client_addr 的反向 DNS 查找得到。这个字段只有在启动 log_hostname 且使用

名称	类型	描述
		IP 连接时才非空。
client_port	integer	客户端用于与后端通讯的 TCP 端口号，如果使用 Unix 套接字，则为-1。
query_band	text	用于标示作业类型，可通过 GUC 参数 query_band 进行设置，默认为空字符串。
block_time	bigint	语句执行前的阻塞时间，包含语句解析和优化时间，单位 ms。
start_time	timestamp with time zone	语句执行的开始时间。
finish_time	timestamp with time zone	语句执行的结束时间。
duration	bigint	语句实际执行的时间，单位 ms。
estimate_total_time	bigint	语句预估执行时间，单位 ms。
status	text	语句执行结束状态：正常为 finished，异常为 aborted。
abort_info	text	语句执行结束状态为 aborted 时显示异常信息。
resource_pool	text	用户使用的资源池。
control_group	text	语句所使用的 Cgroup。
estimate_memory	integer	语句预估使用内存。
min_peak_memory	integer	语句在数据库节点上的最小内存峰值，单位 MB。
max_peak_memory	integer	语句在数据库节点上的最大内存峰值，单位 MB。
average_peak_memory	integer	语句执行过程中的内存使用平均值，单位 MB。
memory_skew_percent	integer	语句数据库节点间的内存使用倾斜率。
spill_info	text	语句在数据库节点上的下盘信息：None：数据库节点均未下盘。All：数据库节点均下盘。[a:b]：数量为 b 个数据库节点中有

名称	类型	描述
		a 个数据库节点下盘。
min_spill_size	integer	若发生下盘，数据库节点上下盘的最小数据量，单位 MB，默认为 0。
max_spill_size	integer	若发生下盘，数据库节点上下盘的最大数据量，单位 MB，默认为 0。
average_spill_size	integer	若发生下盘，数据库节点上下盘的平均数据量，单位 MB，默认为 0。
spill_skew_percent	integer	若发生下盘，数据库节点间下盘倾斜率。
min_dn_time	bigint	语句在数据库节点上的最小执行时间，单位 ms。
max_dn_time	bigint	语句在数据库节点上的最大执行时间，单位 ms。
average_dn_time	bigint	语句在数据库节点上的平均执行时间，单位 ms。
dntime_skew_percent	integer	语句在数据库节点的执行时间倾斜率。
min_cpu_time	bigint	语句在数据库节点上的最小 CPU 时间，单位 ms。
max_cpu_time	bigint	语句在数据库节点上的最大 CPU 时间，单位 ms。
total_cpu_time	bigint	语句在数据库节点上的 CPU 总时间，单位 ms。
cpu_skew_percent	integer	语句在数据库节点间的 CPU 时间倾斜率。
min_peak_iops	integer	语句在数据库节点上的每秒最小 IO 峰值（列存单位是次/s，行存单位是万次/s）。
max_peak_iops	integer	语句在数据库节点上的每秒最大 IO 峰值（列存单位是次/s，行存单位是万次/s）。
average_peak_iops	integer	语句在数据库节点上的每秒平均 IO 峰值（列存单位是次/s，行存单位是万次/s）。
iops_skew_percent	integer	语句在数据库节点间的 IO 倾斜率。
warning	text	主要显示如下几类告警信息：Spill file size large than 256MBBroadcast size large than 100MBEarly spillSpill times is greater than 3Spill on memory

名称	类型	描述
		adaptiveHash table conflict
queryid	bigint	语句执行使用的内部 query id。
query	text	执行的语句。
query_plan	text	语句的执行计划。
node_group	text	语句所属用户对应的逻辑 panweidb。
cpu_top1_node_name	text	cpu 使用率第 1 的节点名称。
cpu_top2_node_name	text	cpu 使用率第 2 的节点名称。
cpu_top3_node_name	text	cpu 使用率第 3 的节点名称。
cpu_top4_node_name	text	cpu 使用率第 4 的节点名称。
cpu_top5_node_name	text	cpu 使用率第 5 的节点名称。
mem_top1_node_name	text	内存使用量第 1 的节点名称。
mem_top2_node_name	text	内存使用量第 2 的节点名称。
mem_top3_node_name	text	内存使用量第 3 的节点名称。
mem_top4_node_name	text	内存使用量第 4 的节点名称。
mem_top5_node_name	text	内存使用量第 5 的节点名称。
cpu_top1_value	bigint	cpu 使用率。
cpu_top2_value	bigint	cpu 使用率。
cpu_top3_value	bigint	cpu 使用率。
cpu_top4_value	bigint	cpu 使用率。
cpu_top5_value	bigint	cpu 使用率。
mem_top1_value	bigint	内存使用量。
mem_top2_value	bigint	内存使用量。
mem_top3_value	bigint	内存使用量。
mem_top4_value	bigint	内存使用量。
mem_top5_value	bigint	内存使用量。
top_mem_dn	text	内存使用量 topN 信息。
top_cpu_dn	text	cpu 使用量 topN 信息。

3.2.7.2.SUMMARY_STATEMENT

获得各数据库主节点的执行语句（归一化 SQL）的全量信息（包含数据库节点）。

表 1 SUMMARY_STATEMENT 字段

名称	类型	描述
node_name	name	数据库进程名称。
node_id	integer	节点的 ID。
user_name	name	用户名称。
user_id	oid	用户 OID。
unique_sql_id	bigint	归一化的 SQL ID。
query	text	归一化的 SQL。备注：长度受 track_activity_query_size 控制。
n_calls	bigint	调用次数。
min_elapse_time	bigint	SQL 在内核内的最小运行时间（单位：微秒）。
max_elapse_time	bigint	SQL 在内核内的最大运行时间（单位：微秒）。
total_elapse_time	bigint	SQL 在内核内的总运行时间（单位：微秒）。
n_returned_rows	bigint	SELECT 返回的结果集行数。
n_tuples_fetched	bigint	随机扫描行。
n_tuples_returned	bigint	顺序扫描行。
n_tuples_inserted	bigint	插入行。
n_tuples_updated	bigint	更新行。
n_tuples_deleted	bigint	删除行。
n_blocks_fetched	bigint	buffer 的块访问次数。
n_blocks_hit	bigint	buffer 的块命中次数。
n_soft_parse	bigint	软解析次数。
n_hard_parse	bigint	硬解析次数。
db_time	bigint	有效的 DB 时间花费，多线程将累加（单位：微秒）。
cpu_time	bigint	CPU 时间（单位：微秒）。
execution_time	bigint	执行器内执行时间（单位：微秒）。
parse_time	bigint	SQL 解析时间（单位：微秒）。
plan_time	bigint	SQL 生成计划时间（单位：微

名称	类型	描述
		秒）。
rewrite_time	bigint	SQL 重写时间（单位：微秒）。
pl_execution_time	bigint	plpgsql 上的执行时间（单位：微秒）。
pl_compilation_time	bigint	plpgsql 上的编译时间（单位：微秒）。
data_io_time	bigint	IO 上的时间花费（单位：微秒）。
net_send_info	text	通过物理连接发送消息的网络状态，包含时间（微秒）、调用次数、吞吐量（字节）。通过该字段可以分析 SQL 在分布式系统下的网络开销，单机模式下不支持该字段。例如：{“time” :xxx, “n_calls” :xxx, “size” :xxx}。
net_recv_info	text	通过物理连接接收消息的网络状态，包含时间（微秒）、调用次数、吞吐量（字节）。通过该字段可以分析 SQL 在分布式系统下的网络开销，单机模式下不支持该字段。例如：{“time” :xxx, “n_calls” :xxx, “size” :xxx}。
net_stream_send_info	text	通过逻辑连接发送消息的网络状态，包含时间（微秒）、调用次数、吞吐量（字节）。通过该字段可以分析 SQL 在分布式系统下的网络开销，单机模式下不支持该字段。例如：{“time” :xxx,

名称	类型	描述
		“n_calls” :xxx, “size” :xxx}。
net_stream_recv_info	text	通过逻辑连接接收消息的网络状态，包含时间（微秒）、调用次数、吞吐量（字节）。通过该字段可以分析 SQL 在分布式系统下的网络开销，单机模式下不支持该字段。例如： {“time” :xxx, “n_calls” :xxx, “size” :xxx}。
last_updated	timestamp with time zone	最后一次更新该语句的时间。
sort_count	bigint	排序执行的次数。
sort_time	bigint	排序执行的时间（单位：微秒）。
sort_mem_used	bigint	排序过程中使用的 work memory 大小（单位：KB）。
sort_spill_count	bigint	排序过程中，若发生落盘，写文件的次数。
sort_spill_size	bigint	排序过程中，若发生落盘，使用的文件大小（单位：KB）。
hash_count	bigint	hash 执行的次数。
hash_time	bigint	hash 执行的时间（单位：微秒）。
hash_mem_used	bigint	hash 过程中使用的 work memory 大小（单位：KB）。
hash_spill_count	bigint	hash 过程中，若发生落盘，写文件的次数。
hash_spill_size	bigint	hash 过程中，若发生落盘，使用的文件大小（单位：KB）。

3.2.7.3.STATEMENT

获得当前节点的执行语句（归一化 SQL）的信息。查询视图必须具有 sysadmin 权限或者 monitor admin 权限。数据库主节点上可以看到此数据库主节点接收到的归一化的 SQL 的全量统计信息（包含数据库节点）；数据库节点上仅可看到归一化的 SQL 的此节点执行的统计信息。

表 1 STATEMENT 字段

名称	类型	描述
node_name	name	数据库进程名称。
node_id	integer	节点的 ID。
user_name	name	用户名称。
user_id	oid	用户 OID。
unique_sql_id	bigint	归一化的 SQL ID。
query	text	归一化的 SQL。备注：长度受 track_activity_query_size 控制。
n_calls	bigint	调用次数。
min_elapse_time	bigint	SQL 在内核内的最小运行时间（单位：微秒）。
max_elapse_time	bigint	SQL 在内核内的最大运行时间（单位：微秒）。
total_elapse_time	bigint	SQL 在内核内的总运行时间（单位：微秒）。
n_returned_rows	bigint	SELECT 返回的结果集行数。
n_tuples_fetched	bigint	随机扫描行。
n_tuples_returned	bigint	顺序扫描行。
n_tuples_inserted	bigint	插入行。
n_tuples_updated	bigint	更新行。
n_tuples_deleted	bigint	删除行。
n_blocks_fetched	bigint	buffer 的块访问次数。
n_blocks_hit	bigint	buffer 的块命中次数。
n_soft_parse	bigint	软解析次数，n_soft_parse + n_hard_parse 可能大于 n_calls，因为子查询未计入 n_calls。
n_hard_parse	bigint	硬解析次数，n_soft_parse + n_hard_parse 可能大于 n_calls，因为子查询未计入

名称	类型	描述
		n_calls。
db_time	bigint	有效的 DB 时间花费，多线程将累加（单位：微秒）。
cpu_time	bigint	CPU 时间（单位：微秒）。
execution_time	bigint	执行器内执行时间（单位：微秒）。
parse_time	bigint	SQL 解析时间（单位：微秒）。
plan_time	bigint	SQL 生成计划时间（单位：微秒）。
rewrite_time	bigint	SQL 重写时间（单位：微秒）。
pl_execution_time	bigint	plpgsql 上的执行时间（单位：微秒）。
pl_compilation_time	bigint	plpgsql 上的编译时间（单位：微秒）。
data_io_time	bigint	IO 上的时间花费（单位：微秒）。
net_send_info	text	通过物理连接发送消息的网络状态，包含时间（微秒）、调用次数、吞吐量（字节）。通过该字段可以分析 SQL 在分布式系统下的网络开销，单机模式下不支持该字段。例如：{“time”:xxx,“n_calls”:xxx,“size”:xxx}。
net_recv_info	text	通过物理连接接收消息的网络状态，包含时间（微秒）、调用次数、吞吐量（字节）。通过该字段可以分析 SQL 在分布式系统下的网络开销，单机模式下不支持该字段。例如：{“time”:xxx,“n_calls”:xxx,“size”:xxx}。
net_stream_send_info	text	通过逻辑连接发送消息的网络状态，包含时间（微秒）、调用次数、吞吐量（字节）。通过该字段可以分析 SQL 在分布式系统下的网络开销，单机模式下不支持该字段。例如：{“time”:xxx,“n_calls”:xxx,“size”:xxx}。
net_stream_recv_info	text	通过逻辑连接接收消息的网络状态，包含时间（微秒）、调用次数、吞吐量（字节）。通过该字段可以分析 SQL 在分布式系统下的网络开销，单机模式下不支持该字段。例如：{“time”:xxx,“n_calls”:xxx,“size”:xxx}。
sort_count	bigint	排序执行的次数。
sort_time	bigint	排序执行的时间（单位：微秒）。
sort_mem_used	bigint	排序过程中使用的 work memory 大小（单

名称	类型	描述
		位：KB）。
sort_spill_count	bigint	排序过程中，若发生落盘，写文件的次数。
sort_spill_size	bigint	排序过程中，若发生落盘，使用的文件大小（单位：KB）。
hash_count	bigint	hash 执行的次数。
hash_time	bigint	hash 执行的时间（单位：微秒）。
hash_mem_used	bigint	hash 过程中使用的 work memory 大小（单位：KB）。
hash_spill_count	bigint	hash 过程中，若发生落盘，写文件的次数。
hash_spill_size	bigint	hash 过程中，若发生落盘，使用的文件大小（单位：KB）。
last_updated	timestamp with time zone	最后一次更新该语句的时间。

相关特性

STATEMENT 对应系统函数 get_instr_unique_sql，主要目的是保留数据库启动后，运行的 SQL 的状态记录。

一般使用形式：

```
select * from dbe_perf.statement;
```

主要受到以下参数控制：

- enable_resource_track：允许运行时候的资源使用追踪。
- instr_unique_sql_count：允许记录在内存中的 SQL 总数量。每次修改此参数，都会重置掉内存中已经存在的所有的归一化 SQL。
- instr_unique_sql_track_type：归一化 SQL 追踪的方式，参数取值为 top、all，目前仅支持 top，对于存储过程，仅记录最外层调用而非所有 SQL。
- enable_auto_clean_unique_sql：是否打开归一化 SQL 的自动清理机制，当记录达到上限后，可以自动随机清理其中 10%的记录。如果不打开，则会记录错误日志，SQL 相关内容也不会记录到内存中。

另请参阅：[GUC 参数 - Query]

3.2.7.4.STATEMENT_COUNT

显示数据库当前节点当前时刻执行的五类语句（SELECT、INSERT、UPDATE、DELETE、MERGE INTO）和（DDL、DML、DCL）统计信息。

【说明】
普通用户查询 STATEMENT_COUNT 视图仅能看到该用户当前节点的统计信息；管理员权限用户查询 STATEMENT_COUNT 视图则能看到所有用户当前节点的统计信息。当 PanWeiDB 或该节点重启时，计数将清零，并重新开始计数。计数以节点收到的查询数为准，PanWeiDB 内部进行的查询。例如，数据库主节点收到一条查询，若下发多条查询数据库节点，那将在数据库节点上进行相应次数的计数。

表 1 STATEMENT_COUNT 字段

名称	类型	描述
node_name	text	数据库进程名称。
user_name	text	用户名。
select_count	bigint	select 语句统计结果。
update_count	bigint	update 语句统计结果。
insert_count	bigint	insert 语句统计结果。
delete_count	bigint	delete 语句统计结果。
mergeinto_count	bigint	merge into 语句统计结果。
ddl_count	bigint	DDL 语句的数量。
dml_count	bigint	DML 语句的数量。
dcl_count	bigint	DCL 语句的数量。
total_select_elapse	bigint	总 select 的时间花费（单位：微秒）。
avg_select_elapse	bigint	平均 select 的时间花费（单位：微秒）。
max_select_elapse	bigint	最大 select 的时间花费(单位：微秒)。
min_select_elapse	bigint	最小 select 的时间花费（单位：微秒）。
total_update_elapse	bigint	总 update 的时间花费（单位：微秒）。
avg_update_elapse	bigint	平均 update 的时间花费(单位：微秒)。
max_update_elapse	bigint	最大 update 的时间花费（单位：微秒）。
min_update_elapse	bigint	最小 update 的时间花费（单位：微秒）。
total_insert_elapse	bigint	总 insert 的时间花费（单位：微秒）。

名称	类型	描述
avg_insert_elapse	bigint	平均 insert 的时间花费（单位：微秒）。
max_insert_elapse	bigint	最大 insert 的时间花费（单位：微秒）。
min_insert_elapse	bigint	最小 insert 的时间花费（单位：微秒）。
total_delete_elapse	bigint	总 delete 的时间花费（单位：微秒）。
avg_delete_elapse	bigint	平均 delete 的时间花费（单位：微秒）。
max_delete_elapse	bigint	最大 delete 的时间花费（单位：微秒）。
min_delete_elapse	bigint	最小 delete 的时间花费（单位：微秒）。

3.2.7.5.GLOBAL_STATEMENT_COUNT

显示数据库各节点当前时刻执行的五类语句（SELECT、INSERT、UPDATE、DELETE、MERGE INTO）和（DDL、DML、DCL）统计信息。

表 1 GLOBAL_STATEMENT_COUNT 字段

名称	类型	描述
node_name	text	数据库进程名称。
user_name	text	用户名。
select_count	bigint	select 语句统计结果。
update_count	bigint	update 语句统计结果。
insert_count	bigint	insert 语句统计结果。
delete_count	bigint	delete 语句统计结果。
mergeinto_count	bigint	merge into 语句统计结果。
ddl_count	bigint	DDL 语句的数量。
dml_count	bigint	DML 语句的数量。
dcl_count	bigint	DCL 语句的数量。
total_select_elapse	bigint	总 select 的时间花费（单位：微秒）。
avg_select_elapse	bigint	平均 select 的时间花费（单位：微秒）。
max_select_elapse	bigint	最大 select 的时间花费（单位：微秒）。
min_select_elapse	bigint	最小 select 的时间花费（单位：微秒）。
total_update_elapse	bigint	总 update 的时间花费（单位：微秒）。
avg_update_elapse	bigint	平均 update 的时间花费（单位：微秒）。
max_update_elapse	bigint	最大 update 的时间花费（单位：微秒）。
min_update_elapse	bigint	最小 update 的时间花费（单位：微秒）。
total_insert_elapse	bigint	总 insert 的时间花费（单位：微秒）。

名称	类型	描述
avg_insert_elapse	bigint	平均 insert 的时间花费（单位：微秒）。
max_insert_elapse	bigint	最大 insert 的时间花费（单位：微秒）。
min_insert_elapse	bigint	最小 insert 的时间花费（单位：微秒）。
total_delete_elapse	bigint	总 delete 的时间花费（单位：微秒）。
avg_delete_elapse	bigint	平均 delete 的时间花费（单位：微秒）。
max_delete_elapse	bigint	最大 delete 的时间花费（单位：微秒）。
min_delete_elapse	bigint	最小 delete 的时间花费（单位：微秒）。

3.2.7.6.SUMMARY_STATEMENT_COUNT

显示数据库汇聚各节点（数据库节点）当前时刻执行的五类语句（SELECT、INSERT、UPDATE、DELETE、MERGE INTO）和（DDL、DML、DCL）统计信息。

表 1 SUMMARY_STATEMENT_COUNT 字段

名称	类型	描述
user_name	text	用户名。
select_count	numeric	select 语句统计结果。
update_count	numeric	update 语句统计结果。
insert_count	numeric	insert 语句统计结果。
delete_count	numeric	delete 语句统计结果。
mergeinto_count	numeric	merge into 语句统计结果。
ddl_count	numeric	DDL 语句的数量。
dml_count	numeric	DML 语句的数量。
dcl_count	numeric	DCL 语句的数量。
total_select_elapse	numeric	总 select 的时间花费（单位：微秒）。
avg_select_elapse	bigint	平均 select 的时间花费（单位：微秒）。
max_select_elapse	bigint	最大 select 的时间花费（单位：微秒）。
min_select_elapse	bigint	最小 select 的时间花费（单位：微秒）。
total_update_elapse	numeric	总 update 的时间花费（单位：微秒）。
avg_update_elapse	bigint	平均 update 的时间花费（单位：微秒）。
max_update_elapse	bigint	最大 update 的时间花费（单位：微秒）。
min_update_elapse	bigint	最小 update 的时间花费（单位：微秒）。
total_insert_elapse	numeric	总 insert 的时间花费（单位：微秒）。

名称	类型	描述
avg_insert_elapse	bigint	平均 insert 的时间花费（单位：微秒）。
max_insert_elapse	bigint	最大 insert 的时间花费（单位：微秒）。
min_insert_elapse	bigint	最小 insert 的时间花费（单位：微秒）。
total_delete_elapse	numeric	总 delete 的时间花费（单位：微秒）。
avg_delete_elapse	bigint	平均 delete 的时间花费（单位：微秒）。
max_delete_elapse	bigint	最大 delete 的时间花费（单位：微秒）。
min_delete_elapse	bigint	最小 delete 的时间花费（单位：微秒）。

3.2.7.7.GLOBAL_STATEMENT_COMPLEX_HISTORY

显示各个节点执行作业结束后的负载管理记录。

表 1 GLOBAL_STATEMENT_COMPLEX_HISTORY 的字段

名称	类型	描述
datid	oid	连接后端的数据库 OID。
dbname	text	连接后端的数据库名称。
schemaname	text	模式的名称。
nodename	text	数据库进程名称。
username	text	连接到后端的用户名。
application_name	text	连接到后端的应用名。
client_addr	inet	连接到后端的客户端的 IP 地址。如果此字段是 null，它表明通过服务器机器上 UNIX 套接字连接客户端或者这是内部进程，如 autovacuum。
client_hostname	text	客户端的主机名，这个字段是通过 client_addr 的反向 DNS 查找得到。这个字段只有在启动 log_hostname 且使用 IP 连接时才非空。
client_port	integer	客户端用于与后端通讯的 TCP 端口号，

名称	类型	描述
		如果使用 Unix 套接字，则为-1。
query_band	text	用于标示作业类型，可通过 GUC 参数 query_band 进行设置，默认为空字符串。
block_time	bigint	语句执行前的阻塞时间，包含语句解析和优化时间，单位 ms。
start_time	timestamp with time zone	语句执行的开始时间。
finish_time	timestamp with time zone	语句执行的结束时间。
duration	bigint	语句实际执行的时间，单位 ms。
estimate_total_time	bigint	语句预估执行时间，单位 ms。
status	text	语句执行结束状态：正常为 finished，异常为 aborted。
abort_info	text	语句执行结束状态为 aborted 时显示异常信息。
resource_pool	text	用户使用的资源池。
control_group	text	语句所使用的 Cgroup。
estimate_memory	integer	语句预估使用内存。
min_peak_memory	integer	语句在数据库节点上的最小内存峰值，单位 MB。
max_peak_memory	integer	语句在数据库节点上的最大内存峰值，单位 MB。
average_peak_memory	integer	语句执行过程中的内存使用平均值，单位

名称	类型	描述
		MB。
memory_skew_percent	integer	语句数据库节点间的内存使用倾斜率。
spill_info	text	语句在数据库节点上的下盘信息： None：数据库节点均未下盘。 All：数据库节点均下盘。 [a:b]：数量为 b 个数据库节点中有 a 个数据库节点下盘。
min_spill_size	integer	若发生下盘，数据库节点上下盘的最小数据量，单位 MB，默认为 0。
max_spill_size	integer	若发生下盘，数据库节点上下盘的最大数据量，单位 MB，默认为 0。
average_spill_size	integer	若发生下盘，数据库节点上下盘的平均数据量，单位 MB，默认为 0。
spill_skew_percent	integer	若发生下盘，数据库节点间下盘倾斜率。
min_dn_time	bigint	语句在数据库节点上的最小执行时间，单位 ms。
max_dn_time	bigint	语句在数据库节点上的最大执行时间，单位 ms。
average_dn_time	bigint	语句在数据库节点上的平均执行时间，单位 ms。
dntime_skew_percent	integer	语句在数据库节点的执行时间倾斜率。
min_cpu_time	bigint	语句在数据库节点上的最小 CPU 时间，单位 ms。
max_cpu_time	bigint	语句在数据库节点上的最大 CPU 时间，单位 ms。
total_cpu_time	bigint	语句在数据库节点上的 CPU 总时间，单

名称	类型	描述
		位 ms。
cpu_skew_percent	integer	语句在数据库节点间的 CPU 时间倾斜率。
min_peak_iops	integer	语句在数据库节点上的每秒最小 IO 峰值（列存单位是次/s，行存单位是万次/s）。
max_peak_iops	integer	语句在数据库节点上的每秒最大 IO 峰值（列存单位是次/s，行存单位是万次/s）。
average_peak_iops	integer	语句在数据库节点上的每秒平均 IO 峰值（列存单位是次/s，行存单位是万次/s）。
iops_skew_percent	integer	语句在数据库节点间的 IO 倾斜率。
warning	text	主要显示如下几类告警信息： Spill file size large than 256MB。 Broadcast size large than 100MB。 Early spill。 Spill times is greater than 3。 Spill on memory adaptive。 Hash table conflict。
queryid	bigint	语句执行使用的内部 query id。
query	text	执行的语句。
query_plan	text	语句的执行计划。
node_group	text	语句所属用户对应的逻辑 PanWeiDB。
cpu_top1_node_name	text	cpu 使用率第 1 的节点名称。
cpu_top2_node_name	text	cpu 使用率第 2 的节点名称。

名称	类型	描述
cpu_top3_node_name	text	cpu 使用率第 3 的节点名称。
cpu_top4_node_name	text	cpu 使用率第 4 的节点名称。
cpu_top5_node_name	text	cpu 使用率第 5 的节点名称。
mem_top1_node_name	text	内存使用量第 1 的节点名称。
mem_top2_node_name	text	内存使用量第 2 的节点名称。
mem_top3_node_name	text	内存使用量第 3 的节点名称。
mem_top4_node_name	text	内存使用量第 4 的节点名称。
mem_top5_node_name	text	内存使用量第 5 的节点名称。
cpu_top1_value	bigint	cpu 使用率。
cpu_top2_value	bigint	cpu 使用率。
cpu_top3_value	bigint	cpu 使用率。
cpu_top4_value	bigint	cpu 使用率。
cpu_top5_value	bigint	cpu 使用率。
mem_top1_value	bigint	内存使用量。
mem_top2_value	bigint	内存使用量。
mem_top3_value	bigint	内存使用量。
mem_top4_value	bigint	内存使用量。
mem_top5_value	bigint	内存使用量。
top_mem_dn	text	内存使用量 topN 信息。
top_cpu_dn	text	cpu 使用量 topN 信息。

3.2.7.8.GLOBAL_STATEMENT_COMPLEX_HISTORY_TABLE

显示各个节点执行作业结束后的负载管理记录。此数据是从内核中转储到系统表中的数据。具体的字段请参考：系统表和系统视图->schema->DBE_PERF Schema->Query->GLOBAL_STATEMENT_COMPLEX_HISTORY 中的字段。

3.2.7.9.STATEMENT_COMPLEX_HISTORY_TABLE

STATEMENT_COMPLEX_HISTORY_TABLE 系统表显示数据库主节点执行作业结束后的负载管理记录。此数据是从内核中转储到系统表中的数据。具体的字段请参考 GS_WLM_SESSION_HISTORY 的表 1。

3.2.7.10.GLOBAL_STATEMENT_COMPLEX_RUNTIME

显示当前用户在各个节点上正在执行的作业的负载管理记录。

表 1 GLOBAL_STATEMENT_COMPLEX_RUNTIME 的字段

名称	类型	描述
datid	oid	连接后端的数据 OID。
dbname	name	连接后端的数据库名称。
schemaname	text	模式的名称。
nodename	text	数据库进程名称
username	name	连接到后端的用户名。
application_name	text	连接到后端的应用名。
client_addr	inet	连接到后端的客户端的 IP 地址。如果此字段是 null，它表明通过服务器机器上 UNIX 套接字连接客户端或者这是内部进程，如 autovacuum。
client_hostname	text	客户端的主机名，这个字段是通过 client_addr 的反向 DNS 查找得到。这个字段只有在启动 log_hostname 且使用 IP 连接时才非空。

名称	类型	描述
client_port	integer	客户端用于与后端通讯的 TCP 端口号，如果使用 Unix 套接字，则为-1。
query_band	text	用于标示作业类型，可通过 GUC 参数 query_band 进行设置，默认为空字符串。
pid	bigint	后端线程 ID。
block_time	bigint	语句执行前的阻塞时间，单位 ms。
start_time	timestamp with time zone	语句执行的开始时间。
duration	bigint	语句已经执行的时间，单位 ms。
estimate_total_time	bigint	语句执行预估总时间，单位 ms。
estimate_left_time	bigint	语句执行预估剩余时间，单位 ms。
enqueue	text	工作负载管理资源状态。
resource_pool	name	用户使用的资源池。
control_group	text	语句所使用的 Cgroup。
estimate_memory	integer	语句预估使用内存，单位 MB。
min_peak_memory	integer	语句在数据库节点上的最小内存峰值，单位 MB。
max_peak_memory	integer	语句在数据库节点上的最大内存峰值，单位 MB。
average_peak_memory	integer	语句执行过程中的内存使用平均值，单位 MB。
memory_skew_percent	integer	语句在数据库节点间的内存使用倾斜率。

名称	类型	描述
spill_info	text	语句在数据库节点上的下盘信息： None：数据库节点均未下盘。 All：数据库节点均下盘。 [a:b]：数量为 b 个数据库节点中有 a 个数据库节点下盘。
min_spill_size	integer	若发生下盘，数据库节点上下盘的最小数据量，单位 MB，默认为 0。
max_spill_size	integer	若发生下盘，数据库节点上下盘的最大数据量，单位 MB，默认为 0。
average_spill_size	integer	若发生下盘，数据库节点上下盘的平均数据量，单位 MB，默认为 0。
spill_skew_percent	integer	若发生下盘，数据库节点间下盘倾斜率。
min_dn_time	bigint	语句在数据库节点上的最小执行时间，单位 ms。
max_dn_time	bigint	语句在数据库节点上的最大执行时间，单位 ms。
average_dn_time	bigint	语句在数据库节点上的平均执行时间，单位 ms。
dntime_skew_percent	integer	语句在数据库节点的执行时间倾斜率。
min_cpu_time	bigint	语句在数据库节点上的最小 CPU 时间，单位 ms。
max_cpu_time	bigint	语句在数据库节点上的最大 CPU 时间，单位 ms。
total_cpu_time	bigint	语句在数据库节点上的 CPU 总时间，单位 ms。
cpu_skew_percent	integer	语句在数据库节点间的 CPU 时间倾斜率。

名称	类型	描述
min_peak_iops	integer	语句在数据库节点上的每秒最小 IO 峰值（列存单位是次/s，行存单位是万次/s）。
max_peak_iops	integer	语句在数据库节点上的每秒最大 IO 峰值（列存单位是次/s，行存单位是万次/s）。
average_peak_iops	integer	语句在数据库节点上的每秒平均 IO 峰值（列存单位是次/s，行存单位是万次/s）。
iops_skew_percent	integer	语句在数据库节点间的 IO 倾斜率。
warning	text	主要显示如下几类告警信息： Spill file size large than 256MB。 Broadcast size large than 100MB。 Early spill。 Spill times is greater than 3。 Spill on memory adaptive。 Hash table conflict。
queryid	bigint	语句执行使用的内部 query id。
query	text	正在执行的语句。
query_plan	text	语句的执行计划。
node_group	text	语句所属用户对应的逻辑 PanWeiDB。
top_cpu_dn	text	cpu 使用量 topN 信息。
top_mem_dn	text	内存使用量 topN 信息。

3.2.7.11.STATEMENT_RESPONSETIME_PERCENTILE

获取 PanWeiDBSQL 响应时间 P80，P95 分布信息。

表 1 STATEMENT_RESPONSETIME_PERCENTILE 的字段

名称	类型	描述
p80	bigint	PanWeiDB80%的 SQL 的响应时间（单位：微秒）。
p95	bigint	PanWeiDB95%的 SQL 的响应时间（单位：微秒）。

3.2.7.12.STATEMENT_COMPLEX_RUNTIME

STATEMENT_COMPLEX_RUNTIME 视图显示当前用户在数据库主节点上正在执行的作业的负载管理记录。

表 1 STATEMENT_COMPLEX_RUNTIME 的字段

名称	类型	描述
datid	oid	连接后端的数据 OID。
dbname	name	连接后端的数据库名称。
schemaname	text	模式的名称。
nodename	text	数据库进程名称。
username	name	连接到后端的用户名。
application_name	text	连接到后端的应用名。
client_addr	inet	连接到后端的客户端的 IP 地址。 如果此字段是 null，它表明通过服务器机器上 UNIX 套接字连接客户端或者这是内部进程，如 autovacuum。
client_hostname	text	客户端的主机名，这个字段是通过 client_addr 的反向 DNS 查找得到。这个字段只有在启动 log_hostname 且使用 IP 连接时才非空。
client_port	integer	客户端用于与后端通讯的 TCP 端口号，如果使用 Unix 套接字，则为-1。
query_band	text	用于标示作业类型，可通过 GUC 参数

名称	类型	描述
		query_band 进行设置，默认为空字符串。
pid	bigint	后端线程 ID。
block_time	bigint	语句执行前的阻塞时间，单位 ms。
start_time	timestamp with time zone	语句执行的开始时间。
duration	bigint	语句已经执行的时间，单位 ms。
estimate_total_time	bigint	语句执行预估总时间，单位 ms。
estimate_left_time	bigint	语句执行预估剩余时间，单位 ms。
enqueue	text	工作负载管理资源状态。
resource_pool	name	用户使用的资源池。
control_group	text	语句所使用的 Cgroup。
estimate_memory	integer	语句预估使用内存，单位 MB。
min_peak_memory	integer	语句在数据库节点上的最小内存峰值，单位 MB。
max_peak_memory	integer	语句在数据库节点上的最大内存峰值，单位 MB。
average_peak_memory	integer	语句执行过程中的内存使用平均值，单位 MB。
memory_skew_percent	integer	语句在数据库节点间的内存使用倾斜率。
spill_info	text	语句在数据库节点上的下盘信息： None：数据库节点均未下盘。 All：数据库节点均下盘。 [a:b]：数量为 b 个数据库节点中有 a 个数据库节点下盘。

名称	类型	描述
min_spill_size	integer	若发生下盘，数据库节点上下盘的最小数据量，单位 MB，默认为 0。
max_spill_size	integer	若发生下盘，数据库节点上下盘的最大数据量，单位 MB，默认为 0。
average_spill_size	integer	若发生下盘，数据库节点上下盘的平均数据量，单位 MB，默认为 0。
spill_skew_percent	integer	若发生下盘，数据库节点间下盘倾斜率。
min_dn_time	bigint	语句在数据库节点上的最小执行时间，单位 ms。
max_dn_time	bigint	语句在数据库节点上的最大执行时间，单位 ms。
average_dn_time	bigint	语句在数据库节点上的平均执行时间，单位 ms。
dntime_skew_percent	integer	语句在数据库节点的执行时间倾斜率。
min_cpu_time	bigint	语句在数据库节点上的最小 CPU 时间，单位 ms。
max_cpu_time	bigint	语句在数据库节点上的最大 CPU 时间，单位 ms。
total_cpu_time	bigint	语句在数据库节点上的 CPU 总时间，单位 ms。
cpu_skew_percent	integer	语句在数据库节点间的 CPU 时间倾斜率。
min_peak_iops	integer	语句在数据库节点上的每秒最小 IO 峰值（列存单位是次/s，行存单位是万次/s）。
max_peak_iops	integer	语句在数据库节点上的每秒最大 IO 峰值（列存单位是次/s，行存单位是万次/s）。
average_peak_iops	integer	语句在数据库节点上的每秒平均 IO 峰值

名称	类型	描述
		（列存单位是次/s，行存单位是万次/s）。
iops_skew_percent	integer	语句在数据库节点间的 IO 倾斜率。
warning	text	主要显示如下几类告警信息： Spill file size large than 256MB。 Broadcast size large than 100MB。 Early spill。 Spill times is greater than 3。 Spill on memory adaptive。 Hash table conflict。
queryid	bigint	语句执行使用的内部 query id。
query	text	正在执行的语句。
query_plan	text	语句的执行计划。
node_group	text	语句所属用户对应的逻辑 PanWeiDB。
top_cpu_dn	text	cpu 使用量 topN 信息。
top_mem_dn	text	内存使用量 topN 信息。

3.2.7.13.STATEMENT_WLMSTAT_COMPLEX_RUNTIME

STATEMENT_WLMSTAT_COMPLEX_RUNTIME 视图显示和当前用户执行作业正在运行时的负载管理相关信息。

表 1 STATEMENT_WLMSTAT_COMPLEX_RUNTIME 字段

名称	类型	描述
datid	oid	连接后端的数据库 OID。
datname	name	连接后端的数据库名称。
threadid	bigint	后端线程 ID。
processid	integer	后端线程的 pid。
usesysid	oid	登录后端的用户 OID。

名称	类型	描述
appname	text	连接到后端的应用名。
usename	name	登录到该后端的用户名。
priority	bigint	语句所在 Cgroups 的优先级。
attribute	text	语句的属性： ❖ Ordinary：语句发送到数据库后被解析前的默认属性。 ❖ Simple：简单语句。 ❖ Complicated：复杂语句。 ❖ Internal：数据库内部语句。
block_time	bigint	语句当前为止的 pending 的时间，单位 s。
elapsed_time	bigint	语句当前为止的实际执行时间，单位 s。
total_cpu_time	bigint	语句在上一时间周期内的数据库节点上 CPU 使用的总时间，单位 s。
cpu_skew_percent	integer	语句在上一时间周期内的数据库节点上 CPU 使用的倾斜率。
statement_mem	integer	语句执行使用的 statement_mem，预留字段。
active_points	integer	语句占用的资源池并发点数。
dop_value	integer	语句的从资源池中获取的 dop 值。
control_group	text	该字段不支持。
status	text	该字段不支持。
enqueue	text	语句当前的排队情况，包括： ❖ Global：在全局队列中排队。 ❖ Respool：在资源池队列中排队。 ❖ CentralQueue：在中心协调节点（CCN）中排队。 ❖ Transaction：语句处于一个事务块中。 ❖ StoredProc：句处于一个存储过程中。 ❖ None：未在排队。 ❖ Forced None：事务块语句或存储过程语句由于超出设定的等待时间而强制执行。
resource_pool	name	语句当前所在的资源池。
query	text	该后端的最新查询。如果 state 状态是 active（活的），此字段显示当前正在执行的查询。所有其他情况表示上一个查询。
is_plana	boolean	逻辑 PanWeiDB 模式下，语句当前是否占

名称	类型	描述
		用其他逻辑 PanWeiDB 的资源执行。该值默认为 f（否）。
node_group	text	语句所属用户对应的逻辑 PanWeiDB。

3.2.7.14.GS_SLOW_QUERY_INFO（废弃）

GS_SLOW_QUERY_INFO 视图显示当前节点上已经转储的慢查询信息。此数据是从内核中转储到系统表中的数据。当设置 GUC 参数 enable_resource_record 为 on 时，系统会定时（周期为 3 分钟）将内核中 query 信息导入 GS_WLM_SESSION_QUERY_INFO_ALL 系统表，开启此功能会占用系统存储空间并对性能有一定影响。用户通过查询 GS_SLOW_QUERY_INFO 视图，可以查看已经转储的慢查询信息，本版本中已废弃。

表 1 GS_SLOW_QUERY_INFO 字段

名称	类型	描述
dbname	text	数据库名称。
schemaname	text	schema 名称。
nodename	text	节点名称。
username	text	用户名。
queryid	bigint	归一化 ID。
query	text	query 语句。
start_time	timestamp with time zone	开始执行时间。
finish_time	timestamp with time zone	结束执行时间。
duration	bigint	执行持续时间（毫秒）。
query_plan	text	计划信息。

名称	类型	描述
n_returned_rows	bigint	Select 返回的结果集行数。
n_tuples_fetched	bigint	随机扫描行数。
n_tuples_returned	bigint	顺序扫描行数。
n_tuples_inserted	bigint	插入行数。
n_tuples_updated	bigint	更新行数。
n_tuples_deleted	bigint	删除行数。
n_blocks_fetched	bigint	Cache 加载次数。
n_blocks_hit	bigint	Cache 命中数。
db_time	bigint	有效的 DB 时间花费，多线程将累加（单位：微秒）。
cpu_time	bigint	CPU 时间（单位：微秒）。
execution_time	bigint	执行器内执行时间（单位：微秒）。
parse_time	bigint	SQL 解析时间（单位：微秒）。
plan_time	bigint	SQL 生成计划时间（单位：微秒）。
rewrite_time	bigint	SQL 重写时间（单位：微秒）。
pl_execution_time	bigint	plpgsql 上的执行时间（单位：微秒）。
pl_compilation_time	bigint	plpgsql 上的编译时间

名称	类型	描述
		（单位：微秒）。
net_send_time	bigint	网络上的时间花费（单位：微秒）。
data_io_time	bigint	IO 上的时间花费(单位：微秒)。

3.2.7.15.GS_SLOW_QUERY_HISTORY（废弃）

GS_SLOW_QUERY_HISTORY 显示当前节点上未转储的慢查询信息。具体字段信息请参考 GS_SLOW_QUERY_INFO。该视图只有 system admin 和 monitor admin 用户有权限查询，本版本中已废弃。

3.2.7.16.GLOBAL_SLOW_QUERY_HISTORY（废弃）

GS_SLOW_QUERY_HISTORY 显示所有节点上未转储的慢查询信息，本版本中已废弃。具体字段信息请参考 GS_SLOW_QUERY_INFO。

3.2.7.17.GLOBAL_SLOW_QUERY_INFO（废弃）

GS_SLOW_QUERY_HISTORY 显示所有节点上已经转储的慢查询信息，本版本中已废弃。具体字段信息请参考 GS_SLOW_QUERY_INFO。

3.2.8.Lock

3.2.8.1.GLOBAL_LOCKS

GLOBAL_LOCKS 视图用于查看各节点各打开事务所持有的锁信息。

名称	类型	描述
node_name	name	数据库进程名称。
locktype	text	被锁定对象的类型：relation、extend、page、tuple、transactionid、virtualxid、object、userlock、advisory。

名称	类型	描述
database	oid	被锁定对象所在数据库的 OID： 如果被锁定的对象是共享对象，则 OID 为 0。 如果是一个事务 ID，则为 NULL。
relation	oid	关系的 OID，如果锁定的对象不是关系，也不是关系的一部分，则为 NULL。
page	integer	关系内部的页面编号，如果对象不是关系页或者不是行页，则为 NULL。
tuple	smallint	页面里边的行编号，如果对象不是行，则为 NULL。
virtualxid	text	事务的虚拟 ID，如果对象不是一个虚拟事务 ID，则为 NULL。
transactionid	xid	事务的 ID，如果对象不是一个事务 ID，则为 NULL。
classid	oid	包含该对象的系统表的 OID，如果对象不是普通的数据库对象，则为 NULL。
objid	oid	对象在其系统表内的 OID，如果对象不是普通数据库对象，则为 NULL。
objsubid	smallint	对于表的一个字段，这是字段编号；对于其他对象类型，这个字段是零；如果这个对象不是普通数据库对象，则为 NULL。
virtualtransaction	text	持有此锁或者在等待此锁的事务的虚拟 ID。
pid	bigint	持有或者等待这个锁的服务器线程的逻辑 ID。如果锁是被一个预备事务持有的，则为 NULL。
mode	text	这个线程持有的或者是期望的锁模式。
granted	boolean	如果锁是持有锁，则为 TRUE。 如果锁是等待锁，则为 FALSE。

名称	类型	描述
fastpath	boolean	如果通过 fast-path 获得锁，则为 TRUE；如果通过主要的锁表获得，则为 FALSE。

3.2.8.2.LOCKS

LOCKS 视图用于查看各打开事务所持有的锁信息。

表 1 LOCKS 字段

名称	类型	描述
locktype	text	被锁定对象的类型：relation、extend、page、tuple、transactionid、virtualxid、object、userlock、advisory。
database	oid	被锁定对象所在数据库的 OID： 如果被锁定的对象是共享对象，则 OID 为 0。 如果是一个事务 ID，则为 NULL。
relation	oid	关系的 OID，如果锁定的对象不是关系，也不是关系的一部分，则为 NULL。
page	integer	关系内部的页面编号，如果对象不是关系页或者不是行页，则为 NULL。
tuple	smallint	页面里边的行编号，如果对象不是行，则为 NULL。
bucket	integer	哈希桶号。
virtualxid	text	事务的虚拟 ID，如果对象不是一个虚拟事务 ID，则为 NULL。
transactionid	xid	事务的 ID，如果对象不是一个事务 ID，则为 NULL。
classid	oid	包含该对象的系统表的 OID，如果对象不是普通的数据库对象，则为 NULL。

名称	类型	描述
objid	oid	对象在其系统表内的 OID，如果对象不是普通数据库对象，则为 NULL。
objsubid	smallint	对于表的一个字段，这是字段编号；对于其他对象类型，这个字段是 0；如果这个对象不是普通数据库对象，则为 NULL。
virtualtransaction	text	持有此锁或者在等待此锁的事务的虚拟 ID。
pid	bigint	持有或者等待这个锁的服务器线程的逻辑 ID。如果锁是被一个预备事务持有的，则为 NULL。
sessionid	bigint	持有或者等待这个锁的会话 ID。如果锁是被一个预备事务持有的，则为 NULL。
mode	text	这个线程持有的或者是期望的锁模式。
granted	boolean	如果锁是持有锁，则为 TRUE。 如果锁是等待锁，则为 FALSE。
fastpath	boolean	如果通过 fast-path 获得锁，则为 TRUE；如果通过主要的锁表获得，则为 FALSE。
locktag	text	会话等待锁信息，可通过 locktag_decode()函数解析。
global_sessionid	text	全局会话 ID。

3.2.9.Wait Events

3.2.9.1.WAIT_EVENTS

WAIT_EVENTS 显示当前节点的 event 的等待相关的统计信息。具体事件信息见参考：系统表和系统视图->系统视图->PG_THREAD_WAIT_STATUS 章节等待状态列表、轻量级锁等待事件列表、IO 等待事件列表和事务锁等待事件列表。关于每种事务锁对业务的影响程度，请参考 LOCK 语法小节的详细描述。

表 1 WAIT_EVENTS 字段

名称	类型	描述
nodename	text	数据库进程名称。
type	text	event 类型。
event	text	event 名称。
wait	bigint	等待次数。
failed_wait	bigint	失败的等待次数。
total_wait_time	bigint	总等待时间（单位：微秒）。
avg_wait_time	bigint	平均等待时间（单位：微秒）。
max_wait_time	bigint	最大等待时间（单位：微秒）。
min_wait_time	bigint	最小等待时间（单位：微秒）。
last_updated	timestamp with time zone	最后一次更新该事件的时间。

3.2.9.2.GLOBAL_WAIT_EVENTS

GLOBAL_WAIT_EVENTS 视图显示各节点的 event 的等待相关的统计信息。

表 1 GLOBAL_WAIT_EVENTS 字段

名称	类型	描述
nodename	text	数据库进程名称。
type	text	event 类型。
event	text	event 名称。
wait	bigint	等待次数。
failed_wait	bigint	失败的等待次数。
total_wait_time	bigint	总等待时间（单位：微秒）。

名称	类型	描述
avg_wait_time	bigint	平均等待时间（单位：微秒）。
max_wait_time	bigint	最大等待时间（单位：微秒）。
min_wait_time	bigint	最小等待时间（单位：微秒）。
last_updated	timestamp with time zone	最后一次更新该事件的时间。

3.2.10.Configuration

3.2.10.1.CONFIG_SETTINGS

CONFIG_SETTINGS 视图显示数据库运行时参数的相关信息。

表 1 CONFIG_SETTINGS 字段

名称	类型	描述
name	text	参数名称。
setting	text	参数当前值。
unit	text	参数的隐式结构。
category	text	参数的逻辑组。
short_desc	text	参数的简单描述。
extra_desc	text	参数的详细描述。
context	text	设置参数值的上下文，包括 internal、postmaster、sighup、backend、superuser、user。
vartype	text	参数类型，包括 bool、enum、integer、real、string。
source	text	参数的赋值方式。
min_val	text	参数最大值。如果参数类型不是数值型，那么该字段值

名称	类型	描述
		为 null。
max_val	text	参数最小值。如果参数类型不是数值型，那么该字段值为 null。
enumvals	text[]	enum 类型参数合法值。如果参数类型不是 enum 型，那么该字段值为 null。
boot_val	text	数据库启动时参数默认值。
reset_val	text	数据库重置时参数默认值。
sourcefile	text	设置参数值的配置文件。如果参数不是通过配置文件赋值，那么该字段值为 null。
sourceline	integer	设置参数值的配置文件的行号。如果参数不是通过配置文件赋值，那么该字段值为 null。

3.2.10.2.GLOBAL_CONFIG_SETTINGS

GLOBAL_CONFIG_SETTINGS 显示各节点数据库运行时参数的相关信息。

表 1 GLOBAL_CONFIG_SETTINGS 的字段

名称	类型	描述
node_name	text	数据库进程名称。
name	text	参数名称。
setting	text	参数当前值。
unit	text	参数的隐式结构。
category	text	参数的逻辑组。
short_desc	text	参数的简单描述。
extra_desc	text	参数的详细描述。
context	text	设置参数值的上下文，包括 internal、postmaster、

名称	类型	描述
		sighup、backend、superuser、user。
vartype	text	参数类型，包括 bool、enum、integer、real、string。
source	text	参数的赋值方式。
min_val	text	参数最小值。如果参数类型不是数值型，那么该字段值为 null。
max_val	text	参数最大值。如果参数类型不是数值型，那么该字段值为 null。
enumvals	text[]	enum 类型参数合法值。如果参数类型不是 enum 型，那么该字段值为 null。
boot_val	text	数据库启动时参数默认值。
reset_val	text	数据库重置时参数默认值。
sourcefile	text	设置参数值的配置文件。如果参数不是通过配置文件赋值，那么该字段值为 null。
sourceline	integer	设置参数值的配置文件的行号。如果参数不是通过配置文件赋值，那么该字段值为 null。

3.2.11.Operator

3.2.11.1.OPERATOR_HISTORY_TABLE

OPERATOR_HISTORY_TABLE 系统表显示执行作业结束后的算子相关的记录。此数据是从内核中转储到系统表中的数据。

表 1 OPERATOR_HISTORY_TABLE 的字段

名称	类型	描述
queryid	bigint	语句执行使用的内部 query_id。
pid	bigint	后端线程 id。

名称	类型	描述
plan_node_id	integer	查询对应的执行计划的 plan node id。
plan_node_name	text	对应于 plan_node_id 的算子的名称。
start_time	timestamp with time zone	该算子处理第一条数据的开始时间。
duration	bigint	该算子到结束时候总的执行时间（ms）。
query_dop	integer	当前算子执行时的并行度。
estimated_rows	bigint	优化器估算的行数信息。
tuple_processed	bigint	当前算子返回的元素个数。
min_peak_memory	integer	当前算子在数据库节点上的最小内存峰值（MB）。
max_peak_memory	integer	当前算子在数据库节点上的最大内存峰值（MB）。
average_peak_memory	integer	当前算子在数据库节点上的平均内存峰值（MB）。
memory_skew_percent	integer	当前算子在数据库节点间的内存使用倾斜率。
min_spill_size	integer	若发生下盘，数据库节点上下盘的最小数据量（MB），默认为 0。
max_spill_size	integer	若发生下盘，数据库节点上下盘的最大数据量（MB），默认为 0。
average_spill_size	integer	若发生下盘，数据库节点上下盘的平均数据量（MB），默认为 0。
spill_skew_percent	integer	若发生下盘，数据库节点间下盘倾斜率。
min_cpu_time	bigint	该算子在数据库节点上的最小执行时间（ms）。
max_cpu_time	bigint	该算子在数据库节点上的最大执行时间（ms）。
total_cpu_time	bigint	该算子在数据库节点上的总执行时间（ms）。
cpu_skew_percent	integer	数据库节点间执行时间的倾斜率。
warning	text	主要显示如下几类告警信息： ❖ Sort/SetOp/HashAgg/HashJoin

名称	类型	描述
		spill。 ❖ Spill file size large than 256MB ❖ Broadcast size large than 100MB。 ❖ Early spill。 ❖ Spill times is greater than 3。 ❖ Spill on memory adaptive。 ❖ Hash table conflict。

3.2.11.2.OPERATOR_HISTORY

OPERATOR_HISTORY 视图显示的是当前用户数据库主节点上执行作业结束后的算子的相关记录。记录的数据同 GS_WLM_OPERATOR_INFO 表（参考：系统表和系统视图->系统表->GS_WLM_OPERATOR_INFO 章节）。

3.2.11.3.OPERATOR_RUNTIME

OPERATOR_RUNTIME 视图显示当前用户正在执行的作业的算子相关信息。

表 1 OPERATOR_RUNTIME 的字段

名称	类型	描述
queryid	bigint	语句执行使用的内部 query_id。
pid	bigint	后端线程 id。
plan_node_id	integer	查询对应的执行计划的 plan node id。
plan_node_name	text	对应于 plan_node_id 的算子的名称。
start_time	timestamp with time zone	该算子处理第一条数据的开始时间。
duration	bigint	该算子到结束时候总的执行时间（ms）。
status	text	当前算子的执行状态，包括 finished 和 running。
query_dop	integer	当前算子执行时的并行度。
estimated_rows	bigint	优化器估算的行数信息。
tuple_processed	bigint	当前算子返回的元素个数。
min_peak_memory	integer	当前算子在数据库节点上的最小内存峰值（MB）。

名称	类型	描述
max_peak_memory	integer	当前算子在数据库节点上的最大内存峰值（MB）。
average_peak_memory	integer	当前算子在数据库节点上的平均内存峰值（MB）。
memory_skew_percent	integer	当前算子在数据库节点的内存使用倾斜率。
min_spill_size	integer	若发生下盘，数据库节点上下盘的最小数据量（MB），默认为 0。
max_spill_size	integer	若发生下盘，数据库节点上下盘的最大数据量（MB），默认为 0。
average_spill_size	integer	若发生下盘，数据库节点上下盘的平均数据量（MB），默认为 0。
spill_skew_percent	integer	若发生下盘，数据库节点间下盘倾斜率。
min_cpu_time	bigint	该算子在数据库节点上的最小执行时间（ms）。
max_cpu_time	bigint	该算子在数据库节点上的最大执行时间(ms)。
total_cpu_time	bigint	该算子在数据库节点上的总执行时间(ms)。
cpu_skew_percent	integer	数据库节点间执行时间的倾斜率。
warning	text	主要显示如下几类告警信息： ❖ Sort/SetOp/HashAgg/HashJoin spill。 ❖ Spill file size large than 256MB ❖ Broadcast size large than 100MB。 ❖ Early spill。 ❖ Spill times is greater than 3。 ❖ Spill on memory adaptive。 ❖ Hash table conflict。

3.2.11.4.GLOBAL_OPERATOR_HISTORY

GLOBAL_OPERATOR_HISTORY 系统视图显示的是当前用户在数据库主节点上执行作业结束后的算子的相关记录。

表 1 GLOBAL_OPERATOR_HISTORY 的字段

名称	类型	描述
queryid	bigint	语句执行使用的内部 query_id。
pid	bigint	后端线程 id。
plan_node_id	integer	查询对应的执行计划的 plan node id。
plan_node_name	text	对应于 plan_node_id 的算子的名称。
start_time	timestamp with time zone	该算子处理第一条数据的开始时间。
duration	bigint	该算子到结束时候总的执行时间(ms)。
query_dop	integer	当前算子执行时的并行度。
estimated_rows	bigint	优化器估算的行数信息。
tuple_processed	bigint	当前算子返回的元素个数。
min_peak_memory	integer	当前算子在数据库节点上的最小内存峰值(MB)。
max_peak_memory	integer	当前算子在数据库节点上的最大内存峰值(MB)。
average_peak_memory	integer	当前算子在数据库节点上的平均内存峰值(MB)。
memory_skew_percent	integer	当前算子在数据库节点间的内存使用倾斜率。
min_spill_size	integer	若发生下盘，数据库节点上下盘的最小数据量(MB)，默认为 0。
max_spill_size	integer	若发生下盘，数据库节点上下盘的最大数据量(MB)，默认为 0。
average_spill_size	integer	若发生下盘，数据库节点上下盘的平均数据量(MB)，默认为 0。
spill_skew_percent	integer	若发生下盘，数据库节点间下盘倾斜率。
min_cpu_time	bigint	该算子在数据库节点上的最小执行时间(ms)。
max_cpu_time	bigint	该算子在数据库节点上的最大执行时间(ms)。
total_cpu_time	bigint	该算子在数据库节点上的总执行时间(ms)。

名称	类型	描述
cpu_skew_percent	integer	数据库节点间执行时间的倾斜率。
warning	text	主要显示如下几类告警信息： ❖ Sort/SetOp/HashAgg/HashJoin spill。 ❖ Spill file size large than 256MB ❖ Broadcast size large than 100MB。 ❖ Early spill。 ❖ Spill times is greater than 3。 ❖ Spill on memory adaptive。 ❖ Hash table conflict。

3.2.11.5.GLOBAL_OPERATOR_HISTORY_TABLE

GLOBAL_OPERATOR_HISTORY_TABLE 视图显示数据库主节点执行作业结束后的算子相关的记录。此数据是从内核中转储到系统表 GS_WLM_OPERATOR_INFO 中的数据。该视图是查询数据库主节点系统表 GS_WLM_OPERATOR_INFO 的汇聚视图。表字段同 GLOBAL_OPERATOR_HISTORY 表 1（参考：系统表和系统视图->schema->DBE_PERF Schema->OperatorGLOBAL_OPERATOR_HISTORY 章节）。

3.2.11.6.GLOBAL_OPERATOR_RUNTIME

GLOBAL_OPERATOR_RUNTIME 视图显示当前用户在数据库主节点上正在执行的作业的算子相关信息。

表 1 GLOBAL_OPERATOR_RUNTIME 的字段

名称	类型	描述
queryid	bigint	语句执行使用的内部 query_id。
pid	bigint	后端线程 id。
plan_node_id	integer	查询对应的执行计划的 plan node id。
plan_node_name	text	对应于 plan_node_id 的算子的名称。
start_time	timestamp with time zone	该算子处理第一条数据的开始时间。
duration	bigint	该算子到结束时候总的执行时间(ms)。

名称	类型	描述
status	text	当前算子的执行状态，包括 finished 和 running。
query_dop	integer	当前算子执行时的并行度。
estimated_rows	bigint	优化器估算的行数信息。
tuple_processed	bigint	当前算子返回的元素个数。
min_peak_memory	integer	当前算子在数据库节点上的最小内存峰值(MB)。
max_peak_memory	integer	当前算子在数据库节点上的最大内存峰值(MB)。
average_peak_memory	integer	当前算子在数据库节点上的平均内存峰值(MB)。
memory_skew_percent	integer	当前算子在数据库节点的内存使用倾斜率。
min_spill_size	integer	若发生下盘，数据库节点上下盘的最小数据量(MB)，默认为 0。
max_spill_size	integer	若发生下盘，数据库节点上下盘的最大数据量(MB)，默认为 0。
average_spill_size	integer	若发生下盘，数据库节点上下盘的平均数据量(MB)，默认为 0。
spill_skew_percent	integer	若发生下盘，数据库节点间下盘倾斜率。
min_cpu_time	bigint	该算子在数据库节点上的最小执行时间(ms)。
max_cpu_time	bigint	该算子在数据库节点上的最大执行时间(ms)。
total_cpu_time	bigint	该算子在数据库节点上的总执行时间(ms)。
cpu_skew_percent	integer	数据库节点间执行时间的倾斜率。
warning	text	主要显示如下几类告警信息： ❖ Sort/SetOp/HashAgg/HashJoin spill。 ❖ Spill file size large than 256MB ❖ Broadcast size large than 100MB。 ❖ Early spill。 ❖ Spill times is greater than 3。

名称	类型	描述
		❖ Spill on memory adaptive。 ❖ Hash table conflict。

3.2.12.Workload Manager

3.2.12.1.WLM_USER_RESOURCE_CONFIG

WLM_USER_RESOURCE_CONFIG 视图显示用户的资源配置信息。

表 1 WLM_USER_RESOURCE_CONFIG 字段

名称	类型	描述
userid	oid	用户 oid。
username	name	用户名称。
sysadmin	boolean	是否是 sysadmin。
rpoid	oid	资源池的 oid。
respool	name	资源池的名称。
parentid	oid	父用户的 oid。
totalspace	bigint	占用总空间大小。
spacelimit	bigint	空间大上限。
childcount	integer	子用户数量。
childlist	text	子用户的列表。

3.2.12.2.WLM_USER_RESOURCE_RUNTIME

WLM_USER_RESOURCE_RUNTIME 视图显示所有用户资源使用情况，需要使用管理员用户进行查询。此视图在 GUC 参数“use_workload_manager”为“on”时才有效。

表 1 WLM_USER_RESOURCE_RUNTIME 字段

名称	类型	描述
username	name	用户名。
used_memory	integer	正在使用的内存大小，单位 MB。
total_memory	integer	可以使用的内存大小，单位 MB。值为 0 表示未限制最大可用内存，其限制取决于数据库最大可用内存。
used_cpu	integer	正在使用的 CPU 核数。

名称	类型	描述
total_cpu	integer	在该机器节点上，用户关联控制组的 CPU 核数总和。
used_space	bigint	已使用的存储空间大小，单位 KB。
total_space	bigint	可使用的存储空间大小，单位 KB，值为-1 表示未限制最大存储空间。
used_temp_space	bigint	已使用的临时空间大小（预留字段，暂未使用），单位 KB。
total_temp_space	bigint	可使用的临时空间大小（预留字段，暂未使用），单位 KB，值为-1 表示未限制最大临时存储空间。
used_spill_space	bigint	已使用的下盘空间大小（预留字段，暂未使用），单位 KB。
total_spill_space	bigint	可使用的下盘空间大小（预留字段，暂未使用），单位 KB，值为-1 表示未限制最大下盘空间。

3.2.13.Global Plancache

GPC 相关视图在 enable_global_plancache 打开且线程池打开的状态下才有效。

3.2.13.1.GLOBAL_PLANCACHE_STATUS

GLOBAL_PLANCACHE_STATUS 视图显示 GPC 全局计划缓存状态信息。

表 1 GLOBAL_PLANCACHE_STATUS 字段

3.2.13.2.GLOBAL_PLANCACHE_CLEAN

GLOBAL_PLANCACHE_CLEAN 视图用于清理所有节点上无人使用的全局计划缓存。返回值为 Boolean 类型。

名称	类型	描述
nodename	text	所属节点名称。
query	text	查询语句 text。
refcount	integer	被引用次数。
valid	bool	是否合法。
databaseid	oid	所属数据库 id。
schema_name	text	所属 schema。

名称	类型	描述
params_num	integer	参数数量。
func_id	oid	该 plancache 所在存储过程 oid，如果不属于存储过程则为 0。

3.2.14.RTO

3.2.14.1.global_rto_status

global_rto_status 视图显示关于主机和备机的日志流控信息（本节点除外、备 DN 上不可使用）。

表 1 remote_rto_status 参数说明

参数	类型	描述
node_name	text	节点的名称，包含主机和备机。
rto_info	text	流控的信息，包含了备机当前的日志流控时间（单位：秒），备机通过 GUC 参数设置的预期流控时间（单位：秒），为了达到这个预期主机所需要的睡眠时间（单位：微秒）。

3.2.15.Cache/IO

3.2.15.1.STATIO_USER_TABLES

STATIO_USER_TABLES 视图显示命名空间中所有用户关系表的 I/O 状态信息。

表 1 STATIO_USER_TABLES 字段

名称	类型	描述
relid	oid	表 OID。
schemaname	name	该表模式名。
relname	name	表名。
heap_blks_read	bigint	从该表中读取的磁盘块数。
heap_blks_hit	bigint	该表缓存命中数。
idx_blks_read	bigint	从表中所有索引读取的磁盘块数。
idx_blks_hit	bigint	表中所有索引命中缓存数。
toast_blks_read	bigint	该表的 TOAST 表读取的磁盘块数（如果存在）。

名称	类型	描述
toast_blks_hit	bigint	该表的 TOAST 表命中缓冲区数（如果存在）。
tidx_blks_read	bigint	该表的 TOAST 表索引读取的磁盘块数（如果存在）。
tidx_blks_hit	bigint	该表的 TOAST 表索引命中缓冲区数（如果存在）。

3.2.15.2.SUMMARY_STATIO_USER_TABLES

SUMMARY_STATIO_USER_TABLES 视图显示 PanWeiDB 内汇聚的命名空间中所有用户关系表的 I/O 状态信息。

表 1 SUMMARY_STATIO_USER_TABLES 字段

名称	类型	描述
schemaname	name	该表模式名。
relname	name	表名。
heap_blks_read	numeric	从该表中读取的磁盘块数。
heap_blks_hit	numeric	此表缓存命中数。
idx_blks_read	numeric	从表中所有索引读取的磁盘块数。
idx_blks_hit	numeric	表中所有索引命中缓存数。
toast_blks_read	numeric	此表的 TOAST 表读取的磁盘块数（如果存在）。
toast_blks_hit	numeric	此表的 TOAST 表命中缓冲区数（如果存在）。
tidx_blks_read	numeric	此表的 TOAST 表索引读取的磁盘块数（如果存在）。
tidx_blks_hit	numeric	此表的 TOAST 表索引命中缓冲区数（如果存在）。

3.2.15.3.GLOBAL_STATIO_USER_TABLES

GLOBAL_STATIO_USER_TABLES 视图显示各节点的命名空间中所有用户关系表的 I/O 状态信息。

表 1 GLOBAL_STATIO_USER_TABLES 字段

名称	类型	描述
node_name	name	节点名称。
relid	oid	表 OID。
schemaname	name	该表模式名。
relname	name	表名。
heap_blks_read	bigint	从该表中读取的磁盘块数。
heap_blks_hit	bigint	此表缓存命中数。

名称	类型	描述
idx_blks_read	bigint	从表中所有索引读取的磁盘块数。
idx_blks_hit	bigint	表中所有索引命中缓存数。
toast_blks_read	bigint	此表的 TOAST 表读取的磁盘块数（如果存在）。
toast_blks_hit	bigint	此表的 TOAST 表命中缓冲区数（如果存在）。
tidx_blks_read	bigint	此表的 TOAST 表索引读取的磁盘块数（如果存在）。
tidx_blks_hit	bigint	此表的 TOAST 表索引命中缓冲区数（如果存在）。

3.2.15.4.STATIO_USER_INDEXES

STATIO_USER_INDEXES 视图显示当前节点命名空间中所有用户关系表索引的 I/O 状态信息。

表 1 STATIO_USER_INDEXES 字段

名称	类型	描述
relid	oid	索引的表的 OID。
indexrelid	oid	该索引的 OID。
schemaname	name	该索引的模式名。
relname	name	该索引的表名。
indexrelname	name	索引名称。
idx_blks_read	bigint	从索引中读取的磁盘块数。
idx_blks_hit	bigint	索引命中缓存数。

3.2.15.5.SUMMARY_STATIO_USER_INDEXES

SUMMARY_STATIO_USER_INDEXES 视图显示 PanWeiDB 内汇聚的命名空间中所有用户关系表索引的 I/O 状态信息。

表 1 SUMMARY_STATIO_USER_INDEXES 字段

名称	类型	描述
schemaname	name	该索引的模式名。
relname	name	该索引的表名。
indexrelname	name	索引名称。
idx_blks_read	numeric	从索引中读取的磁盘块数。
idx_blks_hit	numeric	索引命中缓存数。

3.2.15.6.GLOBAL_STATIO_USER_INDEXES

GLOBAL_STATIO_USER_INDEXES 视图显示各节点的命名空间中所有用户关系表索引的 I/O 状态信息。

表 1 GLOBAL_STATIO_USER_INDEXES 字段

名称	类型	描述
node_name	name	数据库进程名称。
relid	oid	索引的表的 OID。
indexrelid	oid	该索引的 OID。
schemaname	name	该索引的模式名。
relname	name	该索引的表名。
indexrelname	name	索引名称。
idx_blks_read	numeric	从索引中读取的磁盘块数。
idx_blks_hit	numeric	索引命中缓存数。

3.2.15.7.STATIO_USER_SEQUENCES

STATIO_USER_SEQUENCE 视图显示当前节点的命名空间中所有用户关系表类型为序列的 I/O 状态信息。

表 1 STATIO_USER_SEQUENCE 字段

名称	类型	描述
relid	oid	序列 OID。
schemaname	name	序列中模式名。
relname	name	序列名。
blks_read	bigint	从序列中读取的磁盘块数。
blks_hit	bigint	序列中缓存命中数。

3.2.15.8.SUMMARY_STATIO_USER_SEQUENCES

SUMMARY_STATIO_USER_SEQUENCES 视图显示 PanWeiDB 内汇聚的命名空间中所有用户关系表类型为序列的 I/O 状态信息。

表 1 SUMMARY_STATIO_USER_SEQUENCES 字段

名称	类型	描述

名称	类型	描述
schemaname	name	序列中模式名。
relname	name	序列名。
blks_read	numeric	从序列中读取的磁盘块数。
blks_hit	numeric	序列中缓存命中数。

3.2.15.9.GLOBAL_STATIO_USER_SEQUENCES

GLOBAL_STATIO_USER_SEQUENCES 视图显示各节点的命名空间中所有用户关系表类型为序列的 I/O 状态信息。

表 1 GLOBAL_STATIO_USER_SEQUENCES 字段

名称	类型	描述
node_name	name	数据库进程名称。
relid	oid	序列 OID。
schemaname	name	序列中模式名。
relname	name	序列名。
blks_read	bigint	从序列中读取的磁盘块数。
blks_hit	bigint	序列中缓存命中数。

3.2.15.10.STATIO_SYS_TABLES

STATIO_SYS_TABLES 视图显示命名空间中所有系统表的 I/O 状态信息。

表 1 STATIO_SYS_TABLES 字段

名称	类型	描述
relid	oid	表 OID。
schemaname	name	该表模式名。
relname	name	表名。
heap_blks_read	bigint	从该表中读取的磁盘块数。
heap_blks_hit	bigint	该表缓存命中数。
idx_blks_read	bigint	从表中所有索引读取的磁盘块数。
idx_blks_hit	bigint	表中所有索引命中缓存数。
toast_blks_read	bigint	该表的 TOAST 表读取的磁盘块数（如果存在）。
toast_blks_hit	bigint	该表的 TOAST 表命中缓冲区数（如果存在）。
tidx_blks_read	bigint	该表的 TOAST 表索引读取的磁盘块数（如果存在）。

名称	类型	描述
tidx_blks_hit	bigint	该表的 TOAST 表索引命中缓冲区数（如果存在）。

3.2.15.11.SUMMARY_STATIO_SYS_TABLES

SUMMARY_STATIO_SYS_TABLES 视图显示 PanWeiDB 内汇聚的命名空间中所有系统表的 I/O 状态信息。

表 1 SUMMARY_STATIO_SYS_TABLES 字段

名称	类型	描述
schemaname	name	该表模式名。
relname	name	表名。
heap_blks_read	numeric	从该表中读取的磁盘块数。
heap_blks_hit	numeric	此表缓存命中数。
idx_blks_read	numeric	从表中所有索引读取的磁盘块数。
idx_blks_hit	numeric	表中所有索引命中缓存数。
toast_blks_read	numeric	此表的 TOAST 表读取的磁盘块数（如果存在）。
toast_blks_hit	numeric	此表的 TOAST 表命中缓冲区数（如果存在）。
tidx_blks_read	numeric	此表的 TOAST 表索引读取的磁盘块数（如果存在）。
tidx_blks_hit	numeric	此表的 TOAST 表索引命中缓冲区数（如果存在）。

3.2.15.12.GLOBAL_STATIO_SYS_TABLES

GLOBAL_STATIO_SYS_TABLES 视图显示各节点的命名空间中所有系统表的 I/O 状态信息。

表 1 GLOBAL_STATIO_SYS_TABLES 字段

名称	类型	描述
node_name	name	数据库进程名称。
relid	oid	表 OID。
schemaname	name	该表模式名。
relname	name	表名。
heap_blks_read	bigint	从该表中读取的磁盘块数。
heap_blks_hit	bigint	此表缓存命中数。
idx_blks_read	bigint	从表中所有索引读取的磁盘块数。
idx_blks_hit	bigint	表中所有索引命中缓存数。

名称	类型	描述
toast_blks_read	bigint	此表的 TOAST 表读取的磁盘块数（如果存在）。
toast_blks_hit	bigint	此表的 TOAST 表命中缓冲区数（如果存在）。
tidx_blks_read	bigint	此表的 TOAST 表索引读取的磁盘块数（如果存在）。
tidx_blks_hit	bigint	此表的 TOAST 表索引命中缓冲区数（如果存在）。

3.2.15.13.STATIO_SYS_INDEXES

STATIO_SYS_INDEXES 显示命名空间中所有系统表索引的 I/O 状态信息。

表 1 STATIO_SYS_INDEXES 字段

名称	类型	描述
relid	oid	索引的表的 OID。
indexrelid	oid	该索引的 OID。
schemaname	name	该索引的模式名。
relname	name	该索引的表名。
indexrelname	name	索引名称。
idx_blks_read	bigint	从索引中读取的磁盘块数。
idx_blks_hit	bigint	索引命中缓存数。

3.2.15.14.SUMMARY_STATIO_SYS_INDEXES

SUMMARY_STATIO_SYS_INDEXES 视图显示 PanWeiDB 内汇聚的命名空间中所有系统表索引的 I/O 状态信息。

表 1 SUMMARY_STATIO_SYS_INDEXES 字段

名称	类型	描述
schemaname	name	该索引的模式名。
relname	name	该索引的表名。
indexrelname	name	索引名称。
idx_blks_read	numeric	从索引中读取的磁盘块数。
idx_blks_hit	numeric	索引命中缓存数。

3.2.15.15.GLOBAL_STATIO_SYS_INDEXES

GLOBAL_STATIO_SYS_INDEXES 视图显示各节点的命名空间中所有系统表索引的 I/O 状态信息。

表 1 GLOBAL_STATIO_SYS_INDEXES 字段

名称	类型	描述
node_name	name	数据库进程名称。
relid	oid	索引的表的 OID。
indexrelid	oid	该索引的 OID。
schemaname	name	该索引的模式名。
relname	name	该索引的表名。
indexrelname	name	索引名称。
idx_blks_read	numeric	从索引中读取的磁盘块数。
idx_blks_hit	numeric	索引命中缓存数。

3.2.15.16.STATIO_SYS_SEQUENCES

STATIO_SYS_SEQUENCES 显示命名空间中所有系统表为序列的 I/O 状态信息。

表 1 STATIO_SYS_SEQUENCES 字段

名称	类型	描述
relid	oid	序列 OID。
schemaname	name	序列中模式名。
relname	name	序列名。
blks_read	bigint	从序列中读取的磁盘块数。
blks_hit	bigint	序列中缓存命中数。

3.2.15.17.SUMMARY_STATIO_SYS_SEQUENCES

SUMMARY_STATIO_SYS_SEQUENCES 视图显示 PanWeiDB 内汇聚的命名空间中所有系统表为序列的 I/O 状态信息。

表 1 SUMMARY_STATIO_SYS_SEQUENCES 字段

名称	类型	描述
schemaname	name	序列中模式名。
relname	name	序列名。
blks_read	numeric	从序列中读取的磁盘块数。
blks_hit	numeric	序列中缓存命中数。

3.2.15.18.GLOBAL_STATIO_SYS_SEQUENCES

GLOBAL_STATIO_SYS_SEQUENCES 视图显示各节点的命名空间中所有系统表为序列的 I/O 状态信息。

表 1 GLOBAL_STATIO_SYS_SEQUENCES 字段

名称	类型	描述
node_name	name	数据库进程名称。
relid	oid	序列 OID。
schemaname	name	序列中模式名。
relname	name	序列名。
blks_read	bigint	从序列中读取的磁盘块数。
blks_hit	bigint	序列中缓存命中数。

3.2.15.19.STATIO_ALL_TABLES

STATIO_ALL_TABLES 视图将包含数据库中每个表（包括 TOAST 表）的一行，显示出特定表 I/O 的统计。

表 1 STATIO_ALL_TABLES 字段

名称	类型	描述
relid	oid	表 OID。
schemaname	name	该表模式名。
relname	name	表名。
heap_blks_read	bigint	从该表中读取的磁盘块数。
heap_blks_hit	bigint	该表缓存命中数。
idx_blks_read	bigint	从表中所有索引读取的磁盘块数。
idx_blks_hit	bigint	表中所有索引命中缓存数。
toast_blks_read	bigint	该表的 TOAST 表读取的磁盘块数（如果存在）。
toast_blks_hit	bigint	该表的 TOAST 表命中缓冲区数（如果存在）。
tidx_blks_read	bigint	该表的 TOAST 表索引读取的磁盘块数（如果存在）。
tidx_blks_hit	bigint	该表的 TOAST 表索引命中缓冲区数（如果存在）。

3.2.15.20.SUMMARY_STATIO_ALL_TABLES

SUMMARY_STATIO_ALL_TABLES 视图将包含 PanWeiDB 内汇聚的数据库中每个表（包括 TOAST 表）的 I/O 的统计。

表 1 SUMMARY_STATIO_ALL_TABLES 字段

名称	类型	描述
schemaname	name	该表模式名。
relname	name	表名。
heap_blks_read	numeric	从该表中读取的磁盘块数。
heap_blks_hit	numeric	此表缓存命中数。
idx_blks_read	numeric	从表中所有索引读取的磁盘块数。
idx_blks_hit	numeric	表中所有索引命中缓存数。
toast_blks_read	numeric	此表的 TOAST 表读取的磁盘块数（如果存在）。
toast_blks_hit	numeric	此表的 TOAST 表命中缓冲区数（如果存在）。
tidx_blks_read	numeric	此表的 TOAST 表索引读取的磁盘块数（如果存在）。
tidx_blks_hit	numeric	此表的 TOAST 表索引命中缓冲区数（如果存在）。

3.2.15.21.GLOBAL_STATIO_ALL_TABLES

GLOBAL_STATIO_ALL_TABLES 视图将包含各节点的数据库中每个表（包括 TOAST 表）的 I/O 的统计。

表 1 GLOBAL_STATIO_ALL_TABLES 字段

名称	类型	描述
node_name	name	数据库进程名称。
relid	oid	表 OID。
schemaname	name	该表模式名。
relname	name	表名。
heap_blks_read	bigint	从该表中读取的磁盘块数。
heap_blks_hit	bigint	此表缓存命中数。
idx_blks_read	bigint	从表中所有索引读取的磁盘块数。
idx_blks_hit	bigint	表中所有索引命中缓存数。
toast_blks_read	bigint	此表的 TOAST 表读取的磁盘块数（如果存在）。
toast_blks_hit	bigint	此表的 TOAST 表命中缓冲区数（如果存在）。

名称	类型	描述
tidx_blks_read	bigint	此表的 TOAST 表索引读取的磁盘块数（如果存在）。
tidx_blks_hit	bigint	此表的 TOAST 表索引命中缓冲区数（如果存在）。

3.2.15.22.STATIO_ALL_INDEXES

STATIO_ALL_INDEXES 视图包含数据库中的每个索引行，显示特定索引的 I/O 的统计。

表 1 STATIO_ALL_INDEXES 字段

名称	类型	描述
relid	oid	索引的表的 OID。
indexrelid	oid	该索引的 OID。
schemaname	name	该索引的模式名。
relname	name	该索引的表名。
indexrelname	name	索引名称。
idx_blks_read	bigint	从索引中读取的磁盘块数。
idx_blks_hit	bigint	索引命中缓存数。

3.2.15.23.GLOBAL_STATIO_ALL_INDEXES

GLOBAL_STATIO_ALL_INDEXES 视图包含各节点的数据库中的每个索引行，显示特定索引的 I/O 的统计。

表 1 GLOBAL_STATIO_ALL_INDEXES 字段

名称	类型	描述
node_name	name	数据库进程名称。
relid	oid	索引的表的 OID。
indexrelid	oid	该索引的 OID。
schemaname	name	该索引的模式名。
relname	name	该索引的表名。
indexrelname	name	索引名称。
idx_blks_read	numeric	从索引中读取的磁盘块数。
idx_blks_hit	numeric	索引命中缓存数。

3.2.15.24.STATIO_ALL_SEQUENCES

STATIO_ALL_SEQUENCES 视图包含数据库中每个序列的每一行，显示特定序列关于 I/O 的统计。

表 1 STATIO_ALL_SEQUENCES 字段

名称	类型	描述
relid	oid	序列 OID。
schemaname	name	序列中模式名。
relname	name	序列名。
blks_read	bigint	从序列中读取的磁盘块数。
blks_hit	bigint	序列中缓存命中数。

3.2.15.25.SUMMARY_STATIO_ALL_SEQUENCES

SUMMARY_STATIO_ALL_SEQUENCES 视图包含 PanWeiDB 内汇聚的数据库中每个序列的每一行,显示特定序列关于 I/O 的统计。

表 1 SUMMARY_STATIO_ALL_SEQUENCES 字段

名称	类型	描述
schemaname	name	序列中模式名。
relname	name	序列名。
blks_read	numeric	从序列中读取的磁盘块数。
blks_hit	numeric	序列中缓存命中数。

3.2.15.26.GLOBAL_STATIO_ALL_SEQUENCES

GLOBAL_STATIO_ALL_SEQUENCES 包含各节点的数据库中每个序列的每一行，显示特定序列关于 I/O 的统计。

表 1 GLOBAL_STATIO_ALL_SEQUENCES 字段

名称	类型	描述
node_name	name	数据库进程名称。
relid	oid	序列 OID。
schemaname	name	序列中模式名。
relname	name	序列名。

名称	类型	描述
blks_read	bigint	从序列中读取的磁盘块数。
blks_hit	bigint	序列中缓存命中数。

3.2.15.27.GLOBAL_STAT_DB_CU

GLOBAL_STAT_DB_CU 视图用于查询 PanWeiDB，每个数据库的 CU 命中情况。可以通过 pg_stat_reset()进行清零。

表 1 GLOBAL_STAT_DB_CU 字段

名称	类型	描述
node_name1	text	数据库进程名称。
db_name	text	数据库名。
mem_hit	bigint	内存命中次数。
hdd_sync_read	bigint	硬盘同步读次数。
hdd_asyn_read	bigint	硬盘异步读次数。

3.2.15.28.GLOBAL_STAT_SESSION_CU

GLOBAL_STAT_SESSION_CU 用于查询 PanWeiDB 各个节点，当前运行 session 的 CU 命中情况。session 退出相应的统计数据会清零。PanWeiDB 重启后，统计数据也会清零。

表 1 GLOBAL_STAT_SESSION_CU 字段

名称	类型	描述
mem_hit	integer	内存命中次数。
hdd_sync_read	integer	硬盘同步读次数。
hdd_asyn_read	integer	硬盘异步读次数。

3.2.16.Utility

3.2.16.1.REPLICATION_STAT

REPLICATION_STAT 用于描述日志同步状态信息，如发起端发送日志位置、收端接收日志位置等。

表 1 REPLICATION_STAT 字段

名称	类型	描述
pid	bigint	线程的 PID。
usesysid	oid	用户系统 ID。
usename	name	用户名。
application_name	text	程序名称。
client_addr	inet	客户端地址。
client_hostname	text	客户端名。
client_port	integer	客户端端口。
backend_start	timestamp with time zone	程序启动时间。
state	text	日志复制的状态： 追赶状态。 一致的流状态。
sender_sent_location	text	发送端发送日志位置。
receiver_write_location	text	接收端 write 日志位置。
receiver_flush_location	text	接收端 flush 日志位置。
receiver_replay_location	text	接收端 replay 日志位置。
sync_priority	integer	同步复制的优先级（0 表示异步）。
sync_state	text	同步状态：异步复制、同步复制、潜在同步者。

3.2.16.2.GLOBAL_REPLICATION_STAT

GLOBAL_REPLICATION_STAT 视图用于获得各节点描述日志同步状态信息，如发起端发送日志位置、收端接收日志位置等。

表 1 GLOBAL_REPLICATION_STAT 字段

名称	类型	描述
node_name	name	数据库进程名称。
pid	bigint	线程的 PID。
usesysid	oid	用户系统 ID。
usename	name	用户名。
application_name	text	程序名称。
client_addr	inet	客户端地址。

名称	类型	描述
client_hostname	text	客户端名。
client_port	integer	客户端端口。
backend_start	timestamp with time zone	程序启动时间。
state	text	日志复制的状态：追赶状态、一致的流状态。
sender_sent_location	text	发送端发送日志位置。
receiver_write_location	text	接收端 write 日志位置。
receiver_flush_location	text	接收端 flush 日志位置。
receiver_replay_location	text	接收端 replay 日志位置。
sync_priority	integer	同步复制的优先级（0 表示异步）。
sync_state	text	同步状态：异步复制、同步复制、潜在同步者。

3.2.16.3.REPLICATION_SLOTS

REPLICATION_SLOTS 视图用于查看复制节点的信息。

表 1 REPLICATION_SLOTS 字段

名称	类型	描述
slot_name	text	复制节点的名称。
plugin	text	插件名称。
slot_type	text	复制节点的类型。
datoid	oid	复制节点的数据库 OID。
database	name	复制节点的数据库名称。
active	boolean	复制节点是否为激活状态。
xmin	xid	复制节点事务标识。
catalog_xmin	xid	逻辑复制槽对应的最早解码事务标识。
restart_lsn	text	复制节点的 Xlog 文件信息。
dummy_standby	boolean	复制节点是否为假备。

3.2.16.4.GLOBAL_REPLICATION_SLOTS

GLOBAL_REPLICATION_SLOTS 视图用于查看 PanWeiDB 各节点的复制节点的信息。

表 1 GLOBAL_REPLICATION_SLOTS 字段

名称	类型	描述
node_name	name	数据库进程名称。
slot_name	text	复制节点的名称。
plugin	text	插件名称。
slot_type	text	复制节点的类型。
datoid	oid	复制节点的数据库 OID。
database	name	复制节点的数据库名称。
active	boolean	复制节点是否为激活状态。
x_min	xid	复制节点事务标识。
catalog_xmin	xid	逻辑复制槽对应的最早解码事务标识.
restart_lsn	text	复制节点的 Xlog 文件信息。
dummy_standby	boolean	复制节点是否为假备。

3.2.16.5.BGWRITER_STAT

BGWRITER_STAT 视图显示关于后端写进程活动的统计信息。

表 1 BGWRITER_STAT 字段

名称	类型	描述
checkpoints_timed	bigint	执行的定期检查点数。
checkpoints_req	bigint	执行的需求检查点数。
checkpoint_write_time	double precision	花费在检查点处理部分的时间总量，其中文件被写入到磁盘，以毫秒为单位。
checkpoint_sync_time	double precision	花费在检查点处理部分的时间总量，其中文件被同步到磁盘，以毫秒为单位。
buffers_checkpoint	bigint	检查点写缓冲区数量。
buffers_clean	bigint	后端写进程写缓冲区数量。
maxwritten_clean	bigint	后端写进程停止清理扫描时间数，因为它写了太多缓冲区。
buffers_backend	bigint	通过后端直接写缓冲区数。

名称	类型	描述
buffers_backend_fsync	bigint	后端不得不执行自己的 fsync 调用的时间数（通常后端写进程处理这些即使后端确实自己写）。
buffers_alloc	bigint	分配的缓冲区数量。
stats_reset	timestamp with time zone	这些统计被重置的时间。

3.2.16.6.GLOBAL_BGWRITER_STAT

GLOBAL_BGWRITER_STAT 视图显示各节点关于后端写进程活动的统计信息。

表 1 GLOBAL_BGWRITER_STAT 字段

名称	类型	描述
node_name	name	数据库进程名称。
checkpoints_timed	bigint	执行的定期检查点数。
checkpoints_req	bigint	执行的需求检查点数。
checkpoint_write_time	double precision	花费在检查点处理部分的时间总量，其中文件被写入到磁盘，以毫秒为单位。
checkpoint_sync_time	double precision	花费在检查点处理部分的时间总量，其中文件被同步到磁盘，以毫秒为单位。
buffers_checkpoint	bigint	检查点写缓冲区数量。
buffers_clean	bigint	后端写进程写缓冲区数量。
maxwritten_clean	bigint	后端写进程停止清理扫描时间数，因为它写了太多缓冲区。
buffers_backend	bigint	通过后端直接写缓冲区数。
buffers_backend_fsync	bigint	后端不得不执行自己的 fsync 调用的时间数（通常后端写进程处理这些即使后端确实自己写）。
buffers_alloc	bigint	分配的缓冲区数量。
stats_reset	timestamp with time zone	这些统计被重置的时间。

3.2.16.7.GLOBAL_CKPT_STATUS

GLOBAL_CKPT_STATUS 视图用于显示 PanWeiDB 所有实例的检查点信息和各类日志刷页情况。

表 1 GLOBAL_CKPT_STATUS 字段

名称	类型	描述
node_name	text	数据库进程名称。
ckpt_redo_point	test	当前实例的检查点。
ckpt_clog_flush_num	bigint	从启动到当前时间 clog 刷盘页面数。
ckpt_csnlog_flush_num	bigint	从启动到当前时间 csnlog 刷盘页面数。
ckpt_multixact_flush_num	bigint	从启动到当前时间 multixact 刷盘页面数。
ckpt_predicate_flush_num	bigint	从启动到当前时间 predicate 刷盘页面数。
ckpt_twopcme_flush_num	bigint	从启动到当前时间 twopcme 刷盘页面数。

3.2.16.8.GLOBAL_DOUBLE_WRITE_STATUS

GLOBAL_DOUBLE_WRITE_STATUS 视图显示 PanWeiDB 所有实例的双写文件的情况。它是由每个节点的 local_double_write_stat 视图组成，属性完全一致。

表 1 GLOBAL_DOUBLE_WRITE_STATUS 字段

名称	类型	描述
node_name	text	节点名称。
curr_dwn	bigint	当前双写文件的序列号。
curr_start_page	bigint	当前双写文件恢复起始页面。
file_trunc_num	bigint	当前双写文件复用的次数。
file_reset_num	bigint	当前双写文件写满后发生重置的次数。
total_writes	bigint	当前双写文件总的 I/O 次数。
low_threshold_writes	bigint	低效率写双写文件的 I/O 次数（一次 I/O 刷页数量少于 16 页面）。
high_threshold_writes	bigint	高效率写双写文件的 I/O 次数（一次 I/O 刷页数量多于一批，421 个页面）。
total_pages	bigint	当前刷页到双写文件区的总的页面个数。
low_threshold_pages	bigint	低效率刷页的页面个数。

名称	类型	描述
high_threshold_pages	bigint	高效率刷页的页面个数。
file_id	bigint	当前双写文件的 id 号。

3.2.16.9.GLOBAL_PAGEWRITER_STATUS

GLOBAL_PAGEWRITER_STATUS 视图显示 PanWeiDB 实例的刷页信息和检查点信息。

表 1 GLOBAL_PAGEWRITER_STATUS 字段

名称	类型	描述
node_name	text	数据库进程名称。
pgwr_actual_flush_total_num	bigint	从启动到当前时间总计刷脏页数量。
pgwr_last_flush_num	integer	上一批刷脏页数量。
remain_dirty_page_num	bigint	当前预计还剩余多少脏页。
queue_head_page_rec_lsn	text	当前实例的脏页队列第一个脏页的 recovery_lsn。
queue_rec_lsn	text	当前实例的脏页队列的 recovery_lsn。
current_xlog_insert_lsn	text	当前实例 XLog 写入的位置。
ckpt_redo_point	text	当前实例的检查点。

3.2.16.10.GLOBAL_RECORD_RESET_TIME

GLOBAL_RECORD_RESET_TIME 用于重置（重启、主备倒换、数据库删除）汇聚 PanWeiDB 统计信息时间。

表 1 GLOBAL_RECORD_RESET_TIME 字段

名称	类型	描述
node_name	text	数据库进程名称。
reset_time	timestamp with time zone	重置时间点。

3.2.16.11.GLOBAL_REDO_STATUS

GLOBAL_REDO_STATUS 视图显示 PanWeiDB 实例的日志回放情况。

表 1 GLOBAL_REDO_STATUS 字段

名称	类型	描述

名称	类型	描述
node_name	text	数据库进程名称。
redo_start_ptr	bigint	当前实例日志回放的起始点。
redo_start_time	bigint	当前实例日志回放的起始 UTC 时间。
redo_done_time	bigint	当前实例日志回放的结束 UTC 时间。
curr_time	bigint	当前实例的当前 UTC 时间。
min_recovery_point	bigint	当前实例日志的最小一致性点位置。
read_ptr	bigint	当前实例日志的读取位置。
last_replayed_read_ptr	bigint	当前实例的日志回放位置。
recovery_done_ptr	bigint	当前实例启动完成时的回放位置。
read_xlog_io_counter	bigint	当前实例读取回放日志的 io 次数计数。
read_xlog_io_total_dur	bigint	当前实例读取回放日志的 io 总时延。
read_data_io_counter	bigint	当前实例回放过程中读取数据页面的 io 次数计数。
read_data_io_total_dur	bigint	当前实例回放过程中读取数据页面的 io 总时延。
write_data_io_counter	bigint	当前实例回放过程中写数据页面的 io 次数计数。
write_data_io_total_dur	bigint	当前实例回放过程中写数据页面的 io 总时延。
process_pending_counter	bigint	当前实例回放过程中日志分发线程的同步次数计数。
process_pending_total_dur	bigint	当前实例回放过程中日志分发线程的同步总时延。
apply_counter	bigint	当前实例回放过程中回放线程的同步次数计数。
apply_total_dur	bigint	当前实例回放过程中回放线程的同步总时延。
speed	bigint	当前实例日志回放速率。
local_max_ptr	bigint	当前实例启动成功后本地收到的回放日志的最大值。
primary_flush_ptr	bigint	主机落盘日志的位置。
worker_info	text	当前实例回放线程信息，若没有开并行回放则该值为空。

3.2.16.12.GLOBAL_RECOVERY_STATUS

GLOBAL_RECOVERY_STATUS 视图显示关于主机和备机的日志流控信息。

表 1 GLOBAL_RECOVERY_STATUS 字段

名称	类型	描述
node_name	text	主机进程名称，包含主机和备机。
standby_node_name	text	备机进程名称。
source_ip	text	主机的 IP 地址。
source_port	integer	主机的端口号。
dest_ip	text	备机的 IP 地址。
dest_port	integer	备机的端口号。
current_rto	bigint	备机当前的日志流控时间，单位秒。
target_rto	bigint	备机通过 GUC 参数设置的预期流控时间，单位秒。
current_sleep_time	bigint	为了达到这个预期主机所需要的睡眠时间，单位微妙。

3.2.16.13.CLASS_VITAL_INFO

CLASS_VITAL_INFO 视图用于做 WDR 时校验相同的表或者索引的 oid 是否一致。

表 1 CLASS_VITAL_INFO 字段

名称	类型	描述
relid	oid	表的 oid。
schemaname	name	schema 名称。
relname	name	表名。
relkind	“char”	表示对象类型，取值范围如下： ❖ r：表示普通表。 ❖ t：表示 toast 表。 ❖ i：表示索引。

3.2.16.14.USER_LOGIN

USER_LOGIN 用来记录用户登录和退出次数的相关信息。

表 1 USER_LOGIN 字段

名称	类型	描述
node_name	text	数据库进程名称。
user_name	text	用户名称。
user_id	integer	用户 oid（同 pg_authid 中的 oid 字段）。
login_counter	bigint	登录次数。
logout_counter	bigint	退出次数。

3.2.16.15.SUMMARY_USER_LOGIN

SUMMARY_USER_LOGIN 用来记录数据库主节点上用户登录和退出次数的相关信息。

表 1 SUMMARY_USER_LOGIN 字段

名称	类型	描述
node_name	text	数据库进程名称。
user_name	text	用户名称。
user_id	integer	用户 oid（同 pg_authid 中的 oid 字段）。
login_counter	bigint	登录次数。
logout_counter	bigint	退出次数。

3.2.16.16.GLOBAL_SINGLE_FLUSH_DW_STATUS

GLOBAL_SINGLE_FLUSH_DW_STATUS 视图显示数据库所有实例单页面淘汰双写文件信息。展示内容中，/前是第一个版本双写文件刷页情况，/后是第二个版本双写文件刷页情况。

表 1 GLOBAL_SINGLE_FLUSH_DW_STATUS 字段

名称	类型	描述
node_name	text	实例名称。
bgwr_actual_flush_total_num	bigint	从启动到当前时间 bgwriter 线程总计刷脏页数量。
bgwr_last_flush_num	integer	bgwriter 线程上一批刷脏页数量。
candidate_slots	integer	当前候选 buffer 链中页面个数。
get_buffer_from_list	bigint	buffer 淘汰从候选 buffer 链中获取页面的次数。
get_buffer_clock_sweep	bigint	buffer 淘汰从原淘汰方案中获取页面的次

名称	类型	描述
		数。

3.2.16.17.GLOBAL_CANDIDATE_STATUS

GLOBAL_CANDIDATE_STATUS 视图显示整个数据库所有实例候选 buffer 个数，buffer 淘汰信息。

表 1 GLOBAL_GET_BGWRITER_STATUS 字段

名称	类型	描述
node_name	text	节点名称。
candidate_slots	integer	当前 Normal Buffer Pool 候选 buffer 链中页面个数。
get_buf_from_list	bigint	Normal Buffer Pool，buffer 淘汰从候选 buffer 链中获取页面的次数。
get_buf_clock_sweep	bigint	Normal Buffer Pool，buffer 淘汰从原淘汰方案中获取页面的次数。
seg_candidate_slots	integer	当前 Segment Buffer Pool 候选 buffer 链中页面个数。
seg_get_buf_from_list	bigint	Segment Buffer Pool，buffer 淘汰从候选 buffer 链中获取页面的次数。
seg_get_buf_clock_sweep	bigint	Segment Buffer Pool，buffer 淘汰从原淘汰方案中获取页面的次数。

3.2.17.Object

3.2.17.1.STAT_USER_TABLES

显示当前节点所有命名空间中用户自定义普通表的状态信息。

表 1 STAT_USER_TABLES 字段

名称	类型	描述
relid	oid	表的 OID。
schemaname	name	该表的模式名。
relname	name	表名。
seq_scan	bigint	该表发起的顺序扫描数。

名称	类型	描述
seq_tup_read	bigint	顺序扫描抓取的活跃行数。
idx_scan	bigint	该表发起的索引扫描数。
idx_tup_fetch	bigint	索引扫描抓取的活跃行数。
n_tup_ins	bigint	插入行数。
n_tup_upd	bigint	更新行数。
n_tup_del	bigint	删除行数。
n_tup_hot_upd	bigint	HOT 更新行数（即没有更新所需的单独索引）。
n_live_tup	bigint	估计活跃行数。
n_dead_tup	bigint	估计死行数。
last_vacuum	timestamp with time zone	最后一次该表是手动清理的（不计算 VACUUM FULL）时间。
last_autovacuum	timestamp with time zone	上次被 autovacuum 守护进程清理的时间。
last_analyze	timestamp with time zone	上次手动分析该表的时间。
last_autoanalyze	timestamp with time zone	上次被 autovacuum 守护进程分析的时间。
vacuum_count	bigint	该表被手动清理的次数（不计算 VACUUM FULL）。
autovacuum_count	bigint	该表被 autovacuum 清理的次数。
analyze_count	bigint	该表被手动分析的次数。
autoanalyze_count	bigint	该表被 autovacuum 守护进程分析的次数。

3.2.17.2.SUMMARY_STAT_USER_TABLES

PanWeiDB 内汇聚所有命名空间中用户自定义普通表的状态信息。

表 1 SUMMARY_STAT_USER_TABLES 字段

名称	类型	描述
schemaname	name	此表的模式名。
relname	name	表名。
seq_scan	numeric	此表发起的顺序扫描数。

名称	类型	描述
seq_tup_read	numeric	顺序扫描抓取的活跃行数。
idx_scan	numeric	此表发起的索引扫描数。
idx_tup_fetch	numeric	索引扫描抓取的活跃行数。
n_tup_ins	numeric	插入行数。
n_tup_upd	numeric	更新行数。
n_tup_del	numeric	删除行数。
n_tup_hot_upd	numeric	HOT 更新行数（即没有更新所需的单独索引）。
n_live_tup	numeric	估计活跃行数。
n_dead_tup	numeric	估计死行数。
last_vacuum	timestamp with time zone	最后一次此表是手动清理的（不计算 VACUUM FULL）时间。
last_autovacuum	timestamp with time zone	上次被 autovacuum 守护进程清理的时间。
last_analyze	timestamp with time zone	上次手动分析这个表的时间。
last_autoanalyze	timestamp with time zone	上次被 autovacuum 守护进程分析的时间。
vacuum_count	numeric	这个表被手动清理的次数（不计算 VACUUM FULL）。
autovacuum_count	numeric	这个表被 autovacuum 清理的次数。
analyze_count	numeric	这个表被手动分析的次数。
autoanalyze_count	numeric	这个表被 autovacuum 守护进程分析的次数。

3.2.17.3.GLOBAL_STAT_USER_TABLES

得到各节点所有命名空间中用户自定义普通表的状态信息。

表 1 GLOBAL_STAT_USER_TABLES 字段

名称	类型	描述
node_name	name	数据库进程名称。
relid	oid	表的 OID。
schemaname	name	此表的模式名。

名称	类型	描述
relname	name	表名。
seq_scan	bigint	此表发起的顺序扫描数。
seq_tup_read	bigint	顺序扫描抓取的活跃行数。
idx_scan	bigint	此表发起的索引扫描数。
idx_tup_fetch	bigint	索引扫描抓取的活跃行数。
n_tup_ins	bigint	插入行数。
n_tup_upd	bigint	更新行数。
n_tup_del	bigint	删除行数。
n_tup_hot_upd	bigint	HOT 更新行数（即没有更新所需的单独索引）。
n_live_tup	bigint	估计活跃行数。
n_dead_tup	bigint	估计死行数。
last_vacuum	timestamp with time zone	最后一次此表是手动清理的（不计算 VACUUM FULL）时间。
last_autovacuum	timestamp with time zone	上次被 autovacuum 守护进程清理的时间。
last_analyze	timestamp with time zone	上次手动分析这个表的时间。
last_autoanalyze	timestamp with time zone	上次被 autovacuum 守护进程分析的时间。
vacuum_count	bigint	这个表被手动清理的次数（不计算 VACUUM FULL）。
autovacuum_count	bigint	这个表被 autovacuum 清理的次数。
analyze_count	bigint	这个表被手动分析的次数。
autoanalyze_count	bigint	这个表被 autovacuum 守护进程分析的次数。

3.2.17.4.STAT_USER_INDEXES

显示数据库中用户自定义普通表的索引状态信息。

表 1 STAT_USER_INDEXES 字段

名称	类型	描述
relid	oid	此索引的表的 OID。

名称	类型	描述
indexrelid	oid	索引的 OID。
schemaname	name	索引的模式名。
relname	name	索引的表名。
indexrelname	name	索引名。
idx_scan	bigint	索引上开始的索引扫描数。
idx_tup_read	bigint	通过索引上扫描返回的索引项数。
idx_tup_fetch	bigint	通过使用索引的简单索引扫描抓取的活表行数。

3.2.17.5.SUMMARY_STAT_USER_INDEXES

PanWeiDB 内汇聚所有数据库中用户自定义普通表的索引状态信息。

表 1 SUMMARY_STAT_USER_INDEXES 字段

名称	类型	描述
schemaname	name	索引中模式名。
relname	name	索引的表名。
indexrelname	name	索引名。
idx_scan	numeric	索引上开始的索引扫描数。
idx_tup_read	numeric	通过索引上扫描返回的索引项数。
idx_tup_fetch	numeric	通过使用索引的简单索引扫描抓取的活表行数。

3.2.17.6.GLOBAL_STAT_USER_INDEXES

得到各节点数据库中用户自定义普通表的索引状态信息。

表 1 GLOBAL_STAT_USER_INDEXES 字段

名称	类型	描述
node_name	name	数据库进程名称。
relid	oid	这个索引的表的 OID。
indexrelid	oid	索引的 OID。
schemaname	name	索引中模式名。
relname	name	索引的表名。
indexrelname	name	索引名。
idx_scan	bigint	索引上开始的索引扫描数。
idx_tup_read	bigint	通过索引上扫描返回的索引项数。

名称	类型	描述
idx_tup_fetch	bigint	通过使用索引的简单索引扫描抓取的活表行数。

3.2.17.7.STAT_SYS_TABLES

显示单节点内 pg_catalog、information_schema 以及 pg_toast 模式下的所有系统表的统计信息。

表 1 STAT_SYS_TABLES 字段

名称	类型	描述
relid	oid	表的 OID。
schemaname	name	该表的模式名。
relname	name	表名。
seq_scan	bigint	该表发起的顺序扫描数。
seq_tup_read	bigint	顺序扫描抓取的活跃行数。
idx_scan	bigint	该表发起的索引扫描数。
idx_tup_fetch	bigint	索引扫描抓取的活跃行数。
n_tup_ins	bigint	插入行数。
n_tup_upd	bigint	更新行数。
n_tup_del	bigint	删除行数。
n_tup_hot_upd	bigint	HOT 更新行数（比如没有更新所需的单独索引）。
n_live_tup	bigint	估计活跃行数。
n_dead_tup	bigint	估计死行数。
last_vacuum	timestamp with time zone	最后一次该表是手动清理的（不计算 VACUUM FULL）时间。
last_autovacuum	timestamp with time zone	上次被 autovacuum 守护进程清理的时间。
last_analyze	timestamp with time zone	上次手动分析该表的时间。
last_autoanalyze	timestamp with time zone	上次被 autovacuum 守护进程分析的时间。
vacuum_count	bigint	这个表被手动清理的次数（不计算 VACUUM FULL）。
autovacuum_count	bigint	该表被 autovacuum 清理的次数。

名称	类型	描述
analyze_count	bigint	该表被手动分析的次数。
autoanalyze_count	bigint	该表被 autovacuum 守护进程分析的次数。

3.2.17.8.SUMMARY_STAT_SYS_TABLES

PanWeiDB 内汇聚 pg_catalog、information_schema 以及 pg_toast 模式下的所有系统表的统计信息。

表 1 SUMMARY_STAT_SYS_TABLES 字段

名称	类型	描述
schemaname	name	此表的模式名。
relname	name	表名。
seq_scan	numeric	此表发起的顺序扫描数。
seq_tup_read	numeric	顺序扫描抓取的活跃行数。
idx_scan	numeric	此表发起的索引扫描数。
idx_tup_fetch	numeric	索引扫描抓取的活跃行数。
n_tup_ins	numeric	插入行数。
n_tup_upd	numeric	更新行数。
n_tup_del	numeric	删除行数。
n_tup_hot_upd	numeric	HOT 更新行数（比如没有更新所需的单独索引）。
n_live_tup	numeric	估计活跃行数。
n_dead_tup	numeric	估计死行数。
last_vacuum	timestamp with time zone	最后一次此表是手动清理的（不计算 VACUUM FULL）时间。
last_autovacuum	timestamp with time zone	上次被 autovacuum 守护进程清理的时间。
last_analyze	timestamp with time zone	上次手动分析这个表的时间。
last_autoanalyze	timestamp with time zone	上次被 autovacuum 守护进程分析的时间。
vacuum_count	numeric	这个表被手动清理的次数（不计算 VACUUM FULL）。

名称	类型	描述
autovacuum_count	numeric	这个表被 autovacuum 清理的次数。
analyze_count	numeric	这个表被手动分析的次数。
autoanalyze_count	numeric	这个表被 autovacuum 守护进程分析的次数。

3.2.17.9.GLOBAL_STAT_SYS_TABLES

得到各节点 pg_catalog、information_schema 以及 pg_toast 模式下的所有系统表的统计信息。

表 1 GLOBAL_STAT_SYS_TABLES 字段

名称	类型	描述
node_name	name	节点名称。
relid	oid	表的 OID。
schemaname	name	此表的模式名。
relname	name	表名。
seq_scan	bigint	此表发起的顺序扫描数。
seq_tup_read	bigint	顺序扫描抓取的活跃行数。
idx_scan	bigint	此表发起的索引扫描数。
idx_tup_fetch	bigint	索引扫描抓取的活跃行数。
n_tup_ins	bigint	插入行数。
n_tup_upd	bigint	更新行数。
n_tup_del	bigint	删除行数。
n_tup_hot_upd	bigint	HOT 更新行数（比如没有更新所需的单独索引）。
n_live_tup	bigint	估计活跃行数。
n_dead_tup	bigint	估计死行数。
last_vacuum	timestamp with time zone	最后一次此表是手动清理的（不计算 VACUUM FULL）时间。
last_autovacuum	timestamp with time zone	上次被 autovacuum 守护进程清理的时间。
last_analyze	timestamp with time zone	上次手动分析这个表的时间。
last_autoanalyze	timestamp with	上次被 autovacuum 守护进程分析的时

名称	类型	描述
	time zone	间。
vacuum_count	bigint	这个表被手动清理的次数（不计算 VACUUM FULL）。
autovacuum_count	bigint	这个表被 autovacuum 清理的次数。
analyze_count	bigint	这个表被手动分析的次数。
autoanalyze_count	bigint	这个表被 autovacuum 守护进程分析的次数。

3.2.17.10.STAT_SYS_INDEXES

显示 pg_catalog、information_schema 以及 pg_toast 模式中所有系统表的索引状态信息。

表 1 STAT_SYS_INDEXES 字段

名称	类型	描述
relid	oid	此索引的表的 OID。
indexrelid	oid	索引的 OID。
schemaname	name	索引的模式名。
relname	name	索引的表名。
indexrelname	name	索引名。
idx_scan	bigint	索引上开始的索引扫描数。
idx_tup_read	bigint	通过索引上扫描返回的索引项数。
idx_tup_fetch	bigint	通过使用索引的简单索引扫描抓取的活表行数。

3.2.17.11.SUMMARY_STAT_SYS_INDEXES

PanWeiDB 内汇聚 pg_catalog、information_schema 以及 pg_toast 模式中所有系统表的索引状态信息。

表 1 SUMMARY_STAT_SYS_INDEXES 字段

名称	类型	描述
schemaname	name	索引中模式名。
relname	name	索引的表名。
indexrelname	name	索引名。
idx_scan	numeric	索引上开始的索引扫描数。

名称	类型	描述
idx_tup_read	numeric	通过索引上扫描返回的索引项数。
idx_tup_fetch	numeric	通过使用索引的简单索引扫描抓取的活表行数。

3.2.17.12.GLOBAL_STAT_SYS_INDEXES

得到各节点 pg_catalog、information_schema 以及 pg_toast 模式中所有系统表的索引状态信息。

表 1 GLOBAL_STAT_SYS_INDEXES 字段

名称	类型	描述
node_name	name	数据库进程名称。
relid	oid	这个索引的表的 OID。
indexrelid	oid	索引的 OID。
schemaname	name	索引中模式名。
relname	name	索引的表名。
indexrelname	name	索引名。
idx_scan	bigint	索引上开始的索引扫描数。
idx_tup_read	bigint	通过索引上扫描返回的索引项数。
idx_tup_fetch	bigint	通过使用索引的简单索引扫描抓取的活表行数。

3.2.17.13.STAT_ALL_TABLES

本节点内数据库中每个表（包括 TOAST 表）的一行的统计信息。

表 1 STAT_ALL_TABLES 字段

名称	类型	描述
relid	oid	表的 OID。
schemaname	name	该表的模式名。
relname	name	表名。
seq_scan	bigint	该表发起的顺序扫描数。
seq_tup_read	bigint	顺序扫描抓取的活跃行数。
idx_scan	bigint	该表发起的索引扫描数。
idx_tup_fetch	bigint	索引扫描抓取的活跃行数。
n_tup_ins	bigint	插入行数。
n_tup_upd	bigint	更新行数。

名称	类型	描述
n_tup_del	bigint	删除行数。
n_tup_hot_upd	bigint	HOT 更新行数（比如没有更新所需的单独索引）。
n_live_tup	bigint	估计活跃行数。
n_dead_tup	bigint	估计死行数。
last_vacuum	timestamp with time zone	最后一次该表是手动清理的（不计算 VACUUM FULL）的时间。
last_autovacuum	timestamp with time zone	上次被 autovacuum 守护进程清理的时间。
last_analyze	timestamp with time zone	上次手动分析该表的时间。
last_autoanalyze	timestamp with time zone	上次被 autovacuum 守护进程分析时间。
vacuum_count	bigint	该表被手动清理的次数（不计算 VACUUM FULL）。
autovacuum_count	bigint	该表被 autovacuum 清理的次数。
analyze_count	bigint	该表被手动分析的次数。
autoanalyze_count	bigint	该表被 autovacuum 守护进程分析的次数。

3.2.17.14.SUMMARY_STAT_ALL_TABLES

PanWeiDB 内汇聚数据库中每个表的一行（包括 TOAST 表）的统计信息。

表 1 SUMMARY_STAT_ALL_TABLES 字段

名称	类型	描述
schemaname	name	此表的模式名。
relname	name	表名。
seq_scan	numeric	此表发起的顺序扫描数。
seq_tup_read	numeric	顺序扫描抓取的活跃行数。
idx_scan	numeric	此表发起的索引扫描数。
idx_tup_fetch	numeric	索引扫描抓取的活跃行数。
n_tup_ins	numeric	插入行数。
n_tup_upd	numeric	更新行数。

名称	类型	描述
n_tup_del	numeric	删除行数。
n_tup_hot_upd	numeric	HOT 更新行数（比如没有更新所需的单独索引）。
n_live_tup	numeric	估计活跃行数。
n_dead_tup	numeric	估计死行数。
last_vacuum	timestamp with time zone	最后一次此表是手动清理的（不计算 VACUUM FULL）的时间。
last_autovacuum	timestamp with time zone	上次被 autovacuum 守护进程清理的时间。
last_analyze	timestamp with time zone	上次手动分析这个表的时间。
last_autoanalyze	timestamp with time zone	上次被 autovacuum 守护进程分析时间。
vacuum_count	numeric	这个表被手动清理的次数（不计算 VACUUM FULL）。
autovacuum_count	numeric	这个表被 autovacuum 清理的次数。
analyze_count	numeric	这个表被手动分析的次数。
autoanalyze_count	numeric	这个表被 autovacuum 守护进程分析的次数。

3.2.17.15.GLOBAL_STAT_ALL_TABLES

得到各节点数据中每个表的一行（包括 TOAST 表）的统计信息。

表 1 GLOBAL_STAT_ALL_TABLES 字段

名称	类型	描述
node_name	name	数据库进程名称。
relid	oid	表的 OID。
schemaname	name	此表的模式名。
relname	name	表名。
seq_scan	bigint	此表发起的顺序扫描数。
seq_tup_read	bigint	顺序扫描抓取的活跃行数。
idx_scan	bigint	此表发起的索引扫描数。
idx_tup_fetch	bigint	索引扫描抓取的活跃行数。

名称	类型	描述
n_tup_ins	bigint	插入行数。
n_tup_upd	bigint	更新行数。
n_tup_del	bigint	删除行数。
n_tup_hot_upd	bigint	HOT 更新行数（比如没有更新所需的单独索引）。
n_live_tup	bigint	估计活跃行数。
n_dead_tup	bigint	估计死行数。
last_vacuum	timestamp with time zone	最后一次此表是手动清理的（不计算 VACUUM FULL）的时间。
last_autovacuum	timestamp with time zone	上次被 autovacuum 守护进程清理的时间。
last_analyze	timestamp with time zone	上次手动分析这个表的时间。
last_autoanalyze	timestamp with time zone	上次被 autovacuum 守护进程分析时间。
vacuum_count	bigint	这个表被手动清理的次数（不计算 VACUUM FULL）。
autovacuum_count	bigint	这个表被 autovacuum 清理的次数。
analyze_count	bigint	这个表被手动分析的次数。
autoanalyze_count	bigint	这个表被 autovacuum 守护进程分析的次数。

3.2.17.16.STAT_ALL_INDEXES

将包含本节点数据库中的每个索引行，显示访问特定索引的统计。

表 1 STAT_ALL_INDEXES 字段

名称	类型	描述
relid	oid	这个索引的表的 OID。
indexrelid	oid	索引的 OID。
schemaname	name	索引中模式名。
relname	name	索引的表名。
indexrelname	name	索引名。
idx_scan	bigint	索引上开始的索引扫描数。

名称	类型	描述
idx_tup_read	bigint	通过索引上扫描返回的索引项数。
idx_tup_fetch	bigint	通过使用索引的简单索引扫描抓取的活表行数。

3.2.17.17.SUMMARY_STAT_ALL_INDEXES

将包含 PanWeiDB 内汇聚数据库中的每个索引行，显示访问特定索引的统计。

表 1 SUMMARY_STAT_ALL_INDEXES 字段

名称	类型	描述
schemaname	name	索引中模式名。
relname	name	索引的表名。
indexrelname	name	索引名。
idx_scan	numeric	索引上开始的索引扫描数。
idx_tup_read	numeric	通过索引上扫描返回的索引项数。
idx_tup_fetch	numeric	通过使用索引的简单索引扫描抓取的活表行数。

3.2.17.18.GLOBAL_STAT_ALL_INDEXES

将包含各节点数据库中的每个索引行，显示访问特定索引的统计。

表 1 GLOBAL_STAT_ALL_INDEXES 字段

名称	类型	描述
node_name	name	数据库进程名称。
relid	oid	这个索引的表的 OID。
indexrelid	oid	索引的 OID。
schemaname	name	索引中模式名。
relname	name	索引的表名。
indexrelname	name	索引名。
idx_scan	bigint	索引上开始的索引扫描数。
idx_tup_read	bigint	通过索引上扫描返回的索引项数。
idx_tup_fetch	bigint	通过使用索引的简单索引扫描抓取的活表行数。

3.2.17.19.STAT_DATABASE

视图将包含本节点中每个数据库的统计信息。

表 1 STAT_DATABASE 字段

名称	类型	描述
datid	oid	数据库的 OID。
datname	name	此数据库的名称。
numbackends	integer	当前连接到该数据库的后端数。这是在返回一个反映目前状态值的视图中唯一的列；自上次重置所有其他列返回累积值。
xact_commit	bigint	此数据库中已经提交的事务数。
xact_rollback	bigint	此数据库中已经回滚的事务数。
blks_read	bigint	在这个数据库中读取的磁盘块的数量。
blks_hit	bigint	高速缓存中已经发现的磁盘块的次数，这样读取是不必要的（这只包括 PostgreSQL 缓冲区高速缓存，没有操作系统的文件系统缓存）。
tup_returned	bigint	通过数据库查询返回的行数。
tup_fetched	bigint	通过数据库查询抓取的行数。
tup_inserted	bigint	通过数据库查询插入的行数。
tup_updated	bigint	通过数据库查询更新的行数。
tup_deleted	bigint	通过数据库查询删除的行数。
conflicts	bigint	由于数据库恢复冲突取消的查询数量（只在备用服务器发生的冲突）。请参见 STAT_DATABASE_CONFLICTS 章节获取更多信息。
temp_files	bigint	通过数据库查询创建的临时文件数量。计算所有临时文件，不论为什么创建临时文件（比如排序或者哈希），而且不管 log_temp_files 设置。
temp_bytes	bigint	通过数据库查询写入临时文件的数据总量。计算所有临时文件，不论为什么创建临时文件，而且不管 log_temp_files 设置。
deadlocks	bigint	在该数据库中检索的死锁数。
blk_read_time	double precision	通过数据库后端读取数据文件块花费的时间，以毫秒计算。
blk_write_time	double precision	通过数据库后端写入数据文件块花费的时间，以毫秒计算。
stats_reset	timestamp with time	重置当前状态统计的时间。

名称	类型	描述
	zone	

3.2.17.20.SUMMARY_STAT_DATABASE

视图将包含数据库内汇聚的每个数据库的每一行，显示数据库统计。

表 1 SUMMARY_STAT_DATABASE 字段

名称	类型	描述
datname	name	这个数据库的名称。
numbackends	bigint	当前连接到该数据库的后端数。这是在返回一个反映目前状态值的视图中唯一的列；自上次重置所有其他列返回累积值。
xact_commit	numeric	此数据库中已经提交的事务数。
xact_rollback	numeric	此数据库中已经回滚的事务数。
blks_read	numeric	在这个数据库中读取的磁盘块的数量。
blks_hit	numeric	高速缓存中已经发现的磁盘块的次数，这样读取是不必要的（这只包括 openGauss 缓冲区高速缓存，没有操作系统的文件系统缓存）。
tup_returned	numeric	通过数据库查询返回的行数。
tup_fetched	numeric	通过数据库查询抓取的行数。
tup_inserted	bigint	通过数据库查询插入的行数。
tup_updated	bigint	通过数据库查询更新的行数。
tup_deleted	bigint	通过数据库查询删除的行数。
conflicts	bigint	由于数据库恢复冲突取消的查询数量（只在备用服务器发生的冲突）。请参见 STAT_DATABASE_CONFLICTS 章节获取更多信息。
temp_files	numeric	通过数据库查询创建的临时文件数量。计算所有临时文件，不论为什么创建临时文件（比如排序或者哈希），而且不管 log_temp_files 设置。
temp_bytes	numeric	通过数据库查询写入临时文件的数据总量。计算所有临时文件，不论为什么创建临时文件，而且不管 log_temp_files 设置。
deadlocks	bigint	在该数据库中检索的死锁数。
blk_read_time	double	通过数据库后端读取数据文件块花费的时间，以毫

名称	类型	描述
	precision	秒计算。
blk_write_time	double precision	通过数据库后端写入数据文件块花费的时间，以毫秒计算。
stats_reset	timestamp with time zone	重置当前状态统计的时间。

3.2.17.21.GLOBAL_STAT_DATABASE

视图将包含 PanWeiDB 中各节点的每个数据库的每一行，显示数据库统计。

表 1 GLOBAL_STAT_DATABASE 字段

名称	类型	描述
node_name	name	数据库进程名称。
datid	oid	数据库的 OID。
datname	name	这个数据库的名称。
numbackends	integer	当前连接到该数据库的后端数。这是在返回一个反映目前状态值的视图中唯一的列；自上次重置所有其他列返回累积值。
xact_commit	bigint	此数据库中已经提交的事务数。
xact_rollback	bigint	此数据库中已经回滚的事务数。
blks_read	bigint	在这个数据库中读取的磁盘块的数量。
blks_hit	bigint	高速缓存中已经发现的磁盘块的次数，这样读取是不必要的（这只包括数据库内核缓冲区高速缓存，没有操作系统的文件系统缓存）。
tup_returned	bigint	通过数据库查询返回的行数。
tup_fetched	bigint	通过数据库查询抓取的行数。
tup_inserted	bigint	通过数据库查询插入的行数。
tup_updated	bigint	通过数据库查询更新的行数。
tup_deleted	bigint	通过数据库查询删除的行数。
conflicts	bigint	由于数据库恢复冲突取消的查询数量（只在备用服务器发生的冲突）。请参见 STAT_DATABASE_CONFLICTS 获取更多信息。
temp_files	bigint	通过数据库查询创建的临时文件数量。计算所有临

名称	类型	描述
		时文件，不论为什么创建临时文件（比如排序或者哈希），而且不管 log_temp_files 设置。
temp_bytes	bigint	通过数据库查询写入临时文件的数据总量。计算所有临时文件，不论为什么创建临时文件，而且不管 log_temp_files 设置。
deadlocks	bigint	在该数据库中检索的死锁数。
blk_read_time	double precision	通过数据库后端读取数据文件块花费的时间，以毫秒计算。
blk_write_time	double precision	通过数据库后端写入数据文件块花费的时间，以毫秒计算。
stats_reset	timestamp with time zone	重置当前状态统计的时间。

3.2.17.22.STAT_DATABASE_CONFLICTS

显示当前节点数据库冲突状态的统计信息。

表 1 STAT_DATABASE_CONFLICTS 字段

名称	类型	描述
datid	oid	数据库标识。
datname	name	数据库名称。
confl_tablespace	bigint	冲突的表空间的数目。
confl_lock	bigint	冲突的锁数目。
confl_snapshot	bigint	冲突的快照数目。
confl_bufferpin	bigint	冲突的缓冲区数目。
confl_deadlock	bigint	冲突的死锁数目。

3.2.17.23.SUMMARY_STAT_DATABASE_CONFLICTS

显示 PanWeiDB 内汇聚的数据库冲突状态的统计信息。

表 1 SUMMARY_STAT_DATABASE_CONFLICTS 字段

名称	类型	描述
datname	name	数据库名称。

名称	类型	描述
confl_tablespace	bigint	冲突的表空间的数目。
confl_lock	bigint	冲突的锁数目。
confl_snapshot	bigint	冲突的快照数目。
confl_bufferpin	bigint	冲突的缓冲区数目。
confl_deadlock	bigint	冲突的死锁数目。

3.2.17.24.GLOBAL_STAT_DATABASE_CONFLICTS

显示每个节点的数据库冲突状态的统计信息。

表 1 GLOBAL_STAT_DATABASE_CONFLICTS 字段

名称	类型	描述
node_name	name	数据库进程名称。
datid	oid	数据库标识。
datname	name	数据库名称。
confl_tablespace	bigint	冲突的表空间的数目。
confl_lock	bigint	冲突的锁数目。
confl_snapshot	bigint	冲突的快照数目。
confl_bufferpin	bigint	冲突的缓冲区数目。
confl_deadlock	bigint	冲突的死锁数目。

3.2.17.25.STAT_XACT_ALL_TABLES

显示命名空间中所有普通表和 toast 表的事务状态信息。

表 1 STAT_XACT_ALL_TABLES 字段

名称	类型	描述
relid	oid	表的 OID。
schemaname	name	该表的模式名。
relname	name	表名。
seq_scan	bigint	该表发起的顺序扫描数。
seq_tup_read	bigint	顺序扫描抓取的活跃行数。
idx_scan	bigint	该表发起的索引扫描数。
idx_tup_fetch	bigint	索引扫描抓取的活跃行数。
n_tup_ins	bigint	插入行数。

名称	类型	描述
n_tup_upd	bigint	更新行数。
n_tup_del	bigint	删除行数。
n_tup_hot_upd	bigint	HOT 更新行数（比如没有更新所需的单独索引）。

3.2.17.26.SUMMARY_STAT_XACT_ALL_TABLES

显示 PanWeiDB 内汇聚的命名空间中所有普通表和 toast 表的事务状态信息。

表 1 SUMMARY_STAT_XACT_ALL_TABLES 字段

名称	类型	描述
schemaname	name	此表的模式名。
relname	name	表名。
seq_scan	numeric	此表发起的顺序扫描数。
seq_tup_read	numeric	顺序扫描抓取的活跃行数。
idx_scan	numeric	此表发起的索引扫描数。
idx_tup_fetch	numeric	索引扫描抓取的活跃行数。
n_tup_ins	numeric	插入行数。
n_tup_upd	numeric	更新行数。
n_tup_del	numeric	删除行数。
n_tup_hot_upd	numeric	HOT 更新行数（比如没有更新所需的单独索引）。

3.2.17.27.GLOBAL_STAT_XACT_ALL_TABLES

显示各节点的命名空间中所有普通表和 toast 表的事务状态信息。

表 1 GLOBAL_STAT_XACT_ALL_TABLES 字段

名称	类型	描述
node_name	name	数据库进程名称。
relid	oid	表的 OID。
schemaname	name	此表的模式名。
relname	name	表名。
seq_scan	bigint	此表发起的顺序扫描数。
seq_tup_read	bigint	顺序扫描抓取的活跃行数。
idx_scan	bigint	此表发起的索引扫描数。
idx_tup_fetch	bigint	索引扫描抓取的活跃行数。

名称	类型	描述
n_tup_ins	bigint	插入行数。
n_tup_upd	bigint	更新行数。
n_tup_del	bigint	删除行数。
n_tup_hot_upd	bigint	HOT 更新行数（比如没有更新所需的单独索引）。

3.2.17.28.STAT_XACT_SYS_TABLES

显示当前节点命名空间中系统表的事务状态信息。

表 1 STAT_XACT_SYS_TABLES 字段

名称	类型	描述
relid	oid	表的 OID。
schemaname	name	该表的模式名。
relname	name	表名。
seq_scan	bigint	该表发起的顺序扫描数。
seq_tup_read	bigint	顺序扫描抓取的活跃行数。
idx_scan	bigint	该表发起的索引扫描数。
idx_tup_fetch	bigint	索引扫描抓取的活跃行数。
n_tup_ins	bigint	插入行数。
n_tup_upd	bigint	更新行数。
n_tup_del	bigint	删除行数。
n_tup_hot_upd	bigint	HOT 更新行数（比如没有更新所需的单独索引）。

3.2.17.29.SUMMARY_STAT_XACT_SYS_TABLES

显示 PanWeiDB 内汇聚的命名空间中系统表的事务状态信息。

表 1 SUMMARY_STAT_XACT_SYS_TABLES 字段

名称	类型	描述
schemaname	name	此表的模式名。
relname	name	表名。
seq_scan	numeric	此表发起的顺序扫描数。
seq_tup_read	numeric	顺序扫描抓取的活跃行数。
idx_scan	numeric	此表发起的索引扫描数。
idx_tup_fetch	numeric	索引扫描抓取的活跃行数。

名称	类型	描述
n_tup_ins	numeric	插入行数。
n_tup_upd	numeric	更新行数。
n_tup_del	numeric	删除行数。
n_tup_hot_upd	numeric	HOT 更新行数（比如没有更新所需的单独索引）。

3.2.17.30.GLOBAL_STAT_XACT_SYS_TABLES

显示各节点命名空间中系统表的事务状态信息。

表 1 GLOBAL_STAT_XACT_SYS_TABLES 字段

名称	类型	描述
node_name	name	节点名称。
relid	oid	表的 OID。
schemaname	name	此表的模式名。
relname	name	表名。
seq_scan	bigint	此表发起的顺序扫描数。
seq_tup_read	bigint	顺序扫描抓取的活跃行数。
idx_scan	bigint	此表发起的索引扫描数。
idx_tup_fetch	bigint	索引扫描抓取的活跃行数。
n_tup_ins	bigint	插入行数。
n_tup_upd	bigint	更新行数。
n_tup_del	bigint	删除行数。
n_tup_hot_upd	bigint	HOT 更新行数（比如没有更新所需的单独索引）。

3.2.17.31.STAT_XACT_USER_TABLES

显示当前节点命名空间中用户表的事务状态信息。

表 1 STAT_XACT_USER_TABLES 字段

名称	类型	描述
relid	oid	表的 OID。
schemaname	name	该表的模式名。
relname	name	表名。
seq_scan	bigint	该表发起的顺序扫描数。
seq_tup_read	bigint	顺序扫描抓取的活跃行数。

名称	类型	描述
idx_scan	bigint	该表发起的索引扫描数。
idx_tup_fetch	bigint	索引扫描抓取的活跃行数。
n_tup_ins	bigint	插入行数。
n_tup_upd	bigint	更新行数。
n_tup_del	bigint	删除行数。
n_tup_hot_upd	bigint	HOT 更新行数（比如没有更新所需的单独索引）。

3.2.17.32.SUMMARY_STAT_XACT_USER_TABLES

显示数据库内汇聚的命名空间中用户表的事务状态信息。

表 1 SUMMARY_STAT_XACT_USER_TABLES 字段

名称	类型	描述
schemaname	name	此表的模式名。
relname	name	表名。
seq_scan	numeric	此表发起的顺序扫描数。
seq_tup_read	numeric	顺序扫描抓取的活跃行数。
idx_scan	numeric	此表发起的索引扫描数。
idx_tup_fetch	numeric	索引扫描抓取的活跃行数。
n_tup_ins	numeric	插入行数。
n_tup_upd	numeric	更新行数。
n_tup_del	numeric	删除行数。
n_tup_hot_upd	numeric	HOT 更新行数（比如没有更新所需的单独索引）。

3.2.17.33.GLOBAL_STAT_XACT_USER_TABLES

显示各节点命名空间中用户表的事务状态信息。

表 1 GLOBAL_STAT_XACT_USER_TABLES 字段

名称	类型	描述
node_name	name	数据库进程名称。
relid	oid	表的 OID。
schemaname	name	此表的模式名。
relname	name	表名。
seq_scan	bigint	此表发起的顺序扫描数。

名称	类型	描述
seq_tup_read	bigint	顺序扫描抓取的活跃行数。
idx_scan	bigint	此表发起的索引扫描数。
idx_tup_fetch	bigint	索引扫描抓取的活跃行数。
n_tup_ins	bigint	插入行数。
n_tup_upd	bigint	更新行数。
n_tup_del	bigint	删除行数。
n_tup_hot_upd	bigint	HOT 更新行数（比如没有更新所需的单独索引）。

3.2.17.34.STAT_XACT_USER_FUNCTIONS

视图包含当前节点本事务内函数执行的统计信息。

表 1 STAT_XACT_USER_FUNCTIONS 字段

名称	类型	描述
funcid	oid	函数标识。
schemaname	name	模式的名称。
funcname	name	函数名称。
calls	bigint	函数被调用的次数。
total_time	double precision	函数的总执行时长。
self_time	double precision	当前线程调用函数的总的时长。

3.2.17.35.SUMMARY_STAT_XACT_USER_FUNCTIONS

视图包含 PanWeiDB 内汇聚的本事务内函数执行的统计信息。

表 1 SUMMARY_STAT_XACT_USER_FUNCTIONS 字段

名称	类型	描述
schemaname	name	模式的名称。
funcname	name	函数名称。
calls	numeric	函数被调用的次数。
total_time	double precision	函数的总执行时长。
self_time	double precision	当前线程调用函数的总的时长。

3.2.17.36.GLOBAL_STAT_XACT_USER_FUNCTIONS

视图包含各节点本事务内函数执行的统计信息。

表 1 GLOBAL_STAT_XACT_USER_FUNCTIONS 字段

名称	类型	描述
node_name	name	节点名称。
funcid	oid	函数标识。
schemaname	name	模式的名称。
funcname	name	函数名称。
calls	bigint	函数被调用的次数。
total_time	double precision	此函数及其调用的所有其他函数所花费的总时间。
self_time	double precision	在此函数本身中花费的总时间（不包括它调用的其他函数）。

3.2.17.37.STAT_BAD_BLOCK

获得当前节点表、索引等文件的读取失败信息。

表 1 STAT_BAD_BLOCK 字段

名称	类型	描述
nodename	text	数据库进程名称。
databaseid	integer	database 的 oid。
tablespaceid	integer	tablespace 的 oid。
relfilenode	integer	relation 的 file node。
bucketid	smallint	一致性 hash bucket ID。
forknum	integer	fork 编号。
error_count	integer	error 的数量。
first_time	timestamp with time zone	坏块第一次出现的时间。
last_time	timestamp with time zone	坏块最后出现的时间。

3.2.17.38.SUMMARY_STAT_BAD_BLOCK

获得 PanWeiDB 内汇聚的表、索引等文件的读取失败信息。

表 1 SUMMARY_STAT_BAD_BLOCK 字段

名称	类型	描述
databaseid	integer	database 的 oid。
tablespaceid	integer	tablespace 的 oid。

名称	类型	描述
relfilenode	integer	relation 的 file node。
forknum	bigint	fork 编号。
error_count	bigint	error 的数量。
first_time	timestamp with time zone	坏块第一次出现的时间。
last_time	timestamp with time zone	坏块最后出现的时间。

3.2.17.39.GLOBAL_STAT_BAD_BLOCK

获得各节点的表、索引等文件的读取失败信息。

表 1 GLOBAL_STAT_BAD_BLOCK 字段

名称	类型	描述
node_name	text	数据库进程名称。
databaseid	integer	database 的 oid。
tablespaceid	integer	tablespace 的 oid。
relfilenode	integer	relation 的 file node。
forknum	integer	fork 编号。
error_count	integer	error 的数量。
first_time	timestamp with time zone	坏块第一次出现的时间。
last_time	timestamp with time zone	坏块最后出现的时间。

3.2.17.40.STAT_USER_FUNCTIONS

STAT_USER_FUNCTIONS 视图显示命名空间中用户自定义函数（函数语言为非内部语言）的状态信息。

表 1 STAT_USER_FUNCTIONS 字段

名称	类型	描述
funcid	oid	函数标识。
schemaname	name	schema 的名称。
funcname	name	用户自定义函数的名称。
calls	bigint	函数被调用的次数。
total_time	double precision	调用此函数花费的总时间，包含调用其他函数的时间（单位：毫秒）。
self_time	double	调用此函数自己花费的时间，不包含调用其他函数的

名称	类型	描述
	precision	时间（单位：毫秒）。

3.2.17.41.SUMMARY_STAT_USER_FUNCTIONS

SUMMARY_STAT_USER_FUNCTIONS 用来统计所数据库节点用户自定义函数的相关统计信息。

表 1 SUMMARY_STAT_USER_FUNCTIONS 字段

名称	类型	描述
schemaname	name	schema 的名称。
funcname	name	用户 function 的名称。
calls	numeric	总调用次数。
total_time	double precision	调用此 function 的总时间花费，包含调用其他 function 的时间（单位：毫秒）。
self_time	double precision	调用此 function 自己时间的花费，不包含调用其他 function 的时间（单位：毫秒）。

3.2.17.42.GLOBAL_STAT_USER_FUNCTIONS

提供 PanWeiDB 中各个节点的用户所创建的函数的状态的统计信息。

表 1 GLOBAL_STAT_USER_FUNCTIONS 字段

名称	类型	描述
node_name	name	数据库进程名称。
funcid	oid	函数的 id。
schemaname	name	此函数所在模式的名称。
funcname	name	函数名称。
calls	bigint	该函数被调用的次数。
total_time	double precision	此函数及其调用的所有其他函数所花费的总时间（以毫秒为单位）。
self_time	double precision	在此函数本身中花费的总时间（不包括它调用的其他函数），以毫秒为单位。

3.2.18.Workload

3.2.18.1.WORKLOAD_SQL_COUNT

显示当前节点 workload 上的 SQL 数量分布。普通用户只可以看到自己在 workload 上的 SQL 分布；初始用户可以看到总的 workload 的负载情况。

表 1 WORKLOAD_SQL_COUNT 字段

名称	类型	描述
workload	name	负载名称。
select_count	bigint	select 数量。
update_count	bigint	update 数量。
insert_count	bigint	insert 数量。
delete_count	bigint	delete 数量。
ddl_count	bigint	ddl 数量。
dml_count	bigint	dml 数量。
dcl_count	bigint	dcl 数量。

3.2.18.2.SUMMARY_WORKLOAD_SQL_COUNT

显示 PanWeiDB 内各数据库主节点的 workload 上的 SQL 数量分布。

表 1 SUMMARY_WORKLOAD_SQL_COUNT 字段

名称	类型	描述
node_name	name	数据库进程名称。
workload	name	负载名称。
select_count	bigint	select 数量。
update_count	bigint	update 数量。
insert_count	bigint	insert 数量。
delete_count	bigint	delete 数量。
ddl_count	bigint	ddl 数量。
dml_count	bigint	dml 数量。
dcl_count	bigint	dcl 数量。

3.2.18.3.WORKLOAD_TRANSACTION

当前节点上负载的事务信息。

表 1 WORKLOAD_TRANSACTION 字段

名称	类型	描述
workload	name	负载的名称。
commit_counter	bigint	用户事务 commit 数量。
rollback_counter	bigint	用户事务 rollback 数量。
resp_min	bigint	用户事务最小响应时间（单位：微秒）。
resp_max	bigint	用户事务最大响应时间（单位：微秒）。
resp_avg	bigint	用户事务平均响应时间（单位：微秒）。
resp_total	bigint	用户事务总响应时间（单位：微秒）。
bg_commit_counter	bigint	后台事务 commit 数量。
bg_rollback_counter	bigint	后台事务 rollback 数量。
bg_resp_min	bigint	后台事务最小响应时间（单位：微秒）。
bg_resp_max	bigint	后台事务最大响应时间（单位：微秒）。
bg_resp_avg	bigint	后台事务平均响应时间（单位：微秒）。
bg_resp_total	bigint	后台事务总响应时间（单位：微秒）。

3.2.18.4.SUMMARY_WORKLOAD_TRANSACTION

显示 PanWeiDB 内汇聚的负载事务信息。

表 1 SUMMARY_WORKLOAD_TRANSACTION 字段

名称	类型	描述
workload	name	负载的名称。
commit_counter	numeric	用户事务 commit 数量。
rollback_counter	numeric	用户事务 rollback 数量。
resp_min	bigint	用户事务最小响应时间（单位：微秒）。
resp_max	bigint	用户事务最大响应时间（单位：微秒）。
resp_avg	bigint	用户事务平均响应时间（单位：微秒）。
resp_total	numeric	用户事务总响应时间（单位：微秒）。
bg_commit_counter	numeric	后台事务 commit 数量。
bg_rollback_counter	numeric	后台事务 rollback 数量。
bg_resp_min	bigint	后台事务最小响应时间（单位：微秒）。
bg_resp_max	bigint	后台事务最大响应时间（单位：微秒）。
bg_resp_avg	bigint	后台事务平均响应时间（单位：微秒）。
bg_resp_total	numeric	后台事务总响应时间（单位：微秒）。

3.2.18.5.GLOBAL_WORKLOAD_TRANSACTION

显示各节点上的 workload 的负载信息。

表 1 GLOBAL_WORKLOAD_TRANSACTION 字段

名称	类型	描述
node_name	name	数据库进程名称。
workload	name	负载的名称。
commit_counter	bigint	用户事务 commit 数量。
rollback_counter	bigint	用户事务 rollback 数量。
resp_min	bigint	用户事务最小响应时间（单位：微秒）。
resp_max	bigint	用户事务最大响应时间（单位：微秒）。
resp_avg	bigint	用户事务平均响应时间（单位：微秒）。
resp_total	bigint	用户事务总响应时间（单位：微秒）。
bg_commit_counter	bigint	后台事务 commit 数量。
bg_rollback_counter	bigint	后台事务 rollback 数量。
bg_resp_min	bigint	后台事务最小响应时间（单位：微秒）。
bg_resp_max	bigint	后台事务最大响应时间（单位：微秒）。
bg_resp_avg	bigint	后台事务平均响应时间（单位：微秒）。
bg_resp_total	bigint	后台事务总响应时间（单位：微秒）。

3.2.18.6.WORKLOAD_SQL_ELAPSE_TIME

WORKLOAD_SQL_ELAPSE_TIME 用来统计 workload（业务负载）上的 SUID 信息。

表 1 WORKLOAD_SQL_ELAPSE_TIME 字段

名称	类型	描述
workload	name	workload（业务负载）名称。
total_select_elapse	bigint	总 select 的时间花费（单位：微秒）。
max_select_elapse	bigint	最大 select 的时间花费（单位：微秒）。
min_select_elapse	bigint	最小 select 的时间花费（单位：微秒）。
avg_select_elapse	bigint	平均 select 的时间花费（单位：微秒）。
total_update_elapse	bigint	总 update 的时间花费（单位：微秒）。
max_update_elapse	bigint	最大 update 的时间花费（单位：微秒）。

名称	类型	描述
min_update_elapse	bigint	最小 update 的时间花费（单位：微秒）。
avg_update_elapse	bigint	平均 update 的时间花费（单位：微秒）。
total_insert_elapse	bigint	总 insert 的时间花费（单位：微秒）。
max_insert_elapse	bigint	最大 insert 的时间花费（单位：微秒）。
min_insert_elapse	bigint	最小 insert 的时间花费（单位：微秒）。
avg_insert_elapse	bigint	平均 insert 的时间花费（单位：微秒）。
total_delete_elapse	bigint	总 delete 的时间花费（单位：微秒）。
max_delete_elapse	bigint	最大 delete 的时间花费（单位：微秒）。
min_delete_elapse	bigint	最小 delete 的时间花费（单位：微秒）。
avg_delete_elapse	bigint	平均 delete 的时间花费（单位：微秒）。

3.2.18.7.SUMMARY_WORKLOAD_SQL_ELAPSE_TIME

SUMMARY_WORKLOAD_SQL_ELAPSE_TIME 用来统计数据库主节点上 workload（业务）负载的 SUID 信息。

表 1 SUMMARY_WORKLOAD_SQL_ELAPSE_TIM 字段

名称	类型	描述
node_name	name	数据库进程名称。
workload	name	workload（业务负载）名称。
total_select_elapse	bigint	总 select 的时间花费（单位：微秒）。
max_select_elapse	bigint	最大 select 的时间花费（单位：微秒）。
min_select_elapse	bigint	最小 select 的时间花费（单位：微秒）。
avg_select_elapse	bigint	平均 select 的时间花费（单位：微秒）。
total_update_elapse	bigint	总 update 的时间花费（单位：微秒）。
max_update_elapse	bigint	最大 update 的时间花费（单位：微秒）。
min_update_elapse	bigint	最小 update 的时间花费（单位：微秒）。
avg_update_elapse	bigint	平均 update 的时间花费（单位：微秒）。
total_insert_elapse	bigint	总 insert 的时间花费（单位：微秒）。
max_insert_elapse	bigint	最大 insert 的时间花费（单位：微秒）。
min_insert_elapse	bigint	最小 insert 的时间花费（单位：微秒）。
avg_insert_elapse	bigint	平均 insert 的时间花费（单位：微秒）。
total_delete_elapse	bigint	总 delete 的时间花费（单位：微秒）。
max_delete_elapse	bigint	最大 delete 的时间花费（单位：微秒）。

名称	类型	描述
min_delete_elapse	bigint	最小 delete 的时间花费（单位：微秒）。
avg_delete_elapse	bigint	平均 delete 的时间花费（单位：微秒）。

3.2.18.8.USER_TRANSACTION

USER_TRANSACTION 用来统计用户执行的事务信息。monadmin 用户能看到所有用户执行事务的信息，普通用户只能查询到自己执行的事务信息。

表 1 USER_TRANSACTION 字段

名称	类型	描述
username	name	用户的名称。
commit_counter	bigint	用户事务 commit 数量。
rollback_counter	bigint	用户事务 rollback 数量。
resp_min	bigint	用户事务最小响应时间（单位：微秒）。
resp_max	bigint	用户事务最大响应时间（单位：微秒）。
resp_avg	bigint	用户事务平均响应时间（单位：微秒）。
resp_total	bigint	用户事务总响应时间（单位：微秒）。
bg_commit_counter	bigint	后台事务 commit 数量。
bg_rollback_counter	bigint	后台事务 rollback 数量。
bg_resp_min	bigint	后台事务最小响应时间（单位：微秒）。
bg_resp_max	bigint	后台事务最大响应时间（单位：微秒）。
bg_resp_avg	bigint	后台事务平均响应时间（单位：微秒）。
bg_resp_total	bigint	后台事务总响应时间（单位：微秒）。

3.2.18.9.GLOBAL_USER_TRANSACTION

GLOBAL_USER_TRANSACTION 用来统计全局用户执行的事务信息。

表 1 GLOBAL_USER_TRANSACTION 字段

名称	类型	描述
node_name	name	节点名称。
username	name	用户的名称。
commit_counter	bigint	用户事务 commit 数量。
rollback_counter	bigint	用户事务 rollback 数量。
resp_min	bigint	用户事务最小响应时间（单位：微秒）。

名称	类型	描述
resp_max	bigint	用户事务最大响应时间（单位：微秒）。
resp_avg	bigint	用户事务平均响应时间（单位：微秒）。
resp_total	bigint	用户事务总响应时间（单位：微秒）。
bg_commit_counter	bigint	后台事务 commit 数量。
bg_rollback_counter	bigint	后台事务 rollback 数量。
bg_resp_min	bigint	后台事务最小响应时间（单位：微秒）。
bg_resp_max	bigint	后台事务最大响应时间（单位：微秒）。
bg_resp_avg	bigint	后台事务平均响应时间（单位：微秒）。
bg_resp_total	bigint	后台事务总响应时间（单位：微秒）。

3.3.Information Schema

信息模式本身是一个名为 information_schema 的模式。这个模式自动存在于所有数据库中。信息模式由一组视图构成，它们包含定义在当前数据库中对象的信息。这个模式的拥有者是初始数据库用户，并且该用户自然地拥有这个模式上的所有特权，包括删除它的能力。

3.3.1.ADMINISTRABLE_ROLE_AUTHORIZATIONS

administrable_role_authorizations 标识了当前用户拥有 admin 选项的所有角色。

名称	数据类型	描述
grantee	sql_identifier	授予此角色成员资格的角色的名称（可以是当前用户，或者在嵌套角色成员资格的情况下是不同的角色）
role_name	sql_identifier	角色名称
is_grantable	yes or no	总是 yes

3.3.2.APPLICABLE_ROLES

标识当前用户可以使用其权限的所有角色。当前用户本身也是一个适用的角色。适用角色的集合通常用于权限检查。

名称	数据类型	描述
grantee	sql_identifier	授予此角色成员资格的角色的名称（可以是当前用户，

名称	数据类型	描述
		或者在嵌套角色成员资格的情况下是不同的角色）
role_name	sql_identifier	角色名称
is_grantable	yes or no	YES，授予者对角色有管理员选项。NO，没有。

3.3.3.ATTRIBUTES

包含有关数据库中定义的复合数据类型的属性的信息（该视图不提供有关表列的信息，这些信息有时在 PostgreSQL 上下文中称为属性）。仅显示当前用户可以访问的那些属性（通过成为该类型的所有者或对该类型具有某些特权）。

名称	数据类型	描述
outt_catalog	sql_identifier	包含数据类型的数据库的名称（始终为当前数据库）。
udt_schema	sql_identifier	包含数据类型的模式名称。
udt_name	sql_identifier	数据类型的名称。
attribute_name	sql_identifier	属性名称。
ordinal_position	cardinal_number	数据类型中属性的序号位置（计数从 1 开始）。
attribute_default	character_data	属性的默认表达式。
is_nullable	yes_or_no	如果属性可能为空，则为 YES，如果已知它不可为空，则为 NO 。
data_type	character_data	属性的数据类型，如果它是内置类型，或者 ARRAY 如果它是某个数组（在这种情况下，请参见视图 element_types），否则为 USER-DEFINED （在这种情况下，类型在 attribute_udt_name 中标识 并关联列）。
character_maximum_length	cardinal_number	如果 data_type 标识字符或位串类型，则声明最大长度；对于所有其他数据类型，或者如果没有声明最大长度，则为 null。
character_octet_leng	cardinal_n	如果 data_type 标识字符类型，则以八位字节

名称	数据类型	描述
th	umber	（字节）为单位的数据的最大可能长度；所有其他数据类型为 null。最大八位字节长度取决于声明的字符最大长度（见上文）和服务器编码。
character_set_catalog	sql_identifier	适用于 PostgreSQL 中不可用的功能
character_set_schema	sql_identifier	适用于 PostgreSQL 中不可用的功能
character_set_name	sql_identifier	适用于 PostgreSQL 中不可用的功能
collation_catalog	sql_identifier	包含属性排序规则的数据库的名称（始终为当前数据库），如果默认或属性的数据类型不可排序则为 null
collation_schema	sql_identifier	包含属性排序规则的模式的名称，如果默认或属性的数据类型不可排序则为 null
collation_name	sql_identifier	属性排序规则的名称，默认为 null 或者属性的数据类型不可排序
numeric_precision	cardinal_number	如果 data_type 标识数字类型，则此列包含此属性的类型的（声明的或隐含的）精度。精度表示有效位数。它可以用十进制（以 10 为底）或二进制（以 2 为底）表示，如 numeric_precision_radix 列中指定的那样。对于所有其他数据类型，此列为空。
numeric_precision_radix	cardinal_number	如果 data_type 标识数字类型，则此列指示 numeric_precision 和 numeric_scale 列中的值以哪个基表示。该值为 2 或 10。对于所有其他数据类型，此列为空。
numeric_scale	cardinal_number	如果 data_type 标识一个精确的数字类型，则此列包含此属性的类型的（声明的或隐含的）比例。刻度表示小数点右侧的有效位数。它可以用十进制（以 10 为底）或二进制（以 2 为底）表示，如 numeric_precision_radix 列中指定的那样。对于所有其他数据类型，此列为空。

名称	数据类型	描述
datetime_precision	cardinal_number	如果 data_type 标识日期、时间、时间戳或间隔类型，则此列包含此属性类型的（声明的或隐含的）小数秒精度，即秒值中小数点后保留的小数位数. 对于所有其他数据类型，此列为空。
interval_type	character_data	如果 data_type 标识一个区间类型，则此列包含该属性的区间包含哪些字段的规范，例如 YEAR TO MONTH、DAY TO SECOND 等。如果未指定字段限制（即，区间接受所有字段），对于所有其他数据类型，此字段为空。
interval_precision	cardinal_number	适用于 PostgreSQL 中不可用的功能（请参阅 datetime_precision 了解间隔类型属性的小数秒精度）。
attribute_udt_catalog	sql_identifier	定义属性数据类型的数据库的名称（始终为当前数据库）。
attribute_udt_schema	sql_identifier	定义属性数据类型的模式的名称。
attribute_udt_name	sql_identifier	属性数据类型的名称。
scope_catalog	sql_identifier	适用于 PostgreSQL 中不可用的功能。
scope_schema	sql_identifier	适用于 PostgreSQL 中不可用的功能。
scope_name	sql_identifier	适用于 PostgreSQL 中不可用的功能。
maximum_cardinality	cardinal_number	始终为 null，因为数组在 PostgreSQL 中始终具有无限的最大基数。
dtd_identifier	sql_identifier	列的数据类型描述符的标识符，在与表有关的数据类型描述符中唯一。这主要用于与此类标识符的其他实例连接（标识符的具体格式没有定义，也不保证在以后的版本中保持不变）。
is_derived_reference_attribute	yes_or_no	适用于 PostgreSQL 中不可用的功能

3.3.4.CHARACTER_SETS

character_sets 标识当前数据库中可用的字符集。由于 PanWeiDB 不支持一个数据库中的多个字符集，因此该视图仅显示一个，即数据库编码。

- 角色曲目：字符的抽象集合，例如 UNICODE、UCS 或 LATIN1。不作为 SQL 对象公开，但在此视图中可见。
- 字符编码形式：一些字符库的编码。大多数较旧的字符集仅使用一种编码形式，因此它们没有单独的名称（例如，LATIN1 是适用于 LATIN1 集的编码形式）。但例如 Unicode 有 UTF8、UTF16 等编码形式（PostgreSQL 并不都支持）。编码表单不作为 SQL 对象公开，但在此视图中可见。
- 字符集：一个命名的 SQL 对象，用于标识字符库、字符编码和默认排序规则。预定义的字符集通常与编码形式具有相同的名称，但用户可以定义其他名称。例如，字符集 UTF8 通常会识别字符集 UCS、编码形式 UTF8 和一些默认排序规则。

名称	数据类型	描述
character_set_catalog	sql_identifier	字符集当前未实现为模式对象，因此该列为空。
character_set_schema	sql_identifier	字符集当前未实现为模式对象，因此该列为空。
character_set_name	sql_identifier	字符集名称，目前实现为显示数据库编码的名称。
character_repertoire	sql_identifier	字符集，如果编码为 UTF8 则显示 UCS，否则仅显示编码名称。
form_of_use	sql_identifier	字符编码形式，与数据库编码相同。
default_collate_catalog	sql_identifier	包含默认排序规则的数据库的名称（始终是当前数据库，如果标识了任何排序规则）。
default_collate_schema	sql_identifier	包含默认排序规则的架构名称。
default_collate_name	sql_identifier	默认排序规则的名称。默认排序规则被标识为与 当前数据库的 COLLATE 和 CTYPE 设置匹配的排序规则。如果没有这样的排序规则，则此列以及关联的架构和目录列为空。

3.3.5.CHECK_CONSTRAINT_ROUTINE_USAGE

标识检查约束使用的例程（函数和过程）。仅显示当前启用的角色拥有的那些例程。

名称	数据类型	描述
constraint_catalog	sql_identifier	包含约束的数据库的名称（始终为当前数据库）
constraint_schema	sql_identifier	包含约束的模式的名称
constraint_name	sql_identifier	约束的名称
specific_catalog	sql_identifier	包含函数的数据库的名称（始终为当前数据库）
specific_schema	sql_identifier	包含函数的模式的名称
specific_name	sql_identifier	函数的“特定名称”。

3.3.6.CHECK_CONSTRAINTS

包含所有检查约束，定义在表或域上，由当前启用的角色拥有（表或域的所有者是约束的所有者）。

名称	数据类型	描述
constraint_catalog	sql_identifier	包含约束的数据库的名称（始终为当前数据库）。
constraint_schema	sql_identifier	包含约束的模式的名称。
constraint_name	sql_identifier	约束的名称。
check_clause	character_data	检查约束的检查表达式。

3.3.7.COLLATIONS

视图 collations 包含当前数据库中可用的排序规则。

名称	数据类型	描述
collation_catalog	sql_identifier	包含排序规则的数据库的名称（始终为当前数据库）。
collation_schema	sql_identifier	包含排序规则的架构名称。
collation_name	sql_identifier	默认排序规则的名称。
pad_attribute	character_data	始终没有 PAD （ PostgreSQL 不支持替代的 PAD SPACE ）。

3.3.8.COLLATION_CHARACTER_SET_APPLICABILITY

标识可用排序规则适用于哪个字符集。在 PanWeiDB 中，每个数据库只有一个字符集，所以这个视图没有提供太多有用的信息。

名称	数据类型	描述
collation_catalog	sql_identifier	包含排序规则的数据库的名称（始终为当前数据库）。
collation_schema	sql_identifier	包含排序规则的架构名称。
collation_name	sql_identifier	默认排序规则的名称。
character_set_catalog	sql_identifier	字符集当前未实现为模式对象，因此该列为null。
character_set_schema	sql_identifier	字符集当前未实现为模式对象，因此该列为null。
character_set_name	sql_identifier	字符集的名称。

3.3.9.COLUMN_DOMAIN_USAGE

标识所有列（表或视图的），这些列使用当前数据库中定义的并由当前启用的角色拥有的某些域。

名称	数据类型	描述
domain_catalog	sql_identifier	包含域的数据库的名称（始终是当前数据库）。
domain_schema	sql_identifier	包含域的架构名称。
domain_name	sql_identifier	域名。
table_catalog	sql_identifier	包含表的数据库的名称（始终为当前数据库）。
table_schema	sql_identifier	包含表的模式的名称。
table_name	sql_identifier	表名。
column_name	sql_identifier	列名。

3.3.10.COLUMN_OPTIONS

包含为当前数据库中的外部表列定义的所有选项。仅显示当前用户有权访问的那些外部表列（通过成为所有者或具有某些特权）。

名称	数据类型	描述
table_catalog	sql_identifier	包含外部表的数据库的名称（始终是当前数据库）。

名称	数据类型	描述
table_schema	sql_identifier	包含外部表的模式的名称。
table_name	sql_identifier	外部表的名称。
column_name	sql_identifier	列名。
option_name	sql_identifier	选项名称。
option_value	字符数据	期权价值。

3.3.11.COLUNM_PRIVILEGES

标识在列上授予当前启用的角色或当前启用的角色的所有权限。列、授予者和被授予者的每个组合都有一行。如果已在整个表上授予权限，它将在此视图中显示为对每一列的授予，但仅适用于列粒度可能的权限类型：SELECT、INSERT、 UPDATE、REFERENCES。

名称	数据类型	描述
grantor	sql_identifier	授予权限的角色的名称。
grantee	sql_identifier	授予权限的角色的名称。
table_catalog	sql_identifier	包含该列的表的数据库名称（始终为当前数据库）。
table_schema	sql_identifier	包含包含该列的表的架构的名称。
table_name	sql_identifier	包含该列的表的名称。
column_name	sql_identifier	列名。
privilege_type	character_data	权限类型：SELECT、INSERT、 UPDATE 或 REFERENCES。
is_grantable	yes_or_no	如果特权是可授予的，则为 YES ，否则为 NO。

3.3.12.COLUMN_UDT_USAGE

标识使用当前启用的角色拥有的数据类型的所有列。请注意，在 PanWeiDB 中，内置数据类型的行为类似于用户定义的类型，因此它们也包含在此处。

名称	数据类型	描述
outt_catalog	sql_identifier	定义列数据类型（域的基础类型，如果适用）的数据库的名称（始终为当前数据库）。
udt_schema	sql_identifier	列数据类型（域的基础类型，如果适用）在其中定义的模式的名称。

名称	数据类型	描述
udt_name	sql_identifier	列数据类型的名称（域的基础类型，如果适用）。
table_catalog	sql_identifier	包含表的数据库的名称（始终为当前数据库）。
table_schema	sql_identifier	包含表的模式的名称。
table_name	sql_identifier	表名。
column_name	sql_identifier	列名。

3.3.13.COLUMNS

包含有关数据库中所有表列（或视图列）的信息。不包括系统列（oid 等）。仅显示当前用户有权访问的那些列（通过成为所有者或具有某些特权）。

由于数据类型可以在 SQL 中以多种方式定义，并且 PanWeiDB 包含定义数据类型的其他方法，因此它们在信息模式中的表示可能有些困难。列 data_type 应该标识列的底层内置类型。在 PanWeiDB 中，这意味着类型是在系统目录模式 pg_catalog 中定义的。如果应用程序可以专门处理众所周知的内置类型（例如，以不同的方式格式化数字类型或使用精度列中的数据），则此列可能很有用。udt_name、udt_schema 和 udt_catalog 列始终标识列的基础数据类型，即使列基于域。（由于 PanWeiDB 将内置类型视为用户定义类型，因此此处也出现内置类型。这是 SQL 标准的扩展。）如果应用程序希望根据类型以不同方式处理数据，则应使用这些列，因为在这种情况下，列是否真的基于域并不重要。如果列基于域，则域的标识存储在 domain_name、domain_schema 和 domain_catalog 列中。如果需要将列与其关联的数据类型配对并将域视为单独的类型，可以编写 coalesce(domain_name, udt_name)等。

名称	数据类型	描述
table_catalog	sql_identifier	包含表的数据库的名称（始终为当前数据库）。
table_schema	sql_identifier	包含表的模式的名称。
table_name	sql_identifier	表名。
column_name	sql_identifier	列名。
ordinal_position	cardinal_number	表中列的序号位置（计数从 1 开始）。
column_default	character_data	列的默认表达式。

名称	数据类型	描述
is_nullable	yes_or_no	如果该列可能为空，则为 YES，如果已知它不可为空，则为 NO。非空约束是已知列不可为空的一种方式，但也可以有其他方式。
data_type	character_data	列的数据类型，如果它是内置类型，或者 ARRAY 如果它是某个数组（在这种情况下，请参阅视图 element_types），否则为 USER-DEFINED（在这种情况下，类型在 udt_name 中标识并关联列）。如果该列基于域，则该列引用该域的基础类型（并且该域在 domain_name 和相关列中标识）。
character_maximum_length	cardinal_number	如果 data_type 标识字符或位串类型，则声明最大长度；对于所有其他数据类型，或者如果没有声明最大长度，则为 null。
character_octet_length	cardinal_number	如果 data_type 标识字符类型，则以八位字节（字节）为单位的数据的最大可能长度；所有其他数据类型为 null。最大八位字节长度取决于声明的字符最大长度（见上文）和服务器编码。
numeric_precision	cardinal_number	如果 data_type 标识数字类型，则此列包含此列类型的（声明的或隐含的）精度。精度表示有效位数。它可以用十进制（以 10 为底）或二进制（以 2 为底）表示，如 numeric_precision_radix 列中指定的那样。对于所有其他数据类型，此列为空。
numeric_precision_radix	cardinal_number	如果 data_type 标识数字类型，则此列指示 numeric_precision 和 numeric_scale 列中的值以哪个基表示。该值为 2 或 10。对于所有其他数据类型，此列为空。
numeric_scale	cardinal_numbe	如果 data_type 标识了一个精确的数字

名称	数据类型	描述
	r	类型，则此列包含此列的类型的（声明的或隐含的）比例。刻度表示小数点右侧的有效位数。它可以用十进制（以 10 为底）或二进制（以 2 为底）表示，如 numeric_precision_radix 列中指定的那样。对于所有其他数据类型，此列为空。
datetime_precision	cardinal_number	如果 data_type 标识日期、时间、时间戳或间隔类型，则此列包含此列类型的（声明的或隐含的）小数秒精度，即秒值中小数点后保留的小数位数. 对于所有其他数据类型，此列为空。
interval_type	character_data	如果 data_type 标识一个区间类型，则此列包含该列的区间包含哪些字段的规范，例如 YEAR TO MONTH、DAY TO SECOND 等。如果未指定字段限制（即，区间接受所有字段），对于所有其他数据类型，此字段为空。
interval_precision	cardinal_number	适用于 PanWeiDB 中不可用的功能（请参阅 datetime_precision 了解间隔类型列的小数秒精度）。
character_set_catalog	sql_identifier	适用于 PanWeiDB 中不可用的功能。
character_set_schema	sql_identifier	适用于 PanWeiDB 中不可用的功能。
character_set_name	sql_identifier	适用于 PanWeiDB 中不可用的功能。
collation_catalog	sql_identifier	包含列排序规则的数据库的名称（始终为当前数据库），默认为 null 或列的数据类型不可排序。
collation_schema	sql_identifier	包含列排序规则的架构名称，默认为 null 或列的数据类型不可排序。
collation_name	sql_identifier	列的排序规则名称，默认为 null 或列的数据类型不可排序。
domain_catalog	sql_identifier	如果列具有域类型，则为定义域的数据库的名称（始终为当前数据库），否则为

名称	数据类型	描述
		null。
domain_schema	sql_identifier	如果该列具有域类型，则为定义域的模式的名称，否则为 null。
domain_name	sql_identifier	如果该列具有域类型，则为域的名称，否则为 null。
outt_catalog	sql_identifier	定义列数据类型（域的基础类型，如果适用）的数据库的名称（始终为当前数据库）。
udt_schema	sql_identifier	列数据类型（域的基础类型，如果适用）在其中定义的模式的名称。
udt_name	sql_identifier	列数据类型的名称（域的基础类型，如果适用）。
scope_catalog	sql_identifier	适用于 PanWeiDB 中不可用的功能。
scope_schema	sql_identifier	适用于 PanWeiDB 中不可用的功能。
scope_name	sql_identifier	适用于 PanWeiDB 中不可用的功能。
maximum_cardinality	cardinal_number	始终为 null，因为数组在 PanWeiDB 中始终具有无限的最大基数。
dtd_identifier	sql_identifier	列的数据类型描述符的标识符，在与表有关的数据类型描述符中唯一。这主要用于与此类标识符的其他实例连接。（标识符的具体格式没有定义，也不保证在以后的版本中保持不变）。
is_self_reference	yes_or_no	PanWeiDB 中不可用的功能。
is_identity	yes_or_no	PanWeiDB 中不可用的功能。
identity_generation	character_data	PanWeiDB 中不可用的功能。
identity_start	character_data	PanWeiDB 中不可用的功能。
identity_increment	character_data	PanWeiDB 中不可用的功能。
identity_maximum	character_data	PanWeiDB 中不可用的功能。
identity_minimum	character_data	PanWeiDB 中不可用的功能。
identity_cycle	yes_or_no	PanWeiDB 中不可用的功能。
is_generated	character_data	PanWeiDB 中不可用的功能。
generation_expression	character_data	PanWeiDB 中不可用的功能。

名称	数据类型	描述
is_updateable	yes_or_no	YES 如果列是可更新的，NO 如果不是（基表中的列总是可更新的，视图中的列不一定）。

3.3.14.CONSTRAINT_COLUMN_USAGE

标识当前数据库中由某个约束使用的所有列。仅显示当前启用的角色拥有的表中包含的那些列。对于检查约束，此视图标识检查表达式中使用的列。对于外键约束，此视图标识外键引用的列。对于唯一或主键约束，此视图标识受约束的列。

名称	数据类型	描述
table_catalog	sql_identifier	包含某个约束使用的列的表的数据库的名称（始终是当前数据库）。
table_schema	sql_identifier	包含表的模式的名称，该表包含某些约束使用的列。
table_name	sql_identifier	包含某个约束使用的列的表的名称。
column_name	sql_identifier	某些约束使用的列的名称。
constraint_catalog	sql_identifier	包含约束的数据库的名称（始终为当前数据库）。
constraint_schema	sql_identifier	包含约束的模式的名称。
constraint_name	sql_identifier	约束的名称。

3.3.15.CONSTRAINT_TABLE_USAGE

标识当前数据库中由某个约束使用并由当前启用的角色拥有的所有表。（这与视图 table_constraints 不同，后者标识所有表约束以及定义它们的表。）对于外键约束，此视图标识外键引用的表。对于唯一键或主键约束，此视图仅标识约束所属的表。此视图中不包括检查约束和非空约束。

名称	数据类型	描述
table_catalog	sql_identifier	包含某些约束使用的表的数据库的名称（始终是当前数据库）。
table_schema	sql_identifier	包含由某些约束使用的表的模式的名称。
table_name	sql_identifier	某些约束使用的表的名称。

名称	数据类型	描述
constraint_catalog	sql_identifier	包含约束的数据库的名称（始终为当前数据库）。
constraint_schema	sql_identifier	包含约束的模式的名称。
constraint_name	sql_identifier	约束的名称。

3.3.16.DATA_TYPE_PRIVILEGES

标识当前用户可以访问的所有数据类型描述符，方法是作为所描述对象的所有者或对其具有某些特权。每当在表列、域或函数的定义中使用数据类型（作为参数或返回类型）时，都会生成数据类型描述符，并存储有关该数据类型在该实例中如何使用的一些信息（例如，声明的最大长度，如果适用）。每个数据类型描述符都被分配了一个任意标识符，该标识符在为一个对象（表、域、函数）分配的数据类型描述符标识符中是唯一的。此视图可能对应用程序没有用处，但它用于定义信息模式中的一些其他视图。

名称	数据类型	描述
object_catalog	sql_identifier	包含所描述对象的数据库的名称（始终为当前数据库）。
object_schema	sql_identifier	包含所描述对象的模式的名称。
object_name	sql_identifier	描述对象的名称。
object_type	character_data	所描述对象的类型：TABLE（数据类型描述符属于该表的列）、DOMAIN（数据类型描述符属于该域）、ROUTINE（数据类型描述符属于参数或返回数据）之一该函数的类型）。
dtd_identifier	sql_identifier	数据类型描述符的标识符，在同一对象的数据类型描述符中是唯一的。

3.3.17.DOMAIN_CONSTRAINTS

包含属于当前数据库中定义的域的所有约束。仅显示当前用户有权访问的那些域（通过成为所有者或具有某些特权）。

名称	数据类型	描述
constraint_catalog	sql_identifier	包含约束的数据库的名称（始终为当前数据库）。

名称	数据类型	描述
constraint_schema	sql_identifier	包含约束的模式的名称。
constraint_name	sql_identifier	约束的名称。
domain_catalog	sql_identifier	包含域的数据库的名称（始终为当前数据库）。
domain_schema	sql_identifier	包含域的架构的名称。
domain_name	sql_identifier	域名。
is_deferrable	yes_or_no	如果约束是可延迟的，则为 YES，否则为 NO。
initially_deferred	yes_or_no	如果约束是可延迟的并且最初是延迟的，则为 YES，否则为 NO。

3.3.18.DOMAIN_UDT_USAGE

标识基于当前启用角色拥有的数据类型的所有域。请注意，在 PanWeiDB 中，内置数据类型的行为类似于用户定义的类型，因此它们也包含在此处。

名称	数据类型	描述
outt_catalog	sql_identifier	定义域数据类型的数据库的名称（始终为当前数据库）。
udt_schema	sql_identifier	定义域数据类型的模式的名称。
udt_name	sql_identifier	域数据类型的名称。
domain_catalog	sql_identifier	包含域的数据库的名称（始终为当前数据库）。
domain_schema	sql_identifier	包含域的架构的名称。
domain_name	sql_identifier	域名。

3.3.19.DOMAINS

包含当前数据库中定义的所有域。仅显示当前用户有权访问的那些域（通过成为所有者或具有某些特权）。

名称	数据类型	描述
domain_catalog	sql_identifier	包含域的数据库的名称（始终为当前数据库）。
domain_schema	sql_identifier	包含域的架构的名称。
domain_name	sql_identifier	域名。
data_type	character_data	域的数据类型，如果它是内置类型，或者

名称	数据类型	描述
		ARRAY，如果它是某个数组（在这种情况下，请参见视图 element_types），否则为 USER-DEFINED（在这种情况下，类型在 udt_name 中标识并关联列）。
character_maximum_length	cardinal_number	如果域有字符或位串类型，则声明最大长度；对于所有其他数据类型，或者如果没有声明最大长度，则为 null。
character_octet_length	cardinal_number	如果域具有字符类型，则以八位字节（字节）为单位的数据的最大可能长度；所有其他数据类型为 null。最大八位字节长度取决于声明的字符最大长度（见上文）和服务器编码。
character_set_catalog	sql_identifier	适用于 PostgreSQL 中不可用的功能。
character_set_schema	sql_identifier	适用于 PostgreSQL 中不可用的功能。
character_set_name	sql_identifier	适用于 PostgreSQL 中不可用的功能。
collation_catalog	sql_identifier	包含域排序规则的数据库的名称（始终为当前数据库），默认为 null 或域的数据类型不可排序。
collation_schema	sql_identifier	包含域排序规则的架构名称，默认为 null 或域的数据类型不可排序。
collation_name	sql_identifier	域的排序规则的名称，默认为空或域的数据类型不可排序。
numeric_precision	cardinal_number	如果域具有数字类型，则此列包含此域的类型的（声明的或隐含的）精度。精度表示有效位数。它可以用十进制（以 10 为底）或二进制（以 2 为底）表示，如 numeric_precision_radix 列中指定的那样。对于所有其他数据类型，此列为空。
numeric_precision_radix	cardinal_number	如果域具有数字类型，则此列指示 numeric_precision 和 numeric_scale 列中的值在哪个基 中表示。该值为 2 或 10。对于所有其他数据类型，此列为空。

名称	数据类型	描述
numeric_scale	cardinal_number	如果域具有精确的数字类型，则此列包含此域的类型的（声明的或隐含的）比例。刻度表示小数点右侧的有效位数。它可以用十进制（以 10 为底）或二进制（以 2 为底）表示，如 numeric_precision_radix 列中指定的那样。对于所有其他数据类型，此列为空。
datetime_precision	cardinal_number	如果 data_type 标识日期、时间、时间戳或间隔类型，则此列包含此域的类型的（声明的或隐含的）小数秒精度，即秒值中小数点后保留的小数位数. 对于所有其他数据类型，此列为空。
interval_type	character_data	如果 data_type 标识一个区间类型，则此列包含该区间包含的字段的规范，例如 YEAR TO MONTH、DAY TO SECOND 等。如果未指定字段限制（即，区间接受所有字段），对于所有其他数据类型，此字段为空。
interval_precision	cardinal_number	适用于 PostgreSQL 中不可用的功能（请参阅 datetime_precision 了解间隔类型域的小数秒精度）。
domain_default	character_data	域的默认表达式。
outt_catalog	sql_identifier	定义域数据类型的数据库的名称（始终为当前数据库）。
udt_schema	sql_identifier	定义域数据类型的模式的名称。
udt_name	sql_identifier	域数据类型的名称。
scope_catalog	sql_identifier	适用于 PostgreSQL 中不可用的功能。
scope_schema	sql_identifier	适用于 PostgreSQL 中不可用的功能。
scope_name	sql_identifier	适用于 PostgreSQL 中不可用的功能。
maximum_cardinality	cardinal_number	始终为 null，因为数组在 PostgreSQL 中始终具有无限的最大基数。
dtd_identifier	sql_identifier	域的数据类型描述符的标识符，在与域有关的数据类型描述符中唯一（这是微不足道的，因为域仅包含一个数据类型描述

名称	数据类型	描述
		符）。这主要用于与此类标识符的其他实例连接（标识符的具体格式没有定义，也不保证在以后的版本中保持不变）。

3.3.20.ELEMENT_TYPES

包含数组元素的数据类型描述符。当表列、复合类型属性、域、函数参数或函数返回值被定义为数组类型时，相应的 information schema 视图仅在 data_type 列中包含 ARRAY。要获取有关数组元素类型的信息，您可以将相应的视图与此视图连接起来。例如，要显示具有数据类型和数组元素类型的表的列，如果适用，您可以执行以下操作：

```
SELECT c.column_name, c.data_type, e.data_type AS element_type
FROM information_schema.columns c LEFT JOIN information_schema.element_types e
    ON ((c.table_catalog, c.table_schema, c.table_name, 'TABLE', c.dtd_identifier)
     = (e.object_catalog, e.object_schema, e.object_name, e.object_type, e.collection_type_identifier))
WHERE c.table_schema = '...' AND c.table_name = '...'
ORDER BY c.ordinal_position;
```

此视图仅包括当前用户通过成为所有者或具有某些特权而可以访问的对象。

名称	数据类型	描述
object_catalog	sql_identifier	包含使用所描述数组的对象的数据库的名称（始终为当前数据库）。
object_schema	sql_identifier	包含使用正在描述的数组的对象的模式的名称。
object_name	sql_identifier	使用正在描述的数组的对象的名称。
object_type	character_data	使用所描述数组的对象的类型：TABLE（该数组由该表的列使用）、USER-DEFINED TYPE（该数组由该复合类型的属性使用）、DOMAIN（该数组由该域使用）， ROUTINE（该数组由参数或该函数的返回数据

名称	数据类型	描述
		类型使用）。
collection_type_identifier	sql_identifier	正在描述的数组的数据类型描述符的标识符。使用它来连接其他信息模式视图的 dtd_identifier 列。
data_type	character_data	数组元素的数据类型，如果是内置类型，则为 USER-DEFINED（在这种情况下，类型在 udt_name 和相关列中标识）。
character_maximum_length	cardinal_number	始终为 null，因为此信息不适用于 PostgreSQL 中的数组元素数据类型。
character_octet_length	cardinal_number	始终为 null，因为此信息不适用于 PostgreSQL 中的数组元素数据类型。
character_set_catalog	sql_identifier	适用于 PostgreSQL 中不可用的功能。
character_set_schema	sql_identifier	适用于 PostgreSQL 中不可用的功能。
character_set_name	sql_identifier	适用于 PostgreSQL 中不可用的功能。
collation_catalog	sql_identifier	包含元素类型排序规则的数据库名称（始终为当前数据库），如果默认为 null 或元素的数据类型不可排序。
collation_schema	sql_identifier	包含元素类型排序规则的模式名称，默认为 null 或元素的数据类型不可排序。
collation_name	sql_identifier	元素类型的排序规则名称，默认为 null 或者元素的数据类型不可排序。
numeric_precision	cardinal_number	始终为 null，因为此信息不适用于 PostgreSQL 中的数组元素数据类型。
numeric_precision_radix	cardinal_number	始终为 null，因为此信息不适用于

名称	数据类型	描述
		PostgreSQL 中的数组元素数据类型。
numeric_scale	cardinal_number	始终为 null，因为此信息不适用于 PostgreSQL 中的数组元素数据类型。
datetime_precision	cardinal_number	始终为 null，因为此信息不适用于 PostgreSQL 中的数组元素数据类型。
interval_type	character_data	始终为 null，因为此信息不适用于 PostgreSQL 中的数组元素数据类型。
interval_precision	cardinal_number	始终为 null，因为此信息不适用于 PostgreSQL 中的数组元素数据类型。
domain_default	character_data	尚未实现。
outt_catalog	sql_identifier	定义元素数据类型的数据库的名称（始终为当前数据库）。
udt_schema	sql_identifier	元素的数据类型在其中定义的模式的名称。
udt_name	sql_identifier	元素的数据类型名称。
scope_catalog	sql_identifier	适用于 PostgreSQL 中不可用的功能。
scope_schema	sql_identifier	适用于 PostgreSQL 中不可用的功能。
scope_name	sql_identifier	适用于 PostgreSQL 中不可用的功能。
maximum_cardinality	cardinal_number	始终为 null，因为数组在 PostgreSQL 中始终具有无限的最大基数。
dtd_identifier	sql_identifier	元素的数据类型描述符的标识符。这目前没有用。

3.3.21.ENABLED_ROLES

标识当前"启用的角色"。启用的角色被递归地定义为当前用户以及所有已授予具有自动继承属性的启用角色的角色。这些都是当前用户直接或间接拥有的所有角色，自动继承其成员资格。

对于权限检查，应用"适用角色"的集合，它可以比启用的角色集合更广泛。所以一般来说，最好使用 applicable_roles。

名称	数据类型	描述
role_name	sql_identifier	角色名称

3.3.22.FOREIGN_DATA_WRAPPER_OPTIONS

包含为当前数据库中的外部数据包装器定义的所有选项。仅显示当前用户有权访问的那些外部数据包装器（通过成为所有者或具有某些特权）。

名称	数据类型	描述
foreign_data_wrapper_catalog	sql_identifier	定义外部数据包装器的数据库的名称（始终为当前数据库）。
foreign_data_wrapper_name	sql_identifier	外部数据包装器的名称。
option_name	sql_identifier	选项名称。
option_value	character_data	期权价值。

3.3.23.FOREIGN_DATA_WRAPPERS

包含当前数据库中定义的所有外部数据包装器。仅显示当前用户有权访问的那些外部数据包装器（通过成为所有者或具有某些特权）。

名称	数据类型	描述
foreign_data_wrapper_catalog	sql_identifier	包含外部数据包装器的数据库的名称（始终为当前数据库）。
foreign_data_wrapper_name	sql_identifier	外部数据包装器的名称。
authorization_identifier	sql_identifier	国外服务器所有者的名字。
library_name	character_data	实现此外部数据包装器的库的文件名。

名称	数据类型	描述
foreign_data_wrapper_language	character_data	用于实现此外部数据包装器的语言。

3.3.24.FOREIGN_SERVER_OPTIONS

包含为当前数据库中的外部服务器定义的所有选项。仅显示当前用户有权访问的那些外部服务器（通过成为所有者或具有某些特权）。

名称	数据类型	描述
foreign_server_catalog	sql_identifier	定义外部服务器的数据库的名称（始终为当前数据库）。
foreign_server_name	sql_identifier	国外服务器名称。
option_name	sql_identifier	选项名称。
option_value	character_data	选项值。

3.3.25.FOREIGN_SERVERS

包含当前数据库中定义的所有外部服务器。仅显示当前用户有权访问的那些外部服务器（通过成为所有者或具有某些特权）。

名称	数据类型	描述
foreign_server_catalog	sql_identifier	定义外部服务器的数据库的名称（始终为当前数据库）。
foreign_server_name	sql_identifier	国外服务器名称。
foreign_data_wrapper_catalog	sql_identifier	包含外部服务器使用的外部数据包装器的数据库的名称（始终是当前数据库）。
foreign_data_wrapper_name	sql_identifier	外部服务器使用的外部数据包装器的名称。
foreign_server_type	字符数据	外部服务器类型信息，如果在创建时指定。
foreign_server_version	字符数据	外部服务器版本信息，如果在创建时指定。
authorization_identifier	sql_identifier	国外服务器所有者的名字。

3.3.26.FOREIGN_TABLE_OPTIONS

包含为当前数据库中的外部表定义的所有选项。仅显示当前用户有权访问的那些外部表（通过成为所有者或具有某些特权）。

名称	数据类型	描述
foreign_table_catalog	sql_identifier	包含外部表的数据库的名称（始终是当前数据库）。
foreign_table_schema	sql_identifier	包含外部表的模式的名称。
foreign_table_name	sql_identifier	外部表的名称。
option_name	sql_identifier	选项名称。
option_value	character_data	选项值。

3.3.27.FOREIGN_TABLES

包含当前数据库中定义的所有外部表。仅显示当前用户有权访问的那些外部表（通过成为所有者或具有某些特权）。

名称	数据类型	描述
foreign_table_catalog	sql_identifier	定义外部表的数据库的名称（始终为当前数据库）。
foreign_table_schema	sql_identifier	包含外部表的模式的名称。
foreign_table_name	sql_identifier	外部表的名称。
foreign_server_catalog	sql_identifier	定义外部服务器的数据库的名称（始终为当前数据库）。
foreign_server_name	sql_identifier	国外服务器名称。

3.3.28.KEY_COLUMN_USAGE

标识当前数据库中受某些唯一、主键或外键约束的所有列。此视图中不包括检查约束。仅显示当前用户通过成为所有者或具有某些特权而可以访问的那些列。

名称	数据类型	描述
constraint_catalog	sql_identifier	包含约束的数据库的名称（始终为当前数据库）。
constraint_schema	sql_identifier	包含约束的模式的名称。
constraint_name	sql_identifier	约束的名称。

名称	数据类型	描述
table_catalog	sql_identifier	包含受此约束限制的列的表的数据库的名称（始终为当前数据库）。
table_schema	sql_identifier	包含受此约束限制的列的表的架构的名称。
table_name	sql_identifier	包含受此约束限制的列的表的名称。
column_name	sql_identifier	受此约束限制的列的名称。
ordinal_position	cardinal_number	约束键中列的序号位置（计数从 1 开始）。
position_in_unique_constraint	cardinal_number	对于外键约束，引用列在其唯一约束中的序号位置（计数从 1 开始）；否则为空。

3.3.29.PARAMETERS

包含有关当前数据库中所有函数的参数（参数）的信息。仅显示当前用户有权访问的那些功能（通过成为所有者或具有某些特权）。

名称	数据类型	描述
specific_catalog	sql_identifier	包含函数的数据库的名称（始终为当前数据库）。
specific_schema	sql_identifier	包含函数的模式的名称。
specific_name	sql_identifier	函数的“特定名称”。有关详细信息，请参阅：schema->information-Schema->ROUTINES 章节获取更多信息。
ordinal_position	cardinal_number	参数在函数参数列表中的序号位置（计数从 1 开始）。
parameter_mode	character_data	IN 为输入参数， OUT 为输出参数， INOUT 为输入/输出参数。
is_result	yes_or_no	适用于 PanWeiDB 中不可用的功能。
as_locator	yes_or_no	适用于 PanWeiDB 中不可用的功能。
parameter_name	sql_identifier	参数的名称，如果参数没有名称，则返回 null。

名称	数据类型	描述
data_type	character_data	参数的数据类型，如果它是内置类型，或者 ARRAY，如果它是某个数组（在这种情况下，请参见视图 element_types），否则为 USER-DEFINED（在这种情况下，类型在 udt_name 中标识并关联列）。
character_maximum_length	cardinal_number	始终为 null，因为此信息不适用于 PanWeiDB 中的参数数据类型。
character_octet_length	cardinal_number	始终为 null，因为此信息不适用于 PanWeiDB 中的参数数据类型。
character_set_catalog	sql_identifier	适用于 PanWeiDB 中不可用的功能。
character_set_schema	sql_identifier	适用于 PanWeiDB 中不可用的功能。
character_set_name	sql_identifier	适用于 PanWeiDB 中不可用的功能。
collation_catalog	sql_identifier	始终为 null，因为此信息不适用于 PanWeiDB 中的参数数据类型。
collation_schema	sql_identifier	始终为 null，因为此信息不适用于 PanWeiDB 中的参数数据类型。
collation_name	sql_identifier	始终为 null，因为此信息不适用于 PanWeiDB 中的参数数据类型。
numeric_precision	cardinal_number	始终为 null，因为此信息不适用于 PanWeiDB 中的参数数据类型。
numeric_precision_radix	cardinal_number	始终为 null，因为此信息不适用于 PanWeiDB 中的参数数据类型。
numeric_scale	cardinal_number	始终为 null，因为此信息不适用于 PanWeiDB 中的参数数据类型。
datetime_precision	cardinal_number	始终为 null，因为此信息不适用于 PanWeiDB 中的参数数据类型。
interval_type	character_data	始终为 null，因为此信息不适用于 PanWeiDB 中的参数数据类型。
interval_precision	cardinal_number	始终为 null，因为此信息不适用于 PanWeiDB 中的参数数据类型。
outt_catalog	sql_identifier	定义参数的数据类型的数据库的名称（始终

名称	数据类型	描述
		为当前数据库）。
udt_schema	sql_identifier	参数的数据类型在其中定义的模式的名称。
udt_name	sql_identifier	参数的数据类型名称。
scope_catalog	sql_identifier	适用于 PanWeiDB 中不可用的功能。
scope_schema	sql_identifier	适用于 PanWeiDB 中不可用的功能。
scope_name	sql_identifier	适用于 PanWeiDB 中不可用的功能。
maximum_cardinality	cardinal_number	始终为 null，因为数组在 PanWeiDB 中始终具有无限的最大基数。
dtd_identifier	sql_identifier	参数的数据类型描述符的标识符，在与函数有关的数据类型描述符中唯一。这主要用于与此类标识符的其他实例连接。（标识符的具体格式没有定义，也不保证在以后的版本中保持不变）。

3.3.30.REFERENTIAL_CONSTRAINTS

包含当前数据库中的所有引用（外键）约束。仅显示当前用户对引用表具有写访问权限的那些约束（通过成为所有者或具有除 SELECT 之外的某些特权）。

名称	数据类型	描述
constraint_catalog	sql_identifier	包含约束的数据库的名称（始终为当前数据库）。
constraint_schema	sql_identifier	包含约束的模式的名称。
constraint_name	sql_identifier	约束的名称。
unique_constraint_catalog	sql_identifier	包含外键约束引用的唯一或主键约束的数据库的名称（始终为当前数据库）。
unique_constraint_schema	sql_identifier	包含外键约束引用的唯一键或主键约束的模式的名称。
unique_constraint_name	sql_identifier	外键约束引用的唯一键或主键约束的名称。
match_option	character_data	外键约束的匹配选项： FULL、PARTIAL 或 NONE。
update_rule	character_data	外键约束的更新规则：CASCADE、

名称	数据类型	描述
		SET NULL、SET DEFAULT、RESTRICT 或 NO ACTION。
delete_rule	character_data	外键约束的删除规则：CASCADE、SET NULL、SET DEFAULT、RESTRICT 或 NO ACTION。

3.3.31.ROLE_COLUMN_GRANTS

标识那些在字段上赋予或被赋予为当前角色的所有权限。 更多信息可以在 column_privileges 中找到。这个视图和 column_privileges 唯一有效的不同就是这个视图忽略了某些字段，这些字段是通过授予 PUBLIC 使得当前用户可以访问的字段。

名字	数据类型	描述
grantor	sql_identifier	被赋予这个权限的用户名称
grantee	sql_identifier	被赋予这个权限的角色名称
table_catalog	sql_identifier	包含此字段的表所在的数据库的名字（总是当前数据库）
table_schema	sql_identifier	包含该字段的表所在模式的名称
table_name	sql_identifier	包含该字段的表名称
column_name	sql_identifier	该字段的名称
privilege_type	character_data	权限的类型：SELECT,INSERT, UPDATE 或者 REFERENCES
is_grantable	yes_or_no	如果权限是可以赋予的，则为 YES，否则，为 NO

3.3.32.ROLE_ROUTINE_GRANTS

标出所有在函数上赋予或被赋予当前角色的权限。 更多的信息可以在 routine_privileges 里找到。在该视图与 routine_privileges 之间实际仅有的差异是该视图忽略那些通过赋权给 PUBLIC 使当前用户可以访问的函数。

名字	数据类型	描述
grantor	sql_identifier	被赋予该权限的角色名称
grantee	sql_identifier	被赋予此权限的角色的名称
specific_catalog	sql_identifier	包含此函数的数据库名称（总是当前数据库）

名字	数据类型	描述
specific_schema	sql_identifier	包含此函数的模式名称
specific_name	sql_identifier	函数的"specific name"（具体的名称）。 参阅：schema->information-Schema->ROUTINES 章节获取更多信息。
routine_catalog	sql_identifier	包含此函数的数据库名称（总是当前数据库）
routine_schema	sql_identifier	包含此函数的模式名称
routine_name	sql_identifier	函数的名称（可能重复，因为有重载）
privilege_type	character_data	总是 EXECUTE（函数的唯一的权限类型）
is_grantable	yes_or_no	如果权限可以赋予，那么是 YES，否则为 NO

3.3.33.ROLE_TABLE_GRANTS

标识在表或者视图上赋予或被赋予当前角色的全部权限。 更多信息可以在 table_privileges 找到。在该视图与 table_privileges 之间实际仅有的差异是该视图忽略那些通过赋权给 PUBLIC 使当前用户可以访问的表。

名称	数据类型	描述
grantor	sql_identifier	赋予权限的角色名
grantee	sql_identifier	被赋予权限的角色名
table_catalog	sql_identifier	包含此表的数据库名（总是当前数据库）
table_schema	sql_identifier	包含此表的模式名
table_name	sql_identifier	表名
privilege_type	character_data	权限类型：SELECT, INSERT, UPDATE, DELETE, TRUNCATE, REFERENCES, 或 TRIGGER
is_grantable	yes_or_no	如果权限可以赋予则为 YES，否则为 NO
with_hierarchy	yes_or_no	在 SQL 标准里，WITH HIERARCHY OPTION 是一个分开的（子）权限，允许在表继承层次结构上进行创建操作。 在 PostgreSQL 中，这包含在了 SELECT 权限中， 所以如果这个权限为 SELECT 则这个字段显示 YES， 否则为 NO。

3.3.34.ROLE_UDT_GRANTS

用于标出赋予或被赋予当前角色的用户定义类型上的 USAGE 权限。更多的信息可以在 udt_privileges 里找到。 在该视图与 udt_privileges 之间实际仅有

的差异是该视图忽略那些通过赋权给 PUBLIC 使当前用户可以访问的对象。 因为数据类型在 PostgreSQL 中并没有真实的权力，但是只有一个隐式的赋权给 PUBLIC， 所以这个视图是空的。

名称	数据类型	描述
grantor	sql_identifier	赋予该权限的角色的名字
grantee	sql_identifier	被赋予该权限的角色的名字
udt_catalog	sql_identifier	包含该类型的数据库的名字（总是当前数据库）
udt_schema	sql_identifier	包含该类型的模式的名字
udt_name	sql_identifier	类型名
privilege_type	character_data	总是 TYPE USAGE
is_grantable	yes_or_no	如果权限是可赋予的，那么就是 YES，否则为 NO

3.3.35.ROLE_USAGE_GRANTS

用于标出当前角色赋予或被赋予的各种对象的 USAGE 权限。更多的信息可以在 usage_privileges 里找到。 在该视图与 usage_privileges 之间实际仅有的差异是该视图忽略那些通过赋权给 PUBLIC 使当前用户可以访问的对象。

名称	数据类型	描述
grantor	sql_identifier	赋予该权限的角色名称。
grantee	sql_identifier	被赋予该权限的角色的名称。
object_catalog	sql_identifier	包含该对象的数据库的名字（总是当前数据库）。
object_schema	sql_identifier	如果适用，是包含该对象的模式的名字，否则是一个空字符串。
object_name	sql_identifier	对象的名字。
object_type	character_data	COLLATION 或 DOMAIN 或 FOREIGN DATA WRAPPER 或 FOREIGN SERVER 或 SEQUENCE。
privilege_type	character_data	Always USAGE。
is_grantable	yes_or_no	如果权限可以赋予，则为 YES，否则为 NO。

3.3.36.ROUTINE_PRIVILEGES

标识在函数上所有赋予当前用户或者由当前用户赋予的权限。 每个函数，授权人，和权限接受人的组合都有一行。

名称	数据类型	描述
grantor	sql_identifier	赋予权限的角色名
grantee	sql_identifier	被授予权限的角色名
specific_catalog	sql_identifier	包含该函数的数据库名称（总是当前数据库）
specific_schema	sql_identifier	包含该函数的模式名
specific_name	sql_identifier	函数的"specific name"（具体的名字）。 参阅：schema->information-Schema->ROUTINES 章节获取更多信息。
routine_catalog	sql_identifier	包含该函数的数据库名称（总是当前数据库）
routine_schema	sql_identifier	包含该函数的模式名称
routine_name	sql_identifier	函数名（可能会因重载而重复）
privilege_type	character_data	总是 EXECUTE (用于函数的唯一的权限类型)
is_grantable	yes_or_no	如果权限是可赋予的，则为 YES，否则为 NO

3.3.37.ROUTINES

包含当前数据库中的所有函数。仅显示当前用户有权访问的哪些功能（通过成为所有者或具有某些特权）。

名称	数据类型	描述
specific_catalog	sql_identifier	包含函数的数据库的名称（始终为当前数据库）。
specific_schema	sql_identifier	包含函数的模式的名称。
specific_name	sql_identifier	函数的“特定名称”。这是一个在模式中唯一标识函数的名称，即使函数的真实名称被重载。特定名称的格式没有定义，它应该仅用于将其与特定例程名称的其他实例进行比较。
routine_catalog	sql_identifier	包含函数的数据库的名称（始终为当前数据库）。
routine_schema	sql_identifier	包含函数的模式的名称。
routine_name	sql_identifier	函数名（重载时可能重复）。
routine_type	character_data	AlwaysFUNCTION（将来可能会有其他类型的例程）。
module_catalog	sql_identifier	适用于 PanWeiDB 中不可用的功能。

名称	数据类型	描述
module_schema	sql_identifier	适用于 PanWeiDB 中不可用的功能。
module_name	sql_identifier	适用于 PanWeiDB 中不可用的功能。
udt_catalog	sql_identifier	适用于 PanWeiDB 中不可用的功能。
udt_schema	sql_identifier	适用于 PanWeiDB 中不可用的功能。
udt_name	sql_identifier	适用于 PanWeiDB 中不可用的功能。
data_type	character_data	返回函数的数据类型，如果它是内置类型，或者 ARRAY 如果它是某个数组（在这种情况下，请参阅视图 ELEMENT_TYPES），否则为 USER-DEFINED（在这种情况下，类型在 type_udt_name 和相关列）。
character_maximum_length	cardinal_number	始终为 null，因为此信息不适用于 PanWeiDB 中的返回数据类型。
character_octet_length	cardinal_number	始终为 null，因为此信息不适用于 PanWeiDB 中的返回数据类型。
character_set_catalog	sql_identifier	适用于 PanWeiDB 中不可用的功能。
character_set_schema	sql_identifier	适用于 PanWeiDB 中不可用的功能。
character_set_name	sql_identifier	适用于 PanWeiDB 中不可用的功能。
collation_catalog	sql_identifier	始终为 null，因为此信息不适用于 PanWeiDB 中的返回数据类型。
collation_schema	sql_identifier	始终为 null，因为此信息不适用于 PanWeiDB 中的返回数据类型。
collation_name	sql_identifier	始终为 null，因为此信息不适用于 PanWeiDB 中的返回数据类型。
numeric_precision	cardinal_number	始终为 null，因为此信息不适用于 PanWeiDB 中的返回数据类型。
numeric_precision_radix	cardinal_number	始终为 null，因为此信息不适用于 PanWeiDB 中的返回数据类型。
numeric_scale	cardinal_number	始终为 null，因为此信息不适用于 PanWeiDB 中的返回数据类型。
datetime_precision	cardinal_number	始终为 null，因为此信息不适用于 PanWeiDB 中的返回数据类型。
interval_type	character_data	始终为 null，因为此信息不适用于

名称	数据类型	描述
		PanWeiDB 中的返回数据类型。
interval_precision	cardinal_number	始终为 null，因为此信息不适用于 PanWeiDB 中的返回数据类型。
type_udt_catalog	sql_identifier	定义函数返回数据类型的数据库名称（始终为当前数据库）。
type_udt_schema	sql_identifier	定义函数返回数据类型的模式名称。
type_udt_name	sql_identifier	函数返回数据类型的名称。
scope_catalog	sql_identifier	适用于 PanWeiDB 中不可用的功能。
scope_schema	sql_identifier	适用于 PanWeiDB 中不可用的功能。
scope_name	sql_identifier	适用于 PanWeiDB 中不可用的功能。
maximum_cardinality	cardinal_number	始终为 null，因为数组在 PanWeiDB 中始终具有无限的最大基数。
dtd_identifier	sql_identifier	此函数的返回数据类型的数据类型描述符的标识符，在与该函数有关的数据类型描述符中是唯一的。这主要用于与此类标识符的其他实例连接。（标识符的具体格式没有定义，也不保证在以后的版本中保持不变）。
routine_body	character_data	如果该函数是 SQL 函数，则为 SQL，否则为 EXTERNAL。
routine_definition	character_data	函数的源文本（如果函数不属于当前启用的角色，则为 null）。（根据 SQL 标准，此列仅在 routine_body 为 SQL 时适用，但在 PanWeiDB 中，它将包含创建函数时指定的任何源文本）。
external_name	character_data	如果此函数是 C 函数，则为函数的外部名称（链接符号）；否则为空。（这与 routine_definition 中显示的值相同）。
external_language	character_data	编写函数的语言。
parameter_style	character_data	AlwaysGENERAL（SQL 标准定义了其他参数样式，这些样式在 PanWeiDB 中不可用）。

名称	数据类型	描述
is_deterministic	yes_or_no	如果函数被声明为不可变（在 SQL 标准中称为确定性），则为 YES，否则为 NO。（您不能通过信息模式查询 PanWeiDB 中可用的其他波动级别）。
sql_data_access	character_data	总是 MODIFIES，这意味着该函数可能会修改 SQL 数据。此信息对 PanWeiDB 没有用处。
is_null_call	yes_or_no	如果函数在其任何参数为 null 时自动返回 null，则为 YES，否则为 NO。
sql_path	character_data	适用于 PanWeiDB 中不可用的功能。
schema_level_routine	yes_or_no	总是 YES（相反的是用户定义类型的方法，这是 PanWeiDB 中不可用的功能）。
max_dynamic_result_sets	cardinal_number	适用于 PanWeiDB 中不可用的功能。
is_user_defined_cast	yes_or_no	适用于 PanWeiDB 中不可用的功能。
is_implicitly_invocable	yes_or_no	适用于 PanWeiDB 中不可用的功能。
security_type	character_data	如果该函数以当前用户的权限运行，则为 INVOKER，如果该函数以定义它的用户的权限运行，则为 DEFINER。
to_sql_specific_catalog	sql_identifier	适用于 PanWeiDB 中不可用的功能。
to_sql_specific_schema	sql_identifier	适用于 PanWeiDB 中不可用的功能。
to_sql_specific_name	sql_identifier	适用于 PanWeiDB 中不可用的功能。
as_locator	yes_or_no	适用于 PanWeiDB 中不可用的功能。
created	time_stamp	适用于 PanWeiDB 中不可用的功能。
last_altered	time_stamp	适用于 PanWeiDB 中不可用的功能。
new_savepoint_level	yes_or_no	适用于 PanWeiDB 中不可用的功能。
is_udt_dependent	yes_or_no	目前总是 NO。替代 YES 适用于 PanWeiDB 中不可用的功能。
result_cast_from_data_type	character_data	适用于 PanWeiDB 中不可用的功能。
result_cast_as_locator	yes_or_no	适用于 PanWeiDB 中不可用的功能。

名称	数据类型	描述
result_cast_char_max_length	cardinal_number	适用于 PanWeiDB 中不可用的功能。
result_cast_char_octet_length	character_data	适用于 PanWeiDB 中不可用的功能。
result_cast_char_set_catalog	sql_identifier	适用于 PanWeiDB 中不可用的功能。
result_cast_char_set_schema	sql_identifier	适用于 PanWeiDB 中不可用的功能。
result_cast_char_set_name	sql_identifier	适用于 PanWeiDB 中不可用的功能。
result_cast_collation_catalog	sql_identifier	适用于 PanWeiDB 中不可用的功能。
result_cast_collation_schema	sql_identifier	适用于 PanWeiDB 中不可用的功能。
result_cast_collation_name	sql_identifier	适用于 PanWeiDB 中不可用的功能。
result_cast_numeric_precision	cardinal_number	适用于 PanWeiDB 中不可用的功能。
result_cast_numeric_precision_radix	cardinal_number	适用于 PanWeiDB 中不可用的功能。
result_cast_numeric_scale	cardinal_number	适用于 PanWeiDB 中不可用的功能。
result_cast_datetime_precision	character_data	适用于 PanWeiDB 中不可用的功能。
result_cast_interval_type	character_data	适用于 PanWeiDB 中不可用的功能。
result_cast_interval_precision	cardinal_number	适用于 PanWeiDB 中不可用的功能。
result_cast_type_udt_catalog	sql_identifier	适用于 PanWeiDB 中不可用的功能。
result_cast_type_udt_schema	sql_identifier	适用于 PanWeiDB 中不可用的功能。
result_cast_type_udt_name	sql_identifier	适用于 PanWeiDB 中不可用的功能。

名称	数据类型	描述
result_cast_scope_catalog	sql_identifier	适用于 PanWeiDB 中不可用的功能。
result_cast_scope_schema	sql_identifier	适用于 PanWeiDB 中不可用的功能。
result_cast_scope_name	sql_identifier	适用于 PanWeiDB 中不可用的功能。
result_cast_maximum_cardinality	cardinal_number	适用于 PanWeiDB 中不可用的功能。
result_cast_dtd_identifier	sql_identifier	适用于 PanWeiDB 中不可用的功能。

3.3.38.SCHEMATA

包含当前数据库中由当前启用的角色拥有的所有模式。

名称	数据类型	描述
catalog_name	sql_identifier	包含架构的数据库的名称（始终为当前数据库）。
schema_name	sql_identifier	架构的名称。
schema_owner	sql_identifier	架构所有者的名称。
default_character_set_catalog	sql_identifier	适用于 PanWeiDB 中不可用的功能。
default_character_set_schema	sql_identifier	适用于 PanWeiDB 中不可用的功能。
default_character_set_name	sql_identifier	适用于 PanWeiDB 中不可用的功能。
sql_path	character_data	适用于 PanWeiDB 中不可用的功能。

3.3.39.SEQUENCES

包含当前数据库中定义的所有序列。仅显示当前用户有权访问的那些序列（通过成为所有者或具有某些特权）。

名称	数据类型	描述

名称	数据类型	描述
sequence_catalog	sql_identifier	包含序列的数据库的名称（始终为当前数据库）。
sequence_schema	sql_identifier	包含序列的模式的名称。
sequence_name	sql_identifier	序列名称。
data_type	character_data	序列的数据类型。在 PanWeiDB 中，这目前总是 bigint。
numeric_precision	cardinal_number	此列包含序列数据类型的（声明的或隐含的）精度（见上文）。精度表示有效位数。它可以用十进制（以 10 为底）或二进制（以 2 为底）表示，如 numeric_precision_radix 列中指定的那样。
numeric_precision_radix	cardinal_number	此列指示 numeric_precision 和 numeric_scale 列中的值以哪个基数表示。该值为 2 或 10。
numeric_scale	cardinal_number	此列包含序列数据类型的（声明的或隐含的）比例（见上文）。刻度表示小数点右侧的有效位数。它可以用十进制（以 10 为底）或二进制（以 2 为底）表示，如 numeric_precision_radix 列中指定的那样。
start_value	character_data	序列的起始值。
minimum_value	character_data	序列的最小值。
maximum_value	character_data	序列的最大值。
increment	character_data	序列的增量。
cycle_option	yes_or_no	如果序列循环，则为 YES，否则为 NO。

3.3.40.SQL_FEATURES

含有关 SQL 标准中定义的正式特性受 PanWeiDB 支持的信息。

名称	数据类型	描述

名称	数据类型	描述
feature_id	character_data	特征的标识符字符串。
feature_name	character_data	功能的描述性名称。
sub_feature_id	character_data	子特征的标识符字符串，如果不是子特征，则为零长度字符串。
sub_feature_name	character_data	子功能的描述性名称，如果不是子功能，则为零长度字符串。
is_supported	yes_or_no	如果当前版本的 PanWeiDB 完全支持该功能，则为 YES，否则为 NO。
is_verified_by	character_data	始终为 null，因为 PanWeiDB 开发组不执行功能一致性的正式测试。
comments	character_data	可能是关于功能支持状态的评论。

3.3.41.SQL_IMPLEMENTATION_INFO

包含有关由 SQL 标准实现定义的各个方面的信息。此信息主要用于 ODBC 接口的上下文中；其他界面的用户可能会发现这些信息没什么用。因此，这里不描述各个实现信息项；您可以在 ODBC 接口的描述中找到它们。

名称	数据类型	描述
implementation_info_id	character_data	实现信息项的标识符字符串。
implementation_info_name	character_data	实施信息项的描述性名称。
integer_value	cardinal_number	实现信息项的值，如果该值包含在 character_value 列中，则为 null。
character_value	character_data	实现信息项的值，如果该值包含在 integer_value 列中，则为 null。
comments	character_data	可能是与实施信息项有关的评论。

3.3.42.SQL_LANGUAGES

标识 PanWeiDB 支持的每种 SQL 语言绑定的一行。PanWeiDB 支持 C 语言中的直接 SQL 和嵌入式 SQL 用户都可以从这张表中学到的全部内容。

此表已从 SQL:2008 中的 SQL 标准中删除，因此没有引用 SQL:2003 之后的标准的条目。

名称	数据类型	描述
sql_language_source	character_data	语言定义的来源名称；总是 ISO9075，即 SQL 标准。
sql_language_year	character_data	sql_language_source 中引用的标准获得批准的年份。
sql_language_conformance	character_data	语言绑定的标准一致性级别。对于 ISO9075:2003，这始终是 CORE。
sql_language_integrity	character_data	始终为空（此值与 SQL 标准的早期版本相关）。
sql_language_implementation	character_data	始终为空。
sql_language_binding_style	character_data	语言绑定样式，DIRECT 或 EMBEDDED。
sql_language_programming_language	character_data	编程语言，如果绑定样式是 EMBEDDED，否则为 null。PanWeiDB 只支持 C 语言。

3.3.43.SQL_PACKAGES

包含有关 SQL 标准中定义的哪些功能包受 PanWeiDB 支持的信息。

名称	数据类型	描述
feature_id	character_data	包的标识符字符串。
feature_name	character_data	包的描述性名称。
is_supported	yes_or_no	如果当前版本的 PanWeiDB 完全支持包，则为 YES，否则为 NO。
is_verified_by	character_data	始终为 null，因为 PanWeiDB 开发组不执行功能一致性的正式测试。
comments	character_data	可能是关于包的支持状态的评论。

3.3.44.SQL_PARTS

包含有关 SQL 标准的几个部分中的哪些部分受 PanWeiDB 支持的信息。

名称	数据类型	描述
feature_id	character_data	包含零件编号的标识符字符串。
feature_name	character_data	零件的描述性名称。
is_supported	yes_or_no	如果当前版本的 PanWeiDB 完全支持该部分，则为 YES，否则为 NO。
is_verified_by	character_data	始终为 null，因为 PanWeiDB 开发组不执行功能一致性的正式测试。
comments	character_data	可能是关于部件支持状态的评论。

3.3.45.SQL_SIZING

包含有关 PanWeiDB 中各种大小限制和最大值的信息。此信息主要用于 ODBC 接口的上下文中；其他界面的用户可能会发现这些信息没什么用。出于这个原因，这里没有描述各个尺寸的项目；用户可以在 ODBC 接口的描述中找到它们。

名称	数据类型	描述
sizing_id	cardinal_number	尺码项目的标识符。
sizing_name	character_data	尺码项目的描述性名称。
supported_value	cardinal_number	尺码项目的值，如果尺寸不受限制或无法确定，则为 0，如果不支持尺码项目适用的功能，则为 null。
comments	character_data	可能是与尺码项目有关的评论。

3.3.46.SQL_SIZING_PROFILES

包含有关 SQL 标准的各种配置文件所需的 sql_sizing 值的信息。PanWeiDB 不跟踪任何 SQL 配置文件，因此该表为空。

名称	数据类型	描述
sizing_id	cardinal_number	尺码项目的标识符。
sizing_name	character_data	尺码项目的描述性名称。
profile_id	character_data	配置文件的标识符字符串。

名称	数据类型	描述
required_value	cardinal_number	SQL 配置文件对尺码项目所需的值，如果配置文件对尺码项目没有限制，则为 0，如果配置文件不需要尺码项目适用的任何功能，则为 null。
comments	character_data	可能是与配置文件中的尺码项目有关的评论。

3.3.47.TABLE_CONSTRAINTS

包含属于当前用户拥有或具有除 SELECT 之外的某些特权的表的所有约束 。

名称	数据类型	描述
constraint_catalog	sql_identifier	包含约束的数据库的名称（始终为当前数据库）。
constraint_schema	sql_identifier	包含约束的模式的名称。
constraint_name	sql_identifier	约束的名称。
table_catalog	sql_identifier	包含表的数据库的名称（始终为当前数据库）。
table_schema	sql_identifier	包含表的模式的名称。
table_name	sql_identifier	表名。
constraint_type	character_data	约束类型：CHECK、FOREIGN KEY、PRIMARY KEY 或 UNIQUE。
is_deferrable	yes_or_no	如果约束是可延迟的，则为 YES ，否则为 NO。
initially_deferred	yes_or_no	如果约束是可延迟的并且最初是延迟的，则为 YES ，否则为 NO。

3.3.48.TABLE_PRIVILAGES

标识表或视图上授予当前启用的角色或当前启用的角色的所有特权。表、授予者和被授予者的每个组合都有一行。

名称	数据类型	描述
grantor	sql_identifier	授予权限的角色的名称。
grantee	sql_identifier	授予权限的角色的名称。
table_catalog	sql_identifier	包含表的数据库的名称（始终为当前数据库）。

名称	数据类型	描述
table_schema	sql_identifier	包含表的模式的名称。
table_name	sql_identifier	表名。
privilege_type	character_data	权限类型：SELECT、INSERT、 UPDATE、DELETE、TRUNCATE、 REFERENCES 或 TRIGGER。
is_grantable	yes_or_no	如果特权是可授予的，则为 YES ，否则为 NO。
with_hierarchy	yes_or_no	在 SQL 标准中，WITH HIERARCHY OPTION 是一个单独的（子）权限，允许对表继承层次结构进行某些操作。在 PanWeiDB 中，这包含在 SELECT 权限中，因此如果权限为 SELECT ，则此列显示 YES，否则显示 NO。

3.3.49.TABLES

包含当前数据库中定义的所有表和视图。仅显示当前用户有权访问的那些表和视图（通过成为所有者或具有某些特权）。

名称	数据类型	描述
table_catalog	sql_identifier	包含表的数据库的名称（始终为当前数据库）。
table_schema	sql_identifier	包含表的模式的名称。
table_name	sql_identifier	表名。
table_type	character_data	表的类型：BASE TABLE 用于持久基表（普通表类型），VIEW 用于视图， FOREIGN TABLE 用于外部表，或 LOCAL TEMPORARY 用于临时表。
self_referencing_column_name	sql_identifier	适用于 PanWeiDB 中不可用的功能。
reference_generation	character_data	适用于 PanWeiDB 中不可用的功能。
user_defined_type_catalog	sql_identifier	如果表是类型表，则为包含基础数据类型的数据库的名称（始终为当前数据库），否则为 null。

名称	数据类型	描述
user_defined_type_schema	sql_identifier	如果表是类型表，则为包含基础数据类型的模式的名称，否则为 null。
user_defined_type_name	sql_identifier	如果表是类型表，则为基础数据类型的名称，否则为 null。
is_insertable_into	yes_or_no	YES，如果表是可插入的，NO，如果不是（基表总是可插入的，视图不一定）。
is_typed	yes_or_no	如果表是类型表，则为 YES，否则为 NO。
commit_action	character_data	尚未实现。

3.3.50.TRIGGERED_UPDATE_COLUMNS

对于当前数据库中指定列列表的触发器（如 UPDATE OF column1, column2），视图 trigger_update_columns 标识这些列。此视图中不包含未指定列列表的触发器。仅显示当前用户拥有或具有除 SELECT on 之外的某些特权的那些列。

名称	数据类型	描述
trigger_catalog	sql_identifier	包含触发器的数据库的名称（始终为当前数据库）。
trigger_schema	sql_identifier	包含触发器的架构名称。
trigger_name	sql_identifier	触发器名称。
event_object_catalog	sql_identifier	包含定义触发器的表的数据库的名称（始终为当前数据库）。
event_object_schema	sql_identifier	包含定义触发器的表的模式的名称。
event_object_table	sql_identifier	定义触发器的表的名称。
event_object_column	sql_identifier	定义触发器的列的名称。

3.3.51.TRIGGERS

视图触发器包含当前数据库中针对当前用户拥有或具有除 SELECT 之外的某些特权的表和视图定义的所有触发器。

名称	数据类型	描述
trigger_catalog	sql_identifier	包含触发器的数据库的名称（始终为当前数据库）。
trigger_schema	sql_identifier	包含触发器的架构名称。
trigger_name	sql_identifier	触发器名称。
event_manipulation	character_data	触发触发器的事件（INSERT、UPDATE 或 DELETE）。
event_object_catalog	sql_identifier	包含定义触发器的表的数据库的名称（始终为当前数据库）。
event_object_schema	sql_identifier	包含定义触发器的表的模式的名称。
event_object_table	sql_identifier	定义触发器的表的名称。
action_order	cardinal_number	尚未实现。
action_condition	character_data	触发器的 WHEN 条件，如果没有则为 null（如果表不属于当前启用的角色，则为 null）。
action_statement	character_data	由触发器执行的语句（目前总是 EXECUTE PROCEDURE function (...)）。
action_orientation	character_data	标识触发器是为每个已处理的行触发一次还是为每个语句（ROW 或 STATEMENT）触发一次。
action_timing	character_data	触发器触发的时间（BEFORE、AFTER 或 INSTEAD OF）。
action_reference_old_table	sql_identifier	适用于 PanWeiDB 中不可用的功能。
action_reference_new_table	sql_identifier	适用于 PanWeiDB 中不可用的功能。
action_reference_old_row	sql_identifier	适用于 PanWeiDB 中不可用的功能。
action_reference_new_row	sql_identifier	适用于 PanWeiDB 中不可用的功能。
created	time_stamp	适用于 PanWeiDB 中不可用的功能。

PanWeiDB 中的触发器与影响信息模式中表示的 SQL 标准有两个不兼容之处。首先，触发器名称对于 PanWeiDB 中的每个表都是本地的，而不是独立的模式对象。因此，可以在一个模式中定义重复的触发器名称，只要它们属于不同的表。（trigger_catalog 和 trigger_schema 实际上是与定义触发器的表有关的值。）其次，触发器可以定义为在 PanWeiDB 中触发多个事件（例如，ON INSERT OR UPDATE），而 SQL 标准只允许一个。如果触发器被定义为触发多个事件，则它在信息模式中表示为多行，每种类型的事件一个。

由于这两个问题，视图触发器的主键实际上是(trigger_catalog, trigger_schema, event_object_table, trigger_name, event_manipulation) 而不是 SQL 标准指定的(trigger_catalog, trigger_schema, trigger_name) 。尽管如此，如果您以符合 SQL 标准的方式定义触发器（触发器名称在模式中是唯一的，并且每个触发器只有一个事件类型），这不会影响您。

3.3.52.UDT_PRIVILEGES

标识 在用户定义类型上授予当前启用的角色或当前启用的角色的 USAGE 权限。类型、授予者和被授予者的每种组合都有一行。此视图仅显示复合类型请参阅 USER_DEFINED_TYPES 了解原）； 有关域权限，请参见 USAGE_PRIVILEGES。

名称	数据类型	描述
grantor	sql_identifier	授予权限的角色的名称。
grantee	sql_identifier	授予权限的角色的名称。
udt_catalog	sql_identifier	包含该类型的数据库的名称（始终为当前数据库）。
udt_schema	sql_identifier	包含类型的模式的名称。
udt_name	sql_identifier	类型名称。
privilege_type	character_data	始终键入用法。
is_grantable	yes_or_no	如果特权是可授予的，则为 YES ，否则为 NO。

3.3.53.USAGE_PRIVILEGES

标识了在各种对象上授予当前启用的角色或当前启用的角色的 USAGE 特权。在 PanWeiDB 中，这目前适用于排序规则、域、外部数据包装器、外部服务器和序列。对象、授予者和被授予者的每个组合都有一行。

由于排序规则在 PanWeiDB 中没有真正的权限，因此该视图显示了所有者为所有排序规则授予 PUBLIC 的隐式不可授予的 USAGE 权限。然而，其他对象类型显示了真正的特权。

在 PanWeiDB 中，除了 USAGE 权限之外，序列还支持 SELECT 和 UPDATE 权限。这些是非标准的，因此在信息模式中不可见。

名称	数据类型	描述
grantor	sql_identifier	授予权限的角色的名称。
grantee	sql_identifier	授予权限的角色的名称。
object_catalog	sql_identifier	包含对象的数据库的名称（始终为当前数据库）。
object_schema	sql_identifier	包含对象的模式的名称（如果适用），否则为空字符串。
object_name	sql_identifier	对象名称。
object_type	character_data	COLLATION 或 DOMAIN 或 FOREIGNDATAWRAPPER 或 FOREIGNSERVER 或 SEQUENCE。
privilege_type	character_data	始终使用。
is_grantable	yes_or_no	如果特权是可授予的，则为 YES，否则为 NO。

3.3.54.USER_DEFINED_TYPES

包含当前数据库中定义的所有复合类型。仅显示当前用户有权访问的那些类型（通过成为所有者或具有某些特权）。

SQL 知道两种用户定义类型：结构化类型（在 PanWeiDB 中也称为复合类型）和不同类型（在 PanWeiDB 中未实现）。为了面向未来，请使用 user_defined_type_category 列来区分这些。其他用户定义的类型，例如基本类型和枚举，它们是 PanWeiDB 扩展，这里没有显示。对于域，请参阅：schema->information-Schema->DOMAINs 章节。

名称	数据类型	描述
user_defined_type_catalog	sql_identifier	包含该类型的数据库的名称（始终为当前数据库）。
user_defined_type_schema	sql_identifier	包含类型的架构的名称。
user_defined_type_name	sql_identifier	类型名称。
user_defined_type_category	character_data	目前始终是结构化的。
is_instantiable	yes_or_no	适用于 PanWeiDB 中不可用的功能。
is_final	yes_or_no	适用于 PanWeiDB 中不可用的功能。
ordering_form	character_data	适用于 PanWeiDB 中不可用的功能。
ordering_category	character_data	适用于 PanWeiDB 中不可用的功能。
ordering_routine_catalog	sql_identifier	适用于 PanWeiDB 中不可用的功能。
ordering_routine_schema	sql_identifier	适用于 PanWeiDB 中不可用的功能。
ordering_routine_name	sql_identifier	适用于 PanWeiDB 中不可用的功能。
reference_type	character_data	适用于 PanWeiDB 中不可用的功能。
data_type	character_data	适用于 PanWeiDB 中不可用的功能。
character_maximum_length	cardinal_number	适用于 PanWeiDB 中不可用的功能。
character_octet_length	cardinal_number	适用于 PanWeiDB 中不可用的功能。
character_set_catalog	sql_identifier	适用于 PanWeiDB 中不可用的功能。
character_set_schema	sql_identifier	适用于 PanWeiDB 中不可用的功能。
character_set_name	sql_identifier	适用于 PanWeiDB 中不可用的功能。

名称	数据类型	描述
collation_catalog	sql_identifier	适用于 PanWeiDB 中不可用的功能。
collation_schema	sql_identifier	适用于 PanWeiDB 中不可用的功能。
collation_name	sql_identifier	适用于 PanWeiDB 中不可用的功能。
numeric_precision	cardinal_number	适用于 PanWeiDB 中不可用的功能。
numeric_precision_radix	cardinal_number	适用于 PanWeiDB 中不可用的功能。
numeric_scale	cardinal_number	适用于 PanWeiDB 中不可用的功能。
datetime_precision	cardinal_number	适用于 PanWeiDB 中不可用的功能。
interval_type	character_data	适用于 PanWeiDB 中不可用的功能。
interval_precision	cardinal_number	适用于 PanWeiDB 中不可用的功能。
source_dtd_identifier	sql_identifier	适用于 PanWeiDB 中不可用的功能。
ref_dtd_identifier	sql_identifier	适用于 PanWeiDB 中不可用的功能。

3.3.55.USER_MAPPING_OPTIONS

包含为当前数据库中的用户映射定义的所有选项。仅显示当前用户可以访问相应外部服务器的那些用户映射（通过成为所有者或具有某些特权）。

名称	数据类型	描述
authorization_identifier	sql_identifier	被映射的用户的名称，如果映射是公共的，则为 PUBLIC。
foreign_server_catalog	sql_identifier	此映射使用的外部服务器在其中定义的数据库的名称（始终为当前数据库）。
foreign_server_name	sql_identifier	此映射使用的外部服务器的名称。

名称	数据类型	描述
option_name	sql_identifier	选项名称。
option_value	character_data	期权的价值。此列将显示为空，除非当前用户是被映射的用户，或者映射是针对 PUBLIC 并且当前用户是服务器所有者，或者当前用户是超级用户。目的是保护作为用户映射选项存储的密码信息。

3.3.56.USER_MAPPINGS

包含当前数据库中定义的所有用户映射。仅显示当前用户可以访问相应外部服务器的那些用户映射（通过成为所有者或具有某些特权）。

名称	数据类型	描述
authorization_identifier	sql_identifier	被映射的用户的名称，如果映射是公共的，则为 PUBLIC。
foreign_server_catalog	sql_identifier	此映射使用的外部服务器在其中定义的数据库的名称（始终为当前数据库）。
foreign_server_name	sql_identifier	此映射使用的外部服务器的名称。

3.3.57.VIEW_COLUMN_USAGE

标识在视图的查询表达式（定义视图的 SELECT 语句）中使用的所有列 。仅当包含该列的表由当前启用的角色拥有时，才会包含该列。

名称	数据类型	描述
view_catalog	sql_identifier	包含视图的数据库的名称（始终为当前数据库）。
view_schema	sql_identifier	包含视图的架构名称。
view_name	sql_identifier	视图名称。
table_catalog	sql_identifier	包含视图使用的列的表的数据库的名称（始终为当前数据库）。
table_schema	sql_identifier	包含表的架构的名称，该表包含视图使用的列。
table_name	sql_identifier	包含视图使用的列的表的名称。
column_name	sql_identifier	视图使用的列的名称。

3.3.58.VIEW_ROUTINE_USAGE

标识在视图的查询表达式（定义视图的 SELECT 语句）中使用的所有例程（函数和过程）。仅当该例程由当前启用的角色拥有时，才会包含该例程。

名称	数据类型	描述
table_catalog	sql_identifier	包含视图的数据库的名称（始终为当前数据库）。
table_schema	sql_identifier	包含视图的架构名称。
table_name	sql_identifier	视图名称。
specific_catalog	sql_identifier	包含函数的数据库的名称（始终为当前数据库）。
specific_schema	sql_identifier	包含函数的模式的名称。
specific_name	sql_identifier	函数的“特定名称”。有关详细信息，请参阅：schema->information-Schema->ROUTINES 章节获取更多信息。

3.3.59.VIEW_TABLE_USAGE

标识在视图的查询表达式（定义视图的 SELECT 语句）中使用的所有表。仅当该表由当前启用的角色拥有时，才会包含该表。

名称	数据类型	描述
view_catalog	sql_identifier	包含视图的数据库的名称（始终为当前数据库）。
view_schema	sql_identifier	包含视图的架构名称。
view_name	sql_identifier	视图名称。
table_catalog	sql_identifier	包含视图使用的表的数据库的名称（始终是当前数据库）。
table_schema	sql_identifier	包含视图使用的表的模式的名称。
table_name	sql_identifier	视图使用的表的名称。

3.3.60.VIEWS

包含当前数据库中定义的所有视图。仅显示当前用户有权访问的那些视图（通过成为所有者或具有某些特权）。

名称	数据类型	描述
table_catalog	sql_identifier	包含视图的数据库的名称（始终为当前数据库）。

名称	数据类型	描述
table_schema	sql_identifier	包含视图的架构名称。
table_name	sql_identifier	视图名称。
view_definition	character_data	定义视图的查询表达式（如果视图不属于当前启用的角色，则为 null）。
check_option	character_data	适用于 PanWeiDB 中不可用的功能。
is_updatable	yes_or_no	YES 如果视图是可更新的（允许 UPDATE 和 DELETE），NO 如果不是。
is_insertable_into	yes_or_no	YES 如果视图可插入（允许 INSERT）， NO 如果不是。
is_trigger_updatable	yes_or_no	如果视图上定义了 INSTEAD OF UPDATE 触发器，则为 YES，否则为 NO。
is_trigger_deletable	yes_or_no	如果视图上定义了 INSTEAD OF DELETE 触发器，则为 YES，否则为 NO。
is_trigger_insertable_into	yes_or_no	如果视图上定义了 INSTEAD OF INSERT 触发器，则为 YES，否则为 NO。

3.3.61._PG_FOREIGN_DATA_WRAPPERS

显示外部数据封装器的信息。该视图只有 sysadmin 权限可以查看。

表 1 _PG_FOREIGN_DATA_WRAPPERS 字段

名称	类型	描述
oid	oid	外部数据封装器的 oid。
fdwowner	oid	外部数据封装器的所有者的 oid。
fdwoptions	text[]	外部数据封装器指定选项，使用“keyword=value”格式的字符串。
foreign_data_wrapper_catalog	information_schema.sql_identifier	外部封装器所在的数据库名称（永远为当前数据库）。
foreign_data_wrapper_name	information_schema.sql_identifier	外部数据封装器名称。

名称	类型	描述
authorization_identifier	information_schema.sql_identifier	外部数据封装器所有者的角色名称。
foreign_data_wrapper_language	information_schema.character_data	外部数据封装器的实现语言。

3.3.62._PG_FOREIGN_SERVERS

显示外部服务器的信息。该视图只有 sysadmin 权限可以查看。

表 1 _PG_FOREIGN_SERVERS 字段

名称	类型	描述
oid	oid	外部服务器的 oid。
srvoptions	text[]	外部服务器指定选项，使用“keyword=value”格式的字符串。
foreign_server_catalog	information_schema.sql_identifier	外部服务器所在 database 名称（永远为当前数据库）。
foreign_server_name	information_schema.sql_identifier	外部服务器名称。
foreign_data_wrapper_catalog	information_schema.sql_identifier	外部数据封装器所在 database 名称（永远为当前数据库）。
foreign_data_wrapper_name	information_schema.sql_identifier	外部数据封装器名称。
foreign_server_type	information_schema.character_data	外部服务器的类型。
foreign_server_version	information_schema.character_data	外部服务器的版本。
authorization_identifier	information_schema.sql_identifier	外部服务器的所有者的角色名称。

3.3.63._PG_FOREIGN_TABLE_COLUMNS

显示外部表的列信息。该视图只有 sysadmin 权限可以查看。

表 1 _PG_FOREIGN_TABLE_COLUMNS 字段

名称	类型	描述
nspname	name	schema 名称。
relname	name	表名称。
attname	name	列名称。
attfdwoptions	text[]	外部数据封装器的属性选项，使用 “keyword=value” 格式的字符串。

3.3.64._PG_FOREIGN_TABLES

存储所有的定义在本数据库的外部表信息。只显示当前用户有权访问的外部表信息。该视图只有 sysadmin 权限可以查看。

表 1 _PG_FOREIGN_TABLES 字段

名称	类型	描述
foreign_table_catalog	information_schema.sql_identifier	外部表所在的数据库名称（永远是当前数据库）。
foreign_table_schema	name	外部表的 schema 名称。
foreign_table_name	name	外部表的名称。
ftoptions	text[]	外部表的可选项。
foreign_server_catalog	information_schema.sql_identifier	外部服务器所在的数据库名称（永远是当前数据库）。
foreign_server_name	information_schema.sql_identifier	外部服务器的名称。
authorization_identifier	information_schema.sql_identifier	所有者的角色名称。

3.3.65._PG_USER_MAPPINGS

存储从本地用户到远程的映射。该视图只有 sysadmin 权限可以查看。

表 1 _PG_USER_MAPPINGS 字段

名称	类型	描述
oid	oid	从本地用户到远程的映射的 oid。
umoptions	text[]	用户映射指定选项，使用“keyword=value”格式的字符串。
umuser	oid	被映射的本地用户的 OID，如果用户映射是公共的则为 0。
authorization_identifier	information_schema.sql_identifier	本地用户角色名称。
foreign_server_catalog	information_schema.sql_identifier	外部服务器定义所在的 database 名称。
foreign_server_name	information_schema.sql_identifier	外部服务器名称。
srvowner	information_schema.sql_identifier	外部服务器所有者。

3.3.66.INFORMATION_SCHEMA_CATALOG_NAME

用来显示当前所在的 database 的名称。

表 1 INFORMATION_SCHEMA_CATALOG_NAME 字段

名称	类型	描述
catalog_name	information_schema.sql_identifier	当前 database 的名称。

3.4.DBE_PLDEBUGGER Schema

DBE_PLDEBUGGER 下系统函数用于单机下调试存储过程，目前支持的接口及其描述如下所示。仅管理员有权限执行这些调试接口，且无权限修改和创建新函数。

【须知】
当在函数体中创建用户时，调用 attach、next、continue、 info_code、step、info_breakpoint、backtrace、 finish 中会返回密码的明文。因此不建议用户在函数体中创建用户。

对应权限角色为 gs_role_pldebugger，可以由管理员用户通过如下命令将 debugger 权限赋权给该用户。

```
GRANT gs_role_pldebugger to user;
```

需要有两个客户端连接数据库，一个客户端负责执行调试接口作为 debug 端，另一个客户端执行调试函数，控制 server 端存储过程执行。示例如下。

❖ **准备调试**

通过 PG_PROC，查找到待调试存储过程的 oid，并执行 DBE_PLDEBUGGER.turn_on(oid)。本客户端就会作为 server 端使用。

调试匿名块时，执行 DBE_PLDEBUGGER.turn_on(0)。

1、创建存储过程。

```
   CREATE OR REPLACE PROCEDURE test_debug ( IN  x INT)
AS
BEGIN
  INSERT INTO t1 (a) VALUES (x);
  DELETE FROM t1 WHERE a = x;
END;
/
```

2、执行如下语句进行查询操作。

```
SELECT OID FROM PG_PROC WHERE PRONAME='test_debug';
SELECT * FROM DBE_PLDEBUGGER.turn_on(16389);
```

查询结果为如下：

```
 oid
-------
 16389
(1 row)

 nodename | port
----------+------
 datanode |    0
(1 row)
```

❖ **开始调试**

server 端执行存储过程，会在存储过程内第一条 SQL 语句前 hang 住，等待 debug 端发送的调试消息。仅支持直接执行存储过程的调试，不支持通过 trigger 调用执行的存储过程调试。匿名块在 turn_on 后输入执行即可。

```
call test_debug(1);

--匿名块
do $$
declare
  # declaration_section
begin
  # executable_section
end;
$$
LANGUAGE plpgsql;
```

再起一个客户端，作为 debug 端，通过 turn_on 返回的数据，调用 DBE_PLDEBUGGER.attach 关联到该存储过程上进行调试。

```
SELECT * FROM DBE_PLDEBUGGER.attach('datanode',0);
```

查询结果显示为如下：

```
 funcoid |  funcname  | lineno |          query
---------+------------+--------+---------------------------------
   16389 | test_debug |      3 |  INSERT INTO t1 (a) VALUES (x);
(1 row)
```

在执行 attach 的客户端调试，执行下一条 statement。

```
SELECT * FROM DBE_PLDEBUGGER.next();
```

查询结果显示为如下：

```
 funcoid |  funcname  | lineno |      query
---------+------------+--------+---------------------
   16389 | test_debug |      0 | [EXECUTION FINISHED]
(1 row)
```

在执行 attach 的客户端调试，可以执行以下变量操作

```
SELECT * FROM DBE_PLDEBUGGER.info_locals(); --打印全部变量
SELECT * FROM DBE_PLDEBUGGER.set_var('x', 2); --变量赋值
SELECT * FROM DBE_PLDEBUGGER.print_var('x'); --打印单个变量
```

查询结果显示为如下：

```
varname | vartype | value | package_name | isconst
---------+---------+-------+--------------+---------
 x       | int4    | 1     |              | f
(1 row)

 set_var
---------
 t
(1 row)

 varname | vartype | value | package_name | isconst
---------+---------+-------+--------------+---------
 x       | int4    | 2     |              | f
(1 row)
```

❖ **以下几种方式均可以完成当前调试。**

➢ 直接执行完成当前正在调试的存储过程。

```
SELECT * FROM DBE_PLDEBUGGER.continue();
```

查询结果显示为如下：

```
 funcoid |  funcname  | lineno |      query
---------+------------+--------+---------------------
   16389 | test_debug |      0 | [EXECUTION FINISHED]
(1 row)
```

➢ 直接退出当前正在调试的存储过程，不执行尚未执行的语句。

```
SELECT * FROM DBE_PLDEBUGGER.abort();
```

查询结果显示为如下：

```
 abort
-------
 t
(1 row)
```

✧ client 端操作：

1、查看代码信息并识别可以设置断点行号。

```
SELECT * FROM DBE_PLDEBUGGER.info_code(16389);
```

查询结果显示为如下：

```
lineno |                  query                  | canbreak
--------+-----------------------------------------------------+----------
       | CREATE OR REPLACE PROCEDURE public.test_debug( IN  x INT) | f
     1 | AS  DECLARE                                         | f
     2 | BEGIN                                               | f
     3 |    INSERT INTO t1 (a) VALUES (x);                   | t
     4 |    DELETE FROM t1 WHERE a = x;                      | t
     5 | END;                                                | f
     6 | /                                                   | f
(7 rows)
```

2、设置断点。

```
SELECT * FROM DBE_PLDEBUGGER.add_breakpoint(16389,4);
```

结果显示为如下：

```
 lineno |                  query                  | canbreak
--------+-----------------------------------------------------+----------
       | CREATE OR REPLACE PROCEDURE public.test_debug( IN  x INT) | f
     1 | AS  DECLARE                                         | f
     2 | BEGIN                                               | f
     3 |    INSERT INTO t1 (a) VALUES (x);                   | t
```

```
          4 |    DELETE FROM t1 WHERE a = x;                  | t
          5 | END;                                            | f
          6 | /                                               | f
    (7 rows)
```

3、查看断点信息。

```
SELECT * FROM DBE_PLDEBUGGER.info_breakpoints();
```

结果显示为如下：

```
breakpointno | funcoid | lineno | query | enable
--------------+---------+--------+-------------------------------+--------
0 | 16389 | 4 | DELETE FROM t1 WHERE a = x; | t
(1 row)
```

4、执行至断点。

```
SELECT * FROM DBE_PLDEBUGGER.continue();
```

结果显示为如下：

```
    funcoid |  funcname  | lineno |          query
   ---------+------------+--------+--------------------------------
     16389 | test_debug |    4 |    DELETE FROM t1 WHERE a = x;
  (1 row)
```

存储过程执行结束后，调试会自动退出，再进行调试需要重新 attach 关联。如果 server 端不需要继续调试，可执行 turn_off 关闭，或退出 session。具体调试接口请见下面列表。

表 1 DBE_PLDEBUGGER

接口名称	描述
DBE_PLDEBUGGER.turn_on]	server 端调用，标记存储过程可以调试，调用后执行该存储过程时会 hang 住等待调试信息。
DBE_PLDEBUGGER.turn_off	server 端调用，标记存储过程关闭调试。
DBE_PLDEBUGGER.local_debug_server_info	server 端调用，打印本 session 内所有已 turn_on 的存储过程。
DBE_PLDEBUGGER.attach	debug 端调用，关联到正在调试存储

接口名称	描述
	过程。
DBE_PLDEBUGGER.info_locals	debug 端调用，打印正在调试的存储过程中的变量当前值。
DBE_PLDEBUGGER.next	debug 端调用，单步执行。
DBE_PLDEBUGGER.continue	debug 端调用，继续执行，直到断点或存储过程结束。
DBE_PLDEBUGGER.abort	debug 端调用，停止调试，server 端报错长跳转。
DBE_PLDEBUGGER.print_var	debug 端调用，打印正在调试的存储过程中指定的变量当前值。
DBE_PLDEBUGGER.info_code	debug 和 server 端都可以调用，打印指定存储过程的源语句和各行对应的行号。
DBE_PLDEBUGGER.step	debug 端调用，单步进入执行。
DBE_PLDEBUGGER.add_breakpoint	debug 端调用，新增断点。
DBE_PLDEBUGGER.delete_breakpoint	debug 端调用，删除断点。
DBE_PLDEBUGGER.info_breakpoints	debug 端调用，查看当前的所有断点。
DBE_PLDEBUGGER.backtrace	debug 端调用，查看当前的调用栈。
DBE_PLDEBUGGER.enable_breakpoint	debug 端调用，激活被禁用的断点。
DBE_PLDEBUGGER.disable_breakpoint	debug 端调用，禁用已激活的断点。
DBE_PLDEBUGGER.finish	debug 端调用，继续调试，直到断点或返回上一层调用栈。
DBE_PLDEBUGGER.set_var	debug 端调用，为变量进行赋值操作。

3.4.1.DBE_PLDEBUGGER.turn_on

该函数用于标记某一存储过程为可调试，执行 turn_on 后 server 端可以执行该存储过程来进行调试。需要用户根据系统表 PG_PROC 手动获取存储过程 oid，传入函数中。turn_on 后本 session 内执行该存储过程会停在第一条 sql 前等待 debug 端的调试操作。该设置会在 session 断连后默认被清理掉。目前不支持对启用自治事务的存储过程/函数进行调试。

函数原型为：

```
DBE_PLDEBUGGER.turn_on(Oid)
RETURN Record;
```

表 1 turn_on 入参和返回值列表

名称	类型	描述
func_oid	IN oid	函数 oid。
nodename	OUT text	节点名称。
port	OUT integer	连接端口号。

3.4.2.DBE_PLDEBUGGER.turn_off

用于去掉 turn_on 添加的调试标记，返回值表示成功或失败。可通过 DBE_PLDEBUGGER.local_debug_server_info 查找已经 turn_on 的存储过程 oid。

函数原型为：

```
DBE_PLDEBUGGER.turn_off(Oid)
RETURN boolean;
```

表 1 turn_off 入参和返回值列表

名称	类型	描述
func_oid	IN oid	函数 oid。
turn_off	OUT boolean	turn off 是否成功。

3.4.3.DBE_PLDEBUGGER.local_debug_server_info

用于查找当前连接中已经 turn_on 的存储过程 oid。便于用户确认在调试哪些存储过程，需要通过 funcoid 和 pg_proc 配合使用。

表 1 local_debug_server_info 返回值列表

名称	类型	描述
nodename	OUT text	节点名称。
port	OUT bigint	端口号。
funcoid	OUT oid	存储过程 oid。

3.4.4.DBE_PLDEBUGGER.attach

server 端执行存储过程，停在第一条语句前，等待 debug 端关联。debug 端调用 attach，传入 nodename 和 port，关联到该存储过程上。

如果调试过程中报错，attach 会自动失效；如果调试过程中 attach 到其他存储过程上，当前 attach 的调试也会失效。

表 1 attach 入参和返回值列表

名称	类型	描述
nodename	IN text	节点名称。
port	IN integer	连接端口号。
funcoid	OUT oid	函数 id。
funcname	OUT text	函数名。
lineno	OUT integer	当前调试运行的下一行行号。
query	OUT text	当前调试的下一行函数源码。

3.4.5.DBE_PLDEBUGGER.info_locals

debug 端调试过程中，调用 info_locals，打印当前存储过程内变量。该函数入参 frameno 表示查询遍历的栈层数，支持无入参调用，缺省为查看最上层栈变量。

表 1 info_locals 入参和返回值列表

名称	类型	描述
frameno	IN integer （可选）	指定的栈层数，缺省为最顶层
varname	OUT text	变量名
vartype	OUT text	变量类型
value	OUT text	变量值
package_name	OUT text	变量对应的 package 名，非 package 时为空
isconst	OUT boolean	是否为常量

3.4.6.DBE_PLDEBUGGER.next

执行存储过程中当前的 sql，返回执行的下一条的行数和对应 query。

表 1 next 返回值列表

名称	类型	描述
funcoid	OUT oid	函数 id。
funcname	OUT text	函数名。
lineno	OUT integer	当前调试运行的下一行行号。
query	OUT text	当前调试的下一行函数源码。

3.4.7.DBE_PLDEBUGGER.continue

执行当前存储过程，直到下一个断点或结束。返回值表示执行的下一条的行数和对应 query。

函数原型为：

```
DBE_PLDEBUGGER.continue()
RETURN Record;
```

表 1 continue 返回值列表

名称	类型	描述
funcoid	OUT oid	函数 id。
funcname	OUT text	函数名。
lineno	OUT integer	当前调试运行的下一行行号。
query	OUT text	当前调试的下一行函数源码。

3.4.8.DBE_PLDEBUGGER.abort

令 server 端执行的存储过程报错跳出。返回值表示是否成功发送 abort。

函数原型为：

```
DBE_PLDEBUGGER.abort()
RETURN boolean;
```

表 1 abort 返回值列表

名称	类型	描述
abort	OUT boolean	表示成功或失败。

3.4.9.DBE_PLDEBUGGER.print_var

debug 端调试过程中，调用 print_var，打印当前存储过程内变量中指定的变量名及其取值。该函数入参 frameno 表示查询遍历的栈层数，支持不加入该参数调用，缺省为查看最上层栈变量。

表 1 print_var 入参和返回值列表

名称	类型	描述
var_name	IN text	变量。
frameno	IN integer（可选）	指定的栈层数，缺省为最顶层。
varname	OUT text	变量名。
vartype	OUT text	变量类型。
value	OUT text	变量值。
package_name	OUT text	变量对应的 package 名，预留使用，当前均为空。
isconst	OUT boolean	是否为常量。

3.4.10.DBE_PLDEBUGGER.info_code

debug 端调试过程中，调用 info_code，查看指定存储过程的源语句和各行对应的行号，行号从函数体开始，函数头部分行号为空。

表 1 info_code 入参和返回值列表

名称	类型	描述
funcoid	IN oid	函数 ID。
lineno	OUT integer	行号。
query	OUT text	源语句。
canbreak	OUT bool	当前行是否支持断点。

3.4.11.DBE_PLDEBUGGER.step

debug 端调试过程中，如果当前执行的是一个存储过程，则进入该存储过程继续调试，返回该存储过程第一行的行号等信息，如果当前执行的不是存储过程，则和 next 行为一致，执行该 sql 后返回下一行的行号等信息。

表 1 step 入参和返回值列表

名称	类型	描述
funcoid	OUT oid	函数 id。
funcname	OUT text	函数名。
lineno	OUT integer	当前调试运行的下一行行号。
query	OUT text	当前调试的下一行函数源码。

3.4.12.DBE_PLDEBUGGER.add_breakpoint

debug 端调试过程中，调用 add_breakpoint 增加新的断点。如果返回-1 则说明指定的断点不合法，请参考 DBE_PLDEBUGGER.info_code 的 canbreak 字段确定断点合适的位置。

表 1 add_breakpoint 入参和返回值列表

名称	类型	描述
funcoid	IN text	函数 ID。
lineno	IN integer	行号。
breakpointno	OUT integer	断点编号。

3.4.13.DBE_PLDEBUGGER.delete_breakpoint

debug 端调试过程中，调用 delete_breakpoint 删除已有的断点。

表 1 delete_breakpoint 入参和返回值列表

名称	类型	描述
breakpointno	OUT integer	断点编号。
funcoid	OUT oid	函数 ID。
lineno	OUT integer	行号。
query	OUT text	断点内容。

名称	类型	描述
enable	OUT boolean	是否有效。

3.4.14.DBE_PLDEBUGGER.info_breakpoints

debug 端调试过程中，调用 info_breakpoints，查看当前的函数断点。

表 1 info_breakpoints 返回值列表

名称	类型	描述
breakpointno	IN integer	断点编号。
result	OUT bool	是否成功。

3.4.15.DBE_PLDEBUGGER.backtrace

debug 端调试过程中，调用 backtrace，查看当前的调用堆栈。

表 1 backtrace 返回值列表

名称	类型	描述
frameno	OUT integer	调用栈编号。
funcname	OUT oid	函数名。
lineno	OUT integer	行号。
query	OUT text	断点内容。
funcoid	OUT oid	函数 oid。

3.4.16.DBE_PLDEBUGGER.enable_breakpoint

debug 端调试过程中，调用 enable_breakpoint 激活已被禁用的断点。

表 1 enable_breakpoint 入参和返回值列表

名称	类型	描述
breakpointno	IN integer	断点编号
result	OUT bool	是否成功

3.4.17.DBE_PLDEBUGGER.disable_breakpoint

debug 端调试过程中，调用 disable_breakpoint 禁用已被激活的断点。

表 1 disable_breakpoint 入参和返回值列表

名称	类型	描述
breakpointno	IN integer	断点编号
result	OUT bool	是否成功

3.4.18.DBE_PLDEBUGGER.finish

执行存储过程中当前的 SQL 直到下一个断点触发或执行到上层栈的下一行。

表 1 finish 入参和返回值列表

名称	类型	描述
funcoid	OUT oid	函数 id。
funcname	OUT text	函数名。
lineno	OUT integer	当前调试运行的下一行行号。
query	OUT text	当前调试的下一行函数源码。

3.4.19.DBE_PLDEBUGGER.set_var

将指定的调试的存储过程中最上层栈上的变量修改为入参的取值。如果存储过程中包含同名的变量，set_var 只支持第一个变量值的设置。

表 1 set_var 入参和返回值列表

名称	类型	描述
var_name	IN text	变量名。
value	IN text	修改值。
result	OUT boolean	结果，是否成功。

3.5.DBE_PLDEVELOPER

DBE_PLDEVELOPER 下系统表用于记录 PLPGSQL 包、函数及存储过程编译过程中需要记录的信息。

3.5.1.DBE_PLDEVELOPER.gs_source

用于记录 PLPGSQL 对象（存储过程、函数、包、包体）编译相关信息，具体内容见下列字段描述。

无论 PLPGSQL 对象编译成功或失败，均会把编译信息记录在此表中。

【注意】

- gs_source 表中只记录用户定义的原始对象语句，即使用户使用了 ALTER 改变了创建的 SCHEMA 或者名字，gs_source 表中的信息也不会发生变化，如果用户更改了对象的 SCHEMA 或者名字，会导致用户在删除对象后，对象仍存在于 gs_source 表中。
- gs_source 表中的 owner 指创建的用户，不是用户创建存储过程或者 package 时指定的用户。
- 数据库默认情况下没有对 gs_source 表中设置行级访问控制，如果用户想使用数据库隔离性特性，请参考以下语句，自行添加行级访问控制。

  ALTER TABLE dbe_pldeveloper.gs_source ENABLE ROW LEVEL SECURITY;

  CREATE ROW LEVEL SECURITY POLICY all_data_rls ON dbe_pldeveloper.gs_source USING(owner = (select oid from pg_roles where rolname=current_user));

表 1 DBE_PLDEVELOPER.gs_source 字段

名称	类型	描述
id	oid	对象的 ID。
owner	bigint	对象创建用户 ID。
nspid	oid	对象的模式 ID。
name	name	对象名。
type	text	对象类型（procedure/function/package/package body）。
status	boolean	是否创建成功。
src	text	对象创建的原始语句。

3.5.2.DBE_PLDEVELOPER.gs_errors

用于记录 PLPGSQL 对象（存储过程、函数、包、包体）编译过程中遇到的报错信息，具体内容见下列字段描述。

打开 plsql_show_all_error 参数后，如果编译过程中存在报错，则会跳过报错继续编译并把报错信息记录在 gs_errors 中，如果关闭 plsql_show_all_error 参数则不会将相关信息插入此表中。

表 1 DBE_PLDEVELOPER.gs_errors 字段

名称	类型	描述
id	oid	对象的 ID。
owner	bigint	对象创建用户 ID。
nspid	oid	对象的模式 ID。
name	name	对象名。
type	text	对象类型（procedure/function/package/package body）。
line	integer	行号。
src	text	对象创建的原始语句。

3.6.DB4AI Schema

DB4AI 模式在 AI 特性中主要是用来存储和管理数据集版本。模式中保存数据表的原始视图快照，每一个数据版本的更改记录以及版本快照的管理信息。模式面向普通用户，用户可在该模式下查找特性 DB4AI.SNAPSHOT 创建的快照版本信息。

3.6.1.DB4AI.SNAPSHOT

SNAPSHOT 表记录当前用户通过特性 DB4AI.SNAPSHOT 存储的快照。

表 1 db4ai.snapshot 表属性

名称	类型	描述	实例
id	bigint	当前快照的 ID。	1
parent_id	bigint	父快照的 ID。	0
matrix_id	bigint	CSS 模式下快照的矩阵 ID，否则为 NULL。	0

名称	类型	描述	实例
root_id	bigint	初始快照的 ID，通过 db4ai.create_snapshot()从操作数据构建。	0
schema	name	导出快照视图的模式。	public
name	name	快照的名称，包括版本后缀。	example0@1.1.0
owner	name	创建此快照的用户的名称。	nw
commands	text[]	记录如何从其根快照生成到此快照的 SQL 语句的完整列表。	{DELETE, “WHERE id > 7” }
comment	text	快照说明。	inherits from @1.0.0
published	boolean	TRUE，当且仅当快照当前已发布。	f
archived	boolean	TRUE，当且仅当快照当前已存档。	f
created	timestamp without time zone	快照创建日期的时间戳。	2021-08-25 10:59:52.955604
row_count	bigint	此快照中的数据行数。	8

3.6.2.DB4AI.CREATE_SNAPSHOT

CREATE_SNAPSHOT 是 DB4AI 特性用于创建快照的接口函数。通过语法 CREATE SNAPSHOT 调用。

表 1 DB4AI.CREATE_SNAPSHOT 入参和返回值列表

参数	类型	描述
i_schema	IN NAME	快照存储的模式名字，默认值是当前用户或者 PUBLIC。
i_name	IN NAME	快照名称。
i_commands	IN TEXT[]	定义数据获取的 SQL 命令。
i_vers	IN NAME	版本后缀。
i_comment	IN TEXT	快照描述。
res	OUT db4ai.snapshot_name	结果。

3.6.3.DB4AI.CREATE_SNAPSHOT_INTERNAL

CREATE_SNAPSHOT_INTERNAL 是 db4ai.create_snapshot 函数的内置执行函数。函数存在信息校验，无法直接调用。

表 1 DB4AI.CREATE_SNAPSHOT_INTERNAL 入参和返回值列表

参数	类型	描述
s_id	IN BIGINT	快照 ID。
i_schema	IN NAME	快照存储的名字空间。
i_name	IN NAME	快照名称。
i_commands	IN TEXT[]	定义数据获取的 SQL 命令。
i_comment	IN TEXT	快照描述。
i_owner	IN NAME	快照拥有者。

3.6.4.DB4AI.PREPARE_SNAPSHOT

PREPARE_SNAPSHOT 是 DB4AI 特性中数据准备模型训练和解释快照进行协作。快照为所有应用更改的数据和文档提供了完整的序列。通过语法 PREPARE SNAPSHOT 调用。

表 1 DB4AI.PREPARE_SNAPSHOT 入参和返回值列表

参数	类型	描述
i_schema	IN NAME	快照存储的模式名字，默认值是当前用户或者 PUBLIC。
i_parent	IN NAME	父快照名称。
i_commands	IN TEXT[]	定义快照修改的 DDL 和 DML 命令。
i_vers	IN NAME	版本后缀。
i_comment	IN TEXT	此数据策展单元的说明。
res	OUT db4ai.snapshot_name	结果。

3.6.5.DB4AI.PREPARE_SNAPSHOT_INTERNAL

PREPARE_SNAPSHOT_INTERNAL 是 db4ai.prepare_snapshot 函数的内置执行函数。函数存在信息校验，无法直接调用。

表 1 DB4AI.PREPARE_SNAPSHOT_INTERNAL 入参和返回值列表

参数	类型	描述
s_id	IN BIGINT	快照 ID。
p_id	IN BIGINT	父快照 ID。
m_id	IN BIGINT	矩阵 id。
r_id	IN BIGINT	根快照 ID。
i_schema	IN NAME	快照模式。
i_name	IN NAME	快照名称。
i_commands	IN TEXT[]	定义快照修改的 DDL 和 DML 命令。
i_comment	IN TEXT	快照描述。
i_owner	IN NAME	快照所有者。
i_idx	INOUT INT	exec_cmds 的索引。
i_exec_cmds	INOUT TEXT[]	用于执行的 DDL 和 DML。
i_mapping	IN NAME[]	将用户列映射到备份列；如果不为 NULL，则生成规则。

3.6.6.DB4AI.ARCHIVE_SNAPSHOT

ARCHIVE_SNAPSHOT 是 DB4AI 特性用于存档快照的接口函数。通过语法 ARCHIVE SNAPSHOT 调用。生效后的快照无法参数训练等任务。

表 1 DB4AI.ARCHIVE_SNAPSHOT 入参和返回值列表

参数	类型	描述
i_schema	IN NAME	快照存储的模式名字，默认值是当前用户
i_name	IN NAME	快照名称
res	OUT db4ai.snapshot_name	结果

3.6.7.DB4AI.PUBLISH_SNAPSHOT

PUBLISH_SNAPSHOT 是 DB4AI 特性用于发布快照的接口函数。通过语法 PUBLISH SNAPSHOT 调用。

表 1 DB4AI.PUBLISH_SNAPSHOT 入参和返回值列表

参数	类型	描述

参数	类型	描述
i_schema	IN NAME	快照存储的模式名字，默认值是当前用户或者 PUBLIC
i_name	IN NAME	快照名称
res	OUT db4ai.snapshot_name	结果

3.6.8.DB4AI.MANAGE_SNAPSHOT_INTERNAL

MANAGE_SNAPSHOT_INTERNAL 是 DB4AI.PUBLISH_SNAPSHOT 和 DB4AI.ARCHIVE_SNAPSHOT 函数的内置执行函数。函数存在信息校验，无法直接调用。

表 1 DB4AI.MANAGE_SNAPSHOT_INTERNAL 入参和返回值列表

参数	类型	描述
i_schema	IN NAME	快照存储的模式名字
i_name	IN NAME	快照名称
publish	IN BOOLEN	是否是发布状态
res	OUT db4ai.snapshot_name	结果

3.6.9.DB4AI.SAMPLE_SNAPSHOT

SAMPLE_SNAPSHOT 是 DB4AI 特性用于对基数据进行采样生成快照的接口函数。通过语法 SAMPLE SNAPSHOT 调用。

表 1 DB4AI.SAMPLE_SNAPSHOT 入参和返回值列表

参数	类型	描述
i_schema	IN NAME	快照存储的模式名字
i_parent	IN NAME	父快照名称
i_sample_infixes	IN NAME[]	示例快照名称中缀
i_sample_ratios	IN NUMBER[]	每个样本的大小，作为父集的比率
i_stratify	IN NAME[]	分层策略
i_sample_comments	IN TEXT[]	示例快照描述
res	OUT	结果

参数	类型	描述
	db4ai.snapshot_name	

3.6.10.DB4AI.PURGE_SNAPSHOT

PURGE_SNAPSHOT 是 DB4AI 特性用于删除快照的接口函数。通过语法 PURGE SNAPSHOT 调用。

表 1 DB4AI.PURGE_SNAPSHOT 入参和返回值列表

参数	类型	描述
i_schema	IN NAME	快照存储的模式名字
i_name	IN NAME	快照名称
res	OUT db4ai.snapshot_name	结果

3.6.11.DB4AI.PURGE_SNAPSHOT_INTERNAL

PURGE_SNAPSHOT_INTERNAL 是 DB4AI.PURGE_SNAPSHOT 函数的内置执行函数。函数存在信息校验，无法直接调用

表 1 DB4AI.PURGE_SNAPSHOT_INTERNAL 入参和返回值列表

参数	类型	描述
i_schema	IN NAME	快照存储的模式名字
i_name	IN NAME	快照名称

4. SQL语法参考

概述

SQL 是用于访问和处理数据库的标准计算机语言。它是数据库用户与数据库交互的方式。

SQL 提供了各种任务的语句，包括但不限于：

- 查询数据。
- 在表中插入、更新和删除行。
- 创建、替换、更改和删除对象。
- 控制对数据库及其对象的访问。
- 保证数据库的一致性和完整性。

SQL 语言由用于处理数据库和数据库对象的命令和函数组成。该语言还会强制实施有关数据类型、表达式和文本使用的规则。因此在本章节，除了 SQL 语法外，还会看到有关数据类型、表达式、函数和操作符等信息。

SQL 发展简史

SQL 发展简史如下：

1986 年，ANSI X3.135-1986，ISO/IEC 9075:1986，SQL-86

1989 年，ANSI X3.135-1989，ISO/IEC 9075:1989，SQL-89

1992 年，ANSI X3.135-1992，ISO/IEC 9075:1992，SQL-92（SQL2）

1999 年，ISO/IEC 9075:1999，SQL:1999（SQL3）

2003 年，ISO/IEC 9075:2003，SQL:2003（SQL4）

2011 年，ISO/IEC 9075:200N，SQL:2011（SQL5）

PanWeiDB 支持的 SQL 标准

PanWeiDB 默认支持 SQL2、SQL3 和 SQL4 的主要特性。

4.1.SQL 基本要素

本章主要介绍 SQL 基本要素。

4.1.1.关键字

SQL 里有保留字和非保留字之分。根据标准，保留字决不能用做其他标识符。非保留字只是在特定的环境里有特殊的含义，而在其他环境里是可以用做标识符的。

标识符的命名需要遵守如下规范：

- 标识符需要为字母、下划线、数字（0-9）或美元符号（$）。
- 标识符必须以字母（a-z）或下划线（_）开头。

此命名规范为建议项，非强制项。特殊情况下可以使用双引号规避特殊字符报错。

4.1.1.1.Oracle 兼容模式关键字

在 PanWeiDB 的 Oracle 兼容模式下，关键字及其类别的清单如表 1（SQL 关键字）和表 2（PL/pgSQL 关键字）所示。

表 1 Oracle 兼容模式 SQL 关键字

关键字	类别
abort	非保留
absolute	非保留
access	非保留
accessed	非保留
account	非保留
action	非保留
add	非保留
admin	非保留
after	非保留
aggregate	非保留

关键字	类别
algorithm	非保留
all	保留
also	非保留
alter	非保留
always	非保留
analyse	保留
analyze	保留
and	保留
any	保留
app	非保留
append	非保留
archive	非保留
array	保留
as	保留
asc	保留
assertion	非保留
assignment	非保留
asymmetric	保留
at	非保留
attribute	非保留
audit	非保留
authid	非保留 (不可用于函数、类型名)
authorization	保留 (可用于函数、类型名)
auto_increment	保留
autoextend	非保留
automapped	非保留
backward	非保留
barrier	非保留
before	非保留
begin	非保留
begin_non_anoyblock	非保留
between	非保留 (不可用于函数、类型名)

关键字	类别
bfile	非保留
bigint	非保留 (不可用于函数、类型名)
binary	保留 (可用于函数、类型名)
binary_double	非保留 (不可用于函数、类型名)
binary_float	非保留 (不可用于函数、类型名)
binary_integer	非保留 (不可用于函数、类型名)
bit	非保留 (不可用于函数、类型名)
blanks	非保留
blob	非保留
block	非保留
blockchain	非保留
body	保留
boolean	非保留 (不可用于函数、类型名)
both	保留
bucketcnt	非保留 (不可用于函数、类型名)
buckets	保留
by	非保留
byte	非保留 (不可用于函数、类型名)
byteawithoutorder	非保留 (不可用于函数、类型名)
byteawithoutorderwithequal	非保留 (不可用于函数、类型名)
cache	非保留
call	非保留
called	非保留
cancelable	非保留
cascade	非保留
cascaded	非保留
case	保留
cast	保留
catalog	非保留
chain	非保留
change	非保留
char	非保留 (不可用于函数、类型名)

关键字	类别
character	非保留 (不可用于函数、类型名)
characteristics	非保留
characterset	非保留
charset	非保留
check	保留
checkpoint	非保留
checktable	保留
class	非保留
clean	非保留
client	非保留
client_master_key	非保留
client_master_keys	非保留
clob	非保留
close	非保留
cluster	非保留
coalesce	非保留 (不可用于函数、类型名)
collate	保留
collation	保留 (可用于函数、类型名)
column	保留
column_encryption_key	非保留
column_encryption_keys	非保留
columns	非保留
comment	非保留
comments	非保留
commit	非保留
committed	非保留
compact	保留 (可用于函数、类型名)
compatible_illegal_chars	非保留
complete	非保留
completion	非保留
compress	非保留
concurrently	保留 (可用于函数、类型名)

关键字	类别
condition	非保留
configuration	非保留
conflict	非保留
connect	非保留
connect_by_root	非保留
connection	非保留
constant	非保留
constraint	保留
constraints	非保留
constructor	非保留
contains	非保留
content	非保留
context	非保留
continue	非保留
contview	非保留
conversion	非保留
convert	非保留 (不可用于函数、类型名)
coordinator	非保留
coordinators	非保留
copy	非保留
cost	非保留
create	保留
cross	保留 (可用于函数、类型名)
csn	保留 (可用于函数、类型名)
csv	非保留
cube	非保留
current	非保留
current_catalog	保留
current_date	保留
current_role	保留
current_schema	保留 (可用于函数、类型名)
current_time	保留

关键字	类别
current_timestamp	保留
current_user	保留
cursor	非保留
cycle	非保留
data	非保留
database	非保留
datafile	非保留
datanode	非保留
datanodes	非保留
datatype_cl	非保留
date	非保留 (不可用于函数、类型名)
date_add	非保留 (不可用于函数、类型名)
date_format	非保留
datetime	非保留
datetime2	非保留
day	非保留
dbcc	保留
dbcompatibility	非保留
dblink	非保留
dbname	非保留
dbtimezone	保留
deallocate	非保留
dec	非保留 (不可用于函数、类型名)
decimal	非保留 (不可用于函数、类型名)
declare	非保留
decode	非保留
default	保留
defaults	非保留
deferrable	保留
deferred	非保留
definer	非保留
delayed	非保留

关键字	类别
delete	非保留
delimiter	非保留
delimiters	非保留
delta	非保留
deltamerge	保留 (可用于函数、类型名)
demand	非保留
dense_rank	非保留
desc	保留
deterministic	非保留
dictionary	非保留
direct	非保留
directory	非保留
disable	非保留
discard	非保留
disconnect	非保留
distinct	保留
distribute	非保留
distribution	非保留
do	保留
document	非保留
domain	非保留
double	非保留
double_compatible	非保留 (不可用于函数、类型名)
drop	非保留
dumpfile	非保留
duplicate	非保留
each	非保留
elastic	非保留
else	保留
enable	非保留
enclosed	非保留
encoding	非保留

关键字	类别
encrypted	非保留
encrypted_value	非保留
encryption	非保留
encryption_type	非保留
end	保留
ends	非保留
enforced	非保留
enum	非保留
eol	非保留
error	非保留
errors	非保留
escape	非保留
escaped	非保留
escaping	非保留
event	非保留
events	非保留
every	非保留
except	保留
exchange	非保留
exclude	非保留
excluded	保留
excluding	非保留
exclusive	非保留
exec	非保留
execute	非保留
exists	非保留 (不可用于函数、类型名)
expired	非保留
explain	非保留
extension	非保留
external	非保留
extract	非保留 (不可用于函数、类型名)
FALSE	保留

关键字	类别
family	非保留
fast	非保留
features	非保留
fenced	非保留
fetch	保留
fields	非保留
fileheader	非保留
fill_missing_fields	非保留
filler	非保留
filter	非保留
final	非保留
first	非保留
fixed	非保留
flashback	保留
float	非保留 (不可用于函数、类型名)
following	非保留
follows	非保留
for	保留
force	非保留
foreign	保留
formatter	非保留
forward	非保留
freeze	保留 (可用于函数、类型名)
from	保留
full	保留 (可用于函数、类型名)
function	非保留
functions	非保留
generated	非保留
global	非保留
globally	非保留
grant	保留
granted	非保留

关键字	类别
greatest	非保留 (不可用于函数、类型名)
group	保留
grouping	非保留 (不可用于函数、类型名)
grouping_id	非保留 (不可用于函数、类型名)
groupparent	保留
handler	非保留
having	保留
hdfsdirectory	保留 (可用于函数、类型名)
header	非保留
hold	非保留
hour	非保留
identified	非保留
identity	非保留
if	非保留
ignore_extra_data	非保留
ilike	保留 (可用于函数、类型名)
image	保留 (可用于函数、类型名)
immediate	非保留
immutable	非保留
implicit	非保留
in	保留
include	非保留
including	非保留
increment	非保留
incremental	非保留
index	非保留
indexes	非保留
infile	非保留
inherit	非保留
inherits	非保留
initial	非保留
initially	保留

关键字	类别
initrans	非保留
inline	非保留
inner	保留 (可用于函数、类型名)
inout	非保留 (不可用于函数、类型名)
input	非保留
insensitive	非保留
insert	非保留
instead	非保留
int	非保留 (不可用于函数、类型名)
integer	非保留 (不可用于函数、类型名)
internal	非保留
intersect	保留
interval	非保留 (不可用于函数、类型名)
into	保留
invisible	非保留
invoker	非保留
ip	非保留
is	保留
isnull	非保留
isolation	非保留
join	保留 (可用于函数、类型名)
json_mergepatch	保留
keep	非保留
key	非保留
key_path	非保留
key_store	非保留
kill	非保留
label	非保留
language	非保留
large	非保留
last	非保留
lc_collate	非保留

关键字	类别
lc_ctype	非保留
leading	保留
leakproof	非保留
least	非保留 (不可用于函数、类型名)
left	保留 (可用于函数、类型名)
less	保留
level	非保留
like	保留 (可用于函数、类型名)
limit	保留
lines	非保留
link	非保留
list	非保留
listen	非保留
load	非保留
local	非保留
localtime	保留
localtimestamp	保留
location	非保留
lock	非保留
locked	非保留
log	非保留
logging	非保留
logical_data	保留
logical_file	保留
login_any	非保留
login_failure	非保留
login_success	非保留
logout	非保留
long	保留 (可用于函数、类型名)
loop	非保留
lower_case_table_names	非保留
map	非保留

关键字	类别
mapping	非保留
masking	非保留
master	非保留
match	非保留
matched	非保留
materialized	非保留
maxextents	非保留
maxsize	非保留
maxtrans	非保留
maxvalue	非保留
member	非保留
merge	非保留
minextents	非保留
minus	保留
minute	非保留
minvalue	非保留
mode	非保留
model	非保留
modify	保留
month	非保留
move	非保留
movement	非保留
name	非保留
names	非保留
national	非保留 (不可用于函数、类型名)
natural	保留 (可用于函数、类型名)
nchar	非保留 (不可用于函数、类型名)
never	非保留
next	非保留
no	非保留
nocache	非保留
nocompress	非保留

关键字	类别
nocycle	保留
node	非保留
noforce	非保留
nologging	非保留
nomaxvalue	非保留
nominvalue	非保留
none	非保留 (不可用于函数、类型名)
noorder	非保留
normally	非保留
not	保留
nothing	非保留
notify	非保留
notnull	保留 (可用于函数、类型名)
nowait	非保留
null	保留
nullcols	非保留
nullif	非保留 (不可用于函数、类型名)
nulls	非保留
number	非保留 (不可用于函数、类型名)
numeric	非保留 (不可用于函数、类型名)
numstr	非保留
nvarchar	非保留 (不可用于函数、类型名)
nvarchar2	非保留 (不可用于函数、类型名)
nvl	非保留 (不可用于函数、类型名)
nvl2	非保留 (不可用于函数、类型名)
object	非保留
of	非保留
off	保留
offset	保留
oids	非保留
on	保留
only	保留

关键字	类别
operator	非保留
optimization	非保留
option	非保留
optionally	非保留
options	非保留
or	保留
ora_hash	非保留 (不可用于函数、类型名)
order	保留
out	非保留 (不可用于函数、类型名)
outer	保留 (可用于函数、类型名)
outfile	非保留
output	保留
over	非保留
overlaps	保留 (可用于函数、类型名)
overlay	非保留 (不可用于函数、类型名)
overriding	非保留
owned	非保留
owner	非保留
package	非保留
packages	非保留
pad_attribute	非保留
parallel	非保留
parser	非保留
partial	非保留
partition	非保留
partitions	非保留
passing	非保留
password	非保留
pctfree	非保留
per	非保留
percent	非保留
performance	保留

关键字	类别
perm	非保留
physical_only	保留
pipelined	非保留
pivot	非保留
placing	保留
plan	非保留
plans	非保留
policy	非保留
pool	非保留
position	非保留 (不可用于函数、类型名)
precedes	非保留
preceding	非保留
precision	非保留 (不可用于函数、类型名)
predict	非保留
preferred	非保留
prefix	非保留
prepare	非保留
prepared	非保留
preserve	非保留
primary	保留
prior	非保留
priorer	非保留
private	非保留
privilege	非保留
privileges	非保留
procedural	非保留
procedure	保留
profile	非保留
public	非保留
publication	非保留
publish	非保留
purge	非保留

关键字	类别
query	非保留
quote	非保留
randomized	非保留
range	非保留
ratio	非保留
raw	非保留
read	非保留
real	非保留 (不可用于函数、类型名)
reassign	非保留
rebuild	非保留
recheck	非保留
recursive	非保留
recyclebin	保留 (可用于函数、类型名)
redisanyvalue	非保留
ref	非保留
references	保留
refresh	非保留
reindex	非保留
reject	保留
relative	非保留
release	非保留
reloptions	非保留
remote	非保留
remove	非保留
rename	非保留
repeat	非保留
repeatable	非保留
replace	非保留
replica	非保留
reserve	非保留
reset	非保留
resize	非保留

关键字	类别
resource	非保留
restart	非保留
restrict	非保留
result	非保留
return	保留
returning	保留
returns	非保留
reuse	非保留
revoke	非保留
right	保留 (可用于函数、类型名)
role	非保留
roles	非保留
rollback	非保留
rollup	非保留
rotation	非保留
row	非保留 (不可用于函数、类型名)
rowid	非保留
rownum	保留
rows	非保留
rowtype	非保留
rule	非保留
sample	保留 (可用于函数、类型名)
savepoint	非保留
schedule	非保留
schema	非保留
score	非保留
scroll	非保留
search	非保留
second	非保留
security	非保留
seed	非保留
select	保留

关键字	类别
self	非保留
sensitive	非保留
sequence	非保留
sequences	非保留
serializable	非保留
server	非保留
session	非保留
session_user	保留
sessiontimezone	保留
set	非保留
setof	非保留 (不可用于函数、类型名)
sets	非保留
share	非保留
shippable	非保留
show	非保留
shrink	保留
shutdown	非保留
siblings	非保留
similar	保留 (可用于函数、类型名)
simple	非保留
size	非保留
skip	非保留
slave	非保留
slice	非保留
smalldatetime	非保留 (不可用于函数、类型名)
smalldatetime_format	非保留
smallint	非保留 (不可用于函数、类型名)
smallmoney	非保留
snapshot	非保留
some	保留
source	非保留
space	非保留

关键字	类别
spill	非保留
split	非保留
sql	非保留
stable	非保留
standalone	非保留
start	非保留
starting	非保留
starts	非保留
statement	非保留
statement_id	非保留
static	非保留
statistics	非保留
stdin	非保留
stdout	非保留
storage	非保留
store	非保留
stored	非保留
stratify	非保留
stream	非保留
strict	非保留
strip	非保留
subpartition	非保留
subpartitions	非保留
subscription	非保留
substring	非保留 (不可用于函数、类型名)
symmetric	保留
synonym	非保留
sys_refcursor	非保留
sysdate	保留
sysid	非保留
system	非保留
systimestamp	非保留

关键字	类别
table	保留
tables	非保留
tablesample	保留 (可用于函数、类型名)
tablespace	非保留
target	非保留
temp	非保留
template	非保留
temporary	非保留
terminated	非保留
text	非保留
than	非保留
then	保留
threshold	非保留
time	非保留 (不可用于函数、类型名)
time_format	非保留
time_period	非保留
timecapsule	保留 (可用于函数、类型名)
timestamp	非保留 (不可用于函数、类型名)
timestamp_format	非保留
timestampdiff	非保留 (不可用于函数、类型名)
tinyint	非保留 (不可用于函数、类型名)
to	保留
trailing	保留
transaction	非保留
transform	非保留
treat	非保留 (不可用于函数、类型名)
trigger	非保留
trim	非保留 (不可用于函数、类型名)
TRUE	保留
truncate	非保留
trusted	非保留
tsfield	非保留

关键字	类别
tstag	非保留
tstime	非保留
type	非保留
types	非保留
unbounded	非保留
uncommitted	非保留
under	非保留
unencrypted	非保留
union	保留
unique	保留
unknown	非保留
unlimited	非保留
unlisten	非保留
unlock	非保留
unlogged	非保留
unpivot	非保留
until	非保留
unusable	非保留
update	非保留
useeof	非保留
user	保留
using	保留
vacuum	非保留
valid	非保留
validate	非保留
validation	非保留
validator	非保留
value	非保留
values	非保留 (不可用于函数、类型名)
varbinary	保留 (可用于函数、类型名)
varchar	非保留 (不可用于函数、类型名)
varchar2	非保留 (不可用于函数、类型名)

关键字	类别
variables	非保留
variadic	保留
varray	非保留
varying	非保留
pw_hash	非保留 (不可用于函数、类型名)
vcgroup	非保留
verbose	保留 (可用于函数、类型名)
verify	保留
version	非保留
view	非保留
visible	非保留
volatile	非保留
wait	非保留
warning	非保留
warnings	非保留
weak	非保留
when	保留
where	保留
while	非保留
whitespace	非保留
window	保留
with	保留
within	非保留
without	非保留
work	非保留
workload	非保留
wrapper	非保留
write	非保留
xml	非保留
xmlattributes	非保留 (不可用于函数、类型名)
xmlconcat	非保留 (不可用于函数、类型名)
xmlelement	非保留 (不可用于函数、类型名)

关键字	类别
xmlexists	非保留 (不可用于函数、类型名)
xmlforest	非保留 (不可用于函数、类型名)
xmlparse	非保留 (不可用于函数、类型名)
xmlpi	非保留 (不可用于函数、类型名)
xmlroot	非保留 (不可用于函数、类型名)
xmlserialize	非保留 (不可用于函数、类型名)
year	非保留
yes	非保留
zone	非保留

表 2 PL/pgSQL 保留关键字

关键字	类别
absolute	非保留
all	保留
alias	非保留
alter	非保留
array	非保留
as	非保留
backward	非保留
begin	保留
bulk	非保留
by	保留
call	保留
case	保留
close	保留
collate	保留
collect	非保留
commit	非保留
constant	非保留
continue	非保留
current	非保留

关键字	类别
cursor	非保留
debug	非保留
declare	保留
default	保留
delete	保留
detail	非保留
diagnostics	保留
distinct	非保留
do	非保留
dump	非保留
else	保留
elseif	保留
elsif	保留
end	保留
errcode	非保留
error	非保留
except	非保留
exception	保留
exceptions	非保留
execute	保留
exit	非保留
fetch	保留
first	非保留
for	保留
forall	保留
foreach	保留
forward	非保留
from	保留
function	非保留
get	非保留
goto	保留
hint	非保留

关键字	类别
if	保留
immediate	非保留
in	保留
index	非保留
info	非保留
insert	保留
instantiation	非保留
intersect	非保留
into	保留
is	非保留
iterate	非保留
last	非保留
leave	非保留
limit	保留
log	非保留
merge	非保留
message	非保留
message_text	非保留
move	保留
multiset	非保留
next	非保留
no	非保留
not	保留
notice	非保留
null	保留
of	保留
oop	保留
open	保留
option	非保留
or	保留
out	保留
package	非保留

关键字	类别
perform	非保留
pg_exception_context	非保留
pg_exception_detail	非保留
pg_exception_hint	非保留
pipe	非保留
pls_integer	非保留
pragma	非保留
print_strict_params	非保留
prior	非保留
procedure	保留
query	非保留
raise	保留
range	非保留
record	非保留
ref	保留
relative	非保留
release	非保留
repeat	非保留
result_oid	非保留
return	保留
returned_sqlstate	非保留
reverse	非保留
rollback	非保留
row	非保留
row_count	非保留
rowtype	非保留
save	非保留
savepoint	非保留
scroll	非保留
select	保留
slice	非保留
sqlstate	非保留

关键字	类别
stacked	非保留
strict	保留
subtype	非保留
sys_refcursor	非保留
table	非保留
then	保留
to	保留
top	保留
type	非保留
union	非保留
until	非保留
update	保留
updating	保留
use_column	非保留
use_variable	非保留
using	保留
values	非保留
variable_conflict	非保留
varray	非保留
warning	非保留
when	保留
while	保留
with	非保留

4.1.1.2.MySQL 兼容模式关键字

在 PanWeiDB 的 MySQL 兼容模式下，关键字及其类别的清单如表 1（SQL 关键字）和表 2（PL/pgSQL 关键字）所示。

表 1 SQL 关键字

关键字	类别
abort	非保留

关键字	类别
absolute	非保留
abstime	非保留
access	非保留
accessed	非保留
account	非保留
action	非保留
add	非保留
admin	非保留
after	非保留
against	保留 (可用于函数、类型名)
aggregate	非保留
algorithm	非保留
all	保留
also	非保留
alter	非保留
always	非保留
analyse	保留
analyze	保留
and	保留
any	保留
app	非保留
append	非保留
archive	非保留
array	保留
as	保留
asc	保留
assertion	非保留
assignment	非保留

关键字	类别
asymmetric	保留
at	非保留
attribute	非保留
audit	非保留
authid	非保留 (不可用于函数、类型名)
authorization	非保留
auto_increment	非保留
autoextend	非保留
autoextend_size	非保留
automapped	非保留
avg_row_length	非保留
backward	非保留
barrier	非保留
before	非保留
begin	非保留
begin_non_anoyblock	非保留
between	非保留 (不可用于函数、类型名)
bfile	非保留
bigint	非保留 (不可用于函数、类型名)
binary	非保留
binary_double	非保留 (不可用于函数、类型名)
binary_float	非保留 (不可用于函数、类型名)
binary_integer	非保留 (不可用于函数、类型名)
bit	非保留 (不可用于函数、类型名)
blanks	非保留
blob	非保留
block	非保留
blockchain	非保留

关键字	类别
body	保留
boolean	非保留 (不可用于函数、类型名)
bool	非保留
both	保留
box	非保留
bucketcnt	非保留 (不可用于函数、类型名)
buckets	保留
by	非保留
byte	非保留 (不可用于函数、类型名)
bytea	非保留
byteawithoutorder	非保留 (不可用于函数、类型名)
byteawithoutorderwithequal	非保留 (不可用于函数、类型名)
cache	非保留
call	非保留
called	非保留
cancelable	非保留
cascade	非保留
cascaded	非保留
case	保留
cast	保留
catalog	非保留
catalog_name	非保留
chain	非保留
change	非保留
char	非保留 (不可用于函数、类型名)
character	非保留 (不可用于函数、类型名)
characteristics	非保留
characterset	非保留

关键字	类别
charset	非保留
check	保留
checkpoint	非保留
checksum	非保留
checktable	保留
circle	非保留
class	非保留
class_origin	非保留
clean	非保留
client	非保留
client_master_key	非保留
client_master_keys	非保留
clob	非保留
close	非保留
cluster	非保留
coalesce	非保留 (不可用于函数、类型名)
collate	保留
collation	保留 (可用于函数、类型名)
column	保留
column_encryption_key	非保留
column_encryption_keys	非保留
column_name	非保留
columns	非保留
comment	非保留
comments	非保留
commit	非保留
committed	非保留
compact	非保留

关键字	类别
compatible_illegal_chars	非保留
complete	非保留
completion	非保留
compress	非保留
compression	非保留
concurrently	非保留
condition	非保留
configuration	非保留
conflict	非保留
connect	非保留
connect_by_root	保留
connection	非保留
constant	非保留
constraint	保留
constraint_catalog	非保留
constraint_name	非保留
constraint_schema	非保留
constraints	非保留
contains	非保留
content	非保留
context	非保留
continue	非保留
contview	非保留
conversion	非保留
convert	非保留 (不可用于函数、类型名)
coordinator	非保留
coordinators	非保留
copy	非保留

关键字	类别
cost	非保留
create	保留
cross	保留 (可用于函数、类型名)
csn	保留 (可用于函数、类型名)
csv	非保留
cube	非保留
current	非保留
current_catalog	保留
current_date	保留
current_role	保留
current_schema	非保留
current_time	保留
current_timestamp	保留
current_user	保留
cursor	非保留
cursor_name	非保留
curtime	保留
cycle	非保留
data	非保留
database	非保留
databases	非保留
datafile	非保留
datanode	非保留
datanodes	非保留
datatype_cl	非保留
date	非保留 (不可用于函数、类型名)
date_add	非保留 (不可用于函数、类型名)
date_format	非保留

关键字	类别
date_sub	非保留
datetime	非保留
datetime2	非保留
day	非保留
day_hour	非保留
day_microsecond	非保留
day_minute	非保留
day_second	非保留
dbcc	保留
dbcompatibility	非保留
dblink	非保留
dbname	非保留
dbtimezone	保留
deallocate	非保留
dec	非保留 (不可用于函数、类型名)
decimal	非保留 (不可用于函数、类型名)
declare	非保留
decode	非保留
default	保留
defaults	非保留
deferrable	保留
deferred	非保留
definer	非保留
delay_key_write	非保留
delayed	非保留
delete	非保留
delimiter	非保留
delimiters	非保留

关键字	类别
delta	非保留
deltamerge	保留 (可用于函数、类型名)
desc	保留
describe	非保留
deterministic	非保留
diagnostics	非保留
dictionary	非保留
direct	非保留
directory	非保留
disable	非保留
discard	非保留
disconnect	非保留
disk	非保留
distinct	保留
distinctrow	保留
distribute	非保留
distribution	非保留
div	非保留
do	保留
document	非保留
domain	非保留
double	非保留
double_compatible	非保留 (不可用于函数、类型名)
drop	非保留
dumpfile	非保留
duplicate	非保留
each	非保留
elastic	非保留

关键字	类别
else	保留
enable	非保留
enclosed	非保留
encoding	非保留
encrypted	非保留
encrypted_value	非保留
encryption	非保留
encryption_type	非保留
end	保留
ends	非保留
enforced	非保留
engine	非保留
engine_attribute	非保留
enum	保留
eol	非保留
error	非保留
errors	非保留
escape	非保留
escaped	非保留
escaping	非保留
event	非保留
events	非保留
every	非保留
except	保留
exchange	非保留
exclude	非保留
excluded	保留
excluding	非保留

关键字	类别
exclusive	非保留
exec	非保留
execute	非保留
exists	非保留 (不可用于函数、类型名)
expansion	非保留
expired	非保留
explain	非保留
extended	非保留
extension	非保留
external	非保留
extract	非保留 (不可用于函数、类型名)
false	保留
family	非保留
fast	非保留
features	非保留
fenced	非保留
fetch	保留
fields	非保留
fileheader	非保留
fill_missing_fields	非保留
filler	非保留
filter	非保留
first	保留
fixed	非保留 (不可用于函数、类型名)
flashback	保留
float	非保留 (不可用于函数、类型名)
float8	非保留 (不可用于函数、类型名)
flush	非保留

关键字	类别
following	非保留
follows	非保留
for	保留
force	非保留
foreign	保留
format	非保留 (不可用于函数、类型名)
formatter	非保留
forward	非保留
freeze	保留 (可用于函数、类型名)
from	保留
full	非保留
fulltext	非保留
function	非保留
functions	非保留
generated	非保留
get	非保留
get_format	非保留 (不可用于函数、类型名)
global	非保留
globally	非保留
grant	保留
granted	非保留
grants	非保留
greatest	非保留 (不可用于函数、类型名)
group	保留
group_concat	非保留
grouping	非保留 (不可用于函数、类型名)
grouping_id	非保留 (不可用于函数、类型名)
groupparent	保留

关键字	类别
handler	非保留
having	保留
hdfsdirectory	保留 (可用于函数、类型名)
header	非保留
hold	非保留
hour	非保留
hour_microsecond	非保留
hour_minute	非保留
hour_second	非保留
identified	非保留
identity	非保留
if	非保留 (不可用于函数、类型名)
ifnull	非保留 (不可用于函数、类型名)
ignore	非保留
ignore_extra_data	非保留
ilike	保留 (可用于函数、类型名)
image	保留 (可用于函数、类型名)
immediate	非保留
immutable	非保留
implicit	非保留
in	非保留
include	非保留
including	非保留
increment	非保留
incremental	非保留
indexes	非保留
infile	非保留
inherit	非保留

关键字	类别
inherits	非保留
initial	非保留
initially	保留
initrans	非保留
inline	非保留
inner	保留 (可用于函数、类型名)
inout	非保留 (不可用于函数、类型名)
inplace	非保留
input	非保留
insensitive	非保留
insert	非保留
insert_method	非保留
instead	非保留
int	非保留
int2	非保留
int4	非保留
int8	非保留
integer	非保留 (不可用于函数、类型名)
internal	非保留
intersect	保留
interval	非保留 (不可用于函数、类型名)
into	保留
invisible	非保留
invoker	非保留
ip	非保留
is	保留
isnull	非保留
isolation	非保留

关键字	类别
join	保留 (可用于函数、类型名)
json	非保留
json_mergepatch	保留
json_object	非保留 (不可用于函数、类型名)
jsonb	非保留
key	非保留
key_block_size	非保留
key_path	非保留
key_store	非保留
keys	非保留
kill	非保留
label	非保留
language	非保留
large	非保留
last	非保留
last_day	非保留
lc_collate	非保留
lc_ctype	非保留
leading	保留
leakproof	非保留
least	非保留 (不可用于函数、类型名)
left	保留 (可用于函数、类型名)
less	保留
level	非保留
like	保留 (可用于函数、类型名)
limit	保留
lines	非保留
link	非保留

关键字	类别
list	非保留
listen	非保留
load	非保留
local	非保留
localtime	保留
localtimestamp	保留
location	非保留
lock	非保留
locked	非保留
log	非保留
logging	非保留
logical_data	保留
logical_file	保留
login_any	非保留
login_failure	非保留
login_success	非保留
logout	非保留
logs	非保留
long	非保留
longtext	非保留
loop	非保留
low_priority	非保留
lseg	非保留
mapping	非保留
masking	非保留
master	非保留
match	非保留
matched	非保留

关键字	类别
materialized	非保留
max_rows	非保留
maxextents	非保留
maxsize	非保留
maxtrans	非保留
maxvalue	保留
mediumint	非保留 (不可用于函数、类型名)
memory	非保留
merge	非保留
message_text	非保留
microsecond	非保留
mid	非保留 (不可用于函数、类型名)
min_rows	非保留
minextents	非保留
minus	保留
minute	非保留
minute_microsecond	非保留
minute_second	非保留
minvalue	非保留
mod	非保留
mode	非保留
model	非保留
modifies	非保留
money	非保留
modify	保留
month	非保留
move	非保留
movement	非保留

关键字	类别
mysql_errno	非保留
name	非保留
names	非保留
national	非保留 (不可用于函数、类型名)
natural	非保留
nchar	非保留 (不可用于函数、类型名)
next	非保留
ngram	非保留
no	非保留
no_write_to_binlog	非保留
nocache	非保留
nocompress	非保留
nocycle	保留
node	非保留
noforce	非保留
nologging	非保留
nomaxvalue	非保留
nominvalue	非保留
none	非保留 (不可用于函数、类型名)
noorder	非保留
normally	非保留
not	保留
nothing	非保留
notify	非保留
notnull	非保留
nowait	非保留
null	保留
nullcols	非保留

关键字	类别
nullif	非保留 (不可用于函数、类型名)
nulls	非保留
number	非保留 (不可用于函数、类型名)
numeric	非保留 (不可用于函数、类型名)
numstr	非保留
nvarchar	非保留 (不可用于函数、类型名)
nvarchar2	非保留 (不可用于函数、类型名)
nvl	非保留 (不可用于函数、类型名)
nvl2	非保留 (不可用于函数、类型名)
object	非保留
of	非保留
off	保留
offset	保留
oids	非保留
on	保留
only	保留
operator	非保留
optimization	非保留
optimize	非保留
option	非保留
optionally	非保留
options	非保留
or	保留
ora_hash	非保留 (不可用于函数、类型名)
order	保留
out	非保留 (不可用于函数、类型名)
outer	非保留
outfile	非保留

关键字	类别
output	保留
over	非保留
overlaps	非保留
overlay	非保留 (不可用于函数、类型名)
owned	非保留
owner	非保留
pack_keys	非保留
package	非保留
packages	非保留
pad_attribute	非保留
parallel	非保留
parser	非保留
partial	非保留
partition	非保留
partitioning	非保留
partitions	非保留
passing	非保留
password	非保留
path	非保留
pctfree	非保留
per	非保留
percent	非保留
performance	保留
perm	非保留
physical_only	保留
pipelined	非保留
pivot	非保留
placing	保留

关键字	类别
plan	非保留
plans	非保留
point	非保留
policy	非保留
polygon	非保留
pool	非保留
position	非保留 (不可用于函数、类型名)
precedes	非保留
preceding	非保留
precision	非保留 (不可用于函数、类型名)
predict	非保留
preferred	非保留
prefix	非保留
prepare	非保留
prepared	非保留
preserve	非保留
primary	保留
prior	非保留
priorer	非保留
private	非保留
privilege	非保留
privileges	非保留
procedural	非保留
procedure	保留
processlist	非保留
profile	非保留
proxy	非保留
public	非保留

关键字	类别
publication	非保留
publish	非保留
purge	非保留
quarter	非保留
query	非保留
quick	非保留
quote	非保留
randomized	非保留
range	非保留
ratio	非保留
raw	非保留
read	非保留
reads	非保留
real	非保留 (不可用于函数、类型名)
reassign	非保留
rebuild	非保留
recheck	非保留
recursive	非保留
recyclebin	非保留
redisanyvalue	非保留
ref	非保留
references	保留
refresh	非保留
regexp	保留 (可用于函数、类型名)
reindex	非保留
reject	保留
relative	非保留
release	非保留

关键字	类别
reloptions	非保留
reltime	非保留
remote	非保留
remove	非保留
rename	非保留
reorganize	非保留
repair	非保留
repeat	非保留
repeatable	非保留
replace	非保留
replica	非保留
reserve	非保留
reset	非保留
resize	非保留
resource	非保留
restart	非保留
restrict	非保留
return	保留
returned_sqlstate	非保留
returning	保留
returns	非保留
reuse	非保留
revoke	非保留
right	非保留
rlike	非保留
role	非保留
roles	非保留
rollback	非保留

关键字	类别
rollup	非保留
rotation	非保留
routine	非保留
row	非保留 (不可用于函数、类型名)
row_count	非保留
row_format	非保留
rowid	非保留
rows	非保留
rowtype	非保留
rule	非保留
sample	保留 (可用于函数、类型名)
savepoint	非保留
schedule	非保留
schema	非保留
schema_name	非保留
score	非保留
scroll	非保留
search	非保留
second	非保留
second_microsecond	非保留
secondary_engine_attribute	非保留
security	非保留
seed	非保留
select	保留
sensitive	非保留
separator	保留
sequence	非保留
sequences	非保留

关键字	类别
serializable	非保留
server	非保留
session	非保留
session_user	保留
sessiontimezone	保留
set	非保留
setof	非保留 (不可用于函数、类型名)
sets	非保留
share	非保留
shink	保留
shippable	非保留
show	非保留
shutdown	非保留
siblings	非保留
signed	非保留 (不可用于函数、类型名)
similar	保留 (可用于函数、类型名)
simple	非保留
size	非保留
skip	非保留
slave	非保留
slice	非保留
smalldatetime	非保留 (不可用于函数、类型名)
smalldatetime_format	非保留
smallint	非保留 (不可用于函数、类型名)
smallmoney	非保留
snapshot	非保留
some	保留
sounds	保留 (可用于函数、类型名)

关键字	类别
source	非保留
space	非保留
spill	非保留
split	非保留
sql	非保留
stable	非保留
stacked	非保留
standalone	非保留
start	非保留
starting	非保留
starts	非保留
statement	非保留
statement_id	非保留
statistics	非保留
stats_auto_recalc	非保留
stats_persistent	非保留
stats_sample_pages	非保留
status	非保留
stdin	非保留
stdout	非保留
storage	非保留
store	非保留
stored	非保留
stratify	非保留
stream	非保留
strict	非保留
strip	非保留
subclass_origin	非保留

关键字	类别
subpartition	非保留
subpartitions	非保留
subscription	非保留
substr	非保留 (不可用于函数、类型名)
substring	非保留 (不可用于函数、类型名)
symmetric	保留
synonym	非保留
sys_refcursor	非保留
sysdate	保留
sysid	非保留
system	非保留
systimestamp	保留
table	保留
table_name	非保留
tables	非保留
tablesample	非保留
tablespace	非保留
target	非保留
temp	非保留
template	非保留
temporary	非保留
temptable	非保留
terminated	非保留
text	非保留 (不可用于函数、类型名)
than	非保留
then	保留
threshold	非保留
time	非保留 (不可用于函数、类型名)

关键字	类别
time_format	非保留
time_period	非保留
timecapsule	非保留
timestamp	非保留 (不可用于函数、类型名)
timestamp_format	非保留
timestampadd	非保留 (不可用于函数、类型名)
timestampdiff	非保留 (不可用于函数、类型名)
timestamptz	非保留
timetz	非保留
tinterval	非保留
tinyint	非保留 (不可用于函数、类型名)
tinytext	非保留
to	保留
trailing	保留
transaction	非保留
transform	非保留
treat	非保留 (不可用于函数、类型名)
trigger	非保留
trim	非保留 (不可用于函数、类型名)
true	保留
truncate	非保留
trusted	非保留
tsfield	非保留
tstag	非保留
tstime	非保留
txid_snapshot	非保留
type	非保留
types	非保留

关键字	类别
unbounded	非保留
uncommitted	非保留
undefined	非保留
unencrypted	非保留
union	保留
unique	保留
unknown	非保留
unlimited	非保留
unlisten	非保留
unlock	非保留
unlogged	非保留
unpivot	非保留
unsigned	非保留 (不可用于函数、类型名)
until	非保留
unusable	非保留
update	非保留
use	非保留
useeof	非保留
user	非保留（是否能作为关键字由 b_format_behavior_compat_options 控制）
using	保留
utc_date	保留
utc_time	保留
utc_timestamp	保留
vacuum	非保留
valid	非保留
validate	非保留

关键字	类别
validation	非保留
validator	非保留
value	非保留
values	非保留
varbinary	非保留
varchar	非保留 (不可用于函数、类型名)
varchar2	非保留 (不可用于函数、类型名)
variables	非保留
variadic	保留
varying	非保留
pw_hash	非保留 (不可用于函数、类型名)
vcgroup	非保留
verbose	保留 (可用于函数、类型名)
verify	保留
version	非保留
view	非保留
visible	非保留
volatile	非保留
wait	非保留
warning	非保留
warnings	非保留
weak	非保留
when	保留
where	保留
while	非保留
whitespace	非保留
window	保留
with	保留

关键字	类别
within	非保留
without	非保留
work	非保留
workload	非保留
wrapper	非保留
write	非保留
xml	非保留
xmlattributes	非保留 (不可用于函数、类型名)
xmlconcat	非保留 (不可用于函数、类型名)
xmlelement	非保留 (不可用于函数、类型名)
xmlexists	非保留 (不可用于函数、类型名)
xmlforest	非保留 (不可用于函数、类型名)
xmlparse	非保留 (不可用于函数、类型名)
xmlpi	非保留 (不可用于函数、类型名)
xmlroot	非保留 (不可用于函数、类型名)
xmlserialize	非保留 (不可用于函数、类型名)
xor	非保留
year	非保留
year_month	非保留
yes	非保留
zerofill	非保留
zone	非保留

表 2 PL/pgSQL 保留关键字

关键字	类别
absolute	非保留
alias	非保留
all	保留

关键字	类别
alter	非保留
array	非保留
as	非保留
backward	非保留
begin	保留
bulk	非保留
by	保留
call	保留
case	保留
catalog_name	非保留
class_origin	非保留
close	保留
collate	保留
collect	非保留
column_name	非保留
commit	非保留
condition	非保留
constant	非保留
constraint_catalog	非保留
constraint_name	非保留
constraint_schema	非保留
continue	非保留
current	非保留
cursor	非保留
cursor_name	非保留
debug	非保留
declare	保留
default	保留
delete	保留
detail	非保留
diagnostics	保留
distinct	非保留

关键字	类别
do	非保留
dump	非保留
else	保留
elseif	保留
elsif	保留
end	保留
errcode	非保留
error	非保留
except	非保留
exception	保留
exceptions	非保留
execute	保留
exit	保留
fetch	保留
first	非保留
for	保留
forall	保留
foreach	保留
forward	非保留
found	非保留
from	保留
function	非保留
get	保留
goto	保留
handler	非保留
hint	非保留
if	保留
ignore	非保留
immediate	非保留
in	保留
index	保留
index	非保留

关键字	类别
info	非保留
insert	保留
instantiation	非保留
intersect	非保留
into	保留
is	非保留
iterate	非保留
last	非保留
leave	非保留
limit	保留
log	非保留
loop	保留
merge	非保留
message	非保留
message_text	非保留
move	保留
multiset	非保留
mysql_errno	非保留
next	非保留
no	非保留
not	保留
notice	非保留
null	保留
number	非保留
of	保留
open	保留
option	非保留
or	保留
out	保留
package	非保留
perform	非保留
pg_exception_context	非保留

关键字	类别
pg_exception_detail	非保留
pg_exception_hint	非保留
pipe	非保留
pls_integer	非保留
pragma	非保留
print_strict_params	非保留
prior	非保留
procedure	保留
query	非保留
raise	保留
record	非保留
ref	保留
relative	非保留
release	非保留
repeat	非保留
replace	非保留
resignal	非保留
result_oid	非保留
return	保留
returned_sqlstate	非保留
reverse	非保留
rollback	非保留
row	非保留
row_count	非保留
rowtype	非保留
save	非保留
savepoint	非保留
schema_name	非保留
scroll	非保留
select	保留
set	非保留
signal	非保留

关键字	类别
slice	非保留
sqlexception	非保留
sqlstate	非保留
sqlwarning	非保留
stacked	非保留
start	非保留
strict	保留
subclass_origin	非保留
sys_refcursor	非保留
table	非保留
table_name	非保留
then	保留
to	保留
transaction	非保留
type	非保留
union	非保留
until	非保留
update	保留
updating	保留
use_column	非保留
use_variable	非保留
using	保留
value	非保留
values	非保留
variable_conflict	非保留
varray	非保留
warning	非保留
when	保留
while	保留
with	非保留

4.1.1.3.PostgreSQL 兼容模式关键字

在 PanWeiDB 的 PostgreSQL 兼容模式下，关键字及其类别的清单如表 1（SQL 关键字）和表 2（PL/pgSQL 关键字）所示。

表 1 SQL 关键字

关键字	类别
abort	非保留
absolute	非保留
access	非保留
accessed	非保留
account	非保留
action	非保留
add	非保留
admin	非保留
after	非保留
aggregate	非保留
algorithm	非保留
all	保留
also	非保留
alter	非保留
always	非保留
analyse	保留
analyze	保留
and	保留
any	保留
app	非保留
append	非保留
archive	非保留
array	保留
as	保留
asc	保留
assertion	非保留
assignment	非保留

关键字	类别
asymmetric	保留
at	非保留
attach	非保留
attribute	非保留
audit	非保留
authid	非保留 (不可用于函数、类型名)
authorization	保留 (可用于函数、类型名)
auto_increment	保留
autoextend	非保留
automapped	非保留
backward	非保留
barrier	非保留
before	非保留
begin	非保留
begin_non_anoyblock	非保留
between	非保留 (不可用于函数、类型名)
bfile	非保留
bigint	非保留 (不可用于函数、类型名)
binary	保留 (可用于函数、类型名)
binary_double	非保留 (不可用于函数、类型名)
binary_float	非保留 (不可用于函数、类型名)
binary_integer	非保留 (不可用于函数、类型名)
bit	非保留 (不可用于函数、类型名)
blanks	非保留
blob	非保留
block	非保留
blockchain	非保留
body	保留
boolean	非保留 (不可用于函数、类型名)
both	保留
bucketcnt	非保留 (不可用于函数、类型名)
buckets	保留

关键字	类别
by	非保留
byte	非保留 (不可用于函数、类型名)
byteawithoutorder	非保留 (不可用于函数、类型名)
byteawithoutorderwithequal	非保留 (不可用于函数、类型名)
cache	非保留
call	非保留
called	非保留
cancelable	非保留
cascade	非保留
cascaded	非保留
case	保留
cast	保留
catalog	非保留
chain	非保留
change	非保留
char	非保留 (不可用于函数、类型名)
character	非保留 (不可用于函数、类型名)
characteristics	非保留
characterset	非保留
charset	非保留
check	保留
checkpoint	非保留
checktable	保留
class	非保留
clean	非保留
client	非保留
client_master_key	非保留
client_master_keys	非保留
clob	非保留
close	非保留
cluster	非保留
coalesce	非保留 (不可用于函数、类型名)

关键字	类别
collate	保留
collation	保留 (可用于函数、类型名)
column	保留
column_encryption_key	非保留
column_encryption_keys	非保留
columns	非保留
comment	非保留
comments	非保留
commit	非保留
committed	非保留
compact	保留 (可用于函数、类型名)
compatible_illegal_chars	非保留
complete	非保留
completion	非保留
compress	非保留
concurrently	保留 (可用于函数、类型名)
condition	非保留
configuration	非保留
conflict	非保留
connect	非保留
connect_by_root	非保留
connection	非保留
constant	非保留
constraint	保留
constraints	非保留
contains	非保留
content	非保留
context	非保留
continue	非保留
contview	非保留
conversion	非保留
convert	非保留 (不可用于函数、类型名)

关键字	类别
coordinator	非保留
coordinators	非保留
copy	非保留
cost	非保留
create	保留
cross	保留 (可用于函数、类型名)
csn	保留 (可用于函数、类型名)
csv	非保留
cube	非保留
current	非保留
current_catalog	保留
current_date	保留
current_role	保留
current_schema	保留 (可用于函数、类型名)
current_time	保留
current_timestamp	保留
current_user	保留
cursor	非保留
cycle	非保留
data	非保留
database	非保留
datafile	非保留
datanode	非保留
datanodes	非保留
datatype_cl	非保留
date	非保留 (不可用于函数、类型名)
date_add	非保留 (不可用于函数、类型名)
date_format	非保留
datetime	非保留
datetime2	非保留
day	非保留
dbcc	保留

关键字	类别
dbcompatibility	非保留
dblink	非保留
dbname	非保留
dbtimezone	保留
deallocate	非保留
dec	非保留 (不可用于函数、类型名)
decimal	非保留 (不可用于函数、类型名)
declare	非保留
decode	非保留 (不可用于函数、类型名)
default	保留
defaults	非保留
deferrable	保留
deferred	非保留
definer	非保留
delayed	非保留
delete	非保留
delimiter	非保留
delimiters	非保留
delta	非保留
deltamerge	保留 (可用于函数、类型名)
desc	保留
deterministic	非保留
dictionary	非保留
direct	非保留
directory	非保留
disable	非保留
discard	非保留
disconnect	非保留
distinct	保留
distribute	非保留
distribution	非保留
do	保留

关键字	类别
document	非保留
domain	非保留
double	非保留
double_compatible	非保留 (不可用于函数、类型名)
drop	非保留
dumpfile	非保留
duplicate	非保留
each	非保留
elastic	非保留
else	保留
enable	非保留
enclosed	非保留
encoding	非保留
encrypted	非保留
encrypted_value	非保留
encryption	非保留
encryption_type	非保留
end	保留
ends	非保留
enforced	非保留
enum	非保留
eol	非保留
error	非保留
errors	非保留
escape	非保留
escaped	非保留
escaping	非保留
event	非保留
events	非保留
every	非保留
except	保留
exchange	非保留

关键字	类别
exclude	非保留
excluded	保留
excluding	非保留
exclusive	非保留
exec	非保留
execute	非保留
exists	非保留 (不可用于函数、类型名)
expired	非保留
explain	非保留
extension	非保留
external	非保留
extract	非保留 (不可用于函数、类型名)
FALSE	保留
family	非保留
fast	非保留
features	非保留
fenced	保留
fetch	保留
fields	非保留
fileheader	非保留
fill_missing_fields	非保留
filler	非保留
filter	非保留
first	非保留
fixed	非保留
flashback	保留
float	非保留 (不可用于函数、类型名)
following	非保留
follows	非保留
for	保留
force	非保留
foreign	保留

关键字	类别
formatter	非保留
forward	非保留
freeze	保留 (可用于函数、类型名)
from	保留
full	保留 (可用于函数、类型名)
function	非保留
functions	非保留
generated	非保留
global	非保留
globally	非保留
grant	保留
granted	非保留
greatest	非保留 (不可用于函数、类型名)
group	保留
grouping	非保留 (不可用于函数、类型名)
grouping_id	非保留 (不可用于函数、类型名)
groupparent	保留
handler	非保留
having	保留
hdfsdirectory	保留 (可用于函数、类型名)
header	非保留
hold	非保留
hour	非保留
identified	非保留
identity	非保留
if	非保留
ignore_extra_data	非保留
ilike	保留 (可用于函数、类型名)
image	保留 (可用于函数、类型名)
immediate	非保留
immutable	非保留
implicit	非保留

关键字	类别
import	非保留
in	保留
include	非保留
including	非保留
increment	非保留
incremental	非保留
index	非保留
indexes	非保留
infile	非保留
inherit	非保留
inherits	非保留
initial	非保留
initially	保留
initrans	非保留
inline	非保留
inner	保留 (可用于函数、类型名)
inout	非保留 (不可用于函数、类型名)
input	非保留
insensitive	非保留
insert	非保留
instead	非保留
int	非保留 (不可用于函数、类型名)
integer	非保留 (不可用于函数、类型名)
internal	非保留
intersect	保留
interval	非保留 (不可用于函数、类型名)
into	保留
invisible	非保留
invoker	非保留
ip	非保留
is	保留
isnull	非保留

关键字	类别
isolation	非保留
join	保留 (可用于函数、类型名)
json_mergepatch	保留
key	非保留
key_path	非保留
key_store	非保留
kill	非保留
label	非保留
language	非保留
large	非保留
last	非保留
lc_collate	非保留
lc_ctype	非保留
leading	保留
leakproof	非保留
least	非保留 (不可用于函数、类型名)
left	保留 (可用于函数、类型名)
less	保留
level	非保留
like	保留 (可用于函数、类型名)
limit	保留
lines	非保留
link	非保留
list	非保留
listen	非保留
load	非保留
local	非保留
localtime	保留
localtimestamp	保留
location	非保留
lock	非保留
locked	非保留

关键字	类别
log	非保留
logging	非保留
logical_data	保留
logical_file	保留
login_any	非保留
login_failure	非保留
login_success	非保留
logout	非保留
long	非保留
loop	非保留
low_priority	非保留
mapping	非保留
masking	非保留
master	非保留
match	非保留
matched	非保留
materialized	非保留
maxextents	非保留
maxsize	非保留
maxtrans	非保留
maxvalue	保留
merge	非保留
minextents	非保留
minus	保留
minute	非保留
minvalue	非保留
mode	非保留
model	非保留
modify	保留
month	非保留
move	非保留
movement	非保留

关键字	类别
name	非保留
names	非保留
national	非保留 (不可用于函数、类型名)
natural	保留 (可用于函数、类型名)
nchar	非保留 (不可用于函数、类型名)
next	非保留
no	非保留
nocache	非保留
nocompress	非保留
nocycle	保留
node	非保留
noforce	非保留
nologging	非保留
nomaxvalue	非保留
nominvalue	非保留
none	非保留 (不可用于函数、类型名)
noorder	非保留
normally	非保留
not	保留
nothing	非保留
notify	非保留
notnull	保留 (可用于函数、类型名)
nowait	非保留
null	保留
nullcols	非保留
nullif	非保留 (不可用于函数、类型名)
nulls	非保留
number	非保留 (不可用于函数、类型名)
numeric	非保留 (不可用于函数、类型名)
numstr	非保留
nvarchar	非保留 (不可用于函数、类型名)
nvarchar2	非保留 (不可用于函数、类型名)

关键字	类别
nvl	非保留 (不可用于函数、类型名)
nvl2	非保留 (不可用于函数、类型名)
object	非保留
of	非保留
off	保留
offset	保留
oids	非保留
on	保留
only	保留
operator	非保留
optimization	非保留
option	非保留
optionally	非保留
options	非保留
or	保留
ora_hash	非保留 (不可用于函数、类型名)
order	保留
out	非保留 (不可用于函数、类型名)
outer	保留 (可用于函数、类型名)
outfile	非保留
output	保留
over	非保留
overlaps	保留 (可用于函数、类型名)
overlay	非保留 (不可用于函数、类型名)
overriding	非保留
owned	非保留
owner	非保留
package	非保留
packages	非保留
pad_attribute	非保留
parallel	非保留
parser	非保留

关键字	类别
partial	非保留
partition	非保留
partitions	非保留
passing	非保留
password	非保留
pctfree	非保留
per	非保留
percent	非保留
performance	保留
perm	非保留
physical_only	保留
pipelined	非保留
pivot	非保留
placing	保留
plan	非保留
plans	非保留
policy	非保留
pool	非保留
position	非保留 (不可用于函数、类型名)
precedes	非保留
preceding	非保留
precision	非保留 (不可用于函数、类型名)
predict	非保留
preferred	非保留
prefix	非保留
prepare	非保留
prepared	非保留
preserve	非保留
primary	保留
prior	非保留
priorer	保留
private	非保留

关键字	类别
privilege	非保留
privileges	非保留
procedural	非保留
procedure	保留
profile	非保留
public	非保留
publication	非保留
publish	非保留
purge	非保留
query	非保留
quote	非保留
randomized	非保留
range	非保留
ratio	非保留
raw	非保留
read	非保留
real	非保留 (不可用于函数、类型名)
reassign	非保留
rebuild	非保留
recheck	非保留
recursive	非保留
recyclebin	保留 (可用于函数、类型名)
redisanyvalue	非保留
ref	非保留
references	保留
refresh	非保留
reindex	非保留
reject	保留
relative	非保留
release	非保留
reloptions	非保留
remote	非保留

关键字	类别
remove	非保留
rename	非保留
repeat	非保留
repeatable	非保留
replace	非保留
replica	非保留
reserve	非保留
reset	非保留
resize	非保留
resource	非保留
restart	非保留
restrict	非保留
return	保留
returning	保留
returns	非保留
reuse	非保留
revoke	非保留
right	保留 (可用于函数、类型名)
role	非保留
roles	非保留
rollback	非保留
rollup	非保留
rotation	非保留
row	非保留 (不可用于函数、类型名)
rowid	非保留
rownum	保留
rows	非保留
rowtype	非保留
rule	非保留
sample	保留 (可用于函数、类型名)
savepoint	非保留
schedule	非保留

关键字	类别
schema	非保留
score	非保留
scroll	非保留
search	非保留
second	非保留
security	非保留
seed	非保留
select	保留
sensitive	非保留
sequence	非保留
sequences	非保留
serializable	非保留
server	非保留
session	非保留
session_user	保留
sessiontimezone	保留
set	非保留
setof	非保留 (不可用于函数、类型名)
sets	非保留
share	非保留
shippable	非保留
show	非保留
shutdown	非保留
siblings	非保留
similar	保留 (可用于函数、类型名)
simple	非保留
size	非保留
skip	非保留
slave	非保留
slice	非保留
smalldatetime	非保留 (不可用于函数、类型名)
smalldatetime_format	非保留

关键字	类别
smallint	非保留 (不可用于函数、类型名)
smallmoney	非保留
snapshot	非保留
some	保留
source	非保留
space	非保留
spill	非保留
split	非保留
sql	非保留
stable	非保留
standalone	非保留
start	非保留
starting	非保留
starts	非保留
statement	非保留
statement_id	非保留
statistics	非保留
stdin	非保留
stdout	非保留
storage	非保留
store	非保留
stored	非保留
stratify	非保留
stream	非保留
strict	非保留
strip	非保留
subpartition	非保留
subpartitions	非保留
subscription	非保留
substring	非保留 (不可用于函数、类型名)
symmetric	保留
synonym	非保留

关键字	类别
sys_refcursor	非保留
sysdate	保留
sysid	非保留
system	非保留
systimestamp	非保留
table	保留
tables	非保留
tablesample	保留 (可用于函数、类型名)
tablespace	非保留
target	非保留
temp	非保留
template	非保留
temporary	非保留
terminated	非保留
text	非保留
than	非保留
then	保留
threshold	非保留
time	非保留 (不可用于函数、类型名)
time_format	非保留
time_period	非保留
timecapsule	保留 (可用于函数、类型名)
timestamp	非保留 (不可用于函数、类型名)
timestamp_format	非保留
timestampdiff	非保留 (不可用于函数、类型名)
tinyint	非保留 (不可用于函数、类型名)
to	保留
trailing	保留
transaction	非保留
transform	非保留
treat	非保留 (不可用于函数、类型名)
trigger	非保留

关键字	类别
trim	非保留 (不可用于函数、类型名)
TRUE	保留
truncate	非保留
trusted	非保留
tsfield	非保留
tstag	非保留
tstime	非保留
type	非保留
types	非保留
unbounded	非保留
uncommitted	非保留
unencrypted	非保留
union	保留
unique	保留
unknown	非保留
unlimited	非保留
unlisten	非保留
unlock	非保留
unlogged	非保留
unpivot	非保留
until	非保留
unusable	非保留
update	非保留
useeof	非保留
user	保留
using	保留
vacuum	非保留
valid	非保留
validate	非保留
validation	非保留
validator	非保留
value	非保留

关键字	类别
values	非保留 (不可用于函数、类型名)
varbinary	保留 (可用于函数、类型名)
varchar	非保留 (不可用于函数、类型名)
varchar2	非保留 (不可用于函数、类型名)
variables	非保留
variadic	保留
varying	非保留
pw_hash	非保留 (不可用于函数、类型名)
vcgroup	非保留
verbose	保留 (可用于函数、类型名)
verify	保留
version	非保留
view	非保留
visible	非保留
volatile	非保留
wait	非保留
warning	非保留
warnings	非保留
weak	非保留
when	保留
where	保留
while	非保留
whitespace	非保留
window	保留
with	保留
within	非保留
without	非保留
work	非保留
workload	非保留
wrapper	非保留
write	非保留
xml	非保留

关键字	类别
xmlattributes	非保留 (不可用于函数、类型名)
xmlconcat	非保留 (不可用于函数、类型名)
xmlelement	非保留 (不可用于函数、类型名)
xmlexists	非保留 (不可用于函数、类型名)
xmlforest	非保留 (不可用于函数、类型名)
xmlparse	非保留 (不可用于函数、类型名)
xmlpi	非保留 (不可用于函数、类型名)
xmlroot	非保留 (不可用于函数、类型名)
xmlserialize	非保留 (不可用于函数、类型名)
year	非保留
yes	非保留
zone	非保留

表 2 PL/pgSQL 保留关键字

关键字	类别
absolute	非保留
alias	非保留
all	保留
alter	非保留
array	非保留
as	非保留
backward	非保留
begin	保留
bulk	非保留
by	保留
call	保留
case	保留
close	保留
collate	保留
collect	非保留
commit	非保留

关键字	类别
constant	非保留
continue	非保留
current	非保留
cursor	非保留
debug	非保留
declare	保留
default	保留
delete	保留
detail	非保留
diagnostics	保留
distinct	非保留
do	非保留
dump	非保留
else	保留
elseif	保留
elsif	保留
end	保留
errcode	非保留
error	非保留
except	非保留
exception	保留
exceptions	非保留
execute	保留
exit	保留
fetch	保留
first	非保留
for	保留
forall	保留
foreach	保留
forward	非保留
from	保留
function	非保留

关键字	类别
get	保留
goto	保留
hint	非保留
if	保留
immediate	非保留
in	保留
index	非保留
info	非保留
insert	保留
instantiation	非保留
intersect	非保留
into	保留
is	非保留
iterate	非保留
last	非保留
leave	非保留
limit	保留
log	非保留
loop	保留
merge	非保留
message	非保留
message_text	非保留
move	保留
multiset	非保留
next	非保留
no	非保留
not	保留
notice	非保留
null	保留
of	保留
open	保留
option	非保留

关键字	类别
or	保留
out	保留
package	非保留
perform	非保留
pg_exception_context	非保留
pg_exception_detail	非保留
pg_exception_hint	非保留
pipe	非保留
pls_integer	非保留
pragma	非保留
print_strict_params	非保留
prior	非保留
procedure	保留
query	非保留
raise	保留
record	非保留
ref	保留
relative	非保留
release	非保留
repeat	非保留
result_oid	非保留
return	保留
returned_sqlstate	非保留
reverse	非保留
rollback	非保留
row	非保留
row_count	非保留
rowtype	非保留
save	非保留
savepoint	非保留
scroll	非保留
select	保留

关键字	类别
slice	非保留
sqlstate	非保留
stacked	非保留
strict	保留
sys_refcursor	非保留
table	非保留
then	保留
to	保留
type	非保留
union	非保留
until	非保留
update	保留
updating	保留
use_column	非保留
use_variable	非保留
using	保留
values	非保留
variable_conflict	非保留
varray	非保留
warning	非保留
when	保留
while	保留
with	非保留

4.1.1.4.SQL Server 兼容模式关键字

在 PanWeiDB 的 SQL Server 兼容模式下，关键字及其类别的清单如表 1（SQL 关键字）和表 2（PL/pgSQL 关键字）所示。

表 1 SQL 关键字

关键字	类别
abort	非保留

关键字	类别
absolute	非保留
access	非保留
accessed	非保留
account	非保留
action	非保留
add	非保留
admin	非保留
after	非保留
aggregate	非保留
algorithm	非保留
all	保留
also	非保留
alter	非保留
always	非保留
analyse	保留
analyze	保留
and	保留
any	保留
app	非保留
append	非保留
apply	保留
archive	非保留
array	保留
as	保留
asc	保留
assertion	非保留
assignment	非保留
asymmetric	保留
at	非保留
attribute	非保留
audit	非保留
authid	非保留 (不可用于函数、类型名)

关键字	类别
authorization	保留 (可用于函数、类型名)
auto_increment	保留
autoextend	非保留
automapped	非保留
backward	非保留
barrier	非保留
before	非保留
begin	非保留
begin_non_anoyblock	非保留
between	非保留 (不可用于函数、类型名)
bfile	非保留
bigint	非保留 (不可用于函数、类型名)
binary	保留 (可用于函数、类型名)
binary_double	非保留 (不可用于函数、类型名)
binary_float	非保留 (不可用于函数、类型名)
binary_integer	非保留 (不可用于函数、类型名)
bit	非保留 (不可用于函数、类型名)
blanks	非保留
blob	非保留
block	非保留
blockchain	非保留
body	保留
boolean	非保留 (不可用于函数、类型名)
both	保留
bucketcnt	非保留 (不可用于函数、类型名)
buckets	保留
by	非保留
byte	非保留 (不可用于函数、类型名)
byteawithoutorder	非保留 (不可用于函数、类型名)
byteawithoutorderwithequal	非保留 (不可用于函数、类型名)
cache	非保留
call	非保留

关键字	类别
called	非保留
cancelable	非保留
cascade	非保留
cascaded	非保留
case	保留
cast	保留
catalog	非保留
catch	非保留
chain	非保留
change	非保留
char	非保留 (不可用于函数、类型名)
character	非保留 (不可用于函数、类型名)
characteristics	非保留
characterset	非保留
charset	非保留
check	保留
checkpoint	非保留
checktable	保留
class	非保留
clean	非保留
client	非保留
client_master_key	非保留
client_master_keys	非保留
clob	非保留
close	非保留
cluster	非保留
coalesce	非保留 (不可用于函数、类型名)
collate	保留
collation	保留 (可用于函数、类型名)
column	保留
column_encryption_key	非保留
column_encryption_keys	非保留

关键字	类别
columns	非保留
comment	非保留
comments	非保留
commit	非保留
committed	非保留
compact	保留 (可用于函数、类型名)
compatible_illegal_chars	非保留
complete	非保留
completion	非保留
compress	非保留
concurrently	保留 (可用于函数、类型名)
condition	非保留
configuration	非保留
conflict	非保留
connect	非保留
connect_by_root	非保留
connection	非保留
constant	非保留
constraint	保留
constraints	非保留
contains	非保留
content	非保留
context	非保留
continue	非保留
contview	非保留
conversion	非保留
convert	非保留 (不可用于函数、类型名)
coordinator	非保留
coordinators	非保留
copy	非保留
cost	非保留
create	保留

关键字	类别
cross	保留 (可用于函数、类型名)
csn	保留 (可用于函数、类型名)
csv	非保留
cube	非保留
current	非保留
current_catalog	保留
current_date	保留
current_role	保留
current_schema	保留 (可用于函数、类型名)
current_time	保留
current_timestamp	保留
current_user	保留
cursor	非保留
cycle	非保留
data	非保留
database	非保留
datafile	非保留
datanode	非保留
datanodes	非保留
datatype_cl	非保留
date	非保留 (不可用于函数、类型名)
date_add	非保留
date_format	非保留
datediff	非保留
datetime	非保留
datetime2	非保留
day	非保留
dbcc	保留
dbcompatibility	非保留
dblink	非保留
dbname	非保留
dbtimezone	保留

关键字	类别
deallocate	非保留
dec	非保留 (不可用于函数、类型名)
decimal	非保留 (不可用于函数、类型名)
declare	非保留
decode	非保留 (不可用于函数、类型名)
default	保留
defaults	非保留
deferrable	保留
deferred	非保留
definer	非保留
delayed	非保留
delete	非保留
delimiter	非保留
delimiters	非保留
delta	非保留
deltamerge	保留 (可用于函数、类型名)
desc	保留
deterministic	非保留
dictionary	非保留
direct	非保留
directory	非保留
disable	非保留
discard	非保留
disconnect	非保留
distinct	保留
distribute	非保留
distribution	非保留
do	保留
document	非保留
domain	非保留
double	非保留
double_compatible	非保留 (不可用于函数、类型名)

关键字	类别
drop	非保留
dumpfile	非保留
duplicate	非保留
each	非保留
elastic	非保留
else	保留
enable	非保留
enclosed	非保留
encoding	非保留
encrypted	非保留
encrypted_value	非保留
encryption	非保留
encryption_type	非保留
end	保留
ends	非保留
enforced	非保留
enum	非保留
eol	非保留
error	非保留
errors	非保留
escape	非保留
escaped	非保留
escaping	非保留
event	非保留
events	非保留
every	非保留
except	保留
exchange	非保留
exclude	非保留
excluded	保留
excluding	非保留
exclusive	非保留

关键字	类别
exec	非保留
execute	非保留
exists	非保留 (不可用于函数、类型名)
expired	非保留
explain	非保留
extension	非保留
external	非保留
extract	非保留 (不可用于函数、类型名)
FALSE	保留
family	非保留
fast	非保留
features	非保留
fenced	保留
fetch	保留
fields	非保留
fileheader	非保留
fill_missing_fields	非保留
filler	非保留
filter	非保留
first	非保留
fixed	非保留
flashback	保留
float	非保留 (不可用于函数、类型名)
following	非保留
follows	非保留
for	保留
force	非保留
foreign	保留
formatter	非保留
forward	非保留
freeze	保留 (可用于函数、类型名)
from	保留

关键字	类别
full	保留 (可用于函数、类型名)
function	非保留
functions	非保留
generated	非保留
global	非保留
globally	非保留
grant	保留
granted	非保留
greatest	非保留 (不可用于函数、类型名)
group	保留
grouping	非保留 (不可用于函数、类型名)
grouping_id	非保留 (不可用于函数、类型名)
groupparent	保留
handler	非保留
having	保留
hdfsdirectory	保留 (可用于函数、类型名)
header	非保留
hold	非保留
hour	非保留
identified	非保留
identity	保留
if	非保留
ignore_extra_data	非保留
ilike	保留 (可用于函数、类型名)
image	保留 (可用于函数、类型名)
immediate	非保留
immutable	非保留
implicit	非保留
in	保留
include	非保留
including	非保留
increment	非保留

关键字	类别
incremental	非保留
index	非保留
indexes	非保留
infile	非保留
inherit	非保留
inherits	非保留
initial	非保留
initially	保留
initrans	非保留
inline	非保留
inner	保留 (可用于函数、类型名)
inout	非保留 (不可用于函数、类型名)
input	非保留
insensitive	非保留
insert	非保留
instead	非保留
int	非保留 (不可用于函数、类型名)
integer	非保留 (不可用于函数、类型名)
internal	非保留
intersect	保留
interval	非保留 (不可用于函数、类型名)
into	保留
invisible	非保留
invoker	非保留
ip	非保留
is	保留
isnull	非保留
isolation	非保留
join	保留 (可用于函数、类型名)
json_mergepatch	保留
key	非保留
key_path	非保留

关键字	类别
key_store	非保留
kill	非保留
label	非保留
language	非保留
large	非保留
last	非保留
lc_collate	非保留
lc_ctype	非保留
leading	保留
leakproof	非保留
least	非保留 (不可用于函数、类型名)
left	保留 (可用于函数、类型名)
less	保留
level	非保留
like	保留 (可用于函数、类型名)
limit	保留
lines	非保留
link	非保留
list	非保留
listen	非保留
load	非保留
local	非保留
localtime	保留
localtimestamp	保留
location	非保留
lock	非保留
locked	非保留
log	非保留
logging	非保留
logical_data	保留
logical_file	保留
login_any	非保留

关键字	类别
login_failure	非保留
login_success	非保留
logout	非保留
long	非保留
loop	非保留
mapping	非保留
masking	非保留
master	非保留
match	非保留
matched	非保留
materialized	非保留
max	非保留
maxextents	非保留
maxsize	非保留
maxtrans	非保留
maxvalue	保留
merge	非保留
minextents	非保留
minus	保留
minute	非保留
minvalue	非保留
mode	非保留
model	非保留
modify	保留
month	非保留
move	非保留
movement	非保留
name	非保留
names	非保留
national	非保留 (不可用于函数、类型名)
natural	保留 (可用于函数、类型名)
nchar	非保留 (不可用于函数、类型名)

关键字	类别
next	非保留
no	非保留
nocache	非保留
nocompress	非保留
nocycle	保留
node	非保留
nologging	非保留
nomaxvalue	非保留
nominvalue	非保留
none	非保留 (不可用于函数、类型名)
noorder	非保留
normally	非保留
not	保留
nothing	非保留
notify	非保留
notnull	保留 (可用于函数、类型名)
nowait	非保留
ntext	非保留
null	保留
nullcols	非保留
nullif	非保留 (不可用于函数、类型名)
nulls	非保留
number	非保留 (不可用于函数、类型名)
numeric	非保留 (不可用于函数、类型名)
numstr	非保留
nvarchar	非保留 (不可用于函数、类型名)
nvarchar2	非保留 (不可用于函数、类型名)
nvl	非保留 (不可用于函数、类型名)
nvl2	非保留 (不可用于函数、类型名)
object	非保留
of	非保留
off	保留

关键字	类别
offset	保留
oids	非保留
on	保留
only	保留
openxml	保留
operator	非保留
optimization	非保留
option	非保留
optionally	非保留
options	非保留
or	保留
ora_hash	非保留 (不可用于函数、类型名)
order	保留
out	非保留 (不可用于函数、类型名)
outer	保留 (可用于函数、类型名)
outfile	非保留
output	保留
over	非保留
overlaps	保留 (可用于函数、类型名)
overlay	非保留 (不可用于函数、类型名)
owned	非保留
owner	非保留
package	非保留
packages	非保留
pad_attribute	非保留
parallel	非保留
parser	非保留
partial	非保留
partition	非保留
partitions	非保留
passing	非保留
password	非保留

关键字	类别
pctfree	非保留
per	非保留
percent	保留
performance	保留
perm	非保留
physical_only	保留
pipelined	非保留
pivot	非保留
placing	保留
plan	非保留
plans	非保留
policy	非保留
pool	非保留
position	非保留 (不可用于函数、类型名)
precedes	非保留
preceding	非保留
precision	非保留 (不可用于函数、类型名)
predict	非保留
preferred	非保留
prefix	非保留
prepare	非保留
prepared	非保留
preserve	非保留
primary	保留
prior	非保留
priorer	保留
private	非保留
privilege	非保留
privileges	非保留
procedural	非保留
procedure	保留
profile	非保留

关键字	类别
public	非保留
publication	非保留
publish	非保留
purge	非保留
query	非保留
quote	非保留
randomized	非保留
range	非保留
ratio	非保留
raw	非保留
read	非保留
real	非保留 (不可用于函数、类型名)
reassign	非保留
rebuild	非保留
recheck	非保留
recursive	非保留
recyclebin	保留 (可用于函数、类型名)
redisanyvalue	非保留
ref	非保留
references	保留
refresh	非保留
reindex	非保留
reject	保留
relative	非保留
release	非保留
reloptions	非保留
remote	非保留
remove	非保留
rename	非保留
repeat	非保留
repeatable	非保留
replace	非保留

关键字	类别
replica	非保留
reserve	非保留
reset	非保留
resize	非保留
resource	非保留
restart	非保留
restrict	非保留
result_case_mode	非保留
return	保留
returning	保留
returns	非保留
reuse	非保留
revoke	非保留
right	保留 (可用于函数、类型名)
role	非保留
roles	非保留
rollback	非保留
rollup	非保留
rotation	非保留
row	非保留 (不可用于函数、类型名)
rowid	非保留
rownum	保留
rows	非保留
rowtype	非保留
rule	非保留
sample	保留 (可用于函数、类型名)
savepoint	非保留
schedule	非保留
schema	非保留
score	非保留
scroll	非保留
search	非保留

关键字	类别
second	非保留
security	非保留
seed	非保留
select	保留
sensitive	非保留
sequence	非保留
sequences	非保留
serializable	非保留
server	非保留
session	非保留
session_user	保留
sessiontimezone	保留
set	非保留
setof	非保留 (不可用于函数、类型名)
sets	非保留
share	非保留
shippable	非保留
show	非保留
shrink	保留
shutdown	非保留
siblings	非保留
similar	保留 (可用于函数、类型名)
simple	非保留
size	非保留
skip	非保留
slave	非保留
slice	非保留
smalldatetime	非保留 (不可用于函数、类型名)
smalldatetime_format	非保留
smallint	非保留 (不可用于函数、类型名)
smallmoney	非保留
snapshot	非保留

关键字	类别
some	保留
source	非保留
space	非保留
spill	非保留
split	非保留
sql	非保留
stable	非保留
standalone	非保留
start	非保留
starting	非保留
starts	非保留
statement	非保留
statement_id	非保留
statistics	非保留
stdin	非保留
stdout	非保留
storage	非保留
store	非保留
stored	非保留
stratify	非保留
stream	非保留
strict	非保留
strip	非保留
subpartition	非保留
subpartitions	非保留
subscription	非保留
substring	非保留 (不可用于函数、类型名)
symmetric	保留
synonym	非保留
sys_refcursor	非保留
sysdate	保留
sysid	非保留

关键字	类别
system	非保留
systimestamp	非保留
table	保留
tables	非保留
tablesample	保留 (可用于函数、类型名)
tablespace	非保留
target	非保留
temp	非保留
template	非保留
temporary	非保留
terminated	非保留
text	非保留
than	非保留
then	保留
threshold	非保留
ties	保留
time	非保留 (不可用于函数、类型名)
time_format	非保留
time_period	非保留
timecapsule	保留 (可用于函数、类型名)
timestamp	非保留 (不可用于函数、类型名)
timestamp_format	非保留
timestampdiff	非保留 (不可用于函数、类型名)
tinyint	非保留 (不可用于函数、类型名)
to	保留
top	保留
trailing	保留
transaction	非保留
transform	非保留
treat	非保留 (不可用于函数、类型名)
trigger	非保留
trim	非保留 (不可用于函数、类型名)

关键字	类别
TRUE	保留
truncate	非保留
trusted	非保留
try	非保留
tsfield	非保留
tstag	非保留
tstime	非保留
type	非保留
types	非保留
unbounded	非保留
uncommitted	非保留
unencrypted	非保留
union	保留
unique	保留
uniqueidentifier	非保留 (不可用于函数、类型名)
unknown	非保留
unlimited	非保留
unlisten	非保留
unlock	非保留
unlogged	非保留
unpivot	非保留
until	非保留
unusable	非保留
update	非保留
useeof	非保留
user	保留
using	保留
vacuum	非保留
valid	非保留
validate	非保留
validation	非保留
validator	非保留

关键字	类别
value	非保留
values	非保留 (不可用于函数、类型名)
varbinary	保留 (可用于函数、类型名)
varchar	非保留 (不可用于函数、类型名)
varchar2	非保留 (不可用于函数、类型名)
variables	非保留
variadic	保留
varying	非保留
pw_hash	非保留 (不可用于函数、类型名)
vcgroup	非保留
verbose	保留 (可用于函数、类型名)
verify	保留
version	非保留
view	非保留
visible	非保留
volatile	非保留
wait	非保留
warning	非保留
warnings	非保留
weak	非保留
when	保留
where	保留
while	非保留
whitespace	非保留
window	保留
with	保留
within	非保留
without	非保留
work	非保留
workload	非保留
wrapper	非保留
write	非保留

关键字	类别
xml	非保留
xmlattributes	非保留 (不可用于函数、类型名)
xmlconcat	非保留 (不可用于函数、类型名)
xmlelement	非保留 (不可用于函数、类型名)
xmlexists	非保留 (不可用于函数、类型名)
xmlforest	非保留 (不可用于函数、类型名)
xmlnamespaces	非保留 (不可用于函数、类型名)
xmlparse	非保留 (不可用于函数、类型名)
xmlpi	非保留 (不可用于函数、类型名)
xmlroot	非保留 (不可用于函数、类型名)
xmlserialize	非保留 (不可用于函数、类型名)
xmltable	非保留 (不可用于函数、类型名)
year	非保留
yes	非保留
zone	非保留

表 2 PL/pgSQL 保留关键字

关键字	类别
absolute	非保留
alias	非保留
all	保留
alter	非保留
array	非保留
as	非保留
backward	非保留
begin	保留
bulk	非保留
by	保留
call	保留
case	保留
catch	保留

close	保留
collate	保留
collect	非保留
commit	非保留
constant	非保留
continue	非保留
current	非保留
cursor	非保留
debug	非保留
declare	保留
default	保留
delete	保留
detail	非保留
diagnostics	保留
distinct	非保留
do	非保留
dump	非保留
eave	非保留
else	保留
elseif	保留
elsif	保留
end	保留
errcode	非保留
error	非保留
except	非保留
exception	保留
exceptions	非保留
exec	保留
execute	保留
exit	保留
fetch	保留
first	非保留
for	保留

forall	保留
foreach	保留
forward	非保留
from	保留
function	非保留
get	保留
goto	保留
hint	非保留
if	保留
immediate	非保留
in	保留
index	非保留
info	非保留
insert	保留
instantiation	非保留
intersect	非保留
into	保留
is	非保留
iterate	非保留
last	非保留
limit	保留
log	非保留
loop	保留
merge	非保留
message	非保留
message_text	非保留
move	保留
multiset	非保留
next	非保留
no	非保留
not	保留
notice	非保留
null	保留

of	保留
open	保留
option	非保留
or	保留
out	保留
package	非保留
perform	非保留
pg_exception_context	非保留
pg_exception_detail	非保留
pg_exception_hint	非保留
pipe	非保留
pls_integer	非保留
pragma	非保留
print_strict_params	非保留
prior	非保留
procedure	保留
query	非保留
raise	保留
record	非保留
ref	保留
relative	非保留
release	非保留
repeat	非保留
result_oid	非保留
return	保留
returned_sqlstate	非保留
reverse	非保留
rollback	非保留
row	非保留
row_count	非保留
rowtype	非保留
save	非保留
savepoint	非保留

scroll	非保留
select	保留
slice	非保留
sqlstate	非保留
stacked	非保留
strict	保留
sys_refcursor	非保留
table	保留
then	保留
to	保留
top	保留
try	保留
type	非保留
union	非保留
until	非保留
update	保留
updating	保留
use_column	非保留
use_variable	非保留
using	保留
values	非保留
variable_conflict	非保留
varray	非保留
warning	非保留
when	保留
while	保留
with	非保留

4.1.2.数据类型

数据类型是一组值的集合以及定义在这个值集上的一组操作。PanWeiDB 数据库是由表的集合组成的，而各表中的列定义了该表，每一列都属于一种数据类型，PanWeiDB 根据数据类型有相应函数对其内容进行操作，例如 PanWeiDB 可对数值型数据进行加、减、乘、除操作。

4.1.2.1.日期-时间类型

PanWeiDB 支持的日期/时间类型请参见表 1。该类型的操作符和内置函数请参考：SQL 语法参考->函数和操作符->时间和日期处理函数和操作符章节。

【说明】
如果其他的数据库时间格式和 panweidb 的时间格式不一致，可通过修改配置参数 DateStyle 的值来保持一致。

表 1 日期/时间类型

名称	描述	存储空间
DATE	日期和时间。	4 字节（兼容模式 A 下存储空间大小为 8 字节）
TIME [(p)] [WITHOUT TIME ZONE]	只用于一日内时间。 p 表示小数点后的精度，取值范围为 0~6。	8 字节
TIME [(p)] [WITH TIME ZONE]	只用于一日内时间，带时区。别名为 TIMETZ p 表示小数点后的精度，取值范围为 0~6。	12 字节
TIMESTAMP[(p)] [WITHOUT TIME ZONE]	日期和时间。 p 表示小数点后的精度，取值范围为 0~6。	8 字节
TIMESTAMP[(p)][	日期和时间，带时区。别名为	8 字节

名称	描述	存储空间
WITH TIME ZONE]	TIMESTAMPTZ。 p 表示小数点后的精度，取值范围为 0~6。	
SMALLDATETIME	日期和时间，不带时区。 精确到分钟，秒位大于等于 30 秒进一位。	8 字节
INTERVAL DAY (l) TO SECOND (p)	时间间隔，X 天 X 小时 X 分 X 秒。 ❖ l：天数的精度，取值范围为 0~6。兼容性考虑，目前未实现具体功能。 ❖ p：秒数的精度，取值范围为 0~6。小数末尾的零不显示。	16 字节
INTERVAL [FIELDS] [(p)]	时间间隔。 fields：可以是 YEAR、MONTH、DAY、HOUR、MINUTE、SECOND、DAY TO HOUR、DAY TO MINUTE、DAY TO SECOND、HOUR TO MINUTE、HOUR TO SECOND、MINUTE TO SECOND。 p：秒数的精度，取值范围为 0~6，且 fields 为 SECOND、DAY TO SECOND、HOUR TO SECOND 或 MINUTE TO SECOND 时，参数 p 才有效。小数末尾的零不显示。	12 字节
reltime	相对时间间隔。格式为： X years X mons X days XX:XX:XX。 采用儒略历计时，规定一年为 365.25 天，一个月为 30 天，计算输入值对应的相对时间间隔，输出采用 POSTGRES 格式。	4 字节
abstime	日期和时间。格式为： YYYY-MM-DD hh:mm:ss+timezone 取值范围为 1901-12-13 20:45:53	4 字节

名称	描述	存储空间
	GMT~2038-01-18 23:59:59 GMT，精度为秒。	

示例 1：创建 date 类型数据。

1、创建表。

```
CREATE TABLE date_type_tab(coll date);
```

2、插入数据。

```
INSERT INTO date_type_tab VALUES (date '12-10-2010');
```

3、查看数据。

```
SELECT * FROM date_type_tab;
```

结果返回如下：

```
     coll
---------------------
 2010-12-10 00:00:00
(1 row)
```

4、删除表。

```
DROP TABLE date_type_tab;
```

示例 2：创建 time without time zone、time with time zone、timestamp without time zone、timestamp with time zone、smalldatetime 五种类型数据。

1、创建表。

```
CREATE TABLE time_type_tab (da time without time zone ,dai time with time zone,dfgh timestamp without time zone,dfga timestamp with time zone, pwg smalldatetime);
```

2、插入数据。

```
INSERT INTO time_type_tab VALUES ('21:21:21','21:21:21 pst','2010-12-12','2013-12
-11 pst','2003-04-12 04:05:06');
```

3、查看数据。

```
SELECT * FROM time_type_tab;
```

结果返回如下

```
  da   |   dai   |      dfgh      |       dfga       |      pwg
-------+---------+----------------+------------------+---------------
-----
 21:21:21 | 21:21:21-08 | 2010-12-12 00:00:00 | 2013-12-11 16:00:00+08 | 2003-04-1
2 04:05:00
(1 row)
```

4、删除表。

```
DROP TABLE time_type_tab;
```

示例 3：创建天数间隔 INTERVAL 类型数据。

1、创建表。

```
CREATE TABLE day_type_tab (a int,b INTERVAL DAY(3) TO SECOND (4));
```

2、插入数据。

```
INSERT INTO day_type_tab VALUES (1, INTERVAL '3' DAY);
```

3、查看数据。

```
SELECT * FROM day_type_tab;
```

结果返回如下：

```
a |  b
---+--------
 1 | 3 days
(1 row)
```

4、删除表。

```
DROP TABLE day_type_tab;
```

示例 4：创建年份间隔 INTERVAL 类型数据。

1、创建表。

```
CREATE TABLE year_type_tab(a int, b interval year (6));
```

2、插入数据。

```
INSERT INTO year_type_tab VALUES(1,interval '2' year);
```

3、查看数据。

```
SELECT * FROM year_type_tab;
```

结果返回如下：

```
 a |   b
---+---------
 1 | 2 years
(1 row)
```

4、删除表。

```
DROP TABLE year_type_tab;
```

日期输入

日期和时间的输入几乎可以是任何合理的格式，包括 ISO-8601 格式、SQL-兼容格式、传统 POSTGRES 格式或者其他的形式。系统支持按照日、月、年的顺序自定义日期输入。如果把 DateStyle 参数设置为 MDY 就按照“月-日-年”解析，设置为 DMY 就按照“日-月-年”解析，设置为 YMD 就按照“年-月-日”解析。

日期的文本输入需要加单引号包围，语法如下：

```
type [(p)]'value'
```

可选的精度声明中的 p 是一个整数，表示在秒域中小数部分的位数。表 2 显示了 date 类型的输入方式。

【说明】

合法的日期分隔符是“-”和“/”，混用日期分隔符（“-”和“/”也不能混用）可能导致解析错误。

表 2 日期输入方式

例子	描述
1999-01-08	ISO 8601 格式（建议格式），任何方式下都是 1999 年 1 月 8 号。
January 8, 1999	在任何 datestyle 输入模式下都无歧义。
1/8/1999	有歧义，在 MDY 模式下是一月八号，在 DMY 模式下是八月一号。
1/18/1999	MDY 模式下是一月十八日，其他模式下被拒绝。
01/02/03	MDY 模式下的 2003 年 1 月 2 日。 DMY 模式下的 2003 年 2 月 1 日。 YMD 模式下的 2001 年 2 月 3 日。
1999-Jan-08	任何模式下都是 1 月 8 日。
Jan-08-1999	任何模式下都是 1 月 8 日。
08-Jan-1999	任何模式下都是 1 月 8 日。
99-Jan-08	YMD 模式下是 1 月 8 日，否则错误。
08-Jan-99	一月八日，除了在 YMD 模式下是错误的之外。
Jan-08-99	一月八日，除了在 YMD 模式下是错误的之外。
19990108	ISO 8601；任何模式下都是 1999 年 1 月 8 日。
990108	ISO 8601；任何模式下都是 1999 年 1 月 8 日。
1999.008	年和年里的第几天。
J2451187	儒略日。

例子	描述
January 8, 99 BC	公元前 99 年。

示例：

```
--创建表。
panweidb=# CREATE TABLE date_type_tab(coll date);
--插入数据。
panweidb=# INSERT INTO date_type_tab VALUES (date '12-10-2010');
--查看数据。
panweidb=# SELECT * FROM date_type_tab;
      coll
---------------------
 2010-12-10 00:00:00
(1 row)

--查看日期格式。
panweidb=# SHOW datestyle;
 DateStyle
-----------
 ISO, MDY
(1 row)

--设置日期格式。
panweidb=# SET datestyle='YMD';
SET
--插入数据。
panweidb=# INSERT INTO date_type_tab VALUES(date '2010-12-11');
--查看数据。
panweidb=# SELECT * FROM date_type_tab;
      coll
---------------------
 2010-12-10 00:00:00
 2010-12-11 00:00:00
(2 rows)
```

```
--删除表。
panweidb=# DROP TABLE date_type_tab;
```

时间

时间类型包括 time [(p)] without time zone 和 time [(p)] with time zone。如果只写 time 等效于 time without time zone。

如果在 time without time zone 类型的输入中声明了时区，则会忽略这个时区。

时间输入类型的详细信息请参见表 3，时区输入类型的详细信息请参加表 4。

表 3 时间输入

例子	描述
05:06.8	ISO 8601
4:05:06	ISO 8601
4:05	ISO 8601
040506	ISO 8601
4:05 AM	与 04:05 一样，AM 不影响数值
4:05 PM	与 16:05 一样，输入小时数必须<= 12
04:05:06.789-8	ISO 8601
04:05:06-08:00	ISO 8601
04:05-08:00	ISO 8601
040506-08	ISO 8601
04:05:06 PST	缩写的时区
2003-04-12 04:05:06 America/New_York	用名称声明的时区

表 4 时区输入

例子	描述
PST	太平洋标准时间（Pacific Standard Time）
America/New_York	完整时区名称
-8:00	ISO 8601 与 PST 的偏移
-800	ISO 8601 与 PST 的偏移
-8	ISO 8601 与 PST 的偏移

示例：

```
panweidb=# SELECT time '04:05:06';
  time
----------
 04:05:06
(1 row)

panweidb=# SELECT time '04:05:06 PST';
  time
----------
 04:05:06
(1 row)

panweidb=# SELECT time with time zone '04:05:06 PST';
  timetz
------------
 04:05:06-08
(1 row)
```

特殊值

panweidb 支持几个特殊值，在读取的时候将被转换成普通的日期/时间值，请参考表 5。

表 5 特殊值

输入字符串	适用类型	描述
epoch	date、timestamp	1970-01-01 00:00:00+00 （Unix 系统零时）
infinity	timestamp	比任何其他时间戳都晚
-infinity	timestamp	比任何其他时间戳都早
now	date、time、timestamp	当前事务的开始时间
today	date、timestamp	今日午夜
tomorrow	date、timestamp	明日午夜
yesterday	date、timestamp	昨日午夜
allballs	time	00:00:00.00 UTC

示例：

1、创建测试表并插入数据。

```
CREATE TABLE realtime_type_special(col1 varchar(20), col2 date, col3 timestamp, c
ol4 time);
INSERT INTO realtime_type_special VALUES('epoch', 'epoch', 'epoch', NULL);
INSERT INTO realtime_type_special VALUES('infinity', NULL, 'epoch', NULL);
INSERT INTO realtime_type_special VALUES('now', 'now', 'now', 'now');
 INSERT INTO realtime_type_special VALUES('today', 'today', 'today', NULL);
 INSERT INTO realtime_type_special VALUES('tomorrow', 'tomorrow', 'tomorrow', N
ULL);
INSERT INTO realtime_type_special VALUES('yesterday', 'yesterday', 'yesterday', NU
LL);
```

2、查看数据。

```
SELECT * FROM realtime_type_special;
SELECT * FROM realtime_type_special WHERE col3 < 'infinity';
SELECT * FROM realtime_type_special WHERE col3 > '-infinity';
SELECT * FROM realtime_type_special WHERE col3 > 'now';
SELECT * FROM realtime_type_special WHERE col3 = 'today';
```

```
SELECT * FROM realtime_type_special WHERE col3 = 'tomorrow';
SELECT * FROM realtime_type_special WHERE col3 > 'yesterday';
SELECT TIME 'allballs';
```

返回结果依次为：

```
 col1      |        col2         |            col3            |      col4
-----------+---------------------+----------------------------+-----------------
 epoch     | 1970-01-01 00:00:00 | 1970-01-01 00:00:00        |
 now       | 2023-02-27 11:38:13 | 2023-02-27 11:38:13.032815 | 11:38:13.032815
 today     | 2023-02-27 00:00:00 | 2023-02-27 00:00:00        |
 tomorrow  | 2023-02-28 00:00:00 | 2023-02-28 00:00:00        |
 yesterday | 2023-02-26 00:00:00 | 2023-02-26 00:00:00        |
(5 rows)

   col1    |        col2         |            col3            |      col4
-----------+---------------------+----------------------------+-----------------
 epoch     | 1970-01-01 00:00:00 | 1970-01-01 00:00:00        |
 now       | 2023-02-27 11:38:13 | 2023-02-27 11:38:13.032815 | 11:38:13.032815
 today     | 2023-02-27 00:00:00 | 2023-02-27 00:00:00        |
 tomorrow  | 2023-02-28 00:00:00 | 2023-02-28 00:00:00        |
 yesterday | 2023-02-26 00:00:00 | 2023-02-26 00:00:00        |
(5 rows)

   col1    |        col2         |            col3            |      col4
-----------+---------------------+----------------------------+-----------------
 epoch     | 1970-01-01 00:00:00 | 1970-01-01 00:00:00        |
 now       | 2023-02-27 11:38:13 | 2023-02-27 11:38:13.032815 | 11:38:13.032815
 today     | 2023-02-27 00:00:00 | 2023-02-27 00:00:00        |
 tomorrow  | 2023-02-28 00:00:00 | 2023-02-28 00:00:00        |
 yesterday | 2023-02-26 00:00:00 | 2023-02-26 00:00:00        |
(5 rows)

   col1   |        col2         |        col3         | col4
----------+---------------------+---------------------+------
 tomorrow | 2023-02-28 00:00:00 | 2023-02-28 00:00:00 |
(1 row)
```

```
 col1 |      col2       |      col3       | col4
------+-------------------+-------------------+------
 today | 2023-02-27 00:00:00 | 2023-02-27 00:00:00 |
(1 row)

  col1  |      col2       |      col3       | col4
----------+-------------------+-------------------+------
 tomorrow | 2023-02-28 00:00:00 | 2023-02-28 00:00:00 |
(1 row)

  col1  |      col2       |         col3          |     col4
----------+-------------------+-------------------------+----------------
 now      | 2023-02-27 11:38:13 | 2023-02-27 11:38:13.032815 | 11:38:13.032815
 today    | 2023-02-27 00:00:00 | 2023-02-27 00:00:00       |
 tomorrow | 2023-02-28 00:00:00 | 2023-02-28 00:00:00       |
(3 rows)

  time
----------
 00:00:00
(1 row)
```

3、删除表。

```
DROP TABLE realtime_type_special;
```

时间段输入

reltime 的输入方式可以采用任何合法的时间段文本格式，包括数字形式（含负数和小数）及时间形式，其中时间形式的输入支持 SQL 标准格式、ISO-8601 格式、POSTGRES 格式等。另外，文本输入需要加单引号。

时间段输入的详细信息请参考表 6。

表 6 时间段输入

输入示例	输出结果	描述

输入示例	输出结果	描述
60	2 mons	采用数字表示时间段，默认单位是 day，可以是小数或负数。特别的，负数时间段，在语义上，可以理解为“早于多久”。
31.25	1 mons 1 days 06:00:00	
-365	-12 mons -5 days	
1 years 1 mons 8 days 12:00:00	1 years 1 mons 8 days 12:00:00	采用 POSTGRES 格式表示时间段，可以正负混用，不区分大小写，输出结果为将输入时间段计算并转换得到的简化 POSTGRES 格式时间段。
-13 months -10 hours	-1 years -25 days -04:00:00	
-2 YEARS +5 MONTHS 10 DAYS	-1 years -6 mons -25 days -06:00:00	
P-1.1Y10M	-3 mons -5 days -06:00:00	采用 ISO-8601 格式表示时间段，可以正负混用，"H"不区分大小写，输出结果为将输入时间段计算并转换得到的简化 POSTGRES 格式时间段。
-12H	-12:00:00	

示例：

```
--创建表。
panweidb=# CREATE TABLE reltime_type_tab(col1 character(30), col2 reltime);

--插入数据。
panweidb=# INSERT INTO reltime_type_tab VALUES ('90', '90');
panweidb=# INSERT INTO reltime_type_tab VALUES ('-366', '-366');
panweidb=# INSERT INTO reltime_type_tab VALUES ('1975.25', '1975.25');
panweidb=# INSERT INTO reltime_type_tab VALUES ('-2 YEARS +5 MONTHS 10 DAYS',
'-2 YEARS +5 MONTHS 10 DAYS');
panweidb=# INSERT INTO reltime_type_tab VALUES ('30 DAYS 12:00:00', '30 DAYS 1
2:00:00');
panweidb=# INSERT INTO reltime_type_tab VALUES ('P-1.1Y10M', 'P-1.1Y10M');
```

```
--查看数据。
panweidb=# SELECT * FROM reltime_type_tab;
          col1           |             col2
-------------------------------+-----------------------------------
 1975.25                      | 5 years 4 mons 29 days
 -2 YEARS +5 MONTHS 10 DAYS   | -1 years -6 mons -25 days -06:00:00
 P-1.1Y10M                    | -3 mons -5 days -06:00:00
 -366                         | -1 years -18:00:00
 90                           | 3 mons
 30 DAYS 12:00:00             | 1 mon 12:00:00
(6 rows)

--删除表。
panweidb=# DROP TABLE reltime_type_tab;
```

4.1.2.2.数值类型

本小节列出了所有的可用的数值类型（整数类型、任意精度型、浮点类型、序列整型）。数字操作符和相关的内置函数请参见：SQL 语法参考->SQL 基本要素->函数和操作符->数字操作函数和操作符。

4.1.2.2.1. 整数类型

表 1 整数类型

名称	描述	存储空间	范围
TINYINT	微整数，别名为 INT1。	1 字节	0 ~ 255
SMALLINT	小范围整数，别名为 INT2。	2 字节	-32,768 ~ +32,767
INTEGER	常用的整数，别名为 INT4。	4 字节	-2,147,483,648 ~ +2,147,483,647

名称	描述	存储空间	范围
BINARY_INTEGER	常用的整数 INTEGER 的别名。	4 字节	-2,147,483,648 ~ +2,147,483,647
BIGINT	大范围的整数，别名为 INT8。	8 字节	-9,223,372,036,854,775,808 ~ +9,223,372,036,854,775,807
int16	十六字节的大范围证书，目前不支持用户用于建表等使用。	16 字节	-170,141,183,460,469,231,731,687,303,715,884,105,728 ~ +170,141,183,460,469,231,731,687,303,715,884,105,727

说明

- TINYINT、SMALLINT、INTEGER、BIGINT 和 INT16 类型存储各种范围的数字，也就是整数。试图存储超出范围以外的数值将会导致错误。
- 常用的类型是 INTEGER，因为它提供了在范围、存储空间、性能之间的最佳平衡。一般只有取值范围确定不超过 SMALLINT 的情况下，才会使用 SMALLINT 类型。而只有在 INTEGER 的范围不够的时候才使用 BIGINT，因为前者相对快得多。

示例

示例 1：创建具有 TINYINT 类型数据的表。

1、创建测试表。

```
CREATE TABLE int_type_t1(IT_COL1 TINYINT);
```

2、向表中插入数据。

```
INSERT INTO int_type_t1 VALUES(10);
```

3、查看数据。

```
SELECT * FROM int_type_t1;
```

返回结果为：

```
 it_col1
---------
 10
(1 row)
```

4、删除表。

```
DROP TABLE int_type_t1;
```

示例 2：创建具有 TINYINT,INTEGER,BIGINT 类型数据的表。

1、创建测试表。

```
CREATE TABLE int_type_t2 (
   a TINYINT,
   b TINYINT,
   c INTEGER,
   d BIGINT
);
```

2、向表中插入数据。

```
INSERT INTO int_type_t2 VALUES(100, 10, 1000, 10000);
```

3、查看数据。

```
 SELECT * FROM int_type_t2;
```

返回结果为：

```
a |b | c  |  d
-----+----+------+-------
 100 | 10 | 1000 | 10000
(1 row)
```

4、删除表。

```
DROP TABLE int_type_t2;
```

4.1.2.2.2. 任意精度型

表 2 任意精度型

名称	描述	存储空间	范围
NUMERIC[(p[,s])], DECIMAL[(p[,s])]	精度 p 取值范围为[1,1000]，标度 s 取值范围为[0,p]。 说明： p 为总位数，s 为小数位数。	用户声明精度。每四位（十进制位）占用两个字节，然后在整个数据上加上八个字节的额外开销。	未指定精度的情况下，小数点前最大 131,072 位，小数点后最大 16,383 位。
NUMBER[(p[,s])]	NUMERIC 类型的别名。	用户声明精度。每四位（十进制位）占用两个字节，然后在整个数据上加上八个字节的额外开销。	未指定精度的情况下，小数点前最大 131,072 位，小数点后最大 16,383 位。

示例

示例 1：创建具有 DECIMAL 类型数据的表。

1、创建测试表。

```
CREATE TABLE decimal_type_t1 ( DT_COL1 DECIMAL(10,4));
```

2、向表中插入数据。

```
INSERT INTO decimal_type_t1 VALUES(123456.122331);
```

3、查询表中的数据。

```
SELECT * FROM decimal_type_t1;
```

返回结果为：

```
 dt_col1
-------------
```

```
 123456.1223
(1 row)
```

4、删除表。

```
DROP TABLE decimal_type_t1;
```

示例 2：创建具有 NUMERIC 类型数据的表。

1、创建测试表。

```
CREATE TABLE numeric_type_t1 (NT_COL1 NUMERIC(10,4));
```

2、向表中插入数据。

```
INSERT INTO numeric_type_t1 VALUES(123456.12354);
```

3、查看数据。

```
SELECT * FROM numeric_type_t1;
```

返回结果为：

```
  nt_col1
-------------
 123456.1235
(1 row)
```

4、删除表。

```
DROP TABLE numeric_type_t1;
```

说明

- 与整数类型相比，任意精度类型需要更大的存储空间，其存储效率、运算效率以及压缩比效果都要差一些。在进行数值类型定义时，优先选择整数类型。当且仅当数值超出整数可表示最大范围时，再选用任意精度类型。
- 使用 Numeric/Decimal 进行列定义时，建议指定该列的精度 p 以及标度 s。

4.1.2.2.3. 序列整型

表 3 序列整型

名称	描述	存储空间	范围
SMALLSERIAL	二字节序列整型。	2 字节	-32,768 ~ +32,767
SERIAL	四字节序列整型。	4 字节	-2,147,483,648 ~ +2,147,483,647
BIGSERIAL	八字节序列整型。	8 字节	-9,223,372,036,854,775,808 ~ +9,223,372,036,854,775,807
LARGESERIAL	默认插入十六字节序列整形，实际数值类型和 numeric 相同。	变长类型，每四位（十进制位）占用两个字节，然后在整个数据上加上八个字节的额外开销。	小数点前最大 131,072 位，小数点后最大 16,383 位。

说明

SMALLSERIAL、SERIAL、BIGSERIAL 和 LARGESERIAL 类型不是真正的类型，只是为在表中设置唯一标识做的概念上的便利。因此，创建一个整数字段，并且把它的缺省数值安排为从一个序列发生器读取。应用了一个 NOT NULL 约束以确保 NULL 不会被插入。在大多数情况下用户可能还希望附加一个 UNIQUE 或 PRIMARY KEY 约束避免意外地插入重复的数值，但这个不是自动的。最后，将序列发生器从属于那个字段，这样当该字段或表被删除的时候也一并删除它。目前只支持在创建表时候指定 SERIAL 列，不可以在已有的表中，

增加 SERIAL 列。另外临时表也不支持创建 SERIAL 列。因为 SERIAL 不是真正的类型，也不可以将表中存在的列类型转化为 SERIAL。

示例

示例 1：创建具有 SMALLSERIAL 类型数据的表。

1、创建表。

```
CREATE TABLE smallserial_type_tab(a SMALLSERIAL);
```

2、插入数据。

```
INSERT INTO smallserial_type_tab VALUES(default);
INSERT INTO smallserial_type_tab VALUES(default);
```

3、查看数据。

```
SELECT * FROM smallserial_type_tab;
```

返回结果为：

```
 a
---
 1
 2
(2 rows)
```

示例 2：创建具有 SERIAL 类型数据的表。

1、创建表。

```
CREATE TABLE serial_type_tab(b SERIAL);
```

2、插入数据。

```
INSERT INTO serial_type_tab VALUES(default);
INSERT INTO serial_type_tab VALUES(default);
```

3、查看结果。

```
SELECT * FROM serial_type_tab;
```

返回结果为：

```
b
---
 1
 2
(2 rows)
```

示例 3：创建具有 BIGSERIAL 类型数据的表。

1、创建表。

```
CREATE TABLE bigserial_type_tab(c BIGSERIAL);
```

2、插入数据。

```
INSERT INTO bigserial_type_tab VALUES(default);
INSERT INTO bigserial_type_tab VALUES(default);
```

3、查看数据。

```
SELECT * FROM bigserial_type_tab;
```

返回结果：

```
 b
---
 1
 2
(2 rows)
```

示例 4：创建具有 LARGESERIAL 类型数据的表。

1、创建表。

```
CREATE TABLE largeserial_type_tab(c LARGESERIAL);
```

2、插入数据。

```
INSERT INTO largeserial_type_tab VALUES(default);
INSERT INTO largeserial_type_tab VALUES(default);
```

3、查看数据。

```
SELECT * FROM largeserial_type_tab;
```

返回结果为：

```
c
---
1
2
(2 rows)
```

4.1.2.2.4. 浮点类型

表 4 浮点类型

名称	描述	存储空间	范围
REAL	单精度浮点数，不精准。	4 字节	-3.402E+38~3.402E+38，6 位十进制数字精度。
FLOAT4			
DOUBLE PRECISION，别名 BINARY_DOUBLE	双精度浮点数，不精准。	8 字节	-1.79E+308~1.79E+308，15 位十进制数字精度。
FLOAT8			
FLOAT[(p)]	浮点数，不精准。精度 p 取值范围为[1,53]。说明：p 为精度，表示总位数。	4 字节或 8 字节	根据精度 p 不同选择 REAL 或 DOUBLE PRECISION 作为内部表示。如不指定精度，内部用 DOUBLE PRECISION 表示。
BINARY_DOUBLE	是 DOUBLE PRECISION 的别名。	8 字节	-1.79E+308~1.79E+308，15 位十进制数字精度。
DEC[(p[,s])]	精度 p 取值范围为[1,1000]，标度 s 取值范围为[0,p]。说明：p 为总位数，s 为小数	用户声明精度。每四位（十进制位）占用两个字节，然后在整个数据上加上	未指定精度的情况下，小数点前最大 131,072 位，小数点后最大 16,383 位。

名称	描述	存储空间	范围
	位位数。	八个字节的额外开销。	
INTEGER[(p[,s])]	精度 p 取值范围为[1,1000]，标度 s 取值范围为[0,p]。	用户声明精度。每四位（十进制位）占用两个字节，然后在整个数据上加上八个字节的额外开销。	-

示例

1、创建测试表。

```
CREATE TABLE float_type_t2
(
  FT_COL1 INTEGER,
  FT_COL2 FLOAT4,
  FT_COL3 FLOAT8,
  FT_COL4 FLOAT(3),
  FT_COL5 BINARY_DOUBLE,
  FT_COL6 DECIMAL(10,4),
  FT_COL7 INTEGER(6,3)
);
```

2、插入数据。

```
INSERT INTO float_type_t2 VALUES(10,10.365456,123456.1234,10.3214, 321.321, 12
3.123654, 123.123654);
```

3、查看数据。

```
SELECT * FROM float_type_t2 ;
```

返回结果为：

```
ft_col1 | ft_col2 |  ft_col3  | ft_col4 | ft_col5 | ft_col6  | ft_col7
```

```
---------+---------+------------+---------+---------+----------+---------
    10 | 10.3655 | 123456.1234 | 10.3214 | 321.321 | 123.1237 | 123.124
(1 row)
```

4、删除表。

```
DROP TABLE float_type_t2;
```

【说明】

REAL、FLOAT4、DOUBLE PRECISION、FLOAT8、FLOAT、BINARY_DOUBLE 等浮点数类型是不精确的，不精确意味着一些数值不能精确地转换成内部格式并且是以近似值存储的，因此存储后再把数据打印出来可能有一些差异。如果想用精确的数值和计算，应使用`numeric`类型。如果想用这些不精确的类型做任何重要的复杂计算，尤其是那些对范围情况（无穷/下溢）严重依赖的事情，应该仔细评估 SQL 和应用实现，直接拿两个浮点数值进行比较，不一定总是能得到预期的结果。

4.1.2.3.货币类型

货币类型存储带有固定小数精度的货币金额。

表 1 中显示的范围假设有两位小数。可以以任意格式输入，包括整型、浮点型或者典型的货币格式（如“$1,000.00”）。根据区域字符集，输出一般是最后一种形式。

表 1 货币类型

名称	存储容量	描述	范围
money	8 字节	货币金额	-92233720368547758.08 到 +92233720368547758.07

numeric、int 和 bigint 类型的值可以转化为 money 类型。如果从 real 和 double precision 类型转换到 money 类型，可以先转化为 numeric 类型，再转化为 money 类型，例如：

```
panweidb=# SELECT '12.34'::float8::numeric::money;
```

这种用法是不推荐使用的。浮点数不应该用来处理货币类型，因为小数点的位数可能会导致错误。

money 类型的值可以转换为 numeric 类型而不丢失精度。转换为其他类型可能丢失精度，并且必须通过以下两步来完成：

```
panweidb=# SELECT '52093.89'::money::numeric::float8;
```

当一个 money 类型的值除以另一个 money 类型的值时，结果是 double precision（也就是一个纯数字，而不是 money 类型）；在运算过程中货币单位相互抵消。

4.1.2.4.布尔类型

表 1 布尔类型

名称	描述	存储空间	取值
BOOLEAN	布尔类型	1 字节。	❖ true：真 ❖ false：假 ❖ null：未知（unknown）

❖ “真”值的有效文本值是：

TRUE、't'、'true'、'y'、'yes'、'1' 、'TRUE'、true、整数范围内 1~2^63-1、整数范围内-1~-2^63。

❖ “假”值的有效文本值是：

FALSE、'f'、'false'、'n'、'no'、'0'、0、'FALSE'、false。

使用 TRUE 和 FALSE 是比较规范的用法（也是 SQL 兼容的用法）。

示例

用字母 t 和 f 输出 Boolean 值。

```
--创建表。
panweidb=# CREATE TABLE bool_type_t1
(
  BT_COL1 BOOLEAN,
  BT_COL2 TEXT
);
```

```
--插入数据。
panweidb=# INSERT INTO bool_type_t1 VALUES (TRUE, 'sic est');

panweidb=# INSERT INTO bool_type_t1 VALUES (FALSE, 'non est');

--查看数据。
panweidb=# SELECT * FROM bool_type_t1;
 bt_col1 | bt_col2
---------+---------
 t       | sic est
 f       | non est
(2 rows)

panweidb=# SELECT * FROM bool_type_t1 WHERE bt_col1 = 't';
 bt_col1 | bt_col2
---------+---------
 t       | sic est
(1 row)

--删除表。
panweidb=# DROP TABLE bool_type_t1;
```

4.1.2.5.字符类型

PanWeiDB 支持的字符类型请参见表 1。字符串操作符和相关的内置函数请参考：SQL 语法参考->SQL 基本要素->函数和操作符->字符处理函数和操作符。

表 1 字符类型

名称	描述	存储空间
CHAR(n)CHARACTER(n)NCHAR(n)	定长字符串，不足补空格。n 是指字节长度，如不带精度 n，默认精度为 1。	最大为 10MB。
VARCHAR(n)CHARACTER VARYING(n)	变长字符串。PG 兼容模式下，n 是字符长度。其他兼容模式下，n 是指字节长度。	最大为 10MB。
VARCHAR2(n)	变长字符串。是 VARCHAR(n)类型的别名。n 是指字节长度。	最大为 10MB。

名称	描述	存储空间
NVARCHAR2(n)	变长字符串。n 是指字符长度。	最大为 10MB。
NVARCHAR(n)	变长字符串。是 NVARCHAR2(n)类型的别名。n 是指字符长度。	最大为 10MB。
TEXT	变长字符串。	最大为 1GB-1，但还需要考虑到列描述头信息的大小，以及列所在元组的大小限制（也小于 1GB-1），因此 TEXT 类型最大大小可能小于 1GB-1。
CLOB	文本大对象。是 TEXT 类型的别名。	最大为 1GB-1，但还需要考虑到列描述头信息的大小，以及列所在元组的大小限制（也小于 1GB-1），因此 CLOB 类型最大大小可能小于 1GB-1。
bpchar	无具体含义，是 varchar 类型的别名。	

说明

- 除了每列的大小限制以外，每个元组的总大小也不可超过 1GB-1 字节，主要受列的控制头信息、元组控制头信息以及元组中是否存在 NULL 字段等影响。
- NCHAR(n)的长度单位为 character，bpchar(n)的长度单位为 byte。
- NCHAR 为 bpchar(n)类型的别名，但长度单位定义为 character。
- 在 PostgreSQL 兼容模式下，CHAR(n)、CHARACTER(n)、NCHAR(n)、VARCHAR(n)、CHARACTER VARYING(n)、VARCHAR2(n)中的 n 代表字符长度。

在 PanWeiDB 里另外还有两种定长字符类型。在表 2 里显示。name 类型只用在内部系统表中，作为存储标识符，不建议普通用户使用。该类型长度当前定为 64 字节（63 可用字符加结束符）。类型 “char” 只用了一个字节的存储空间。他在系统内部主要用于系统表，主要作为简单化的枚举类型使用。

表 2 特殊字符类型

名称	描述	存储空间
name	用于对象名的内部类型。	64 字节。
"char"	单字节内部类型。	1 字节。

示例 1：创建数据类型为 CHARACTER()的字段。

1、创建表。

```
CREATE TABLE char_type_t1
(
  CT_COL1 CHARACTER(4)
);
```

2、插入数据。

```
INSERT INTO char_type_t1 VALUES ('ok');
```

3、查询表中的数据。

```
SELECT ct_col1, char_length(ct_col1) FROM char_type_t1;
```

结果返回如下：

```
 ct_col1 | char_length
---------+-------------
 ok      |           4
(1 row)
```

4、删除表。

```
DROP TABLE char_type_t1;
```

示例 2：创建数据类型为 VARCHAR()的字段。

1、创建表。

```
CREATE TABLE char_type_t2
(
```

```
  CT_COL1 VARCHAR(5)
);
```

2、插入数据。

```
INSERT INTO char_type_t2 VALUES ('ok');
INSERT INTO char_type_t2 VALUES ('good');
```

3、插入的数据长度超过类型规定的长度报错。

```
INSERT INTO char_type_t2 VALUES ('too long');
```

结果返回如下：

```
ERROR:  value too long for type character varying(5)
CONTEXT:  referenced column: ct_col1
```

4、明确类型的长度，超过数据类型长度后会自动截断。

```
INSERT INTO char_type_t2 VALUES ('too long'::varchar(5));
```

5、查询数据。

```
SELECT ct_col1, char_length(ct_col1) FROM char_type_t2;
```

结果返回如下：

```
 ct_col1 | char_length
---------+-------------
 ok      |           2
 good    |           4
 too l   |           5
(3 rows)
```

6、删除数据。

```
DROP TABLE char_type_t2;
```

4.1.2.6.二进制类型

PanWeiDB 支持的二进制类型请参见表 1。

表 1 二进制类型

名称	描述	存储空间
BLOB	二进制大对象。说明：列存不支持 BLOB 类型。	1GB-8203 字节（即 1073733621 字节）

名称	描述	存储空间
RAW	变长的十六进制类型。说明：列存不支持 RAW 类型。	4 字节加上实际的十六进制字符串。最大为 1GB-8203 字节（即 1073733621 字节）。
BYTEA	变长的二进制字符串。	4 字节加上实际的二进制字符串。最大为 1GB-8203 字节（即 1073733621 字节）。
BYTEAWITHOUTORDERWITHEQUALCOL	变长的二进制字符串（密态特性新增的类型，如果加密列的加密类型指定为确定性加密，则该列的实际类型为BYTEAWITHOUTORDERWITHEQUALCOL），元命令打印加密表将显示原始数据类型。	4 字节加上实际的二进制字符串。最大为 1GB 减去 53 字节（即 1073741771 字节）。
BYTEAWITHOUTORDERCOL	变长的二进制字符串（密态特性新增的类型，如果加密列的加密类型指定为随机加密，则该列的实际类型为BYTEAWITHOUTORDERCOL），元命令打印加密表将显示原始数据类型。	4 字节加上实际的二进制字符串。最大为 1GB 减去 53 字节（即 1073741771 字节）。
_BYTEAWITHOUTORDERWITHEQUALCOL	变长的二进制字符串，密态特性新增的类型。	4 字节加上实际的二进制字符串。最大为 1GB 减去 53 字节（即 1073741771 字节）。
_BYTEAWITHOUTORDERCOL	变长的二进制字符串，密态特性新增的类型。	4 字节加上实际的二进制字符串。最大为 1GB 减去 53 字节（即 1073741771 字节）。

说明

- 除了每列的大小限制以外，每个元组的总大小也不可超过 1GB-8203 字节（即 1073733621 字节）。
- 不支持直接使用 BYTEAWITHOUTORDERWITHEQUALCOL、BYTEAWITHOUTORDERCOL、BYTEAWITHOUTORDERWITHEQUALCOL 和 BYTEAWITHOUTORDERCOL 类型创建表。

示例：

```
--创建表。
panweidb=# CREATE TABLE blob_type_t1
(
  BT_COL1 INTEGER,
  BT_COL2 BLOB,
  BT_COL3 RAW,
  BT_COL4 BYTEA
) ;

--插入数据。
panweidb=# INSERT INTO blob_type_t1 VALUES(10,empty_blob(),
HEXTORAW('DEADBEEF'),E'xDEADBEEF');

--查询表中的数据。
panweidb=# SELECT * FROM blob_type_t1;
 bt_col1 | bt_col2 | bt_col3 | bt_col4
---------+---------+----------+------------
      10 |         | DEADBEEF | xdeadbeef
(1 row)

--删除表。
panweidb=# DROP TABLE blob_type_t1;
```

4.1.2.7.几何类型

PanWeiDB 支持的几何类型请参见表 1。最基本的类型：点，是其他类型的基础。

表 1 几何类型

名称	存储空间	说明	表现形式
point	16 字节	平面中的点	(x,y)
lseg	32 字节	（有限）线段	((x1,y1),(x2,y2))
box	32 字节	矩形	((x1,y1),(x2,y2))
path	16+16n 字节	闭合路径（与多边形类似）	((x1,y1),...)
path	16+16n 字节	开放路径	[(x1,y1),...]
polygon	40+16n 字节	多边形（与闭合路径相似）	((x1,y1),...)
circle	24 字节	圆	<(x,y),r> （圆心和半径）

PanWeiDB 提供了一系列的函数和操作符用来进行各种几何计算，如拉伸、转换、旋转、计算相交等。详细信息请参见：SQL 语法参考->SQL 基本要素->函数和操作符->几何函数和操作符。

点

点是几何类型的基本二维构造单位。用下面语法描述 point 的数值：

```
( x , y )
x , y
```

x 和 y 是用浮点数表示的点的坐标。

点输出使用第一种语法。

示例段（lseg）是用

```
SELECT point(23.4, -44.5) AS RESULT;
```

返回结果为：

```
 result
--------------
   (23.4,-44.5)
(1 row)
```

线段

线一对点来代表的。用下面的语法描述 lseg 的数值：

```
[ ( x1 , y1 ) , ( x2 , y2 ) ]
( ( x1 , y1 ) , ( x2 , y2 ) )
( x1 , y1 ) , ( x2 , y2 )
x1 , y1  ,  x2 , y2
```

(x1,y1)和(x2,y2)表示线段的端点。

线段输出使用第一种语法。

矩形

矩形是用一对对角点来表示的。用下面的语法描述 box 的值：

```
( ( x1 , y1 ) , ( x2 , y2 ) )
( x1 , y1 ) , ( x2 , y2 )
x1 , y1  ,  x2 , y2
```

(x1,y1)和(x2,y2)表示矩形的一对对角点。

矩形的输出使用第二种语法。

任何两个对角都可以出现在输入中，但按照那样的顺序，右上角和左下角的值会被重新排序以存储。

示例

将点转换成矩形。

```
SELECT box(point '(0,0)', point '(1,1)') AS RESULT;
```

返回结果为：

```
result
-------------
   (1,1),(0,0)
(1 row)
```

路径

路径由一系列连接的点组成。路径可能是开放的，也就是认为列表中第一个点和最后一个点没有连接，也可能是闭合的，这时认为第一个和最后一个点连接起来。

用下面的语法描述 path 的数值：

```
[ ( x1 , y1 ) , ... , ( xn , yn ) ]
( ( x1 , y1 ) , ... , ( xn , yn ) )
( x1 , y1 ) , ... , ( xn , yn )
( x1 , y1   , ... ,  xn , yn )
x1 , y1   , ... ,  xn , yn
```

点表示组成路径的线段的端点。方括弧（[]）表明一个开放的路径，圆括弧（()）表明一个闭合的路径。当最外层的括号被省略，如在第三至第五语法，会假定一个封闭的路径。

路径的输出使用第一种或第二种语法输出。

示例

将[(0,0),(1,1),(2,0)]开放的路径转换为闭合路径。

```
SELECT pclose(path '[(0,0),(1,1),(2,0)]') AS RESULT;
```

返回结果为：

```
   result
---------------------
   ((0,0),(1,1),(2,0))
(1 row)
```

多边形

多边形由一系列点代表（多边形的顶点）。多边形可以认为与闭合路径一样，但是存储方式不一样而且有自己的一套支持函数。

用下面的语法描述 polygon 的数值：

```
( ( x1 , y1 ) , ... , ( xn , yn ) )
( x1 , y1 ) , ... , ( xn , yn )
( x1 , y1 ) , ... , xn , yn )
x1 , y1 , ... , xn , yn
```

点表示多边形的端点。

多边形输出使用第一种语法。

示例

将路径转换成多边形。

```
SELECT polygon(path '((0,0),(1,1),(2,0))') AS RESULT;
```

返回结果为：

```
   result
--------------------
   ((0,0),(1,1),(2,0))
(1 row)
```

圆

圆由一个圆心和半径标识。用下面的语法描述 circle 的数值：

```
< ( x , y ) , r >
( ( x , y ) , r )
( x , y ) , r
x , y , r
```

(x,y)表示圆心，r 表示半径。

圆的输出用第一种格式。

示例

将圆转换成 12 点多边形。

```
SELECT polygon(circle '((0,0),2.0)') AS RESULT;
                result

--------------------------------------------------------------------------------
--------------------------------
```

((-2,0),(-1.73205080756888,1),(-1,1.73205080756888),(-1.22464679914735e-16,2),(1,1.73205080756888),(1.返回结果为：

```
73205080756888,1),(2
,2.44929359829471e-16),(1.73205080756888,-0.999999999999999),(1,-1.73205080
756888),(3.67394039744206e-16,-2),(-0.9999999999
99999,-1.73205080756888),(-1.73205080756888,-1))
(1 row)
```

4.1.2.8.网络地址类型

PanWeiDB 提供用于存储 IPv4、IPv6、MAC 地址的数据类型。

用这些数据类型存储网络地址比用纯文本类型好，因为这些类型提供输入错误检查和特殊的操作和功能（请参考：SQL 语法参考->SQL 基本要素->函数和操作符->网络地址函数和操作符）。

表 1 网络地址类型

名称	存储空间	描述
cidr	7 或 19 字节	IPv4 或 IPv6 网络
inet	7 或 19 字节	IPv4 或 IPv6 主机和网络
macaddr	6 字节	MAC 地址

在对 inet 或 cidr 数据类型进行排序的时候，IPv4 地址总是排在 IPv6 地址前面，包括那些封装或者是映射在 IPv6 地址里的 IPv4 地址，比如::10.2.3.4 或::ffff:10.4.3.2。

cidr

cidr（无类别域间路由，Classless Inter-Domain Routing）类型，保存一个 IPv4 或 IPv6 网络地址。声明网络格式为 address/y，address 表示 IPv4 或者 IPv6 地址，y 表示子网掩码的二进制位数。如果省略 y，则掩码部分使用已有类别的网络编号系统进行计算，但要求输入的数据已经包括了确定掩码所需的所有字节。

表 2 cidr 类型输入举例

cidr 输入	cidr 输出	abbrev（cidr）
192.168.100.128/25	192.168.100.128/25	192.168.100.128/25
192.168/24	192.168.0.0/24	192.168.0/24
192.168/25	192.168.0.0/25	192.168.0.0/25
192.168.1	192.168.1.0/24	192.168.1/24
192.168	192.168.0.0/24	192.168.0/24
10.1.2	10.1.2.0/24	10.1.2/24
10.1	10.1.0.0/16	10.1/16
10	10.0.0.0/8	10/8
10.1.2.3/32	10.1.2.3/32	10.1.2.3/32
2001:4f8:3:ba::/64	2001:4f8:3:ba::/64	2001:4f8:3:ba::/64
2001:4f8:3:ba:2e0:81ff:fe22:d1f1/128	2001:4f8:3:ba:2e0:81ff:fe22:d1f1/128	2001:4f8:3:ba:2e0:81ff:fe22:d1f1
::ffff:1.2.3.0/120	::ffff:1.2.3.0/120	::ffff:1.2.3/120
::ffff:1.2.3.0/128	::ffff:1.2.3.0/128	::ffff:1.2.3.0/128

inet

inet 类型在一个数据区域内保存主机的 IPv4 或 IPv6 地址，以及一个可选子网。主机地址中网络地址的位数表示子网（“子网掩码”）。如果子网掩码是

32 并且地址是 IPv4，则这个值不表示任何子网，只表示一台主机。在 IPv6 里，地址长度是 128 位，因此 128 位表示唯一的主机地址。

该类型的输入格式是 address/y，address 表示 IPv4 或者 IPv6 地址，y 是子网掩码的二进制位数。如果省略/y，则子网掩码对 IPv4 是 32，对 IPv6 是 128，所以该值表示只有一台主机。如果该值表示只有一台主机，/y 将不会显示。

inet 和 cidr 类型之间的基本区别是 inet 接受子网掩码，而 cidr 不接受。

macaddr

macaddr 类型存储 MAC 地址，也就是以太网卡硬件地址（尽管 MAC 地址还用于其他用途）。可以接受下列格式：

```
'08:00:2b:01:02:03'
'08-00-2b-01-02-03'
'08002b:010203'
'08002b-010203'
'0800.2b01.0203'
'08002b010203'
```

这些示例都表示同一个地址。对于数据位 a 到 f，大小写都行。输出时都是以第一种形式展示。

4.1.2.9.位串类型

位串就是一串 1 和 0 的字符串。它们可以用于存储位掩码。

PanWeiDB 支持两种位串类型：bit(n)和 bit varying(n)，这里的 n 是一个正整数。

bit 类型的数据必须准确匹配长度 n，如果存储短或者长的数据都会报错。bit varying 类型的数据是最长为 n 的变长类型，超过 n 的类型会被拒绝。一个没有长度的 bit 等效于 bit(1)，没有长度的 bit varying 表示没有长度限制。

【说明】

如果用户明确地把一个位串值转换成 bit(n)，则此位串右边的内容将被截断或者在右边补齐零，直到刚好 n 位，而不会抛出任何错误。

如果用户明确地把一个位串数值转换成 bit varying(n)，如果它超过了 n 位，则它的右边将被截断。

```
--创建表。
panweidb=# CREATE TABLE bit_type_t1
(
   BT_COL1 INTEGER,
   BT_COL2 BIT(3),
   BT_COL3 BIT VARYING(5)
) ;

--插入数据。
panweidb=# INSERT INTO bit_type_t1 VALUES(1, B'101', B'00');

--插入数据的长度不符合类型的标准会报错。
panweidb=# INSERT INTO bit_type_t1 VALUES(2, B'10', B'101');
ERROR:  bit string length 2 does not match type bit(3)
CONTEXT:  referenced column: bt_col2

--将不符合类型长度的数据进行转换。
panweidb=# INSERT INTO bit_type_t1 VALUES(2, B'10'::bit(3), B'101');

--查看数据。
panweidb=# SELECT * FROM bit_type_t1;
 bt_col1 | bt_col2 | bt_col3
---------+---------+---------
       1 | 101     | 00
       2 | 100     | 101
(2 rows)

--删除表。
panweidb=# DROP TABLE bit_type_t1;
```

4.1.2.10.文本搜索类型

PanWeiDB 提供了两种数据类型用于支持全文检索。tsvector 类型表示为文本搜索优化的文件格式，tsquery 类型表示文本查询。

tsvector

tsvector 类型表示一个检索单元，通常是一个数据库表中一行的文本字段或者这些字段的组合，tsvector 类型的值是一个标准词位的有序列表，标准词位就是把同一个词的变型体都标准化成相同的，在输入的同时会自动排序和消除重复。to_tsvector 函数通常用于解析和标准化文档字符串。

tsvector 的值是唯一分词的分类列表，把一句话的词格式化为不同的词条，在进行分词处理的时候 tsvector 会自动去掉分词中重复的词条，按照一定的顺序录入。如：

```
panweidb=#SELECT 'a fat cat sat on a mat and ate a fat rat'::tsvector;
                    tsvector
----------------------------------------------------
 'a' 'and' 'ate' 'cat' 'fat' 'mat' 'on' 'rat' 'sat'
(1 row)
```

从上面的例子可以看出，通过 tsvector 把一个字符串按照空格进行分词，分词的顺序是按照长短和字母排序的。但是如果词条中需要包含空格或标点符号，可以用引号标记：

```
panweidb=#SELECT $$the lexeme '    ' contains spaces$$::tsvector;
               tsvector
-----------------------------------------
 '    ' 'contains' 'lexeme' 'spaces' 'the'
(1 row)
```

如果在词条中使用引号，可以使用双$符号（$$）作为标记：

```
panweidb=#SELECT $$the lexeme 'Joe''s' contains a quote$$::tsvector;
                  tsvector
-----------------------------------------------
 'Joe''s' 'a' 'contains' 'lexeme' 'quote' 'the'
(1 row)
```

词条位置常量也可以放到词汇中：

```
panweidb=#SELECT 'a:1 fat:2 cat:3 sat:4 on:5 a:6 mat:7 and:8 ate:9 a:10 fat:11 rat:
12'::tsvector;
                       tsvector
-------------------------------------------------------------------------------
 'a':1,6,10 'and':8 'ate':9 'cat':3 'fat':2,11 'mat':7 'on':5 'rat':12 'sat':4
(1 row)
```

位置常量通常表示文档中源字的位置。位置信息可以用于进行排名。位置常量的范围是 1 到 16383，最大值默认是 16383。相同词的重复位会被忽略掉。

拥有位置的词汇甚至可以用一个权来标记，这个权可以是 A、B、C 或 D。默认的是 D，因此输出中不会出现：

```
panweidb=#SELECT 'a:1A fat:2B,4C cat:5D'::tsvector;
      tsvector
----------------------------
 'a':1A 'cat':5 'fat':2B,4C
(1 row)
```

权可以用来反映文档结构，如：标记标题与主体文字的区别。全文检索排序函数可以为不同的权标记分配不同的优先级。

下面的示例是 tsvector 类型标准用法。如：

```
panweidb=#SELECT 'The Fat Rats'::tsvector;
    tsvector
--------------------
 'Fat' 'Rats' 'The'
(1 row)
```

但是对于英文全文检索应用来说，上面的单词会被认为非规范化的，所以需要通过 to_tsvector 函数对这些单词进行规范化处理：

```
panweidb=#SELECT to_tsvector('english', 'The Fat Rats');
  to_tsvector
-----------------
 'fat':2 'rat':3
```

```
(1 row)
```

tsquery

tsquery 类型表示一个检索条件，存储用于检索的词汇，并且使用布尔操作符&（AND），|（OR）和!（NOT）来组合他们，括号用来强调操作符的分组。to_tsquery 函数及 plainto_tsquery 函数会将单词转换为 tsquery 类型前进行规范化处理。

```
panweidb=#SELECT 'fat & rat'::tsquery;
   tsquery
---------------
 'fat' & 'rat'
(1 row)

panweidb=#SELECT 'fat & (rat | cat)'::tsquery;
       tsquery
---------------------------
 'fat' & ( 'rat' | 'cat' )
(1 row)

panweidb=#SELECT 'fat & rat & ! cat'::tsquery;
     tsquery
-----------------------
 'fat' & 'rat' & !'cat'
(1 row)
```

在没有括号的情况下，！（非）结合的最紧密，而&（和）结合的比|（或）紧密。

tsquery 中的词汇可以用一个或多个权字母来标记，这些权字母限制这次词汇只能与带有匹配权的 tsvector 词汇进行匹配。

```
panweidb=#SELECT 'fat:ab & cat'::tsquery;
    tsquery
------------------
 'fat':AB & 'cat'
(1 row)
```

同样，tsquery 中的词汇可以用*标记来指定前缀匹配：

```
panweidb=#SELECT 'super:*'::tsquery;
 tsquery
-----------
 'super':*
(1 row)
```

这个查询可以匹配 tsvector 中以“super”开始的任意单词。

请注意，前缀首先被文本搜索分词器处理，这也就意味着下面的结果为真：

```
panweidb=#SELECT to_tsvector( 'postgraduate' ) @@ to_tsquery( 'postgres:*' ) AS R
ESULT;
 result
----------
 t
(1 row)
```

因为 postgres 经过处理后得到 postgr：

```
panweidb=#SELECT to_tsquery('postgres:*');
 to_tsquery
------------
 'postgr':*
(1 row)
```

这样就匹配 postgraduate 了。

'Fat:ab & Cats'规范化转为 tsquery 类型结果如下：

```
panweidb=#SELECT to_tsquery('Fat:ab & Cats');
    to_tsquery
------------------
 'fat':AB & 'cat'
(1 row)
```

4.1.2.11.UUID 类型

UUID 数据类型用来存储 RFC 4122，ISO/IEF 9834-8:2005 以及相关标准定义的通用唯一标识符（UUID）。这个标识符是一个由算法产生的 128 位标识符，确保它不可能使用相同算法在已知的模块中产生的相同标识符。

UUID 是一个小写十六进制数字的序列，由分字符分成几组，一组 8 位数字+三组 4 位数字+一组 12 位数字，总共 32 个数字代表 128 位，标准的 UUID 示例如下：

```
a0eebc99-9c0b-4ef8-bb6d-6bb9bd380a11
```

PanWeiDB 同样支持以其他方式输入：大写字母和数字、由花括号包围的标准格式、省略部分或所有连字符、在任意一组四位数字之后加一个连字符。示例：

```
A0EEBC99-9C0B-4EF8-BB6D-6BB9BD380A11
{a0eebc99-9c0b-4ef8-bb6d-6bb9bd380a11}
a0eebc999c0b4ef8bb6d6bb9bd380a11
a0ee-bc99-9c0b-4ef8-bb6d-6bb9-bd38-0a11
```

一般是以标准格式输出。

4.1.2.12.HLL 数据类型

HLL（HyperLoglog）是统计数据集中唯一值个数的高效近似算法。它有着计算速度快、节省空间的特点，不需要直接存储集合本身，而是存储一种名为 HLL 的数据结构。每当有新数据加入进行统计时，只需要把数据经过哈希计算并插入到 HLL 中，最后根据 HLL 就可以得到结果。

HLL 与其他算法的比较请参见表 1。

表 1 HLL 与其他算法比较

项目	Sort 算法	Hash 算法	HLL
时间复杂度	O(nlogn)	O(n)	O(n)
空间复杂度	O(n)	O(n)	log(logn)
误差率	0	0	≈0.8%

项目	Sort 算法	Hash 算法	HLL
所需存储空间	原始数据大小	原始数据大小	默认规格下最大 16KB

HLL 在计算速度和所占存储空间上都占优势。在时间复杂度上，Sort 算法需要排序至少 O(nlogn)的时间，虽说 Hash 算法和 HLL 一样扫描一次全表 O(n)的时间就可以得出结果，但是存储空间上，Sort 算法和 Hash 算法都需要先把原始数据存起来再进行统计，会导致存储空间消耗巨大，而对 HLL 来说不需要存原始数据，只需要维护 HLL 数据结构，故占用空间有很大的压缩，默认规格下 HLL 数据结构的最大空间约为 16KB。

【须知】

- 当前默认规格下可计算最大 distinct 值的数量约为 1.1e+15 个，误差率为 0.8%。用户应注意如果计算结果超过当前规格下 distinct 最大值会导致计算结果误差率变大，或导致计算结果失败并报错。
- 用户在首次使用该特性时，应该对业务的 distinct value 做评估，选取适当的配置参数并做验证，以确保精度符合要求：
- 当前默认参数下，可以计算的 distinct 值为 1.1e+15，如果计算得到的 distinct 值为 NaN，需要调整 log2m，或者采用其他算法计算 distinct 值。
- 虽然 hash 算法存在极低的 hash collision 概率，但是建议用户在首次使用时，选取 2-3 个 hash seed 验证，如果得到的 distinct value 相差不大，则可以从该组 seed 中任选一个作为 hash seed。

HLL 中主要的数据结构，请参见表 2。

表 2 HyperLogLog 中主要数据结构

数据类型	功能描述
hll	hll 头部为 27 字节长度字段，默认规格下数据段长度 0~16KB，可直接计算得到 distinct 值。

创建 HLL 数据类型时，可以支持 0~4 个参数入参，具体的参数含义与参数规格同函数 hll_empty 一致。第一个参数为 log2m，表示分桶数的对数值，取值范围 10~16；第二个参数为 log2explicit，表示 Explicit 模式的阈值大小，

取值范围 0~12；第三个参数为 log2sparse，表示 Sparse 模式的阈值大小，取值范围 0~14；第四个参数为 duplicatecheck，表示是否启用 duplicatecheck，取值范围为 0~1。当入参输入值为-1 时，会采用默认值设定 HLL 的参数。可以通过 d 或 d+查看 HLL 类型的参数。

【说明】

- 创建 HLL 数据类型时，根据入参的行为不同，结果不同：
- 创建 HLL 类型时对应入参不输入或输入-1，采用默认值设定对应的 HLL 参数。
- 输入合法范围的入参，对应 HLL 参数采用输入值。
- 输入不合法范围的入参，创建 HLL 类型报错。

```
-- 创建 hll 类型的表，不指定入参
panweidb=# create table t1 (id integer, set hll);
panweidb=# /d t1
    Table "public.t1"
 Column | Type   | Modifiers
--------+---------+-----------
 id     | integer |
 set    | hll     |

-- 创建 hll 类型的表，指定前两个入参，后两个采用默认值
panweidb=# create table t2 (id integer, set hll(12,4));
panweidb=# /d t2
       Table "public.t2"
 Column |     Type      | Modifiers
--------+---------------+-----------
 id     | integer       |
 set    | hll(12,4,12,0) |

--创建 hll 类型的表，指定第三个入参，其余采用默认值
panweidb=# create table t3(id int, set hll(-1,-1,8,-1));
panweidb=# /d t3
       Table "public.t3"
 Column |     Type      | Modifiers
```

```
--------+---------------+-----------
 id     | integer       |
 set    | hll(14,10,8,0) |

--创建 hll 类型的表，指定入参不合法报错
panweidb=# create table t4(id int, set hll(5,-1));
ERROR:  log2m = 5 is out of range, it should be in range 10 to 16, or set -1 as defaul
t
```

对含有 HLL 类型的表插入 HLL 对象时，HLL 类型的设定参数须同插入对象的设定参数一致，否则报错。

```
-- 创建带有 hll 类型的表
panweidb=# create table t1(id integer, set hll(14));

-- 向表中插入 hll 对象,参数一致，成功
panweidb=# insert into t1 values (1, hll_empty(14,-1));

-- 向表中插入 hll 对象，参数不一致，失败
panweidb=# insert into t1(id, set) values (1, hll_empty(14,5));
ERROR:  log2explicit does not match: source is 5 and dest is 10
```

HLL 的应用场景。

❖ 场景 1：“Hello World”

通过下面的示例说明如何使用 hll 数据类型：

```
-- 创建带有 hll 类型的表
panweidb=# create table helloworld (id integer, set hll);

-- 向表中插入空的 hll
panweidb=# insert into helloworld(id, set) values (1, hll_empty());

-- 把整数经过哈希计算加入到 hll 中
panweidb=# update helloworld set set = hll_add(set, hll_hash_integer(12345)) whe
re id = 1;

-- 把字符串经过哈希计算加入到 hll 中
```

```
panweidb=# update helloworld set set = hll_add(set, hll_hash_text('hello world'))
where id = 1;

-- 得到 hll 中的 distinct 值
panweidb=# select hll_cardinality(set) from helloworld where id = 1;
  hll_cardinality
 -----------------
        2
(1 row)

-- 删除表
panweidb=#  drop table helloworld;
```

❖ 场景 2：“网站访客数量统计”

通过下面的示例说明 hll 如何统计在一段时间内访问网站的不同用户数量：

```
-- 创建原始数据表，表示某个用户在某个时间访问过网站。
panweidb=# create table facts (
          date          date,
          user_id       integer
   );

   -- 构造数据，表示一天中有哪些用户访问过网站。
panweidb=# insert into facts values ('2019-02-20', generate_series(1,100));
panweidb=# insert into facts values ('2019-02-21', generate_series(1,200));
panweidb=# insert into facts values ('2019-02-22', generate_series(1,300));
panweidb=# insert into facts values ('2019-02-23', generate_series(1,400));
panweidb=# insert into facts values ('2019-02-24', generate_series(1,500));
panweidb=# insert into facts values ('2019-02-25', generate_series(1,600));
panweidb=# insert into facts values ('2019-02-26', generate_series(1,700));
panweidb=# insert into facts values ('2019-02-27', generate_series(1,800));

-- 创建表并指定列为 hll。
panweidb=# create table daily_uniques (
      date          date UNIQUE,
      users         hll
   );
```

```
-- 根据日期把数据分组，并把数据插入到 hll 中。
panweidb=# insert into daily_uniques(date, users)
 select date, hll_add_agg(hll_hash_integer(user_id))
     from facts
     group by 1;

-- 计算每一天访问网站不同用户数量
panweidb=# select date, hll_cardinality(users) from daily_uniques order by date;
     date    | hll_cardinality
  ------------+------------------
   2019-02-20 |              100
   2019-02-21 | 200.217913059312
   2019-02-22 |  301.76494508014
   2019-02-23 | 400.862858326446
   2019-02-24 | 502.626933349694
   2019-02-25 | 601.922606454213
   2019-02-26 | 696.602316769498
   2019-02-27 | 798.111731634412
  (8 rows)

-- 计算在 2019.02.20 到 2019.02.26 一周中有多少不同用户访问过网站
panweidb=# select hll_cardinality(hll_union_agg(users)) from daily_uniques where
 date >= '2019-02-20'::date and date <= '2019-02-26'::date;
   hll_cardinality
  ------------------
   702.941844662509
 (1 row)

-- 计算昨天访问过网站而今天没访问网站的用户数量。
panweidb=# SELECT date, (#hll_union_agg(users) OVER two_days) - #users AS lost_
uniques FROM daily_uniques WINDOW two_days AS (ORDER BY date ASC ROWS 1 P
RECEDING);
     date    | lost_uniques
  ------------+--------------
```

```
  2019-02-20|       0
  2019-02-21|       0
  2019-02-22|       0
  2019-02-23|       0
  2019-02-24|       0
  2019-02-25|       0
  2019-02-26|       0
  2019-02-27|       0
 (8 rows)

-- 删除表
panweidb=# drop table facts;
panweidb=# drop table daily_uniques;
```

场景 3：“插入数据不满足 hll 数据结构要求”

当用户给 hll 类型的字段插入数据的时候，必须保证插入的数据满足 hll 数据结构要求，如果解析后不满足就会报错。如下示例中： 插入数据'E1234'时，该数据不满足 hll 数据结构，不能解析成功因此失败报错。

```
panweidb=# create table test(id integer, set hll);
panweidb=# insert into test values(1, 'E1234');
ERROR:  not a hll type, size=6 is not enough
panweidb=# drop table test;
```

4.1.2.13.范围类型

范围类型是表达某种元素类型（称为范围的 *subtype*）的一个值的范围的数据类型。例如，timestamp 的范围可以被用来表达一个会议室被保留的时间范围。在这种情况下，数据类型是 tsrange（“timestamp range”的简写）而 timestamp 是 subtype。subtype 必须具有一种总体的顺序，这样对于元素值是在一个范围值之内、之前或之后就是界线清楚的。

范围类型非常有用，因为它们可以表达一种单一范围值中的多个元素值，并且可以很清晰地表达诸如范围重叠等概念。用于时间安排的时间和日期范围是最清晰的例子；但是价格范围、一种仪器的量程等等也都有用。

内建范围类型

有下列内建范围类型：

- int4range – integer 的范围
- int8range – bigint 的范围
- numrange – numeric 的范围
- tsrange – 不带时区的 timestamp 的范围
- tstzrange – 带时区的 timestamp 的范围
- daterange – date 的范围

此外，你可以定义自己的范围类型，参考：SQL 语法参考->SQL 语法->CREATE TYPE 章节。

例子

```
CREATE TABLE reservation (room int, during tsrange);
INSERT INTO reservation VALUES (1108, '[2010-01-01 14:30, 2010-01-01 15:30)');
-- 包含
SELECT int4range(10, 20) @> 3;
-- 重叠
SELECT numrange(11.1, 22.2) && numrange(20.0, 30.0);
-- 抽取上界
SELECT upper(int8range(15, 25));
-- 计算交集
SELECT int4range(10, 20) * int4range(15, 25);
-- 范围为空吗?
SELECT isempty(numrange(1, 5));
```

范围类型上的操作符和函数的完整列表可参考：SQL 语法参考->SQL 基本要素->函数和操作符->范围函数和操作符章节。

包含和排除边界

每一个非空范围都有两个界限，下界和上界。上下界之间的所有值都被包括在范围内。一个包含界限意味着边界点本身也被包括在范围内，而一个排除边界意味着边界点不被包括在范围内。

在一个范围的文本形式中，一个包含下界被表达为“[”而一个排除下界被表达为“(”。同样，一个包含上界被表达为“]”而一个排除上界被表达为“)”（详见"范围输入/输出"）。

函数 lower_inc 和 upper_inc 分别测试一个范围值的上下界。

无限（无界）范围

一个范围的下界可以被忽略，意味着所有小于上界的值都被包括在范围中，例如(,3]。 同样，如果范围的上界被忽略，那么所有比上界大的值都被包括在范围中。如果上下界都被忽略，该元素类型的所有值都被认为在该范围中。 规定缺失的包括界限自动转换为排除，例如，[,] 转换为 (,)。 你可以认为这些缺失值为 +/- 无穷大，但它们是特殊范围类型值，并且被视为超出任何范围元素类型的 +/- 无穷大值。

具有“infinity”概念的元素类型可以用它们作为显式边界值。例如，在时间戳范围，[today,infinity)不包括特殊的 timestamp 值 infinity，尽管 [today,infinity] 包括它，就好比 [today,) 和 [today,]。

函数 lower_inf 和 upper_inf 分别测试一个范围的无限上下界。

范围输入/输出

一个范围值的输入必须遵循下列模式之一：

```
(lower-bound,upper-bound)
(lower-bound,upper-bound]
[lower-bound,upper-bound)
[lower-bound,upper-bound]
empty
```

圆括号或方括号指示上下界是否为排除的或者包含的。注意最后一个模式是 empty，它表示一个空范围（一个不包含点的范围）。

lower-bound 可以是作为 subtype 的合法输入的一个字符串，或者是空表示没有下界。同样，upper-bound 可以是作为 subtype 的合法输入的一个字符串，或者是空表示没有上界。

每个界限值可以使用"（双引号）字符引用。如果界限值包含圆括号、方括号、逗号、双引号或反斜线时，这样做是必须的，因为否则那些字符会被认作范围

语法的一部分。要把一个双引号或反斜线放在一个被引用的界限值中，就在它前面放一个反斜线（还有，在一个双引号引用的界限值中的一对双引号表示一个双引号字符，这与 SQL 字符串中的单引号规则类似）。此外，你可以避免引用并且使用反斜线转义来保护所有数据字符，否则它们会被当做返回语法的一部分。还有，要写一个是空字符串的界限值，则可以写成""，因为什么都不写表示一个无限界限。

范围值前后允许有空格，但是圆括号或方括号之间的任何空格会被当做上下界值的一部分（取决于元素类型，它可能是也可能不是有意义的）。

例子：

```
-- 包括 3，不包括 7，并且包括 3 和 7 之间的所有点
SELECT '[3,7)'::int4range;
-- 既不包括 3 也不包括 7，但是包括之间的所有点
SELECT '(3,7)'::int4range;
-- 只包括单独一个点 4
SELECT '[4,4]'::int4range;
-- 不包括点（并且将被标准化为 '空'）
SELECT '[4,4)'::int4range;
```

构造范围

每一种范围类型都有一个与其同名的构造器函数。使用构造器函数常常比写一个范围文字常数更方便，因为它避免了对界限值的额外引用。构造器函数接受两个或三个参数。两个参数的形式以标准的形式构造一个范围（下界是包含的，上界是排除的），而三个参数的形式按照第三个参数指定的界限形式构造一个范围。第三个参数必须是下列字符串之一：“()”、“(]”、“[)”或者“[]”。例如：

```
-- 完整形式是：下界、上界以及指示界限包含性/排除性的文本参数。
SELECT numrange(1.0, 14.0, '(]');
-- 如果第三个参数被忽略，则假定为 '[)'。
SELECT numrange(1.0, 14.0);
-- 尽管这里指定了 '(]'，显示时该值将被转换成标准形式，因为 int8range 是一种离散范围类型（见下文）。
SELECT int8range(1, 14, '(]');
```

```
-- 为一个界限使用 NULL 导致范围在那一边是无界的。
SELECT numrange(NULL, 2.2);
```

离散范围类型

一种范围的元素类型具有一个良定义的“步长”，例如 integer 或 date。在这些类型中，如果两个元素之间没有合法值，它们可以被说成是相邻。这与连续范围相反，连续范围中总是（或者几乎总是）可以在两个给定值之间标识其他元素值。例如，numeric 类型之上的一个范围就是连续的，timestamp 上的范围也是（尽管 timestamp 具有有限的精度，并且在理论上可以被当做离散的，最好认为它是连续的，因为通常并不关心它的步长）。

另一种考虑离散范围类型的方法是对每一个元素值都有一种清晰的“下一个”或“上一个”值。了解了这种思想之后，通过选择原来给定的下一个或上一个元素值来取代它，就可以在一个范围界限的包含和排除表达之间转换。例如，在一个整数范围类型中，[4,8]和(3,9)表示相同的值集合，但是对于 numeric 上的范围就不是这样。

一个离散范围类型应该具有一个*正规化*函数，它知道元素类型期望的步长。正规化函数负责把范围类型的相等值转换成具有相同的表达，特别是与包含或者排除界限一致。如果没有指定一个正规化函数，那么具有不同格式的范围将总是会被当作不等，即使它们实际上是表达相同的一组值。

内建的范围类型 int4range、int8range 和 daterange 都使用一种正规的形式，该形式包括下界并且排除上界，也就是[)。不过，用户定义的范围类型可以使用其他习惯。

定义新的范围类型

用户可以定义他们自己的范围类型。这样做最常见的原因是为了使用内建范围类型中没有提供的 subtype 上的范围。例如，要创建一个 subtype float8 的范围类型：

```
CREATE TYPE floatrange AS RANGE (
    subtype = float8,
    subtype_diff = float8mi
);
SELECT '[1.234, 5.678]'::floatrange;
```

因为 float8 没有有意义的“步长”，我们在这个例子中没有定义一个正规化函数。

定义自己的范围类型也允许你指定使用一个不同的子类型 B-树操作符类或者集合， 以便更改排序顺序来决定哪些值会落入到给定的范围中。

如果 subtype 被认为是具有离散值而不是连续值，CREATE TYPE 命令应当指定一个 canonical 函数。正规化函数接收一个输入的范围值，并且必须返回一个可能具有不同界限和格式的等价的范围值。对于两个表示相同值集合的范围（例如[1, 7]和[1, 8)），正规的输出必须一样。选择哪一种表达作为正规的没有关系，只要两个具有不同格式的等价值总是能被映射到具有相同格式的相同值就行。除了调整包含/排除界限格式外，假使期望的补偿比 subtype 能够存储的要大，一个正规化函数可能会舍入边界值。例如，一个 timestamp 之上的范围类型可能被定义为具有一个一小时的步长，这样正规化函数可能需要对不是一小时的倍数的界限进行舍入，或者可能直接抛出一个错误。

另外，任何打算要和 GiST 或 SP-GiST 索引一起使用的范围类型应当定一个 subtype 差异或 subtype_diff 函数（没有 subtype_diff 时索引仍然能工作，但是可能效率不如提供了差异函数时高）。subtype 差异函数采用两个 subtype 输入值，并且返回表示为一个 float8 值的差（即 *X* 减 *Y*）。在我们上面的例子中，可以使用常规 float8 减法操作符之下的函数。但是对于任何其他 subtype，可能需要某种类型转换。还可能需要一些关于如何把差异表达为数字的创新型想法。为了最大的可扩展性，subtype_diff 函数应该同意选中的操作符类和排序规则所蕴含的排序顺序，也就是说，只要它的第一个参数根据排序顺序大于第二个参数，它的结果就应该是正值。

subtype_diff 函数的一个不那么过度简化的例子：

```
CREATE FUNCTION time_subtype_diff(x time, y time) RETURNS float8 AS 'SELECT EXT
RACT(EPOCH FROM (x - y))' LANGUAGE sql STRICT IMMUTABLE;
CREATE TYPE timerange AS RANGE (
    subtype = time,
    subtype_diff = time_subtype_diff
);
SELECT '[11:10, 23:00]'::timerange;
```

更多关于创建范围类型的信息请参考：SQL 语法参考->SQL 语法->CREATE TYPE 章节。

索引

可以为范围类型的表列创建 GiST 和 SP-GiST 索引。例如，要创建一个 GiST 索引：

```
CREATE INDEX reservation_idx ON reservation USING GIST (during);
```

一个 GiST 或 SP-GiST 索引可以加速涉及以下范围操作符的查询： =、 &&、 <@、 @>、 <<、 >>、 -|-、 &<以及 &> （参考：SQL 语法参考->SQL 基本要素->函数和操作符->范围函数和操作符章节）。

此外，B-树和哈希索引可以在范围类型的表列上创建。对于这些索引类型，基本上唯一有用的范围操作就是等值。使用相应的< 和 >操作符，对于范围值定义有一种 B-树排序顺序，但是该顺序相当任意并且在真实世界中通常不怎么有用。范围类型的 B-树和哈希支持主要是为了允许在查询内部进行排序和哈希，而不是创建真正的索引。

4.1.2.14.对象标识符类型

PanWeiDB 在内部使用对象标识符（OID）作为各种系统表的主键。系统不会给用户创建的表增加一个 OID 系统字段，OID 类型代表一个对象标识符。

目前 OID 类型用一个四字节的无符号整数实现。因此不建议在创建的表中使用 OID 字段做主键。

表 1 对象标识符类型

名称	引用	描述	示例
OID	-	数字化的对象标识符。	564182
CID	-	命令标识符。它是系统字段 cmin 和 cmax 的数据类型。命令标识符是 32 位的量。	-
XID	-	事务标识符。它是系	-

名称	引用	描述	示例
		统字段 xmin 和 xmax 的数据类型。事务标识符也是 64 位的量。	
TID	-	行标识符。它是系统表字段 ctid 的数据类型。行 ID 是一对数值（块号，块内的行索引），它标识该行在其所在表内的物理位置。	-
REGCONFIG	pg_ts_config	文本搜索配置。	english
REGDICTIONARY	pg_ts_dict	文本搜索字典。	simple
REGOPER	pg_operator	操作符名。	-
REGOPERATOR	pg_operator	带参数类型的操作符。	*(integer,integer) 或-(NONE,integer)
REGPROC	pg_proc	函数名称。	sum
REGPROCEDURE	pg_proc	带参数类型的函数。	sum(int4)
REGCLASS	pg_class	关系名。	pg_type
REGTYPE	pg_type	数据类型名。	integer

OID 类型：主要作为数据库系统表中字段使用。

示例：

```
panweidb=# SELECT oid FROM pg_class WHERE relname = 'pg_type';
 oid
------
 1247
(1 row)
```

OID 别名类型 REGCLASS：主要用于对象 OID 值的简化查找。

示例：

```
panweidb=# SELECT attrelid,attname,atttypid,attstattarget FROM pg_attribute WHE
RE attrelid = 'pg_type'::REGCLASS;
 attrelid |  attname   | atttypid | attstattarget
----------+------------+----------+---------------
     1247 | xc_node_id |       23 |             0
     1247 | tableoid   |       26 |             0
     1247 | cmax       |       29 |             0
     1247 | xmax       |       28 |             0
     1247 | cmin       |       29 |             0
     1247 | xmin       |       28 |             0
     1247 | oid        |       26 |             0
     1247 | ctid       |       27 |             0
     1247 | typname    |       19 |            -1
     1247 | typnamespace |     26 |            -1
     1247 | typowner   |       26 |            -1
     1247 | typlen     |       21 |            -1
     1247 | typbyval   |       16 |            -1
     1247 | typtype    |       18 |            -1
     1247 | typcategory |      18 |            -1
     1247 | typispreferred |   16 |            -1
     1247 | typisdefined |     16 |            -1
     1247 | typdelim   |       18 |            -1
     1247 | typrelid   |       26 |            -1
     1247 | typelem    |       26 |            -1
     1247 | typarray   |       26 |            -1
     1247 | typinput   |       24 |            -1
     1247 | typoutput  |       24 |            -1
     1247 | typreceive |       24 |            -1
     1247 | typsend    |       24 |            -1
     1247 | typmodin   |       24 |            -1
     1247 | typmodout  |       24 |            -1
     1247 | typanalyze |       24 |            -1
     1247 | typalign   |       18 |            -1
     1247 | typstorage |       18 |            -1
     1247 | typnotnull |       16 |            -1
```

```
  1247 | typbasetype   |     26 |        -1
  1247 | typtypmod     |     23 |        -1
  1247 | typndims      |     23 |        -1
  1247 | typcollation  |     26 |        -1
  1247 | typdefaultbin |    194 |        -1
  1247 | typdefault    |     25 |        -1
  1247 | typacl        |   1034 |        -1
(38 rows)
```

4.1.2.15.伪类型

PanWeiDB 数据类型中包含一系列特殊用途的类型，这些类型按照类别被称为伪类型。伪类型不能作为字段的数据类型，但是可以用于声明函数的参数或者结果类型。

当一个函数不仅是简单地接受并返回某种 SQL 数据类型的情况下伪类型是很有用的。表 1 列出了所有的伪类型。

表 1 伪类型

名称	描述
any	表示函数接受任何输入数据类型。
anyelement	表示函数接受任何数据类型。
anyarray	表示函数接受任意数组数据类型。
anynonarray	表示函数接受任意非数组数据类型。
anyenum	表示函数接受任意枚举数据类型。
anyrange	表示函数接受任意范围数据类型。
cstring	表示函数接受或者返回一个空结尾的 C 字符串。
internal	表示函数接受或者返回一种服务器内部的数据类型。
language_handler	声明一个过程语言调用句柄返回 language_handler。
fdw_handler	声明一个外部数据封装器返回 fdw_handler。

名称	描述
record	标识函数返回一个未声明的行类型。
trigger	声明一个触发器函数返回 trigger。
void	表示函数不返回数值。
opaque	一个已经过时的类型，以前用于所有上面这些用途。

声明用 C 编写的函数（不管是内置的还是动态装载的）都可以接受或者返回任何这样的伪数据类型。当伪类型作为参数类型使用时，用户需要保证函数的正常运行。

用过程语言编写的函数只能使用实现语言允许的伪类型。目前，过程语言都不允许使用作为参数类型的伪类型，并且只允许使用 void 和 record 作为结果类型。一些多态的函数还支持使用 anyelement、anyarray、anynonarray anyenum 和 anyrange 类型。

每一个被声明为 anyelement 的位置（参数或返回值）被允许具有任意特定的实际数据类型，但是再任何给定的查询中他们必须全部是相同的实际类型。

伪类型 internal 用于声明那种只能在数据库系统内部调用的函数，他们不能直接在 SQL 查询里调用。如果函数至少有一个 internal 类型的参数，则不能从 SQL 里调用他。建议不要创建任何声明返回 internal 的函数，除非他至少有一个 internal 类型的参数。

示例：

```
--创建表
panweidb=# create table t1 (a int);

--插入两条数据
panweidb=# insert into t1 values(1),(2);

--创建函数 showall()。
panweidb=# CREATE OR REPLACE FUNCTION showall() RETURNS SETOF record
AS $$ SELECT count(*) from t1; $$
LANGUAGE SQL;
```

```
--调用函数 showall()。
panweidb=# SELECT showall();
 showall
---------
 (2)
(1 row)

--删除函数。
panweidb=# DROP FUNCTION showall();

--删除表
panweidb=# drop table t1;
```

4.1.2.16.XML 类型

XML 数据类型可以被用来存储 XML 数据。它的内部格式和 TEXT 类型相同，它比直接在一个 TEXT 域中存储 XML 数据的优势在于：它会使用 LIBXML2 对于 XML 格式文本的处理能力，检查输入值的结构是不是符合 XML 标准，并且基于 LIBXML2 提供了支持函数用于在其上执行类型安全的操作。

XML 类型可以存储格式良好的遵循 XML 标准定义的“文档”、以及“内容”片段，它是通过引用更宽泛的“DOCUMENT NODE” XQUERY 和 XPATH 数据模型来定义的。大致上说，这意味着内容片段中可以有多于一个的顶层元素或字符节点。表达式 XMLVALUE IS DOCUMENT 可以被用来评估一个特定的 XML 值是一个完整文档或者仅仅是一个文档片段。

XML 解析器把 XML 文档转换为 XML DOM 对象。DOM（DOCUMENT OBJECT MODEL 文档对象模型）定义了访问和操作文档的标准方法。XML DOM（XML DOCUMENT OBJECT MODEL）定义了访问和操作 XML 文档的标准方法。XML DOM 把 XML 文档作为树结构来查看。所有元素可以通过 DOM 树来访问。可以修改或删除它们的内容，并创建新的元素。元素，它们的文本，以及它们的属性，都被认为是节点。

XML 底层使用和 text 类型一样的数据结构进行存储，最大为 1GB。

示例：

1、 创建带 xml 数据类型的测试表。

```
CREATE TABLE xmltest ( id int, data xml );
```

2、向测试表中插入数据。

```
INSERT INTO xmltest VALUES (1, '<value>one</value>');
INSERT INTO xmltest VALUES (2, '<value>two</value>');
```

3、查询测试表记录。

```
SELECT * FROM xmltest ORDER BY 1;
SELECT xmlconcat(xmlcomment('hello'),  xmlelement(NAME qux, 'xml'),
 xmlcomment('world'));
```

结果分别显示为：

```
 id | data
----+--------------------
1 | <value>one</value>
2 | <value>two</value>
(2 rows)

          xmlconcat
--------------------------------------
 <!--hello--><qux>xml</qux><!--world-->
(1 row)
```

- **XML 类型不支持如下操作：**
  - 逻辑表达式 and、or、not。
  - 作为分区键、二级分区键、外键、主键、唯一约束。
  - XML 相关的隐式类型转换（赋值时 XML 类型可隐式转换为字符类型）。
  - 数组表达式、行表达式、子查询表达式[not]in/any/all。
  - XML 类型字段上创建普通索引、unique 索引、global 索引、local 索引、部分索引。

- 比较表达式>、<、>=、<=、=、<>、!=、^=、between and、is distinct from、is not distinct from、<=>。
- 条件表达式 decode、nullif、greatest、least。
- 作为 distinct/group by/order by 参数。
- 聚合函数 sum、max、min、avg、list_agg、corr、covar_pop、covar_samp、stddev、stddev_pop、stddev_samp、var_pop、var_samp、variance、bit_and、bit_or、bool_and、bool_or、every、regr_avgx、regr_avgy、regr_count、regr_intercept、regr_r2、regr_slope、regr_sxx、regr_sxy、regr_syy。
- 不支持 ODBC 相关绑定传参接口。

❖ **XML 类型支持如下操作：**

- 物理备份恢复。
- 比较表达式 is null、is not null。
- 子查询表达式[not]exists。
- 条件表达式 case、coalesce。
- 全局临时表和本地临时表。
- 强制类型转换。
- 表达式索引。
- XML 类型的值是根据"xmloption"会话配置参数决定的。
- 支持 pw_dump 导出和 pw_restore 导入操作。
- 并行查询，支持 astore 和 ustore 存储引擎。
- 作为自定义函数的入参，出参，自定义变量，返回值。
- 作为存储过程的入参，出参，自定义变量，返回值。支持自治事务的存储过程。
- 字符处理函数 quote_literal(value anyelement)、quote_nullable(value anyelement)。
- 聚集函数 count、array_agg、checksum（需显式转换为字符类型）、string_agg（需显式转换为字符类型）。
- 自定义类型中复合类型涉及 XML type 时的增删改查，且与普通表中的 XML 字段一样，需要按照 XML 的语法插入和修改。

- 支持 JDBC 和 ODBC 支持对 XML 数据类型操作，支持对该字段进行 select，update，insert，delete，使用 SQL 语法输入 XML 值，使用 ResultSet 类的 getSQLXML 方法获取 XML 值。支持 JDBC 相关绑定传参接口、可使用 PreparedStatement 预处理语句接口中的 setSQLXML 方法、和 ResultSet 执行结果集接口中的 getSQLXML(int columnIndex)方法。

❖ **调用流程**

需要使用 java.sql.SQLXML 接口类构造 XML 对象，再设置指定的对象类型为 Oid.XML，然后将类型 ID 和 XML 值发送到服务端，从服务端获取到返回结果后，先调用 ResultSet.getString，然后通过获取的字符串使用 java.sql.SQLXML 接口类构造 XML 对象，此时会再次检查内容是否符合 XML 标准格式。所以 xml 值也可以使用 ResultSet.getString 直接获取 XML 的字符串对象。

4.1.2.17.账本数据库使用的数据类型

账本数据库使用 HASH16 数据类型来存储行级 hash 摘要或表级 hash 摘要，使用 HASH32 数据类型来存储全局 hash 摘要或者历史表校验 hash。

表 1 账本数据库 HASH 类型

名称	描述	存储空间	范围
HASH16	以无符号 64 位整数存储。	8 字节	0 ~ +18446744073709551615
HASH32	以包含 16 个的无符号整形元素数的组存储。	16 字节	16 个元素的无符号整形数组能够包含的取值范围

HASH16 数据类型用来在账本数据库中存储行级或表级 hash 摘要，在获得长度为 16 个字符串的十六进制字符串的 hash 序列后，系统将调用 hash16in 函数将该序列转换为一个无符号 64 位整数存储进 HASH16 类型变量中。示例如下：

```
十六进制字符串：e697da2eaa3a775b
对应的无符号 64 位整数：16615989244166043483
十六进制字符串：ffffffffffffffff
```

```
对应的无符号 64 位整数：18446744073709551615
```

HASH32 数据类型用来在账本数据库中存储全局 hash 摘要或者历史表校验 hash，在获得长度为 32 个字符串的十六进制字符串的 hash 序列后，系统将调用 hash32in 函数将该序列转换到一个包含 16 个无符号整形元素的数组中。示例如下：

```
十六进制字符串：685847ed1fe38e18f6b0e2b18c00edee
对应的 HASH32 数组：[104,88,71,237,31,227,142,24,246,176,226,177,140,0,237,238]
```

4.1.2.18.事务 ID 和快照数据类型 txid_snapshot

功能描述

PanWeiDB 通过 txid_snapshot 类型记录事务 ID 和快照信息。

语法格式

txid_snapshot 的文本表示为：

```
xmin:xmax:xip_list
```

参数解释

名称	描述
xmin	仍然活动的最早的事务 ID（txid）。所有较早事务将是已经提交可见的，或者是直接回滚。
xmax	作为尚未分配的 txid。所有大于或等于此 txids 的都是尚未开始的快照时间，因此不可见。
xip_list	当前快照中活动的 txids。这个列表只包含在 xmin 和 xmax 之间活动的 txids；有可能活动的 txids 高于 xmax。介于大于等于 xmin、小于 xmax，并且不在这个列表中的 txid，在这个时间快照已经完成的，因此按照提交状态查看他是可见还是回滚。这个列表不包含子事务的 txids。

示例

获取当前快照

```
select txid_current_snapshot();
```

结果返回如下：

```
txid_current_snapshot
----------------------
25091:25101:
(1 row)
```

【说明】

txid_current_snapshot()函数返回类型为 txid_snapshot。

4.1.2.19.JSON/JSONB 类型

JSON（JavaScript Object Notation）数据，可以是单独的一个标量，也可以是一个数组，也可以是一个键值对象，其中数组和对象可以统称容器（container）：

- 标量（scalar）：单一的数字、bool、string、null 都可以叫做标量。
- 数组（array）：[]结构，里面存放的元素可以是任意类型的 JSON，并且不要求数组内所有元素都是同一类型。
- 对象（object）：{}结构，存储 key:value 的键值对，其键只能是用“”包裹起来的字符串，值可以是任意类型的 JSON，对于重复的键，按最后一个键值对为准。

PanWeiDB 内存在两种数据类型 JSON 和 JSONB，可以用来存储 JSON 数据。其中 JSON 是对输入的字符串的完整拷贝，使用时再去解析，所以它会保留输入的空格、重复键以及顺序等；JSONB 解析输入后保存的二进制，它在解析时会删除语义无关的细节和重复的键，对键值也会进行排序，使用时不用再次解析。

因此可以发现，两者其实都是 JSON，它们接受相同的字符串作为输入。它们实际的主要差别是效率。JSON 数据类型存储输入文本的精确拷贝，处理函数必须在每个执行上重新解析； 而 JSONB 数据以分解的二进制格式存储，这使得它由于添加了转换机制而在输入上稍微慢些，但是在处理上明显更快，因为不需要重新解析。同时由于 JSONB 类型存在解析后的格式归一化等操作，同等的语义下只会有一种格式，因此可以更好更强大的支持很多其他额外的操作，比如按照一定的规则进行大小比较等。JSONB 也支持索引，这也是一个明显的优势。

输入格式示例

输入必须是一个符合 JSON 数据格式的字符串，此字符串用单引号 ' ' 声明。

❖ null（null-json）：仅 null，全小写。

```
select 'null'::json;
```

结果显示如下：

```
 json
--------------
 null
(1 row)
```

❖ 数字（num-json）：正负整数、小数、0，支持科学计数法，不支持多余的前导 0，正数的+号，以及 NaN 和 infinity。

```
select '1'::json,'-1.5'::json,'-1.5e-5'::jsonb, '-1.5e+2'::jsonb;
```

结果显示如下：

```
 json | json |  jsonb   | jsonb
------+------+----------+-------
 1    | -1.5 | -.000015 | -150
(1 row)
```

❖ 布尔（bool-json）：仅 true、false，全小写。

```
select 'true'::json;
select 'false'::jsonb;
```

结果显示如下：

```
 json
-----------
 true
(1 row)
 jsonb
-----------
 false
(1 row)
```

❖ 字符串（str-json）：必须是加双引号的字符串。

```
select '"a"'::json;
select '"abc"'::jsonb;
```

结果显示如下：

```
json
---------
 "a"
(1 row)
jsonb
----------
 "abc"
(1 row)
```

❖ 数组（array-json）：使用中括号[]包裹，满足数组书写条件。数组内元素类型可以是任意合法的 JSON，且不要求类型一致。

```
select '[1, 2, "foo", null]'::json;select '[]'::json;
select '[1, 2, "foo", null, [[]], {}]'::jsonb;
```

结果显示如下：

```
    json
--------------------
[1, 2, "foo", null]
(1 row)
json
-----------
[]
(1 row)
        jsonb
------------------------------
[1, 2, "foo", null, [[]], {}]
(1 row)
```

❖ 对象（object-json）：使用大括号{}包裹，键必须是满足 JSON 字符串规则的字符串，值可以是任意合法的 JSON。

```
select '{}'::json;select '{"a": 1, "b": {"a": 2,  "b": null}}'::json;
select '{"foo": [true, "bar"], "tags": {"a": 1, "b": null}}'::jsonb;
```

结果显示如下：

```
json
```

```
------
 {}
(1 row)
          json
-----------------------------------
 {"a": 1, "b": {"a": 2,  "b": null}}
(1 row)
                jsonb
--------------------------------------------------
 {"foo": [true, "bar"], "tags": {"a": 1, "b": null}}
(1 row)
```

- 区分 'null'::json 和 null::json 是两个不同的概念，类似于字符串 str= “” 和 str=null。
- 对于数字，当使用科学计数法的时候，jsonb 类型会将其展开，而 json 会精准拷贝输入。

JSONB 高级特性

JSON 和 JSONB 的主要差异在于存储方式上的不同，JSONB 存储的是解析后的二进制，能够体现 JSON 的层次结构，更方便直接访问等，因此 JSONB 会有很多 JSON 所不具有的高级特性。

- 注意事项
  - 不支持列存。
  - 不支持作为分区键。
  - 不支持外表、mot。
- 格式归一化，对于输入的 object-json 字符串，解析成 jsonb 二进制后，会天然的丢弃语义上无关紧要的细节，比如空格：

```
select '  [1, " a ", {"a"  :1   }] '::jsonb;
```

结果显示如下：

```
   jsonb
---------------------
 [1, " a ", {"a": 1}]
(1 row)
```

❖ 对于 object-json，键值会重新进行排序，排序规则：长度长的在后、长度相等则 ascii 码大的在后，如：

```
select '{"aa" : 1, "b" : 2, "a" : 3}'::jsonb;
```

结果显示如下：

```
      jsonb
--------------------------
{"a": 3, "b": 2, "aa": 1}
(1 row)
```

❖ 对于 object-json，会删除重复的键值，只保留最后一个出现的，如：

```
select '{"a" : 1, "a" : 2}'::jsonb;
```

结果显示如下：

```
 jsonb
----------
{"a": 2}
(1 row)
```

❖ 大小比较：由于经过了格式归一化，保证了同一种语义下的 jsonb 只会有一种存在形式，因此按照制定的规则，可以比较大小。

- 首先比较类型：object-jsonb > array-jsonb > bool-jsonb > num-jsonb > str-jsonb > null-jsonb
- 同类型则比较内容：
  - ✧ str-json 类型：依据 text 比较的方法，使用数据库默认排序规则进行比较，返回值正数代表大于，负数代表小于，0 表示相等。
  - ✧ num-json 类型：数值比较。
  - ✧ bool-json 类型：true > false。
  - ✧ array-jsonb 类型：长度长的 > 长度短的，长度相等则依次比较每个元素。
  - ✧ object-jsonb 类型：长度长的 > 长度短的，长度相等则依次比较每个键值对，先比较键，在比较值。
- object-jsonb 类型内比较时，比较时使用的是格式整理后的最终结果进行比较，因此相对于我们直接的输入未必会很直观。

- 创建索引、主外键
  - BTREE 索引
  - jsonb 类型支持创建 btree 索引，支持创建主键、外键。
  - GIN 索引
    - GIN 索引可以用来有效地搜索出现在大量 jsonb 文档（datums）中的键或者键/值对。提供了两个 GIN 操作符类（jsonb_ops、jsonb_hash_ops），提供了不同的性能和灵活性取舍。缺省的 GIN 操作符类支持使用@>、<@、?、 ?&和?|操作符查询，非缺省的 GIN 操作符类 jsonb_path_ops 只支持索引@>、<@操作符。
    - 相关的操作符请参见 JSON/JSONB 函数和操作符
- 包含存在

查询一个 JSON 之中是否包含某些元素，或者某些元素是否存在于某个 JSON 中是 jsonb 的一个重要能力。简单的标量/原始值只包含相同的值。

```
SELECT '"foo"'::jsonb @> '"foo"'::jsonb;
```

结果显示如下：

```
 ?column?
----------
 t
(1 row)
```

- 左侧数组包含了右侧字符串。

```
SELECT '[1, "aa", 3]'::jsonb ? 'aa';
```

结果显示如下：

```
 ?column?
----------
 t
(1 row)
```

- 左侧数组包含了右侧的数组所有元素，顺序、重复不重要。

```
SELECT '[1, 2, 3]'::jsonb @> '[1, 3, 1]'::jsonb;
```

结果显示如下：

```
 ?column?
----------
 t
(1 row)
```

❖ 左侧 object-json 包含了右侧 object-json 的所有键值对。

```
SELECT '{"product": "PostgreSQL", "version": 9.4, "jsonb":true}'::jsonb @> '{"version":
9.4}'::jsonb;
```

结果显示如下：

```
 ?column?
----------
 t
(1 row)
```

❖ 左侧数组并没有包含右侧的数组所有元素，因为左侧数组的三个元素为 1、2、[1,3]，右侧的为 1、3。

```
SELECT '[1, 2, [1, 3]]'::jsonb @> '[1, 3]'::jsonb;
```

结果显示如下：

```
 ?column?
----------
 f
(1 row)
```

相关的操作符请参见 JSON/JSONB 函数和操作符。

❖ 函数和操作符

json/jsonb 类型相关支持的函数和操作符请参见 JSON/JSONB 函数和操作符。

4.1.2.20.ROWID 数据类型

功能描述

PanWeiDB 数据库的表中的每一行数据都有一个唯一的标识符，或者称为 rowid，rowid 是一个伪列在 PanWeiDB 内部通常就是使用它来访问数据。

该值表明了该行在数据库中的物理具体位置。可以在一个查询中使用 rowid 来表明查询结果中包含该值。

ROWID 可以分为物理 rowid 和逻辑 rowid 两种。普通的表中的 rowid 是物理 rowid，索引组织表(IOT)的 rowid 是逻辑 rowid。

示例

1、创建测试表 ridtest。

```
CREATE TABLE ridtest (id int,name varchar(10));
```

2、插入测试数据。

```
INSERT INTO ridtest values(1,'zhangsan');
```

3、查询 rowid。

```
SELECT  rowid,id FROM ridtest;
```

返回结果如下：

```
    rowid       | id
--------------------+----
 BWEAAA==AAAAAA==AQA= | 1
(1 row)
```

4.1.2.21.BFILE 数据类型

功能描述

bfile 用于存储服务器文件系统中的物理二进制文件数据对象。bfile 文件存储在操作系统文件中，而不是在数据库中，bfile 列存储对操作系统文件的引用。操作系统访问的任何存储设备都可以保存 bfile 数据，包括硬盘驱动器、CD-ROM、PhotoCD 和 DVD。如果操作系统支持对操作系统文件的流模式访问，则数据库可以访问 bfile。

bfile 类型的数据是：

- 在应用程序运行时不会更改的二进制数据，例如图形。

- 加载到其他大型对象类型（例如 BLOB 或 CLOB）中的数据，可以在其中操作数据。
- 适用于字节流访问的数据，例如多媒体数据。

在对 bfile 类型的 SQL 操作中，需要支持 bfilename 函数。bfilename 函数返回一个 bfile 文件定位器，该定位器指向操作系统中一个物理文件。

bfilename 函数主要用于 bfile 类型数据的构造，构造的数据源为一个 directory 对象和一个操作系统物理二进制文件名，函数语法格式为：

```
BFILENAME('directory', 'filename')
```

- directory 对象：指向一个操作系统的物理目录。该目录不应为数据库数据目录，以及操作系统控制文件、日志文件和其他系统文件所在的目录。必须在 bfilename 函数调用前创建，并且对名区分大小写。
- filename：文件名，操作系统上一个物理二进制文件，由于 bfile 为只读数据类型，必须在访问文件数据前创建该文件。

bfile 数据类型主要用于对 bfilename 函数构造的数据进行存储，然后调用 dbms_lob 内置包的接口对该数据进行操作。bfile 数据类型的功能主要分为三个部分：

- bfile 数据的存储和打印。
- bfile 数据的使用，主要为对操作系统物理文件的读取过程，即分为以下三部分：
  - 打开操作系统物理文件。bfile 为只读数据类型，因此打开文件的模式只能为只读。
  - 对文件进行读取。可根据参数设置来决定读取文件的字节数，如需读取整个文件，可先调用 dbms_lob 包的 getlength，函数获取文件的大小。需注意的是，单次读取文件的字节数不能超过 32767 字节，如文件过大，可通过设置参数对文件进行多次读取。
  - 关闭文件。对打开的物理文件进行关闭。

bfile 数据的修改：对于 bfile 数据的持久引用（如存储在表中），bfile 数据可以作为表的列值进行修改。

注意事项

bfilename 函数依赖 panweidb 自带的 directory 功能。

使用流程

1、创建 directory 对象，创建需访问的文件并将其置于 directory 对象所指向的操作系统目录中。

2、创建包含 bfile 数据类型列的表。

3、调用 bfilename 函数对数据进行构造并插入到表中进行存储。

示例

1、操作系统环境下创建文件，并输入数据。

```
vi /home/panweidb/bfile.data
cat /home/panweidb/bfile.data
```

文件内容为：

```
张三
李四
```

2、在数据库中执行如下命令创建目录和表。

```
drop directory d_bfile;
drop table testbfile1;
create directory d_bfile as '/home/panweidb';
create table testbfile1(id number,bfile_name bfile);
```

3、插入数据。

```
insert into testbfile1 values (1,bfilename('d_bfile','bfile.data'));
```

4、查询数据。

```
select * from  testbfile1;
```

返回结果为：

```
 id |          bfile_name
----+-------------------------------
  1 | bfilename('d_bfile','bfile.data')
```

4.1.2.22.ACLITEM 权限类型

功能描述

PanWeiDB 中对象的访问权限使用 aclitem 类型进行记录。

语法格式

```
rolename=privileges/rolegrant
```

参数说明

- rolename：被授权者。
- privileges：授予的权限，权限列表如下所示：
  - r：SELECT
  - w：UPDATE
  - a：INSERT
  - d：DELETE
  - D：TRUNCATE
  - x：REFERENCES
  - t：TRIGGER
  - X：EXECUTE
  - U：USAGE
  - C：CREATE
  - c：CONNECT
  - T：TEMPORARY
  - arwdDxt：ALL PRIVILEGES
  - *：授予优先权选项
- rolegrant：授权者。

示例

系统表中记录了对象的访问权限，如表 1 所示。

表 1 系统表中的 ACL 列

系统表	字段名
pg_database	datacl

pg_tablespace	spcacl
pg_class	relacl
pg_attribute	attacl
pg_type	typacl
pg_proc	proacl
pg_language	lanacl

查询 pg_database 中记录的 aclitem 类型字段

```
select datname,datacl from pg_database;
```

结果显示如下：

```
 datname |           datacl
-----------+-----------------------------------
template1 | {=c/panweidb,panweidb=CTc/panweidb}
db_oracle |
my_test   |
template0 | {=c/panweidb,panweidb=CTc/panweidb}
panweidb  |
postgres  |
(6 rows)
```

4.1.3.常量与宏

PanWeiDB 支持的常量和宏请参见表 1。

表 1 常量和宏

参数	描述	示例
CURRENT_CATALOG	当前数据库	panweidb=# SELECT CURRENT_CATALOG; current_database ------------------ PanWeiDB (1 row)
CURRENT_ROLE	当前用户	panweidb=# SELECT CURRENT_ROLE;

参数	描述	示例
		current_user -------------- panweidb (1 row)
CURRENT_SCHEMA	当前数据库模式	panweidb=# SELECT CURRENT_SCHEMA; current_schema ---------------- public (1 row)
CURRENT_USER	当前用户	panweidb=# SELECT CURRENT_USER; current_user -------------- panweidb (1 row)
LOCALTIMESTAMP	当前会话时间（无时区）	panweidb=# SELECT LOCALTIMESTAMP; timestamp ---------------------------- 2015-10-10 15:37:30.968538 (1 row)
NULL	空值	-
SESSION_USER	当前系统用户	panweidb=# SELECT SESSION_USER; session_user -------------- panweidb (1 row)
SYSDATE	当前系统日期	panweidb=# SELECT SYSDATE; sysdate ---------------------

参数	描述	示例
		2015-10-10 15:48:53 (1 row)
USER	当前用户，此用户为 CURRENT_USER 的别名	panweidb=# SELECT USER; current_user -------------- panweidb (1 row)

4.1.4.函数和操作符

本章节主要介绍 PanWeiDB 支持的 SQL 函数和操作符。

❖ 函数

将经常用到的逻辑语句进行封装，需要的时候直接调用即可。每次使用时无需重新书写逻辑语句，直接调用函数即可。

❖ 操作符

SQL 操作符是一系列用于比较的函数，这些操作符广泛的用于 select 语句的 where 从句中。

4.1.4.1.时间和日期处理函数和操作符

时间日期操作符

【警告】

用户在使用时间和日期操作符时，对应的操作数请使用明确的类型前缀修饰，以确保数据库在解析操作数的时候能够与用户预期一致，不会产生用户非预期的结果。比如下面示例没有明确数据类型就会出现异常错误，例如：

```
SELECT date '2001-10-01' - '7' AS RESULT;
```

报错如下：

```
ERROR:  invalid input syntax for type oradate: "7"
```

```
LINE 1: SELECT date '2001-10-01' - '7' AS RESULT;
                                 ^
CONTEXT:  referenced column: result
```

下面介绍 4 中操作符，分别是“+”，“-”，“*”，“/”

❖ “'+”

示例如下：

```
SELECT date '2001-9-28' + integer '7' AS RESULT;
     result
---------------------
 2001-10-05 00:00:00
(1 row)

SELECT date '2001-09-28' + interval '1 hour' AS RESULT;
 result
---------------------
 2001-09-28 01:00:00
(1 row)

 SELECT date '2001-09-28' + time '03:00' AS RESULT;
 result
---------------------
 2001-09-28 03:00:00
(1 row)

 SELECT interval '1 day' + interval '1 hour' AS RESULT;
 result
----------------
 1 day 01:00:00
(1 row)

SELECT timestamp '2001-09-28 01:00' + interval '23 hours' AS RESULT;
 result
```

```
---------------------
 2001-09-29 00:00:00
(1 row)

SELECT time '01:00' + interval '3 hours' AS RESULT;
 result
----------
 04:00:00
(1 row)
```

❖ “-”

示例如下：

```
SELECT date '2001-10-01' - date '2001-09-28' AS RESULT;
 result
--------
 3days
(1 row)

SELECT date '2001-10-01' - integer '7' AS RESULT;
 result
---------------------
 2001-09-24 00:00:00
(1 row)

 SELECT date '2001-09-28' - interval '1 hour' AS RESULT;
 result
---------------------
 2001-09-27 23:00:00
(1 row)

SELECT time '05:00' - time '03:00' AS RESULT;
 result
----------
 02:00:00
(1 row)
```

```
SELECT time '05:00' - interval '2 hours' AS RESULT;
 result
----------
 03:00:00
(1 row)

SELECT timestamp '2001-09-28 23:00' - interval '23 hours' AS RESULT;
 result
---------------------
 2001-09-28 00:00:00
(1 row)

SELECT interval '1 day' - interval '1 hour' AS RESULT;
 result
----------
 23:00:00
(1 row)

SELECT timestamp '2001-09-29 03:00' - timestamp '2001-09-27 12:00' AS RESULT;
 result
----------------
 1 day 15:00:00
(1 row)
```

❖ “-”

示例如下：

```
SELECT 900 * interval '1 second' AS RESULT;
 result
----------
 00:15:00
(1 row)

SELECT 21 * interval '1 day' AS RESULT;
 result
```

```
---------
 21 days
(1 row)

 SELECT double precision '3.5' * interval '1 hour' AS RESULT;
 result
----------
 03:30:00
(1 row)
```

- “/”

示例如下：

```
SELECT interval '1 hour' / double precision '1.5' AS RESULT;
 result
----------
 00:40:00
(1 row)
```

时间/日期函数

- **age(timestamp, timestamp)**

  描述：将两个参数相减，并以年、月、日作为返回值。若相减值为负，则函数返回亦为负，入参可以都带 timezone 或都不带 timezone。

  返回值类型：interval

  示例：

```
SELECT age(timestamp '2001-04-10', timestamp '1957-06-13');
```

```
       age
-------------------------
 43 years 9 mons 27 days
(1 row)
```

- age(timestamp)

  描述：当前时间和参数相减，入参可以带或者不带 timezone。

  返回值类型：interval

  示例：

```
SELECT age(timestamp '1957-06-13');

        age
------------------------
 60 years 2 mons 18 days
(1 row)
```

❖ clock_timestamp()

描述：实时时钟的当前时间戳。

返回值类型：timestamp with time zone

示例：

```
SELECT clock_timestamp();

  clock_timestamp
-------------------------------
 2022-06-23 16:26:09.726396+08
(1 row)
```

❖ current_date

描述：当前日期。

返回值类型：date

示例：

```
SELECT current_date;
   date
------------
2022-06-23
(1 row)
```

❖ current_time

描述：当前时间。

返回值类型：time with time zone

示例：

```
SELECT current_time;
```

```
    timetz
--------------------
  16:26:37.704326+08
(1 row)
```

❖ current_timestamp

描述：当前日期及时间。

返回值类型：timestamp with time zone

示例：

```
SELECT current_timestamp;
       pg_systimestamp
-------------------------------
  2022-06-23 16:27:29.336502+08
(1 row)
```

❖ date_part(text, timestamp)

描述：获取日期/时间值中子域的值，例如年或者小时的值。等效于 extract(field from timestamp)。

timestamp 类型：abstime、date、interval、reltime、time with time zone、time without time zone、timestamp with time zone、timestamp without timezone。

返回值类型：double precision

示例：

```
SELECT date_part('hour', timestamp '2001-02-16 20:38:40');
  date_part
-----------
     20
(1 row)
```

❖ date_part(text, interval)

描述：获取日期/时间值中子域的值。获取月份值时，如果月份值大于 12，则 取与 12 的模。等效于 extract(field from timestamp)。

返回值类型：double precision

示例：

```
SELECT date_part('month', interval '2 years 3 months');
 date_part
-----------
         3
(1 row)
```

❖ date_trunc(text, timestamp)

描述：截取到参数 text 指定的精度。

返回值类型：interval、timestamp with time zone、timestamp without timezone

示例：

```
SELECT date_trunc('hour', timestamp  '2001-02-16 20:38:40');
      date_trunc
 ---------------------
  2001-02-16 20:00:00
(1 row)
```

❖ trunc(timestamp)

描述：默认按天截取。

示例：

```
SELECT trunc(timestamp  '2001-02-16 20:38:40');
                trunc
 ---------------------
  2001-02-16 00:00:00
(1 row)
```

❖ daterange(arg1, arg2)

描述：获取时间边界信息。arg1 和 arg2 的类型为 date。

返回值类型：daterange

示例：

```
SELECT daterange('2000-05-06','2000-08-08');
        daterange
 ------------------------
   [2000-05-06,2000-08-08)
(1 row)
```

- daterange(arg1, arg2, text)

  描述：获取时间边界信息。arg1 和 arg2 的类型为 date，text 类型为 text。

  返回值类型：daterange

  示例：

```
SELECT daterange('2000-05-06','2000-08-08','[]');
       daterange
 ------------------------
   [2000-05-06,2000-08-09)
(1 row)
```

- extract(field from timestamp)

  描述：获取小时的值。

  返回值类型：double precision

  示例：

```
SELECT extract(hour from timestamp '2001-02-16 20:38:40');
 date_part
-----------
     20
(1 row)
```

- extract(field from interval)

  描述：获取月份的值。如果大于 12，则取与 12 的模。

  返回值类型：double precision

  示例：

```
SELECT extract(month from interval '2 years 3 months');
 date_part
-----------
     3
(1 row)
```

- isfinite(date)

  描述：测试是否为有效日期。

  返回值类型：Boolean

  示例：

```
SELECT isfinite(date '2001-02-16');
 isfinite
----------
  t
(1 row)
```

- isfinite(timestamp)

  描述：测试判断是否为有效时间。

  返回值类型：Boolean

  示例：

```
SELECT isfinite(timestamp '2001-02-16 21:28:30');
 isfinite
----------
 t
(1 row)
```

- isfinite(interval)

  描述：测试是否为有效区间。

  返回值类型：Boolean

  示例：

```
SELECT isfinite(interval '4 hours');
 isfinite
----------
 t
(1 row)
```

- justify_days(interval)

  描述：将时间间隔以月（30 天为一月）为单位。

  返回值类型：interval

  示例：

```
SELECT justify_days(interval '35 days');
 justify_days
--------------
 1 mon 5 days
```

```
(1 row)
```

- justify_hours(interval)

  描述：将时间间隔以天（24 小时为一天）为单位。

  返回值类型：interval

  示例：

```
 SELECT JUSTIFY_HOURS(INTERVAL '27 HOURS');
justify_hours
---------------
 1 day 03:00:00
(1 row)
```

- justify_interval(interval)

  描述：结合 justify_days 和 justify_hours，调整 interval。

  返回值类型：interval

  示例：

```
SELECT JUSTIFY_INTERVAL(INTERVAL '1 MON -1 HOUR');
justify_interval
------------------
 29 days 23:00:00
```

- localtime

  描述：当前时间。

  返回值类型：time

  示例：

```
 SELECT localtime AS RESULT;
result
---------------
 16:05:55.664681
(1 row)
```

- localtimestamp

  描述：当前日期及时间。

  返回值类型：timestamp

  示例：

```
SELECT localtimestamp;
     timestamp
----------------------------
   2017-09-01 17:03:30.781902
(1 row)
```

- now()

  描述：当前日期及时间。

  返回值类型：timestamp with time zone

  示例：

```
SELECT now();
          now
------------------------------
 2022-08-01 18:00:22.753185+08
(1 row)
```

- timenow

  描述：当前日期及时间。

  返回值类型：timestamp with time zone

  示例：

```
select timenow();
      timenow
-----------------------
 2022-08-01 18:00:50+08
(1 row)
```

- numtodsinterval(num, interval_unit)

  描述：将数字转换为 interval 类型。num 为 numeric 类型数字，interval_unit 为固定格式字符串（'DAY' | 'HOUR' | 'MINUTE' | 'SECOND'）。

  可以通过设置参数 IntervalStyle 为 a，兼容该函数 interval 输出格式。

  示例：

```
SELECT numtodsinterval(100, 'HOUR');
```

```
 numtodsinterval
-----------------
 100:00:00
(1 row)

SET intervalstyle = a;
 SET

SELECT numtodsinterval(100, 'HOUR');
      numtodsinterval
------------------------------
 +000000004 04:00:00.000000000
(1 row)
```

- pg_sleep(seconds)

  描述：服务器线程延迟时间，单位为秒。

  返回值类型：void

  示例：

```
SELECT pg_sleep(10);
   pg_sleep
  ----------


(1 row)
```

- statement_timestamp()

  描述：当前日期及时间。

  返回值类型：timestamp with time zone

  示例：

```
SELECT statement_timestamp();
     statement_timestamp
 ------------------------------
  2022-08-01 17:26:08.15684+08
(1 row)
```

- sysdate

描述：当前日期及时间。

返回值类型：timestamp

示例：

```
SELECT sysdate;
       sysdate
 --------------------
   2022-08-01 16:49:22
(1 row)
```

- timeofday()

  描述：当前日期及时间（像 clock_timestamp，但是返回时为 text）。

  返回值类型：text

  示例：

```
SELECT timeofday();

           timeofday
-----------------------------------
  Mon Aug 01 16:49:48.255352 2022 CST
(1 row)
```

- transaction_timestamp()

  描述：当前日期及时间，与 current_timestamp 等效。

  返回值类型：timestamp with time zone

  示例：

```
SELECT transaction_timestamp();
      transaction_timestamp
 ------------------------------
   2022-08-01 16:50:05.538612+08
(1 row)
```

- add_months(d,n)

  描述：用于计算时间点 d 再加上 n 个月的时间。可通过 end_month_calculate 兼容性配置项控制表现。

d：timestamp 类型的值，以及可以隐式转换为 timestamp 类型的值。

n：INTEGER 类型的值，以及可以隐式转换为 INTEGER 类型的值。

返回值类型：timestamp

示例：

```
SELECT add_months(to_date('2017-5-29', 'yyyy-mm-dd'), 11) FROM sys_dummy;
     add_months
---------------------
  2018-04-29 00:00:00
(1 row)
```

- last_day(d)

  描述：用于计算时间点 d 当月最后一天的时间。

  返回值类型：timestamp

  示例：

```
select last_day(to_date('2017-01-01', 'YYYY-MM-DD')) AS cal_result;
     cal_result
---------------------
  2017-01-31 00:00:00
(1 row)
```

- next_day(x,y)

  描述：用于计算时间点 x 开始的下一个星期几（y）的时间。

  返回值类型：timestamp

  示例：

```
select next_day(timestamp '2017-05-25 00:00:00','Sunday')AS cal_result;
     cal_result
---------------------
  2017-05-28 00:00:00
(1 row)
```

- tinterval(abstime, abstime)

  描述：用两个绝对时间创建时间间隔。

  返回值类型：tinterval

示例：

```
call tinterval(abstime 'May 10, 1947 23:59:12', abstime 'Mon May  1 00:30:30 199
5');
                tinterval
--------------------------------------------------
  ["1947-05-10 23:59:12+09" "1995-05-01 00:30:30+08"]
(1 row)
```

- tintervalend(tinterval)

  描述：返回 tinteval 的结束时间。

  返回值类型：abstime

  示例：

```
select tintervalend('["Sep 4, 1983 23:59:12" "Oct4, 1983 23:59:12"]');
      tintervalend
-----------------------
   1983-10-04 23:59:12+08
(1 row)
```

- tintervalrel(tinterval)

  描述：计算并返回 tinterval 的相对时间。

  返回值类型：reltime

  示例：

```
select tintervalrel('["Sep 4, 1983 23:59:12" "Oct4, 1983 23:59:12"]');
   tintervalrel
-------------
   1 mon
```

- smalldatetime_ge

  描述：判断是否第一个参数大于等于第二个参数。

  参数：smalldatetime, smalldatetime

  返回值类型：boolean

- smalldatetime_cmp

  描述：对比 smalldatetime 是否相等。

  参数：smalldatetime, smalldatetime

返回值类型：integer

- smalldatetime_eq

  描述：对比 smalldatetime 是否相等。

  参数：smalldatetime, smalldatetime

  返回值类型：boolean。

- smalldatetime_gt

  描述：判断是否第一个参数大于第二个参数。

  参数：smalldatetime, smalldatetime

  返回值类型：boolean

- smalldatetime_hash

  描述：计算 timestamp 对应的哈希值。

  参数：smalldatetime

  返回值类型：integer

- smalldatetime_in

  描述：输入 timestamp。

  参数：cstring, oid, integer

  返回值类型：smalldatetime

- smalldatetime_larger

  描述：返回较大的 timestamp。

  参数：smalldatetime, smalldatetime

  返回值类型：smalldatetime

- smalldatetime_le

  描述：判断是否第一个参数小于等于第二个参数。

  参数：smalldatetime, smalldatetime

  返回值类型：boolean

- smalldatetime_lt

  描述：判断是否第一个参数小于第二个参数。

  参数：smalldatetime, smalldatetime

  返回值类型：boolean

- smalldatetime_ne

  描述：比较两个 timestamp 是否不相等。

参数：smalldatetime, smalldatetime

返回值类型：boolean

- smalldatetime_out

  描述：timestamp 转换为外部形式。

  参数：smalldatetime

  返回值类型：cstring

- smalldatetime_send

  描述：timestamp 转换为二进制格式。

  参数：smalldatetime

  返回值类型：bytea

- smalldatetime_smaller

  描述：返回较小的一个 smalldatetime。

  参数：smalldatetime, smalldatetime

  返回值类型：smalldatetime

- smalldatetime_to_abstime

  描述：smalldatetime 转换为 abstime。

  参数：smalldatetime

  返回值类型：abstime

- smalldatetime_to_time

  描述：smalldatetime 转换为 time。

  参数：smalldatetime

  返回值类型：time without time zone

- smalldatetime_to_timestamp

  描述：smalldatetime 转换为 timestamp。

  参数：smalldatetime

  返回值类型：timestamp without time zone

- smalldatetime_to_timestamptz

  描述：smalldatetime 转换为 timestamptz。

  参数：smalldatetime

  返回值类型：timestamp with time zone

- smalldatetime_to_varchar2

描述：smalldatetime 转换为 varchar2。

参数：smalldatetime

返回值类型：character varying

说明：

获取当前时间有多种方式，请根据实际业务从场景选择合适的接口：

1、以下接口按照当前事务的开始时刻返回值：

```
CURRENT_DATE CURRENT_TIME CURRENT_TIME(precision) CURRENT_TIMESTAMP(precision) LOCALTIME LOCALTIMESTAMP LOCALTIME(precision) LOCALTIMESTAMP(precision)
```

其中 CURRENT_TIME 和 CURRENT_TIMESTAMP(precision)传递带有时区的值；LOCALTIME 和 LOCALTIMESTAMP 传递的值不带时区。CURRENT_TIME、LOCALTIME 和 LOCALTIMESTAMP 可以有选择地接受一个精度参数， 该精度导致结果的秒域被园整为指定小数位。如果没有精度参数，结果将被给予所能得到的全部精度。

因为这些函数全部都按照当前事务的开始时刻返回结果，所以它们的值在事务运行的整个期间内都不改变。 我们认为这是一个特性：目的是为了允许一个事务在“当前”时间上有一致的概念， 这样在同一个事务里的多个修改可以保持同样的时间戳。

2、以下接口返回当前语句开始时间：

```
transaction_timestamp() statement_timestamp() now()
```

其中 transaction_timestamp()等价于 CURRENT_TIMESTAMP(precision)，但是其命名清楚地反映了它的返回值。statement_timestamp()返回当前语句的开始时刻（更准确的说是收到 客户端最后一条命令的时间）。statement_timestamp()和 transaction_timestamp()在一个事务的第一条命令期间返回值相同，但是在随后的命令中却不一定相同。

now()等效于 transaction_timestamp()。

3、以下接口返回函数被调用时的真实当前时间：

```
clock_timestamp() timeofday()
```

clock_timestamp()返回真正的当前时间，因此它的值甚至在同一条 SQL 命令中都会变化。timeofday()和 clock_timestamp()相似，timeofday()也返回真实的当前时间，但是它的结果是一个格式化的 text 串，而不是 timestamp with time zone 值。

EXTRACT

- EXTRACT(field FROM source)

  extract 函数从日期或时间的数值里抽取子域，比如年、小时等。source 必须是一个 timestamp、time 或 interval 类型的值表达式（类型为 date 的表达式转换为 timestamp，因此也可以用）。field 是一个标识符或者字符串，它指定从源数据中抽取的域。extract 函数返回类型为 double precision 的数值。field 的取值范围如下所示。

- century

  世纪。

  第一个世纪从 0001-01-01 00:00:00 AD 开始。这个定义适用于所有使用阳历的国家。没有 0 世纪，直接从公元前 1 世纪到公元 1 世纪。

  示例：

```
SELECT EXTRACT(CENTURY FROM TIMESTAMP '2000-12-16 12:21:13');
 date_part
-----------
     20
(1 row)
```

- day

  如果 source 为 timestamp，表示月份里的日期（1-31）。

```
SELECT EXTRACT(DAY FROM TIMESTAMP '2001-02-16 20:38:40');
   date_part
  -----------
      16
 (1 row)
```

  如果 source 为 interval，表示天数。

```
SELECT EXTRACT(DAY FROM INTERVAL '40 days 1 minute');
```

```
 date_part
-----------
    40
(1 row)
```

❖ decade

年份除以 10。

```
SELECT EXTRACT(DECADE FROM TIMESTAMP '2001-02-16 20:38:40');
 date_part
-----------
    200
(1 row)
```

❖ dow

每周的星期几，星期天（0）到星期六（6）。

```
SELECT EXTRACT(DOW FROM TIMESTAMP '2001-02-16 20:38:40');
 date_part
-----------
     5
(1 row)
```

❖ doy

一年的第几天（1~365/366）。

```
SELECT EXTRACT(DOY FROM TIMESTAMP '2001-02-16 20:38:40');
 date_part
-----------
    47
(1 row)
```

❖ epoch

如果 source 为 timestamp with time zone，表示自 1970-01-01 00:00:00-00 UTC 以来的秒数（结果可能是负数）；

如果 source 为 date 和 timestamp，表示自 1970-01-01 00:00:00-00 当地时间以来的秒数；

如果 source 为 interval，表示时间间隔的总秒数。

```
SELECT EXTRACT(EPOCH FROM TIMESTAMP WITH TIME ZONE '2001-02-16 20:38:40.12-08');
```

```
   date_part
--------------
  982384720.12
(1 row)

SELECT EXTRACT(EPOCH FROM INTERVAL '5 days 3 hours');
   date_part
-----------
  442800
(1 row)
```

将 epoch 值转换为时间戳的方法。

```
SELECT TIMESTAMP WITH TIME ZONE 'epoch' + 982384720.12 * INTERVAL '1 secon
    d' AS RESULT;
     result
---------------------------
 2001-02-17 12:38:40.12+08
(1 row)
```

❖ hour

小时域（0-23）。

```
SELECT EXTRACT(HOUR FROM TIMESTAMP '2001-02-16 20:38:40');
   date_part
-----------
        20
(1 row)
```

❖ isodow

一周的第几天（1-7）。

星期一为 1，星期天为 7。

📖 **说明：**

除了星期天外，都与 dow 相同。

```
SELECT EXTRACT(ISODOW FROM TIMESTAMP '2001-02-18 20:38:40');
 date_part
-----------
```

```
      7
(1 row)
```

❖ isoyear

日期中的 ISO 8601 标准年（不适用于间隔）。

每个带有星期一开始的周中包含 1 月 4 日的 ISO 年，所以在年初的 1 月或 12 月下旬的 ISO 年可能会不同于阳历的年。详细信息请参见后续的 week 描述。

```
SELECT EXTRACT(ISOYEAR FROM DATE '2006-01-01');
   date_part
-----------
    2005
(1 row)

SELECT EXTRACT(ISOYEAR FROM DATE '2006-01-02');

   date_part
-----------

     2006
(1 row)
```

❖ microseconds

秒域（包括小数部分）乘以 1,000,000。

```
SELECT EXTRACT(MICROSECONDS FROM TIME '17:12:28.5');
   date_part
-----------
    28500000
(1 row)
```

❖ millennium

千年。

20 世纪（19xx 年）里面的年份在第二个千年里。第三个千年从 2001 年 1 月 1 日零时开始。

```
SELECT EXTRACT(MILLENNIUM FROM TIMESTAMP '2001-02-16 20:38:40');
   date_part
```

```
-----------
           3
(1 row)
```

❖ milliseconds

秒域（包括小数部分）乘以 1000。请注意它包括完整的秒。

```
SELECT EXTRACT(MILLISECONDS FROM TIME '17:12:28.5');
   date_part
-----------
       28500
(1 row)
```

❖ minute

分钟域（0-59）。

```
SELECT EXTRACT(MINUTE FROM TIMESTAMP '2001-02-16 20:38:40');
   date_part
-----------
          38
(1 row)
```

❖ month

如果 source 为 timestamp，表示一年里的月份数（1-12）。

```
SELECT EXTRACT(MONTH FROM TIMESTAMP '2001-02-16 20:38:40');
   date_part
-----------
           2
(1 row)
```

如果 source 为 interval，表示月的数目，然后对 12 取模（0-11）。

```
SELECT EXTRACT(MONTH FROM INTERVAL '2 years 13 months');
   date_part
-----------
           1
(1 row)
```

❖ quarter

该天所在的该年的季度（1-4）。

```
SELECT EXTRACT(QUARTER FROM TIMESTAMP '2001-02-16 20:38:40');
```

```
 date_part
-----------
         1
(1 row)
```

- second

  秒域，包括小数部分（0-59）。

```
SELECT EXTRACT(SECOND FROM TIME '17:12:28.5');
 date_part
-----------
   28.5
(1 row)
```

- timezone

  与 UTC 的时区偏移量，单位为秒。正数对应 UTC 东边的时区，负数对应 UTC 西边的时区。

- timezone_hour

  时区偏移量的小时部分。

- timezone_minute

  时区偏移量的分钟部分。

- week

  该天在所在的年份里是第几周。ISO 8601 定义一年的第一周包含该年的一月四日（ISO-8601 的周从星期一开始）。换句话说，一年的第一个星期四在第一周。

  在 ISO 定义里，一月的头几天可能是前一年的第 52 或者第 53 周，十二月的后几天可能是下一年第一周。比如，2005-01-01 是 2004 年的第 53 周，而 2006-01-01 是 2005 年的第 52 周，2012-12-31 是 2013 年的第一周。建议 isoyear 字段和 week 一起使用以得到一致的结果。

```
SELECT EXTRACT(WEEK FROM TIMESTAMP '2001-02-16 20:38:40');
 date_
-----------
     7
(1 row)
```

❖ year

年份域。

```
SELECT EXTRACT(YEAR FROM TIMESTAMP '2001-02-16 20:38:40');
date_part
-----------
   2001
```

date_part

date_part 函数是在传统的 Ingres 函数的基础上制作的（该函数等效于 SQL 标准函数 extract）：

❖ **date_part**('field', source)

这里的 field 参数必须是一个字符串，而不是一个名称。有效的 field 与 extract 一样，详细信息请参见 EXTRACT。

示例：

```
SELECT date_part('day', TIMESTAMP '2001-02-16 20:38:40');
 date_part
-----------
      16
(1 row)
SELECT date_part('hour', INTERVAL '4 hours 3 minutes');
 date_part
-----------
       4
(1 row)
```

表 3 显示了可以用于格式化日期和时间值的模版。

表 3 用于日期/时间格式化的模式

类别	模式	描述
小时	HH	一天的小时数（01-12）

类别	模式	描述
	HH12	一天的小时数（01-12）
	HH24	一天的小时数（00-23）
分钟	MI	分钟（00-59）
秒	SS	秒（00-59）
	FF	微秒（000000-999999）
	SSSSS	午夜后的秒（0-86399）
上、下午	AM 或 A.M.	上午标识
	PM 或 P.M.	下午标识
年	Y,YYY	带逗号的年（4 和更多位）
	SYYYY	公元前四位年
	YYYY	年（4 和更多位）
	YYY	年的后三位
	YY	年的后两位
	Y	年的最后一位
	IYYY	ISO 年（4 位或更多位）
	IYY	ISO 年的最后三位
	IY	ISO 年的最后两位
	I	ISO 年的最后一位
	RR	年的后两位（可在 21 世纪存储 20 世纪的年份）
	RRRR	可接收 4 位年或两位年。若是两位，则和 RR 的返回值相同，若是四位，则和 YYYY 相同。

类别	模式	描述
	BC 或 B.C. AD 或 A.D.	纪元标识。BC（公元前），AD（公元后）。
月	MONTH	全长大写月份名（空白填充为 9 字符）
	MON	大写缩写月份名（3 字符）
	MM	月份数（01-12）
	RM	罗马数字的月份（I-XII；I=JAN）（大写）
天	DAY	全长大写日期名（空白填充为 9 字符）
	DY	缩写大写日期名（3 字符）
	DDD	一年里的日（001-366）
	DD	一个月里的日（01-31）
	D	一周里的日（1-7；周日是 1）
周	W	一个月里的周数（1-5）（第一周从该月第一天开始）
	WW	一年里的周数（1-53）（第一周从该年的第一天开始）
	IW	ISO 一年里的周数（第一个星期四在第一周里）
世纪	CC	世纪（2 位）（21 世纪从 2001-01-01 开始）
儒略日	J	儒略日（自公元前 4712 年 1 月 1 日来的天数）
季度	Q	季度

说明：

- 上表中 RR 计算年的规则如下：
- 输入的两位年份在 00~49 之间：当前年份的后两位在 00~49 之间，返回值年份的前两位和当前年份的前两位相同；当前年份的后两位在 50~99 之间，返回值年份的前两位是当前年份的前两位加 1。
- 输入的两位年份在 50~99 之间：当前年份的后两位在 00~49 之间，返回值年份的前两位是当前年份的前两位减 1；当前年份的后两位在 50~99 之间，返回值年份的前两位和当前年份的前两位相同。

4.1.4.2.JSON/JSONB 函数和操作符

JSON/JSONB 通用操作符参考表 1，JSONB 额外支持操作符参考表 2。

表 1 JSON/JSONB 通用操作符

操作符	左操作数类型	右操作数类型	返回类型	描述	例子	例子结果
->	Array-json(b)	int	json(b)	获得 array-json 元素。下标不存在返回空。	'[{ “a” : “foo” },{ “b” : “bar” },{ “c” : “baz” }]'::json->2	{ “c” : “baz” }
->	object-json(b)	text	json(b)	通过键获得值。不存在则返回空。	'{ “a” : { “b” : “foo” }}'::json->'a'	{ “b” : “foo” }
->>	Array-json(b)	int	text	获得 JSON 数组元素。下标不存在返回空。	'[1,2,3]'::json->>2	3
-	object	tex	te	通过键获得	'{ “a” :1, “b” :2}'::json->>'b'	2

操作符	左操作数类型	右操作数类型	返回类型	描述	例子	例子结果
>>	-json(b)	t	xt	值。不存在则返回空。		
#>	container-json(b)	text[]	json(b)	获取在指定路径的JSON 对象，路径不存在则返回空。	'{ “a” : { “b” :{ “c” : “foo” }}}'::json #>'{a,b}'	{ “c” : “foo” }
#>>	container-json(b)	text[]	text	获取在指定路径的JSON 对象，路径不存在则返回空。	'{ “a” :[1,2,3], “b” :[4,5,6]}'::json #>>'{a,2}'	3

【须知】

- 对于 #> 和 #>> 操作符，当给出的路径无法查找到数据时，不会报错，会返回空。
- 对于 MySQL 兼容性（即初始化数据库时指定 dbcompatibility='B'），#> 与 #>> 操作符应在前面加空格，否则会将井号（#）识别为操作符前字符串的一部分。

表 2 JSONB 额外支持操作符

操作符	右操作数类型	描述	例子
@>	jsonb	左边的 JSON 的顶层是否包含右边	'{ “a” :1,

操作符	右操作数类型	描述	例子
		JSON 的顶层所有项。	“b” :2}'::jsonb @> '{ “b” :2}'::jsonb
<@	jsonb	左边的 JSON 的所有项是否全部存在于右边 JSON 的顶层。	'{ “b” :2}'::jsonb <@ '{ “a” :1, “b” :2}'::jsonb
?	text	键/元素的字符串是否存在于 JSON 值的顶层。	'{ “a” :1, “b” :2}'::jsonb ? 'b'
?		text[]	这些数组字符串中的任何一个是否做为顶层键存在。
?&	text[]	是否所有这些数组字符串都作为顶层键存在。	'[“a” , “b”]'::jsonb ?& array['a', 'b']
=	jsonb	判断两个 jsonb 的大小关系，同函数 jsonb_eq。	/
<>	jsonb	判断两个 jsonb 的大小关系，同函数 jsonb_ne。	/
<	jsonb	判断两个 jsonb 的大小关系，同函数 jsonb_lt。	/
>	jsonb	判断两个 jsonb 的大小关系，同函数 jsonb_gt。	/
<=	jsonb	判断两个 jsonb 的大小关系，同函数 jsonb_le。	/
>=	jsonb	判断两个 jsonb 的大小关系，同函数 jsonb_ge。	/

JSON/JSONB 支持的函数

- array_to_json(anyarray [, pretty_bool])

  描述：返回 JSON 类型的数组。一个多维数组成为一个 JSON 数组的数组。

  如果 pretty_bool 为 true，将在一维元素之间添加换行符。

  返回类型：json

  示例：

```
SELECT array_to_json('{{1,5},{99,100}}'::int[]);
```

当结果显示如下信息，则表示函数调用成功。

```
  array_to_json
 ------------------
  [[1,5],[99,10  ]]
 (1 row)
```

❖ row_to_json(record [, pretty_bool])

描述：返回 JSON 类型的行。如果 pretty_bool 为 true，将在第一级元素之间添加换行符。

返回类型：json

示例：

```
SELECT row_to_json(row(1,'foo'));
```

当结果显示如下信息，则表示函数调用成功。

```
   row_to_json
 ---------------------
  {"f1":1,"f2":"foo"}
(1 row)
```

❖ json_array_element(array-json, integer)、
jsonb_array_element(array-jsonb, integer)

描述：同操作符'->', 返回数组中指定下标的元素。

返回类型：json、jsonb

示例：

```
select json_array_          ('[1,true,[1,[2,3]],null]',2);
```

当结果显示如下信息，则表示函数调用成功。

```
   json_array_element
--------------------
   [1,[2,3]]
(1 row)
```

❖ json_array_element_text(array-json, integer)、
jsonb_array_element_text(array-jsonb, integer)

描述：同操作符'->>', 返回数组中指定下标的元素。

返回类型：text、text

示例：

```
select json_array_          _text('[1,true,[1,[2,3]],null]',2);
```

当结果显示如下信息，则表示函数调用成功。

```
 json_array_element_text
 ----------------------
  [1,[2,3]]
(1 row)
```

- json_object_field(object-json, text)、jsonb_object_field(object-jsonb, text)

  描述：同操作符'->', 返回对象中指定键对应的值。

  返回类型：json、json

  示例：

```
select json_object_      ('{"a": {"b":"foo"}}','a');
```

当结果显示如下信息，则表示函数调用成功。

```
 json_object_field
 ------------------
 {"b":"foo"}
 (1 row)
```

- json_object_field_text(object-json, text)、jsonb_object_field_text(object-jsonb, text)

  描述：同操作符'->>', 返回对象中指定键对应的值。

  返回类型：text、text

  示例：

```
select json_object_field_text('{"a": {"b":"foo"}}','a');
```

当结果显示如下信息，则表示函数调用成功。

```
  json_object_field_text
----------------------
```

```
 {"b":"     "}
(1 row)
```

❖ json_extract_path(json, VARIADIC text[])、jsonb_extract_path((jsonb, VARIADIC text[])

描述：等价于操作符'#>'。根据$2 所指的路径，查找 json，并返回。

返回类型：json、jsonb

示例：

```
select json_extract_path('{"f2":{"f3":1},"f4":{"f5":99,"f6":"stringy"}}', 'f4','f6');
```

当结果显示如下信息，则表示函数调用成功。

```
  json_extract_path
------------------
  "stringy"
(1 row)
```

❖ json_extract_path_text(json, VARIADIC text[])、jsonb_extract_path_text((jsonb, VARIADIC text[])

描述：等价于操作符'#>>'。根据$2 所指的路径，查找 json，并返回。

返回类型：text、text

示例：

```
select json_extract_path_text('{"f2":{"f3":1},"f4":{"f5":99,"f6":"stringy"}}', 'f4','f6');
```

当结果显示如下信息，则表示函数调用成功。

```
 json_extract_path_text
 ----------------------
 "stringy"
(1 row)
```

❖ json_array_elements(array-json)、jsonb_array_elements(array-jsonb)

描述：拆分数组，每一个元素返回一行。

返回类型：json、jsonb

示例：

```
select json_array_elements('[1,      ,[1,[2,3]],null]');
```

当结果显示如下信息，则表示函数调用成功。

```
 json_array_elements
 --------------------
 1
 true
 [1,[2,3]]
 null
 (4 rows)
```

❖ json_array_elements_text(array-json)、
jsonb_array_elements_text(array-jsonb)

描述：拆分数组，每一个元素返回一行。

返回类型：text、text

示例：

```
select * from       _array_elements_text('[1,true,[1,[2,3]],null]');
```

当结果显示如下信息，则表示函数调用成功。

```
  value
 -----------
 1
 true
 [1,[2,3]]
 (4 rows)
```

❖ json_array_length(array-json)、jsonb_array_length(array-jsonb)

描述：返回数组长度。

返回类型：integer

示例：

```
SELECT json_       _length('[1,2,3,{"f1":1,"f2":[5,6]},4,null]');
```

当结果显示如下信息，则表示函数调用成功。

```
 json_array_length
 ------------------
                 6
 (1 row)
```

❖ json_each(object-json)、jsonb_each(object-jsonb)

描述：将对象的每个键值对拆分转换成一行两列。

返回类型：setof(key text, value json)、setof(key text, value jsonb)

示例：

```
select * from  json_each('{"f1":[1,2,3],"f2":{"f3":1},"f4":null}');
```

当结果显示如下信息，则表示函数调用成功。

```
 key | value
-----+----------
 f1  | [1,2,3]
 f2  | {"f3":1}
 f4  | null
(3 rows)
```

❖ json_each_text(object-json)、jsonb_each_text(object-jsonb)

描述：将对象的每个键值对拆分转换成一行两列。

返回类型：setof(key text, value text)、setof(key text, value text)

示例：

```
select * from  json_each_text('{"f1":[1,2,3],"f2":{"f3":1},"f4":null}');
```

当结果显示如下信息，则表示函数调用成功。

```
 key | value
-----+----------
 f1  | [1,2,3]
 f2  | {"f3":1}
 f4  |
(3 rows)
```

❖ json_object_keys(object-json)、jsonb_object_keys(object-jsonb)

描述：返回对象中顶层的所有键。

返回类型：SETOF text

示例：

```
select json_object_keys('{"f1":"     ","f2":{"f3":"a", "f4":"b"}, "f1":"abcd"}');
```

当结果显示如下信息，则表示函数调用成功。

```
json_object_keys
------------------
   f1
   f2

(3 rows)
```

❖ jsonb 中会有去重操作

```
select jsonb_object_keys('{"f1":"abc","f2":{"f3":"a", "f4":"b"}, "f1":"abcd"}');
```

当结果显示如下信息，则表示函数调用成功。

```
 jsonb_object_keys
-------------------
   f1
   f2
  (2 rows)
```

❖ json_populate_record(anyelement, object-json [, bool])、jsonb_populate_record(anyelement, object-jsonb [, bool])

描述：$1 必须是一个复合类型的参数。将会把 object-json 里的每个对键值进行拆分，以键当做列名，与$1 中的列名进行匹配查找，并填充到$1 的格式中。

返回类型：anyelement、anyelement

示例：

1、创建类型 jpop。

```
create type jpop as (a text, b int, c bool);
```

2、查询验证。

```
select * from json_              _record(null::jpop,'{"a":"blurfl","x":43.2}');
```

当结果显示如下信息，则表示函数调用成功。

```
    a    |b|c
---------+---+---
   blurfl| |
(1 row)
```

3、查询验证。

```
select * from json_            _record((1,1,null)::jpop,'{"a":"blurfl","x":43.2}');
```

当结果显示如下信息，则表示函数调用成功。

```
    a    | b | c
---------+---+---
  blurfl | 1 |
(1 row)
```

❖ json_populate_record_set(anyelement, array-json [, bool])、jsonb_populate_record_set(anyelement, array-jsonb [, bool])

描述：参考上述函数 json_populate_record、jsonb_populate_record，对$2 数组的每一个元素进行上述参数函数的操作，因此这也要求$2 数组的每个元素都是 object-json 类型的。

返回类型：setof anyelement、setof anyelement

示例：

1、创建类型 jpop

```
create type jpop as (a text, b     , c bool);
```

2、查询验证

```
select * from json_populate_recordset(null::jpop, '[{"a":1,"b":2},{"a":3,"b":4}]');
```

当结果显示如下信息，则表示函数调用成功。

```
 a | b | c
---+---+---
 1 | 2 |
 3 | 4 |
```

❖ json_typeof(json)、jsonb_typeof(jsonb)

描述：检测 json 类型

返回类型：text、text

示例：

```
select value, json_typeof(value)        (values (json '123.4'), (json '"foo"'), (json 'tr
    ue'), (json 'null'), (json '[1, 2, 3]'), (json '{"x":"foo", "y":123}'), (NULL::json))  a
    s data(value);
```

当结果显示如下信息，则表示函数调用成功。

```
       value       | json_typeof
-------------------+-------------
  123.4            | number
  "foo"            | string
  true             | boolean
  null             | null
  [1, 2, 3]        |
  {"x":"foo", "y":123} | object
                   |
 (7 rows)
```

❖ json_build_array([VARIADIC “any”])

描述：从一个可变参数列表构造出一个 JSON 数组。

返回类型：array-json

示例：

```
select json_build_array('a',1,'b',1.2,   ,true,'d',null,'e',json '{"x": 3, "y": [1,2,3]}','');
```

当结果显示如下信息，则表示函数调用成功。

```
                    json_build_array
 ------------------------------------------------------------------------------
 ["a", 1, "b", 1.2, "c", true, "d",       , "e", {"x": 3, "y": [1,2,3]}, null]
(1 row)
```

❖ json_build_object([VARIADIC “any”])

描述：从一个可变参数列表构造出一个 JSON 对象，其入参必须为偶数个，两两一组组成键值对。注意键不可为 null。

返回类型：object-json

示例：

```
select json_build_        (1,2);
```

当结果显示如下信息，则表示函数调用成功。

```
  json_build_object
------------------
  {"1" : 2}
(1 row)
```

- json_to_record(object-json, bool)

  描述：正如所有返回 record 的函数一样，调用者必须用一个 AS 子句显式地定义记录的结构。会将 object-json 的键值对进行拆分重组，把键当做列名，去匹配填充 as 显示指定的记录的结构。

  返回类型：record

  示例：

```
select * from json_to_record('{"a":1,"b":"foo","c":"bar"}',true) as x(a int, b text, d
    text);
```

当结果显示如下信息，则表示函数调用成功。

```
  a | b |d
 ---+-----+---
  1 |foo |
(1 row)
```

- json_to_recordset(array-json, bool)

  描述：参考函数 json_to_record，对数组内个每个元素，执行上述函数的操作，因此这要求数组内的每个元素都得是 object-json。

  返回类型：setof record

  示例：

```
select * from json_to_recordset('[{"a":1,"b":"foo","d":false},{"a":2,"b":"bar","c":tr
    ue}]',false) as x(a int, b text, c             );
```

当结果显示如下信息，则表示函数调用成功。

```
  a | b |c
 ---+-----+---
  1 |foo |
  2 |bar |t
(2 rows)
```

❖ json_object(text[])、json_object(text[], text[])

描述：从一个文本数组构造一个 object-json。这是个重载函数，当入参为一个文本数组的时候，其数组长度必须为偶数，成员被当做交替出现的键/值对。两个文本数组的时候，第一个数组认为是键，第二个认为是值，两个数组长度必须相等。键不可为 null。

返回类型：object-json

示例 1：

```
select json_object('{a,1,b,2,3,NULL,"d e f","a b c"}');
```

当结果显示如下信息，则表示函数调用成功。

```
                json_object
-----------------------------------------------------
   {"a" : "1", "b" : "2", "3" :      , "d e f" : "a b c"}
(1 row)
```

示例 2：

```
select json_object('{a,b,"   b c"}', '{a,1,1}');
```

当结果显示如下信息，则表示函数调用成功。

```
           json_object
-------------------------------------
   {"a" : "a", "b" : "1"  "a b c" : "1"}
(1 row)
```

❖ json_agg(any)

描述：将值聚集为 json 数组。

返回类型：array-json

示例：

1、创建测试表 class。

```
CREATE TABLE class(name varchar,score int);
INSERT INTO class(name,score) values ('A' ,'2');
INSERT INTO class(name,score) values ('A' ,'3');
INSERT INTO class(name,score  values ('D' ,'5');
INSERT INTO class(name,score) values ('D' ,'');
```

2、调用函数进行验证。

```
select name, json_agg(score),score from classes group by name order by name;
```

当结果显示如下信息，则表示函数调用成功。

```
 name | json_agg
------+-----------
 A    | [2, 3]
 D    | [5, null]
(2 rows)
```

❖ json_object_agg(any, any)

描述：将值聚集为 json 对象。

返回类型：object-json

示例：

1、创建测试表 class。

```
CREATE TABLE class(name varchar,score int);
INSERT INTO class(name,score) values ('A' ,'2');
INSERT INTO class(name,score) values ('A' ,'3');
INSERT INTO class(name,score) values ('D' ,'5');
INSERT INTO class(name,score) values ('D' ,'');
```

2、调用函数进行验证。

```
select json_object_      (name, score) from class group by name order by name;
```

当结果显示如下信息，则表示函数调用成功。

```
      json_object_agg
------------------------
   { "A" : 2, "A" : 3 }
   { "D" : 5, "D" : null }
(2 rows)
```

❖ jsonb_contained(jsonb, jsonb)

描述：同操作符 '<@', 判断$1 中的所有元素是否在$2 的顶层存在。

返回类型：bool

示例：

```
select jsonb_contain                    ;
```

当结果显示如下信息，则表示函数调用成功。

```
 jsonb_contained
-----------------
 t
(1 row)
```

❖ jsonb_contains(jsonb, jsonb)

描述：同操作符 '@>', 判断$1 中的顶层所有元素是否包含在$2 的所有元素。

返回类型：bool

示例：

```
select jsonb_contains('[1,2,3,4]', '[1,2,3]');
```

当结果显示如下信息，则表示函数调用成功。

```
 jsonb_contains
----------------
 t
(1 row)
```

❖ jsonb_exists(jsonb, text)

描述：同操作符 '?', 字符串$2 是否存在$1 的顶层以 keyelemscalar 的形式存在。

返回类型：bool

示例：

```
select jsonb_exists('["1",2,3]', '1');
```

当结果显示如下信息，则表示函数调用成功。

```
 jsonb_exists
--------------
 t
(1 row)
```

❖ jsonb_exists_all(jsonb, text[])

描述：同操作符 '?&', 字符串数组$2 里面，是否所有的元素，都在$1 的顶层以 keyelemscalar 的形式存在。

返回类型：bool

示例：

```
select jsonb_exists_all('["1","2",3]', '{1, 2}');
```

当结果显示如下信息，则表示函数调用成功。

```
 jsonb_exists_all
------------------
 t
(1 row)
```

- jsonb_exists_any(jsonb, text[])

描述：同操作符 '?|', 字符串数组$2 里面，是否存在的元素，在$1 的顶层以 keyelemscalar 的形式存在。

返回类型：bool

示例：

```
select jsonb_exists_any('["1","2",3]', '{1, 2, 4}');
```

当结果显示如下信息，则表示函数调用成功。

```
 jsonb_exists_any
------------------
 t
(1 row)
```

- jsonb_cmp(jsonb, jsonb)

描述：比较大小，正数代表大于，负数代表小于，0 表示相等。

返回类型：integer

示例：

```
select jsonb_cmp('["a", "b"]', '{"a":1, "b":2}');
```

当结果显示如下信息，则表示函数调用成功。

```
 jsonb_cmp
-----------
      -1
```

```
(1 row)
```

❖ jsonb_eq(jsonb, jsonb)

描述：同操作符 '=', 比较两个值的大小。

返回类型：bool

示例：

```
select jsonb_eq('["a", "b"]', '{"a":1, "b":2}');
```

当结果显示如下信息，则表示函数调用成功。

```
 jsonb_eq
----------
 f
(1 row)
```

❖ jsonb_ne(jsonb, jsonb)

描述：同操作符 '<>', 比较两个值的大小。

返回类型：bool

示例：

```
select jsonb_ne('["a", "b"]', '{"a":1, "b":2}');
```

当结果显示如下信息，则表示函数调用成功。

```
 jsonb_ne
----------
 t
(1 row)
```

❖ jsonb_gt(jsonb, jsonb)

描述：同操作符 '>', 比较两个值的大小。

返回类型：bool

示例：

```
select jsonb_gt('["a", "b"]', '{"a":1, "b":2}');
```

当结果显示如下信息，则表示函数调用成功。

```
 jsonb_gt
----------
 f
```

```
(1 row)
```

❖ jsonb_ge(jsonb, jsonb)

描述：同操作符 '>=', 比较两个值的大小。

返回类型：bool

示例：

```
select jsonb_ge('["a", "b"]', '{"a":1, "b":2}');
```

当结果显示如下信息，则表示函数调用成功。

```
 jsonb_ge
----------
 f
(1 row)
```

❖ jsonb_lt(jsonb, jsonb)

描述：同操作符 '<', 比较两个值的大小。

返回类型：bool

示例：

```
select jsonb_lt('["a", "b"]', '{"a":1, "b":2}');
```

当结果显示如下信息，则表示函数调用成功。

```
 jsonb_lt
----------
 t
(1 row)
```

❖ jsonb_le(jsonb, jsonb)

描述：同操作符 '<=', 比较两个值的大小。

返回类型：bool

示例：

```
select jsonb_le('["a", "b"]', '{"a":1, "b":2}');
```

当结果显示如下信息，则表示函数调用成功。

```
 jsonb_le
----------
 t
```

```
(1 row)
```

❖ to_json(anyelement)

描述：把参数转换为'json'。

返回类型：json

示例：

```
select to_json('{1,5}'::text[]);
```

当结果显示如下信息，则表示函数调用成功。

```
   to_json
-----------
  ["1","5"]
(1 row)
```

❖ jsonb_hash(jsonb)

描述：对 jsonb 进行 hash 运算。

返回类型：integer

示例：

```
select jsonb_hash('[1,2,3]');
```

当结果显示如下信息，则表示函数调用成功。

```
 jsonb_hash
------------
 -559968547
(1 row)
```

❖ 其他函数

描述：gin 索引以及 jsonjsonb 聚集函数所用到的内部函数，功能不过多赘述。

```
gin_compare_jsonb
gin_consistent_jsonb
gin_consistent_jsonb_hash
gin_extract_jsonb
gin_extract_jsonb_hash
gin_extract_jsonb_query
gin_extract_jsonb_query_hash
```

```
gin_triconsistent_jsonb
gin_triconsistent_jsonb_hash

json_agg_transfn
json_agg_finalfn
json_object_agg_transfn
json_object_agg_finalfn
```

4.1.4.3.安全函数

安全函数

- gs_encrypt_aes128(encryptstr,keystr)

  描述：以 keystr 为密钥对 encryptstr 字符串进行加密，返回加密后的字符串。keystr 的长度范围为 8~16 字节，至少包含 3 种字符（大写字母、小写字母、数字、特殊字符）。

  返回值类型：text

  返回值长度：至少为 92 字节，不超过 4*[(Len+68)/3]字节，其中 Len 为加密前数据长度（单位为字节）。

  示例：

```
panweidb=# SELECT gs_encrypt_aes128('MPPDB','Asdf1234');

                  gs_encrypt_aes128
----------------------------------------------------------------------------------
    -
 gwditQLQG8NhFw4OuoKhhQJoXojhFlYkjeG0aYdSCtLCnIUgkNwvYI04KbuhmcGZ
    p8jWizBdR1vU9CspjuzI0lbz12A=
(1 row)
```

  说明：

  由于该函数的执行过程需要传入解密口令，为了安全起见，gsql 工具不会将该函数记录入执行历史。即无法在 gsql 里通过上下翻页功能找到该函数的执行历史。

- gs_encrypt(encryptstr,keystr,encrypttype)

描述：根据 encrypttype，以 keystr 为密钥对 encryptstr 字符串进行加密，返回加密后的字符串。keystr 的长度范围为 8~16 字节，至少包含 3 种字符（大写字母、小写字母、数字、特殊字符），encrypttype 可以是 aes128 或 sm4。

返回值类型：text

示例：

```
panweidb=#  SELECT gs_encrypt('MPPDB','Asdf1234','sm4');
      gs_encrypt
 ------------------------------
  ZBzOmaGA4Bb+coyucJ0B8AklShqc
(1 row)
```

说明：

由于该函数的执行过程需要传入解密口令，为了安全起见，gsql 工具不会将该函数记录入执行历史。即无法在 gsql 里通过上下翻页功能找到该函数的执行历史。

- gs_decrypt_aes128(decryptstr,keystr)

  描述：以 keystr 为密钥对 decrypt 字符串进行解密，返回解密后的字符串。解密使用的 keystr 必须保证与加密时使用的 keystr 一致才能正常解密。keystr 不得为空。

说明：

此参数需要结合 gs_encrypt_aes128 加密函数共同使用。

返回值类型：text

示例：

```
panweidb=# SELECT gs_decrypt_aes128('gwditQLQG8NhFw4OuoKhhQJoXojhFlY
    kjeG0aYdSCtLCnIUgkNwvYI04KbuhmcGZp8jWizBdR1vU9CspjuzI0lbz12A=','1
    234');
  gs_decrypt_aes128
--------------------
  MPPDB
```

```
(1 row)
```

说明：

由于该函数的执行过程需要传入解密口令，为了安全起见，gsql 工具不会将该函数记录入执行历史；即无法在 gsql 里通过上下翻页功能找到该函数的执行历史。

- gs_decrypt(decryptstr,keystr,decrypttype)

  描述：根据 decrypttype，以 keystr 为密钥对 decrypt 字符串进行解密，返回解密后的字符串。解密使用的 decrypttype 及 keystr 必须保证与加密时使用的 encrypttype 及 keystr 一致才能正常解密。keystr 不得为空。decrypttype 可以是 aes128 或 sm4。

  此函数需要结合 gs_encrypt 加密函数共同使用。

  返回值类型：text

  示例：

```
panweidb=# select gs_decrypt('ZBzOmaGA4Bb+coyucJ0B8AkIShqc','Asdf1234','s
    m4');
 gs_decrypt
--------------
 MPPDB
(1 row)
```

说明：

由于该函数的执行过程需要传入解密口令，为了安全起见，gsql 工具不会将该函数记录入执行历史；即无法在 gsql 里通过上下翻页功能找到该函数的执行历史。

- gs_password_deadline

  描述：显示当前帐户密码离过期还距离多少天。

  返回值类型：interval

  示例：

```
panweidb=# SELECT gs_password_deadline();
  gs_password_deadline
---------------------------
 83 days 17:44:32.196094
```

```
(1 row)
```

- gs_password_notifytime

  描述：显示帐户密码到期前提醒的天数。

  返回值类型：int32

- login_audit_messages

  描述：查看登录用户的登录信息。

  返回值类型：元组

  示例：

  ➢查看上一次登录认证通过的日期、时间和 IP 等信息。

```
panweidb=> select * from login_audit_messages(true);
username | database |      logintime      |    mytype    | result | client_conninfo
------------+----------+----------------------+--------------+--------+-----------
      ------
omm      | openGauss | 2020-06-29 21:56:40+08 | login_success | ok     | gsql@\[loc
      al\]
(1 row)
panweidb=> select * from login_audit_messages(true);
username | database |      logintime      |    mytype    | result | client_conninfo
------------+----------+----------------------+--------------+--------+-----------
      ------
omm      | panweidb | 2020-06-29 21:56:40+08 | login_success | ok     | gsql@\[loca
      l\]
(1 row)
panweidb=> select * from login_audit_messages(true);
  username | database |      logintime      |    mytype    | result | client_conninfo
------------+----------+----------------------+--------------+--------+-----------
      ------
  omm      | openGauss | 2020-06-29 21:56:40+08 | login_success | ok     | gsql@\[l
      ocal\]
(1 row)
```

  ➢ 查看上一次登录认证失败的日期、时间和 IP 等信息。

```
panweidb=>  select * from login_audit_messages(false) ORDER BY logintime des
      c limit 1;
```

```
 username | database |      logintime       |    mytype    | result |  client_conninfo----
     --------+----------+----------------------+--------------+--------+-----------
     --------
 omm      | openGauss | 2020-06-29 21:57:55+08 | login_failed | failed | [unknown]
     @\[local\]
(1 row)

panweidb=>  select * from login_audit_messages(false) ORDER BY logintime des
     c limit 1;
 username | database |      logintime       |    mytype    | result |  client_conninfo
------------+----------+----------------------+--------------+--------+------------
     -------
 omm      | panweidb | 2020-06-29 21:57:55+08 | login_failed | failed | [unknown]
     @\[local\]
(1 row)

panweidb=>  select * from login_audit_messages(false) ORDER BY logintime des
     c limit 1;
username | database |      logintime       |    mytype    | result |  client_conninfo
------------+----------+----------------------+--------------+--------+------------
     -------
  omm      | openGauss | 2020-06-29 21:57:55+08 | login_failed | failed | [unknown]
     @\[local\]
(1 row)
panweidb=>  select * from login_audit_messages(false);
 username | database |      logintime       |    mytype    | result |  client_conninfo
 ------------+----------+----------------------+--------------+--------+-----------
     --------
 omm      | openGauss | 2020-06-29 21:57:55+08 | login_failed | failed | [unknown]
     @\[local\]
 omm      | openGauss | 2020-06-29 21:57:53+08 | login_failed | failed | [unknown]
     @\[local\]
(2 rows)
panweidb=>  select * from login_audit_messages(false);
  username | database |      logintime       |    mytype    | result |  client_conninfo
```

```
------------+----------+----------------------+--------------+--------+-----------
    -------
 omm     | panweidb | 2020-06-29 21:57:55+08 | login_failed | failed | [unknown]
    @\[local\]
omm     | panweidb | 2020-06-29 21:57:53+08 | login_failed | failed | [unknown]@
    \[local\]
(2 rows)
```

➢ 查看自从最后一次认证通过以来失败的尝试次数、日期和时间。

```
panweidb=>  select * from login_audit_messages(false);
 username | database |      logintime      |    mytype    | result |  client_conninfo
------------+----------+----------------------+--------------+--------+-----------
    -------
 omm     | openGauss | 2020-06-29 21:57:55+08 | login_failed | failed | [unknown]
    @\[local\]
 omm     | openGauss | 2020-06-29 21:57:53+08 | login_failed | failed | [unknown]
    @\[local\]
(2 rows)
```

❖ login_audit_messages_pid

描述：查看登录用户的登录信息。与 login_audit_messages 的区别在于结果基于当前 backendid 向前查找。所以不会因为同一用户的后续登录，而影响本次登录的查询结果。也就是查询不到该用户后续登录的信息。

返回值类型：元组

```
panweidb=> SELECT * FROM login_audit_messages_pid(true);
 username | database |      logintime      |    mytype    | result | client_conninfo |
    backendid
------------+----------+----------------------+--------------+--------+-----------
    ------+-----------------
 omm     | openGauss | 2020-06-29 21:56:40+08 | login_success | ok     | gsql@\[lo
    cal\]   | 139823109633792
(1 row)
panweidb=> SELECT * FROM login_audit_messages_pid(true);
```

```
username | database |      logintime      |    mytype    | result | client_conninfo |   b
    ackendid
-----------+----------+---------------------+--------------+--------+-----------
    ------+----------------
omm      | panweidb | 2020-06-29 21:56:40+08 | login_success | ok     | gsql@\[loca
    l\]    | 139823109633792
(1 row)
```

示例：

- 查看上一次登录认证通过的日期、时间和 IP 等信息。

```
panweidb=> SELECT * FROM login_audit_messages_pid(false) ORDER BY loginti
    me desc limit 1;
username | database |      logintime      |    mytype    | result |  client_conninfo  |
    backendid
-----------+----------+---------------------+--------------+--------+------------
    -------+-----------------
omm      | openGauss | 2020-06-29 21:57:55+08 | login_failed | failed | [unknown]
    @\[local\] | 139823109633792
(1 row)
panweidb=> SELECT * FROM login_audit_messages_pid(false) ORDER BY loginti
    me desc limit 1;
username | database |      logintime      |    mytype    | result |  client_conninfo  |
    backendid
-----------+----------+---------------------+--------------+--------+------------
    -------+-----------------
omm      | panweidb | 2020-06-29 21:57:55+08 | login_failed | failed | [unknown]@
    \[local\] | 139823109633792
(1 row)

panweidb=> SELECT * FROM login_audit_messages_pid(true);
  username | database |      logintime      |    mytype    | result | client_conninfo |
     backendid
-----------+----------+---------------------+--------------+--------+-----------
    ------+-----------------
```

```
 omm     | openGauss | 2020-06-29 21:56:40+08 | login_success | ok    | gsql@\[l
     ocal\]   | 139823109633792
(1 row)
panweidb=> SELECT * FROM login_audit_messages_pid(false);
username | database |      logintime      |   mytype   | result |  client_conninfo  |
     backendid
-----------+----------+----------------------+-------------+--------+-----------
     -------+-----------------
omm     | openGauss | 2020-06-29 21:57:55+08 | login_failed | failed | [unknown]
     @\[local\] | 139823109633792
omm     | openGauss | 2020-06-29 21:57:53+08 | login_failed | failed | [unknown]
     @\[local\] | 139823109633792
     (2 rows)
panweidb=> SELECT * FROM login_audit_messages_pid(false);
username | database |      logintime      |   mytype   | result |  client_conninfo  |
     backendid
-----------+----------+----------------------+-------------+--------+-----------
     -------+-----------------
omm     | panweidb | 2020-06-29 21:57:55+08 | login_failed | failed | [unknown]@
     \[local\] | 139823109633792
omm     | panweidb | 2020-06-29 21:57:53+08 | login_failed | failed | [unknown]@
     \[local\] | 139823109633792
     (2 rows)
```

➢ 查看上一次登录认证失败的日期、时间和 IP 等信息。

```
panweidb=> SELECT * FROM login_audit_messages_pid(false) ORDER BY loginti
     me desc limit 1;
 username | database |      logintime      |   mytype   | result |  client_conninfo  |
      backendid
-----------+----------+----------------------+-------------+--------+-----------
     -------+-----------------
 omm     | openGauss | 2020-06-29 21:57:55+08 | login_failed | failed | [unknown]
     @\[local\] | 139823109633792
(1 row)
```

➢ 查看自从最后一次认证通过以来失败的尝试次数、日期和时间。

```
panweidb=> SELECT * FROM login_audit_messages_pid(false);
```

```
 username | database |     logintime      |   mytype   | result | client_conninfo |
     backendid
-----------+----------+---------------------+-------------+--------+-----------
    -------+-----------------
 omm       | openGauss | 2020-06-29 21:57:55+08 | login_failed | failed | [unknown]
    @\[local\] | 139823109633792
 omm       | openGauss | 2020-06-29 21:57:53+08 | login_failed | failed | [unknown]
    @\[local\] | 139823109633792
(2 rows)
```

- inet_server_addr

  描述：显示服务器 IP 信息。

  返回值类型：inet

  示例：

```
panweidb=# SELECT inet_server_addr();
 inet_server_addr
--------------------
 10.10.0.13
(1 row)
```

  说明：

- 上面是以客户端在 10.10.0.50 上，服务器端在 10.10.0.13 上为例。
- 如果是通过本地连接，使用此接口显示为空。
- inet_client_addr

  描述：显示客户端 IP 信息。

  返回值类型：inet

  示例：

```
panweidb=# SELECT inet_client_addr();
 inet_client_addr
--------------------
 10.10.0.50
(1 row)
```

  说明：

- 上面是以客户端在 10.10.0.50 上，服务器端在 10.10.0.13 上为例。

- 如果是通过本地连接，使用此接口显示为空。
- pg_query_audit

  描述：查看数据库主节点审计日志。

  返回值类型：record

  函数返回字段如下：

名称	类型	描述
time	timestamp with time zone	操作时间
type	text	操作类型
result	text	操作结果
userid	oid	用户 id
username	text	执行操作的用户名
database	text	数据库名称
client_conninfo	text	客户端连接信息
object_name	text	操作对象名称
detail_info	text	执行操作详细信息
node_name	text	节点名称
thread_id	text	线程 id
local_port	text	本地端口
remote_port	text	远端端口

  函数使用方法及示例请参考：《PanWeiDB V2.0-S3.0.1_B01 管理员指南->日志参考->审计日志->查看审计结果》章节。

- pg_delete_audit

  描述：删除指定时间段的审计日志。

返回值类型：void

函数使用方法及示例请参考：《PanWeiDB V2.0-S3.0.1_B01 管理员指南->日志参考->审计日志->维护审计日志》

- gs_sm3(text)

  描述：用来计算输入字符串的 SM3 摘要。

  返回值类型：SM3 哈希值的十六进制字符串，长度固定为 64。

  示例：

```
select gs_sm3('abcdef');
```

  结果显示如下：

```
                gs_sm3
------------------------------------------------------------------
 5d60e23c9fe29b5e62517e144ad67541c6eb132c8926637b6393fe8d9b62b3bf
(1 row)
```

4.1.4.4.系统信息函数

4.1.4.4.1. 会话信息函数

- current_catalog

  描述：当前数据库的名称（在标准 SQL 中称“catalog”）。

  返回值类型：name

  示例：

```
panweidb=# SELECT current_catalog;
 current_database
------------------
 panweidb
(1 row)
```

- current_database()

  描述：当前数据库的名称。

  返回值类型：name

  示例：

```
panweidb=# SELECT current_database();
 current_database
```

```
-----------------
 panweidb
(1 row)
```

❖ current_query()

描述：由客户端提交的当前执行语句（可能包含多个声明）。

返回值类型：text

示例：

```
panweidb=# SELECT current_query();
    current_query
------------------------
 SELECT current_query();
(1 row)
```

❖ current_schema[()]

描述：当前模式的名称。

返回值类型：name

示例：

```
panweidb=# SELECT current_schema();
 current_schema
----------------
 public
(1 row)
```

备注：current_schema 返回在搜索路径中第一个顺位有效的模式名。（如果搜索路径为空则返回 NULL，没有有效的模式名也返回 NULL）。如果创建表或者其他命名对象时没有声明目标模式，则将使用这些对象的模式。

❖ current_schemas(Boolean)

描述：搜索路径中的模式名称。

返回值类型：name[]

示例：

```
panweidb=# SELECT current_schemas(true);
  current_schemas
---------------------
 {pg_catalog,public}
```

```
(1 row)
```

备注：

current_schemas(Boolean)返回搜索路径中所有模式名称的数组。布尔选项决定像 pg_catalog 这样隐含包含的系统模式是否包含在返回的搜索路径中。

搜索路径可以通过运行时设置更改。命令是：

```
SET search_path TO schema [, schema, ...]
```

❖ current_user

描述：当前执行环境下的用户名。

返回值类型：name

示例：

```
panweidb=# SELECT current_user;

 current_user
--------------
 panweidb
(1 row)
```

备注：current_user 是用于权限检查的用户标识。通常，他表示会话用户，但是可以通过参考：开发者指南->SQL 语法参考->SQL 语法章节改变他。在函数执行的过程中随着属性 SECURITY DEFINER 的改变，其值也会改变。

❖ definer_current_user

描述：当前执行环境下的用户名。

返回值类型：name

示例：

```
panweidb=# SELECT definer_current_user();
 definer_current_user
----------------------
 panweidb
(1 row)
```

❖ pg_current_sessionid()

描述：当前执行环境下的会话 ID。

返回值类型：text

示例：

```
panweidb=# SELECT pg_current_sessionid();
  pg_current_sessionid
---------------------------
 1659507037.139786426119936
(1 row)
```

备注：pg_current_sessionid()是用于获取当前执行环境下的会话 ID。其组成结构为：时间戳.会话 ID，当线程池模式开启（enable_thread_pool=on）时，会话 ID 为 SessionID；而线程池模式关 闭时，会话 ID 为 ThreadID。

- pg_current_sessid

描述：当前执行环境下的会话 ID。

返回值类型：text

示例：

```
panweidb=# select pg_current_sessid();
pg_current_sessid
-------------------
140308875015936
(1 row)
```

备注：在线程池模式下获得当前会话的会话 ID，非线程池模式下获得当 前会话对应的后台线程 ID。

- pg_current_userid

描述：当前用户 ID。

返回值类型：text

```
panweidb=# SELECT pg_current_userid();
pg_current_userid
-------------------
10
(1 row)
```

- working_version_num()

描述：版本序号信息。返回一个系统兼容性有关的版本序号。

返回值类型：int

示例：

```
panweidb=# SELECT working_version_num();
 working_version_num
---------------------
               92231
(1 row)
```

- tablespace_oid_name()

  描述: 根据表空间 oid，查找表空间名称。

  返回值类型：text

  示例：

```
panweidb=# select tablespace_oid_name(1663);
 tablespace_oid_name
---------------------
 pg_default
(1 row)
```

- inet_client_addr()

  描述：连接的远端地址。inet_client_addr 返回当前客户端的 IP 地址。

说明

此函数只有在远程连接模式下有效。

返回值类型：inet

示例：

```
panweidb=# SELECT inet_client_addr();
 inet_client_addr
------------------

(1 row)
```

- inet_client_port()

  描述：连接的远端端口。inet_client_port 返回当前客户端的端口号。

说明

此函数只有在远程连接模式下有效。

返回值类型：int

示例：

```
panweidb=# SELECT inet_client_port();
 inet_client_port
------------------
            33143
(1 row)
```

❖ inet_server_addr()

描述：连接的本地地址。inet_server_addr 返回服务器接收当前连接用的 IP 地址。

说明

此函数只有在远程连接模式下有效。

返回值类型：inet

示例：

```
panweidb=# SELECT inet_server_addr();
 inet_server_addr
------------------
 10.10.0.13
(1 row)
```

❖ inet_server_port()

描述：连接的本地端口。inet_server_port 返回接收当前连接的端口号。

如果是通过 Unix-domain socket 连接的，则所有这些函数都返回 NULL。

说明

此函数只有在远程连接模式下有效。

返回值类型：int

示例：

```
panweidb=# SELECT inet_server_port();
 inet_server_port
------------------


(1 row)
```

❖ pg_backend_pid()

描述：当前会话连接的服务进程的进程 ID。

返回值类型：bigint

示例：

```
panweidb=# SELECT pg_backend_pid();
 pg_backend_pid
-----------------
 140229352617744
(1 row)
```

❖ pg_conf_load_time()

描述：配置加载时间。pg_conf_load_time 返回最后加载服务器配置文件的时间戳。

返回值类型：timestamp with time zone

示例：

```
panweidb=# SELECT pg_conf_load_time();


    pg_conf_load_time
-------------------------------
 2022-08-03 10:27:43.7626+08
(1 row)
```

❖ pg_my_temp_schema()

描述：会话的临时模式的 OID，不存在则为 0。

返回值类型：oid

示例：

```
panweidb=# SELECT pg_my_temp_schema();
 pg_my_temp_schema
-------------------
                 0
(1 row)
```

备注：pg_my_temp_schema 返回当前会话中临时模式的 OID，如果不存在（没有创建临时表）的话则返回 0。如果给定的 OID 是其他会话中临时模式的 OID，pg_is_other_temp_schema 则返回 true。

- pg_is_other_temp_schema(oid)

  描述：是否为另一个会话的临时模式。

  返回值类型：Boolean

  示例：

```
panweidb=# SELECT pg_is_other_temp_schema(25356);
 pg_is_other_temp_schema
-------------------------
 f
(1 row)
```

- pg_listening_channels()

  描述：会话正在侦听的信道名称。

  返回值类型：setof text

  示例：

```
panweidb=# SELECT pg_listening_channels();
 pg_listening_channels
-----------------------
(0 rows)
```

备注：pg_listening_channels 返回当前会话正在侦听的一组信道名称。

- pg_postmaster_start_time()

  描述：服务器启动时间。pg_postmaster_start_time 返回服务器启动时的 timestamp with time zone。

  返回值类型：timestamp with time zone

  示例：

```
panweidb=# SELECT pg_postmaster_start_time();
  pg_postmaster_start_time
-------------------------------
 2022-08-03 10:27:44.54975+08
(1 row)
```

❖ pg_get_ruledef(rule_oid)

描述：获取规则的 CREATE RULE 命令。

返回值类型：text

示例：

```
panweidb=# select * from pg_get_ruledef(24828);
                        pg_get_ruledef
-------------------------------------------------------------------
 CREATE RULE t1_ins AS ON INSERT TO t1 DO INSTEAD INSERT INTO t2 (id) VALUES (n
ew.id);
(1 row)
```

❖ sessionid2pid()

描述: 从 sessionid 中得到 pid 信息（例如，gs_session_stat 中 sessid 列）。

返回值类型: int8

示例：

```
panweidb=# select sessionid2pid(sessid::cstring) from gs_session_stat limit 2;
 sessionid2pid
-----------------
 139973107902208
 139973107902208
(2 rows)
```

❖ pg_trigger_depth()

描述：触发器的嵌套层次。

返回值类型：int

示例：

```
panweidb=# SELECT pg_trigger_depth();
 pg_trigger_depth
------------------
```

```
          0
(1 row)
```

❖ session_user

描述：会话用户名。

返回值类型：name

示例：

```
panweidb=# SELECT session_user;
 session_user
--------------
 omm
(1 row)
```

备注：session_user 通常是连接当前数据库的初始用户，不过系统管理员可以用开发者指南->SQL 语法参考->SQL 语法->SET-SESSION-AUTHORIZATION 章节修改这个设置。

❖ user

描述：等价于 current_user。

返回值类型：name

示例：

```
panweidb=# SELECT user;

 current_user
--------------
 panweidb
(1 row)
```

❖ getpgusername()

描述：获取数据库用户名。

返回值类型：name

示例：

```
panweidb=# select getpgusername();
 getpgusername
---------------
 panweidb
(1 row)
```

❖ getdatabaseencoding()

描述：获取数据库编码方式。

返回值类型：name

示例：

```
panweidb=# select getdatabaseencoding();
 getdatabaseencoding
---------------------
 UTF8
(1 row)
```

❖ version()

描述：版本信息。version 返回一个描述服务器版本信息的字符串。

返回值类型：text

示例：

```
panweidb=# select version();
                                                version
                                                
-------------------------------------------------------------------------------------------------
-----------------------------------------------
-------------------
 (PanWeiDB V2.0-S3.0.1_B01 Alpha) compiled at 2023-09-10 02:16:40 commit 7387
 last mr   on x86_64-unknown-linux-gnu, compiled by g++
(GCC) 7.3.0, 64-bit
(1 row)
```

❖ pw_version()

描述：使用 pw_version 函数查询 PanWeiDB 产品版本信息。显示内容包括产品名称、版本号、补丁号、提交号和基于 openGauss 版本的信息。

注意事项：

- 本函数只有在 PanWeiDB 数据库能正常启动及登录的前提下使用。
- 本函数大写可识别。

示例：

查询产品信息

```
select pw_version();
```

返回结果为：

```
                    pw_version
------------------------------------------------------------------
(PanWeiDB V2.0-S3.0.1_B01 ) compiled at 2023-09-25 14:51:16 commit 99
13 last mr                                    +
 product name:PanWeiDB                          +
 version:V2.0-S3.0.1_B01                      +
 patch:0                                        +
 commit:9913                                    +
 openGauss version:3.0.1                        +
 host:x86_64-pc-linux-gnu
(1 row)
```

- gs_deployment()

  描述：当前系统的部署形态信息。

  返回值类型：text

  示例：

```
panweidb=# select gs_deployment();
   gs_deployment
----------------------
 OpenSourceCentralized
(1 row)
```

- get_hostname()

  描述：返回当前节点的 hostname。

  返回值类型：text

  示例：

```
panweidb=# SELECT get_hostname();
   get_hostname
----------------------
 localhost.localdomain
(1 row)
```

- get_nodename()

  描述：返回当前节点的名字。

返回值类型：text

示例：

```
panweidb=# SELECT get_nodename();
 get_nodename
--------------
 node1
(1 row)
```

❖ get_schema_oid(cstring)

描述：返回查询 schema 的 oid。

返回值类型：oid

示例：

```
panweidb=# SELECT get_schema_oid('public');
 get_schema_oid
----------------
           2200
(1 row)
```

❖ get_client_info()

描述：返回客户端信息。

返回值类型：record

4.1.4.4.2. 访问权限查询函数

DDL 类权限 ALTER、DROP、COMMENT、INDEX、VACUUM 属于所有者固有的权限，隐式拥有。

❖ cm_any_column_privilege(user, table, privilege)

描述：指定用户是否有访问表任何列的权限。

表 1 参数类型说明

参数名	合法入参类型
user	name, oid
table	text, oid
privilege	text

返回类型：Boolean

- cm_any_column_privilege(table, privilege)

  描述：当前用户是否有访问表任何列的权限，合法参数类型见表 1。

  返回类型：Boolean

  备注：cm_any_column_privilege 检查用户是否以特定方式访问表的任何列。其参数可能与 cm_table_privilege 类似，除了访问权限类型必须是 SELECT、INSERT、UPDATE、COMMENT 或 REFERENCES 的一些组合。

说明

拥有表的表级别权限则隐含的拥有该表每列的列级权限，因此如果与 cm_table_privilege 参数相同，cm_any_column_privilege 总是返回 true。但是如果授予至少一列的列级权限也返回成功。

- cm_column_privilege(user, table, column, privilege)

  描述：指定用户是否有访问列的权限。

  表 2 参数类型说明

参数名	合法入参类型
user	name, oid
table	text, oid
column	text, smallint
privilege	text

返回类型：Boolean

- cm_column_privilege(table, column, privilege)

  描述：当前用户是否有访问列的权限，合法参数类型见表 2。

  返回类型：Boolean

  备注：cm_column_privilege 检查用户是否以特定方式访问一列。其参数类似于 cm_table_privilege，可以通过列名或属性号添加列。想要的访问

权限类型必须是 SELECT、INSERT、UPDATE、COMMENT 或 REFERENCES 的一些组合。

说明

拥有表的表级别权限则隐含的拥有该表每列的列级权限。

- cm_cek_privilege(user, cek, privilege)

  描述：指定用户是否有访问列加密密钥 CEK 的权限。参数说明如下。

  表 3 参数类型说明

参数名	合法入参类型	描述	取值范围
user	name，oid	用户	用户名字或 id。
cek	text，oid	列加密密钥	列加密密钥名称或 id。
privilege	text	权限	❖ USAGE：允许使用指定列加密密钥。 ❖ DROP：允许删除指定列加密密钥。

返回类型：Boolean

- cm_cmk_privilege(user, cmk, privilege)

  描述：指定用户是否有访问客户端加密主密钥 CMK 的权限。参数说明如下。

  表 4 参数类型说明

参数名	合法入参类型	描述	取值范围
user	name，oid	用户	用户名字或 id。
cmk	text，oid	客户端加密主密钥	客户端加密主密钥名称或 id。
privilege	text	权限	❖ USAGE：允许使用指定客户端

参数名	合法入参类型	描述	取值范围
			加密主密钥。 ❖ DROP：允许删除指定客户端加密主密钥。

返回类型：Boolean

❖ cm_database_privilege(user, database, privilege)

描述：指定用户是否有访问数据库的权限。参数说明如下。

表 5 参数类型说明

参数名	合法入参类型
user	name, oid
database	text, oid
privilege	text

返回类型：Boolean

❖ cm_database_privilege(database, privilege)

描述：当前用户是否有访问数据库的权限，合法参数类型请参见表 5。

返回类型：Boolean

备注：cm_database_privilege 检查用户是否能以在特定方式访问数据库。其参数类似 cm_table_privilege。访问权限类型必须是 CREATE、CONNECT、TEMPORARY、ALTER、DROP、COMMENT 或 TEMP（等价于 TEMPORARY）的一些组合。

❖ cm_directory_privilege(user, directory, privilege)

描述：指定用户是否有访问 directory 的权限。

表 6 参数类型说明

参数名	合法入参类型
user	name, oid
database	text, oid

参数名	合法入参类型
privilege	text

返回类型：Boolean

- cm_directory_privilege(directory, privilege)

  描述：当前用户是否有访问 directory 的权限，合法参数类型请参见表 6。

  返回类型：Boolean

- cm_foreign_data_wrapper_privilege(user, fdw, privilege)

  描述：指定用户是否有访问外部数据封装器的权限。

  表 7 参数类型说明

参数名	合法入参类型
user	name, oid
fdw	text, oid
privilege	text

返回类型：Boolean

- cm_foreign_data_wrapper_privilege(fdw, privilege)

  描述：当前用户是否有访问外部数据封装器的权限。合法参数类型请参见表 7。

  返回类型：Boolean

  备注：cm_foreign_data_wrapper_privilege 检查用户是否能以特定方式访问外部数据封装器。其参数类似 cm_table_privilege。访问权限类型必须是 USAGE。

- cm_function_privilege(user, function, privilege)

  描述：指定用户是否有访问函数的权限。

  表 8 参数类型说明

参数名	合法入参类型
user	name, oid
function	text, oid

参数名	合法入参类型
privilege	text

返回类型：Boolean

- cm_function_privilege(function, privilege)

  描述：当前用户是否有访问函数的权限。合法参数类型请参见表 8。

  返回类型：Boolean

  备注：cm_function_privilege 检查一个用户是否能以指定方式访问一个函数。其参数类似 cm_table_privilege。使用文本字符而不是 OID 声明一个函数时，允许输入的类型和 regprocedure 数据类型一样（请参考开发者指南->SQL 语法参考->SQL 基本要素->数据类型->对象标识符类型章节）。访问权限类型必须是 EXECUTE、ALTER、DROP 或 COMMENT。

- cm_language_privilege(user, language, privilege)

  描述：指定用户是否有访问语言的权限。

  表 9 参数类型说明

  返回类型：Boolean

参数名	合法入参类型
user	name, oid
language	text, oid
privilege	text

- cm_language_privilege(language, privilege)

  描述：当前用户是否有访问语言的权限。合法参数类型请参见表 9。

  返回类型：Boolean

  备注：cm_language_privilege 检查用户是否能以特定方式访问一个过程语言。其参数类似 cm_table_privilege。访问权限类型必须是 USAGE。

- cm_nodegroup_privilege(user, nodegroup, privilege)

  描述：检查用户是否有数据库节点访问权限。

返回类型：Boolean

表 10 参数类型说明

参数名	合法入参类型
user	name, oid
nodegroup	text, oid
privilege	text

- cm_nodegroup_privilege(nodegroup, privilege)

  描述：检查用户是否有数据库节点访问权限。参数与 cm_table_privilege 类似。访问权限类型必须是 USAGE、CREATE、COMPUTE、ALTER 或 CROP。

  返回类型：Boolean

- cm_schema_privilege(user, schema, privilege)

  描述：指定用户是否有访问模式的权限。

  返回类型：Boolean

- cm_schema_privilege(schema, privilege)

  描述：当前用户是否有访问模式的权限。

  返回类型：Boolean

  备注：cm_schema_privilege 检查用户是否能以特定方式访问一个模式。其参数类似 cm_table_privilege。访问权限类型必须是 CREATE、USAGE、ALTER、DROP 或 COMMENT 的一些组合。

- cm_server_privilege(user, server, privilege)

  描述：指定用户是否有访问外部服务的权限。

  返回类型：Boolean

- cm_server_privilege(server, privilege)

  描述：当前用户是否有访问外部服务的权限。

  返回类型：Boolean

  备注：cm_server_privilege 检查用户是否能以指定方式访问一个外部服务器。其参数类似 cm_table_privilege。访问权限类型必须是 USAGE、ALTER、DROP 或 COMMENT 之一的值。

- cm_table_privilege(user, table, privilege)

  描述：指定用户是否有访问表的权限。

  返回类型：Boolean

- cm_table_privilege(table, privilege)

  描述：当前用户是否有访问表的权限。

  返回类型：Boolean

  备注：cm_table_privilege 检查用户是否以特定方式访问表。用户可以通过名称或 OID（pg_authid.oid）来指定，public 表明 PUBLIC 伪角色，或如果缺省该参数，则使用 current_user。该表可以通过名称或者 OID 声明。如果用名称声明，则在必要时可以用模式进行修饰。如果使用文本字符串来声明所希望的权限类型，这个文本字符串必须是 SELECT、INSERT、UPDATE、DELETE、TRUNCATE、REFERENCES、TRIGGER、ALTER、DROP、COMMENT、INDEX 或 VACUUM 之一的值。可以给权限类型添加 WITH GRANT OPTION，用来测试权限是否拥有授权选项。也可以用逗号分隔列出的多个权限类型，如果拥有任何所列出的权限，则结果便为 true。

- cm_tablespace_privilege(user, tablespace, privilege)

  描述：指定用户是否有访问表空间的权限。

  返回类型：Boolean

- cm_tablespace_privilege(tablespace, privilege)

  描述：当前用户是否有访问表空间的权限。

  返回类型：Boolean

  备注：cm_tablespace_privilege 检查用户是否能以特定方式访问一个表空间。其参数类似 cm_table_privilege。访问权限类型必须是 CREATE、ALTER、DROP 或 COMMENT 之一的值。

- pg_cm_role(user, role, privilege)

  描述：指定用户是否有角色的权限。

  返回类型：Boolean

- pg_cm_role(role, privilege)

  描述：当前用户是否有角色的权限。

  返回类型：Boolean

备注：pg_cm_role 检查用户是否能以特定方式访问一个角色。其参数类似 cm_table_privilege，除了 public 不能用做用户名。访问权限类型必须是 MEMBER 或 USAGE 的一些组合。MEMBER 表示的是角色中的直接或间接成员关系（也就是 SET ROLE 的权限），而 USAGE 表示无需通过 SET ROLE 也直接拥有角色的使用权限。

- cm_any_privilege(user, privilege)

  描述：指定用户是否有某项 ANY 权限，若同时查询多个权限，只要具有其中一个则返回 true。

  返回类型：Boolean

  表 11 参数类型说明

参数名	合法入参类型	描述	取值范围
user	name	用户	已存在的用户名。
privilege	text	ANY 权限	可选取值： CREATE ANY TABLE [WITH ADMIN OPTION] ALTER ANY TABLE [WITH ADMIN OPTION] DROP ANY TABLE [WITH ADMIN OPTION] SELECT ANY TABLE [WITH ADMIN OPTION] INSERT ANY TABLE [WITH ADMIN OPTION] UPDATE ANY TABLE [WITH ADMIN OPTION] DELETE ANY TABLE [WITH ADMIN OPTION] CREATE ANY SEQUENCE [WITH ADMIN OPTION] CREATE ANY INDEX [WITH ADMIN OPTION] CREATE ANY FUNCTION [WITH ADMIN OPTION] EXECUTE ANY FUNCTION [WITH ADMIN OPTION] CREATE ANY PACKAGE [WITH ADMIN OPTION] EXECUTE ANY PACKAGE [WITH ADMIN OPTION]

参数名	合法入参类型	描述	取值范围
			CREATE ANY TYPE [WITH ADMIN OPTION]

4.1.4.4.3. 模式可见性查询函数

每个函数执行检查数据库对象类型的可见性。对于函数和操作符，如果在前面的搜索路径中没有相同的对象名称和参数的数据类型，则此对象是可见的。对于操作符类，则要同时考虑名称和相关索引的访问方法。

所有这些函数都需要使用 OID 来标识要需要检查的对象。如果用户想通过名称测试对象，则使用 OID 别名类型（regclass、regtype、regprocedure、regoperator、regconfig 或 regdictionary）将会很方便。

比如，如果一个表所在的模式在搜索路径中，并且在前面的搜索路径中没有同名的表，则这个表是可见的。它等效于表可以不带明确模式修饰进行引用。比如，要列出所有可见表的名称：

```
panweidb=# SELECT relname FROM pg_class WHERE pg_table_is_visible(oid);
```

- pg_collation_is_visible(collation_oid)
  描述：该排序是否在搜索路径中可见。
  返回类型：Boolean
- pg_conversion_is_visible(conversion_oid)
  描述：该转换是否在搜索路径中可见。
  返回类型：Boolean
- pg_function_is_visible(function_oid)
  描述：该函数是否在搜索路径中可见。
  返回类型：Boolean
- pg_opclass_is_visible(opclass_oid)
  描述：该操作符类是否在搜索路径中可见。
  返回类型：Boolean
- pg_operator_is_visible(operator_oid)
  描述：该操作符是否在搜索路径中可见。
  返回类型：Boolean

- pg_opfamily_is_visible(opclass_oid)

  描述：该操作符族是否在搜索路径中可见。

  返回类型：Boolean

- pg_table_is_visible(table_oid)

  描述：该表是否在搜索路径中可见。

  返回类型：Boolean

- pg_ts_config_is_visible(config_oid)

  描述：该文本检索配置是否在搜索路径中可见。

  返回类型：Boolean

- pg_ts_dict_is_visible(dict_oid)

  描述：该文本检索词典是否在搜索路径中可见。

  返回类型：Boolean

- pg_ts_parser_is_visible(parser_oid)

  描述：该文本搜索解析是否在搜索路径中可见。

  返回类型：Boolean

- pg_ts_template_is_visible(template_oid)

  描述：该文本检索模板是否在搜索路径中可见。

  返回类型：Boolean

- pg_type_is_visible(type_oid)

  描述：该类型（或域）是否在搜索路径中可见。

  返回类型：Boolean

4.1.4.4.4. 系统表信息函数

- format_type(type_oid, typemod)

  描述：获取数据类型的 SQL 名称。

  返回类型：text

  备注：format_type 通过某个数据类型的类型 OID 以及可能的类型修饰词，返回其 SQL 名称。如果不知道具体的修饰词，则在类型修饰词的位置传入 NULL。类型修饰词一般只对有长度限制的数据类型有意义。

  format_type 所返回的 SQL 名称中包含数据类型的长度值，其大小是：实际存储长度 len - sizeof(int32)，单位字节。原因是数据存储时需要 32

位的空间来存储用户对数据类型的自定义长度信息，即实际存储长度要比用户定义长度多 4 个字节。在下例中，format_type 返回的 SQL 名称为“character varying(6)”，6 表示 varchar 类型的长度值是 6 字节，因此该类型的实际存储长度为 10 字节。

```
panweidb=# SELECT format_type((SELECT oid FROM pg_type WHERE typname='varc
har'), 10);
 format_type
-------------
 varchar(6)
(1 row)
```

- getdistributekey(table_name)

  描述：获取一个 hash 表的分布列。单机环境下不支持分布，该函数返回为空。

- pg_check_authid(role_oid)

  描述：检查是否存在给定 oid 的角色名。

  返回类型：Boolean

  示例：

```
panweidb=# select pg_check_authid(1);
pg_check_authid
-----------------
f
(1 row)
```

- pg_describe_object(catalog_id, object_id, object_sub_id)

  描述：获取数据库对象的描述。

  返回类型：text

  备注：pg_describe_object 返回由目录 OID，对象 OID 和一个（或许 0 个）子对象 ID 指定的数据库对象的描述。这有助于确认存储在 pg_depend 系统表中对象的身份。

- pg_get_constraintdef(constraint_oid)

  描述：获取约束的定义。

  返回类型：text

- pg_get_constraintdef(constraint_oid, pretty_bool)

描述：获取约束的定义。

返回类型：text

备注：pg_get_constraintdef 和 pg_get_indexdef 分别从约束或索引上使用创建命令进行重构。

- pg_get_expr(pg_node_tree, relation_oid)

  描述：反编译表达式的内部形式，假设其中的任何 Vars 都引用第二个参数指定的关系。

  返回类型：text

- pg_get_expr(pg_node_tree, relation_oid, pretty_bool)

  描述：反编译表达式的内部形式，假设其中的任何 Vars 都引用第二个参数指定的关系。

  返回类型：text

  备注：pg_get_expr 反编译一个独立表达式的内部形式，比如一个字段的缺省值。在检查系统表的内容的时候很有用。如果表达式可能包含关键字，则指定他们引用相关的 OID 作为第二个参数；如果没有关键字，零就足够了。

- pg_get_functiondef(func_oid)

  描述：获取函数的定义。

  返回类型：text

  示例：

```
panweidb=# select * from pg_get_functiondef(598);
 headerlines |                  definition
-------------+-------------------------------------------------
           4 | CREATE OR REPLACE FUNCTION pg_catalog.abbrev(inet)+
             | RETURNS text                                     +
             | LANGUAGE internal                                +
             | IMMUTABLE STRICT NOT FENCED NOT SHIPPABLE        +
             | AS $function$inet_abbrev$function$               +
             |
(1 row)
```

- pg_get_function_arguments(func_oid)

  描述：获取函数定义的参数列表（带默认值）。

返回类型：text

备注：pg_get_function_arguments 返回一个函数的参数列表，需要在 CREATE FUNCTION 中使用这种格式。

- pg_get_function_identity_arguments(func_oid)

  描述：获取参数列表来确定一个函数（不带默认值）。

  返回类型：text

  备注：pg_get_function_identity_arguments 返回需要的参数列表用来标识函数，这种形式需要在 ALTER FUNCTION 中使用，并且这种形式省略了默认值。

- pg_get_function_result(func_oid)

  描述：获取函数的 RETURNS 子句。

  返回类型：text

  备注：pg_get_function_result 为函数返回适当的 RETURNS 子句。

- pg_get_indexdef(index_oid)

  描述：获取索引的 CREATE INDEX 命令。

  返回类型：text

  示例：

```
panweidb=# select * from pg_get_indexdef(16416);
                    pg_get_indexdef
------------------------------------------------------------------------
 CREATE INDEX test3_b_idx ON test3 USING btree (b) TABLESPACE pg_default
(1 row)
```

- pg_get_indexdef(index_oid, dump_schema_only)

  描述：获取索引的 CREATE INDEX 命令，仅用于 dump 场景。对于包含 local 索引的间隔分区表，当 dump_schema_only 为 true 时，返回的创建索引语句中不包含自动创建的分区的 local 索引信息；当 dump_schema_only 为 false 时，返回的创建索引语句中包含自动创建的分区的 local 索引信息。对于非间隔分区表或者不包含 local 索引的间隔分区分区表，dump_schema_only 参数取值不影响函数返回结果。

  返回类型：text

  示例：

```
panweidb=# CREATE TABLE sales
 (prod_id NUMBER(6),
  cust_id NUMBER,
  time_id DATE,
  channel_id CHAR(1),
  promo_id NUMBER(6),
  quantity_sold NUMBER(3),
  amount_sold NUMBER(10,2)
  )
  PARTITION BY RANGE( time_id) INTERVAL('1 day')
   (
   partition p1 VALUES LESS THAN ('2019-02-01 00:00:00'),
   partition p2 VALUES LESS THAN ('2019-02-02 00:00:00')
   );

panweidb=# create index index_sales on sales(prod_id) local (PARTITION idx_p1 ,P
ARTITION idx_p2);

-- 插入数据没有匹配的分区，新创建一个分区，并将数据插入该分区
panweidb=# INSERT INTO sales VALUES(1, 12, '2019-02-05 00:00:00', 'a', 1, 1, 1);
panweidb=# select oid from pg_class where relname = 'index_sales';
 oid
-------
 16645
(1 row)

panweidb=# select * from pg_get_indexdef(16645, true);
                                  pg_get_indexdef
-----------------------------------------------------------------------------------
--------------------------------
 CREATE INDEX index_sales ON sales USING btree (prod_id) LOCAL(PARTITION idx_p
1, PARTITION idx_p2)  TABLESPACE pg_default
(1 row)

panweidb=# select * from pg_get_indexdef(16645, false);
```

```
                                pg_get_indexdef

-------------------------------------------------------------------------------------
-----------------------------------------------
---------------
 CREATE INDEX index_sales ON sales USING btree (prod_id) LOCAL(PARTITION idx_p
1, PARTITION idx_p2, PARTITION sys_p1_prod_id_idx)  TABLESP
ACE pg_default
(1 row)
```

- pg_get_indexdef(index_oid, column_no, pretty_bool)

  描述：获取索引的 CREATE INDEX 命令，或者如果 column_no 不为零，则只获取一个索引字段的定义。

  示例：

```
panweidb=# select * from pg_get_indexdef(16645, 0, false);
                        pg_get_indexdef
-------------------------------------------------------------------------------
 CREATE INDEX index_sales ON sales USING btree (prod_id) LOCAL TABLESPACE pg_
default
(1 row)

panweidb=# select * from pg_get_indexdef(16645, 1, false);

 pg_get_indexdef
-----------------
 prod_id
(1 row)
```

  返回类型：text

  备注：pg_get_functiondef 为函数返回一个完整的 CREATE OR REPLACE FUNCTION 语句。

- pg_get_keywords()

  描述：获取 SQL 关键字和类别列表。

  返回类型：setof record

备注：pg_get_keywords 返回一组关于描述服务器识别 SQL 关键字的记录。word 列包含关键字。catcode 列包含一个分类代码：U 表示通用的，C 表示列名，T 表示类型或函数名，或 R 表示保留。catdesc 列包含了一个可能本地化描述分类的字符串。

- pg_get_userbyid(role_oid)

  描述：获取给定 OID 的角色名。

  返回类型：name

  备注：pg_get_userbyid 通过角色的 OID 抽取对应的用户名。

- pg_check_authid(role_id)

  描述：通过 role_id 检查用户是否存在。

  返回类型：text

  示例：

```
panweidb=# select pg_check_authid(20);
pg_check_authid
-----------------
f
(1 row)
```

- pg_get_viewdef(view_name)

  描述：为视图获取底层的 SELECT 命令。

  返回类型：text

- pg_get_viewdef(view_name, pretty_bool)

  描述：为视图获取底层的 SELECT 命令，如果 pretty_bool 为 true，行字段可以包含 80 列。

  返回类型：text

  备注：pg_get_viewdef 重构出定义视图的 SELECT 查询。这些函数大多数都有两种形式，其中带有 pretty_bool 参数，且参数为 true 时，是"适合打印"的结果，这种格式更容易读。另一种是缺省的格式，更有可能被将来的不同版本用同样的方法解释。如果是用于转储，那么尽可能避免使用适合打印的格式。给 pretty-print 参数传递 false 生成的结果和没有这个参数的变种生成的结果是完全一样。

- pg_get_viewdef(view_oid)

描述：为视图获取底层的 SELECT 命令。

返回类型：text

- pg_get_viewdef(view_oid, pretty_bool)

  描述：为视图获取底层的 SELECT 命令，如果 pretty_bool 为 true，行字段可以包含 80 列。

  返回类型：text

- pg_get_viewdef(view_oid, wrap_column_int)

  描述：为视图获取底层的 SELECT 命令；行字段被换到指定的列数，打印是隐含的。

  返回类型：text

- pg_get_tabledef(table_oid)

  描述：根据 table_oid 获取表定义

  示例：

```
panweidb=# select * from pg_get_tabledef(16384);
             pg_get_tabledef
-----------------------------------------------------
 SET search_path = public;                    +
 CREATE  TABLE t1 (                          +
      c1 bigint DEFAULT nextval('serial'::regclass)+
 )                                           +
 WITH (orientation=row, compression=no)           +
 TO GROUP group1;
(1 row)
```

  返回类型：text

- pg_get_tabledef(table_name)

  描述：根据 table_name 获取表定义。

  示例：

```
panweidb=# select * from pg_get_tabledef('t1');
             pg_get_tabledef
-----------------------------------------------------
 SET search_path = public;                    +
 CREATE TABLE t1 (                           +
   c1 integer,                               +
```

```
  c2 integer                        +
)                                   +
WITH (orientation=row, compression=no, fillfactor=80);
(1 row)
```

返回类型：text

备注：pg_get_tabledef 重构出表定义的 CREATE 语句，包含了表定义本身、索引信息、comments 信息。对于表对象依赖的 group、schema、tablespace、server 等信息，需要用户自己去创建，表定义里不会有这些对象的创建语句。

- pg_options_to_table(reloptions)

  描述：获取存储选项名称/值对的集合。

  返回类型：setof record

  备注：pg_options_to_table 当通过 pg_class.reloptions 或 pg_attribute.attoptions 时返回存储选项名称/值对（option_name/option_value）的集合。

- pg_tablespace_databases(tablespace_oid)

  描述：获取在指定的表空间中有对象的数据库 OID 集合。

  返回类型：setof oid

  备注：pg_tablespace_databases 允许检查表空间的状况，返回在该表空间中保存了对象的数据库 OID 集合。如果这个函数返回数据行，则该表空间就是非空的，因此不能删除。要显示该表空间中的特定对象，用户需要连接 pg_tablespace_databases 标识的数据库与查询 pg_class 系统表。

- pg_tablespace_location(tablespace_oid)

  描述：获取表空间所在的文件系统的路径。

  返回类型：text

- pg_typeof(any)

  描述：获取任何值的数据类型。

  返回类型：regtype

  备注：pg_typeof 返回传递给他的值的数据类型 OID。这可能有助于故障排除或动态构造 SQL 查询。声明此函数返回 regtype，这是一个 OID 别

名类型（请参考开发者指南->SQL 语法参考->SQL 基本要素->数据类型->对象标识符类型章节）；这意味着它是一个为了比较而显示类型名称的 OID。

示例：

```
panweidb=# SELECT pg_typeof(33);
 pg_typeof
-----------
 integer
(1 row)

panweidb=# SELECT typlen FROM pg_type WHERE oid = pg_typeof(33);
 typlen
--------
      4
(1 row)
```

- collation for (any)

  描述：获取参数的排序。

  返回类型：text

  备注：表达式 collation for 返回传递给他的值的排序。

  示例：

```
panweidb=# SELECT collation for (description) FROM pg_description LIMIT 1;
 pg_collation_for
------------------
 "default"
(1 row)
```

  值可能是引号括起来的并且模式限制的。如果没有为参数表达式排序，则返回一个 null 值。如果参数不是排序的类型，则抛出一个错误。

- pg_extension_update_paths(name)

  描述：返回指定扩展的版本更新路径。

  返回类型：text(source text), text(path text), text(target text)

- pg_get_serial_sequence(tablename, colname)

  描述：获取对应表名和列名上的序列。

  返回类型：text

示例：

```
panweidb=# select * from pg_get_serial_sequence('t2', 'c1');
 pg_get_serial_sequence
------------------------
 public.serial
(1 row)
```

❖ pg_sequence_parameters(sequence_oid)

描述：获取指定 sequence 的参数，包含起始值，最小值和最大值，递增值等。

返回类型：int16, int16,int16, bigint, Boolean

示例：

```
panweidb=# select * from pg_sequence_parameters(16643);
 start_value | minimum_value |    maximum_value    | increment | cycle_option
-------------+---------------+----------------------+-----------+--------------
         101 |             1 | 9223372036854775807 |         1 | f
(1 row)
```

4.1.4.4.5.查询缓存函数

新增内置函数

pw_result_cache_status

功能描述： 查看查询缓存的状态。

入参： 无

输出：

❖ ACTIVE： 查询缓存特性是激活的。

❖ DISABLED：查询缓存特性是不可用的。

示例

1、设置 GUC 参数。

```
alter system set enable_global_result_cache to on;
alter system set result_cache_max_size to 409600000;
alter system set result_cache_max_result to 10;
```

2、重启数据库。

```
pw_ctl restart;
```

3、查看查询缓存的状态。

```
select pw_result_cache_status();
```

结果显示为：

```
 pw_result_cache_status
------------------------
 active
(1 row)
```

4、手动修改配置文件将步骤 1 中开启的查询缓存相关三个参数注释掉并重启数据库。

```
vi postgresql.conf
```

5、查看查询缓存的状态。

```
select pw_result_cache_status();
```

结果显示为如下：

```
 pw_result_cache_status
------------------------
 invalid
(1 row)
```

pw_result_cache_memory_report

功能描述： 列出结果缓存内存利用的一个概要（默认）或详细的报表。

入参: 无

输出:

- TotalMemory
  查询缓存总共可用内存（字节）。
- UsedMemory
  查询缓存当前已用内存（字节）。
- CacheCo
  当前缓存个数。
- MaxCacheSize
  当前缓存中使用的最大内存（字节）。

示例

参见管理员指南->查询缓存的示例 1。

pw_result_cache_flush

功能描述：清理整个查询缓存，清理期间，查询缓存不可用。

入参： 无

输出： 清理结果

示例：

参见管理员指南->查询缓存的示例 3。

pw_result_cache_invalidate

功能描述： 使指定缓存失效。

入参： cache_id

输出： 清理结果。

示例：

1、设置 GUC 参数。

```
alter system set enable_global_result_cache to on;
alter system set result_cache_max_size to 409600000;
alter system set result_cache_max_result to 10;
```

2、重启数据库。

```
pw_ctl restart;
```

3、创建测试表并插入数据。

```
CREATE TABLE salaries (
emp_no int NOT NULL,
salary int NOT NULL,
from_date date NOT NULL,to_date date NOT NULL,
PRIMARY KEY (emp_no,from_date));
INSERT INTO salaries VALUES(10001,60117,'1986-06-26','1987-06-26');
INSERT INTO salaries VALUES(10001,62102,'1987-06-26','1988-06-25');
INSERT INTO salaries VALUES(10001,66074,'1988-06-25','1989-06-25')
;INSERT INTO salaries VALUES(10001,66596,'1989-06-25','1990-06-25');
INSERT INTO salaries VALUES(10001,66961,'1990-06-25','1991-06-25');
INSERT INTO salaries VALUES(10001,71046,'1991-06-25','1992-06-24');
INSERT INTO salaries VALUES(10001,74333,'1992-06-24','1993-06-24');
INSERT INTO salaries VALUES(10001,75286,'1993-06-24','1994-06-24');
INSERT INTO salaries VALUES(10001,75994,'1994-06-24','1995-06-24');
```

```
INSERT INTO salaries VALUES(10001,76884,'1995-06-24','1996-06-23');
INSERT INTO salaries VALUES(10001,80013,'1996-06-23','1997-06-23');
INSERT INTO salaries VALUES(10001,81025,'1997-06-23','1998-06-23');
INSERT INTO salaries VALUES(10001,81097,'1998-06-23','1999-06-23');
INSERT INTO salaries VALUES(10001,84917,'1999-06-23','2000-06-22');
INSERT INTO salaries VALUES(10001,85112,'2000-06-22','2001-06-22');
INSERT INTO salaries VALUES(10001,85097,'2001-06-22','2002-06-22');
INSERT INTO salaries VALUES(10001,88958,'2002-06-22','9999-01-01');
INSERT INTO salaries VALUES(10002,72527,'1996-08-03','1997-08-03');
set result_cache_mode=force;
SELECT IF(profile LIKE '%female','female','male') gender,COUNT(*) numberFROM us
er_submitGROUP BY gender;
```

4、查看缓存状态。

```
select * from pg_catalog.pw_result_cache_items();
select * from pw_result_cache_memory_report;
```

5、使用函数清除缓存。

```
select pw_result_cache_flush();
select * from pg_catalog.pw_result_cache_items();--返回 0 条数据
select * from pw_result_cache_memory_report;
```

pw_result_cache_items

功能描述： 查询已经生效的缓存的相关信息。

入参： 无

输出：

- cache_id

  缓存 id。

- query_string

  语句字符串。

- ref_count

  对于缓存的引用计数。

示例：

参见 pw_result_cache_flush 的示例。

pw_result_cache_items_debug

功能描述： 查询已经生效的缓存的相关信息，此函数不建议用户使用，其本质属于额外辅助信息的输出，不推荐用户使用此函数的返回内容作为评判依据，建议使用 pw_result_cache_items。

入参： 无

输出：

- **cache_id**
  缓存 id。
- **query_string**
  语句字符串。
- **ref_count**
  对于缓存的引用计数。
- **Querycache_id**
  查询树的 hash 值。
- **Bucket_no**
  缓存对应的 hash 桶下标。

pw_result_cache_items_debug2

功能描述： 查询已经生效的缓存的相关信息，此函数不建议用户使用，其本质属于额外辅助信息的输出，不推荐用户使用此函数的返回内容作为评判依据，建议使用 pw_result_cache_items。

入参： 无

输出：

- cache_id
  缓存 id。
- ref_count
  对于缓存的引用计数。

4.1.4.4.6. 注释信息函数

- col_description(table_oid, column_number)
  描述：获取一个表字段的注释

返回类型：text

备注：col_description 返回一个表中字段的注释，通过表 OID 和字段号来声明。

- obj_description(object_oid, catalog_name)

  描述：获取一个数据库对象的注释

  返回类型：text

  备注：带有两个参数的 obj_description 返回一个数据库对象的注释，该对象是通过其 OID 和其所属的系统表名称声明。比如，obj_description(123456,'pg_class')将返回 OID 为 123456 的表的注释。只带一个参数的 obj_description 只要求对象 OID。obj_description 不能用于表字段，因为字段没有自己的 OID。

- obj_description(object_oid)

  描述：获取一个数据库对象的注释

  返回类型：text

- shobj_description(object_oid, catalog_name)

  描述：获取一个共享数据库对象的注释

  返回类型：text

  备注：shobj_description 和 obj_description 差不多，不同之处仅在于前者用于共享对象。一些系统表是通用于 panweidb 中所有数据库的全局表，因此这些表的注释也是全局存储的。

事务 ID 和快照

内部事务 ID 类型（xid）是 64 位。这些函数使用的数据类型 txid_snapshot，存储在特定时刻事务 ID 可见性的信息。其组件描述在表 11。

表 12 快照组件

名称	描述
xmin	最早的事务 ID（txid）仍然活动。所有较早事务将是已经提交可见的，或者是直接回滚。
xmax	作为尚未分配的 txid。所有大于或等于此 txids 的都是尚未开始的快照时

名称	描述
	间，因此不可见。
xip_list	当前快照中活动的 txids。这个列表只包含在 xmin 和 xmax 之间活动的 txids；有可能活动的 txids 高于 xmax。介于大于等于 xmin、小于 xmax，并且不在这个列表中的 txid，在这个时间快照已经完成的，因此按照提交状态查看他是可见还是回滚。这个列表不包含子事务的 txids。

txid_snapshot 的文本表示为：xmin:xmax:xip_list。

示例：10:20:10,14,15 意思为：xmin=10, xmax=20, xip_list=10, 14, 15。

以下的函数在一个输出形式中提供服务器事务信息。这些函数的主要用途是为了确定在两个快照之间有哪个事务提交。

- txid_current()
  描述：获取当前事务 ID。
  返回类型：bigint
- gs_txid_oldestxmin()
  描述：获取当前最小事务 id 的值 oldesxmin。
  返回类型：bigint
- txid_current_snapshot()
  描述：获取当前快照。
  返回类型：txid_snapshot
- txid_snapshot_xip(txid_snapshot)
  描述：在快照中获取正在进行的事务 ID。
  返回类型：setof bigint
- txid_snapshot_xmax(txid_snapshot)
  描述：获取快照的 xmax。
  返回类型：bigint
- txid_snapshot_xmin(txid_snapshot)
  描述：获取快照的 xmin。
  返回类型：bigint
- txid_visible_in_snapshot(bigint, txid_snapshot)

描述：在快照中事务 ID 是否可见（不使用子事务 ID）。

返回类型：Boolean

- get_local_prepared_xact()

  描述：获取当前节点两阶段残留事务信息，包括事务 id，两阶段 gid 名称，prepared 的时间，owner 的 oid，database 的 oid 及当前节点的 node_name。

  返回类型：xid, text, timestamptz, oid, oid，text

- get_remote_prepared_xacts()

  描述：获取所有远程节点两阶段残留事务信息，包括事务 id，两阶段 gid 名称，prepared 的时间，owner 的名称，database 的名称及 node_name。

  返回类型：xid, text, timestamptz, name, name，text

- global_clean_prepared_xacts(text, text)

  描述：并发清理两阶段残留事务，仅分布式场景下 gs_clean 工具可以调用清理，其他用户调用均返回 false。

  返回类型：Boolean

- gs_get_next_xid_csn()

  描述：返回全局所有节点上的 next_xid 和 next_csn 值。

  返回值如下：

  表 13 gs_get_next_xid_csn 返回参数说明

字段名	描述
nodename	节点名称。
next_xid	当前节点下一个事务 id 号。
next_csn	当前节点下一个 csn 号。

- pg_control_system()

  描述：返回系统控制文件状态。

  返回类型：SETOF record

- pg_control_checkpoint()

  描述：返回系统检查点状态。

返回类型：SETOF record

- pv_builtin_functions

  描述：查看所有内置系统函数信息。

  参数：nan

  返回值类型：proname name, pronamespace oid, proowner oid, prolang oid, procost real, prorows real, provariadic oid, protransform regproc, proisagg boolean, proiswindow boolean, prosecdef boolean, proleakproof boolean, proisstrict boolean, proretset boolean, provolatile "char", pronargs smallint, pronargdefaults smallint, prorettype oid, proargtypes oidvector, proallargtypes integer[], proargmodes "char"[], proargnames text[], proargdefaults pg_node_tree, prosrc text, probin text, proconfig text[], proacl aclitem[], prodefaultargpos int2vector, fencedmode boolean, proshippable boolean, propackage boolean, oid oid

- pv_thread_memory_detail

  描述：返回各线程的内存信息。

  参数：nan

  返回值类型：threadid text, tid bigint, thrdtype text, contextname text, level smallint, parent text, totalsize bigint, freesize bigint, usedsize bigint

- pg_relation_compression_ratio

  描述：查询表压缩率，默认返回 1.0。

  参数：text

  返回值类型：real

- pg_relation_with_compression

  描述：查询表是否压缩。

  参数：text

  返回值类型：boolean

- pg_stat_file_recursive

  描述：列出路径下所有文件。

  参数：location text

- pg_shared_memory_detail

  描述：返回所有已产生的共享内存上下文的使用信息，各列描述请参考开发者指南->schema->DBE_PERF Schema->Memory->GS_SHARED_MEMORY_DETAIL 章节。

  参数：nan

  返回值类型：contextname text, level smallint, parent text, totalsize bigint, freesize bigint, usedsize bigint

- get_gtm_lite_status

  描述：返回 GTM 上的 backupXid 和 csn 号，用来支持问题定位，GTM-FREE 模式下不支持使用本系统函数。

- gs_stat_get_wlm_plan_operator_info

  描述：从内部哈希表中获取算子计划信息。

  参数：oid

  返回值类型：datname text, queryid int8, plan_node_id int4, startup_time int8, total_time int8, actual_rows int8, max_peak_memory int4, query_dop int4, parent_node_id int4, left_child_id int4, right_child_id int4, operation text, orientation text, strategy text, options text, condition text, projection text

- pg_stat_get_partition_tuples_hot_updated

  描述：返回给定分区 id 的分区热更新元组数的统计。

  参数：oid

  返回值类型：bigint

- gs_session_memory_detail_tp

  描述：返回会话的内存使用情况，参考 gs_session_memory_detail。

  参数：nan

  返回值类型：sessid text, sesstype text, contextname text, level smallint, parent text, totalsize bigint, freesize bigint, usedsize bigint

- gs_thread_memory_detail

  描述：返回各线程的内存信息。

  参数：nan

返回值类型：threadid text, tid bigint, thrdtype text, contextname text, level smallint, parent text, totalsize bigint, freesize bigint, usedsize bigint

- pg_stat_get_wlm_realtime_operator_info

  描述：从内部哈希表中获取实时执行计划算子信息。

  参数：nan

  返回值类型：queryid bigint, pid bigint, plan_node_id integer, plan_node_name text, start_time timestamp with time zone, duration bigint, status text, query_dop integer, estimated_rows bigint, tuple_processed bigint, min_peak_memory integer, max_peak_memory integer, average_peak_memory integer, memory_skew_percent integer, min_spill_size integer, max_spill_size integer, average_spill_size integer, spill_skew_percent integer, min_cpu_time bigint, max_cpu_time bigint, total_cpu_time bigint, cpu_skew_percent integer, warning text

- pg_stat_get_wlm_realtime_ec_operator_info

  描述：从内部哈希表中获取 EC 执行计划算子信息。

  参数：nan

  返回值类型：queryid bigint, plan_node_id integer, plan_node_name text, start_time timestamp with time zone, ec_operator integer, ec_status text, ec_execute_datanode text, ec_dsn text, ec_username text, ec_query text, ec_libodbc_type text, ec_fetch_count bigint

- pg_stat_get_wlm_operator_info

  描述：从内部哈希表中获取执行计划算子信息。

  参数：nan

  返回值类型：queryid bigint, pid bigint, plan_node_id integer, plan_node_name text, start_time timestamp with time zone, duration bigint, query_dop integer, estimated_rows bigint, tuple_processed bigint, min_peak_memory integer, max_peak_memory integer, average_peak_memory integer,

memory_skew_percent integer, min_spill_size integer, max_spill_size integer, average_spill_size integer, spill_skew_percent integer, min_cpu_time bigint, max_cpu_time bigint, total_cpu_time bigint, cpu_skew_percent integer, warning text

- pg_stat_get_wlm_node_resource_info

  描述：获取当前节点资源信息。

  参数：nan

  返回值类型：min_mem_util integer, max_mem_util integer, min_cpu_util integer, max_cpu_util integer, min_io_util integer, max_io_util integer, used_mem_rate integer

- pg_stat_get_session_wlmstat

  描述：返回当前会话负载信息。

  参数：pid integer

  返回值类型：datid oid, threadid bigint, sessionid bigint, threadpid integer, usesysid oid, appname text, query text, priority bigint, block_time bigint, elapsed_time bigint, total_cpu_time bigint, skew_percent integer, statement_mem integer, active_points integer, dop_value integer, current_cgroup text, current_status text, enqueue_state text, attribute text, is_plana boolean, node_group text, srespool name

- pg_stat_get_wlm_ec_operator_info

  描述：从内部哈希表中获取 EC 执行计划算子信息。

  参数：nan

  返回值类型：queryid bigint, plan_node_id integer, plan_node_name text, start_time timestamp with time zone, duration bigint, tuple_processed bigint, min_peak_memory integer, max_peak_memory integer, average_peak_memory integer, ec_operator integer, ec_status text, ec_execute_datanode text, ec_dsn text, ec_username text, ec_query text, ec_libodbc_type text, ec_fetch_count bigint

- pg_stat_get_wlm_instance_info

  描述：返回当前实例负载信息。

  参数：nan

  返回值类型：instancename text, timestamp timestamp with time zone, used_cpu integer, free_memory integer, used_memory integer, io_await double precision, io_util double precision, disk_read double precision, disk_write double precision, process_read bigint, process_write bigint, logical_read bigint, logical_write bigint, read_counts bigint, write_counts bigint

- pg_stat_get_wlm_instance_info_with_cleanup

  描述：返回当前实例负载信息，并且保存到系统表中。

  参数：nan

  返回值类型：instancename text, timestamp timestamp with time zone, used_cpu integer, free_memory integer, used_memory integer, io_await double precision, io_util double precision, disk_read double precision, disk_write double precision, process_read bigint, process_write bigint, logical_read bigint, logical_write bigint, read_counts bigint, write_counts bigint

- pg_stat_get_wlm_realtime_session_info

  描述：返回实时会话负载信息。

  参数：nan

  返回值类型：nodename text, threadid bigint, block_time bigint, duration bigint, estimate_total_time bigint, estimate_left_time bigint, schemaname text, query_band text, spill_info text, control_group text, estimate_memory integer, min_peak_memory integer, max_peak_memory integer, average_peak_memory integer, memory_skew_percent integer, min_spill_size integer, max_spill_size integer, average_spill_size integer, spill_skew_percent integer, min_dn_time bigint, max_dn_time bigint, average_dn_time bigint, dntime_skew_percent integer, min_cpu_time bigint, max_cpu_time bigint, total_cpu_time bigint,

cpu_skew_percent integer, min_peak_iops integer, max_peak_iops integer, average_peak_iops integer, iops_skew_percent integer, warning text, query text, query_plan text, cpu_top1_node_name text, cpu_top2_node_name text, cpu_top3_node_name text, cpu_top4_node_name text, cpu_top5_node_name text, mem_top1_node_name text, mem_top2_node_name text, mem_top3_node_name text, mem_top4_node_name text, mem_top5_node_name text, cpu_top1_value bigint, cpu_top2_value bigint, cpu_top3_value bigint, cpu_top4_value bigint, cpu_top5_value bigint, mem_top1_value bigint, mem_top2_value bigint, mem_top3_value bigint, mem_top4_value bigint, mem_top5_value bigint, top_mem_dn text, top_cpu_dn text

- pg_stat_get_wlm_session_iostat_info

  描述：返回会话负载 IO 信息。

  参数：nan

  返回值类型：threadid bigint, maxcurr_iops integer, mincurr_iops integer, maxpeak_iops integer, minpeak_iops integer, iops_limits integer, io_priority integer, curr_io_limits integer

- pg_stat_get_wlm_statistics

  描述：返回会话负载统计数据。

  参数：nan

  返回值类型：statement text, block_time bigint, elapsed_time bigint, total_cpu_time bigint, qualification_time bigint, skew_percent integer, control_group text, status text, action text

4.1.4.5.文本检索函数和操作符

文本检索操作符

- @@

描述：tsvector 类型的词汇与 tsquery 类型的词汇是否匹配

示例：

```
SELECT to_tsvector('fat cats ate rats') @@ to_tsquery('cat & rat') AS RESULT;
 result
--------
 t
(1 row)
```

❖ @@@

描述：@@的同义词

示例：

```
SELECT to_tsvector('fat cats ate rats') @@@ to_tsquery('cat & rat') AS RESULT;
 result
--------
 t
(1 row)
```

❖ ||

描述：连接两个 tsvector 类型的词汇

示例：

```
SELECT 'a:1 b:2'::tsvector || 'c:1 d:2 b:3'::tsvector AS RESULT;
       result
---------------------------
 'a':1 'b':2,5 'c':3 'd':4
(1 row)
```

❖ &&

描述：将两个 tsquery 类型的词汇进行“与”操作

示例：

```
SELECT 'fat | rat'::tsquery && 'cat'::tsquery AS RESULT;
       result
--------------------------
 ( 'fat' | 'rat' ) & 'cat'
(1 row)
```

❖ ||

描述：将两个 tsquery 类型的词汇进行“或”操作

示例：

```
SELECT 'fat | rat'::tsquery || 'cat'::tsquery AS RESULT;
       result
--------------------------
( 'fat' | 'rat' ) | 'cat'
(1 row)
```

❖ !!

描述：tsquery 类型词汇的非关系

示例：

```
SELECT !! 'cat'::tsquery AS RESULT;
result
--------
!'cat'
(1 row)
```

❖ @>

描述：一个 tsquery 类型的词汇是否包含另一个 tsquery 类型的词汇

示例：

```
SELECT 'cat'::tsquery @> 'cat & rat'::tsquery AS RESULT;
result
--------
f
(1 row)
```

❖ <@

描述：一个 tsquery 类型的词汇是否被包含另一个 tsquery 类型的词汇

示例：

```
SELECT 'cat'::tsquery <@ 'cat & rat'::tsquery AS RESULT;
result
--------
```

```
t
(1 row)
```

除了上述的操作符，还为 tsvector 类型和 tsquery 类型的数据定义了普通的 B-tree 比较操作符（=、<等）。

文本检索函数

❖ get_current_ts_config()

描述：获取文本检索的默认配置。

返回类型：regconfig

示例：

```
SELECT get_current_ts_config();
get_current_ts_config
----------------------
english
(1 row)
```

❖ length(tsvector)

描述：tsvector 类型词汇的单词数。

返回类型：integer

示例：

```
SELECT length('fat:2,4 cat:3 rat:5A'::tsvector);
length
--------
   3
(1 row)
```

❖ numnode(tsquery)

描述：tsquery 类型的单词加上操作符的数量。

返回类型：integer

示例：

```
SELECT numnode('(fat & rat) | cat'::tsquery);
 numnode
---------
       5
(1 row)
```

❖ plainto_tsquery([config regconfig ,] query text)

描述：产生 tsquery 类型的词汇，并忽略标点。

返回类型：tsquery

示例：

```
SELECT plainto_tsquery('english', 'The Fat Rats');
 plainto_tsquery
-----------------
 'fat' & 'rat'
(1 row)
```

❖ querytree(query tsquery)

描述：获取 tsquery 类型的词汇可加索引的部分。

返回类型：text

示例：

```
SELECT querytree('foo & ! bar'::tsquery);
 querytree
-----------
 'foo'
(1 row)
```

❖ setweight(tsvector, "char")

描述：给 tsvector 类型的每个元素分配权值。

返回类型：tsvector

示例：

```
SELECT setweight('fat:2,4 cat:3 rat:5B'::tsvector, 'A');
          setweight
```

```
------------------------------
 'cat':3A 'fat':2A,4A 'rat':5A
(1 row)
```

❖ strip(tsvector)

描述：删除 tsvector 类型单词中的 position 和权值。

返回类型：tsvector

示例：

```
SELECT strip('fat:2,4 cat:3 rat:5A'::tsvector);
    strip
-------------------
 'cat' 'fat' 'rat'
(1 row)
```

❖ to_tsquery([config regconfig ,] query text)

描述：标准化单词，并转换为 tsquery 类型。

返回类型：tsquery

示例：

```
SELECT to_tsquery('english', 'The & Fat & Rats');
 to_tsquery
---------------
 'fat' & 'rat'
(1 row)
```

❖ to_tsvector([config regconfig ,] document text)

描述：去除文件信息，并转换为 tsvector 类型。

返回类型：tsvector

示例：

```
SELECT to_tsvector('english', 'The Fat Rats');
  to_tsvector
-----------------
 'fat':2 'rat':3
```

```
(1 row)
```

❖ to_tsvector_for_batch([config regconfig ,] document text)

描述：去除文件信息，并转换为 tsvector 类型。

返回类型：tsvector

示例：

```
SELECT to_tsvector_for_batch('english', 'The Fat Rats');
  to_tsvector
-----------------
 'fat':2 'rat':3
(1 row)
```

❖ ts_headline([config regconfig,] document text, query tsquery [, options text])

描述：高亮显示查询的匹配项。

返回类型：text

示例：

```
SELECT ts_headline('x y z', 'z'::tsquery);
ts_headline
--------------
 x y <b>z</b>
(1 row)
```

❖ ts_ rank([weights float4[],] vector tsvector, query tsquery [, normalization integer])

描述：文档查询排名。

返回类型：float4

示例：

```
SELECT ts_rank ('hello world'::tsvector, 'world'::tsquery);
ts_rank
----------
```

```
.0607927
(1 row)
```

- ts_ rank_cd([weights float4[],] vector tsvector, query tsquery [, normalization integer])

描述：排序文件查询使用覆盖密度。

返回类型：float4

示例：

```
SELECT ts_rank_cd('hello world'::tsvector, 'world'::tsquery);
 ts_rank_cd
------------
          0
(1 row)
```

- ts_rewrite(query tsquery, target tsquery, substitute tsquery)

描述：替换目标 tsquery 类型的单词。

返回类型：tsquery

示例：

```
SELECT ts_rewrite('a & b'::tsquery, 'a'::tsquery, 'foo|bar'::tsquery);
     ts_rewrite
------------------------
 'b' & ( 'foo' | 'bar' )
(1 row)
```

- ts_rewrite(query tsquery, select text)

描述：使用 SELECT 命令的结果替代目标中 tsquery 类型的单词。

返回类型：tsquery

示例：

```
SELECT ts_rewrite('world'::tsquery, 'select ''world''::tsquery, ''hello''::tsquery');
 ts_rewrite
------------
```

```
'hello'
(1 row)
```

文本检索调试函数

- ts_debug([config regconfig,] document text, OUT alias text, OUT description text, OUT token text, OUT dictionaries regdictionary[], OUT dictionary regdictionary, OUT lexemes text[])

描述：测试一个配置。

返回类型：setof record

示例：

```
SELECT ts_debug('english', 'The Brightest supernovaes');
                          ts_debug
-------------------------------------------------------------------------------
(asciiword,"Word, all ASCII",The,{english_stem},english_stem,{})
(blank,"Space symbols"," ",{},,)
(asciiword,"Word, all ASCII",Brightest,{english_stem},english_stem,{brightest})
(blank,"Space symbols"," ",{},,)
(asciiword,"Word, all ASCII",supernovaes,{english_stem},english_stem,{superno
    va})
(5 rows)
```

- ts_lexize(dict regdictionary, token text)

描述：测试一个数据字典。

返回类型：text[]

示例：

```
SELECT ts_lexize('english_stem', 'stars');
 ts_lexize
-----------
 {star}
(1 row)
```

- ts_parse(parser_name text, document text, OUT tokid integer, OUT token text)

描述：测试一个解析。

返回类型：setof record

示例：

```
SELECT ts_parse('default', 'foo - bar');
 ts_parse
-----------
 (1,foo)
 (12," ")
 (12,"- ")
 (1,bar)
(4 rows)
```

- ts_parse(parser_oid oid, document text, OUT tokid integer, OUT token text)

描述：测试一个解析。

返回类型：setof record

示例：

```
SELECT ts_parse(3722, 'foo - bar');
 ts_parse
-----------
 (1,foo)
 (12," ")
 (12,"- ")
 (1,bar)
(4 rows)
```

- ts_token_type(parser_name text, OUT tokid integer, OUT alias text, OUT description text)

描述：获取分析器定义的记号类型。

返回类型：setof record

示例：

```
SELECT ts_token_type('default');
                ts_token_type
---------------------------------------------------------------
(1,asciiword,"Word, all ASCII")
(2,word,"Word, all letters")
(3,numword,"Word, letters and digits")
(4,email,"Email address")
(5,url,URL)
(6,host,Host)
(7,sfloat,"Scientific notation")
(8,version,"Version number")
(9,hword_numpart,"Hyphenated word part, letters and digits")
(10,hword_part,"Hyphenated word part, all letters")
(11,hword_asciipart,"Hyphenated word part, all ASCII")
(12,blank,"Space symbols")
(13,tag,"XML tag")
(14,protocol,"Protocol head")
(15,numhword,"Hyphenated word, letters and digits")
(16,asciihword,"Hyphenated word, all ASCII")
(17,hword,"Hyphenated word, all letters")
(18,url_path,"URL path")
(19,file,"File or path name")
(20,float,"Decimal notation")
(21,int,"Signed integer")
(22,uint,"Unsigned integer")
(23,entity,"XML entity")
(23 rows)
```

- ts_token_type(parser_oid oid, OUT tokid integer, OUT alias text, OUT description text)

描述：获取分析器定义的记号类型。

返回类型：setof record

示例：

```
SELECT ts_token_type(3722);
                ts_token_type
---------------------------------------------------------------
(1,asciiword,"Word, all ASCII")
(2,word,"Word, all letters")
(3,numword,"Word, letters and digits")
(4,email,"Email address")
(5,url,URL)
(6,host,Host)
(7,sfloat,"Scientific notation")
(8,version,"Version number")
(9,hword_numpart,"Hyphenated word part, letters and digits")
(10,hword_part,"Hyphenated word part, all letters")
(11,hword_asciipart,"Hyphenated word part, all ASCII")
(12,blank,"Space symbols")
(13,tag,"XML tag")
(14,protocol,"Protocol head")
(15,numhword,"Hyphenated word, letters and digits")
(16,asciihword,"Hyphenated word, all ASCII")
(17,hword,"Hyphenated word, all letters")
(18,url_path,"URL path")
(19,file,"File or path name")
(20,float,"Decimal notation")
(21,int,"Signed integer")
(22,uint,"Unsigned integer")
(23,entity,"XML entity")
(23 rows)
```

❖ ts_stat(sqlquery text, [weights text,] OUT word text, OUT ndoc integer, OUT nentry integer)

描述：获取 tsvector 列的统计数据。

返回类型：setof record

示例：

```
SELECT ts_stat('select ''hello world''::tsvector');
  ts_stat
-------------
 (world,1,1)
 (hello,1,1)
(2 rows)
```

4.1.4.6.类型转换函数

类型转换函数

- cash_words(money)

描述：类型转换函数，将 money 转换成 text。

示例：

```
SELECT cash_words('1.23');
         cash_words
----------------------------------
 One dollar and twenty three cents
(1 row)
```

- cast(x as y)

描述：类型转换函数，将 x 转换成 y 指定的类型。

示例：

```
 SELECT cast('22-oct-1997' as timestamp);
     timestamp
---------------------
 1997-10-22 00:00:00
(1 row)
```

- hextoraw(raw)

描述：将一个十六进制构成的字符串转换为 raw 类型。

返回值类型：raw

示例：

```
 SELECT hextoraw('7D');
 hextoraw
----------
 7D
(1 row)
```

❖ numtoday(numeric)

描述：将数字类型的值转换为指定格式的时间戳。

返回值类型：timestamp

示例：

```
 SELECT numtoday(2);
 numtoday
----------
 2 days
(1 row)
```

❖ pg_systimestamp()

描述：获取系统时间戳。

返回值类型：timestamp with time zone

示例：

```
 SELECT pg_systimestamp();
      pg_systimestamp
-------------------------------
 2022-08-02 11:57:31.266877+08
(1 row)
```

❖ rawtohex(string)

描述：将一个二进制构成的字符串转换为十六进制的字符串。

结果为输入字符的 ACSII 码，以十六进制表示。

返回值类型：varchar

示例：

```
 SELECT rawtohex('1234567');
  rawtohex
----------------
 31323334353637
(1 row)
```

❖ to_bigint(varchar)

描述：将字符类型转换为 bigint 类型。

返回值类型：bigint

示例：

```
 SELECT to_bigint('123364545554455');
  to_bigint
----------------
 123364545554455
(1 row)
```

❖ to_char(datetime/interval [, fmt])

描述：将一个 DATE、TIMESTAMP、TIMESTAMP WITH TIME ZONE 或者 TIMESTAMP WITH LOCAL TIME ZONE 类型的 DATETIME 或者 INTERVAL 值按照 fmt 指定的格式转换为 TEXT 类型。

- 可选参数 fmt 可以为以下几类：日期、时间、星期、季度和世纪。每类都可以有不同的模板，模板之间可以合理组合，常见的模板有：HH、MI、SS、YYYY、MM、DD。
- 模板可以有修饰词，常用的修饰词是 FM，可以用来抑制前导的零或尾随的空白。

返回值类型：TEXT

示例：

```
SELECT to_char(current_timestamp,'HH12:MI:SS');
 to_char
----------
 10:19:26
(1 row)
```

```
 SELECT to_char(current_timestamp,'FMHH12:FMMI:FMSS');
 to_char
----------
 10:19:46
(1 row)
```

❖ to_char(double precision/real, text)

描述：将浮点类型的值转换为指定格式的字符串。

返回值类型：text

示例：

```
 SELECT to_char(125.8::real, '999D99');
 to_char
---------
  125.80
(1 row)
```

❖ to_char(numeric/smallint/integer/bigint/double precision/real[, fmt])

描述：将一个整型或者浮点类型的值转换为指定格式的字符串。

- 可选参数 fmt 可以为以下几类：十进制字符、“分组”符、正负号和货币符号，每类都可以有不同的模板，模板之间可以合理组合，常见的模板有：9、0、,（千分隔符）、.（小数点）。
- 模板可以有类似 FM 的修饰词，但 FM 不抑制由模板 0 指定而输出的 0。
- 要将整型类型的值转换成对应 16 进制值的字符串，使用模板 X 或 x。

返回值类型：varchar

示例：

```
SELECT to_char(1485,'9,999');
 to_char
---------
  1,485
(1 row)
```

```
SELECT to_char( 1148.5,'9,999.999');
 to_char
----------
  1,148.5
(1 row)
 SELECT to_char(148.5,'990999.909');

  to_char
-----------
   0148.5
(1 row)
 SELECT to_char(123,'XXX');
 to_char
---------
   7B
(1 row)
```

❖ to_char(interval, text)

描述：将时间间隔类型的值转换为指定格式的字符串。

返回值类型：text

示例：

```
 SELECT to_char(interval '15h 2m 12s', 'HH24:MI:SS');
 to_char
----------
 15:02:12
(1 row)
```

❖ to_char(int, text)

描述：将整数类型的值转换为指定格式的字符串。

返回值类型：text

示例：

```
 SELECT to_char(125, '999');
 to_char
```

```
---------
 125
(1 row)
```

❖ to_char(numeric, text)

描述：将数字类型的值转换为指定格式的字符串。

返回值类型：text

示例：

```
SELECT to_char(-125.8, '999D99S');
 to_char
---------
 125.8
(1 row)
```

❖ to_char(string)

描述：将 CHAR、VARCHAR、VARCHAR2、CLOB 类型转换为 VARCHAR 类型。

如使用该函数对 CLOB 类型进行转换，且待转换 CLOB 类型的值超出目标类型的范围，则返回错误。

返回值类型：varchar

示例：

```
 SELECT to_char('01110');
 to_char
---------
 01110
(1 row)
```

❖ to_char(timestamp, text)

描述：将时间戳类型的值转换为指定格式的字符串。

返回值类型：text

示例：

```
 SELECT to_char(current_timestamp, 'HH12:MI:SS');
 to_char
----------
 10:55:59
(1 row)
```

❖ to_clob(char/nchar/varchar/varchar2/nvarchar/nvarchar2/text/raw)

描述：将 RAW 类型或者文本字符集类型 CHAR、NCHAR、VARCHAR、VARCHAR2、NVARCHAR、NVARCHAR2、TEXT 转成 CLOB 类型。

返回值类型：clob

示例：

```
 SELECT to_clob('ABCDEF'::RAW(10));
 to_clob
---------
 ABCDEF
(1 row)
 SELECT to_clob('hello111'::CHAR(15));
 to_clob
----------
 hello111
(1 row)
 SELECT to_clob('gauss123'::NCHAR(10));
 to_clob
----------
 gauss123
(1 row)
 SELECT to_clob('gauss234'::VARCHAR(10));
 to_clob
----------
 gauss234
(1 row)
 SELECT to_clob('gauss345'::VARCHAR2(10));
 to_clob
```

```
----------
 gauss345
(1 row)
 SELECT to_clob('gauss456'::NVARCHAR2(10));
 to_clob
----------
 gauss456
(1 row)
 SELECT to_clob('World222!'::TEXT);
 to_clob
-----------
 World222!
(1 row)
```

❖ to_date(text)

描述：将文本类型的值转换为指定格式的时间戳。目前只支持两类格式。

- 格式一：无分隔符日期，如 20150814，需要包括完整的年月日。
- 格式二：带分隔符日期，如 2014-08-14，分隔符可以是单个任意非数字字符。

返回值类型：timestamp without time zone

示例：

```
 SELECT to_date('2015-08-14');
    to_date
---------------------
 2015-08-14 00:00:00
(1 row)
```

❖ to_date(text, text)

描述：将字符串类型的值转换为指定格式的日期。

返回值类型：timestamp without time zone

示例：

```
 SELECT to_date('05 Dec 2000', 'DD Mon YYYY');
    to_date
```

```
---------------------
 2000-12-05 00:00:00
(1 row)
```

❖ to_number (expr [, fmt])

描述：将 expr 按指定格式转换为一个 NUMBER 类型的值。

类型转换格式请参考表 1。

转换十六进制字符串为十进制数字时，最多支持 16 个字节的十六进制字符串转换为无符号数。

转换十六进制字符串为十进制数字时，格式字符串中不允许出现除'x'或'X'以外的其他字符，否则报错。

返回值类型：number

示例：

```
 SELECT to_number('12,454.8-', '99G999D9S');
 to_number
-----------
  -12454.8
(1 row)
```

❖ to_number(text, text)

描述：将字符串类型的值转换为指定格式的数字。

返回值类型：numeric

示例：

```
 SELECT to_number('12,454.8-', '99G999D9S');
 to_number
-----------
  -12454.8
(1 row)
```

❖ to_timestamp(double precision)

描述：把 UNIX 纪元转换成时间戳。

返回值类型：timestamp with time zone

示例：

```
 SELECT to_timestamp(1284352323);
    to_timestamp
------------------------
 2010-09-13 12:32:03+08
(1 row)
```

❖ to_timestamp(string [,fmt])

描述：将字符串 string 按 fmt 指定的格式转换成时间戳类型的值。不指定 fmt 时，按参数 nls_timestamp_format 所指定的格式转换。

openGauss 的 to_timestamp 中，

- 如果输入的年份 YYYY=0，系统报错。
- 如果输入的年份 YYYY<0，在 fmt 中指定 SYYYY，则正确输出公元前绝对值 n 的年份。

fmt 中出现的字符必须与日期/时间格式化的模式相匹配，否则报错。

返回值类型：timestamp without time zone

示例：

```
SHOW nls_timestamp_format;
   nls_timestamp_format
----------------------------
 DD-Mon-YYYY HH:MI:SS.FF AM
(1 row)

SELECT to_timestamp('12-sep-2014');
   to_timestamp
---------------------
 2014-09-12 00:00:00
(1 row)
 SELECT to_timestamp('12-Sep-10 14:10:10.123000','DD-Mon-YY HH24:MI:SS.FF
     ');
    to_timestamp
```

```
------------------------
 2010-09-12 14:10:10.123
(1 row)
 SELECT to_timestamp('-1','SYYYY');
    to_timestamp
------------------------
 0001-01-01 00:00:00 BC
(1 row)
 SELECT to_timestamp('98','RR');
   to_timestamp
---------------------
 1998-01-01 00:00:00
(1 row)
SELECT to_timestamp('01','RR');
   to_timestamp
---------------------
 2001-01-01 00:00:00
(1 row)
```

❖ to_timestamp(text, text)

描述：将字符串类型的值转换为指定格式的时间戳。

返回值类型：timestamp

示例：

```
 SELECT to_timestamp('05 Dec 2000', 'DD Mon YYYY');
  to_timestamp
---------------------
 2000-12-05 00:00:00
(1 row)
```

表 1 数值格式化的模版模式

模式	描述
9	带有指定数值位数的值

模式	描述
0	带前导零的值
.（句点）	小数点
,（逗号）	分组（千）分隔符
PR	尖括号内负值
S	带符号的数值（使用区域设置）
L	货币符号（使用区域设置）
D	小数点（使用区域设置）
G	分组分隔符（使用区域设置）
MI	在指明的位置的负号（如果数字 < 0）
PL	在指明的位置的正号（如果数字 > 0）
SG	在指明的位置的正/负号
RN	罗马数字（输入在 1 和 3999 之间）
TH 或 th	序数后缀
V	移动指定位（小数）

- abstime_text

  描述：将 abstime 类型转为 text 类型输出。

  参数：abstime

  返回值类型：text

- abstime_to_smalldatetime

  描述：将 abstime 类型转为 smalldatatime 类型。

  参数：abstime

  返回值类型：smalldatetime

- bigint_tid

  描述：将 bigint 转为 tid。

  参数：bigint

返回值类型：tid

- bool_int1

  描述：将 bool 转为 int1。

  参数：boolean

  返回值类型：tinyint

- bool_int2

  描述：将 bool 转为 int2。

  参数：boolean

  返回值类型：smallint

- bool_int8

  描述：将 bool 转为 tinyint。

  参数：boolean

  返回值类型：bigint

- bpchar_date

  描述：将字符串转为日期。

  参数：character

  返回值类型：date

- bpchar_float4

  描述：将字符串转为 float4。

  参数：character

  返回值类型：real

- bpchar_float8

  描述：将字符串转为 float8。

  参数：character

  返回值类型：double precision

- bpchar_int4

  描述：将字符串转为 int4。

  参数：character

  返回值类型：integer

- bpchar_int8

  描述：将字符串转为 tinyint。

参数：character

返回值类型：bigint

- bpchar_numeric

  描述：将字符串转为 numeric。

  参数：character

  返回值类型：numeric

- bpchar_timestamp

  描述：将字符串转为时间戳。

  参数：character

  返回值类型：timestamp without time zone

- bpchar_to_smalldatetime

  描述：将字符串转为 smalldatetime。

  参数：character

  返回值类型：smalldatetime

- cupointer_bigint

  描述：将列存 CU 指针类型转为 bigint 类型。

  参数：text

  返回值类型：bigint

- date_bpchar

  描述：将 date 类型转换为 bpchar 类型。

  参数：date

  返回值类型：character

- date_text

  描述：将 date 类型转换为 text 类型。

  参数：date

  返回值类型：text

- date_varchar

  描述：将 date 类型转换为 varchar 类型。

  参数：date

  返回值类型：character varying

- f4toi1

描述：把 float4 类型强转为 tinyint unsigned 类型。

参数：real

返回值类型：tinyint

- f8toi1

  描述：把 float8 类型强转为 tinyint unsigned 类型。

  参数：double precision

  返回值类型：tinyint

- float4_bpchar

  描述：float4 转换为 bpchar。

  参数：real

  返回值类型：character

- float4_text

  描述：float4 转换为 text。

  参数：real

  返回值类型：text

- float4_varchar

  描述：float4 转换为 varchar。

  参数：real

  返回值类型：character varying

- float8_bpchar

  描述：float8 转换为 bpchar。

  参数：double precision

  返回值类型：character

- float8_interval

  描述：float8 转换为 interval。

  参数：double precision

  返回值类型：interval

- float8_text

  描述：float8 转换为 text。

  参数：double precision

  返回值类型：text

- float8_varchar

  描述：float8 转换为 varchar。

  参数：double precision

  返回值类型：character varying

- i1tof4

  描述：tinyint unsigned 转换为 float4。

  参数：tinyint

  返回值类型：real

- i1tof8

  描述：tinyint unsigned 转换为 float8。

  参数：tinyint

  返回值类型：double precision

- i1toi2

  描述：tinyint unsigned 转换为 smallint。

  参数：tinyint

  返回值类型：smallint

- i1toi4

  描述：tinyint unsigned 转换为 int。

  参数：tinyint

  返回值类型：integer

- i1toi8

  描述：tinyint unsigned 转换为 bigint。

  参数：tinyint

  返回值类型：bigint

- i2toi1

  描述：int16 转换为 uint8。

  参数：smallint

  返回值类型：tinyint

- i4toi1

  描述：int 转换为 tinyint unsigned。

  参数：integer

返回值类型：tinyint

- i8toi1

  描述：bigint 转换为 tinyint unsigned。

  参数：bigint

  返回值类型：tinyint

- int1_avg_accum

  描述：将第二个 tinyint unsigned 类型参数，加入到第一个参数中，一个参数为 bigint 类型数组。

  参数：bigint[], tinyint

  返回值类型：bigint[]

- int1_bool

  描述：tinyint unsigned 转换为 bool。

  参数：tinyint

  返回值类型：boolean

- int1_bpchar

  描述：tinyint unsigned 转换为 bpchar。

  参数：tinyint

  返回值类型：character

- int1_mul_cash

  描述：返回一个 tinyint unsigned 类型参数和一个 cash 类型参数的乘积，返回值为 cash 类型。

  参数：tinyint, money

  返回值类型：money

- int1_numeric

  描述：tinyint unsigned 转换为 numeric。

  参数：tinyint

  返回值类型：numeric

- int1_nvarchar2

  描述：tinyint unsigned 转换为 nvarchar2。

  参数：tinyint

  返回值类型：nvarchar2

- int1_text

  描述：tinyint unsigned 转换为 text。

  参数：tinyint

  返回值类型：text

- int1_varchar

  描述：tinyint unsigned 转换为 varchar。

  参数：tinyint

  返回值类型：character varying

- int1in

  描述：字符串转化为无符号一字节整数。

  参数：cstring

  返回值类型：tinyint

- int1out

  描述：无符号一字节整数转化为字符串。

  参数：tinyint

  返回值类型：cstring

- int1up

  描述：输入整数转化为无符号一字节整数。

  参数：tinyint

  返回值类型：tinyint

- int2_bool

  描述：将有符号二字节整数转化为 bool 型。

  参数：smallint

  返回值类型：boolean

- int2_bpchar

  描述：将有符号二字节整数转化为 BpChar。

  参数：smallint

  返回值类型：character

- int2_text

  描述：有符号二字节整数转化为 text 类型。

  参数：smallint

返回值类型：text

- int2_varchar

  描述：有符号二字节整数转化为 varchar 类型。

  参数：smallint

  返回值类型：character varying

- int8_text

  描述：tinyint 转化为 text 类型。

  参数：bigint

  返回值类型：text

- int8_varchar

  描述：tinyint 转化为 varchar。

  参数：bigint

  返回值类型：character varying

- intervaltonum

  描述：将内部数据类型日期转化为 numeric 类型。

  参数：interval

  返回值类型：numeric

- numeric_bpchar

  描述：numeric 转化为 bpchar。

  参数：numeric

  返回值类型：character

- numeric_int1

  描述：numeric 转化为有符号 1 字节整数。

  参数：numeric

  返回值类型：tinyint

- numeric_text

  描述：numeric 转化为 text。

  参数：numeric

  返回值类型：text

- numeric_varchar

  描述：numeric 转化为 varchar。

参数：numeric

返回值类型：character varying

- nvarchar2in

  描述：将 c 字符串转化为 varchar。

  参数：cstring, oid, integer

  返回值类型：nvarchar2

- nvarchar2out

  描述：将 text 转化为 c 字符串。

  参数：nvarchar2

  返回值类型：cstring

- nvarchar2send

  描述：将 varchar 转化为二进制。

  参数：nvarchar2

  返回值类型：bytea

- oidvectorin_extend

  描述：将字符串转化为 oidvector。

  参数：cstring

  返回值类型：oidvector_extend

- oidvectorout_extend

  描述：将 oidvector 转化为字符串。

  参数：oidvector_extend

  返回值类型：cstring

- oidvectorsend_extend

  描述：将 oidvector 转化为字符串。

  参数：oidvector_extend

  返回值类型：bytea

- reltime_text

  描述：reltime 转换为 text。

  参数：reltime

  返回值类型：text

- text_date

描述：text 类型转换为 date 类型。

参数：text

返回值类型：date

- text_float4

  描述：text 类型转换为 float4 类型。

  参数：text

  返回值类型：real

- text_float8

  描述：text 类型转换为 float8 类型。

  参数：text

  返回值类型：double precision

- text_int1

  描述：text 类型转换为 int1 类型。

  参数：text

  返回值类型：tinyint

- text_int2

  描述：text 类型转换为 int2 类型。

  参数：text

  返回值类型：smallint

- text_int4

  描述：text 类型转换为 int4 类型。

  参数：text

  返回值类型：integer

- text_int8

  描述：text 类型转换为 tinyint 类型。

  参数：text

  返回值类型：bigint

- text_numeric

  描述：text 类型转换为 numeric 类型。

  参数：text

  返回值类型：numeric

- text_timestamp

  描述：text 类型转换为 timestamp 类型。

  参数：text

  返回值类型：timestamp without time zone

- time_text

  描述：time 类型转换为 text 类型。

  参数：time without time zone

  返回值类型：text

- timestamp_text

  描述：timestamp 类型转换为 text 类型。

  参数：timestamp without time zone

  返回值类型：text

- timestamp_to_smalldatetime

  描述：timestamp 类型转换为 smalldatetime 类型。

  参数：timestamp without time zone

  返回值类型：smalldatetime

- timestamp_varchar

  描述：timestamp 类型转换为 varchar 类型。

  参数：timestamp without time zone

  返回值类型：character varying

- timestamptz_to_smalldatetime

  描述：timestamptz 类型转换为 smalldatetime。

  参数：timestamp with time zone

  返回值类型：smalldatetime

- timestampzone_text

  描述：timestampzone 类型转换为 text 类型。

  参数：timestamp with time zone

  返回值类型：text

- timetz_text

  描述：timetz 类型转换为 text 类型。

  参数：time with time zone

返回值类型：text

- to_integer

  描述：转换为 integer 类型。

  参数：character varying

  返回值类型：integer

- to_interval

  描述：转换为 interval 类型。

  参数：character varying

  返回值类型：interval

- to_numeric

  描述：转换为 numeric 类型。

  参数：character varying

  返回值类型：numeric

- to_nvarchar2

  描述：转换为 nvarchar2 类型。

  参数：numeric

  返回值类型：nvarchar2

- to_text

  描述：转换为 text 类型。

  参数：smallint

  返回值类型：text

- to_ts

  描述：转换为 ts 类型。

  参数：character varying

  返回值类型：timestamp without time zone

- to_varchar2

  描述：转换为 varchar2 类型。

  参数：timestamp without time zone

  返回值类型：character varying

- varchar_date

  描述：varchar 类型转换为 date。

参数：character varying

返回值类型：date

- varchar_float4

  描述：varchar 类型转换为 float4。

  参数：character varying

  返回值类型：real

- varchar_float8

  描述：varchar 类型转换为 float8。

  参数：character varying

  返回值类型：double precision

- varchar_int4

  描述：varchar 类型转换为 int4。

  参数：character varying

  返回值类型：integer

- varchar_int8

  描述：varchar 类型转换为 tinyint 。

  参数：character varying

  返回值类型：bigint

- varchar_numeric

  描述：varchar 类型转换为 numeric。

  参数：character varying

  返回值类型：numeric

- varchar_timestamp

  描述：varchar 类型转换为 timestamp。

  参数：character varying

  返回值类型：timestamp without time zone

- varchar2_to_smlldatetime

  描述：varchar2 类型转换为 smlldatetime。

  参数：character varying

  返回值类型：smalldatetime

- xidout4

描述：xid 输出为 4 字节数字。

参数：xid32

返回值类型：cstring

❖ xidsend4

描述：xid 转换为二进制格式。

参数：xid32

返回值类型：bytea

编码类型转换

❖ convert_to_nocase(text, text)

描述：将字符串转换为指定的编码类型。

返回值类型：bytea

示例：

```
SELECT convert_to_nocase('12345', 'GBK');
 convert_to_nocase
-------------------
 x3132333435
(1 row)
```

4.1.4.7.数字操作函数和操作符

数字操作符

❖ +

描述：加

示例：

```
 SELECT 2+3 AS RESULT;
 result
--------
    5
(1 row)
```

❖ -

描述：减

示例：

```
SELECT 2-3 AS RESULT;
 result
--------
   -1
(1 row)
```

- *

描述：乘

示例：

```
SELECT 2*3 AS RESULT;
 result
--------
    6
(1 row)
```

- /

描述：除（除法操作符不会取整）

示例：

```
SELECT 4/2 AS RESULT;
 result
--------
    2
(1 row)
SELECT 4/3 AS RESULT;
    result
------------------
 1.33333333333333
(1 row)
```

- +/-

描述：正/负

示例：

```
SELECT -2 AS RESULT;
 result
```

```
--------
   -2
(1 row)
```

❖ %

描述：模（求余）

示例：

```
SELECT 5%4 AS RESULT;
 result
--------
      1
(1 row)
```

❖ @

描述：绝对值

示例：

```
SELECT @ -5.0 AS RESULT;
 result
--------
    5.0
(1 row)
```

❖ ^

描述：幂（指数运算）

示例：

```
SELECT 2.0^3.0 AS RESULT;
       result
--------------------
 8.0000000000000000
(1 row)
```

❖ |/

描述：平方根

示例：

```
SELECT |/ 25.0 AS RESULT;
 result
--------
```

```
   5
(1 row)
```

- ||/

  描述：立方根

  示例：

```
SELECT ||/ 27.0 AS RESULT;
 result
--------
      3
(1 row)
```

- !

  描述：阶乘

  示例：

```
SELECT 5! AS RESULT;
 result
--------
    120
(1 row)
```

- !!

  描述：阶乘（前缀操作符）

  示例：

```
SELECT !!5 AS RESULT;
 result
--------
    120
(1 row)
```

- &

  描述：二进制 AND

  示例：

```
SELECT 91&15  AS RESULT;
 result
--------
     11
```

```
(1 row)
```

- |

  描述：二进制 OR

  示例：

```
SELECT 32|3 AS RESULT;
 result
--------
     35
(1 row)
```

- #

  描述：二进制 XOR

  示例：

```
SELECT 17#5 AS RESULT;
 result
--------
     20
(1 row)
```

- ~

  描述：二进制 NOT

  示例：

```
SELECT ~1 AS RESULT;
 result
--------
     -2
(1 row)
```

- <<

  描述：二进制左移

  示例：

```
SELECT 1<<4 AS RESULT;
 result
--------
     16
(1 row)
```

❖ >>

描述：二进制右移

示例：

```
SELECT 8>>2 AS RESULT;
result
--------
   2
(1 row)
```

数字操作函数

❖ abs(x)

描述：绝对值。

返回值类型：和输入相同。

示例：

```
SELECT abs(-17.4);
 abs
------
 17.4
(1 row)
```

❖ acos(x)

描述：反余弦。

返回值类型：double precision

示例：

```
SELECT acos(-1);
      acos
------------------
 3.14159265358979
(1 row)
```

❖ asin(x)

描述：反正弦。

返回值类型：double precision

示例：

```
SELECT asin(0.5);
     asin
------------------
 .523598775598299
(1 row)
```

- atan(x)

  描述：反正切。

  返回值类型：double precision

  示例：

```
SELECT atan(1);
     atan
------------------
 .785398163397448
(1 row)
```

- atan2(y, x)

  描述：y/x 的反正切。

  返回值类型：double precision

  示例：

```
SELECT atan2(2, 1);
    atan2
------------------
 1.10714871779409
(1 row)
```

- bitand(integer, integer)

  描述：计算两个数字与运算(&)的结果。

  返回值类型：bigint 类型数字。

  示例：

```
SELECT bitand(127, 63);
 bitand
--------
     63
(1 row)
```

- cbrt(dp)

  描述：立方根。

  返回值类型：double precision

  示例：

```
SELECT cbrt(27.0);
 cbrt
------
    3
(1 row)
```

- ceil(x)

  描述：不小于参数的最小的整数。

  返回值类型：整数。

  示例：

```
SELECT ceil(-42.8);
 ceil
------
  -42
(1 row)
```

- ceiling(dp or numeric)

  描述：不小于参数的最小整数（ceil 的别名）。

  返回值类型：dp or numeric，不考虑隐式类型转换的情况下与输入相同。

  示例：

```
SELECT ceiling(-95.3);
 ceiling
---------
     -95
(1 row)
```

- cos(x)

  描述：余弦。

  返回值类型：double precision

  示例：

```
SELECT cos(-3.1415927);
      cos
```

```
-------------------
 -.999999999999999
(1 row)
```

- cot(x)

  描述：余切。

  返回值类型：double precision

  示例：

```
SELECT cot(1);
      cot
------------------
 .642092615934331
(1 row)
```

- degrees(dp)

  描述：把弧度转为角度。

  返回值类型：double precision

  示例：

```
SELECT degrees(0.5);
    degrees
------------------
 28.6478897565412
(1 row)
```

- div(y numeric, x numeric)

  描述：y 除以 x 的商的整数部分。

  返回值类型：numeric

  示例：

```
SELECT div(9,4);
 div
-----
  2
(1 row)
```

- exp(x)

  描述：自然指数。

  返回值类型：dp or numeric，不考虑隐式类型转换的情况下与输入相同。

示例：

```
SELECT exp(1.0);
        exp
--------------------
 2.7182818284590452
(1 row)
```

- floor(x)

  描述：不大于参数的最大整数。

  返回值类型：与输入相同。

  示例：

```
SELECT floor(-42.8);
floor
-------
  -43
(1 row)
```

- int1(in)

  描述：将传入的 text 参数转换为 int1 类型值并返回。

  返回值类型：int1

  取值范围：0 ~ 255

  示例：

```
select int1('123');
int1
------
 123
(1 row)
```

- int2(in)

  描述：将传入参数转换为 int2 类型值并返回。

  支持的入参类型包括 float4、float8、int16、numeric、text。

  返回值类型：int2

  取值范围：-32,768 ~ +32,767

  示例：

```
select int2('1234');
int2
```

```
------
 1234
(1 row)
 select int2(25.3);
 int2
------
  25
(1 row)
```

❖ int4(in)

描述：将传入参数转换为 int4 类型值并返回。

支持的入参类型包括 bit、boolean、char、duoble precision、int16、numeric、real、smallint、text。

返回值类型：int4

取值范围：-2,147,483,648 ~ +2,147,483,647

示例：

```
 select int4('789');
 int4
------
 789
(1 row)

select int4(99.9);
 int4
------
 100
(1 row)
```

❖ float4(in)

描述：将传入参数转换为 float4 类型值并返回。支持的入参类型包括：bigint，duoble precision，int16, integer, numeric，smallint，text。

返回值类型：float4

示例：

```
select float4('789');
```

```
 float4
--------
   789
(1 row)

 select float4(99.9);
 float4
--------
  99.9
(1 row)
```

- float8(in)

  描述：将传入参数转换为 float8 类型值并返回。支持的入参类型包括：bigint，int16, integer, numeric，real，smallint，text。

  返回值类型：float8

  示例：

```
 select float8('789');
 float8
--------
   789
(1 row)

 select float8(99.9);
 float8
--------
  99.9
(1 row)
```

- int16(in)

  描述：将传入参数转换为 int16 类型值并返回。支持的入参类型包括：bigint，boolean，double precision，integer，numeric，oid，real，smallint，tinyint。

  返回值类型：int16

  示例：

```
select int16('789');
```

```
 int16
--------
   789
(1 row)

 select int16(99.9);
 int16
-------
   100
(1 row)
```

- numeric(in)

  描述：将传入参数转换为 numeric 类型值并返回。支持的入参类型包括：bigint，boolean，double precision，int16，integer，money，real，smallint。

  返回值类型：numeric

  示例：

```
 select "numeric"('789');
 numeric
---------
   789
(1 row)

 select "numeric"(99.9);
 numeric
---------
   99.9
(1 row)
```

- oid(in)

  描述：将传入参数转换为 oid 类型值并返回。支持的入参类型包括：bigint，int16。

  返回值类型：oid

- radians(dp)

  描述：把角度转为弧度。

返回值类型：double precision

示例：

```
SELECT radians(45.0);
    radians
------------------
 .785398163397448
(1 row)
```

- random()

  描述：0.0 到 1.0 之间的随机数。

  返回值类型：double precision

  示例：

```
SELECT random();
     random
------------------
 .824823560658842
(1 row)
```

- multiply(x double precision or text, y double precision or text)

  描述：x 和 y 的乘积。仅支持 multiply(x double precision, y text)和 multiply(x text, y double precision)的两种场景。

  返回值类型：double precision

  示例：

```
SELECT multiply(9.0, '3.0');
    multiply
-------------------
          27
(1 row)
 SELECT multiply('9.0', 3.0);
    multiply
-------------------
          27
(1 row)
```

- ln(x)

  描述：自然对数。

返回值类型：dp or numeric，不考虑隐式类型转换的情况下与输入相同。

示例：

```
SELECT ln(2.0);
        ln
-------------------
 .6931471805599453
(1 row)
```

- log(x)

  描述：以 10 为底的对数。

  返回值类型：与输入相同。

  示例：

```
SELECT log(100.0);
 log
-----
   2
(1 row)
```

- log(b numeric, x numeric)

  描述：以 b 为底的对数。

  返回值类型：numeric

  示例：

```
SELECT log(2.0, 64.0);
 log
-----
   6
(1 row)
```

- mod(x,y)

  描述：x/y 的余数（模）。如果 x 是 0，则返回 0。

  返回值类型：与参数类型相同。

  示例：

```
SELECT mod(9,4);
 mod
-----
   1
```

```
(1 row)
 SELECT mod(9,0);
 mod
-----
  9
(1 row)
```

- pi()

  描述：“π”常量。

  返回值类型：double precision

  示例：

```
SELECT pi();
      pi
------------------
 3.14159265358979
(1 row)
```

- power(a double precision, b double precision)

  描述：a 的 b 次幂。

  返回值类型：double precision

  示例：

```
SELECT power(9.0, 3.0);
 power
-------
  729
(1 row)
```

- round(x)

  描述：离输入参数最近的整数。

  返回值类型：与输入相同（double precision 或者 numeric 类型）。

  示例：

```
SELECT round(42.4);
 round
-------
   42
(1 row)
```

```
SELECT round(42.6);
 round
-------
    43
(1 row)
```

- round(v numeric, s int)

  描述：保留小数点后 s 位，s 后一位进行四舍五入。

  返回值类型：numeric

  示例：

```
SELECT round(42.4382, 2);
 round
-------
 42.44
(1 row)
```

- setseed(dp)

  描述：为随后的 random()调用设置种子(-1.0 到 1.0 之间，包含)。

  返回值类型：void

  示例：

```
SELECT setseed(0.54823);
 setseed
---------

(1 row)
```

- sign(x)

  描述：输出此参数的符号。

  返回值类型：-1 表示负数，0 表示 0，1 表示正数。

  示例：

```
SELECT sign(-8.4);
 sign
------
   -1
(1 row)
```

- sin(x)

  描述：正弦。

  返回值类型：double precision

  示例：

```
SELECT sin(1.57079);
      sin
------------------
 .999999999979986
(1 row)
```

- sqrt(x)

  描述：平方根。

  返回值类型：dp or numeric，不考虑隐式类型转换的情况下与输入相同。

  示例：

```
SELECT sqrt(2.0);
      sqrt
-------------------
 1.414213562373095
(1 row)
```

- tan(x)

  描述：正切。

  返回值类型：double precision

  示例：

```
SELECT tan(20);
      tan
------------------
 2.23716094422474
(1 row)
```

- trunc(x)

  描述：截断（取整数部分）。

  返回值类型：与输入相同。

  示例：

```
SELECT trunc(42.8);
trunc
```

```
-------
  42
(1 row)
```

- trunc(v numeric, s int)

  描述：截断为 s 位小数。

  返回值类型：numeric

  示例：

```
SELECT trunc(42.4382, 2);
trunc
-------
 42.43
(1 row)
```

- smgrne(a smgr, b smgr)

  描述：比较两个 smgr 类型整数是否不相等。

  返回值类型：bool

- smgreq(a smgr, b smgr)

  描述：比较两个 smgr 类型整数是否相等。

  返回值类型：bool

- int1abs

  描述：返回 uint8 类型数据的绝对值。

  参数：tinyint

  返回值类型：tinyint

- int1and

  描述：返回两个 uint8 类型数据按位与的结果。

  参数：tinyint, tinyint

  返回值类型：tinyint

- int1cmp

  描述：返回两个 uint8 类型数据比较的结果，若第一个参数大，则返回 1；若第二个参数大，则返回-1；若相等，则返回 0。

  参数：tinyint, tinyint

  返回值类型：integer

- int1div

描述：返回两个 uint8 类型数据相除的结果，结果为 float8 类型。

参数：tinyint, tinyint

返回值类型：tinyint

- int1eq

  描述：比较两个 uint8 类型数据是否相等。

  参数：tinyint, tinyint

  返回值类型：boolean

- int1ge

  描述：判断两个 uint8 类型数据是否第一个参数大于等于第二个参数。

  参数：tinyint, tinyint

  返回值类型：boolean

- int1gt

  描述：无符号 1 字节整数做大于运算。

  参数：tinyint, tinyint

  返回值类型：boolean

- int1larger

  描述：无符号 1 字节整数求最大值。

  参数：tinyint, tinyint

  返回值类型：tinyint

- int1le

  描述：无符号 1 字节整数做小于等于运算。

  参数：tinyint, tinyint

  返回值类型：boolean

- int1lt

  描述：无符号 1 字节整数做小于运算。

  参数：tinyint, tinyint

  返回值类型：boolean

- int1smaller

  描述：无符号 1 字节整数求最小算。

  参数：tinyint, tinyint

  返回值类型：tinyint

- int1inc

  描述：无符号 1 字节整数加一。

  参数：tinyint

  返回值类型：tinyint

- int1mi

  描述：无符号一字节整数做差运算。

  参数：tinyint, tinyint

  返回值类型：tinyint

- int1mod

  描述：无符号一字节整数做取余运算。

  参数：tinyint, tinyint

  返回值类型：tinyint

- int1mul

  描述：无符号一字节整数做乘法运算。

  参数：tinyint, tinyint

  返回值类型：tinyint

- int1ne

  描述：无符号一字节整数不等于运算。

  参数：tinyint, tinyint

  返回值类型：boolean

- int1pl

  描述：无符号一字节整数加法。

  参数：tinyint, tinyint

  返回值类型：tinyint

- int1um

  描述：无符号一字节数去相反数并返回有符号二字节整数。

  参数：tinyint

  返回值类型：smallint

- int1xor

  描述：无符号一字节整数异或操作。

  参数：tinyint, tinyint

返回值类型：tinyint

- cash_div_int1

  描述：对 money 类型进行除法运算。

  参数：money, tinyint

  返回值类型：money

- cash_mul_int1

  描述：对 money 类型进行乘法运算。

  参数：money, tinyint

  返回值类型：money

- int1not

  描述：无符号一字节整数二进制位翻转。

  参数：tinyint

  返回值类型：tinyint

- int1or

  描述：无符号一字节整数或运算。

  参数：tinyint, tinyint

  返回值类型：tinyint

- int1shl

  描述：无符号一字节整数左移指定位数。

  参数：tinyint, integer

  返回值类型：tinyint

- width_bucket(op numeric, b1 numeric, b2 numeric, count int)

  描述：返回一个桶，这个桶是在一个有 count 个桶，上界为 b1 下界为 b2 的等深柱图中 operand 将被赋予的那个桶。

  返回值类型：int

  示例：

```
SELECT width_bucket(5.35, 0.024, 10.06, 5);
 width_bucket
--------------
            3
(1 row)
```

❖ width_bucket(op dp, b1 dp, b2 dp, count int)

描述：返回一个桶，这个桶是在一个有 count 个桶，上界为 b1 下界为 b2 的等深柱图中 operand 将被赋予的那个桶。

返回值类型：int

示例：

```
SELECT width_bucket(5.35, 0.024, 10.06, 5);
 width_bucket
--------------
            3
(1 row)
```

4.1.4.8.字符处理函数和操作符

PanWeiDB 提供的字符处理函数和操作符主要用于字符串与字符串、字符串与非字符串之间的连接，以及字符串的模式匹配操作。注意：字符串处理函数除了 length 相关函数，其他函数和操作符不支持大于 1GB clob 作为参数。

❖ **bit_length(string)**

描述：字符串的位数。

返回值类型：int

示例：

```
panweidb=# SELECT bit_length('world');
 bit_length
------------
         40
(1 row)
```

❖ **btrim(string text [, characters text])**

描述：从 string 开头和结尾删除只包含 characters 中字符（缺省是空白）的最长字符串。

返回值类型：text

示例：

```
panweidb=# SELECT btrim('sring' , 'ing');
 btrim
```

```
-------
sr
(1 row)
```

❖ **char_length(string)或 character_length(string)**

描述：字符串中的字符个数。

返回值类型：int

示例：

```
panweidb=# SELECT char_length('hello');
 char_length
-------------
           5
(1 row)
```

❖ **instr(text,text,int,int)**

描述：instr(string1,string2,int1,int2)返回在 string1 中从 int1 位置开始匹配到第 int2 次 string2 的位置，第一个 int 表示开始匹配起始位置，第二个 int 表示匹配的次数。

返回值类型：int

示例：

```
panweidb=# SELECT instr( 'abcdabcdabcd', 'bcd', 2, 2 );
 instr
-------
     6
(1 row)
```

❖ **lengthb(text/bpchar)**

描述：获取指定字符串的字节数。

返回值类型：int

示例：

```
panweidb=# SELECT lengthb('hello');
 lengthb
---------
       5
(1 row)
```

❖ **left(str text, n int)**

描述：返回字符串的前 n 个字符。当 n 是负数时，返回除最后|n|个字符以外的所有字符。

返回值类型：text

示例：

```
panweidb=# SELECT left('abcde', 2);
 left
------
 ab
(1 row)
```

❖ **length(string bytea, encoding name)**

描述：指定 encoding 编码格式的 string 的字符数。在这个编码格式中，string 必须是有效的。

返回值类型：int

示例：

```
panweidb=# SELECT length('jose', 'UTF8');
length
--------
    4
(1 row)
```

【说明】

如果是查询 bytea 类型的长度，指定 utf8 编码时，最大长度只能为 536870888。

❖ **lpad(string text, length int [, fill text])**

描述：通过填充字符 fill（缺省时为空白），把 string 填充为 length 长度。如果 string 已经比 length 长则将其尾部截断。

返回值类型：text

示例：

```
panweidb=# SELECT lpad('hi', 5, 'xyza');
 lpad
```

```
-------
 xyzhi
(1 row)
```

- **notlike(x bytea name text, y bytea text)**

  描述：比较 x 和 y 是否不一致。

  返回值类型：bool

  示例：

```
panweidb=# SELECT notlike(1,2);
 notlike
--------------
      t
(1 row)
panweidb=# SELECT notlike(1,1);
   notlike
--------------
      f
(1 row)
```

- **octet_length(string)**

  描述：字符串中的字节数。

  返回值类型：int

  示例：

```
panweidb=# SELECT octet_length('jose');
 octet_length
--------------
      4
(1 row)
```

- **overlay(string placing string FROM int [for int])**

  描述：替换子字符串。FROM int 表示从第一个 string 的第几个字符开始替换，for int 表示替换第一个 string 的字符数目。

  返回值类型：text

  示例：

```
panweidb=# SELECT overlay('hello' placing 'world' from 2 for 3 );
```

```
overlay
---------
hworldo
(1 row)
```

❖ **position(substring in string)**

描述：指定子字符串的位置。字符串区分大小写。

返回值类型：int，字符串不存在时返回 0。

示例：

```
panweidb=# SELECT position('ing' in 'string');
 position
----------
        4
(1 row)
```

❖ **pg_client_encoding()**

描述：当前客户端编码名称。

返回值类型：name

示例：

```
panweidb=# SELECT pg_client_encoding();
 pg_client_encoding
--------------------
 UTF8
(1 row)
```

❖ **quote_ident(string text)**

描述：返回适用于 SQL 语句的标识符形式（使用适当的引号进行界定）。只有在必要的时候才会添加引号（字符串包含非标识符字符或者会转换大小写的字符）。返回值中嵌入的引号都写了两次。

返回值类型：text

示例：

```
panweidb=# SELECT quote_ident('hello world');
 quote_ident
--------------
```

```
"hello world"
(1 row)
```

❖ **quote_literal(string text)**

描述：返回适用于在 SQL 语句里当作文本使用的形式（使用适当的引号进行界定）。

返回值类型：text

示例：

```
panweidb=# SELECT quote_literal('hello');
 quote_literal
---------------
 'hello'
(1 row)
```

如果出现如下写法，text 文本将进行转义。

```
panweidb=# SELECT quote_literal(E'O'hello');
 quote_literal
---------------
 'O''hello'
(1 row)
```

如果出现如下写法，反斜杠会写入两次。

```
panweidb=# SELECT quote_literal('Ohello');
 quote_literal
---------------
 E'Ohello'
(1 row)
```

如果参数为 NULL，返回空。如果参数可能为 null，通常使用函数 quote_nullable 更适用。

```
panweidb=# SELECT quote_literal(NULL);
 quote_literal
---------------

(1 row)
```

❖ **quote_literal(value anyelement)**

描述：将给定的值强制转换为 text，加上引号作为文本。

返回值类型：text

示例：

```
panweidb=# SELECT quote_literal(42.5);
 quote_literal
---------------
 '42.5'
(1 row)
```

如果出现如下写法，定值将进行转义。

```
panweidb=# SELECT quote_literal(E'O'42.5');
 quote_literal
---------------
 '0''42.5'
(1 row)
```

如果出现如下写法，反斜杠会写入两次。

```
panweidb=# SELECT quote_literal('O42.5');
 quote_literal
---------------
 E'O42.5'
(1 row)
```

❖ **quote_nullable(string text)**

描述：返回适用于在 SQL 语句里当作字符串使用的形式（使用适当的引号进行界定）。

返回值类型：text

示例：

```
panweidb=# SELECT quote_nullable('hello');
 quote_nullable
----------------
 'hello'
(1 row)
```

如果出现如下写法，text 文本将进行转义。

```
panweidb=# SELECT quote_nullable(E'O'hello');
```

```
 quote_nullable
----------------
 'O''hello'
(1 row)
```

如果出现如下写法，反斜杠会写入两次。

```
panweidb=# SELECT quote_nullable('Ohello');
   quote_nullable
----------------
   E'Ohello'
(1 row)
```

如果参数为 NULL，返回 NULL。

```
panweidb=# SELECT quote_nullable(NULL);
 quote_nullable
----------------
 NULL
(1 row)
```

❖ **quote_nullable(value anyelement)**

描述：将给定的参数值转化为 text，加上引号作为文本。

返回值类型：text

示例：

```
panweidb=# SELECT quote_nullable(42.5);
 quote_nullable
----------------
'42.5'
(1 row)
```

如果出现如下写法，定值将进行转义。

```
panweidb=# SELECT quote_nullable(E'O'42.5');
quote_nullable
----------------
 'O''42.5'
(1 row)
```

如果出现如下写法，反斜杠会写入两次。

```
panweidb=# SELECT quote_nullable('O42.5');
```

```
quote_nullable
----------------
E'O42.5'
(1 row)
```

如果参数为 NULL，返回 NULL。

```
panweidb=# SELECT quote_nullable(NULL);
quote_nullable
----------------
NULL
(1 row)
```

❖ **substring_inner(string [from int] [for int])**

描述：截取子字符串，from int 表示从第几个字符开始截取，for int 表示截取几个字节。

返回值类型：text

示例：

```
panweidb=# select substring_inner('adcde', 2,3);
substring_inner
-----------------
dcd
(1 row)
```

❖ **substring(string [from int] [for int])**

描述：截取子字符串，from int 表示从第几个字符开始截取，for int 表示截取几个字节。

返回值类型：text

示例：

```
panweidb=# SELECT substring('Thomas' from 2 for 3);
substring
-----------
hom
(1 row)
```

❖ **substring(string from pattern)**

描述：截取匹配 POSIX 正则表达式的子字符串。如果没有匹配它返回空

值，否则返回文本中匹配模式的那部分。

返回值类型：text

示例：

```
panweidb=# SELECT substring('Thomas' from '...$');
 substring
-----------
mas
(1 row)
```

```
panweidb=# SELECT substring('foobar' from 'o(.)b');
result
--------
  o
(1 row)
panweidb=# SELECT substring('foobar' from '(o(.)b)');
result
--------
  oob
(1 row)
```

【说明】

如果 POSIX 正则表达式模式包含任何圆括号，那么将返回匹配第一对子表达式（对应第一个左圆括号的）的文本。如果你想在表达式里使用圆括号而又不想导致这个例外，那么你可以在整个表达式外边放上一对圆括号。

❖ **substring(string from pattern for escape)**

描述：截取匹配 SQL 正则表达式的子字符串。声明的模式必须匹配整个数据串，否则函数失败并返回空值。为了标识在成功的时候应该返回的模式部分，模式必须包含逃逸字符的两次出现，并且后面要跟上双引号（"）。匹配这两个标记之间的模式的文本将被返回。

返回值类型：text

示例：

```
panweidb=# SELECT substring('Thomas' from '%#"o_a#"_' for '#');
 substring
-----------
```

```
oma
(1 row)
```

- **rawcat(raw,raw)**

  描述：字符串拼接函数。

  返回值类型：raw

  示例：

```
panweidb=# SELECT rawcat('ab','cd');
 rawcat
--------
   ABCD
(1 row)
```

- **regexp_like(text,text,text)**

  描述：正则表达式的模式匹配函数。

  返回值类型：bool

  示例：

```
panweidb=# SELECT regexp_like('str','[ac]');
   regexp_like
-------------
   f
(1 row)
```

- **regexp_substr(string text, pattern text [, position int [, occurrence int [, flags text]]])**

  描述：正则表达式的抽取子串函数。与 substr 功能相似，正则表达式出现多个并列的括号时，也全部处理。

  参数说明：

  - string：用于匹配的源字符串。
  - pattern：用于匹配的正则表达式模式串。
  - position：可选参数，表示从源字符串的第几个字符开始匹配，默认值为 1。
  - occurrence：可选参数，表示抽取第几个满足匹配的子串，为，默认值为 1。

- flags：可选参数，包含零个或多个改变函数匹配行为的单字母标记。flags 支持的选项值及含义描述如下表 1 所示：

表 1 flag 支持的选项值及含义描述

选项	描述
'b'	按照无扩展的 BRE 规则匹配。
'c'	大小写敏感匹配。
'e'	按照扩展的 ERE 规则匹配。
'i'	大小写不敏感匹配。
'm'	多行模式匹配。flags 中包含'm'时，按照多行模式匹配，否则按照单行模式匹配。
'n'	n 选项的含义和 GUC 参数 behavior_compat_options 及数据库当前的兼容模式有关。 数据库 SQL 语法兼容模式为 A 或 B，且 GUC 参数 behavior_compat_options 值包含 aformat_regexp_match 时，n 选项表示.能够匹配换行符('\n')，flags 未指定'n'选项时，.不会匹配换行符。 其他情况下，'n' 选项和 'm' 选项的含义一样。
'p'	部分新行敏感的匹配，影响.和方括号表达式，和新行敏感的匹配(‘m’或’n’)一样，但是不影响^和$。
'q'	普通字符匹配。
's'	单行模式匹配，含义与 m, n 相反。
't'	紧凑模式匹配，空白符匹配自身。
'w'	逆部分新行匹配。与 p 含义相反。
'x'	宽松模式匹配，忽略空白符。
'g'	匹配所有实例。

返回值类型：text

示例：

```
panweidb=# SELECT regexp_substr('str','[ac]');
 regexp_substr
---------------

(1 row)

panweidb=# SELECT regexp_substr('foobarbaz', 'b(..)', 3, 2) AS RESULT;
result
--------
 baz
(1 row)
```

❖ **regexp_count(string text, pattern text [, position int [, flags text]])**

描述：获取满足匹配的子串个数。

参数说明：

- string：用于匹配的源字符串。
- pattern：用于匹配的正则表达式模式串。
- position：表示从源字符串的第几个字符开始匹配，为可选参数，默认值为 1。
- flags：可选参数，包含零个或多个改变函数匹配行为的单字母标记。其中：m 表示按照多行模式匹配。SQL 语法兼容 A 和 B 的情况下，n 选项在 GUC 参数 behavior_compat_options 值包含 aformat_regexp_match 时，表示 . 能够匹配 'n' 字符，flags 中没有指定 n 时，默认 . 不能匹配 'n' 字符；值不包含 aformat_regexp_match 时，. 默认能匹配'n'字符。n 选项的含义与 m 选项一致。

  可选的参数包括 b，c，e，i，m，n，p，q，s，t，w，x，g。

返回值类型：int

示例：

```
panweidb=# SELECT regexp_count('foobarbaz','b(..)', 5) AS RESULT;
result
```

```
--------
 1
(1 row)
```

❖ **regexp_instr(string text, pattern text [, position int [, occurrence int [, return_opt int [, flags text]]]])**

描述：获取满足匹配条件的子串位置（从 1 开始）。如果没有匹配的子串，则返回 0。

参数说明：

- string：用于匹配的源字符串。
- pattern：用于匹配的正则表达式模式串。
- position：可选参数，表示从源字符串的第几个字符开始匹配，默认值为 1。
- occurrence：可选参数，表示获取第 occurrence 个匹配子串的位置，默认值为 1。
- return_opt：可选参数，用于控制返回匹配子串的首字符位置还是尾字符位置。取值为 0 时，返回匹配子串的第一个字符的位置（从 1 开始计算），取值为大于 0 的值时，返回匹配子串的尾字符的下一个字符的位置。默认值为 0。
- flags：可选参数，包含零个或多个改变函数匹配行为的单字母标记。其中：m 表示按照多行模式匹配。SQL 语法兼容 A 和 B 的情况下，n 选项在 GUC 参数 behavior_compat_options 值包含 aformat_regexp_match 时，表示 . 能够匹配 'n' 字符，flags 中没有指定 n 时，默认 . 不能匹配 'n' 字符；值不包含 aformat_regexp_match 时，. 默认能匹配'n'字符。n 选项的含义与 m 选项一致。

返回值类型：int

示例：

```
panweidb=# SELECT regexp_instr('foobarbaz','b(..)', 1, 1, 0) AS RESULT;
 result
--------
 4
```

```
(1 row)

panweidb=# SELECT regexp_instr('foobarbaz','b(..)', 1, 2, 0) AS RESULT;
 result
--------
 7
(1 row)
```

❖ **regexp_matches(string text, pattern text [, flags text])**

描述：返回 string 中所有匹配 POSIX 正则表达式的子字符串。如果 pattern 不匹配，该函数不返回行。如果模式不包含圆括号子表达式，则每一个被返回的行都是一个单一元素的文本数组，其中包括匹配整个模式的子串。如果模式包含圆括号子表达式，该函数返回一个文本数组，它的第 n 个元素是匹配模式的第 n 个圆括号子表达式的子串。

flags 参数为可选参数，包含零个或多个改变函数行为的单字母标记。i 表示进行大小写无关的匹配，g 表示替换每一个匹配的子字符串而不仅仅是第一个。

【须知】

如果提供了最后一个参数，但参数值是空字符串（''），且数据库 SQL 兼容模式设置为 A 的情况下，会导致返回结果为空集。这是因为 A 兼容模式将''作为 NULL 处理，避免此类行为的方式有如下几种：

❖ 将数据库 SQL 兼容模式改为 C；

❖ 不提供最后一个参数，或最后一个参数不为空字符串。

返回值类型：setof text[]

示例：

```
panweidb=# SELECT regexp_matches('foobarbequebaz', '(bar)(beque)');
 regexp_matches
----------------
 {bar,beque}
(1 row)
```

```
panweidb=# SELECT regexp_matches('foobarbequebaz', 'barbeque');
  regexp_matches
----------------
  {barbeque}
(1 row)
panweidb=#  SELECT regexp_matches('foobarbequebazilbarfbonk', '(b[^b]+)(b
[^b]+)', 'g');
result
--------------
  {bar,beque}
  {bazil,barf}
 (2 rows)
```

- **regexp_split_to_array(string text, pattern text [, flags text])**

  描述：用 POSIX 正则表达式作为分隔符，分隔 string。和 regexp_split_to_table 相同，不过 regexp_split_to_array 会把它的结果以一个 text 数组的形式返回。

  返回值类型：text[]

  示例：

```
panweidb=# SELECT regexp_split_to_array('hello world', E's+');
 regexp_split_to_array
-----------------------
{hello,world}
(1 row)
```

- **regexp_split_to_table(string text, pattern text [, flags text])**

  描述：用 POSIX 正则表达式作为分隔符，分隔 string。如果没有与 pattern 的匹配，该函数返回 string。如果有至少有一个匹配，对每一个匹配它都返回从上一个匹配的末尾（或者串的开头）到这次匹配开头之间的文本。当没有更多匹配时，它返回从上一次匹配的末尾到串末尾之间的文本。flags 参数包含零个或多个改变函数行为的单字母标记。i 表示进行大小写无关的匹配。

  返回值类型：setof text

示例：

```
panweidb=# SELECT regexp_split_to_table('hello world', E's+');
 regexp_split_to_table
-----------------------
 hello
 world
(2 rows)
```

- **repeat(string text, number int)**

  描述：将 string 重复 number 次。

  返回值类型：text。

  示例：

```
panweidb=# SELECT repeat('Pg', 4);
repeat
----------
PgPgPgPg
(1 row)
```

  【说明】

  由于数据库内存分配机制限制单次内存分配不可超过 1GB，因此 number 最大值不应超过(1G-x)/lengthb(string) - 1。x 为头信息长度，通常大于 4 字节，其具体值在不同的场景下存在差异。

- **replace(string text, from text, to text)**

  描述：把字符串 string 里出现地所有子字符串 from 的内容替换成子字符串 to 的内容。

  返回值类型：text

  示例：

```
panweidb=# SELECT replace('abcdefabcdef', 'cd', 'XXX');
   replace
----------------
 abXXXefabXXXef
(1 row)
```

- **replace(string, substring)**

  描述：删除字符串 string 里出现的所有子字符串 substring 的内容。

string 类型：text

substring 类型：text

返回值类型：text

示例：

```
panweidb=# SELECT replace('abcdefabcdef', 'cd');
replace
----------------
abefabef
(1 row)
```

- **reverse(str)**

描述：返回颠倒的字符串。

返回值类型：text

示例：

```
panweidb=# SELECT reverse('abcde');
 reverse
---------
edcba
(1 row)
```

- **right(str text, n int)**

描述：返回字符串中的后 n 个字符。当 n 是负值时，返回除前|n|个字符以外的所有字符。

返回值类型：text

示例：

```
panweidb=# SELECT right('abcde', 2);
  right
-------
   de
(1 row)

panweidb=# SELECT right('abcde', -2);
 right
-------
```

```
cde
(1 row)
```

❖ **rpad(string text, length int [, fill text])**

描述：使用填充字符 fill（缺省时为空白），把 string 填充到 length 长度。如果 string 已经比 length 长则将其从尾部截断。

返回值类型：text

示例：

```
panweidb=# SELECT rpad('hi', 5, 'xy');
 rpad
-------
 hixyx
(1 row)
```

❖ **rtrim(string text[,characters text])**

描述：从字符串 string 的结尾删除只包含 characters 中字符（缺省是个空白）的最长的字符串。

返回值类型：text

示例：

```
panweidb=# SELECT rtrim('trimxxxx', 'x');
rtrim
-------
   trim
(1 row)
```

❖ **substrb(text,int,int)**

描述：提取子字符串，第一个 int 表示提取的起始位置，第二个表示提取几位字符。

返回值类型：text

示例：

```
panweidb=# SELECT substrb('string',2,3);
   substrb
---------
   tri
(1 row)
```

- **substrb(text,int)**

  描述：提取子字符串，int 表示提取的起始位置。

  返回值类型：text

  示例：

```
panweidb=# SELECT substrb('string',2);
   substrb
---------
   tring
(1 row)
```

- **substr(bytea,from,count)**

  描述：从参数 bytea 中抽取子字符串。from 表示抽取的起始位置，count 表示抽取的子字符串长度。

  返回值类型：text

  示例：

```
panweidb=# SELECT substr('string',2,3);
   substr
--------
   tri
(1 row)
```

- **string || string**

  描述：连接字符串。

  返回值类型：text

  示例：

```
panweidb=# SELECT 'MPP'||'DB' AS RESULT;
 result
--------
   MPPDB
(1 row)
```

- **string || non-string 或 non-string || string**

  描述：连接字符串和非字符串。

  返回值类型：text

示例：

```
panweidb=# SELECT 'Value: '||42 AS RESULT;
result
-----------
  Value: 42
(1 row)
```

- **split_part(string text, delimiter text, field int)**

  描述：根据 delimiter 分隔 string 返回生成的第 field 个子字符串（从出现第一个 delimiter 的 text 为基础）。

  返回值类型：text

  示例：

```
panweidb=# SELECT split_part('abc~@~def~@~ghi', '~@~', 2);
  split_part
------------
  def
(1 row)
```

- **strpos(string, substring)**

  描述：指定的子字符串的位置。和 position(substring in string)一样，不过参数顺序相反。

  返回值类型：int

  示例：

```
panweidb=# SELECT strpos('source', 'rc');
  strpos
--------
       4
(1 row)
```

- **to_hex(number int or bigint)**

  描述：把 number 转换成十六进制表现形式。

  返回值类型：text

  示例：

```
panweidb=# SELECT to_hex(2147483647);
   to_hex
```

```
----------
  7fffffff
(1 row)
```

❖ **translate(string text, from text, to text)**

描述：把在 string 中包含的任何匹配 from 中字符的字符转化为对应的在 to 中的字符。如果 from 比 to 长，删掉在 from 中出现的额外的字符。

返回值类型：text

示例：

```
panweidb=# SELECT translate('12345', '143', 'ax');
  translate
-----------
  a2x5
(1 row)
```

❖ **length(string)**

描述：获取参数 string 中字符的数目。

返回值类型：integer

示例：

```
panweidb=# SELECT length('abcd');
  length
--------
       4
(1 row)
```

❖ **lengthb(string)**

描述：获取参数 string 中字节的数目。与字符集有关，同样的中文字符，在 GBK 与 UTF8 中，返回的字节数不同。

返回值类型：integer

示例：

```
panweidb=# SELECT lengthb('Chinese');
  lengthb
---------
       7
```

```
(1 row)
```

❖ **substr(string,from)**

描述：

从参数 string 中抽取子字符串。

from 表示抽取的起始位置。

- from 为 0 时，按 1 处理。
- from 为正数时，抽取从 from 到末尾的所有字符。
- from 为负数时，抽取字符串的后 n 个字符，n 为 from 的绝对值。

返回值类型：varchar

示例：

- from 为正数时：

```
panweidb=# SELECT substr('ABCDEF',2);
 substr
--------
 BCDEF
(1 row)
From
```

- 为负数时：

```
panweidb=# SELECT substr('ABCDEF',-2);
 substr
--------
 EF
(1 row)
```

❖ **substr(string,from,count)**

描述：

从参数 string 中抽取子字符串。

from 表示抽取的起始位置。

count 表示抽取的子字符串长度。

- from 为 0 时，按 1 处理。
- from 为正数时，抽取从 from 开始的 count 个字符。
- from 为负数时，抽取从倒数第 n 个开始的 count 个字符，n 为

from 的绝对值。

- count 小于 1 时，返回 null。

返回值类型：varchar

示例：

- from 为正数时：

```
panweidb=# SELECT substr('ABCDEF',2,2);
 substr
--------
 BC
(1 row)
from
```

- 为负数时：

```
panweidb=# SELECT substr('ABCDEF',-3,2);
 substr
--------
 DE
(1 row)
```

❖ **substrb(string,from)**

描述：该函数和 SUBSTR(string,from)函数功能一致，但是计算单位为字节。

返回值类型：bytea

示例：

```
panweidb=# SELECT substrb('ABCDEF',-2);
 substrb
---------
 EF
(1 row)
```

❖ **substrb(string,from,count)**

描述：该函数和 SUBSTR(string,from,count)函数功能一致，但是计算单位为字节。

返回值类型：bytea

示例：

```
panweidb=# SELECT substrb('ABCDEF',2,2);
  substrb
---------
  BC
(1 row)
```

- **trim([leading |trailing |both] [characters] from string)**

  描述：从字符串 string 的开头、结尾或两边删除只包含 characters 中字符（缺省是一个空白）的最长的字符串。

  返回值类型：text

  示例：

```
panweidb=# SELECT trim(BOTH 'x' FROM 'xTomxx');
  btrim
-------
  Tom
(1 row)

panweidb=# SELECT trim(LEADING 'x' FROM 'xTomxx');
  ltrim
-------
  Tomxx
(1 row)

panweidb=# SELECT trim(TRAILING 'x' FROM 'xTomxx');
rtrim
-------
  xTom
(1 row)
```

- **rtrim([string [, characters])**

  描述：从字符串 string 的结尾删除只包含 characters 中字符（缺省是个空白）的最长的字符串。

  返回值类型：text

  示例：

```
panweidb=# SELECT rtrim('TRIMxxxx','x');
```

```
rtrim
-------
   TRIM
(1 row)
```

❖ **ltrim(string [, characters])**

描述：从字符串 string 的开头删除只包含 characters 中字符（缺省是一个空白）的最长的字符串。

返回值类型：text

示例：

```
panweidb=# SELECT ltrim('xxxxTRIM','x');
   ltrim
-------
   TRIM
(1 row)
```

❖ **upper(string)**

描述：把字符串转化为大写。

返回值类型：text

示例：

```
panweidb=# SELECT upper('tom');
   upper
-------
   TOM
(1 row)
```

❖ **lower(string)**

描述：把字符串转化为小写。

返回值类型：text

示例：

```
SELECT lower('TOM');
select lower('A0EEBC99-9C0B-4EF8-BB6D-6BB9BD380A11'::uuid);
SELECT lower(sys_guid()) ;
```

返回结果为：

```
 lower
```

```
-------
 tom
(1 row)

           lower
---------------------------------
a0eebc999c0b4ef8bb6d6bb9bd380a11
(1 row)

         lower
---------------------------------
a0eebc999c0b4ef8bb6d6bb9bd380a11
(1 row)

         lower
---------------------------------
00000000650ba5890000000000000002
(1 row)
```

❖ **rpad(string varchar, length int [, fill varchar])**

描述：使用填充字符 fill（缺省时为空白），把 string 填充到 length 长度。如果 string 已经比 length 长则将其从尾部截断。

length 参数在 panweidb 中表示字符长度。一个汉字长度计算为一个字符。

返回值类型：varchar

示例 1：

```
rpad('hi',5,'xyza');
```

返回结果为：

```
 rpad
-------
   hixyz
(1 row)
```

示例 2：

```
SELECT rpad('hi',5,'abcdefg');
```

返回结果为：

```
 rpad
-------
 hiabc
(1 row)
```

- **instr(string,substring[,position,occurrence])**

描述：从字符串 string 的 position（缺省时为 1）所指的位置开始查找并返回第 occurrence（缺省时为 1）次出现子串 substring 的位置的值。

- 当 position 为 0 时，返回 0。
- 当 position 为负数时，从字符串倒数第 n 个字符往前逆向搜索。n 为 position 的绝对值。

本函数以字符为计算单位，如一个汉字为一个字符。

返回值类型：integer

示例 1：

```
SELECT instr('corporate floor','or', 3);
```

返回结果为：

```
 instr
-------
     5
(1 row)
```

示例 2：

```
SELECT instr('corporate floor','or',-3,2);
```

返回结果为：

```
 instr
-------
  2
(1 row)
```

- **initcap(string)**

描述：将字符串中的每个单词的首字母转化为大写，其他字母转化为小写。

返回值类型：text

示例：

```
SELECT initcap('hi THOMAS');
```

返回结果为：

```
 initcap
-----------
 Hi Thomas
(1 row)
```

- **ascii(string)**

  描述：参数 string 的第一个字符的 ASCII 码。

  返回值类型：integer

  示例：

```
SELECT ascii('xyz');
```

  返回结果为：

```
 ascii
-------
  120
(1 row)
```

- **replace(string varchar, search_string varchar, replacement_string varchar)**

  描述：把字符串 string 中所有子字符串 search_string 替换成子字符串 replacement_string。

  返回值类型：varchar

  示例：

```
SELECT replace('jack and jue','j','bl');
```

  返回结果为：

```
   replace
----------------
 black and blue
(1 row)
```

- **lpad(string varchar, length int[, repeat_string varchar])**

  描述：在 string 的左侧添上一系列的 repeat_string（缺省为空白）来组成一个总长度为 n 的新字符串。

如果 string 本身的长度比指定的长度 length 长，则本函数将把 string 截断并把前面长度为 length 的字符串内容返回。

返回值类型：varchar

示例 1：

```
SELECT lpad('PAGE 1',15,'*.');
```

返回结果为：

```
    lpad
-----------------
*.*.*.*.*PAGE 1
(1 row)
```

示例 2：

```
SELECT lpad('hello world',5,'abcd');
```

返回结果为：

```
 lpad
-------
 hello
(1 row)
```

❖ **concat(str1,str2)**

描述：将字符串 str1 和 str2 连接并返回。

【须知】

数据库 SQL 兼容模式设置为 MY 的情况下，参数 str1 或 str2 为 NULL 会导致返回结果为 NULL。

返回值类型：varchar

示例 1：

```
SELECT concat('Hello', ' World!');
```

返回结果为：

```
    concat
--------------
  Hello World!
(1 row)
```

示例 2：

```
SELECT concat('Hello', NULL);
```

返回结果为：

```
 concat
--------
 Hello
(1 row)
```

- **chr(integer)**

  描述：给出 ASCII 码的字符。

  返回值类型：varchar

  示例：

```
panweidb=# SELECT chr(65);
 chr
-----
 A
(1 row)
```

- **regexp_substr(source_char, pattern)**

  描述：正则表达式的抽取子串函数。SQL 语法兼容 A 和 B 的情况下，GUC 参数 behavior_compat_options 的值包含 aformat_regexp_match 时，. 不能匹配 'n' 字符；不包含 aformat_regexp_match 时，. 能够匹配'n'字符。

  返回值类型：text

  示例：

```
panweidb=# SELECT regexp_substr('500 Hello World, Redwood Shores, CA', ',
[^,]+,') "REGEXPR_SUBSTR";
 REGEXPR_SUBSTR
------------------
, Redwood Shores,
(1 row)
```

- **regexp_replace(string, pattern, replacement [,flags])**

  描述：替换匹配 POSIX 正则表达式的子字符串。 如果没有匹配 pattern，那么返回不加修改的 string 串。 如果有匹配，则返回的 string 串里面的匹配子串将被 replacement 串替换掉。

replacement 串可以包含 n，其中 n 是 1 到 9，表明 string 串里匹配模式里第 n 个圆括号子表达式的子串应该被插入，并且它可以包含&表示应该插入匹配整个模式的子串。

可选的 flags 参数包含零个或多个改变函数行为的单字母标记。i 表示进行大小写无关的匹配，g 表示替换每一个匹配的子字符串而不仅仅是第一个。m 表示按照多行模式匹配。SQL 语法兼容 A 和 B 的情况下，n 选项在 GUC 参数 behavior_compat_options 的值包含 aformat_regexp_match 时，表示 . 能够匹配 'n' 字符，flags 中没有指定 n 时，默认.不能匹配 'n' 字符；值不包含 aformat_regexp_match 时，. 默认能匹配'n' 字符。n 选项的含义与 m 选项一致。

返回值类型：varchar

示例：

```
panweidb=# SELECT regexp_replace('Thomas', '.[mN]a.', 'M');
 regexp_replace
----------------
 ThM
(1 row)
panweidb=# SELECT regexp_replace('foobarbaz','b(..)', E'X1Y', 'g') AS RESULT;

 result
-------------
 fooXarYXazY
(1 row)
```

- **concat_ws(sep text, str"any" [, str"any" [, ...]])**

描述：以第一个参数为分隔符，链接第二个以后的所有参数。NULL 参数被忽略。

【须知】

如果第一个参数值是 NULL，会导致返回结果为 NULL。

如果第一个参数值是空字符串（''），且数据库 SQL 兼容模式设置为 A 的情况下，会导致返回结果为 NULL。这是因为 A 兼容模式>将''作为 NULL 处理，避免此类行为，可以将数据库 SQL 兼容模式改为 B、C 或者 P

G。

返回值类型：text

示例：

```
panweidb=# SELECT concat_ws(',', 'ABCDE', 2, NULL, 22);
 concat_ws
------------
 ABCDE,2,22
(1 row)
```

- **nlssort(string text, sort_method text)**

描述：以 sort_method 指定的排序方式返回字符串在该排序模式下的编码值，该编码值可用于排序，其决定了 string 在这种排序模式下的先后位置。目前支持的 sort_method 为'nls_sort=schinese_pinyin_m'和'nls_sort=generic_m_ci'。其中，'nls_sort=generic_m_ci'仅支持纯英文不区分大小写排序。

string 类型：text

sort_method 类型：text

返回值类型：text

示例：

```
panweidb=# SELECT nlssort('A', 'nls_sort=schinese_pinyin_m');
   nlssort
----------------
   01EA0000020006
(1 row)

panweidb=# SELECT nlssort('A', 'nls_sort=generic_m_ci');
    nlssort
----------------
   01EA000002
(1 row)
```

- **convert(string bytea, src_encoding name, dest_encoding name)**

描述：以 dest_encoding 指定的目标编码方式转化字符串 bytea。src_e

ncoding 指定源编码方式，在该编码下，string 必须是合法的。

返回值类型：bytea

示例：

```
panweidb=# SELECT convert('text_in_utf8', 'UTF8', 'GBK');
       convert
----------------------------
 x746578745f696e5f75746638
(1 row)
```

【说明】

如果源编码格式到目标编码格式的转化规则不存在，则字符串不进行任何转换直接返回，如 GBK 和 LATIN1 之间的转换规则是不存在的，具体转换规则可以通过查看系统表 pg_conversion 获得。

示例：

```
panweidb=# show server_encoding;
server_encoding
-----------------
LATIN1
(1 row)
panweidb=# SELECT convert_from('some text', 'GBK');
convert_from
--------------
some text
(1 row)
db_latin1=# SELECT convert_to('some text', 'GBK');
 convert_to
----------------------
x736f6d652074657874
(1 row)
db_latin1=# SELECT convert('some text', 'GBK', 'LATIN1');
```

```
 convert
----------------------
x736f6d652074657874
(1 row)
```

❖ **convert_from(string bytea, src_encoding name)**

描述：以数据库的编码方式转化字符串 bytea。

src_encoding 指定源编码方式，在该编码下，string 必须是合法的。

返回值类型：text

示例：

```
panweidb=# SELECT convert_from('text_in_utf8', 'UTF8');
   convert_from
--------------
   text_in_utf8
(1 row)
```

❖ **convert_to(string text, dest_encoding name)**

描述：将字符串转化为 dest_encoding 的编码格式。

返回值类型：bytea

示例：

```
panweidb=# SELECT convert_to('some text', 'UTF8');
      convert_to
----------------------
   x736f6d652074657874
(1 row)
```

❖ **string [NOT] LIKE pattern [ESCAPE escape-character]**

描述：模式匹配函数。

如果 pattern 不包含百分号或者下划线，该模式只代表它本身，这时候 LIKE 的行为就像等号操作符。在 pattern 里的下划线（_）匹配任何单个字符；而一个百分号（%）匹配零或多个任何字符。

要匹配下划线或者百分号本身，在 pattern 里相应的字符必须前导逃逸字符。缺省的逃逸字符是反斜杠，但是用户可以用 ESCAPE 子句指定一个。

要匹配逃逸字符本身，写两个逃逸字符。

返回值类型：Boolean

示例：

```
panweidb=# SELECT 'AA_BBCC' LIKE '%A@_B%' ESCAPE '@' AS RESULT;
result
--------
   t
(1 row)
```

```
panweidb=# SELECT 'AA_BBCC' LIKE '%A@_B%' AS RESULT;
result
--------
   f
(1 row)

panweidb=# SELECT 'AA@_BBCC' LIKE '%A@_B%' AS RESULT;
result
--------
   t
(1 row)
```

❖ **REGEXP_LIKE(source_string, pattern [, match_parameter])**

描述：正则表达式的模式匹配函数。

source_string 为源字符串，pattern 为正则表达式匹配模式。 match_parameter 为匹配选项，可取值为：

- 'i'：大小写不敏感。
- 'c'：大小写敏感。
- 'n'：允许正则表达式元字符“.”匹配换行符。
- 'm'：将 source_string 视为多行。

若忽略 match_parameter 选项，默认为大小写敏感，“.”不匹配换行符，source_string 视为单行。

返回值类型：Boolean

示例：

```
panweidb=# SELECT regexp_like('ABC', '[A-Z]');
 regexp_like
-------------
 t
(1 row)

panweidb=# SELECT regexp_like('ABC', '[D-Z]');
 regexp_like
-------------
 f
(1 row)

panweidb=# SELECT regexp_like('ABC', '[a-z]','i');
 regexp_like
-------------
 t
(1 row)
```

- **format(formatstr text [, str"any" [, ...]])**

  描述：格式化字符串。

  返回值类型：text

  示例：

```
panweidb=# SELECT format('Hello %s, %1$s', 'World');
      format
-------------------
 Hello World, World
(1 row)
```

- **md5(string)**

  描述：将 string 使用 MD5 加密，并以 16 进制数作为返回值。

【说明】
MD5 加密算法安全性低，存在安全风险，不建议使用。

  返回值类型：text

  示例：

```
panweidb=# SELECT md5('ABC');
               md5
----------------------------------
 902fbdd2b1df0c4f70b4a5d23525e932
(1 row)
```

- **decode(string text, format text)**

  描述：将二进制数据从文本数据中解码。

  返回值类型：bytea

  示例：

```
panweidb=# SELECT decode('MTIzAAE=', 'base64');
     decode
--------------
  x3132330001
(1 row)
```

- **similar_escape(pat text, esc text)**

  描述：将一个 SQL:2008 风格的正则表达式转换为 POSIX 风格。

  返回值类型：text

  示例：

```
panweidb=# select similar_escape('s+ab','2');
 similar_escape
----------------
 ^(?:s+ab)$
(1 row)
```

- **encode(data bytea, format text)**

  描述：将二进制数据编码为文本数据。

  返回值类型：text

  示例：

```
panweidb=# SELECT encode(E'123000001', 'base64');
    encode
----------
  MTIzAAE=
(1 row)
```

说明：

- 若字符串中存在换行符，如字符串由一个换行符和一个空格组成，在 panweidb 中 LENGTH 和 LENGTHB 的值为 2。
- 对于 CHAR(n) 类型，panweidb 中 n 是指字符个数。因此，对于多字节编码的字符集， LENGTHB 函数返回的长度可能大于 n。
- panweidb 支持多种类型的数据库，目前有 4 种，分别是 A 类型、B 类型、C 类型以及 PG 类型。在不指定数据库类型时，PanWeiDB 默认为 B 类型。

  Oracle 兼容模式的词法分析器与另外三种不一样，在 Oracle 兼容模式中，空字符串会被当作是 NULL。所以，当使用 Oracle 兼容模式的数据库时，假如上述字符操作函数中有空字符串作为参数，会出现没有输出的情况。例如：

```
panweidb=# SELECT translate('12345','123','');
translate
-----------
(1 row)
```

这是因为内核在调用相应的函数进行处理前，会判断所输入的参数中是否含有 NULL，假如有，则不会调用相应的函数，因此会没有输出。而在 PG 模式下，字符串的处理方式与 postgresql 保持一致，因此不会有上述问题产生。

4.1.4.9.系统管理函数

4.1.4.9.1. 配置设置函数

配置设置函数是可以用于查询以及修改运行时配置参数的函数。

❖ **current_setting(setting_name)**

描述：当前的设置值。

返回值类型：text

备注：current_setting 用于以查询形式获取 setting_name 的当前值。和 SQL 语句 SHOW 是等效的。比如：

```
panweidb=# SELECT current_setting('datestyle');
```

```
 current_setting
-----------------
 ISO, MDY
(1 row)
```

- **set_working_grand_version_num_manually(tmp_version)**

  描述：通过切换授权版本号来更新和升级数据库的新特性。

  返回值类型：void

- **shell_in(type)**

  描述： 为 shell 类型输入路由（那些尚未填充的类型）。

  返回值类型：void

- **shell_out(type)**

  描述：为 shell 类型输出路由（那些尚未填充的类型）。

  返回值类型：void

- **set_config(setting_name, new_value, is_local)**

  描述：设置参数并返回新值。

  返回值类型：text

  备注：set_config 将参数 setting_name 设置为 new_value。如果 is_local 为 true，则 new_value 将只应用于当前事务。如果希望 new_value 应用于当前会话，可以使用 false，和 SQL 语句 SET 是等效的。例如：

```
panweidb=# SELECT set_config('log_statement_stats', 'off', false);
 set_config
------------
 off
(1 row)
```

4.1.4.9.2. 通用文件访问函数

通用文件访问函数提供了对数据库服务器上的文件的本地访问接口。只有 PanWeiDB 目录和 log_directory 目录里面的文件可以访问。使用相对路径访

问 PanWeiDB 目录里面的文件，以及匹配 log_directory 配置而设置的路径访问日志文件。只有数据库初始化用户才能使用这些函数。

❖ **pg_ls_dir(dirname text)**

描述：列出目录中的文件。

返回值类型：setof text

备注：pg_ls_dir 返回指定目录里面的除了特殊项“.”和“..”之外所有名称。

示例：

```
panweidb=# SELECT pg_ls_dir('./');
    pg_ls_dir
----------------------
.postgresql.conf.swp
postgresql.conf
pg_tblspc
PG_VERSION
pg_ident.conf
core
server.crt
pg_serial
pg_twopcme
postgresql.conf.lock
pg_stat_tmp
pg_notify
pg_subtrans
pg_ctl.lock
pg_xlog
pg_clog
base
pg_snapshots
postmaster.opts
postmaster.pid
server.key.rand
server.key.cipher
```

```
pg_multixact
pg_errorinfo
server.key
pg_hba.conf
pg_replslot
.pg_hba.conf.swp
cacert.pem
pg_hba.conf.lock
global
gaussdb.state
(32 rows)
```

- **pg_read_file(filename text, offset bigint, length bigint)**

  描述：返回一个文本文件的内容。

  返回值类型：text

  备注：pg_read_file 返回一个文本文件的一部分，从 offset 开始，最多返回 length 字节（如果先达到文件结尾，则小于这个数值）。如果 offset 是负数，则它是相对于文件结尾回退的长度。如果省略了 offset 和 length，则返回整个文件。

  示例：

```
panweidb=# SELECT pg_read_file('postmaster.pid',0,100);
        pg_read_file
-------------------------------------
53078                   +
/srv/BigData/hadoop/data1/dbnode+
1500022474                  +
8000                    +
/var/run/FusionInsight          +
localhost                   +
 2
(1 row)
```

- **pg_read_binary_file(filename text [, offset bigint, length bigint,missing_ok boolean])**

  描述：返回一个二进制文件的内容。

返回值类型：bytea

备注：pg_read_binary_file 的功能与 pg_read_file 类似，除了结果的返回值为 bytea 类型不一致，相应地不会执行编码检查。与 convert_from 函数结合，这个函数可以用来读取用指定编码的一个文件。

```
panweidb=# SELECT convert_from(pg_read_binary_file('filename'), 'UTF8');
```

❖ **pg_stat_file(filename text)**

描述：返回一个文本文件的状态信息。

返回值类型：record

备注：pg_stat_file 返回一条记录，其中包含：文件大小、最后访问时间戳、最后更改时间戳、最后文件状态修改时间戳以及标识传入参数是否为目录的 Boolean 值。典型的用法：

```
panweidb=# SELECT * FROM pg_stat_file('filename');
panweidb=# SELECT (pg_stat_file('filename')).modification;
```

示例：

```
panweidb=# SELECT convert_from(pg_read_binary_file('postmaster.pid'), 'UTF
8');
        convert_from
-------------------------------------
 4881                    +
 /srv/BigData/gaussdb/data1/dbnode+
 1496308688                +
 25108                   +
 /opt/user/Bigdata/gaussdb/gaussdb_tmp +
 *                     +
   25108001  43352069          +

(1 row)
panweidb=# SELECT * FROM pg_stat_file('postmaster.pid');

 size |       access       |     modification     |       change
| creation | isdir
------+----------------------+----------------------+----------------------
```

```
+----------+-------
 117 | 2017-06-05 11:06:34+08 | 2017-06-01 17:18:08+08 | 2017-06-01 17:18:0
8+08
|       | f
(1 row)
panweidb=# SELECT (pg_stat_file('postmaster.pid')).modification;
    modification
------------------------
 2017-06-01 17:18:08+08
(1 row)
```

4.1.4.9.3. 服务器信号函数

服务器信号函数向其他服务器进程发送控制信号。只有系统管理员才能使用这些函数。

- **pg_cancel_backend(pid int)**

  描述：取消一个后端的当前查询。

  返回值类型：Boolean

  备注：pg_cancel_backend 向由 pid 标识的后端进程发送一个查询取消（SIGINT）信号。一个活动的后端进程的 PID 可以从 pg_stat_activity 视图的 pid 字段找到，或者在服务器上用 ps 列出数据库进程。具有 SYSADMIN 权限的用户，后端进程所连接的数据库的属主，后端进程的属主或者继承了内置角色 gs_role_signal_backend 权限的用户有权使用该函数。

- **pg_reload_conf()**

  描述：导致所有服务器进程重新装载它们的配置文件（需要系统管理员角色）。

  返回值类型：Boolean

  备注：pg_reload_conf 给服务器发送一个 SIGHUP 信号，导致所有服务器进程重新装载配置文件。

- **pg_rotate_logfile()**

描述：滚动服务器的日志文件（需要系统管理员角色）。

返回值类型：Boolean

备注：pg_rotate_logfile 给日志文件管理器发送信号，告诉它立即切换到一个新的输出文件。这个函数只有在 redirect_stderr 用于日志输出的时候才有用，否则根本不存在日志文件管理器子进程。

- **pg_terminate_backend(pid int)**

描述：终止一个后台线程。

返回值类型：Boolean

备注：如果成功，函数返回 true，否则返回 false。具有 SYSADMIN 权限的用户，后端进程所连接的数据库的属主，后端进程的属主或者继承了内置角色 gs_role_signal_backend 权限的用户有权使用该函数。

示例：

```
panweidb=# SELECT pid from pg_stat_activity;
      pid
 -----------------
  140657876268816
 (1 rows)

 panweidb=# SELECT pg_terminate_backend(140657876268816);
  pg_terminate_backend
 ----------------------
  t
 (1 row)
```

- **pg_terminate_session(pid int64, sessionid int64)**

描述：终止一个后台 session。

返回值类型：Boolean

备注：如果成功，函数返回 true，否则返回 false。具有 SYSADMIN 权限的用户，会话所连接的数据库的属主，会话的属主或者继承了内置角色 gs_role_signal_backend 权限的用户有权使用该函数。

- **pg_terminate_session_socket(pid int64, sessionid int64)**

描述：关闭一个活跃 session 和客户端的 socket 连接。

返回值类型：Boolean

备注：如果成功，函数返回 true，否则返回 false。仅限初始化用户才可使用函数。

4.1.4.9.4. 备份恢复控制函数

备份控制函数

备份控制函数可帮助进行在线备份。

- **pg_create_restore_point(name text)**

  描述：为执行恢复创建一个命名点。（需要管理员角色）

  返回值类型：text

  备注：pg_create_restore_point 创建了一个可以用作恢复目的、有命名的事务日志记录，并返回相应的事务日志位置。在恢复过程中，recovery_target_name 可以通过这个名称定位对应的日志恢复点，并从此处开始执行恢复操作。避免使用相同的名称创建多个恢复点，因为恢复操作将在第一个匹配（恢复目标）的名称上停止。

- **pg_current_xlog_location()**

  描述：获取当前事务日志的写入位置。

  返回值类型：text

  备注：pg_current_xlog_location 使用与前面那些函数相同的格式显示当前事务日志的写入位置。如果是只读操作，不需要系统管理员权限。

- **pg_current_xlog_insert_location()**

  描述：获取当前事务日志的插入位置。

  返回值类型：text

  备注：pg_current_xlog_insert_location 显示当前事务日志的插入位置。插入点是事务日志在某个瞬间的“逻辑终点”，而实际的写入位置则是从服务器内部缓冲区写出时的终点。写入位置是可以从服务器外部检测到的终点，如果要归档部分完成事务日志文件，则该操作即可实现。插入点主要

用于服务器调试目的。如果是只读操作，不需要系统管理员权限。

- **gs_current_xlog_insert_end_location()**

  描述：获取当前事务日志的插入位置。

  返回值类型：text

  备注：gs_current_xlog_insert_end_location 显示当前事务日志的实际插入位置。

- **pg_start_backup(label text [, fast boolean])**

  描述：开始执行在线备份。（需要管理员角色或复制的角色）

  返回值类型：text

  备注：pg_start_backup 接受一个用户定义的备份标签（通常这是备份转储文件存放地点的名称）。这个函数向 PanWeiDB 的数据目录写入一个备份标签文件，然后以文本方式返回备份的事务日志起始位置。

```
panweidb=# SELECT pg_start_backup('label_goes_here');
 pg_start_backup
-----------------
 0/3000020
(1 row)
```

- **pg_stop_backup()**

  描述：完成执行在线备份。（需要管理员角色或复制的角色）

  返回值类型：text

  备注：pg_stop_backup 删除 pg_start_backup 创建的标签文件，并且在事务日志归档区里创建一个备份历史文件。这个历史文件包含给予 pg_start_backup 的标签、备份的事务日志起始与终止位置、备份的起始和终止时间。返回值是备份的事务日志终止位置。计算出中止位置后，当前事务日志的插入点将自动前进到下一个事务日志文件，这样，结束的事务日志文件可以被立即归档从而完成备份。

- **pg_switch_xlog()**

  描述：切换到一个新的事务日志文件。（需要管理员角色）

  返回值类型：text

备注：pg_switch_xlog 移动到下一个事务日志文件，以允许将当前日志文件归档（假定使用连续归档）。返回值是刚完成的事务日志文件的事务日志结束位置+1。如果从最后一次事务日志切换以来没有活动的事务日志，则 pg_switch_xlog 什么事也不做，直接返回当前事务日志文件的开始位置。

- **pg_xlogfile_name(location text)**

  描述：将事务日志的位置字符串转换为文件名。

  返回值类型：text

  备注：pg_xlogfile_name 仅抽取事务日志文件名称。如果给定的事务日志位置恰好位于事务日志文件的交界上，这两个函数都返回前一个事务日志文件的名称。这对于管理事务日志归档来说是非常有利的，因为前一个文件是当前最后一个需要归档的文件。

- **pg_xlogfile_name_offset(location text)**

  描述：将事务日志的位置字符串转换为文件名并返回在文件中的字节偏移量。

  返回值类型：text,integer

  备注：可以使用 pg_xlogfile_name_offset 从前述函数的返回结果中抽取相应的事务日志文件名称和字节偏移量。例如：

```
panweidb=# SELECT * FROM pg_xlogfile_name_offset(pg_stop_backup());
NOTICE:  pg_stop_backup cleanup done, waiting for required WAL segments t
o be archived
NOTICE:  pg_stop_backup complete, all required WAL segments have been arc
hived
     file_name       | file_offset
-------------------------+-------------
000000010000000000000003 |       272
(1 row)
```

- **pg_xlog_location_diff(location text, location text)**

  描述：计算两个事务日志位置之间在字节上的区别。

  返回值类型：numeric

- **pg_cbm_tracked_location()**

  描述：用于查询 cbm 解析到的 lsn 位置。

  返回值类型：text

- **pg_cbm_get_merged_file(startLSNArg text, endLSNArg text)**

  描述：用于将指定 lsn 范围之内的 cbm 文件合并成一个 cbm 文件，并返回合并完的 cbm 文件名。

  返回值类型：text

  备注：必须是系统管理员或运维管理员才能获取 cbm 合并文件。

- **pg_cbm_get_changed_block(startLSNArg text, endLSNArg text)**

  描述：用于将指定 lsn 范围之内的 cbm 文件合并成一个表，并返回表的各行记录。

  返回值类型：records

  备注：pg_cbm_get_changed_block 返回的表字段包含：合并起始的 lsn、合并截止的 lsn、表空间 oid、库 oid、表的 relfilenode、表的 fork number、表是否被删除、表是否被创建、表是否被截断、表被截断后的页面数、有多少页被修改以及被修改的页号的列表。

- **pg_cbm_recycle_file(targetLSNArg text)**

  描述：删除 targetLSNArg 之前的所有 cbm 文件内容，并返回删除后的第一条 lsn。

  返回值类型：text

- **pg_cbm_force_track(targetLSNArg text,timeOut int)**

  描述：强制执行一次 cbm 追踪到指定的 xlog 位置，并返回实际追踪结束点的 xlog 位置。

  返回值类型：text

- **pg_enable_delay_ddl_recycle()**

  描述：开启延迟 DDL 功能，并返回开启点的 xlog 位置。需要管理员角色或运维管理员角色打开 operate_mode。

  返回值类型：text

- **pg_disable_delay_ddl_recycle(barrierLSNArg text, isForce bool)**

  描述：关闭延迟 DDL 功能，并返回本次延迟 DDL 生效的 xlog 范围。需要管理员角色或运维管理员角色打开 operate_mode。

  返回值类型：records

- **pg_enable_delay_xlog_recycle()**

  描述：开启延迟 xlog 回收功能，数据库主节点修复使用。

  返回值类型：void

- **pg_disable_delay_xlog_recycle()**

  描述：关闭延迟 xlog 回收功能，数据库主节点修复使用。

  返回值类型：void

- **pg_cbm_rotate_file(rotate_lsn text)**

  描述：等待 cbm 解析到 rotate_lsn 之后，强制切换文件，在 build 期间调用。

  返回值类型：void。

- **gs_roach_stop_backup(backupid text)**

  描述：停止一个内部备份工具 GaussRoach 开启的备份。与 pg_stop_backup 系统函数类似，但更轻量。

  返回值类型：text，内容为当前日志的插入位置。

  备注：目前 PanWeiDB 不支持。

- **gs_roach_enable_delay_ddl_recycle(backupid name)**

  描述：开启延迟 DDL 功能，并返回开启点的日志位置。与 pg_enable_delay_ddl_recycle 系统函数类似，但更轻量。并且，通过传入不同的 backupid，可以支持并发打开延迟 DDL。

  返回值类型：text，内容为返回开启点的日志位置。

  备注：目前 PanWeiDB 不支持。

- **gs_roach_disable_delay_ddl_recycle(backupid text)**

  描述：关闭延迟 DDL 功能，并返回本次延迟 DDL 生效的日志范围，并删除该范围内被用户删除的列存表物理文件。与 pg_enable_delay_ddl_

recycle 系统函数类似，但更轻量。并且，通过传入不同的 backupid，可以支持并发关闭延迟 DDL 功能。

返回值类型：records，内容为本次延迟 DDL 生效的日志范围。

备注：目前 PanWeiDB 不支持。

- **gs_roach_switch_xlog(request_ckpt bool)**

  描述：切换当前使用的日志段文件，并且，如果 request_ckpt 为 true，则触发一个全量检查点。

  返回值类型：text，内容为切段日志的位置。

  备注：目前 PanWeiDB 不支持。

恢复控制函数

恢复信息函数提供了当前备机状态的信息。这些函数可能在恢复期间或正常运行中执行。

- **pg_is_in_recovery()**

  描述：如果恢复仍然在进行中则返回 true。

  返回值类型：bool

- **pg_last_xlog_receive_location()**

  描述：获取最后接收事务日志的位置并通过流复制将其同步到磁盘。当流复制正在进行时，事务日志将持续递增。如果恢复已完成，则最后一次获取的 WAL 记录会被静态保持并在恢复过程中同步到磁盘。如果流复制不可用，或还没有开始，这个函数返回 NULL。

  返回值类型：text

- **pg_last_xlog_replay_location()**

  描述：获取最后一个事务日志在恢复时重放的位置。如果恢复仍在进行，事务日志将持续递增。如果已经完成恢复，则将保持在恢复期间最后接收 WAL 记录的值。如果未进行恢复但服务器正常启动时，则这个函数返回 NULL。

  返回值类型：text

- **pg_last_xact_replay_timestamp()**

描述：获取最后一个事务在恢复时重放的时间戳。这是为在主节点上生成事务提交或终止 WAL 记录的时间。如果在恢复时没有事务重放，则这个函数返回 NULL。如果恢复仍在进行，则事务日志将持续递增。如果恢复已经完成，则将保持在恢复期间最后接收 WAL 记录的值。如果服务器无需恢复就已正常启动，则这个函数返回 NULL。

返回值类型：timestamp with time zone

恢复控制函数控制恢复的进程。这些函数可能只在恢复时被执行。

- **pg_is_xlog_replay_paused()**

  描述：如果恢复暂停则返回 true。

  返回值类型：bool

- **pg_xlog_replay_pause()**

  描述：立即暂停恢复。

  返回值类型：void

- **pg_xlog_replay_resume()**

  描述：如果恢复处于暂停状态，则重新启动。

  返回值类型：void

当恢复暂停时，没有发生数据库更改。如果是在热备里，所有新的查询将看到一致的数据库快照，并且不会有进一步的查询冲突产生，直到恢复继续。

如果不能使用流复制，则暂停状态将无限的延续。当流复制正在进行时，将连续接收 WAL 记录，最终将填满可用磁盘空间，这个进度取决于暂停的持续时间，WAL 生成的速度和可用的磁盘空间。

4.1.4.9.5. 快照同步函数

快照同步函数是导出当前快照的标识符。

- **pg_export_snapshot()**

  描述：保存当前的快照并返回它的标识符。

  返回值类型：text

  备注：函数 pg_export_snapshot 保存当前的快照并返回一个文本字符

串标识此快照。这个字符串必须传递给想要导入快照的客户端。可用在 set transaction snapshot snapshot_id 时导入 snapshot，但是应用的前提是该事务设置了 SERIALIZABLE 或 REPEATABLE READ 隔离级别。而 PanWeiDB 目前是不支持这两种隔离级别的。该函数的输出不可用做 set transaction snapshot 的输入。

- **pg_export_snapshot_and_csn()**

  描述：保存当前的快照并返回它的标识符。比 pg_export_snapshot()多返回一列 CSN，表示当前快照的 CSN。

  返回值类型：text

4.1.4.9.6. 数据库对象函数

数据库对象尺寸函数

数据库对象尺寸函数计算数据库对象使用的实际磁盘空间。

- **pg_column_size(any)**

  描述：存储一个指定的数值需要的字节数（可能压缩过）。

  返回值类型：int

  备注：pg_column_size 显示用于存储某个独立数据值的空间。

```
panweidb=# SELECT pg_column_size(1);
 pg_column_size
----------------
              4
(1 row)
```

- **pg_database_size(oid)**

  描述：指定 OID 代表的数据库使用的磁盘空间。

  返回值类型：bigint

- **pg_database_size(name)**

  描述：指定名称的数据库使用的磁盘空间。

  返回值类型：bigint

  备注：pg_database_size 接受一个数据库的 OID 或者名称，然后返回

该对象使用的全部磁盘空间。

示例：

```
panweidb=# SELECT pg_database_size('postgres');
 pg_database_size
------------------
     51590112
(1 row)
```

- **pg_relation_size(oid)**

  描述：指定 OID 代表的表或者索引所使用的磁盘空间。

  返回值类型：bigint

- **get_db_source_datasize()**

  描述：估算当前数据库非压缩态的数据总容量。

  返回值类型：bigint

  备注：（1）调用该函数前需要做 analyze；（2）通过估算列存的压缩率计算非压缩态的数据总容量。

  示例：

```
panweidb=# analyze;
ANALYZE
panweidb=# select get_db_source_datasize();
 get_db_source_datasize
------------------------
       35384925667
(1 row)
```

- **pg_relation_size(text)**

  描述：指定名称的表或者索引使用的磁盘空间。表名称可以用模式名修饰。

  返回值类型：bigint

- **pg_relation_size(relation regclass, fork text)**

  描述：指定表或索引的指定分叉树（'main'，'fsm'或'vm'）使用的磁盘空间。

  返回值类型：bigint

- **pg_relation_size(relation regclass)**

描述：pg_relation_size(..., 'main')的简写。

返回值类型：bigint

备注：pg_relation_size 接受一个表、索引、压缩表的 OID 或者名称，然后返回它们的字节大小。

- **pg_partition_size(oid,oid)**

  描述：指定 OID 代表的分区使用的磁盘空间。其中，第一个 oid 为表的 OID，第二个 oid 为分区的 OID。

  返回值类型：bigint

- **pg_partition_size(text, text)**

  描述：指定名称的分区使用的磁盘空间。其中，第一个 text 为表名，第二个 text 为分区名。

  返回值类型：bigint

- **pg_partition_indexes_size(oid,oid)**

  描述：指定 OID 代表的分区的索引使用的磁盘空间。其中，第一个 oid 为表的 OID，第二个 oid 为分区的 OID。

  返回值类型：bigint

- **pg_partitio n_indexes_size(text,text)**

  描述：指定名称的分区的索引使用的磁盘空间。其中，第一个 text 为表名，第二个 text 为分区名。

  返回值类型：bigint

- **pg_indexes_size(regclass)**

  描述：附加到指定表的索引使用的总磁盘空间。

  返回值类型：bigint

- **pg_size_pretty(bigint)**

  描述：将以 64 位整数表示的字节值转换为具有单位的易读格式。

  返回值类型：text

- **pg_size_pretty(numeric)**

  描述：将以数值表示的字节值转换为具有单位的易读格式。

返回值类型：text

备注：pg_size_pretty 用于把其他函数的结果格式化成一种易读的格式，可以根据情况使用 KB 、MB 、GB 、TB。

- **pg_table_size(regclass)**

  描述：指定的表使用的磁盘空间，不计索引（但是包含 TOAST，自由空间映射和可见性映射）。

  返回值类型：bigint

- **pg_tablespace_size(oid)**

  描述：指定 OID 代表的表空间使用的磁盘空间。

  返回值类型：bigint

- **pg_tablespace_size(name)**

  描述：指定名称的表空间使用的磁盘空间。

  返回值类型：bigint

  备注：

  pg_tablespace_size 接受一个数据库的 OID 或者名称，然后返回该对象使用的全部磁盘空间。

- **pg_total_relation_size(oid)**

  描述：指定 OID 代表的表使用的磁盘空间，包括索引和压缩数据。

  返回值类型：bigint

- **pg_total_relation_size(regclass)**

  描述：指定的表使用的总磁盘空间，包括所有的索引和 TOAST 数据。

  返回值类型：bigint

- **pg_total_relation_size(text)**

  描述：指定名称的表所使用的全部磁盘空间，包括索引和压缩数据。表名称可以用模式名修饰。

  返回值类型：bigint

  备注：pg_total_relation_size 接受一个表或者一个压缩表的 OID 或者名称，然后返回以字节计的数据和所有相关的索引和压缩表的尺寸。

❖ datalength(any)

描述：计算一个指定的数据需要的字节数（不考虑数据的管理空间和数据压缩、数据类型转换等情况）。

返回值类型：int

备注：datalength 用于计算某个独立数据值的空间。

示例：

```
panweidb=# SELECT datalength(1);
datalength
------------
4
(1 row)
```

目前支持的数据类型及计算方式见下表：

数据类型			存储空间
数值类型	整数类型	TINYINT	1
		SMALLINT	2
		INTEGER	4
		BINARY_INTEGER	4
		BIGINT	8
	任意精度型	DECIMAL	每 4 位十进制数占两个字节，小数点前后数字分别计算
		NUMERIC	每 4 位十进制数占两个字节，小数点前后数字分别计算
		NUMBER	每 4 位十进制数占两个字节，小数点前后数字分别计算
	序列	SMALLSERIAL	2

数据类型			存储空间
	整型	SERIAL	4
		LARGESERIAL	8
		BIGSERIAL	每 4 位十进制数占两个字节，小数点前后数字分别计算
	浮点类型	FLOAT4	4
		DOUBLE PRECISION	8
		FLOAT8	8
		BINARY_DOUBLE	8
		FLOAT[(p)]	每 4 位十进制数占两个字节，小数点前后数字分别计算
		DEC[(p[,s])]	每 4 位十进制数占两个字节，小数点前后数字分别计算
		INTEGER[(p[,s])]	每 4 位十进制数占两个字节，小数点前后数字分别计算
布尔类型	布尔类型	BOOLEAN	1
字符类型	字符类型	CHAR	n
		CHAR(n)	n
		CHARACTER(n)	n
		NCHAR(n)	n
		VARCHAR(n)	n

数据类型			存储空间
		CHARACTER	字符实际字节数
		VARYING(n)	字符实际字节数
		VARCHAR2(n)	字符实际字节数
		NVARCHAR(n)	字符实 际字节数
		NVARCHAR2(n)	字符实际字节数
		TEXT	字符实际字节数
		CLOB	字符实际字节数
时间类型	时间类型	DATE	8
		TIME	8
		TIMEZ	12
		TIMESTAMP	8
		TIMESTAMPZ	8
		SMALLDATETIME	8
		INTERVAL DAY TO SECOND	16
		INTERVAL	16
		RELTIME	4
		ABSTIME	4
		TINTERVAL	12

数据库对象位置函数

❖ **pg_relation_filenode(relation regclass)**

描述：指定关系的文件节点数。

返回值类型：oid

备注：pg_relation_filenode 接受一个表、索引、序列或压缩表的 OID 或者名称，并且返回当前分配给它的 “filenode” 数。文件节点是关系使用的文件名称的基本组件。对大多数表来说，结果和 pg_class.relfilenode 相同，但对确定的系统目录来说，relfilenode 为 0 而且这个函数必须用来获取正确的值。如果传递一个没有存储的关系，比如一个视图，那么这个函数返回 NULL。

- **pg_relation_filepath(relation regclass)**

  描述：指定关系的文件路径名。

  返回值类型：text

  备注：pg_relation_filepath 类似于 pg_relation_filenode，但是它返回关系的整个文件路径名（相对于 panweidb 的数据目录 PGDATA）。当查询关系为段页式表时，本函数返回结果为关系的逻辑地址，并非实际存储该关系的文件路径。

- **pg_filenode_relation(tablespace oid, filenode oid)**

  描述：获取对应的 tablespace 和 relfilenode 所对应的表名。

  返回类型：regclass

- **pg_partition_filenode(partition_oid)**

  描述：获取到指定分区表的 oid 锁对应的 filenode。

  返回类型：oid

- **pg_partition_filepath(partition_oid)**

  描述：指定分区的文件路径名。

  返回值类型：text

回收站对象函数

- **gs_is_recycle_object(classid, objid, objname)**

  描述：判断是否为回收站对象。

  返回值类型：bool

4.1.4.9.7. 咨询锁函数

咨询锁函数用于管理咨询锁（Advisory Lock）。

- **pg_advisory_lock(key bigint)**

  描述：获取会话级别的排它咨询锁。

  返回值类型：void

  备注：pg_advisory_lock 锁定应用程序定义的资源，该资源可以用一个 64 位或两个不重叠的 32 位键值标识。如果已经有另外的会话锁定了该资源，则该函数将阻塞到该资源可用为止。这个锁是排它的。多个锁定请求将会被压入栈中，因此，如果同一个资源被锁定了三次，它必须被解锁三次以将资源释放给其他会话使用。

- **pg_advisory_lock(key1 int, key2 int)**

  描述：获取会话级别的排它咨询锁。

  返回值类型：void

  备注：只允许 sysadmin 对键值对(65535, 65535)加会话级别的排它咨询锁，普通用户无权限。

- **pg_advisory_lock(int4, int4, Name)**

  描述：获取指定数据库的排它咨询锁。

  返回值类型：void

- **pg_advisory_lock_shared(key bigint)**

  描述：获取会话级别的共享咨询锁。

  返回值类型：void

- **pg_advisory_lock_shared(key1 int, key2 int)**

  描述：获取会话级别的共享咨询锁。

  返回值类型：void

  备注：pg_advisory_lock_shared 类似于 pg_advisory_lock，不同之处仅在于共享锁会话可以和其他请求共享锁的会话共享资源，但排它锁除外。

- **pg_advisory_unlock(key bigint)**

  描述：释放会话级别的排它咨询锁。

返回值类型：Boolean

- **pg_advisory_unlock(key1 int, key2 int)**

  描述：释放会话级别的排它咨询锁。

  返回值类型：Boolean

  备注：pg_advisory_unlock 释放先前取得的排它咨询锁。如果释放成功则返回 true。如果实际上并未持有指定的锁，将返回 false 并在服务器中产生一条 SQL 警告信息。

- **pg_advisory_unlock(int4, int4, Name)**

  描述：释放指定数据库上的排它咨询锁。

  返回值类型：Boolean

  备注：如果释放成功则返回 true；如果未持有锁，则返回 false。

- **pg_advisory_unlock_shared(key bigint)**

  描述：释放会话级别的共享咨询锁。

  返回值类型：Boolean

- **pg_advisory_unlock_shared(key1 int, key2 int)**

  描述：释放会话级别的共享咨询锁。

  返回值类型：Boolean

  备注：pg_advisory_unlock_shared 类似于 pg_advisory_unlock，不同之处在于该函数释放的是共享咨询锁。

- **pg_advisory_unlock_all()**

  描述：释放当前会话持有的所有咨询锁。

  返回值类型：void

  备注：pg_advisory_unlock_all 将会释放当前会话持有的所有咨询锁，该函数在会话结束的时候被隐含调用，即使客户端异常地断开连接也是一样。

- **pg_advisory_xact_lock(key bigint)**

  描述：获取事务级别的排它咨询锁。

  返回值类型：void

- **pg_advisory_xact_lock(key1 int, key2 int)**

描述：获取事务级别的排它咨询锁。

返回值类型：void

备注：pg_advisory_xact_lock 类似于 pg_advisory_lock，不同之处在于锁是自动在当前事务结束时释放，而且不能被显式的释放。只允许 sysadmin 对键值对(65535, 65535)加事务级别的排它咨询锁，普通用户无权限。

- **pg_advisory_xact_lock_shared(key bigint)**

  描述：获取事务级别的共享咨询锁。

  返回值类型：void

- **pg_advisory_xact_lock_shared(key1 int, key2 int)**

  描述：获取事务级别的共享咨询锁。

  返回值类型：void

  备注：pg_advisory_xact_lock_shared 类似于 pg_advisory_lock_shared，不同之处在于锁是在当前事务结束时自动释放，而且不能被显式的释放。

- **pg_try_advisory_lock(key bigint)**

  描述：尝试获取会话级排它咨询锁。

  返回值类型：Boolean

  备注：pg_try_advisory_lock 类似于 pg_advisory_lock，不同之处在于该函数不会阻塞以等待资源的释放。它要么立即获得锁并返回 true，要么返回 false 表示目前不能锁定。

- **pg_try_advisory_lock(key1 int, key2 int)**

  描述：尝试获取会话级排它咨询锁。

  返回值类型：Boolean

  备注：只允许 sysadmin 对键值对(65535, 65535)加会话级别的排它咨询锁，普通用户无权限。

- **pg_try_advisory_lock_shared(key bigint)**

  描述：尝试获取会话级共享咨询锁。

返回值类型：Boolean

- **pg_try_advisory_lock_shared(key1 int, key2 int)**

  描述：尝试获取会话级共享咨询锁。

  返回值类型：Boolean

  备注：pg_try_advisory_lock_shared 类似于 pg_try_advisory_lock，不同之处在于该函数尝试获得共享锁而不是排它锁。

- **pg_try_advisory_xact_lock(key bigint)**

  描述：尝试获取事务级别的排它咨询锁。

  返回值类型：Boolean

- **pg_try_advisory_xact_lock(key1 int, key2 int)**

  描述：尝试获取事务级别的排它咨询锁。

  返回值类型：Boolean

  备注：pg_try_advisory_xact_lock 类似于 pg_try_advisory_lock，不同之处在于如果得到锁，在当前事务的结束时自动释放，而且不能被显式的释放。只允许 sysadmin 对键值对(65535, 65535)加事务级别的排它咨询锁，普通用户无权限。

- **pg_try_advisory_xact_lock_shared(key bigint)**

  描述：尝试获取事务级别的共享咨询锁。

  返回值类型：Boolean

- **pg_try_advisory_xact_lock_shared(key1 int, key2 int)**

  描述：尝试获取事务级别的共享咨询锁。

  返回值类型：Boolean

  备注：pg_try_advisory_xact_lock_shared 类似于 pg_try_advisory_lock_shared，不同之处在于如果得到锁，在当前事务结束时自动释放，而且不能被显式的释放。

- **lock_cluster_ddl()**

  描述：尝试对 PanWeiDB 内所有存活的数据库主节点获取会话级别的排他咨询锁。

返回值类型：Boolean

备注：只允许 sysadmin 调用，普通用户无权限。

- **unlock_cluster_ddl()**

  描述：尝试对数据库主节点会话级别的排他咨询锁。

  返回值类型：Boolean

4.1.4.9.8. 逻辑复制函数

- **pg_create_logical_replication_slot('slot_name', 'plugin_name')**

  描述：创建逻辑复制槽。

  参数说明：

  - slot_name

    流复制槽名称。

    取值范围：字符串，不支持除小写字母，数字，以及（_?-.）以外的字符。

  - plugin_name

    插件名称。

    取值范围：字符串，当前支持 mppdb_decoding。

  返回值类型：name, text

  备注：第一个返回值表示 slot_name，第二个返回值表示该逻辑复制槽解码的起始 LSN 位置。调用该函数的用户需要具有 SYSADMIN 权限或具有 REPLICATION 权限或继承了内置角色 gs_role_replication 的权限。此函数目前只支持在主机调用。

- **pg_create_physical_replication_slot('slot_name', 'isDummyStandby')**

  描述：创建新的物理复制槽。

  参数说明：

  - slot_name

    流复制槽名称。

    取值范围：字符串，不支持除字母，数字，以及（_?-.）以外的字符。

  - isDummyStandby

    是否是从从备连接主机创建的复制槽。

    类型：bool。

  返回值类型：name, text

备注：调用该函数的用户需要具有 SYSADMIN 权限或具有 REPLICATION 权限或继承了内置角色 gs_role_replication 的权限。目前默认不支持主备从部署模式。

- **pg_drop_replication_slot('slot_name')**

  描述：删除流复制槽。

  参数说明：

  - slot_name

    流复制槽名称。

    取值范围：字符串，不支持除字母，数字，以及（_?-.）以外的字符。

  返回值类型：void

  备注：调用该函数的用户需要具有 SYSADMIN 权限或具有 REPLICATION 权限或继承了内置角色 gs_role_replication 的权限。此函数目前只支持在主机调用。

- **pg_logical_slot_peek_changes('slot_name', 'LSN', upto_nchanges, 'options_name', 'options_value')**

  描述：解码并不推进流复制槽（下次解码可以再次获取本次解出的数据）。

  - slot_name

  流复制槽名称。

  取值范围：字符串，不支持除字母，数字，以及（_?-.）以外的字符。

  - LSN

    日志的 LSN，表示只解码小于等于此 LSN 的日志。

    取值范围：字符串（LSN，格式为 xlogid/xrecoff），如 '1/2AAFC60'。为 NULL 时表示不对解码截止的日志位置做限制。

  - upto_nchanges

    解码条数（包含 begin 和 commit）。假设一共有三条事务，分别包含 3、5、7 条记录，如果 upto_nchanges 为 4，那么会解码出 前两个事务共 8 条记录。解码完第二条事务时发现解码条数记录 大于等于 upto_nchanges，会停止解码。

    取值范围：非负整数。

【说明】

LSN 和 upto_nchanges 中任一参数达到限制，解码都会结束。

- options：此项为可选参数，由一系列 options_name 和 options_value 一一对应组成。
  - ✧ include-xids

    解码出的 data 列是否包含 xid 信息。

    取值范围：0 或 1，默认值为 1。

    ✓ 0：设为 0 时，解码出的 data 列不包含 xid 信息。

    ✓ -1：设为 1 时，解码出的 data 列包含 xid 信息。
  - ✧ skip-empty-xacts

    解码时是否忽略空事务信息。

    取值范围：0 或 1，默认值为 0。

    0：设为 0 时，解码时不忽略空事务信息。

    1：设为 1 时，解码时会忽略空事务信息。
  - ✧ include-timestamp

    解码信息是否包含 commit 时间戳。

    取值范围：0 或 1，默认值为 0。

    0：设为 0 时，解码信息不包含 commit 时间戳。

    1：设为 1 时，解码信息包含 commit 时间戳。
  - ✧ only-local

    是否仅解码本地日志。

    取值范围：0 或 1，默认值为 1。

    0：设为 0 时，解码非本地日志和本地日志。

    1：设为 1 时，仅解码本地日志。
  - ✧ force-binary

    是否以二进制格式输出解码结果。

    取值范围：0，默认值为 0。设为 0 时，以文本格式输出解码结果。
  - ✧ white-table-list

    白名单参数，包含需要进行解码的 schema 和表名。

    取值范围：包含白名单中表名的字符串，不同的表以','为分隔符进行隔离；使用'*'来模糊匹配所有情况；schema 名和表名间以'.'分

割，不允许存在任意空白符。

例如： select * from pg_logical_slot_peek_changes('slot1', NULL, 4096, 'white-table-list', 'public.t1,public.t2');

- ✧ max-txn-in-memory

  内存管控参数，单位为 MB，单个事务占用内存大于该值即进行落盘。

  取值范围： 0\~100 的整型，默认值为 0，即不开启此种管控。

- ✧ max-reorderbuffer-in-memory

  内存管控参数，单位为 GB，拼接-发送线程中正在拼接的事务总内存（包含缓存）大于该值则对当前解码事务进行落盘。

  取值范围： 0\~100 的整型，默认值为 0，即不开启此种管控。

返回值类型： text, xid, text

备注： 函数返回解码结果，每一条解码结果包含三列，对应上述返回值类型，分别表示 LSN 位置、xid 和解码内容。

调用该函数的用户需要具有 SYSADMIN 权限或具有 REPLICATION 权限或继承了内置角色 gs_role_replication 的权限。

- ❖ **pg_logical_slot_get_changes('slot_name', 'LSN', upto_nchanges, 'options_name', 'options_value')**

  描述： 解码并推进流复制槽。

  参数说明： 与 pg_logical_slot_peek_changes 一致，详细内容请参见 pg_logical_slot_peek_ch...。

  备注： 调用该函数的用户需要具有 SYSADMIN 权限或具有 REPLICATION 权限或继承了内置角色 gs_role_replication 的权限。此函数目前只支持在主机调用。

- ❖ **pg_logical_slot_peek_binary_changes('slot_name', 'LSN', upto_nchanges, 'options_name', 'options_value')**

  描述： 以二进制格解码且不推进流复制槽（下次解码可以再次获取本次解出的数据）。

  参数说明：

  - ➢ slot_name

    流复制槽名称。

取值范围：字符串，不支持除字母，数字，以及（_?-.）以外的字符。

- LSN

  日志的 LSN，表示只解码小于等于此 LSN 的日志。

  取值范围：字符串（LSN，格式为 xlogid/xrecoff），如 '1/2AAFC60'。为 NULL 时表示不对解码截止的日志位置做限制。

- upto_nchanges

  解码条数（包含 begin 和 commit）。假设一共有三条事务，分别 包含 3、5、7 条记录，如果 upto_nchanges 为 4，那么会解码出 前两个事务共 8 条记录。解码完第二条事务时发现解码条数记录大于等于 upto_nchanges，会停止解码。

  取值范围：非负整数。

【说明】

LSN 和 upto_nchanges 中任一参数达到限制，解码都会结束。

- options：此项为可选参数，由一系列 options_name 和 options_value 一一对应组成。
  - include-xids

    解码出的 data 列是否包含 xid 信息。

    取值范围：0 或 1，默认值为 1。

    0：设为 0 时，解码出的 data 列不包含 xid 信息。

    1：设为 1 时，解码出的 data 列包含 xid 信息。
  - skip-empty-xacts

    解码时是否忽略空事务信息。

    取值范围：0 或 1，默认值为 0。

    0：设为 0 时，解码时不忽略空事务信息。

    1：设为 1 时，解码时会忽略空事务信息。
  - include-timestamp

    解码信息是否包含 commit 时间戳。

    取值范围：0 或 1，默认值为 0。

    0：设为 0 时，解码信息不包含 commit 时间戳。

    1：设为 1 时，解码信息包含 commit 时间戳。
  - only-local

是否仅解码本地日志。

取值范围：0 或 1，默认值为 1。

0：设为 0 时，解码非本地日志和本地日志。

1：设为 1 时，仅解码本地日志。

✧ force-binary

是否以二进制格式输出解码结果。

取值范围：0 或 1，默认值为 0，均以二进制格式输出结果。

✧ white-table-list

白名单参数，包含需要进行解码的 schema 和表名。

取值范围：包含白名单中表名的字符串，不同的表以','为分隔符进行隔离；使用'*'来模糊匹配所有情况；schema 名和表名间以'.'分割，不允许存在任意空白符。例：select * from pg_logical_slot_peek_binary_changes('slot1', NULL, 4096, 'white-table-list', 'public.t1,public.t2');

返回值类型：text, xid, bytea

备注：函数返回解码结果，每一条解码结果包含三列，对应上述返回值类型，分别表示 LSN 位置、xid 和二进制格式的解码内容。调用该函数的用户需要具有 SYSADMIN 权限或具有 REPLICATION 权限或继承了内置角色 gs_role_replication 的权限。

❖ **pg_logical_slot_get_binary_changes('slot_name', 'LSN', upto_nchanges, 'options_name', 'options_value')**

描述：以二进制格式解码并推进流复制槽（下次解码可以再次获取本次解出的数据）。

参数说明：与 pg_logical_slot_peek_binary_changes 一致，详细内容请参见 pg_logical_slot_peek_binary_changes。

备注：调用该函数的用户需要具有 SYSADMIN 权限或具有 REPLICATION 权限或继承了内置角色 gs_role_replication 的权限。

❖ **pg_replication_slot_advance ('slot_name', 'LSN')**

描述：直接推进流复制槽到指定 LSN，不输出解码结果。

参数说明：

- slot_name

  流复制槽名称。

  取值范围：字符串，不支持除字母，数字，以及（_?-.）以外的字 符。

- LSN

  推进到的日志 LSN 位置，下次解码时只会输出提交位置比该 LSN 大则直接返回；如果输入的 LSN 比当前最新物理日志 LSN 还要大，则推进到当前最新物理日志 LSN。

取值范围：字符串（LSN，格式为 xlogid/xrecoff）。

返回值类型：name, text

备注：返回值分别对应 slot_name 和实际推进至的 LSN。调用该函数的用户需要具有 SYSADMIN 权限或具有 REPLICATION 权限或继承了内置角色 gs_role_replication 的权限。此函数目前只支持在主机调用。

❖ **pg_logical_get_area_changes('LSN_start', 'LSN_end', upto_nchanges, 'decoding_plugin', 'xlog_path', 'options_name', 'options_value')**

描述：没有 ddl 的前提下，指定 lsn 区间进行解码，或者指定 xlog 文件进行解码。

约束条件如下：

1. 调用接口时，日志级别 wal_level=logical，且只有在 wal_level=logical 期间产生的日志文件才能被解析，如果使用的 xlog 文件为非 logical 级别，则解码内容没有对应的值和类型，无其他影响。
2. xlog 文件只能被完全同构的 dn 的某个副本解析，确保可以找到数据对应的元信息，且没有 DDL 操作和 VACUUM FULL。
3. 用户可以找到需要解析的 xlog。
4. 用户需要注意一次不要读入过多 xlog 文件，推荐一次一个，一个 xlog 文件估测占用内存为 xlog 文件大小的 2~3 倍。
5. 无法解码扩容前的 xlog 文件。

参数说明：

- LSN_start

  指定开始解码的 lsn。

取值范围：字符串（LSN，格式为 xlogid/xrecoff），如'1/2AAFC60'。为 NULL 时表示不对解码截止的日志位置做限制。

- LSN_end

  指定解码结束的 lsn。

  取值范围：字符串（LSN，格式为 xlogid/xrecoff），如'1/2AAFC60'。为 NULL 时表示不对解码截止的日志位置做限制。

- upto_nchanges

  解码条数（包含 begin 和 commit）。假设一共有三条事务，分别包含 3、5、7 条记录，如果 upto_nchanges 为 4，那么会解码出前两个事务共 8 条记录。解码完第二条事务时发现解码条数记录大于等于 upto_nchanges，会停止解码。

  取值范围：非负整数。

说明

LSN 和 upto_nchanges 中任一参数达到限制，解码都会结束。

- decoding_plugin

  解码插件，指定解码内容输出格式的 so 插件。

  取值范围：提供 mppdb_decoding 和 sql_decoding 两个解码插件。

- xlog_path

  解码插件，指定解码文件的 xlog 绝对路径，文件级别

  取值范围：NULL 或者 xlog 文件绝对路径的字符串。

- options：此项为可选参数，由一系列 options_name 和 options_value 一一对应组成，可以缺省，详见 pg_logical_slot_peek_changes。

示例：

```
panweidb=# SELECT pg_current_xlog_location();
 pg_current_xlog_location
--------------------------
 0/E62E238
(1 row)
```

```
panweidb=# create table t1 (a int primary key,b int,c int);
NOTICE:  CREATE TABLE / PRIMARY KEY will create implicit index "t1_pkey" for t
able "t1"
CREATE TABLE
panweidb=# insert into t1 values(1,1,1);
INSERT 0 1
panweidb=# insert into t1 values(2,2,2);
INSERT 0 1

panweidb=# select data from pg_logical_get_area_changes('0/E62E238',NULL,
NULL,'sql_decoding',NULL);
 location  | xid  |                 data
-----------+------+-------------------------------------------------
 0/E62E8D0 | 27213 | COMMIT (at 2022-01-26 15:08:03.349057+08) 3020226
 0/E6325F0 | 27214 | COMMIT (at 2022-01-26 15:08:07.309869+08) 3020234
   ......
```

- **pg_get_replication_slots()**

  描述：获取复制槽列表。

  返回值类型：text，text，text，oid，boolean，xid，xid，text，boolean

  示例：

```
panweidb=# select * from pg_get_replication_slots();
 slot_name |    plugin     | slot_type | datoid | active | xmin | catalog_xmin | restart
_lsn | dummy_standby
-----------+---------------+-----------+--------+--------+------+--------------+-
-----------+---------------
 wkl001    | mppdb_decoding | logical  | 15914 | f      |      |      2079556 | 4/1B81D9
20  | f
 dn_6002   |               | physical |     0 | t      |      |              | 8/7CB63BD8  | f
 dn_6004   |               | physical |     0 | t      |      |              | 8/7CB63BD8  | f
 dn_6003   |               | physical |     0 | t      |      |              | 8/7CB63BD8  | f
 gfslot001 | mppdb_decoding | logical  | 15914 | f      |      |      2412553 | 4/A54B2
428  | f
```

```
(5 rows)
```

- **gs_get_parallel_decode_status()**

  描述：监控各个解码线程的读取日志队列和解码结果队列的长度，以便定位并行解码性能瓶颈。

  返回值类型：text, int, text, text

  示例：

```
panweidb=# select * from gs_get_parallel_decode_status();
 slot_name | parallel_decode_num |              read_change_queue_length
         |              decode_change_queue_length
-----------+---------------------+-----------------------------------------------
---------------+-------------------------------------------------------------------
 slot1     |                   3 | queue0: 33, queue1: 36, queue2: 1017                    | q
ueue0: 1011, queue1: 1008, queue2: 27
 slot2     |                   5 | queue0: 452, queue1: 1017, queue2: 233, queue3: 585,
queue4: 183 | queue0: 754, queue1: 188, queue2: 972, queue3: 620, queue4:
1022
(2 rows)
```

  备注：返回值的 slot_name 代表复制槽名，parallel_decode_num 代表该复制槽的并行解码线程数，read_change_queue_length 列出了每个解码线程读取日志队列的当前长度，decode_change_queue_length 列出了每个解码线程解码结果队列的当前长度。

- **pg_replication_origin_create (node_name)**

  描述：用给定的外部名称创建一个复制源，并且返回分配给它的内部 ID。

  备注：调用该函数的用户需要具有 SYSADMIN 权限。

  参数说明：

  - node_name

    待创建的复制源的名称。

    取值范围：字符串，不支持除字母，数字，以及（_?-.）以外的字符。

    返回值类型：oid

- **pg_replication_origin_drop (node_name)**

  描述：删除一个以前创建的复制源，包括任何相关的重放进度。

备注：调用该函数的用户需要具有 SYSADMIN 权限。

参数说明：

- node_name

  待删除的复制源的名称。

  取值范围：字符串，不支持除字母，数字，以及（_?-.）以外的字符。

❖ **pg_replication_origin_oid (node_name)**

描述：根据名称查找复制源并返回内部 ID。如果没有发现这样的复制源，则抛出错误。

备注：调用该函数的用户需要具有 SYSADMIN 权限。

参数说明：

- node_name

  要查找的复制源的名称

  取值范围：字符串，不支持除字母，数字，以及（_?-.）以外的字符。

  返回值类型：oid

❖ **pg_replication_origin_session_setup (node_name)**

描述：将当前会话标记为从给定的原点回放，从而允许跟踪回放进度。只能在当前没有选择原点时使用。使用 pg_replication_origin_session_reset 命令来撤销。

备注：调用该函数的用户需要具有 SYSADMIN 权限。

参数说明：

- node_name

  复制源名称。

  取值范围：字符串，不支持除字母，数字，以及（_?-.）以外的字符。

❖ **pg_replication_origin_session_reset ()**

描述：取消 pg_replication_origin_session_setup()的效果。

备注：调用该函数的用户需要具有 SYSADMIN 权限。

❖ **pg_replication_origin_session_is_setup ()**

描述：如果在当前会话中选择了复制源则返回真。

备注：调用该函数的用户需要具有 SYSADMIN 权限。

返回值类型：boolean

- **pg_replication_origin_session_progress (flush)**

  描述：返回当前会话中选择的复制源的重放位置。

  备注：调用该函数的用户需要具有 SYSADMIN 权限。

  参数说明：

  - flush

    决定对应的本地事务是否被确保已经刷入磁盘。

    取值范围：boolean

    返回值类型：LSN

- **pg_replication_origin_xact_setup (origin_lsn, origin_timestamp)**

  描述：将当前事务标记为重放在给定 LSN 和时间戳上提交的事务。只能在使用 pg_replication_origin_session_setup 选择复制源时调用。

  备注：调用该函数的用户需要具有 SYSADMIN 权限。

  参数说明：

  - origin_lsn

    复制源回放位置。

    取值范围：LSN

  - origin_timestamp

    事务提交时间。

    取值范围：timestamp with time zone

- **pg_replication_origin_xact_reset ()**

  描述：取消 pg_replication_origin_xact_setup()的效果。

  备注：调用该函数的用户需要具有 SYSADMIN 权限。

- **pg_replication_origin_advance (node_name, lsn)**

  描述：

  将给定节点的复制进度设置为给定的位置。这主要用于设置初始位置，或在配置更改或类似的变更后设置新位置。

  注意：这个函数的使用不当可能会导致不一致的复制数据。

  备注：调用该函数的用户需要具有 SYSADMIN 权限。

参数说明：

- node_name

  已有复制源名称。

  取值范围：字符串，不支持除字母，数字，以及（_?-.）以外的字符。

- lsn

  复制源回放位置。

  取值范围：LSN

❖ **pg_replication_origin_progress (node_name, flush)**

描述：返回给定复制源的重放位置。

备注：调用该函数的用户需要具有 SYSADMIN 权限。

参数说明：

- node_name

  复制源名称。

  取值范围：字符串，不支持除字母，数字，以及（_?-.）以外的字 符。

- flush

  决定对应的本地事务是否被确保已经刷入磁盘。

  取值范围：boolean

❖ **pg_show_replication_origin_status()**

描述：获取复制源的复制状态。

备注：调用该函数的用户需要具有 SYSADMIN 权限。

返回值类型：

- local_id：oid，复制源 id。
- external_id：text，复制源名称。
- remote_lsn：LSN，复制源的 lsn 位置。
- local_lsn：LSN，本地的 lsn 位置。

❖ **pg_get_publication_tables(pub_name)**

描述：根据发布的名称，返回对应发布要发布的表的 relid 列表

参数说明：

- pub_name

  已存在的发布名称

取值范围：字符串，不支持除字母，数字，以及（_?-.）以外的字符。

返回值类型：relid 列表

- **pg_stat_get_subscription(subid oid)**

描述：

输入订阅的 oid，返回订阅的状态信息。

参数说明：

- subid

描述：输入订阅的 oid，返回订阅的状态信息。

参数说明：

- subid：oid，订阅的 oid。
- 取值范围：oid

返回值类型：

- subid：oid，返回的 oid。
- relid：oid，表的 oid。
- pid：thread_id，后台 apply/sync 线程的 thread id。
- received_lsn：pg_lsn，从发布端接收到的最近的 lsn。
- last_msg_send_time：timestamp，最近发布端发送消息的时间。
- last_msg_receipt_time：timestamp，最新订阅端收到消息的时间。
- latest_end_lsn：pg_lsn，最近一次收到保活消息时发布端的 lsn。
- latest_end_time：timstamp，最近一次收到保活消息的时间。

4.1.4.9.9. 段页式存储函数

- local_segment_space_info(tablespacename TEXT, databasename TEXT)

描述：输出为该表空间下所有 ExtentGroup 的使用信息。

返回值类型：

node_name	节点名称。
extent_size	该 ExtentGroup 的 extent 规格，单位是 block 数。

forknum	Fork 号。
total_blocks	物理文件总 extent 数目。
meta_data_blocks	表空间管理的 metadata 占用的 block 数，只包括 space header、map page 等，不包括 segment head。
used_data_blocks	存数据占用的 extent 数目。包括 segment head。
utilization	使用的 block 数占总 block 数的百分比。即(used_data_blocks+meta_data_block)/total_blocks。
high_water_mark	高水位线，被分配出去的 extent，最大的物理页号。超过高水位线的 block 都没有被使用，可以被直接回收。

例如：

```
select * from local_segment_space_info('pg_default', 'postgres');
   node_name    | extent_size | forknum | total_blocks | meta_data_blocks | used_data_blocks | utilization | high_water_mark
------------------+-------------+---------+--------------+------------------+------------------+-------------+-----------------
 dn_6001_6002_6003 |           1 |       0 |        16384 |             4157 |                1 |   .253784 |            4158
 dn_6001_6002_6003 |           8 |       0 |        16384 |             4157 |                8 |   .254211 |            4165
(2 rows)
```

❖ pg_stat_segment_extent_usage(int4 tablespace oid, int4 database oid, int4 extent_type, int4 forknum)

描述：每次返回一个 ExtentGroup 中，每个被分配出去的 extent 的使用情况。extent_type 表示 ExtentGroup 的类型，合理取值为[1,5]的 int 值。在此范围外的会报 error。forknum 表示 fork 号，合法取值为[0,4]的 int 值，目前只有三种值有效，数据文件为 0，FSM 文件为 1，visibility map 文件为 2。

返回值类型：

名称	描述

名称	描述
start_block	Extent 的起始物理页号。
extent_size	Extent 的大小。
usage_type	Extent 的使用类型，比如 segment head、data extent 等。
ower_location	有指针指向该 extent 的对象的位置。比如 data extent 的 owner 就是它所属的 segment 的 head 位置。
special_data	该 extent 在它 owner 中的位置。该字段的数据跟使用类型有关。比如 data extent 的 special data 就是它在所属 segment 中的 extent id。

其中，usage_type 为枚举类型，每一项的含义为：

- Non-bucket table segment head：非 hashbucket 表的数据段头。
- Non-bucket table fork head：非段页式表的 fork 段头。
- Data extent：数据块。

例如：

```
select * from pg_stat_segment_extent_usage((select oid::int4 from pg_tablespace
where spcname='pg_default'), (select oid::int4 from pg_database where datname
='postgres'), 1, 0);
start_block | extent_size |     usage_type      | ower_location | special_data
------------+-------------+---------------------+---------------+--------------
4157 |           1 | Data extent |   4294967295 |            0
4158 |           1 | Data extent |        4157 |            0
```

- local_space_shrink(tablespacename TEXT, databasename TEXT)

  描述：当前节点上对指定段页式空间做物理空间收缩。注意，目前只支持对当前连接的 database 做 shrink。

  返回值：空

- gs_space_shrink(int4 tablespace, int4 database, int4 extent_type, int4 forknum)

描述：效果跟 local_space_shrink 类似，对指定段页式空间做物理空间收缩,但参数不同，传入的是 tablespace 和 database 的 oid，extent_type 为[1,5]的 int 值。注意：extent_type = 1 表示段页式元数据，当前版本不支持对元数据所在的物理文件做收缩。该函数仅限工具使用，不建议用户直接使用。

返回值：空

- pg_stat_remain_segment_info()

  描述：展示在当前节点上，因为故障等原因而残留的 extent。残留 extent 主要分为两类：分配而未被利用的 segment 和分配出去而未被利用的 extent。两者主要区别在于 segment 会包含多个 extent，回收时，要将 segment 上的 extent 一并全部回收。

  返回值类型：

名称	描述
space_id	表空间 ID
db_id	数据库 ID
block_id	Extent 的 ID
type	Extent 的类型，当前有三种：ALLOC_SEGMENT、DROP_SEGMENT、SHRINK_EXTENT

其中 type 的三种类型分别表示：

- ALLOC_SEGMENT:用户创建一张段页式表，当 segment 刚被分配，但是建表语句所在事务仍未提交时，节点故障，导致该 segment 被分配后，没有被使用。
- DROP_SEGMENT:用户删除段页式表，当该事务成功提交，但是此表的 segment 页面对应的 bit 位未被重置，就发生掉电等故障，造成该 segment 未被使用，也未被释放。
- SHRINK_EXTENT:用户对段页式表执行 shrink 操作，在未对空置出的 extent 进行释放时，发生掉电等故障，造成该 extent 残留，无法被重新利用。

例如：

```
select * from pg_stat_remain_segment_info();
space_id | db_id | block_id | type
----------+-------+----------+------
1663     |  16385|      4156| ALLOC_SEGMENT
```

- pg_free_remain_segment(int4 spaceId, int4 dbId, int4 segmentId)

  描述：释放指定的残留 extent。参数取值必须为从函数 pg_stat_remain_segment_info 中查询获取。函数会对传入值校验，如果指定 extent 不在记录的残留 extent 中，将返回错误信息。指定的 extent 如果为单个 extent，则只将其独自释放；如果为一个 segment，则会将此 segment 以及此 segment 上记录的所有 extent 释放。

  返回值：空

4.1.4.9.10. 其他函数

- **plan_seed()**

  描述：获取前一次查询语句的 seed 值（内部使用）。

  返回值类型：int

- **pg_stat_get_env()**

  描述：获取当前节点的环境变量信息，仅 sysadmin 和 monitor admin 可以访问。

  返回值类型：record

  示例：

```
panweidb=# select pg_stat_get_env();
                                pg_stat_get_env
------------------------------------------------------------------------------------
------------
(node1,localhost,11677,5432,/home/panweidb/local/panweidb,/home/pan
weidb/data/panweidb,pg_log)
(1 row)
```

- **pg_catalog.plancache_clean()**

  描述：清理节点上无人使用的全局计划缓存。

返回值类型：bool

- **pg_catalog.plancache_status()**

  描述：显示节点上全局计划缓存的信息，函数返回信息和GLOBAL_PLANCACHE_STATUS 一致。

  返回值类型：record

- **textlen(text)**

  描述：提供查询 text 的逻辑长度的方法。

  返回值类型：int

- **threadpool_status()**

  描述：显示线程池中工作线程及会话的状态信息。

  返回值类型：record

- **get_local_active_session()**

  描述：提供当前节点保存在内存中的历史活跃 session 状态的采样记 录。

  返回值类型：record

- **pg_stat_get_thread()**

  描述：提供当前节点下所有线程的状态信息，sysadmin 和 monitoradmin 用户可以查看所有线程信息，普通用户查看本用户的线程信息。

  返回值类型：record

- **pg_stat_get_sql_count()**

  描述：提供当前节点中用户执行的 SELECT/UPDATE/INSERT/DELETE/MERGE INTO 语句的计数结果， sysadmin 和 monitor admin 用户可以查看所有用户的信息，普通用户查看本用户的统计信息。

  返回值类型：record

- **pg_stat_get_data_senders()**

  描述：提供当前活跃的数据复制发送线程的详细信息。

  返回值类型：record

- **get_wait_event_info()**

  描述：提供 wait event 事件的具体信息。

返回值类型：record

generate_wdr_report(begin_snap_id bigint, end_snap_id bigint, report_type cstring, report_scope cstring, node_name cstring)

描述：基于两个 snapshot 生成系统诊断报告。需要在 postgres 库下执行，默认初始化用户或 monadmin 用户可以访问，初始化用户或 sysadmin 用户可以访问。只可在系统库中查询到结果，用户库中无法查询。

返回值类型：record

表 1 generate_wdr_report 参数说明

参数	说明	取值范围
begin_snap_id	生成某段时间内性能诊断报告的开始 snapshotid。	-
end_snap_id	结束 snapshot 的 id，默认 end_snap_id 大于 begin_snap_id。	-
report_type	指定生成 report 的类型。	summarydetailall，即同时包含 summary 和 detail。
report_scope	指定生成 report 的范围。	cluster：数据库级别的信息 node：节点级别的信息
node_name	在 report_scope 指定为 node 时，需要把该参数指定为对应节点的名称。（节点名称可以执行 select * from pg_node_env; 查询）。在 report_scope 为 cluster 时，该值可以省略或者指定为空或 NULL。	cluster：省略/空/NULLnode：panweidb 中的节点名称

❖ **create_wdr_snapshot()**

描述：手工生成系统诊断快照，该函数需要 sysadmin 权限。可在单机节点或集群主节点上执行。

返回值类型：text

- **kill_snapshot()**

  描述：kill 后台的 WDR snapshot 线程，调用该函数的用户需要具有 SYSADMIN 权限或具有 REPLICATION 权限或继承了内置角色 gs_role_replication 的权限。

  返回值类型：void

- **capture_view_to_json(text,integer)**

  描述：将视图的结果存入 GUC: perf_directory 所指定的目录，如果 is_crossdb 为 1，则表示对于所有的 database 都会访问一次 view；如果 is_crossdb 为 0，则表示仅对当前 database 进行一次视图访问。该函数只有 sysadmin 和 monitor admin 用户可以执行。

  返回值类型：int

- **reset_unique_sql**

  描述：用来清理数据库节点内存中的 Unique SQL（需要 sysadmin 权限）。

  返回值类型：bool

表 2 reset_unique_sql 参数说明

参数	类型	描述
scope	text	清理范围类型： ❖ 'GLOBAL'：清理所有的节点，如果是'GLOBAL'，则只可以为主节点执行此函数。 ❖ 'LOCAL'：清理本节点。
clean_type	text	❖ 'BY_USERID'：按用户 ID 来进行清理 Unique SQL。 ❖ 'BY_CNID'：按主节点的 ID 来进行清理 Unique SQL。 ❖ 'ALL'：全部清理。
clean_value	int8	具体清理 type 对应的清理值。

【说明】

- scope 的取值 GLOBAL 和 LOCAL 针对分布式，对于 panweidb 而言两者意义相同，均表示清理本节点。
- clean_type 的值 BY_CNID 仅针对分布式，对于 panweidb 无效。

- **wdr_xdb_query(db_name_str text, query text)**

  描述：提供本地跨数据库执行 query 的能力。例如: 在连接到 postgres 库时, 访问 test 库下的表。

```
select col1 from wdr_xdb_query('dbname=test','select col1 from t1') as dd(col1 int);
```

  返回值类型：record

- **pg_wlm_jump_queue(pid int)**

  描述：调整任务到数据库主节点队列的最前端。

  返回值类型：boolean

  true：成功。

  false：失败。

- **gs_wlm_switch_cgroup(pid int, cgroup text)**

  描述：调整作业的优先级到新控制组。

  返回值类型：boolean

  true：成功。

  false：失败。

- **pv_session_memctx_detail(threadid tid, MemoryContextName text)**

  描述：将线程 tid 的 MemoryContextName 内存上下文信息记录到“$GAUSSLOG/pg_log/${node_name}/dumpmem”目录下的“threadid_timestamp.log”文件中。其中 threadid 可通过视图 GS_SESSION_MEMORY_DETAIL 中的 sessid 后获得。在正式发布的版本中仅接受 MemoryContextName 为空串（两个单引号表示输入为空串，即”）的输入，此时会记录所有的内存上下文信息，否则不会有任何操作。对供内部开发人员和测试人员调试用的 DEBUG 版本，可以指定需要统计的 MemoryCon

textName，此时会将该 Context 所有的内存使用情况记录到指定文件。该函数需要管理员权限的用户才能执行。

返回值类型：boolean

true：成功。

false：失败。

- **pg_shared_memctx_detail(MemoryContextName text)**

  描述：将 MemoryContextName 内存上下文信息记录到“$GAUSSLOG/pg_log/${node_name}/dumpmem”目录下的“threadid_timestamp.log”文件中。该函数功能仅在 DEBUG 版本中供内部开发人员和测试人员调试使用，在正式发布版本中调用该函数不会有任何操作。该函数需要管理员权限的用户才能执行。

  返回值类型：boolean

  true：成功。

  false：失败。

- **local_bgwriter_stat()**

  描述：显示本实例的 bgwriter 线程刷页信息，候选 buffer 链中页面个数，buffer 淘汰信息。

  返回值类型：record

- **local_candidate_stat()**

  描述：显示本实例的候选 buffer 链中页面个数，buffer 淘汰信息，包含 normal buffer pool 和 segment buffer pool。

  返回值类型：record

- **local_ckpt_stat()**

  描述：显示本实例的检查点信息和各类日志刷页情况。

  返回值类型：record

- **local_double_write_stat()**

  描述：显示本实例的双写文件的情况。

  返回值类型：record

表 3 local_double_write_stat 参数说明

参数	类型	描述
node_name	text	实例名称。
curr_dwn	int8	当前双写文件的序列号。
curr_start_page	int8	当前双写文件恢复起始页面。
file_trunc_num	int8	当前双写文件复用的次数。
file_reset_num	int8	当前双写文件写满后发生重置的次数。
total_writes	int8	当前双写文件总的 I/O 次数。
low_threshold_writes	int8	低效率写双写文件的 I/O 次数（一次 I/O 刷页数量少于 16 页面）。
high_threshold_writes	int8	高效率写双写文件的 I/O 次数（一次 I/O 刷页数量多于一批，421 个页面）。
total_pages	int8	当前刷页到双写文件区的总的页面个数。
low_threshold_pages	int8	低效率刷页的页面个数。
high_threshold_pages	int8	高效率刷页的页面个数。
file_id	int8	当前双写文件的 id 号。

- **local_single_flush_dw_stat()**

  描述：显示本实例的单页面淘汰双写文件的情况。

  返回值类型：record

- **local_pagewriter_stat()**

  描述：显示本实例的刷页信息和检查点信息。

  返回值类型：record

- **local_redo_stat()**

  描述：显示本实例的备机的当前回放状态。

  返回值类型：record

备注：返回的回放状态主要包括当前回放位置、回放最小恢复点位置等信息。

- **local_recovery_status()**

  描述：显示本实例的主机和备机的日志流控信息。

  返回值类型：record

- **gs_wlm_node_recover(boolean isForce)**

  描述：获取当前内存中记录的 TopSQL 查询语句级别相关统计信息，当传入的参数不为 0 时，会将这部分信息从内存中清理掉。

  返回值类型：record

- **gs_wlm_node_clean(cstring nodename)**

  描述：动态负载管理节点故障后做数据清理操作。该函数只有管理员用户可以执行，属于数据库实例管理模块调用的，不建议用户直接调用。该视图在集中式和单机环境上不支持。

  返回值类型：bool

- **gs_cgroup_map_ng_conf(group name)**

  描述：读取指定逻辑数据库的 cgroup 配置文件。

  返回值类型：record

- **gs_wlm_switch_cgroup(sess_id int8, cgroup name)**

  描述：切换指定会话的控制组。

  返回值类型：record

- **comm_client_info()**

  描述：用于查询单个节点活跃的客户端连接信息。

  返回值类型：setof record

- **pg_sync_cstore_delta(text)**

  描述：同步指定列存表的 delta 表表结构，使其与列存表主表一致。

  返回值类型：bigint

- **pg_sync_cstore_delta()**

  描述：同步所有列存表的 delta 表表结构，使其与列存表主表一致。

返回值类型：bigint

- **pg_get_flush_lsn()**

  描述：返回当前节点 flush 的 xlog 位置。

  返回值类型：text

- **pg_get_sync_flush_lsn()**

  描述：返回当前节点多数派 flush 的 xlog 位置。

  返回值类型：text

- **dbe_perf.get_global_full_sql_by_timestamp(start_timestamp timestamp with time zone, end_timestamp timestamp with time zone)**

  描述：获取数据库级的全量 SQL(Full SQL)信息。只可在系统库中查询到结果，用户库中无法查询。

  返回值类型：record

表 4 dbe_perf.get_global_full_sql_by_timestamp 参数说明

参数	类型	描述
start_timestamp	timestamp with time zone	SQL 启动时间范围的开始时间点。
end_timestamp	timestamp with time zone	SQL 启动时间范围的结束时间点。

- **dbe_perf.get_global_slow_sql_by_timestamp(start_timestamp timestamp with time zone, end_timestamp timestamp with time zone)**

  描述：获取数据库级的慢 SQL(Slow SQL)信息。只可在系统库中查询到结果，用户库中无法查询。

  返回值类型：record

表 5 dbe_perf.get_global_slow_sql_by_timestamp 参数说明

参数	类型	描述

参数	类型	描述
start_timestamp	timestamp with time zone	SQL 启动时间范围的开始时间点。
end_timestamp	timestamp with time zone	SQL 启动时间范围的结束时间点。

- **statement_detail_decode(detail text, format text, pretty boolean)**

  描述：解析全量/慢 SQL 语句中的 details 字段的信息。只可在系统库中查询到结果，用户库中无法查询。

  返回值类型：text

表 6 statement_detail_decode 参数说明

参数	类型	描述
detail	text	SQL 语句产生的事件的集合（不可读）。
format	text	解析输出格式，取值为 plaintext。
pretty	boolean	当 format 为 plaintext 时，是否以优雅的格式展示：true 表示通过“\n”分隔事。false 表示通过“，”分隔事件。

- **get_prepared_pending_xid**

  描述：当恢复完成时，返回 nextxid。

  参数：nan

  返回值类型：text

- **pg_clean_region_info**

  描述：清理 regionmap。

  参数：nan

  返回值类型：character varying

- **pg_get_delta_info**

  描述：从单个 dn 获取 delta info。

  参数：rel text、 schema_name text

  返回值类型：part_name text、 live_tuple bigint、 data_size bigint、

blocknum bigint

- **pg_get_replication_slot_name**

  描述：获取 slot name。

  参数：nan

  返回值类型：text

- **pg_get_running_xacts**

  描述：获取运行中的 xact。

  参数：nan

  返回值类型：handle integer、 gxid xid、 state tinyint、 node text、 xmin xid、 vacuum boolean、 timeline bigint、 prepare_xid xid、 pid bigint、 next_xid xid

- **pg_get_variable_info**

  描述：获取共享内存变量 cache。

  参数：nan

  返回值类型：node_name text、 nextOid oid、 nextXid xid、 oldestXid xid、 xidVacLimit xid、 oldestXidDB oid、 lastExtendCSNLogpage xid、 startExtendCSNLogpage xid、 nextCommitSeqNo xid、 latestCompletedXid xid、 startupMaxXid xid

- **pg_get_xidlimit**

  描述：从共享内存获取事物 id 信息。

  参数：nan

  返回值类型：nextXid xid、 oldestXid xid、 xidVacLimit xid、 xidWarnLimit xid、 xidStopLimit xid、 xidWrapLimit xid、 oldestXidDB oid

- **get_global_user_transaction()**

  描述：返回所有节点上各用户的事务相关信息。

  返回值类型：node_name name、 usename name、 commit_counter bigint、 rollback_counter bigint、 resp_min bigint、 resp_max bigint、 resp_avg bigint、 resp_total bigint、 bg_commit_counter bigint、 b

g_rollback_counter bigint、 bg_resp_min bigint、 bg_resp_max bigint、 bg_resp_avg bigint、 bg_resp_total bigint

- **pg_collation_for**

  描述：返回入参字符串对应的排序规则。

  参数：any（如果是常量必须进行显式类型转换）

  返回值类型：text

- **pgxc_unlock_for_sp_database(name Name)**

  描述：释放指定数据库锁。

  参数：数据库名

  返回值类型：布尔

- **pgxc_lock_for_sp_database(name Name)**

  描述：对指定的数据库加锁。

  参数：数据库名

  返回值类型：布尔

- **copy_error_log_create()**

  描述：创建 COPY FROM 容错机制所需要的错误表 （public.pgxc_copy_error_log）。

  返回值类型：Boolean

【说明】

- 此函数会尝试创建 public.pgxc_copy_error_log 表，表的详细信息请参见表 7。
- 在 relname 列上创建 B-tree 索引，并 REVOKE ALL on public.pgxc_copy_error_log FROM public 对错误表进行权限控制（与 COPY 语句权限一致）。
- 由于尝试创建的 public.pgxc_copy_error_log 定义是一张行存表，因此数据库实例上必须支持行存表的创建才能够正常运行此函数，并使用后续的 COPY 容错功能。需要特别注意的是，enable_hadoop_env 这个 GUC 参数开启后会禁止在数据库实例内创建行存表（GaussDB Kernel 默认为 off）。

- 此函数自身权限为 Sysadmin 及以上（与错误表、COPY 权限一致）。
- 若创建前 public.pgxc_copy_error_log 表已存在或者 copy_error_log_relname_idx 索引已存在，则此函数会报错回滚。

表 7 错误表 public.pgxc_copy_error_log 信息

列名称	类型	描述
relname	character varying	表名称。以模式名.表名形式显示。
begintime	timestamp with time zone	出现数据格式错误的时间。
filename	character varying	出现数据格式错误的数据源文件名。
lineno	bigint	在数据源文件中，出现数据格式错误的行号。
rawrecord	text	在数据源文件中，出现数据格式错误的原始记录。
detail	text	详细错误信息。

- **dynamic_func_control(scope text, function_name text, action text, “{params}” text[])**

  描述：动态开启内置的功能，当前仅支持动态开启全量 SQL。

  返回值类型：record

表 8 dynamic_func_control 参数说明

参数	类型	描述
scope	text	动态开启功能的范围，当前仅支持'LOCAL'。
function_name	text	功能的名称，当前仅支持'STMT'。
action	text	当 function_name 为'STMT'时，action 仅支持 TRACK/UNTRACK/LIST/CLEAN：TRACK - 开始记录归一化 SQL 的全量 SQL 信息。UNTRACK - 取消记录归一化 SQL 的全量 SQL 信息。LIST - 列取当前 TRACK

参数	类型	描述
		的归一化 SQL 的信息。CLEAN - 清理记录当前归一化 SQL 的信息。
params	text[]	当 function_name 为'STMT'时，对应不同的 action 时，对应的 params 设置如下：TRACK - '{ “归一化 SQLID”，“L0/L1/L2” }'UNTRACK - '{ “归一化 SQLID” }'LIST - '{}'CLEAN - '{}'

- **gs_parse_page_bypath(path text, blocknum bigint, relation_type text, read_memory boolean)**

  描述：用于解析指定表页面，并返回存放解析内容的路径。

  返回值类型：text

  备注：必须是系统管理员或运维管理员才能执行此函数。

表 9gs_parse_page_bypath 参数说明

参数	类型	描述
path	text	对于普通表或段页式表，相对路径为：tablespace name/database oid/表的 relfilenode(物理文件名)。例如：base/16603/16394。表文件的相对路径可以通过 pg_relation_filepath(table_name text)查找。合法的 path 格式列举：global/relNodebase/dbNode/relNodepg_tblspc/spcNode/version_dir/dbNode/relNode
blocknum	bigint	-1 所有 block 的信息 0- MaxBlockNumber 对应 block 的信息
relation_type	text	heap(astore 表)uheap(ustore 表)btree_index(BTree 索引)ubtree_index(UBTree 索引)segment(段页式)
read_memory	boolean	false，从磁盘文件解析。true，首先尝试从共享缓冲区中解析该页面；如果共享缓冲区中不存在，则从磁盘文件解析。

- **gs_xlogdump_lsn(start_lsn text, end_lsn text)**

  描述：用于解析指定 lsn 范围之内的 XLOG 日志，并返回存放解析内容

的路径。可以通过 pg_current_xlog_location()获取当前 XLOG 位置。

返回值类型：text

参数：LSN 起始位置，LSN 结束位置

备注：必须是系统管理员或运维管理员才能执行此函数。

- **gs_xlogdump_xid(c_xid xid)**

  描述：用于解析指定 xid 的 XLOG 日志，并返回存放解析内容的路径。可以通过 txid_current()获取当前事务 ID。

  参数：事务 ID

  返回值类型：text

  备注：必须是系统管理员或运维管理员才能执行此函数。

- **gs_xlogdump_tablepath(path text, blocknum bigint, relation_type text)**

  描述：用于解析指定表页面对应的日志，并返回存放解析内容的路径。

  返回值类型：text

  备注：必须是系统管理员或运维管理员才能执行此函数。

表 10 gs_xlogdump_tablepath 参数说明

参数	类型	描述
path	text	对于普通表或段页式表，相对路径为：tablespace name/database oid/表的 relfilenode(物理文件名)。例如：base/16603/16394。表文件的相对路径可以通过 pg_relation_filepath(table_name text)查找。合法的 path 格式列举：global/relNodebase/dbNode/relNodepg_tblspc/spcNode/version_dir/dbNode/relNode
blocknum	bigint	-1 所有 block 的信息 0- MaxBlockNumber 对应 block 的信息
relation_type	text	heap(astore 表)uheap(ustore 表)btree_index(BTree 索引)ubtree_index(UBTree 索引)segment(段页式)

- **gs_xlogdump_parsepage_tablepath(path text, blocknum bigint, relation_type text, read_memory boolean)**

  描述：用于解析指定表页面和表页面对应的日志，并返回存放解析内容的路径。可以看做一次执行 gs_parse_page_bypath 和 gs_xlogdump_tablepath。该函数执行的前置条件是表文件存在。如果 想查看已删除的表的相关日志，请直接调用 gs_xlogdump_tablepath。

  返回值类型：text

  备注：必须是系统管理员或运维管理员才能执行此函数。

表 11 gs_xlogdump_parsepage_tablepath 参数说明

参数	类型	描述
path	text	对于普通表或段页式表，相对路径为：tablespace name/database oid/表的 relfilenode(物理文件名)；例如：base/16603/16394 表文件的相对路径可以通过 pg_relation_filepath(table_name text)查找。合法的 path 格式列举：global/relNodebase/dbNode/relNodepg_tblspc/spcNode/version_dir/dbNode/relNode
blocknum	bigint	-1 所有 block 的信息 0- MaxBlockNumber 对应 block 的信息
relation_type	text	heap(astore 表)uheap(ustore 表)btree_index(BTree 索引)ubtree_index(UBTree 索引)segment(段页式)
read_memory	boolean	false，从磁盘文件解析 true，首先尝试从共享缓冲区中解析该页面；如果共享缓冲区中不存在，则从磁盘文件解析

- **gs_index_verify(Oid oid, uint32:wq blkno)**

  描述：用于校验 UBtree 索引页面或者索引树上 key 的顺序是否正确。

  返回值类型：record

表 12 gs_index_verify 参数说明

参数	类型	描述

参数	类型	描述
oid	Oid	索引文件 relfilenode,可以通过 select relfilenode from pg_class where relname='name'查询，其中 name 表示对应的索引文件名字。
blkno	uint32	0，表示检验整个索引树上所有页面。大于 0，表示校验页面编码等于 blkno 的索引页面。

❖ gs_index_recycle_queue(Oid oid, int type, uint32 blkno)

描述：用于解析 UBtree 索引回收队列信息。

返回值类型：record

表 13 gs_index_recycle_queue 参数说明

参数	类型	描述
oid	Oid	索引文件 relfilenode,可以通过 select relfilenode from pg_class where relname='name'查询，其中 name 表示对应的索引文件名字。
type	int	表示解析整个待回收队列。 表示解析整个空页队列。 表示解析单个页面。
blkno	uint32	回收队列页面编号，该参数只有在 type=2 的时候有效，blkno 有效取值范围为 1~4294967294。

❖ **gs_stat_wal_entrytable(int64 idx)**

描述：用于输出 xlog 中预写日志插入状态表的内容。

返回值类型：record

表 14 gs_stat_wal_entrytable 参数说明

参数类型	参数名	类型	描述
输入参数	idx	int64	-1：查询数组所有元素。0-最大值：具体某个数组元素内容。

参数类型	参数名	类型	描述
输出参数	idx	uint64	记录对应数组中的下标。
输出参数	endlsn	uint64	记录的 LSN 标签。
输出参数	lrc	int32	记录对应的 LRC。
输出参数	status	uint32	标识当前 entry 对应的 xlog 是否已经完全拷贝到 wal buffer 中：0：非 COPIED1：COPIED

- **gs_walwriter_flush_position()**

  描述：输出预写日志的刷新位置。

  返回值类型：record

表 15 gs_walwriter_flush_position 参数说明

参数类型	参数名	类型	描述
输出参数	last_flush_status_entry	int32	Xlog flush 上一个刷盘的 tblEntry 下标索引。
输出参数	last_scanned_lrc	int32	Xlog flush 上一次扫描到的最后一个 tblEntry 记录的 LRC。
输出参数	curr_lrc	int32	WALInsertStatusEntry 状态表中 LRC 最新的使用情况，该 LRC 表示下一个 Xlog 记录写入时在 WALInsertStatusEntry 对应的 LRC 值。
输出参数	curr_byte_pos	uint64	Xlog 记录写入 WAL 文件，最新分配的位置，下一个 xlog 记录插入点。
输出参数	prev_byte_size	uint32	上一个 xlog 记录的长度。

参数类型	参数名	类型	描述
输出参数	flush_result	uint64	当前全局 xlog 刷盘的位置。
输出参数	send_result	uint64	当前主机上 xlog 发送位置。
输出参数	shm_rqst_write_pos	uint64	共享内存中记录的 XLogCtl 中 LogwrtRqst 请求的 write 位置。
输出参数	shm_rqst_flush_pos	uint64	共享内存中记录的 XLogCtl 中 LogwrtRqst 请求的 flush 位置。
输出参数	shm_result_write_pos	uint64	共享内存中记录的 XLogCtl 中 LogwrtResult 的 write 位置。
输出参数	shm_result_flush_pos	uint64	共享内存中记录的 XLogCtl 中 LogwrtResult 的 flush 位置。
输出参数	curr_time	text	当前时间。

- **gs_walwriter_flush_stat(int operation)**

  描述：用于统计预写日志 write 与 sync 的次数频率与数据量，以及 xlog 文件的信息。

  返回值类型：record

表 16 gs_walwriter_flush_stat 参数说明

参数类型	参数名	类型	描述
输入参数	operation	int	-1：关闭统计开关(默认状态为关闭)。 0：打开统计开关。 1：查询统计信息。 2：重置统计信息。
输出参数	write_times	uint64	Xlog 调用 write 接口的次数。
输出参数	sync_times	uint64	Xlog 调用 sync 接口次数。

参数类型	参数名	类型	描述
输出参数	total_xlog_sync_bytes	uint64	Backend 线程请求写入 xlog 总量统计值。
输出参数	total_actual_xlog_sync_bytes	uint64	调用 sync 接口实际刷盘的 xlog 总量统计值。
输出参数	avg_write_bytes	uint32	每次调用 XLogWrite 接口请求写的 xlog 量。
输出参数	avg_actual_write_bytes	uint32	实际每次调用 write 接口写的 xlog 量。
输出参数	avg_sync_bytes	uint32	平均每次请求 sync 的 xlog 量。
输出参数	avg_actual_sync_bytes	uint32	实际每次调用 sync 刷盘 xlog 量。
输出参数	total_write_time	uint64	调用 write 操作总时间统计(单位：us)。
输出参数	total_sync_time	uint64	调用 sync 操作总时间统计(单位：us)。
输出参数	avg_write_time	uint32	每次调用 write 接口平均时间(单位：us)。
输出参数	avg_sync_time	uint32	每次调用 sync 接口平均时间(单位：us)。
输出参数	curr_init_xlog_segno	uint64	当前最新创建的 xlog 段文件编号。
输出参数	curr_open_xlog_segno	uint64	当前正在写的 xlog 段文件编号。
输出参数	last_reset_time	text	上一次重置统计信息的时间。
输出参数	curr_time	text	当前时间。

❖ **gs_comm_proxy_thread_status()**

描述：用于在数据库实例配置用户态网络的场景下，代理通信库 comm_proxy 收发数据包统计。

参数：nan

返回值类型：record

【说明】

此函数的查询仅在集中式环境开始部署用户态网络，且 comm_proxy_attr 参数中 enable_dfx 配置为 true 的条件下显示具体信 息。其他场景报错不支持查询。

❖ **pg_ls_tmpdir(oid)**

描述：返回指定表空间下临时目录中每个文件的名称、大小和最后修改时间。

参数：oid

返回值类型：record

备注：必须是管理员或者监控管理员才能执行此函数。

参数类型	参数名	类型	描述
输入参数	oid	oid	表空间 id
输出参数	name	text	文件名称
输出参数	size	int8	文件大小（单位：byte）
输出参数	modification	timestamptz	文件最后修改时间

❖ **pg_ls_waldir()**

描述：返回预写日志（WAL）目录中每个文件的名称、大小和最后修改时间。

参数：nan

返回值类型：record

备注：必须是管理员或者监控管理员才能执行此函数。

❖ **GS_WRITE_TERM_LOG(void)**

描述：写入一条日志记录数据库节点当前的 term 值。备节点返回 false，主节点写入成功后返回 true。

返回值类型：Boolean

❖ **GS_CREATE_LOG_TABLES()**

描述：用于创建运行日志和性能日志的外表和视图。单机模式下不支持。

返回值类型：Void

示例：

```
select gs_create_log_tables();
```

返回结果为：

```
 gs_create_log_tables
-------------------------


(1 row)
```

4.1.4.9.11. Undo 系统函数

❖ **gs_undo_meta(type, zoneId, location)**

描述：Undo 各模块元信息。

参数说明：

- type(元信息类型)
  - 0 表示 Undo Zone(Record) 对应的元信息。
  - 1 表示 Undo Zone(Transaction Slot) 对应的元信息。
  - 2 表示 Undo Space(Record) 对应的元信息。
  - 3 表示 Undo Space(Transaction Slot) 对应的元信息。
- zoneId(undo zone 编号)
  - -1 表示所有 undo zone 的元信息。
  - 0-1024*1024 表示对应 zoneid 的元信息。
- location(读取位置)
  - 0 表示从当前内存中读取。
  - 1 表示从物理文件中读取。

返回值类型：record

- **gs_undo_translot(location, zoneId)**

  描述：Undo 事务槽信息。

  参数说明：

  - location(读取位置)
    - 0 表示从当前内存中读取。
    - 1 表示从物理文件中读取。
  - zoneId(undo zone 编号)
    - -1 表示所有 undo zone 的元信息。
    - 0-1024*1024 表示对应 zoneId 的元信息。

  返回值类型：record

- **gs_stat_undo()**

  描述：Undo 统计信息。

  返回值类型：record

  表 1 gs_stat_undo 参数说明

参数类型	参数名	类型	描述
输出参数	curr_used_zone_count	uint32	当前使用的 Undo zone 数量。
输出参数	top_used_zones	text	前三个使用量最大的 Undo zone 信息，格式输出为：(zoneId1:使用大小，zoneId2:使用大小，zoneId3:使用大小)。
输出参数	curr_used_undo_size	uint32	当前使用的 Undo 总空间大小，单位为 MB。
输出参数	undo_threshold	uint32	为 guc 参数 undo_space_limit_size * 80%计算的结果,单位为 MB。
输出参	oldest_xid_in_undo	uint64	当前 Undo 空间回收到的事务

参数类型	参数名	类型	描述
数			xid(小于该 xid 事务产生的 Undo 记录都已经被回收)。
输出参数	oldest_xmin	uint64	最老的活跃事务。
输出参数	total_undo_chain_len	int64	所有访问过的 Undo 链总长度。
输出参数	max_undo_chain_len	int64	最大访问过的 Undo 链长度。
输出参数	create_undo_file_count	uint32	创建的 Undo 文件数量统计。
输出参数	discard_undo_file_count	uint32	删除的 Undo 文件数量统计。

❖ **gs_undo_record(undoptr)**

描述：Undo 记录解析。

参数说明：

- undoptr(undo 记录指针)

返回值类型：record

4.1.4.9.12. PW_VERSION 函数

功能描述

使用 pw_version 函数查询 PanWeiDB 产品版本信息。显示内容包括产品名称、版本号、补丁号、提交号和基于 openGauss 版本的信息。

注意事项

❖ 本函数只有在 PanWeiDB 数据库能正常启动及登录的前提下使用。

❖ 本函数大写可识别。

语法格式

查询语句：

```
select pw_version();
```

返回结果格式：

```
                          pw_version
-------------------------------------------------------------------
 (PanWeiDB xxx) compiled at xxx commit xxx last mr
+
  product xxx
+
  version:xxx
+
  patch:xxx
+
  commit:xxx
+
  openGauss version:xxx
+
  host:xxx
```

示例

查询产品信息

```
select pw_version();
```

返回结果为：

```
                          pw_version
-------------------------------------------------------------------------------------
--------
 (PanWeiDB V2.0-S3.0.1_B01 Release) compiled at 2023-09-14 15:17:24 commit 15
623 last mr                                                          +
  product name:PanWeiDB                                          +
  version:V2.0-S3.0.1_B01                                     +
  patch:0                                                        +
  commit:15623                                                      +
  openGauss version:3.0.1                                           +
```

```
 host:x86_64-pc-linux-gnu
(1 row)
```

4.1.4.9.13. IP 白名单查询与更新函数

功能描述

PanWeiDB 支持 IP 白名单查询和更新函数，可以查询和更新 pg_hba.conf 文件中的配置条目。

- 调用 read_hba_config 函数，查询 pg_hba.conf 文件中配置条目信息；
- 调用 modify_hba_config 函数，增加、删除、修改 pg_hba.conf 文件中的配置。

注意事项

IP 白名单查询函数：

```
SELECT * FROM read_hba_config();
```

IP 白名单更新函数：

```
SELECT * FROM modify_hba_config(1,'new_item'::cstring);
SELECT * FROM modify_hba_config(2,item_id);
SELECT * FROM modify_hba_config(3,item_id,'new_item');
```

参数说明

- **1/2/3**

  操作标识符。

  取值范围：

  - 1，新增操作的标识符，表示使用函数为 pg_hba.conf 文件中新增配置。
  - 2，删除操作的标识符，表示使用函数删除指定的 pg_hba.conf 文件配置。

- 3，修改操作的标识符，表示使用函数修改指定的 pg_hba.conf 文件配置。

❖ **new_item**

当操作标识符为 2 时，new_item 代表 pg_hba.conf 文件中新增的内容；操作标识符为 3 时，new_item 代表修改后的配置内容。

new_item 格式需符合 pg_hba.conf 中配置要求，即满足如下形式：

```
TYPE  DATABASE    USER    ADDRESS    METHOD
```

❖ **item_id**

配置条目 ID，即配置条目位于 pg_hba.conf 文件中的行数。

注意事项

只有系统管理员才能执行 read_hba_config 和 modify_hba_config 函数。

示例

1、查询 pg_hba.conf 文件中配置。

```
SELECT * FROM read_hba_config();
```

查询结果为：

```
 id | type  | database | user |    ip     | method
----+-------+----------+------+-----------+--------
 89 | local | all      | all  |           | trust
 91 | host  | all      | all  | 127.0.0.1/32 | trust
 93 | host  | all      | all  | ::1/128   | trust
 99 | host  | all      | all  | 0.0.0.0/0 | md5
(4 rows)
```

2、修改 pg_hba.conf 文件中配置。

```
SELECT * FROM modify_hba_config(3,89,'host all all 172.16.105.58/22 md5');
```

修改成功，回显为：

```
 modify_hba_config
-------------------
 t
(1 row)
```

3、删除 pg_hba.conf 文件中配置。

```
SELECT * FROM modify_hba_config(2,89);
```

删除成功，回显为：

```
 modify_hba_config
-------------------
 t
(1 row)
```

4、新增 pg_hba.conf 文件中配置并查询。

```
SELECT * FROM modify_hba_config(1,'local all all  trust'::cstring);
SELECT * FROM read_hba_config();
```

查询当前配置文件内容：

```
 id | type  | database | user |     ip      | method
----+-------+----------+------+-------------+--------
 90 | host  | all      | all  | 127.0.0.1/32 | trust
 92 | host  | all      | all  | ::1/128      | trust
 98 | host  | all      | all  | 0.0.0.0/0    | md5
 99 | local | all      | all  |              | trust
(4 rows)
```

4.1.4.10.数组函数和操作符

数组操作符

数组比较是使用默认的 B-tree 比较函数对所有元素逐一进行比较的。多维数组的元素按照行顺序进行访问。如果两个数组的内容相同但维数不等，决定排序顺序的首要因素是维数。

数组函数

函数名	描述	返回类型
array_append(anyarray, anyelement)	向数组末尾添加元素，只支持一维数组。	anyarray
array_prepend(anyelement, anyarray)	向数组开头添加元素，只支持一维数组。	anyarray
array_cat(anyarray, anyarray)	连接两个数组，支持多维数组。	anyarray

函数名	描述	返回类型
array_ndims(anyarray)	返回数组的维数。	int
array_dims(anyarray)	返回数组各个维度中的低位下标值和高位下标值。	text
array_length(anyarray, int)	返回指定数组维度的长度。int 为指定数组维度。	int
array_lower(anyarray, int	返回指定数组维数的下界。int 为指定数组维度。	int
array_upper(anyarray, int)	返回指定数组维数的上界。int 为指定数组维度。	int
array_to_string(anyarray, text [, text]	使用第一个 text 作为数组的新分隔符，使用第二个 text 替换数组值为 null 的值。	text
string_to_array(text, text [, text]	使用第二个 text 指定分隔符，使用第三个可选的 text 作为 NULL 值替换模板，如果分隔后的子串与第三个可选的 text 完全匹配，则将其替换为 NULL。	text[]
unnest(anyarray)	扩大一个数组为一组行。	setof anyelement
array_position(anyarray,anyelement,[,integer])	实现在数组搜索元素，返回该元素在数组中第一次出现的下标	int
array_remove(anyarray,anyelement)	移除数组中的所有指定元素。仅支持一维数组。	anyarray
array_replace(anyarray,anyelement, anyelement)	指定数组中的元素进行替换。将数组中包含的第一个 anyelement 的值替换为第二个 anyelement 值。	anyarray
array_delete(anyarray)	清空数组中的元素并返回一个同类型的空数组。	anyarray

示例

- array_append(anyarray, anyelement)

```
select array_append(ARRAY[1,2], 3) AS RESULT;
```

当结果显示如下信息，则表示函数调用成功。

```
result
---------
{1,2,3}
(1 row)
```

❖ array_prepend(anyelement, anyarray)

```
select array_prepend(1, ARRAY[2,3]) AS RESULT;
```

当结果显示如下信息，则表示函数调用成功。

```
result
---------
{1,2,3}
(1 row)
```

❖ array_cat(anyarray, anyarray)

示例 1：

```
select array_cat(ARRAY[1,2,3], ARRAY[4,5]) AS RESULT;
```

当结果显示如下信息，则表示函数调用成功。

```
result
-------------
{1,2,3,4,5}
(1 row)
```

示例 2：

```
select array_cat(ARRAY[[1,2],[4,5]], ARRAY[6,7]) AS RESULT;
```

当结果显示如下信息，则表示函数调用成功。

```
result
---------------------
{{1,2},{4,5},{6,7}}
(1 row)
```

❖ array_ndims(anyarray)

```
select array_ndims(ARRAY[[1,2,3], [4,5,6]]) AS RESULT;
```

当结果显示如下信息，则表示函数调用成功。

```
result
--------
2
(1 row)
```

❖ array_dims(anyarray)

```
select array_dims(ARRAY[[1,2,3], [4,5,6]]) AS RESULT;
```

当结果显示如下信息，则表示函数调用成功。

```
result
------------
[1:2][1:3]
(1 row)
```

❖ array_length(anyarray, int)

示例 1：

```
select array_length(array[1,2,3], 1) AS RESULT;
```

当结果显示如下信息，则表示函数调用成功。

```
result
--------
3
(1 row)
```

示例 2：

```
select array_length(array[[1,2,3],[4,5,6]], 2) AS RESULT;
```

当结果显示如下信息，则表示函数调用成功。

```
result
--------
3
(1 row)
```

❖ array_lower(anyarray, int)

```
select array_lower('[0:2]={1,2,3}'::int[], 1) AS RESULT;
```

当结果显示如下信息，则表示函数调用成功。

```
result
--------
0
(1 row)
```

❖ array_upper(anyarray, int)

```
select array_upper(ARRAY[1,8,3,7], 1) AS RESULT;
```

当结果显示如下信息，则表示函数调用成功。

```
result
--------
4
(1 row)
```

❖ array_to_string(anyarray, text [, text])

描述：

返回类型：

```
select array_to_string(ARRAY[1, 2, 3, NULL, 5], ',', '*') AS RESULT;
```

当结果显示如下信息，则表示函数调用成功。

```
result
-----------
1,2,3,*,5
(1 row)
```

在 array_to_string 中，如果省略 null 字符串参数或为 NULL，运算中将跳过在数组中的任何 null 元素，并且不会在输出字符串中出现。

❖ string_to_array(text, text [, text])

示例 1：

```
select string_to_array('xx~^~yy~^~zz', '~^~', 'yy') AS RESULT;
```

当结果显示如下信息，则表示函数调用成功。

```
result
--------------
{xx,NULL,zz}
(1 row)
```

示例 2：

```
select string_to_array('xx~^~yy~^~zz', '~^~', 'y') AS RESULT;
```

当结果显示如下信息，则表示函数调用成功。

```
result
------------
{xx,yy,zz}
(1 row)
```

【说明】

在 string_to_array 中，如果分隔符参数是 NULL，输入字符串中的每个字符将在结果数组中变成一个独立的元素。如果分隔符是一个空白字符串，则整个输入的字符串将变为一个元素的数组。否则输入字符串将在每个分隔字符串处分开。如果省略 null 字符串参数或为 NULL，将字符串中没有输入内容的子串替换为 NULL。

❖ unnest(anyarray)

```
select unnest(ARRAY[1,2]) AS RESULT;
```

当结果显示如下信息，则表示函数调用成功。

```
result
--------
1
2
(2 rows)
```

❖ array_position(anyarray,anyelement,[,integer])

```
SELECT array_position(array[1,4,2,5,7],2) AS RESULT;
```

当结果显示如下信息，则表示函数调用成功。

```
result
--------
```

```
3
(1 row)
```

- array_remove(anyarray,anyelement)

```
SELECT array_remove(ARRAY[1,8,8,7], 8) AS RESULT;
```

当结果显示如下信息，则表示函数调用成功。

```
result
--------
{1,7}
(1 row)
```

- array_replace(anyarray,anyelement,anyelement)

```
SELECT array_replace(ARRAY[1,8,8,7],1,2) AS RESULT;
```

当结果显示如下信息，则表示函数调用成功。

```
result
--------
{2,8,8,7}
(1 row)
```

- array_delete(anyarray)

```
SELECT array_delete(ARRAY[1,8,8,7]) AS RESULT;
```

当结果显示如下信息，则表示函数调用成功。

```
result
--------
{}
(1 row)
```

4.1.4.11.聚集函数

- **sum(expression)**

描述：所有输入行的 expression 总和。

返回类型：

通常情况下输入数据类型和输出数据类型是相同的，但以下情况会发生类型转换：

- 对于 SMALLINT 或 INT 输入，输出类型为 BIGINT。
- 对于 BIGINT 输入，输出类型为 NUMBER 。
- 对于浮点数输入，输出类型为 DOUBLE PRECISION。

示例：

1、创建测试表并插入数据。

```
CREATE TABLE tab(a int);
Insert into tab values(1);
Insert into tab values(1);
```

2、执行查询语句。

```
SELECT SUM(a) FROM tab;
```

结果显示为：

```
  sum
--------------
3
(1 row)
```

❖ **max(expression)**

描述：所有输入行中 expression 的最大值。

参数类型：任意数组、数值、字符串、日期/时间类型。

返回类型：与参数数据类型相同

示例：

1、创建测试表并插入数据。

```
CREATE TABLE max_t1(a int, b int);
INSERT INTO max_t1 VALUES(1,2),(2,3),(3,4),(4,5);
```

2、执行查询语句。

```
SELECT MAX(a) FROM max_t1;
```

结果显示为：

```
max
```

```
-----
4
(1 row)
```

❖ **min(expression)**

描述：所有输入行中 expression 的最小值。

参数类型：任意数组、数值、字符串、日期/时间类型。

返回类型：与参数数据类型相同

示例：

1、创建测试表并插入数据。

```
CREATE TABLE min_t1(a int, b int);
INSERT INTO min_t1 VALUES(1,2),(2,3),(3,4),(4,5);
```

2、执行查询语句。

```
SELECT MIN(a) FROM min_t1;
```

结果显示为：

```
 min
-----
1
(1 row)
```

❖ **avg(expression)**

描述：所有输入值的均值（算术平均）。

返回类型：

对于任何整数类型输入，结果都是 NUMBER 类型。

对于任何浮点输入，结果都是 DOUBLE PRECISION 类型。

否则和输入数据类型相同。

示例：

1、创建测试表并插入数据。

```
CREATE TABLE avg_t1(a int, b int);
INSERT INTO avg_t1 VALUES(1,2),(2,3),(3,4),(4,5);
```

2、执行查询语句。

```
SELECT AVG(a) FROM avg_t1;
```

结果显示为：

```
     avg
--------------------
2.5000000000000000
(1 row)
```

❖ **count(expression)**

描述：返回表中满足 expression 不为 NULL 的行数。

返回类型：BIGINT

示例：

1、创建测试表并插入数据。

```
CREATE TABLE count_t1(a int, b int);
INSERT INTO count_t1 VALUES (NULL,1),(1,2),(2,3),(3,4),(4,5);
```

2、执行查询语句。

```
SELECT COUNT(a) FROM count_t1;
```

结果显示为：

```
count
-------
   4
(1 row)
```

❖ **count(*)**

描述：返回表中的记录行数。

返回类型：BIGINT

示例：

1、创建测试表并插入数据。

```
CREATE TABLE count_t1(a int, b int);
INSERT INTO count_t1 VALUES (NULL,1),(1,2),(2,3),(3,4),(4,5);
```

2、执行查询语句。

```
SELECT COUNT(*) FROM count_t1;
```

结果显示为：

```
count
-------
   5
(1 row)
```

- **median(expression) [over (query partition clause)]**

  描述：返回表达式的中位数，计算时 NULL 将会被 median 函数忽略。可以使用 distinct 关键字排除表达式中的重复记录。输入 expression 的数据类型可以是数值类型（包括 integer、 double、bigint 等），也可以是 interval 类型。其他数据类型不支持求取中位数。

  返回类型：double 或 interval 类型

  示例：

  执行查询语句：

```
SELECT MEDIAN(id) FROM (values(1), (2), (3), (4), (null)) test(id);
```

  结果显示为：

```
median
--------
   2.5
(1 row)
```

- **array_agg(expression)**

  描述：将所有输入值（包括空）连接成一个数组。

  返回类型：参数类型的数组。

  示例：

  1、创建测试表并插入数据。

```
CREATE TABLE array_agg_t1(a int, b int);
INSERT INTO array_agg_t1 VALUES (NULL,1),(1,2),(2,3),(3,4),(4,5);
```

  2、执行查询语句。

```
SELECT ARRAY_AGG(a) FROM array_agg_t1;
```

  结果显示为：

```
  array_agg
---------------
 {NULL,1,2,3,4}
(1 row)
```

- **string_agg(expression, delimiter)**

  描述：将输入值连接成为一个字符串，用分隔符分开。

  返回类型：和参数数据类型相同。

示例：

1、创建测试表并插入数据。

```
CREATE TABLE string_agg_t1(a int, b int);
INSERT INTO string_agg_t1 VALUES (NULL,1),(1,2),(2,3),(3,4),(4,5);
```

2、执行查询语句。

```
SELECT STRING_AGG(a,';') FROM string_agg_t1;
```

结果显示为：

```
 string_agg
------------
 1;2;3;4
(1 row)
```

❖ **listagg(expression [, delimiter]) WITHIN GROUP(ORDER BY order-list)**

描述：将聚集列数据按 WITHIN GROUP 指定的排序方式排列，并用 delimiter 指定的分隔符拼接成一个字符串。

- expression：必选。指定聚集列名或基于列的有效表达式，不支持 DISTINCT 关键字和 VARIADIC 参数。
- delimiter：可选。指定分隔符，可以是字符串常数或基于分组列的确定性表达式，缺省时表示分隔符为空。
- order-list：必选。指定分组内的排序方式。

返回类型：text

示例：

示例 1：聚集列是文本字符集类型。

1、创建测试表并插入数据。

```
CREATE TABLE listagg_t1(a int, b text);
INSERT INTO listagg_t1 VALUES (NULL,'a1'),(1,'b2'),(1,'c3'),(2,'d4'),(2,'e5'),(3,
'f6');
```

2、执行查询语句。

```
SELECT a,LISTAGG(b,';') WITHIN GROUP(ORDER BY b) FROM listagg_t1 group
 by a;
```

结果显示为：

```
 a | listagg
```

```
---+---------
 1 | b2;c3
 2 | d4;e5
 3 | f6
   | a1
(4 rows)
```

示例 2：聚集列是整型。

1、创建测试表并插入数据。

```
CREATE TABLE listagg_t1(a int, b int);
INSERT INTO listagg_t1 VALUES (NULL,1),(1,2),(1,3),(2,4),(2,5),(3,6);
```

2、执行查询语句。

```
SELECT a,LISTAGG(b,';') WITHIN GROUP(ORDER BY b) FROM listagg_t1 group
 by a;
```

结果显示为：

```
 a | listagg
---+---------
 1 | 2;3
 2 | 4;5
 3 | 6
   | 1
(4 rows)
```

示例 3：聚集列是浮点类型。

1、创建测试表并插入数据。

```
CREATE TABLE listagg_t1(a int, b float);
INSERT INTO listagg_t1 VALUES (NULL,1.111),(1,2.222),(1,3.333),(2,4.444),
(2,5.555),(3,6.666);
```

2、执行查询语句。

```
SELECT a,LISTAGG(b,';') WITHIN GROUP(ORDER BY b) FROM listagg_t1 group
 by a;
```

结果显示为：

```
 a |     listagg
---+------------------
 1 | 2.222000;3.333000
```

```
 2 | 4.444000;5.555000
 3 | 6.666000
   | 1.111000
(4 rows)
```

示例 4：聚集列是时间类型。

1、创建测试表并插入数据。

```
CREATE TABLE listagg_t1(a int, b timestamp);
INSERT INTO listagg_t1 VALUES (NULL,'2000-01-01'),(1,'2000-02-02'),(1,'20
00-03-03'),(2,'2000-04-04'),(2,'2000-05-05'),(3,'2000-06-06');
2、执行查询语句。
SELECT a,LISTAGG(b,';') WITHIN GROUP(ORDER BY b) FROM listagg_t1 group
 by a;
```

结果显示为：

```
 a |              listagg
---+---------------------------------------
 1 | 2000-02-02 00:00:00;2000-03-03 00:00:00
 2 | 2000-04-04 00:00:00;2000-05-05 00:00:00
 3 | 2000-06-06 00:00:00
   | 2000-01-01 00:00:00
(4 rows)
```

示例 5：聚集列是时间间隔类型。

1、创建测试表并插入数据。

```
CREATE TABLE listagg_t1(a int, b interval);
INSERT INTO listagg_t1 VALUES (NULL,'1 days'),(1,'2 days'),(1,'3 days'),(2,'4
days'),(2,'5 days'),(3,'6 days');
```

2、执行查询语句。

```
SELECT a,LISTAGG(b,';') WITHIN GROUP(ORDER BY b) FROM listagg_t1 group
 by a;
```

结果显示为：

```
 a |    listagg
---+---------------
 1 | 2 days;3 days
 2 | 4 days;5 days
```

```
3 | 6 days
  | 1 day
(4 rows)
```

示例 6：分隔符缺省时，默认为空。

1、创建测试表并插入数据。

```
CREATE TABLE listagg_t1(a int, b interval);
INSERT INTO listagg_t1 VALUES (NULL,'1 days'),(1,'2 days'),(1,'3 days'),(2,'4 days'),(2,'5 days'),(3,'6 days');
```

2、执行查询语句。

```
SELECT a,LISTAGG(b) WITHIN GROUP(ORDER BY b) FROM listagg_t1 group by a;
```

结果显示为：

```
 a |   listagg
---+--------------
 1 | 2 days3 days
 2 | 4 days5 days
 3 | 6 days
   | 1 day
(4 rows)
```

示例 7：listagg 作为窗口函数时，OVER 子句不支持 ORDER BY 的窗口排序，listagg 列为对应分组的有序聚集。

1、创建测试表并插入数据。

```
CREATE TABLE listagg_t1(a int, b interval);
INSERT INTO listagg_t1 VALUES (NULL,'1 days'),(1,'2 days'),(1,'3 days'),(2,'4 days'),(2,'5 days'),(3,'6 days');
```

2、执行查询语句。

```
SELECT a,LISTAGG(b) WITHIN GROUP(ORDER BY b) OVER(PARTITION BY a) FROM listagg_t1;
```

结果显示为：

```
 a |   listagg
---+--------------
 1 | 2 days3 days
 1 | 2 days3 days
```

```
 2 | 4 days5 days
 2 | 4 days5 days
 3 | 6 days
   | 1 day
(6 rows)
```

❖ **covar_pop(Y, X)**

描述：总体协方差。

返回类型：double precision

示例：

1、创建测试表并插入数据。

```
CREATE TABLE covar_pop_t1(a int, b int);
INSERT INTO covar_pop_t1 VALUES (NULL,11),(11,21),(11,31),(21,41),(21,5
1),(31,61);
```

2、执行查询语句。

```
SELECT COVAR_POP(a,b) FROM covar_pop_t1;
```

结果显示为：

```
 covar_pop
-----------
       100
(1 row)
```

❖ **covar_samp(Y, X)**

描述：样本协方差。

返回类型：double precision

示例：

1、创建测试表并插入数据。

```
CREATE TABLE covar_samp_t1(a int, b int);
INSERT INTO covar_samp_t1 VALUES (NULL,11),(11,21),(11,31),(21,41),(21,
51),(31,61);
```

2、执行查询语句。

```
SELECT COVAR_SAMP(a,b) FROM covar_samp_t1;
```

结果显示为：

```
 covar_samp
```

```
------------
     125
(1 row)
```

❖ **stddev_pop(expression)**

描述：总体标准差。

返回类型：对于浮点类型的输入返回 double precision，其他输入返回 numeric。

示例：

1、创建测试表并插入数据。

```
CREATE TABLE stddev_pop_t1(a int, b int);
INSERT INTO stddev_pop_t1 VALUES (NULL,11),(11,21),(11,31),(21,41),(21,51),(31,61);
```

2、执行查询语句。

```
SELECT STDDEV_POP(a) FROM stddev_pop_t1;
结果显示为：
    stddev_pop
--------------------
 7.4833147735478828
(1 row)
```

❖ **stddev_samp(expression)**

描述：样本标准差。

返回类型：对于浮点类型的输入返回 double precision，其他输入返回 numeric。

示例：

1、创建测试表并插入数据。

```
CREATE TABLE stddev_samp_t1(a int, b int);
INSERT INTO stddev_samp_t1 VALUES (NULL,11),(11,21),(11,31),(21,41),(21,51),(31,61);
```

2、执行查询语句。

```
SELECT STDDEV_SAMP(a) FROM stddev_samp_t1;
```

结果显示为：

```
   stddev_samp
```

```
--------------------
 8.3666002653407555
(1 row)
```

❖ **var_pop(expression)**

描述：总体方差（总体标准差的平方）

返回类型：对于浮点类型的输入返回 double precision 类型，其他输入返回 numeric 类型。

示例：

1、创建测试表并插入数据。

```
CREATE TABLE var_pop_t1(a int, b int);
INSERT INTO var_pop_t1 VALUES (NULL,11),(11,21),(11,31),(21,41),(21,51),(31,61);
```

2、执行查询语句。

```
SELECT VAR_POP(a) FROM var_pop_t1;
```

结果显示为：

```
     var_pop
---------------------
 56.0000000000000000
(1 row)
```

❖ **var_samp(expression)**

描述：样本方差（样本标准差的平方）

返回类型：对于浮点类型的输入返回 double precision 类型，其他输入返回 numeric 类型。

示例：

1、创建测试表并插入数据。

```
CREATE TABLE var_samp_t1(a int, b int);
INSERT INTO var_samp_t1 VALUES (NULL,11),(11,21),(11,31),(21,41),(21,51),(31,61);
```

2、执行查询语句。

```
SELECT VAR_SAMP(a) FROM var_samp_t1;
```

结果显示为：

```
   var_samp
--------------------
 70.0000000000000000
(1 row)
```

❖ **bit_and(expression)**

描述：所有非 NULL 输入值的按位与(AND)，如果全部输入值皆为 NULL，那么结果也为 NULL 。

返回类型：和参数数据类型相同。

示例：

1、创建测试表并插入数据。

```
CREATE TABLE bit_and_t1(a int, b int);
INSERT INTO bit_and_t1 VALUES (NULL,11),(1,2),(1,3),(2,4),(2,5),(3,6);
```

2、执行查询语句。

```
SELECT BIT_AND(a) FROM bit_and_t1;
```

结果显示为：

```
 bit_and
---------
       0
(1 row)
```

❖ **bit_or(expression)**

描述：所有非 NULL 输入值的按位或(OR)，如果全部输入值皆为 NULL，那么结果也为 NULL。

返回类型：和参数数据类型相同

示例：

1、创建测试表并插入数据。

```
CREATE TABLE bit_or_t1(a int, b int);
INSERT INTO bit_or_t1 VALUES (NULL,11),(1,2),(1,3),(2,4),(2,5),(3,6);
```

2、执行查询语句。

```
SELECT BIT_OR(a) FROM bit_or_t1;
```

结果显示为：

```
 bit_or
```

```
--------
   3
(1 row)
```

❖ **bool_and(expression)**

描述：如果所有输入值都是真，则为真，否则为假。

返回类型：bool

示例：

```
SELECT bool_and(100 <2500);
```

结果显示为：

```
 bool_and
-----------
 t
(1 row)
```

❖ **bool_or(expression)**

描述：如果所有输入值只要有一个为真，则为真，否则为假。

返回类型：bool

示例：

执行查询语句：

```
SELECT bool_or(100 <2500);
```

结果显示为：

```
 bool_or
-----------
  t
(1 row)
```

❖ **corr(Y, X)**

描述：相关系数

返回类型：double precision

示例：

1、创建测试表并插入数据。

```
CREATE TABLE corr_t1(a int, b int);
INSERT INTO corr_t1 VALUES (NULL,11),(1,2),(1,3),(2,4),(2,5),(3,6);
```

2、执行查询语句。

```
SELECT CORR(a,b) FROM corr_t1;
```

结果显示为：

```
    corr
------------------
 .944911182523068
(1 row)
```

❖ **every(expression)**

描述：等效于 bool_and。

返回类型：bool

示例：

执行查询语句：

```
SELECT every(100 <2500);
```

结果显示为：

```
every
--------
  t
(1 row)
```

❖ **regr_avgx(Y, X)**

描述：自变量的平均值 (sum(X)/N)

返回类型：double precision

示例：

1、创建测试表并插入数据。

```
CREATE TABLE regr_t1(a int, b int);
INSERT INTO regr_t1 VALUES (NULL,11),(1,2),(1,3),(2,4),(2,5),(3,6);
```

2、执行查询语句。

```
SELECT REGR_AVGX(a,b) FROM regr_t1;
```

结果显示为：

```
 regr_avgx
-----------
      4
(1 row)
```

- **regr_avgy(Y, X)**

  描述：因变量的平均值 (sum(Y)/N)

  返回类型：double precision

  示例：

  1、创建测试表并插入数据。

```
CREATE TABLE regr_avgy_t1(a int, b int);
INSERT INTO regr_avgy_t1 VALUES (NULL,11),(1,2),(1,3),(2,4),(2,5),(3,6);
```

  2、执行查询语句。

```
SELECT REGR_AVGY(a,b) FROM regr_avgy_t1;
```

  结果显示为：

```
 regr_avgy
-----------
     1.8
(1 row)
```

- **regr_count(Y, X)**

  描述：两个表达式都不为 NULL 的输入行数。

  返回类型：bigint

  示例：

  1、创建测试表并插入数据。

```
CREATE TABLE regr_count_t1(a int, b int);
INSERT INTO regr_count_t1 VALUES (NULL,11),(1,2),(1,3),(2,4),(2,5),(3,6);
```

  2、执行查询语句。

```
SELECT REGR_COUNT(a,b) FROM regr_count_t1;
```

  结果显示为：

```
 regr_count
------------
          5
(1 row)
```

- **regr_intercept(Y, X)**

  描述：根据所有输入的点(X, Y)按照最小二乘法拟合成一个线性方程，然后返回该直线的 Y 轴截距。

返回类型：double precision

示例：

1、创建测试表并插入数据。

```
CREATE TABLE regr_intercept_t1(a int, b int);
INSERT INTO regr_intercept_t1 VALUES (NULL,11),(1,2),(1,3),(2,4),(2,5),(3,6);
```

2、执行查询语句。

```
SELECT REGR_INTERCEPT(b,a) FROM regr_intercept_t1;
```

结果显示为：

```
 regr_intercept
-----------------
 .785714285714286
(1 row)
```

❖ **regr_r2(Y, X)**

描述：相关系数的平方。

返回类型：double precision

示例：

1、创建测试表并插入数据。

```
CREATE TABLE regr_r2_t1(a int, b int);
INSERT INTO regr_r2_t1 VALUES (NULL,11),(1,2),(1,3),(2,4),(2,5),(3,6);
```

2、执行查询语句。

```
SELECT REGR_R2(b,a) FROM regr_r2_t1;
```

结果显示为：

```
   regr_r2
-----------------
 .892857142857143
(1 row)
```

❖ **regr_slope(Y, X)**

描述：根据所有输入的点(X, Y)按照最小二乘法拟合成一个线性方程， 然后返回该直线的斜率。

返回类型：double precision

示例：

1、创建测试表并插入数据。

```
CREATE TABLE regr_slope_t1(a int, b int);
INSERT INTO regr_slope_t1 VALUES (NULL,11),(1,2),(1,3),(2,4),(2,5),(3,6);
```

2、执行查询语句。

```
SELECT REGR_SLOPE(b,a) FROM regr_slope_t1;
```

结果显示为：

```
  regr_slope
------------------
 1.78571428571429
(1 row)
```

- **regr_sxx(Y, X)**

描述：sum(X^2) - sum(X)^2/N （自变量的“平方和”）。

返回类型：double precision

示例：

1、创建测试表并插入数据。

```
CREATE TABLE regr_sxx_t1(a int, b int);
INSERT INTO regr_sxx_t1 VALUES (NULL,11),(1,2),(1,3),(2,4),(2,5),(3,6);
```

2、执行查询语句。

```
SELECT REGR_SXX(b,a) FROM regr_sxx_t1;
```

结果显示为：

```
 regr_sxx
----------
      2.8
(1 row)
```

- **regr_sxy(Y, X)**

描述：sum(X*Y) - sum(X) * sum(Y)/N （自变量和因变量的“乘方积”）。

返回类型：double precision

示例：

1、创建测试表并插入数据。

```
CREATE TABLE regr_sxy_t1(a int, b int);
```

```
INSERT INTO regr_sxy_t1 VALUES (NULL,11),(1,2),(1,3),(2,4),(2,5),(3,6);
```

2、执行查询语句。

```
SELECT REGR_SXY(b,a) FROM regr_sxy_t1;
```

结果显示为：

```
 regr_sxy
----------
        5
(1 row)
```

❖ **regr_syy(Y, X)**

描述：sum(Y^2) - sum(Y)^2/N（因变量的”平方和”）。

返回类型：double precision

示例：

1、创建测试表并插入数据。

```
CREATE TABLE regr_syy_t1(a int, b int);
INSERT INTO regr_syy_t1 VALUES (NULL,11),(1,2),(1,3),(2,4),(2,5),(3,6);
```

2、执行查询语句。

```
SELECT REGR_SYY(b,a) FROM regr_syy_t1;
结果显示为：
 regr_syy
----------
       10
(1 row)
```

❖ **stddev(expression)**

描述：stddev_samp 的别名。

返回类型：对于浮点类型的输入返回 double precision，其他输入返回 numeric。

示例：

1、创建测试表并插入数据。

```
CREATE TABLE stddev_t1(a int, b int);
INSERT INTO stddev_t1 VALUES (NULL,11),(1,2),(1,3),(2,4),(2,5),(3,6);
```

2、执行查询语句。

```
SELECT STDDEV(a) FROM stddev_t1;
结果显示为：
     stddev
----------------------
 .83666002653407554798
(1 row)
```

❖ **variance(expexpression,ression)**

描述：var_samp 的别名。

返回类型：对于浮点类型的输入返回 double precision 类型，其他输入返回 numeric 类型。

示例：

1、创建测试表并插入数据。

```
CREATE TABLE variance_t1(a int, b int);
INSERT INTO variance_t1 VALUES (NULL,11),(1,2),(1,3),(2,4),(2,5),(3,6);
```

2、执行查询语句。

```
SELECT VARIANCE(a) FROM variance_t1;
结果显示为：
    variance
----------------------
 .70000000000000000000
(1 row)
```

❖ **delta**

描述：返回当前行和前一行的差值。

参数：numeric

返回值类型：numeric

❖ **checksum(expression)**

描述：返回所有输入值的 CHECKSUM 值。使用该函数可以用来验证 PanWeiDB 数据库（不支持 PanWeiDB 之外的其他数据库）的备份恢复或者数据迁移操作前后表中的数据是否相同。在备份恢复或者数据迁移操作前后都需要用户通过手工执行 SQL 命令的方式获取执行结果，通过对比获取的执行结果判断操作前后表中的数据是否相同。

【说明】

- 对于大表，CHECKSUM 函数可能会需要很长时间。
- 如果某两表的 CHECKSUM 值不同，则表明两表的内容是不同的。由于 CHECKSUM 函数中使用散列函数不能保证无冲突，因此两个不同内容的表可能会得到相同的 CHECKSUM 值，存在这种情况的可能性较小。对于列进行的 CHECKSUM 也存在相同的情况。
- 对于时间类型 timestamp, timestamptz 和 smalldatetime，计算 CHECKSUM 值时请确保时区设置一致。
- 若计算某列的 CHECKSUM 值，且该列类型可以默认转为 TEXT 类型，则 expression 为列名。
- 若计算某列的 CHECKSUM 值，且该列类型不能默认转为 TEXT 类型，则 expression 为列名::TEXT。
- 若计算所有列的 CHECKSUM 值，则 expression 为表名::TEXT。

可以默认转换为 TEXT 类型的类型包括：char、name、int8、int2、int1、int4、raw、pg_node_tree、float4、float8、bpchar、varchar、nvarchar、nvarchar2、date、timestamp、timestamptz、numeric、smalldatetime，其他类型需要强制转换为 TEXT。

返回类型：numeric。

示例：

示例 1：表中可以默认转为 TEXT 类型的某列的 CHECKSUM 值。

1、创建测试表并插入数据。

```
CREATE TABLE checksum_t1(a int, b int);
INSERT INTO checksum_t1 VALUES (NULL,11),(1,2),(1,3),(2,4),(2,5),(3,6);
```

2、执行查询语句。

```
SELECT CHECKSUM(a) FROM checksum_t1;
```

结果显示为：

```
 checksum
-------------
 18126842830
(1 row)
```

示例 2：表中不能默认转为 TEXT 类型的某列的 CHECKSUM 值。注意此时 CHECKSUM 参数是列名::TEXT。

1、创建测试表并插入数据。

```
CREATE TABLE checksum_t1(a int, b int);
INSERT INTO checksum_t1 VALUES (NULL,11),(1,2),(1,3),(2,4),(2,5),(3,6);
```

2、执行查询语句。

```
SELECT CHECKSUM(a::TEXT) FROM checksum_t1;
结果显示为：
 checksum
-------------
 18126842830
(1 row)
```

3、删除测试表。

```
DROP TABLE checksum_t1;
```

示例 3：表中所有列的 CHECKSUM 值。注意此时 CHECKSUM 参数是表名::TEXT，且表名前不加 Schema。

1、创建测试表并插入数据。

```
CREATE TABLE checksum_t1(a int, b int);
INSERT INTO checksum_t1 VALUES (NULL,11),(1,2),(1,3),(2,4),(2,5),(3,6);
```

2、执行查询语句。

```
SELECT CHECKSUM(checksum_t1::TEXT) FROM checksum_t1;
```

结果显示为：

```
 checksum
-------------
 11160522226
(1 row)
```

3、删除测试表。

```
DROP TABLE checksum_t1;
```

- **first(anyelement)**

描述：返回第一个非 NULL 输入。

返回类型：anyelement

执行查询语句：

```
select * from tba;
select first(name) from tba;
```

结果显示为：

```
name
------
A
A
D
(4 rows)


first
------
A
(1 rows)
```

❖ **last(anyelement)**

描述：返回最后一个非 NULL 输入。

返回类型：anyelement

示例：

执行查询语句：

```
select * from tba;
select last(name) from tba;
```

结果显示为：

```
name
------
A
A
D
(4 rows)

last
------
```

```
D
(1 rows)
```

❖ **mode() within group (order by value anyelement)**

描述：返回某列中出现频率最高的值，如果多个值频率相同，则返回最小的那个值。排序方式和该列类型的默认排序方式相同。其中 value 为输入参数，可以为任意类型。

返回类型：与输入参数类型相同。

示例：

执行查询语句：

```
select mode() within group (order by value) from (values(1, 'a'), (2, 'b'), (2, 'c')) v(value, tag);
select mode() within group (order by tag) from (values(1, 'a'), (2, 'b'), (2, 'c')) v(value, tag);
```

查询结果显示为：

```
 mode
-------
   2
(1 row)

 mode
-------
 a
(1 row)
```

❖ **json_agg(any)**

描述：将值聚集为 json 数组。

返回类型：array-json

示例：

1、创建测试表并插入数据。

```
CREATE TABLE json_agg_t1(a int, b int);
INSERT INTO json_agg_t1 VALUES (NULL,11),(1,2),(1,3),(2,4),(2,5),(3,6);
```

2、执行查询语句。

```
SELECT JSON_AGG(a) FROM json_agg_t1;
```

查询结果显示为：

```
    json_agg
----------------------
[null, 1, 1, 2, 2, 3]
(1 row)
```

3、删除测试表。

```
DROP TABLE json_agg_t1;
```

❖ **json_object_agg(any, any)**

描述：将值聚集为 json 对象。

返回类型：object-json

示例：

1、创建测试表并插入数据。

```
CREATE TABLE json_object_agg_t1(a int, b int);
INSERT INTO json_object_agg_t1 VALUES (11,NULL),(1,2),(1,3),(2,4),(2,5),(3,
6);
```

2、执行查询语句。

```
SELECT JSON_OBJECT_AGG(a,b) FROM json_object_agg_t1;
```

查询结果显示为：

```
              json_object_agg
-------------------------------------------------------------
{ "11" : null, "1" : 2, "1" : 3, "2" : 4, "2" : 5, "3" : 6 }
(1 row)
```

3、删除测试表。

DROP TABLE json_object_agg_t1;

4.1.4.12.逻辑操作符

功能描述

常用的逻辑操作符有 AND、OR 和 NOT，它们的运算结果有三个值，分别为 TRUE、FALSE 和 NULL，其中 NULL 代表未知。它们的运算优先级顺序为：NOT > AND > OR。

运算规则请参见表 1，表中的 a 和 b 代表逻辑表达式。

注意事项

- 操作符 AND 和 OR 具有交换性，即交换左右两个操作数，不影响其结果。
- 逻辑布尔运算符仅支持使用 Boolean 操作数执行逻辑运算。但在 MySQL 兼容模式下，允许 AND，OR 的两侧使用时间类型，详见 MySQL 兼容性手册中的逻辑操作符。

运算规则

表 1 运算规则表

a	b	a AND b 的结果	a OR b 的结果	NOT a 的结果
TRUE	TRUE	TRUE	TRUE	FALSE
TRUE	FALSE	FALSE	TRUE	FALSE
TRUE	NULL	NULL	TRUE	FALSE
FALSE	FALSE	FALSE	FALSE	TRUE
FALSE	NULL	FALSE	NULL	TRUE
NULL	NULL	NULL	NULL	NOT NULL

4.1.4.13.比较操作符

大部分数据类型都可用比较操作符进行比较，并返回一个布尔类型的值。

比较操作符均为双目操作符，被比较的两个数据类型必须是相同的数据类型或者是可以进行隐式转换的类型。

PanWeiDB 提供的比较操作符请参见表。

表 1 比较操作符

操作符	描述
<	小于
>	大于

操作符	描述
<=	小于或等于
>=	大于或等于
=	等于
<> 或 !=或^=	不等于

比较操作符可以用于所有相关的数据类型。所有比较操作符都是双目操作符，返回布尔类型数值。不等号的计算优先级高于等号。当输入的数据不同且无法隐式转换时，比较操作将会失败。例如像 1<2<3 这样的表达式是非法的，因为布尔值和 3 之间无法用小于号（<）比较。

4.1.4.14.二进制字符串函数和操作符

字符串操作符

SQL 定义了一些字符串函数，在这些函数里使用关键字而不是逗号来分隔参数。

- octet_length(string)

  描述：二进制字符串中的字节数。

  返回值类型：int

  示例：

```
panweidb=# SELECT octet_length(E'jo\\000se'::bytea) AS RESULT;
 result
--------
      5
(1 row)
```

- overlay(string placing string from int [for int])

  描述：替换子串。

  返回值类型：bytea

  示例：

```
panweidb=# SELECT overlay(E'Th\\000omas'::bytea placing E'\\002\\003'::bytea f
rom 2 for 3) AS RESULT;
   result
```

```
----------------
 \x5402036d6173
(1 row)
```

❖ position(substring in string)

描述：特定子字符串的位置。

返回值类型：int

示例：

```
panweidb=# SELECT position(E'\\000om'::bytea in E'Th\\000omas'::bytea) AS RES
ULT;
 result
--------
      3
(1 row)
```

❖ substring(string [from int] [for int])

描述：截取子串。

返回值类型：bytea

示例：

```
panweidb=# SELECT substring(E'Th\\000omas'::bytea from 2 for 3) AS RESULT;
 result
----------
 \x68006f
(1 row)
```

❖ substr(string, from int [, for int])

描述：截取子串。

返回值类型：bytea

示例：

```
panweidb=# select substr(E'Th\\000omas'::bytea,2, 3) as result;
 result
----------
 \x68006f
(1 row)
```

❖ trim([both] bytes from string)

描述：从 string 的开头和结尾删除只包含 bytes 中字节的最长字符串。

返回值类型：bytea

示例：

```
panweidb=# SELECT trim(E'\\000'::bytea from E'\\000Tom\\000'::bytea) AS RESUL
T;
 result
----------
 \x546f6d
(1 row)
```

二进制字符串函数

PanWeiDB 也提供了函数调用所使用的常用语法。

- btrim(string bytea,bytes bytea)

  描述：从 string 的开头和结尾删除只包含 bytes 中字节的最长的字符串。

  返回值类型：bytea

  示例：

```
panweidb=# SELECT btrim(E'\\000trim\\000'::bytea, E'\\000'::bytea) AS RESULT;
  result
------------
\x7472696d
(1 row)
```

- get_bit(string, offset)

  描述：从字符串中抽取位。

  返回值类型：int

  示例：

```
panweidb=# SELECT get_bit(E'Th\\000omas'::bytea, 45) AS RESULT;
 result
--------
      1
(1 row)
```

- get_byte(string, offset)

  描述：从字符串中抽取字节。

  返回值类型：int

  示例：

```
panweidb=# SELECT get_byte(E'Th\\000omas'::bytea, 4) AS RESULT;
 result
--------
    109
(1 row)
```

❖ set_bit(string,offset, newvalue)

描述：设置字符串中的位。

返回值类型：bytea

示例：

```
panweidb=# SELECT set_bit(E'Th\\000omas'::bytea, 45, 0) AS RESULT;
    result
------------------
 \x5468006f6d4173
(1 row)
```

❖ set_byte(string,offset, newvalue)

描述：设置字符串中的字节。

返回值类型：bytea

示例：

```
panweidb=# SELECT set_byte(E'Th\\000omas'::bytea, 4, 64) AS RESULT;
    result
------------------
 \x5468006f406173
(1 row)
```

4.1.4.15.位串函数和操作符

位串操作符

除了常用的比较操作符之外，还可以使用以下的操作符。&、|和#的位串操作数必须等长。在位移的时候，保留原始的位串长度（并以 0 填充）。

❖ ||

描述：位串之间进行连接。

示例：

```
panweidb=# SELECT B'10001' || B'011' AS RESULT;
```

```
 result
----------
 10001011
(1 row)
```

说明：

单字段内部连续连接操作不建议超过 180 次。如果超过 180 次，需拆分为多个连续连接的字符串，在它们之间再执行连接操作。例如：str1||str2||str3||str4 拆分为 (str1||str2)||(str3||str4)。

- &

  描述：位串之间进行“与”操作。

  示例：

```
panweidb=# SELECT B'10001' & B'01101' AS RESULT;
 result
--------
 00001
(1 row)
```

- |

  描述：位串之间进行“或”操作。

  示例：

```
panweidb=# SELECT B'10001' | B'01101' AS RESULT;
 result
--------
 11101
(1 row)
```

- #

  描述：位串之间如果不一致进行“或”操作。如果两个位串中对应位置都为 1 或者 0，则该位置返回为 0。

  示例：

```
panweidb=# SELECT B'10001' # B'01101' AS RESULT;
 result
--------
 11100
(1 row)
```

❖ ~

描述：位串之间进行“非”操作。

示例：

```
panweidb=# SELECT ~B'10001'AS RESULT;
 result
----------
 01110
(1 row)
```

❖ <<

描述：位串进行左移操作。

示例：

```
panweidb=# SELECT B'10001' << 3 AS RESULT;
 result
----------
 01000
(1 row)
```

❖ >>

描述：位串进行右移操作。

示例：

```
panweidb=# SELECT B'10001' >> 2 AS RESULT;
 result
----------
 00100
(1 row)
```

下面的 SQL 标准函数除了可以用于字符串之外，也可以用于位串：lengthbit_length、octet_length、position、substring、overlay。

下面的函数用于位串和二进制字符串：get_bit、set_bit。当用于位串时，这些函数位数从字符串的第一位（最左边）作为 0 位。

另外，可以在整数和 bit 之间来回转换。示例：

```
panweidb=# SELECT 44::bit(10) AS RESULT;
  result
```

```
------------
 0000101100
(1 row)

panweidb=# SELECT 44::bit(3) AS RESULT;
 result
--------
 100
(1 row)

panweidb=# SELECT cast(-44 as bit(12)) AS RESULT;
   result
--------------
 111111010100
(1 row)

panweidb=# SELECT '1110'::bit(4)::integer AS RESULT;
 result
--------
    14
(1 row)

panweidb=# select substring('10101111'::bit(8), 2);
 substring
-----------
 0101111
(1 row)
```

说明

只是转换为 “bit” 的意思是转换成 bit(1)，因此只会转换成整数的最低位。

4.1.4.16.模式匹配操作符

数据库提供了三种独立的实现模式匹配的方法：SQL LIKE 操作符、SIMILAR TO 操作符和 POSIX-风格的正则表达式。除了这些基本的操作符外，还有一些函数可用于提取或替换匹配子串并在匹配位置分离一个串。

❖ LIKE

描述：判断字符串是否能匹配上 LIKE 后的模式字符串。如果字符串与提供的模式匹配，则 LIKE 表达式返回为真（NOT LIKE 表达式返回假），否则返回为假（NOT LIKE 表达式返回真）。

匹配规则：

1、此操作符只有在它的模式匹配整个串的时候才能成功。如果要匹配在串内任何位置的序列，该模式必须以百分号开头和结尾。

2、下划线（_）代表（匹配）任何单个字符；百分号（%）代表任意串的通配符。

3、要匹配文本里的下划线或者百分号，在提供的模式里相应字符必须前导逃逸字符。逃逸字符的作用是禁用元字符的特殊含义，缺省的逃逸字符是反斜线，也可以用 ESCAPE 子句指定一个不同的逃逸字符。

4、要匹配逃逸字符本身，写两个逃逸字符。例如要写一个包含反斜线的模式常量，那你就要在 SQL 语句里写两个反斜线。

说明

参数 standard_conforming_strings 设置为 off 时，在文串常量中写的任何反斜线都需要被双写。因此，写一个匹配单个反斜线的模式实际上要在语句里写四个反斜线（你可以通过用 ESCAPE 选择一个不同的逃逸字符来避免这种情况，这样反斜线就不再是 LIKE 的特殊字符了。但仍然是字符文本分析器的特殊字符，所以你还是需要两个反斜线）。在兼容 MYSQL 数据模式时，您也可以通过写 ESCAPE ''的方式不选择逃逸字符，这样可以有效地禁用逃逸机制，但是没有办法关闭下划线和百分号在模式中的特殊含义。

5、关键字 ILIKE 可以用于替换 LIKE，区别是 LIKE 大小写敏感，ILIKE 大小写不敏感。

6、操作符~~等效于 LIKE，操作符~~*等效于 ILIKE。

示例：

```
panweidb=# SELECT 'abc' LIKE 'abc' AS RESULT;
 result
----------
 t
(1 row)
```

```
panweidb=# SELECT 'abc' LIKE 'a%' AS RESULT;
 result
----------
 t
(1 row)
```

```
panweidb=# SELECT 'abc' LIKE '_b_' AS RESULT;
 result
----------
 t
(1 row)
```

```
panweidb=# SELECT 'abc' LIKE 'c' AS RESULT;
 result
----------
 f
(1 row)
```

❖ SIMILAR TO

描述：SIMILAR TO 操作符根据自己的模式是否匹配给定串而返回真或者假。它和 LIKE 非常类似，只不过它使用 SQL 标准定义的正则表达式理解模式。

匹配规则：

1、和 LIKE 一样，此操作符只有在它的模式匹配整个串的时候才能成功。如果要匹配在串内任何位置的序列，该模式必须以百分号开头和结尾。

2、下划线（_）代表（匹配）任何单个字符；百分号（%）代表任意串的通配符。

3、SIMILAR TO 也支持下面这些从 POSIX 正则表达式借用的模式匹配元字符。

元字符	含义
|	表示选择（两个候选之一）
*	表示重复前面的项零次或更多次
+	表示重复前面的项一次或更多次
?	表示重复前面的项零次或一次
{m}	表示重复前面的项刚好 m 次
{m,}	表示重复前面的项 m 次或更多次
{m,n}	表示重复前面的项至少 m 次并且不超过 n 次
()	把多个项组合成一个逻辑项
[...]	声明一个字符类，就像 POSIX 正则表达式一样

4、前导逃逸字符可以禁止所有这些元字符的特殊含义。逃逸字符的使用规则和 LIKE 一样。

正则表达式函数：

支持使用函数 substring(string from pa...(参考：开发者指南->SQL 语法参考->函数与操作符->字符处理函数和操作符章节的 substring(string from pa...)截取匹配 SQL 正则表达式的子字符串。

示例：

```
panweidb=# SELECT 'abc' SIMILAR TO 'abc' AS RESULT;
 result
----------
```

```
 t
(1 row)
```

```
panweidb=# SELECT 'abc' SIMILAR TO 'a' AS RESULT;
 result
----------
 f
(1 row)
```

```
panweidb=# SELECT 'abc' SIMILAR TO '%(b|d)%' AS RESULT;
 result
----------
 t
(1 row)
```

```
panweidb=# SELECT 'abc' SIMILAR TO '(b|c)%'  AS RESULT;
 result
----------
 f
(1 row)
```

❖ POSIX 正则表达式

描述：正则表达式是一个字符序列，它是定义一个串集合（一个正则集）的缩写。如果一个串是正则表达式描述的正则集中的一员时，我们就说这个串匹配该正则表达式。 POSIX 正则表达式提供了比 LIKE 和 SIMILAR TO 操作符更强大的含义。表 1 列出了所有可用于 POSIX 正则表达式模式匹配的操作符。

表 1 正则表达式匹配操作符

操作符	描述	例子
~	匹配正则表达式，大小写敏感	'thomas' ~ '.thomas.'

操作符	描述	例子
~*	匹配正则表达式，大小写不敏感	'thomas' ~* '.Thomas.'
!~	不匹配正则表达式，大小写敏感	'thomas' !~ '.Thomas.'
!~*	不匹配正则表达式，大小写不敏感	'thomas' !~* '.vadim.'

匹配规则：

1、与 LIKE 不同，正则表达式允许匹配串里的任何位置，除非该正则表达式显式地挂接在串的开头或者结尾。

2、除了上文提到的元字符外， POSIX 正则表达式还支持下列模式匹配元字符。

正则表达式函数：

POSIX 正则表达式支持下面函数。

- substring(string from pa...(参考：开发者指南->SQL 语法参考->函数与操作符->字符处理函数和操作符章节的 substring(string from pa...)函数提供了抽取一个匹配 POSIX 正则表达式模式的子串的方法。
- regexp_count(string tex...(参考：开发者指南->SQL 语法参考->函数与操作符->字符处理函数和操作符章节的 regexp_count(string tex...)函数提供了获取匹配 POSIX 正则表达式模式的子串数量的功能。
- regexp_instr(string tex...(参考：开发者指南->SQL 语法参考->函数与操作符->字符处理函数和操作符章节的 regexp_instr(string tex...)函数提供了获取匹配 POSIX 正则表达式模式子串位置的功能。
- regexp_substr(string te...(参考：开发者指南->SQL 语法参考->函数与操作符->字符处理函数和操作符章节的 regexp_substr(string te...)函数提供了抽取一个匹配 POSIX 正则表达式模式的子串的方法。
- regexp_replace(string, p...(参考：开发者指南->SQL 语法参考->函数与操作符->字符处理函数和操作符章节的 regexp_replace(string, p...)函数提供了将匹配 POSIX 正则表达式模式的子串替换为新文本的功能。
- regexp_matches(string te...(参考：开发者指南->SQL 语法参考->函数与操作符->字符处理函数和操作符章节的 regexp_matches(string te...)函数

返回一个文本数组，该数组由匹配一个 POSIX 正则表达式模式得到的所有被捕获子串构成。

- regexp_split_to_table(st...(参考：开发者指南->SQL 语法参考->函数与操作符->字符处理函数和操作符章节的 regexp_split_to_table(st...)函数把一个 POSIX 正则表达式模式当作一个定界符来分离一个串。
- regexp_split_to_array(st...(参考：开发者指南->SQL 语法参考->函数与操作符->字符处理函数和操作符章节的 regexp_split_to_array(st...)和 regexp_split_to_table 类似，是一个正则表达式分离函数，不过它的结果以一个 text 数组的形式返回。

说明

正则表达式分离函数会忽略零长度的匹配，这种匹配发生在串的开头或结尾或者正好发生在前一个匹配之后。这和正则表达式匹配的严格定义是相悖的，后者由 regexp_matches 实现，但是通常前者是实际中最常用的行为。

示例：

```
panweidb=#  SELECT 'abc' ~ 'Abc' AS RESULT;
result
--------
 f
(1 row)
```

```
panweidb=# SELECT 'abc' ~* 'Abc' AS RESULT;
 result
--------
 t
(1 row)
```

```
panweidb=# SELECT 'abc' !~ 'Abc' AS RESULT;
 result
```

```
--------
 t
(1 row)
```

```
panweidb=# SELECT 'abc'!~* 'Abc' AS RESULT;
 result
--------
 f
(1 row)
```

```
panweidb=# SELECT 'abc' ~ '^a' AS RESULT;
 result
--------
 t
(1 row)
```

```
panweidb=# SELECT 'abc' ~ '(b|d)'AS RESULT;
 result
--------
 t
(1 row)
```

```
panweidb=# SELECT 'abc' ~ '^(b|c)'AS RESULT;
 result
--------
 f
(1 row)
```

虽然大部分的正则表达式搜索都能很快地执行，但是正则表达式仍可能被人为地弄成需要任意长的时间和任意量的内存进行处理。不建议从非安全模式来源接受正则表达式搜索模式，如果必须这样做，建议加上语句超时限制。使用

SIMILAR TO 模式的搜索具有同样的安全性危险，因为 SIMILAR TO 提供了很多和 POSIX-风格正则表达式相同的能力。LIKE 搜索比其他两种选项简单得多，因此在接受非安全模式来源搜索时要更安全些。

4.1.4.17.几何函数和操作符

几何操作符

- +

  描述：平移。

  示例：

```
panweidb=# SELECT box '((0,0),(1,1))' + point '(2.0,0)' AS RESULT;
result
-------------
 (3,1),(2,0)
(1 row)
```

- -

  描述：平移。

  示例：

```
panweidb=# SELECT box '((0,0),(1,1))' - point '(2.0,0)' AS RESULT;
 result
---------------
 (-1,1),(-2,0)
(1 row)
```

- *

  描述：伸展/旋转。

  示例：

```
panweidb=# SELECT box '((0,0),(1,1))' * point '(2.0,0)' AS RESULT;
result
-------------
 (2,2),(0,0)
(1 row)
```

- /

  描述：收缩/旋转。

  示例：

```
panweidb=# SELECT box '((0,0),(2,2))' / point '(2.0,0)' AS RESULT;
result
-------------
 (1,1),(0,0)
(1 row)
```

❖ #

描述：两个图形交面。

示例：

```
panweidb=# SELECT box '((1,-1),(-1,1))' # box '((1,1),(-2,-2))' AS RESULT;
 result
---------------
(1,1),(-1,-1)
(1 row)
```

❖ #

描述：图形的路径数目或多边形顶点数。

示例：

```
panweidb=# SELECT # path'((1,0),(0,1),(-1,0))' AS RESULT;
 result
--------
      3
(1 row)
```

❖ @-@

描述：图形的长度或者周长。

示例：

```
panweidb=# SELECT @-@ path '((0,0),(1,0))' AS RESULT;
 result
--------
      2
(1 row)
```

❖ @@

描述：图形的中心。

示例：

```
panweidb=# SELECT @@ circle '((0,0),10)' AS RESULT;
```

```
 result
--------
   (0,0)
(1 row)
```

❖ <->

描述：两个图形之间的距离。

示例：

```
panweidb=# SELECT circle '((0,0),1)' <-> circle '((5,0),1)' AS RESULT;
 result
--------
      3
(1 row)
```

❖ &&

描述：两个图形是否重叠（有一个共同点就为真）。

示例：

```
panweidb=# SELECT box '((0,0),(1,1))' && box '((0,0),(2,2))' AS RESULT;
 result
--------
t
(1 row)
```

❖ <<

描述：图形是否全部在另一个图形的左边（没有相同的横坐标）。

示例：

```
panweidb=# SELECT circle '((0,0),1)' << circle '((5,0),1)' AS RESULT;
 result
--------
   t
(1 row)
```

❖ >>

描述：图形是否全部在另一个图形的右边（没有相同的横坐标）。

示例：

```
panweidb=# SELECT circle '((5,0),1)' >> circle '((0,0),1)' AS RESULT;
 result
```

```
--------
 t
(1 row)
```

❖ &<

描述：图形的最右边是否不超过在另一个图形的最右边。

示例：

```
panweidb=# SELECT box '((0,0),(1,1))' &< box '((0,0),(2,2))' AS RESULT;
 result
--------
 t
(1 row)
```

❖ &>

描述：图形的最左边是否不超过在另一个图形的最左边。

示例：

```
panweidb=# SELECT box '((0,0),(3,3))' &> box '((0,0),(2,2))' AS RESULT;
 result
--------
 t
(1 row)
```

❖ <<|

描述：图形是否全部在另一个图形的下边（没有相同的纵坐标）。

示例：

```
panweidb=# SELECT box '((0,0),(3,3))' <<| box '((3,4),(5,5))' AS RESULT;
 result
--------
 t
(1 row)
```

❖ |>>

描述：图形是否全部在另一个图形的上边（没有相同的纵坐标）。

示例：

```
panweidb=# SELECT box '((3,4),(5,5))' |>> box '((0,0),(3,3))' AS RESULT;
 result
--------
```

```
 t
(1 row)
```

❖ &<|

描述：图形的最上边是否不超过另一个图形的最上边。

示例：

```
panweidb=# SELECT box '((0,0),(1,1))' &<| box '((0,0),(2,2))' AS RESULT;
 result
--------
 t
(1 row)
```

❖ |&>

描述：图形的最下边是否不超过另一个图形的最下边。

示例：

```
panweidb=# SELECT box '((0,0),(3,3))' |&> box '((0,0),(2,2))' AS RESULT;
 result
--------
 t
(1 row)
```

❖ <^

描述：图形是否低于另一个图形（允许两个图形有接触）。

示例：

```
panweidb=# SELECT box '((0,0),(-3,-3))' <^ box '((0,0),(2,2))' AS RESULT;
 result
--------
 t
 (1 row)
```

❖ >^

描述：图形是否高于另一个图形（允许两个图形有接触）。

示例：

```
panweidb=# SELECT box '((0,0),(2,2))' >^ box '((0,0),(-3,-3))'  AS RESULT;
 result
--------
 t
```

```
(1 row)
```

❖ ?#

描述：两个图形是否相交。

示例：

```
panweidb=# SELECT lseg '((-1,0),(1,0))' ?# box '((-2,-2),(2,2))' AS RESULT;
 result
--------
   t
(1 row)
```

❖ ?-

描述：图形是否处于水平位置。

示例：

```
panweidb=# SELECT ?- lseg '((-1,0),(1,0))' AS RESULT;
 result
--------
   t
(1 row)
```

❖ ?-

描述：图形是否水平对齐。

示例：

```
panweidb=# SELECT point '(1,0)' ?- point '(0,0)' AS RESULT;
 result
--------
   t
(1 row)
```

❖ ?|

描述：图形是否处于竖直位置。

示例：

```
panweidb=# SELECT ?| lseg '((-1,0),(1,0))' AS RESULT;
 result
--------
   f
(1 row)
```

❖ ?|

描述：图形是否竖直对齐。

示例：

```
panweidb=# SELECT point '(0,1)' ?| point '(0,0)' AS RESULT;
 result
--------
   t
(1 row)
```

❖ ?-|

描述：两条线是否垂直。

示例：

```
panweidb=# SELECT lseg '((0,0),(0,1))' ?-| lseg '((0,0),(1,0))' AS RESULT;
 result
--------
   t
(1 row)
```

❖ ?||

描述：两条线是否平行。

示例：

```
panweidb=# SELECT lseg '((-1,0),(1,0))' ?|| lseg '((-1,2),(1,2))' AS RESULT;
 result
--------
   t
(1 row)
```

❖ @>

描述：图形是否包含另一个图形。

示例：

```
panweidb=# SELECT circle '((0,0),2)' @> point '(1,1)' AS RESULT;
 result
--------
   t
(1 row)
```

❖ <@

描述：图形是否被包含于另一个图形。

示例：

```
 panweidb=# SELECT point '(1,1)' <@ circle '((0,0),2)' AS RESULT;
 result
--------
  t
(1 row)
```

❖ ~=

描述：两个图形是否相同。

示例：

```
panweidb=# SELECT polygon '((0,0),(1,1))' ~= polygon '((1,1),(0,0))' AS RESULT;
 result
--------
  t
 (1 row)
```

几何函数

❖ area(object)

描述：计算图形的面积。

返回类型：double precision

示例：

```
panweidb=# SELECT area(box '((0,0),(1,1))') AS RESULT;
 result
--------
      1
(1 row)
```

❖ center(object)

描述：计算图形的中心。

返回类型：point

示例：

```
panweidb=# SELECT center(box '((0,0),(1,2))') AS RESULT;
 result
---------
```

```
  (0.5,1)
(1 row)
```

❖ diameter(circle)

描述：计算圆的直径。

返回类型：double precision

示例：

```
 panweidb=# SELECT diameter(circle '((0,0),2.0)') AS RESULT;
 result
--------
      4
 (1 row)
```

❖ height(box)

描述：矩形的竖直高度。

返回类型：double precision

示例：

```
 panweidb=# SELECT height(box '((0,0),(1,1))') AS RESULT;
 result
--------
      1
 (1 row)
```

❖ isclosed(path)

描述：图形是否为闭合路径。

返回类型：Boolean

示例：

```
panweidb=# SELECT isclosed(path '((0,0),(1,1),(2,0))') AS RESULT;
 result
--------
  t
(1 row)
```

❖ isopen(path)

描述：图形是否为开放路径。

返回类型：Boolean

示例：

```
panweidb=# SELECT isopen(path '[(0,0),(1,1),(2,0)]') AS RESULT;
 result
--------
   t
(1 row)
```

- length(object)

  描述：计算图形的长度。

  返回类型：double precision

  示例：

```
panweidb=# SELECT length(path '((-1,0),(1,0))') AS RESULT;
 result
--------
      4
(1 row)
```

- npoints(path)

  描述：计算路径的顶点数。

  返回类型：int

  示例：

```
panweidb=# SELECT npoints(path '[(0,0),(1,1),(2,0)]') AS RESULT;
 result
--------
      3
(1 row)
```

- npoints(polygon)

  描述：计算多边形的顶点数。

  返回类型：int

  示例：

```
panweidb=# SELECT npoints(polygon '((1,1),(0,0))') AS RESULT;
 result
--------
      2
(1 row)
```

- pclose(path)

描述：把路径转换为闭合路径。

返回类型：path

示例：

```
panweidb=# SELECT pclose(path '[(0,0),(1,1),(2,0)]') AS RESULT;
   result
--------------------
   ((0,0),(1,1),(2,0))
(1 row)
```

❖ popen(path)

描述：把路径转换为开放路径。

返回类型：path

示例：

```
panweidb=# SELECT popen(path '((0,0),(1,1),(2,0))') AS RESULT;
   result
--------------------
   [(0,0),(1,1),(2,0)]
(1 row)
```

❖ radius(circle)

描述：计算圆的半径。

返回类型：double precision

示例：

```
panweidb=# SELECT radius(circle '((0,0),2.0)') AS RESULT;
 result
--------
      2
(1 row)
```

❖ width(box)

描述：计算矩形的水平尺寸。

返回类型：double precision

示例：

```
panweidb=# SELECT width(box '((0,0),(1,1))') AS RESULT;
 result
--------
```

```
      1
(1 row)
```

几何类型转换函数

- box(circle)

  描述：将圆转换成矩形

  返回类型：box

  示例：

```
panweidb=# SELECT box(circle '((0,0),2.0)') AS RESULT;
                        result
-----------------------------------------------------------------------
   (1.41421356237309,1.41421356237309),(-1.41421356237309,-1.4142135623730
9)
(1 row)
```

- box(point, point)

  描述：将点转换成矩形

  返回类型：box

  示例：

```
panweidb=# SELECT box(point '(0,0)', point '(1,1)') AS RESULT;
 result
-------------
   (1,1),(0,0)
(1 row)
```

- box(polygon)

  描述：将多边形转换成矩形

  返回类型：box

  示例：

```
panweidb=# SELECT box(polygon '((0,0),(1,1),(2,0))') AS RESULT;
 result
-------------
   (2,1),(0,0)
(1 row)
```

- circle(box)

描述：矩形转换成圆

返回类型：circle

示例：

```
panweidb=# SELECT circle(box '((0,0),(1,1))') AS RESULT;
      result
------------------------------
   <(0.5,0.5),0.707106781186548>
(1 row)
```

❖ circle(point, double precision)

描述：将圆心和半径转换成圆

返回类型：circle

示例：

```
panweidb=# SELECT circle(point '(0,0)', 2.0) AS RESULT;
result
----------
   <(0,0),2>
(1 row)
```

❖ circle(polygon)

描述：将多边形转换成圆

返回类型：circle

示例：

```
panweidb=# SELECT circle(polygon '((0,0),(1,1),(2,0))') AS RESULT;
          result
-----------------------------------------
   <(1,0.333333333333333),0.924950591148529>
(1 row)
```

❖ lseg(box)

描述：矩形对角线转化成线段

返回类型：lseg

示例：

```
panweidb=# SELECT lseg(box '((-1,0),(1,0))') AS RESULT;
  result
---------------
```

```
   [(1,0),(-1,0)]
(1 row)
```

❖ lseg(point, point)

描述：点转换成线段

返回类型：lseg

示例：

```
panweidb=# SELECT lseg(point '(-1,0)', point '(1,0)') AS RESULT;
  result
----------------
   [(-1,0),(1,0)]
(1 row)
```

❖ slope(point, point)

描述：计算两个点构成直线的斜率

返回类型: double

示例：

```
panweidb=# SELECT slope(point '(1,1)', point '(0,0)') AS RESULT;
 result
--------
      1
(1 row)
```

❖ path(polygon)

描述：多边形转换成路径

返回类型：path

示例：

```
panweidb=# SELECT path(polygon '((0,0),(1,1),(2,0))') AS RESULT;
   result
---------------------
   ((0,0),(1,1),(2,0))
(1 row)
```

❖ point(double precision, double precision)

描述：节点

返回类型：point

示例：

```
panweidb=# SELECT point(23.4, -44.5) AS RESULT;
 result
--------------
   (23.4,-44.5)
(1 row)
```

❖ point(box)

描述：矩形的中心

返回类型：point

示例：

```
panweidb=# SELECT point(box '((-1,0),(1,0))') AS RESULT;
 result
--------
   (0,0)
(1 row)
```

❖ point(circle)

描述：圆心

返回类型：point

示例：

```
  panweidb=# SELECT point(circle '((0,0),2.0)') AS RESULT;
 result
--------
   (0,0)
(1 row)
```

❖ point(lseg)

描述：线段的中心

返回类型：point

示例：

```
panweidb=# SELECT point(lseg '((-1,0),(1,0))') AS RESULT;
 result
--------
   (0,0)
(1 row)
```

❖ point(polygon)

描述：多边形的中心

返回类型：point

示例：

```
panweidb=# SELECT point(polygon '((0,0),(1,1),(2,0))') AS RESULT;
   result
-----------------------
  (1,0.333333333333333)
(1 row)
```

❖ polygon(box)

描述：矩形转换成 4 点多边形

返回类型：polygon

示例：

```
panweidb=# SELECT polygon(box '((0,0),(1,1))') AS RESULT;
    result
---------------------------
  ((0,0),(0,1),(1,1),(1,0))
(1 row)
```

❖ polygon(circle)

描述：圆转换成 12 点多边形

返回类型：polygon

示例：

```
panweidb=# SELECT polygon(circle '((0,0),2.0)') AS RESULT;
                                                                                result

----------------------------------------------------------------------------------------
----------------------------------------------------------------------------------------
----------------------------------------------------------------------------------------
----------------------------
  ((-2,0),(-1.73205080756888,1),(-1,1.73205080756888),(-1.22464679914735e-16,
2),(1,1.73205080756888),(1.73205080756888,1),(2,2.44929359829471e-16),(1.7320
5080756888,-0.999999999999999),(1,-1.73205080756888),(3.67394039744206e-16,
-2),(-0.999999999999999,-1.73205080756888),(-1.73205080756888,-1))
```

```
(1 row)
```

- polygon(npts, circle)

  描述：圆转换成 npts 点多边形

  返回类型：polygon

  示例：

```
panweidb=# SELECT polygon(12, circle '((0,0),2.0)') AS RESULT;
                                                                                        result
-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------
 ((-2,0),(-1.73205080756888,1),(-1,1.73205080756888),(-1.22464679914735e-16,2),(1,1.73205080756888),(1.73205080756888,1),(2,2.44929359829471e-16),(1.73205080756888,-0.999999999999999),(1,-1.73205080756888),(3.67394039744206e-16,-2),(-0.999999999999999,-1.73205080756888),(-1.73205080756888,-1))
(1 row)
```

- polygon(path)

  描述：路径转换成多边形

  返回类型：polygon

  示例：

```
panweidb=# SELECT polygon(path '((0,0),(1,1),(2,0))') AS RESULT;
   result
---------------------
 ((0,0),(1,1),(2,0))
(1 row)
```

4.1.4.18.网络地址函数和操作符

cidr 和 inet 操作符

操作符<<、<<=、>>、>>=对子网进行测试。它们只考虑两个地址的网络部分（忽略任何主机部分），然后判断其中一个网络是等于另外一个网络，还是另外一个网络的子网。

❖ <

描述：小于

示例：

```
panweidb=# SELECT inet '192.168.1.5' < inet '192.168.1.6' AS RESULT;
 result
--------
 t
(1 row)
```

❖ <=

描述：小于或等于

示例：

```
panweidb=# SELECT inet '192.168.1.5' <= inet '192.168.1.5' AS RESULT;
 result
--------
 t
(1 row)
```

❖ =

描述：等于

示例：

```
panweidb=# SELECT inet '192.168.1.5' = inet '192.168.1.5' AS RESULT;
 result
--------
 t
(1 row)
```

❖ >=

描述：大于或等于

示例：

```
panweidb=# SELECT inet '192.168.1.5' >= inet '192.168.1.5' AS RESULT;
 result
--------
 t
(1 row)
```

❖ >

描述：大于

示例：

```
panweidb=# SELECT inet '192.168.1.5' > inet '192.168.1.4' AS RESULT;
 result
--------
 t
(1 row)
```

❖ <>

描述：不等于

示例：

```
panweidb=# SELECT inet '192.168.1.5' <> inet '192.168.1.4' AS RESULT;
 result
--------
 t
(1 row)
```

❖ <<

描述：包含于

示例：

```
panweidb=# SELECT inet '192.168.1.5' << inet '192.168.1/24' AS RESULT;
 result
--------
 t
(1 row)
```

❖ <<=

描述：包含于或等于

示例：

```
panweidb=# SELECT inet '192.168.1/24' <<= inet '192.168.1/24' AS RESULT;
 result
--------
 t
(1 row)
```

❖ >>

描述：包含

示例：

```
panweidb=# SELECT inet '192.168.1/24' >> inet '192.168.1.5' AS RESULT;
 result
--------
 t
(1 row)
```

❖ >>=

描述：包含或等于

示例：

```
panweidb=# SELECT inet '192.168.1/24' >>= inet '192.168.1/24' AS RESULT;
 result
--------
 t
(1 row)
```

❖ ~

描述：位非

示例：

```
panweidb=# SELECT ~ inet '192.168.1.6' AS RESULT;
   result
---------------
63.87.254.249
(1 row)
```

❖ &

描述：两个网络地址的每一位都进行“与”操作

示例：

```
panweidb=# SELECT inet '192.168.1.6' & inet '10.0.0.0' AS RESULT;
 result
---------
 0.0.0.0
(1 row)
```

❖ |

描述：两个网络地址的每一位都进行“或”操作

示例：

```
panweidb=# SELECT inet '192.168.1.6' | inet '10.0.0.0' AS RESULT;
  result
-------------
 202.168.1.6
(1 row)
```

❖ +

描述：加

示例：

```
panweidb=# SELECT inet '192.168.1.6' + 25 AS RESULT;
   result
--------------
 192.168.1.31
(1 row)
```

❖ -

描述：减

示例：

```
panweidb=# SELECT inet '192.168.1.43' - 36 AS RESULT;
  result
-------------
 192.168.1.7
(1 row)
```

❖ -

描述：减

示例：

```
panweidb=# SELECT inet '192.168.1.43' - inet '192.168.1.19' AS RESULT;
 result
--------
   24
(1 row)
```

cidr 和 inet 函数

函数 abbrev、host、text 主要是为了提供可选的显示格式。

❖ abbrev(inet)

描述：缩写显示格式文本。

返回类型：text

示例：

```
panweidb=# SELECT abbrev(inet '10.1.0.0/16') AS RESULT;
  result
-------------
 10.1.0.0/16
(1 row)
```

❖ abbrev(cidr)

描述：缩写显示格式文本。

返回类型：text

示例：

```
panweidb=# SELECT abbrev(cidr '10.1.0.0/16') AS RESULT;
 result
---------
 10.1/16
(1 row)
```

❖ broadcast(inet)

描述：网络广播地址。

返回类型：inet

示例：

```
panweidb=# SELECT broadcast('192.168.1.5/24') AS RESULT;
     result
------------------
 192.168.1.255/24
(1 row)
```

❖ family(inet)

描述：抽取地址族，4 为 IPv4，6 为 IPv6。

返回类型：int

示例：

```
panweidb=# SELECT family('127.0.0.1') AS RESULT;
 result
--------
```

```
   4
(1 row)
```

❖ host(inet)

描述：将主机地址类型抽出为文本。

返回类型：text

示例：

```
panweidb=# SELECT host('192.168.1.5/24') AS RESULT;
  result
-------------
 192.168.1.5
(1 row)
```

❖ hostmask(inet)

描述：为网络构造主机掩码。

返回类型：inet

示例：

```
panweidb=# SELECT hostmask('192.168.23.20/30') AS RESULT;
result
---------
0.0.0.3
(1 row)
```

❖ masklen(inet)

描述：抽取子网掩码长度。

返回类型：int

示例：

```
panweidb=# SELECT masklen('192.168.1.5/24') AS RESULT;
 result
--------
   24
(1 row)
```

❖ netmask(inet)

描述：为网络构造子网掩码。

返回类型：inet

示例：

```
panweidb=# SELECT netmask('192.168.1.5/24') AS RESULT;
  result
---------------
 255.255.255.0
(1 row)
```

❖ network(inet)

描述：抽取地址的网络部分。

返回类型：cidr

示例：

```
panweidb=# SELECT network('192.168.1.5/24') AS RESULT;
  result
----------------
 192.168.1.0/24
(1 row)
```

❖ set_masklen(inet, int)

描述：为 inet 数值设置子网掩码长度。

返回类型：inet

示例：

```
panweidb=# SELECT set_masklen('192.168.1.5/24', 16) AS RESULT;
  result
----------------
 192.168.1.5/16
(1 row)
```

❖ set_masklen(cidr, int)

描述：为 cidr 数值设置子网掩码长度。

返回类型：cidr

示例：

```
panweidb=# SELECT set_masklen('192.168.1.0/24'::cidr, 16) AS RESULT;
  result
----------------
 192.168.0.0/16
(1 row)
```

❖ text(inet)

描述：把 IP 地址和掩码长度抽取为文本。

返回类型：text

示例：

```
panweidb=# SELECT text(inet '192.168.1.5') AS RESULT;
  result
----------------
 192.168.1.5/32
(1 row)
```

任何 cidr 值都能以显式或者隐式的方式转换为 inet 值，因此上述能够操作 inet 值的函数也同样能够操作 cidr 值。inet 值也可以转换为 cidr 值，此时 inet 子网掩码右侧的所有位都将转换为零，以创建一个有效的 cidr 值。另外，用户还可以使用常规的类型转换语法将一个文本字符串转换为 inet 或 cidr 值。例如：inet(expression)或 colname::cidr。

macaddr 函数

函数 trunc(macaddr)返回一个 MAC 地址，该地址的最后三个字节设置为零。

❖ trunc(macaddr)

描述：把后三个字节置为零。

返回类型：macaddr

示例：

```
panweidb=# SELECT trunc(macaddr '12:34:56:78:90:ab') AS RESULT;
    result
-------------------
 12:34:56:00:00:00
(1 row)
```

macaddr 类型还支持标准关系操作符（>、<=等）用于词法排序，和按位运算符（~、&和|）非、与和或。

4.1.4.19.HLL 函数和操作符

哈希函数

- hll_hash_boolean(bool)

  描述：对 bool 类型数据计算哈希值。

  返回值类型：hll_hashval

  示例：

```
panweidb=# SELECT hll_hash_boolean(FALSE);
 hll_hash_boolean
---------------------
 -5451962507482445012
(1 row)
```

- hll_hash_boolean(bool, int32)

  描述：设置 hash seed（即改变哈希策略）并对 bool 类型数据计算哈希值。

  返回值类型：hll_hashval

  示例：

```
panweidb=# SELECT hll_hash_boolean(FALSE, 10);
 hll_hash_boolean
--------------------
 -1169037589280886076
(1 row)
```

- hll_hash_smallint(smallint)

  描述：对 smallint 类型数据计算哈希值。

  返回值类型：hll_hashval

  示例：

```
panweidb=# SELECT hll_hash_smallint(100::smallint);
 hll_hash_smallint
---------------------
 962727970174027904
(1 row)
```

  说明：

  数值大小相同的参数使用不同数据类型的哈希函数计算，最后结果会不一样，因为不同类型哈希函数会选取不同的哈希计算策略。

- hll_hash_smallint(smallint, int32)

描述：设置 hash seed（即改变哈希策略）同时对 smallint 类型数据计算哈希值。

返回值类型：hll_hashval

示例：

```
vatbase=# SELECT hll_hash_smallint(100::smallint, 10);
 hll_hash_smallint
--------------------
 -9056177146160443041
(1 row)
```

❖ hll_hash_integer(integer)

描述：对 integer 类型数据计算哈希值。

返回值类型：hll_hashval

示例：

```
panweidb=# SELECT hll_hash_integer(0);
  hll_hash_integer
----------------------
 5156626420896634997
(1 row)
```

❖ hll_hash_integer(integer, int32)

描述：对 integer 类型数据计算哈希值，并设置 hashseed（即改变哈希策略）。

返回值类型：hll_hashval

示例：

```
panweidb=# SELECT hll_hash_integer(0, 10);
 hll_hash_integer
--------------------
 -5035020264353794276
(1 row)
```

❖ hll_hash_bigint(bigint)

描述：对 bigint 类型数据计算哈希值。

返回值类型：hll_hashval

示例：

```
panweidb=# SELECT hll_hash_bigint(100::bigint);
```

```
 hll_hash_bigint
---------------------
 -2401963681423227794
(1 row)
```

❖ hll_hash_bigint(bigint, int32)

描述：对 bigint 类型数据计算哈希值，并设置 hashseed（即改变哈希策略）。

返回值类型：hll_hashval

示例：

```
panweidb=# SELECT hll_hash_bigint(100::bigint, 10);
 hll_hash_bigint
---------------------
 -2305749404374433531
(1 row)
```

❖ hll_hash_bytea(bytea)

描述：对 bytea 类型数据计算哈希值。

返回值类型：hll_hashval

示例：

```
panweidb=# SELECT hll_hash_bytea(E'\x');
 hll_hash_bytea
----------------
 0
(1 row)
```

❖ hll_hash_bytea(bytea, int32)

描述：对 bytea 类型数据计算哈希值，并设置 hashseed（即改变哈希策略）。

返回值类型：hll_hashval

示例：

```
panweidb=# SELECT hll_hash_bytea(E'\x', 10);
 hll_hash_bytea
---------------------
 7233188113542599437
(1 row)
```

- hll_hash_text(text)

  描述：对 text 类型数据计算哈希值。

  返回值类型：hll_hashval

  示例：

```
panweidb=# SELECT hll_hash_text('AB');
  hll_hash_text
---------------------
 -5666002586880275174
(1 row)
```

- hll_hash_text(text, int32)

  描述：对 text 类型数据计算哈希值，并设置 hashseed（即改变哈希策略）。

  返回值类型：hll_hashval

  示例：

```
panweidb=# SELECT hll_hash_text('AB', 10);
hll_hash_text
--------------------
-2215507121143724132
(1 row)
```

- hll_hash_any(anytype)

  描述：对任意类型数据计算哈希值。

  返回值类型：hll_hashval

  示例：

```
panweidb=# select hll_hash_any(1);
   hll_hash_any
----------------------
 -1316670585935156930
(1 row)

panweidb=# select hll_hash_any('08:00:2b:01:02:03'::macaddr);
   hll_hash_any
----------------------
 -3719950434455589360
```

```
(1 row)
```

❖ hll_hash_any(anytype, int32)

描述：对任意类型数据计算哈希值，并设置 hashseed（即改变哈希策略）。

返回值类型：hll_hashval

示例：

```
panweidb=# select hll_hash_any(1, 10);
   hll_hash_any
----------------------
 7048553517657992351
(1 row)
```

❖ hll_hashval_eq(hll_hashval, hll_hashval)

描述：比较两个 hll_hashval 类型数据是否相等。

返回值类型：bool

示例：

```
panweidb=# select hll_hashval_eq(hll_hash_integer(1), hll_hash_integer(1));
 hll_hashval_eq
----------------
 t
(1 row)
```

❖ hll_hashval_ne(hll_hashval, hll_hashval)

描述：比较两个 hll_hashval 类型数据是否不相等。

返回值类型：bool

示例：

```
panweidb=# select hll_hashval_ne(hll_hash_integer(1), hll_hash_integer(1));
 hll_hashval_ne
----------------
 f
(1 row)
```

日志函数

hll 主要存在三种模式 Explicit、Sparse、Full。当数据规模比较小的时候会使用 Explicit 模式，这种模式下 distinct 值的计算是没有误差的；随着

distinct 值越来越多，hll 会先后转换为 Sparse 模式和 Full 模式，这两种模式在计算结果上没有任何区别，只影响 hll 函数的计算效率和 hll 对象的存储空间。下面的函数可以用于查看 hll 的一些参数。

- hll_print(hll)

  描述：打印 hll 的一些 debug 参数信息。

  示例：

```
panweidb=# select hll_print(hll_empty());
                          hll_print
------------------------------------------------------------------------------
 type=1(HLL_EMPTY), log2m=14, log2explicit=10, log2sparse=12, duplicatecheck=0
(1 row)
```

- hll_type(hll)

  描述：查看当前 hll 的类型。返回值具体含义如下：返回值 0，表示 HLL_UNINIT，未初始化的 hll 对象；返回值 1，表示 HLL_EMPTY，hll 空对象；返回值 2，表示 HLL_EXPLICIT，Explicit 模式的 hll 对象；返回值 3，表示 HLL_SPARSE，Sparse 模式的 hll 对象；返回值 4，表示 HLL_FULL，Full 模式的 hll 对象；返回值 5，表示 HLL_UNDEFINED，不合法的 hll 对象。

  示例：

```
panweidb=# select hll_type(hll_empty());
 hll_type
----------
        1
(1 row)
```

- hll_log2m(hll)

  描述：查看当前 hll 数据结构中的 log2m 数值，log2m 是分桶数的对数值，此值会影响最后 hll 计算 distinct 误差率，误差率计算公式为 ±1.04/√(2 ^ log2m)。当显式指定 log2m 的取值为 10-16 之间时，hll 会设置分桶数为 2log2m。当显示指定 log2explicit 为-1 时，会采用内置默认值。

  示例：

```
panweidb=# select hll_log2m(hll_empty());
```

```
 hll_log2m
-----------
      14
(1 row)

panweidb=# select hll_log2m(hll_empty(10));
 hll_log2m
-----------
    10
(1 row)

panweidb=# select hll_log2m(hll_empty(-1));
 hll_log2m
-----------
      14
(1 row)
```

- hll_log2explicit(hll)

  描述：查看当前 hll 数据结构中的 log2explicit 数值。hll 通常会由 Explicit 模式到 Sparse 模式再到 Full 模式，这个过程称为 promotion hierarchy 策略。可以通过调整 log2explicit 值的大小改变策略，比如 log2explicit 为 0 的时候就会跳过 Explicit 模式而直接进入 Sparse 模式。当显式指定 log2explicit 的取值为 1-12 之间时，hll 会在数据段长度超过 2log2explicit 时转为 Sparse 模式。当显示指定 log2explicit 为-1 时，会采用内置默认值。

  示例：

```
panweidb=# select hll_log2explicit(hll_empty());
 hll_log2explicit
------------------
          10
(1 row)

panweidb=# select hll_log2explicit(hll_empty(12, 8));
 hll_log2explicit
------------------
```

```
          8
(1 row)

panweidb=# select hll_log2explicit(hll_empty(12, -1));
 hll_log2explicit
------------------
          10
(1 row)
```

❖ hll_log2sparse(hll)

描述：查看当前 hll 数据结构中的 log2sparse 数值。hll 通常会由 Explicit 模式到 Sparse 模式再到 Full 模式，这个过程称为 promotion hierarchy 策略。可以通过调整 log2sparse 值的大小改变策略，比如 log2sparse 为 0 的时候就会跳过 Sparse 模式而直接进入 Full 模式。当显式指定 Sparse 的取值为 1-14 之间时，hll 会在数据段长度超过 2log2sparse 时转为 Full 模式。当显示指定 log2sparse 为-1 时，会采用内置默认值。

示例：

```
panweidb=# select hll_log2sparse(hll_empty());
 hll_log2sparse
----------------
         12
(1 row)

panweidb=# select hll_log2sparse(hll_empty(12, 8, 10));
 hll_log2sparse
----------------
         10
(1 row)

panweidb=# select hll_log2sparse(hll_empty(12, 8, -1));
 hll_log2sparse
----------------
         12
(1 row)
```

- hll_duplicatecheck(hll)

  描述：是否启用 duplicatecheck，0 是关闭，1 是开启。默认关闭，对于有较多重复值出现的情况，可以开启以提高效率。当显示指定 duplicatecheck 为-1 时，会采用内置默认值。

  示例：

```
panweidb=# select hll_duplicatecheck(hll_empty());
 hll_duplicatecheck
--------------------
                  0
(1 row)

panweidb=# select hll_duplicatecheck(hll_empty(12, 8, 10, 1));
 hll_duplicatecheck
--------------------
                  1
(1 row)

panweidb=# select hll_duplicatecheck(hll_empty(12, 8, 10, -1));
hll_duplicatecheck
--------------------
                  0
(1 row)
```

功能函数

- hll_empty()

  描述：创建一个空的 hll。

  返回值类型：hll

  示例：

```
panweidb=# select hll_empty();
                                hll_empty
------------------------------------------------------------------------
  x484c4c00000000002b050000000000000000000000000000000000000000000000
(1 row)
```

- hll_empty(int32 log2m)

描述：创建空的 hll 并指定参数 log2m，取值范围是 10 到 16。若输入-1，则采用内置默认值。

返回值类型：hll

示例：

```
panweidb=# select hll_empty(10);
                                                          hll_empty
------------------------------------------------------------------------------------------------------------
   x484c4c00000000002b040000000000000000000000000000000000000000
(1 row)

panweidb=# select hll_empty(-1);
                                                          hll_empty
------------------------------------------------------------------------------------------------------------
   x484c4c00000000002b050000000000000000000000000000000000000000
(1 row)
```

❖ hll_empty(int32 log2m, int32 log2explicit)

描述：创建空的 hll 并依次指定参数 log2m、log2explicit。log2explicit 取值范围是 0 到 12，0 表示直接跳过 Explicit 模式。该参数可以用来设置 Explicit 模式的阈值大小，在数据段长度达到 2log2explicit 后切换为 Sparse 模式或者 Full 模式。若输入-1，则 log2explicit 采用内置默认值。

返回值类型: hll

示例：

```
panweidb=# select hll_empty(10, 4);
                                                          hll_empty
------------------------------------------------------------------------------------------------------------
   x484c4c0000000000130400000000000000000000000000000000000000000
(1 row)

panweidb=# select hll_empty(10, -1);
                                                          hll_empty
------------------------------------------------------------------------------------------------------------
   x484c4c00000000002b040000000000000000000000000000000000000000
```

```
(1 row)
```

- hll_empty(int32 log2m, int32 log2explicit, int64 log2sparse)

  描述：创建空的 hll 并依次指定参数 log2m、log2explicit、log2sparse。log2sparse 取值范围是 0 到 14，0 表示直接跳过 Sparse 模式。该参数可以用来设置 Sparse 模式的阈值大小，在数据段长度达到 2log2sparse 后切换为 Full 模式。若输入-1，则 log2sparse 采用内置默认值。

  返回值类型：hll

  示例：

```
panweidb=# select hll_empty(10, 4, 8);
                                                          hll_empty
------------------------------------------------------------------------------------------------------------
   x484c4c0000000000120400000000000000000000000000000000000000000000
(1 row)

panweidb=# select hll_empty(10, 4, -1);
                                                          hll_empty
------------------------------------------------------------------------------------------------------------
   x484c4c0000000000130400000000000000000000000000000000000000000000
(1 row)
```

- hll_empty(int32 log2m, int32 log2explicit, int64 log2sparse, int32 duplicatecheck)

  描述：创建空的 hll 并依次指定参数 log2m、log2explicit、log2sparse、duplicatecheck。duplicatecheck 取 0 或者 1，表示是否开启该模式，默认情况下该模式会关闭。若输入-1，则 duplicatecheck 采用内置默认值。

  返回值类型：hll

  示例：

```
panweidb=# select hll_empty(10, 4, 8, 0);
                                                          hll_empty
------------------------------------------------------------------------------------------------------------
   x484c4c0000000000120400000000000000000000000000000000000000000000
(1 row)
```

```
panweidb=# select hll_empty(10, 4, 8, -1);
                                  hll_empty
------------------------------------------------------------------------
 x484c4c0000000000120400000000000000000000000000000000000000000000
(1 row)
```

❖ hll_add(hll, hll_hashval)

描述：把 hll_hashval 加入到 hll 中。

返回值类型：hll

示例：

```
panweidb=# select hll_add(hll_empty(), hll_hash_integer(1));
                                                                  hll_add
------------------------------------------------------------------------------------------------------------------------
 x484c4c08000002002b0900000000000000f03f3e2921ff133fbaed3e2921ff133fbaed00
(1 row)
```

❖ hll_add_rev(hll_hashval, hll)

描述：把 hll_hashval 加入到 hll 中，和 hll_add 功能一样，只是参数位置进行了交换。

返回值类型：hll

示例：

```
panweidb=# select hll_add_rev(hll_hash_integer(1), hll_empty());
                                                                  hll_add_rev
------------------------------------------------------------------------------------------------------------------------
 x484c4c08000002002b0900000000000000f03f3e2921ff133fbaed3e2921ff133fbaed00
(1 row)
```

❖ hll_eq(hll, hll)

描述：比较两个 hll 是否相等。

返回值类型：bool

示例：

```
panweidb=# select hll_eq(hll_add(hll_empty(), hll_hash_integer(1)), hll_add(hll_empty(), hll_hash_integer(2)));
```

```
hll_eq
--------
f
(1 row)
```

❖ hll_ne(hll, hll)

描述：比较两个 hll 是否不相等。

返回值类型：bool

示例：

```
panweidb=# select hll_ne(hll_add(hll_empty(), hll_hash_integer(1)), hll_add(hll_e
mpty(), hll_hash_integer(2)));
 hll_ne
--------
t
(1 row)
```

❖ hll_cardinality(hll)

描述：计算 hll 的 distinct 值。

返回值类型：int

示例：

```
panweidb=# select hll_cardinality(hll_empty() || hll_hash_integer(1));
 hll_cardinality
-----------------
               1
(1 row)
```

❖ hll_union(hll, hll)

描述：把两个 hll 数据结构 union 成一个。

返回值类型：hll

示例：

```
panweidb=# select hll_union(hll_add(hll_empty(), hll_hash_integer(1)), hll_add(hll
_empty(), hll_hash_integer(2)));
                                hll_union
-----------------------------------------------------------------------------------
---
```

```
 x484c4c10002000002b0900000000000000040000000000000000b3ccc49320cc
a1ae3e2921ff133fbaed00
(1 row)
```

聚合函数

- hll_add_agg(hll_hashval)

  描述：把哈希后的数据按照分组放到 hll 中。

  返回值类型：hll

  示例：

```
--准备数据
panweidb=# create table t_id(id int);
panweidb=# insert into t_id values(generate_series(1,500));
panweidb=# create table t_data(a int, c text);
panweidb=# insert into t_data select mod(id,2), id from t_id;

--创建表并指定列为 hll
panweidb=# create table t_a_c_hll(a int, c hll);

 --根据 a 列 group by 对数据分组，把各组数据加到 hll 中
panweidb=# insert into t_a_c_hll select a, hll_add_agg(hll_hash_text(c)) from t_dat
a group by a;

--得到每组数据中 hll 的 Distinct 值
panweidb=# select a, #c as cardinality from t_a_c_hll order by a;
 a |  cardinality
---+------------------
 0 | 247.862354346299
 1 | 250.908710610377
(2 rows)
```

- hll_add_agg(hll_hashval, int32 log2m)

  描述：把哈希后的数据按照分组放到 hll 中，并指定参数 log2m，取值范围是 10 到 16。若输入-1 或者 NULL，则采用内置默认值。

  返回值类型：hll

  示例：

```
panweidb=# select hll_cardinality(hll_add_agg(hll_hash_text(c), 12)) from t_data;
   hll_cardinality
  -----------------
   497.965240179228
  (1 row)
```

- hll_add_agg(hll_hashval, int32 log2m, int32 log2explicit)

  描述：把哈希后的数据按照分组放到 hll 中，依次指定参数 log2m、log2explicit。log2explicit 取值范围是 0 到 12，0 表示直接跳过 Explicit 模式。该参数可以用来设置 Explicit 模式的阈值大小，在数据段长度达到 2log2explicit 后切换为 Sparse 模式或者 Full 模式。若输入-1 或者 NULL，则 log2explicit 采用内置默认值。

  返回值类型：hll

  示例：

```
panweidb=# select hll_cardinality(hll_add_agg(hll_hash_text(c), NULL, 1)) from t_d
ata;
   hll_cardinality
  -----------------
   498.496062953313
  (1 row)
```

- hll_add_agg(hll_hashval, int32 log2m, int32 log2explicit, int64 log2sparse)

  描述：把哈希后的数据按照分组放到 hll 中，依次指定参数 log2m、log2explicit、log2sparse。log2sparse 取值范围是 0 到 14，0 表示直接跳过 Sparse 模式。该参数可以用来设置 Sparse 模式的阈值大小，在数据段长度达到 2log2sparse 后切换为 Full 模式。若输入-1 或者 NULL，则 log2sparse 采用内置默认值。

  返回值类型：hll

  示例：

```
panweidb=# select hll_cardinality(hll_add_agg(hll_hash_text(c), NULL, 6, 10)) from
t_data;
   hll_cardinality
  -----------------
   498.496062953313
```

```
 (1 row)
```

- hll_add_agg(hll_hashval, int32 log2m, int32 log2explicit, int64 log2sparse, int32 duplicatecheck)

  描述：把哈希后的数据按照分组放到 hll 中，依次制定参数 log2m、log2explicit、log2sparse、duplicatecheck，duplicatecheck 取值范围是 0 或者 1，表示是否开启该模式，默认情况下该模式会关闭。若输入-1 或者 NULL，则 duplicatecheck 采用内置默认值。

  返回值类型：hll

  示例：

```
panweidb=# select hll_cardinality(hll_add_agg(hll_hash_text(c), NULL, 6, 10, -1)) fr
om t_data;
   hll_cardinality
 ------------------
  498.496062953313
 (1 row)
```

- hll_union_agg(hll)

  描述：将多个 hll 类型数据 union 成一个 hll。

  返回值类型：hll

  示例：

```
 --将各组中的 hll 数据 union 成一个 hll，并计算 distinct 值。
panweidb=# select #hll_union_agg(c) as cardinality from t_a_c_hll;
    cardinality
 ------------------
  498.496062953313
 (1 row)
```

说明

注意：当两个或者多个 hll 数据结构做 union 的时候，必须要保证其中每一个 hll 里面的精度参数一样，否则将不可以进行 union。同样的约束也适用于函数 hll_union(hll,hll)。

废弃函数

由于版本升级，HLL（HyperLogLog）有一些旧的函数废弃，用户可以用类似的函数进行替代。

- hll_schema_version(hll)

  描述：查看当前 hll 中的 schema version。旧版本 schema version 是常值 1，用来进行 hll 字段的头部校验，重构后的 hll 在头部增加字段“HLL”进行校验，schema version 不再使用。

- hll_regwidth(hll)

  描述：查看 hll 数据结构中桶的位数大小。旧版本桶的位数 regwidth 取值 1~5，会存在较大的误差，也限制了基数估计上限。 重构后 regwidth 为固定值 6，不再使用 regwidth 变量。

- hll_expthresh(hll)

  描述：得到当前 hll 中 expthresh 大小。采用 hll_log2explicit(hll)替代类似功能。

- hll_sparseon(hll)

  描述：是否启用 Sparse 模式。采用 hll_log2sparse(hll)替代类似功能，0 表示关闭 Sparse 模式。

内置函数

HLL（HyperLogLog）有一系列内置函数用于内部对数据进行处理，一般情况下用户不需要熟知这些函数的使用。详情见表 1。

表 1 内置函数

函数名称	功能描述
hll_in	以 string 格式接收 hll 数据。
hll_out	以 string 格式发送 hll 数据。
hll_recv	以 bytea 格式接收 hll 数据。
hll_send	以 bytea 格式发送 hll 数据。
hll_trans_in	以 string 格式接收 hll_trans_type 数据。
hll_trans_out	以 string 格式发送 hll_trans_type 数据。

函数名称	功能描述
hll_trans_recv	以 bytea 形式接收 hll_trans_type 数据。
hll_trans_send	以 bytea 形式发送 hll_trans_type 数据。
hll_typmod_in	接收 typmod 类型数据。
hll_typmod_out	发送 typmod 类型数据。
hll_hashval_in	接收 hll_hashval 类型数据。
hll_hashval_out	发送 hll_hashval 类型数据。
hll_add_trans0	类似于 hll_add 所提供的功能，初始化时无指定入参，通常在聚合运算的第一阶段 DN 上使用。
hll_add_trans1	类似于 hll_add 所提供的功能，初始化时指定一个入参，通常在聚合运算的第一阶段 DN 上使用。
hll_add_trans2	类似于 hll_add 所提供的功能，初始化时指定两个入参，通常在聚合运算的第一阶段 DN 上使用。
hll_add_trans3	类似于 hll_add 所提供的功能，初始化时指定三个入参，通常在聚合运算的第一阶段 DN 上使用。
hll_add_trans4	类似于 hll_add 所提供的功能，初始化时指定四个入参，通常在聚合运算的第一阶段 DN 上使用。
hll_union_trans	类似 hll_union 所提供的功能，在聚合运算的第一阶段 DN 上使用。
hll_union_collect	类似于 hll_union 所提供的功能，在聚合运算第二阶段 DN 上使用，汇总各个 DN 上的结果。
hll_pack	在聚合运算第三阶段 DN 上使用，把自定义 hll_trans_type 类型最后转换成 hll 类型。
hll	用于 hll 类型转换成 hll 类型，根据输入参数会设定指定参数。
hll_hashval	用于 bigint 类型转换成 hll_hashval 类型。

函数名称	功能描述
hll_hashval_int4	用于 int4 类型转换成 hll_hashval 类型。

操作符

❖ =

描述：比较 hll 或 hll_hashval 的值是否相等。

返回值类型：bool

示例：

```
 --hll
panweidb=# select (hll_empty() || hll_hash_integer(1)) = (hll_empty() || hll_hash_inte
ger(1));
 column
----------
 t
(1 row)

 --hll_hashval
panweidb=# select hll_hash_integer(1) = hll_hash_integer(1);
 ?column?
----------
 t
(1 row)
```

❖ <> or !=

描述：比较 hll 或 hll_hashval 是否不相等。

返回值类型：bool

示例：

```
 --hll
panweidb=# select (hll_empty() || hll_hash_integer(1)) <> (hll_empty() || hll_hash_int
eger(2));
 ?column?
----------
 t
(1 row)
```

```
  --hll_hashval
panweidb=# select hll_hash_integer(1) <> hll_hash_integer(2);
  ?column?
  ----------
   t
  (1 row)
```

❖ ||

描述：可代表 hll_add、hll_union、hll_add_rev 三个函数的功能。

返回值类型：hll

示例：

```
  --hll_add
panweidb=# select hll_empty() || hll_hash_integer(1);
                        ?column?
  ------------------------------------------------------------------------------
   x484c4c08000002002b0900000000000000f03f3e2921ff133fbaed3e2921ff133fba
ed00
  (1 row)

  --hll_add_rev
panweidb=# select hll_hash_integer(1) || hll_empty();
                        ?column?
  ------------------------------------------------------------------------------
   x484c4c08000002002b0900000000000000f03f3e2921ff133fbaed3e2921ff133fba
ed00
  (1 row)

  --hll_union
panweidb=# select (hll_empty() || hll_hash_integer(1)) || (hll_empty() || hll_hash_inte
ger(2));
                              ?column?
  ------------------------------------------------------------------------------
-----
```

```
 x484c4c10002000002b0900000000000000040000000000000000b3ccc49320cc
a1ae3e2921ff133fbaed00
(1 row)
```

❖ #

描述：计算出 hll 的 Dintinct 值, 同 hll_cardinality 函数。

返回值类型：int

示例：

```
panweidb=# select #(hll_empty() || hll_hash_integer(1));
 ?column?
----------
        1
(1 row)
```

4.1.4.20.SEQUENCE 函数

序列函数为用户从序列对象中获取后续的序列值提供了简单的多用户安全的方法。

❖ nextval(regclass)

描述：递增序列并返回新值。

📖 说明

为了避免从同一个序列获取值的并发事务被阻塞，nextval 操作不会回滚；也就是说，一旦一个值已经被抓取，那么就认为它已经被用过了，并且不会再被返回。即使该操作处于事务中，当事务之后中断，或者如果调用查询结束不使用该值，也是如此。这种情况将在指定值的顺序中留下未使用的“空洞”。因此，PanWeiDB 序列对象不能用于获得“无间隙”序列。

⚠ 须知：

nextval 函数只能在主机上执行，备机不支持执行此函数。

返回类型：numeric

---

nextval 函数有两种调用方式（其中第二种调用方式目前不支持 Sequence 命名中有特殊字符"."的情况），如下：

示例 1：

```
panweidb=# select nextval('seqDemo');
nextval
---------
    2
(1 row)
```

示例 2：

```
panweidb=# select seqDemo.nextval;
 nextval
---------
    2
(1 row)
```

❖ currval(regclass)

返回当前会话里最近一次 nextval 返回的指定的 sequence 的数值。如果当前会话还没有调用过指定的 sequence 的 nextval，那么调用 currval 将会报错。

返回类型：numeric

currval 函数有两种调用方式（其中第二种调用方式目前不支持 Sequence 命名中有特殊字符"."的情况），如下：

示例 1：

```
panweidb=# select currval('seq1');
 currval
---------
    2
(1 row)
```

示例 2：

```
panweidb=# select seq1.currval;
 currval
---------
```

```
    2
(1 row)
```

❖ lastval()

描述：返回当前会话里最近一次 nextval 返回的数值。这个函数等效于 currval，只是它不用序列名为参数，它抓取当前会话里面最近一次 nextval 使用的序列。如果当前会话还没有调用过 nextval，那么调用 lastval 将会报错。

返回类型：numeric

示例：

```
panweidb=# select lastval();
 lastval
---------
    2
(1 row)
```

❖ setval(regclass,numeric)

描述：设置序列的当前数值。

返回类型：numeric

示例：

```
panweidb=# select setval('seqDemo',1);
 setval
--------
    1
(1 row)
```

❖ setval(regclass, numeric, Boolean)

描述：设置序列的当前数值以及 is_called 标志。

返回类型：numeric

示例：

```
panweidb=# select setval('seqDemo',1,true);
 setval
--------
    1
(1 row)
```

📖 说明

Setval 后当前会话会立刻生效，但如果其他会话有缓存的序列值，只能等到缓存值用尽才能感知 Setval 的作用。所以为了避免序列值冲突，setval 要谨慎使用。

因为序列是非事务的，setval 造成的改变不会由于事务的回滚而撤销。

⚠ 须知：

nextval 函数只能在主机上执行，备机不支持执行此函数。

- pg_sequence_last_value(sequence_oid oid, OUT cache_value int16, OUT last_value int16)
  描述：获取指定 sequence 的参数，包含缓存值，当前值。
  返回类型：int16，int16

4.1.4.21.范围函数和操作符

范围操作符

- =
  描述：等于
  示例：

```
panweidb=# SELECT int4range(1,5) = '[1,4]'::int4range AS RESULT;
 result
--------
 t
(1 row)
```

- <>
  描述：不等于
  示例：

```
panweidb=# SELECT numrange(1.1,2.2) <> numrange(1.1,2.3) AS RESULT;
 result
--------
```

```
t
(1 row)
```

❖ <

描述：小于

示例：

```
panweidb=# SELECT int4range(1,10) < int4range(2,3) AS RESULT;
 result
--------
 t
(1 row)
```

❖ >

描述：大于

示例：

```
panweidb=# SELECT int4range(1,10) > int4range(1,5) AS RESULT;
 result
--------
 t
(1 row)
```

❖ <=

描述：小于或等于

示例：

```
panweidb=# SELECT numrange(1.1,2.2) <= numrange(1.1,2.2) AS RESULT;
 result
--------
 t
(1 row)
```

❖ >=

描述：大于或等于

示例：

```
panweidb=# SELECT numrange(1.1,2.2) >= numrange(1.1,2.0) AS RESULT;
 result
--------
 t
```

```
(1 row)
```

❖ @>

描述：包含范围

示例：

```
panweidb=# SELECT int4range(2,4) @> int4range(2,3) AS RESULT;
 result
--------
 t
(1 row)
```

❖ @>

描述：包含元素

示例：

```
panweidb=# SELECT '[2011-01-01,2011-03-01)'::tsrange @> '2011-01-10'::timesta
mp AS RESULT;
 result
--------
 t
(1 row)
```

❖ <@

描述：范围包含于

示例：

```
panweidb=# SELECT int4range(2,4) <@ int4range(1,7) AS RESULT;
 result
--------
 t
(1 row)
```

❖ <@

描述：元素包含于

示例：

```
panweidb=# SELECT 42 <@ int4range(1,7) AS RESULT;
 result
--------
 f
```

```
(1 row)
```

- &&

  描述：重叠（有共同点）

  示例：

```
panweidb=# SELECT int8range(3,7) && int8range(4,12) AS RESULT;
 result
--------
 t
(1 row)
```

- <<

  描述：范围值是否比另一个范围值的最小值还小（没有交集）。

  示例：

```
panweidb=# SELECT int8range(1,10) << int8range(100,110) AS RESULT;
 result
--------
 t
(1 row)
```

- >>

  描述：范围值是否比另一个范围值的最大值还大（没有交集）。

  示例：

```
panweidb=# SELECT int8range(50,60) >> int8range(20,30) AS RESULT;
 result
--------
 t
(1 row)
```

- &<

  描述：范围值的最大值是否不超过另一个范围值的最大值。

  示例：

```
panweidb=# SELECT int8range(1,20) &< int8range(18,20) AS RESULT;
 result
--------
 t
(1 row)
```

❖ &>

描述：范围值的最小值是否不小于另一个范围值的最小值。

示例：

```
panweidb=# SELECT int8range(7,20) &> int8range(5,10) AS RESULT;
 result
--------
 t
(1 row)
```

❖ -|-

描述：相邻

示例：

```
panweidb=# SELECT numrange(1.1,2.2) -|- numrange(2.2,3.3) AS RESULT;
 result
--------
 t
(1 row)
```

❖ +

描述：并集

示例：

```
panweidb=# SELECT numrange(5,15) + numrange(10,20) AS RESULT;
 result
--------
 [5,20)
(1 row)
```

❖ *

描述：交集

示例：

```
panweidb=# SELECT int8range(5,15) * int8range(10,20) AS RESULT;
  result
---------
 [10,15)
(1 row)
```

❖ -

描述：差集

示例：

```
panweidb=# SELECT int8range(5,15) - int8range(10,20) AS RESULT;
 result
--------
 [5,10)
(1 row)
```

简单的比较操作符<、>、<=和>=先比较下界，只有下界相等时才比较上界。

<<、>>和-|-操作符当包含空范围时也会返回 false；也就是，不认为空范围在其他范围之前或之后。

并集和差集操作符的执行结果无法包含两个不相交的子范围。

范围函数

- numrange(numeric, numeric, [text])

  描述：表示一个范围。

  返回类型：范围元素类型

  示例：

```
panweidb=# SELECT numrange(1.1,2.2) AS RESULT;
 result
--------
[1.1,2.2)
(1 row)
panweidb=# SELECT numrange(1.1,2.2, '()') AS RESULT;
 result
--------
(1.1,2.2)
(1 row)
```

- lower(anyrange)

  描述：范围的下界。

  返回类型：范围元素类型

  示例：

```
panweidb=# SELECT lower(numrange(1.1,2.2)) AS RESULT;
```

```
 result
--------
   1.1
(1 row)
```

❖ upper(anyrange)

描述：范围的上界。

返回类型：范围元素类型

示例：

```
panweidb=# SELECT upper(numrange(1.1,2.2)) AS RESULT;
 result
--------
  2.2
(1 row)
```

❖ isempty(anyrange)

描述：范围是否为空。

返回类型：Boolean

示例：

```
panweidb=# SELECT isempty(numrange(1.1,2.2)) AS RESULT;
 result
--------
 f
(1 row)
```

❖ lower_inc(anyrange)

描述：是否包含下界。

返回类型：Boolean

示例：

```
panweidb=# SELECT lower_inc(numrange(1.1,2.2)) AS RESULT;
 result
--------
 t
(1 row)
```

❖ upper_inc(anyrange)

描述：是否包含上界。

返回类型：Boolean

示例：

```
panweidb=# SELECT upper_inc(numrange(1.1,2.2)) AS RESULT;
 result
--------
 f
(1 row)
```

❖ lower_inf(anyrange)

描述：下界是否为无穷。

返回类型：Boolean

示例：

```
panweidb=# SELECT lower_inf('(,)'::daterange) AS RESULT;
 result
--------
 t
(1 row)
```

❖ upper_inf(anyrange)

描述：上界是否为无穷。

返回类型：Boolean

示例：

```
panweidb=# SELECT upper_inf('(,)'::daterange) AS RESULT;
 result
--------
 t
(1 row)
```

如果范围是空或者需要的界限是无穷的，lower 和 upper 函数将返回 null。lower_inc、upper_inc、lower_inf 和 upper_inf 函数均对空范围返回 false。

❖ elem_contained_by_range(anyelement, anyrange)

描述：判断元素是否在范围内。

返回类型：Boolean

示例：

```
panweidb=# SELECT elem_contained_by_range('2', numrange(1.1,2.2));
 elem_contained_by_range
-------------------------
 t
(1 row)
```

4.1.4.22.窗口函数

窗口函数

窗口函数用于给组内的值生成序号。窗口函数与 OVER 语句一起使用。OVER 语句用于对数据进行分组，并对组内元素进行排序。

【说明】
窗口函数中的 order by 后面必须跟字段名，若 order by 后面跟数字，该数字会被按照常量处理，因此对目标列没有起到排序的作用。
列存表目前只支持[rank(expression)](#RANK)和[row_number(expression)](#ROW_NUMBER)两个函数。

为方便后续函数的功能演示创建测试表并插入数据。

```
CREATE TABLE student_score (
id VARCHAR ( 10 ) NOT NULL,
name VARCHAR ( 20 ) NOT NULL,
score NUMBER ( 32 ) NOT NULL );
```

```
insert into student_score(id,name,score) values(1,'赵子龙',95);
insert into student_score(id,name,score) values(2,'马超',80);
insert into student_score(id,name,score) values(3,'关羽',95);
insert into student_score(id,name,score) values(1,'诸葛亮',100);
insert into student_score(id,name,score) values(2,'阿斗',60);
insert into student_score(id,name,score) values(3,'刘备',80);

SELECT * FROM student_score;
```

返回结果为：

```
id | name |score
----+--------+-------
1 |赵子龙|  95
2 |马超  |  80
3 |关羽  |  95
1 |诸葛亮|  100
2 |阿斗  |  60
3 |刘备  |  80
(6 rows)
```

- **RANK()**

  描述：RANK 函数为各组内值生成跳跃排序序号，其中，相同的值具有相同序号。

  返回值类型：BIGINT

  示例：

```
SELECT ss.ID,ss.NAME,ss.SCORE,RANK ( ) OVER ( ORDER BY ss.SCORE DESC ) RANK FROM STUDENT_SCORE ss;
```

  返回结果如下：

```
 id | name |score|rank
 ----+--------+-------+------
 1 |诸葛亮|  100|  1
 1 |赵子龙|  95|  2
 3 |关羽  |  95|  2
 2 |马超  |  80|  4
 3 |刘备  |  80|  4
 2 |阿斗  |  60|  6
 (6 rows)
```

- **ROW_NUMBER()**

  描述：ROW_NUMBER 函数为各组内值生成连续排序序号，其中，相同的值其序号也不相同。

  返回值类型：BIGINT

示例：

```
SELECT ss.ID,ss.NAME,ss.SCORE,ROW_NUMBER ( ) OVER ( ORDER BY ss.SCORE D
ESC ) ROW_NUMBER FROM STUDENT_SCORE ss;
```

返回结果如下：

```
 id | name  |score|row_number
----+--------+-------+------------
1 |诸葛亮 |  100|       1
1 |赵子龙 |   95|       2
3 |关羽   |   95|       3
2 |马超   |   80|       4
3 |刘备   |   80|       5
2 |阿斗   |   60|       6
(6 rows)
```

- **DENSE_RANK()**

描述：DENSE_RANK 函数为各组内值生成连续排序序号，其中，相同的值具有相同序号。

返回值类型：BIGINT

示例：

```
 SELECT ss.ID,ss.NAME,ss.SCORE,dense_rank() OVER ( ORDER BY ss.SCORE DE
SC ) dense_rank FROM STUDENT_SCORE ss;
```

返回结果如下：

```
 id | name  |score|dense_rank
----+--------+-------+------------
1 |诸葛亮 |  100|       1
1 |赵子龙 |   95|       2
3 |关羽   |   95|       2
2 |马超   |   80|       3
3 |刘备   |   80|       3
2 |阿斗   |   60|       4
(6 rows)
```

- **PERCENT_RANK()**

描述：PERCENT_RANK 函数为各组内对应值生成相对序号，即根据公式`

(rank - 1) / (total rows - 1)`计算所得的值。其中 rank 为该值依据 RANK 函数所生成的对应序号，totalrows 为该分组内的总元素个数。

返回值类型：DOUBLE PRECISION

示例：

```
SELECT *, percent_rank() OVER (PARTITION BY id ORDER BY score DESC) percent_rank FROM STUDENT_SCORE;
```

返回结果如下：

```
 id | name  | score | percent_rank
----+-------+-------+--------------
 1  | 诸葛亮 |  100 |            0
 1  | 赵子龙 |   95 |            1
 2  | 马超   |   80 |            0
 2  | 阿斗   |   60 |            1
 3  | 关羽   |   95 |            0
 3  | 刘备   |   80 |            1
(6 rows)
```

❖ **CUME_DIST()**

描述：CUME_DIST 函数为各组内对应值生成累积分布序号。即根据公式(小于等于当前值的数据行数)/(该分组总行数 totalrows)计算所得的相对序号。

返回值类型：DOUBLE PRECISION

示例：

```
SELECT *, cume_dist() OVER (PARTITION BY id ORDER BY score DESC) cume_dist FROM STUDENT_SCORE;
```

返回结果如下：

```
 id | name  | score | cume_dist
----+-------+-------+-----------
 1  | 诸葛亮 |  100 |        .5
 1  | 赵子龙 |   95 |         1
 2  | 马超   |   80 |        .5
 2  | 阿斗   |   60 |         1
 3  | 关羽   |   95 |        .5
```

```
 3 |刘备  |  80|      1
(6 rows)
```

❖ **NTILE(num_buckets integer)**

描述：NTILE 函数根据 num_buckets integer 将有序的数据集合平均分配到 num_buckets 所指定数量的桶中，并将桶号分配给每一行。分配时应尽量做到平均分配。

返回值类型：INTEGER

示例：

```
 SELECT ss.ID,ss.NAME,ss.SCORE,NTILE(4) OVER ( ORDER BY ss.SCORE DESC ) N
TILE FROM STUDENT_SCORE ss;
```

返回结果如下：

```
 id | name |score | ntile
----+--------+-------+-------
 1 |诸葛亮 |  100|    1
 1 |赵子龙 |   95|    1
 3 |关羽  |   95|    2
 2 |马超  |   80|    2
 3 |刘备  |   80|    3
 2 |阿斗  |   60|    4
(6 rows)
```

❖ **LAG(value any [, offset integer [, default any]])**

描述：LAG 函数为各组内对应值生成滞后值。即当前值对应的行数往前偏移 offset 位后所得行的 value 值作为序号。若经过偏移后行数不存在，则对应结果取为 default 值。若无指定，在默认情况下，offset 取为 1，default 值取为 NULL。

返回值类型：与参数数据类型相同。

示例：

```
 SELECT ss.ID,ss.NAME,ss.SCORE,LAG(score) OVER ( ORDER BY ss.SCORE DESC )
LAG FROM STUDENT_SCORE ss;
```

返回结果如下：

```
 id | name |score | lag
```

```
----+--------+-------+-----
1 |诸葛亮 |  100|
1 |赵子龙 |   95|100
3 |关羽  |   95| 95
2 |马超  |   80| 95
3 |刘备  |   80| 80
2 |阿斗  |   60| 80
(6 rows)
```

❖ **LEAD(value any [, offset integer [, default any]])**

描述：LEAD 函数为各组内对应值生成提前值。即当前值对应的行数向后偏移 offset 位后所得行的 value 值作为序号。若经过向后偏移后行数超过当前组内的总行数，则对应结果取为 default 值。若无指定，在默认情况下，offset 取为 1，default 值取为 NULL。

返回值类型：与参数数据类型相同。

示例：

```
 SELECT ss.ID,ss.NAME,ss.SCORE,LEAD(score) OVER ( ORDER BY ss.SCORE DES
C ) LEAD FROM STUDENT_SCORE ss;
```

返回结果如下：

```
 id| name |score|lead
 ----+--------+-------+------
 1 |诸葛亮 |  100|  95
 1 |赵子龙 |   95|  95
 3 |关羽  |   95|  80
 2 |马超  |   80|  80
 3 |刘备  |   80|  60
 2 |阿斗  |   60|
 (6 rows)
```

❖ **FIRST_VALUE(value any)**

描述：FIRST_VALUE 函数取各组内的第一个值作为返回结果。

返回值类型：与参数数据类型相同。

示例：

```
 SELECT *,first_value(score) OVER (PARTITION BY id ORDER BY score DESC) firs
```

```
t_score FROM STUDENT_SCORE;
```

返回结果如下：

```
 id | name  | score | first_score
----+--------+-------+-------------
 1 | 诸葛亮 |  100 |       100
 1 | 赵子龙 |   95 |       100
 2 | 马超   |   80 |        80
 2 | 阿斗   |   60 |        80
 3 | 关羽   |   95 |        95
 3 | 刘备   |   80 |        95
(6 rows)
```

- **LAST_VALUE(value any)**

描述：LAST_VALUE 函数取各组内的最后一个值作为返回结果。

返回值类型：与参数数据类型相同。

示例：

```
 SELECT*,last_value(score) OVER (PARTITION BY id ORDER BY score DESC) last
_score FROM STUDENT_SCORE;
```

返回结果如下：

```
 id | name  | score | last_score
----+--------+-------+------------
 1 | 诸葛亮 |  100 |       100
 1 | 赵子龙 |   95 |        95
 2 | 马超   |   80 |        80
 2 | 阿斗   |   60 |        60
 3 | 关羽   |   95 |        95
 3 | 刘备   |   80 |        80
(6 rows)
```

- **NTH_VALUE(value any, nth integer)**

描述：NTH_VALUE 函数返回该组内的第 nth 行作为结果。若该行不存在，则默认返回 NULL。

返回值类型：与参数数据类型相同。

示例：

```
 SELECT *,nth_value(score,2) OVER (PARTITION BY id ORDER BY score DESC) nt
h_value FROM STUDENT_SCORE;
```

返回结果如下：

```
 id | name  | score | nth_value
 ----+--------+-------+-----------
 1 | 诸葛亮 |  100 |
 1 | 赵子龙 |  95 |     95
 2 | 马超  |  80 |
 2 | 阿斗  |  60 |     60
 3 | 关羽  |  95 |
 3 | 刘备  |  80 |     80
 (6 rows)
```

4.1.4.23.账本数据库的函数

- **get_dn_hist_relhash(text, text)**

  描述：返回指定防篡改用户表的表级数据 hash 值。该函数仅供分布式使用。

  参数类型：text

  返回值类型：hash16

- **ledger_hist_check(text, text)**

  描述：校验指定防篡改用户表的表级数据 hash 值与其对应历史表 hash 一致性。

  参数类型：text

  返回值类型：Boolean

  示例：

  1、创建 schema 和测试表。

  ```
  CREATE SCHEMA ledgernsp WITH BLOCKCHAIN;
  CREATE TABLE ledgernsp.tab(a int, b text);
  ```

  2、插入数据。

  ```
  INSERT INTO ledgernsp.tab values(generate_series(1, 10000), 'test');
  ```

  3、调用函数。

  ```
  SELECT ledger_hist_check('ledgernsp','tab');
  ```

返回结果为：

```
ledger_hist_check
------------------
t
(1 row)
```

- **ledger_hist_repair(text, text)**

描述：修复指定防篡改用户表对应的历史表 hash 值，使之与用户表 hash 一致，返回 hash 差值。

参数类型：text

返回值类型：hash16

示例：

```
SELECT ledger_hist_repair('ledgernsp','tab');
```

返回结果为：

```
sql
 ledger_hist_repair
--------------------
0000000000000000
(1 row)
```

- **ledger_hist_archive(text, text)**

描述：归档指定防篡改用户表对应的历史表至审计日志目录中 hist_back 文件夹下。

参数类型：text

返回值类型：Boolean

示例：

```
SELECT ledger_hist_archive('ledgernsp','tab');
```

返回结果为：

```
sql
 ledger_hist_archive
---------------------
t
(1 row)
```

❖ **ledger_gchain_check(text, text)**

描述：校验指定防篡改用户表对应的历史表 hash 与全局历史表对应的 relhash 一致性。

参数类型：text

返回值类型：Boolean

示例：

```
SELECT ledger_gchain_check('ledgernsp','tab');
```

返回结果为：

```
 ledger_gchain_check
---------------------
t
(1 row)
```

❖ **ledger_gchain_repair(text, text)**

描述：修复验指定防篡改用户表在全局历史表中的 relhash，使之与其历史表 hash 一致，返回 hash 差值。

参数类型：text

返回值类型：hash16

示例：

```
SELECT ledger_gchain_repair('ledgernsp','tab');
```

返回结果为：

```
ledger_gchain_repair
----------------------
da30c1260af5be50
(1 row)
```

❖ **ledger_gchain_archive(void)**

描述：归档全局历史表至审计日志目录中 hist_back 文件夹下。

参数类型：void

返回值类型：Boolean

示例：

```
SELECT ledger_gchain_archive();
```

返回结果为：

```
ledger_gchain_archive
---------------------
t
(1 row)
```

- **hash16in(cstring)**

  描述：将输入 16 进制字符串转化成内部 hash16 形式。

  参数类型：cstring

  返回值类型：hash16

- **hash16out(hash16)**

  描述：将内部 hash16 类型的数据转码转化为 16 进制 cstring 类型。

  参数类型：hash16

  返回值类型：cstring

- **hash32in(cstring)**

  描述：将输入 16 进制字符串（32 个字符）转化成内部类型 hash32 形式。

  参数类型：cstring

  返回值类型：hash32

- **hash32out(hash32)**

  描述：将内部 hash32 类型的数据转码转化为 16 进制 cstring 类型。

  参数类型：cstring

  返回值类型：hash32

4.1.4.24.密态等值的函数

- byteawithoutorderwithequalcolin(cstring)

  描述：将输入转码转化成内部 byteawithoutorderwithequalcol 形式。

  参数类型：cstring

  返回值类型：byteawithoutorderwithequalcol

- byteawithoutorderwithequalcolout(byteawithoutorderwithequalcol)

描述：将内部 byteawithoutorderwithequalcol 类型的数据转码转化为 cstring 类型。

参数类型：byteawithoutorderwithequalcol

返回值类型：cstring

- byteawithoutorderwithequalcolsend(byteawithoutorderwithequalcol)

  描述：将 byteawithoutorderwithequalcol 类型的数据转码转化为 bytea 类型。

  参数类型：byteawithoutorderwithequalcol

  返回值类型：bytea

- byteawithoutorderwithequalcolrecv(internal)

  描述：将 byteawithoutorderwithequalcol 类型的数据转码转化为 byteawithoutorderwithequalcol 类型。

  参数类型：internal

  返回值类型：byteawithoutorderwithequalcol

- byteawithoutorderwithequalcoltypmodin(_cstring)

  描述：将 byteawithoutorderwithequalcol 类型的数据转码转化为 byteawithoutorderwithequalcol 类型。

  参数类型：_cstring

  返回值类型：int4

- byteawithoutorderwithequalcoltypmodout(int4)

  描述：将 int4 类型的数据转码转化为 cstring 类型。

  参数类型：int4

  返回值类型：cstring

- byteawithoutordercolin(cstring)

  描述：将输入转码转化成内部 byteawithoutordercolin 形式。

  参数类型：cstring

  返回值类型：byteawithoutordercol

- byteawithoutordercolout(byteawithoutordercol)

  描述：将内部 byteawithoutordercol 类型的数据转码转化为 cstring 类型。

  参数类型：byteawithoutordercol

返回值类型：cstring

- byteawithoutordercolsend(byteawithoutordercol)

  描述：将 byteawithoutordercol 类型的数据转码转化为 bytea 类型。

  参数类型：byteawithoutordercol

  返回值类型：bytea

- byteawithoutordercolrecv(internal)

  描述：将 byteawithoutordercol 类型的数据转码转化为 byteawithoutordercol 类型。

  参数类型：internal

  返回值类型：byteawithoutordercol

- byteawithoutorderwithequalcolcmp(byteawithoutorderwithequalcol, byteawithoutorderwithequalcol)

  描述：比较两个 byteawithoutorderwithequalcol 类型的数据大小，若第一个参数小于第二个参数，返回-1，若等于，返回 0，若大于，则返回 1。

  参数类型：byteawithoutorderwithequalcol, byteawithoutorderwithequalcol

  返回值类型：int4

- byteawithoutorderwithequalcolcmpbytear(byteawithoutorderwithequalcol, bytea)

  描述：比较 byteawithoutorderwithequalcol 和 bytea 数据大小，若第一个参数小于第二个参数，返回-1，若等于，返回 0，若大于，则返回 1。

  参数类型：byteawithoutorderwithequalcol, bytea

  返回值类型：int4

- byteawithoutorderwithequalcolcmpbyteal(bytea, byteawithoutorderwithequalcol)

  描述：比较 bytea 和 byteawithoutorderwithequalcol 数据大小，若第一个参数小于第二个参数，返回-1，若等于，返回 0，若大于，则返回 1。

  参数类型：byteawithoutorderwithequalcol, bytea

返回值类型：int4

- byteawithoutorderwithequalcoleq(byteawithoutorderwithequalcol, byteawithoutorderwithequalcol)

  描述：比较两个 byteawithoutorderwithequalcol 类型的数据是否相同，相同则返回 true，否则返回 false。

  参数类型：byteawithoutorderwithequalcol, bytea

  返回值类型：bool

- byteawithoutorderwithequalcoleqbyteal(bytea, byteawithoutorderwithequalcol)

  描述：比较 bytea 和 byteawithoutorderwithequalcol 数据是否相同，相同则返回 true，否则返回 false。

  参数类型：bytea，byteawithoutorderwithequalcol

  返回值类型：bool

- byteawithoutorderwithequalcoleqbytear(byteawithoutorderwithequalcol, bytea)

  描述：比较 byteawithoutorderwithequalcol 和 bytea 数据是否相同，相同则返回 true，否则返回 false。

  参数类型：byteawithoutorderwithequalcol, bytea

  返回值类型：bool

- byteawithoutorderwithequalcolne(byteawithoutorderwithequalcol, byteawithoutorderwithequalcol)

  描述：比较两个 byteawithoutorderwithequalcol 类型的数据是否不相同，不相同则返回 true，否则返回 false。

  参数类型：byteawithoutorderwithequalcol, byteawithoutorderwithequalcol

  返回值类型：bool

- byteawithoutorderwithequalcolnebyteal(bytea, byteawithoutorderwithequalcol)

  描述：比较 bytea 和 byteawithoutorderwithequalcol 数据是否相同，相同则返回 true，否则返回 false。

  参数类型：bytea，byteawithoutorderwithequalcol

返回值类型：bool

- byteawithoutorderwithequalcolnebytear(byteawithoutorderwithequalcol, bytea)

  描述：比较 byteawithoutorderwithequalcol 和 bytea 数据是否不相同，相同则返回 true，否则返回 false。

  参数类型：byteawithoutorderwithequalcol, bytea

  返回值类型：bool

- hll_hash_byteawithoutorderwithequalcol(byteawithoutorderwithequalcol)

  描述：返回 byteawithoutorderwithequalcol 的 hll 哈希值。

  参数类型：byteawithoutorderwithequalcol

  返回值类型：hll_hashval

由于 byteawithoutorderwithequalcolin 的实现会对 cek 进行查找，并且判断是否为正常加密后的数据类型。因此如果用户输入数据的格式不为加密后的数据格式，并且在本地不存在对应 cek 的情况下，会返回错误。

```
panweidb=# SELECT * FROM byteawithoutorderwithequalcolsend('x907219912381298461289346129'::byteawithoutorderwithequalcol);
ERROR:  cek with OID 596711794 not found
LINE 1: SELECT * FROM byteawithoutorderwithequalcolsend('x907219912...
                                  ^
panweidb=# SELECT * FROM byteawithoutordercolout('x9072190199999999999912381298461289346129');
ERROR:  cek with OID 2566986098 not found
LINE 1: SELECT * FROM byteawithoutordercolout('x9072190199999999999...

SELECT * FROM byteawithoutorderwithequalcolrecv('x907219019999999999999123812981289346129'::byteawithoutorderwithequalcol);
ERROR:  cek with OID 2566986098 not found
                          ^
panweidb=# SELECT * FROM byteawithoutorderwithequalcolsend('x907219019999999999999912381298461289346129'::byteawithoutorderwithequalcol);
ERROR:  cek with OID 2566986098 not found
```

```
LINE 1: SELECT * FROM byteawithoutorderwithequalcolsend('x907219019...
                                                        ^
```

4.1.4.25.返回集合的函数

序列号生成函数

- generate_series(start, stop)

  描述：生成一个数值序列，从 start 到 stop，步长为 1。

  参数类型：int、bigint、numeric

  返回值类型：setof int、setof bigint、setof numeric（与参数类型相同）

- generate_series(start, stop, step)

  描述：生成一个数值序列，从 start 到 stop，步长为 step。

  参数类型：int、bigint、numeric

  返回值类型：setof int、setof bigint、setof numeric（与参数类型相同）

- generate_series(start, stop, step interval)

  描述：生成一个数值序列，从 start 到 stop，步长为 step。

  参数类型：timestamp 或 timestamp with time zone

  返回值类型：setof timestamp 或 setof timestamp with time zone（与参数类型相同）

如果 step 是正数且 start 大于 stop，则返回零行。相反，如果 step 是负数且 start 小于 stop，则也返回零行。如果输入是 NULL，同样产生零行。如果 step 为零则是一个错误。

示例：

```
panweidb=# SELECT * FROM generate_series(2,4);
 generate_series
-----------------
               2
               3
               4
(3 rows)
```

```
panweidb=# SELECT * FROM generate_series(5,1,-2);
 generate_series
-----------------
               5
               3
               1
(3 rows)

panweidb=# SELECT * FROM generate_series(4,3);
 generate_series
-----------------
(0 rows)

--这个示例应用于 date-plus-integer 操作符。
panweidb=# SELECT current_date + s.a AS dates FROM generate_series(0,14,7) AS s
(a);
   dates
------------
 2017-06-02
 2017-06-09
 2017-06-16
(3 rows)

panweidb=# SELECT * FROM generate_series('2008-03-01 00:00'::timestamp, '2008
-03-04 12:00', '10 hours');
   generate_series
---------------------
 2008-03-01 00:00:00
 2008-03-01 10:00:00
 2008-03-01 20:00:00
 2008-03-02 06:00:00
 2008-03-02 16:00:00
 2008-03-03 02:00:00
 2008-03-03 12:00:00
```

```
2008-03-03 22:00:00
2008-03-04 08:00:00
(9 rows)
```

下标生成函数

- generate_subscripts(array anyarray, dim int)

  描述：生成一系列包括给定数组的下标。

  返回值类型：setof int

- generate_subscripts(array anyarray, dim int, reverse boolean)

  描述：生成一系列包括给定数组的下标。当 reverse 为真时，该系列则以相反的顺序返回。

  返回值类型：setof int

generate_subscripts 是一个为给定数组中的指定维度生成有效下标集的函数。如果数组中没有所请求的维度或者 NULL 数组，返回零行（但是会给数组元素为空的返回有效下标）。示例：

```
--基本用法。
panweidb=# SELECT generate_subscripts('{NULL,1,NULL,2}'::int[], 1) AS s;
 s
---
 1
 2
 3
 4
(4 rows)
--unnest 一个 2D 数组。
panweidb=# CREATE OR REPLACE FUNCTION unnest2(anyarray)
RETURNS SETOF anyelement AS $$
SELECT $1[i][j]
  FROM generate_subscripts($1,1) g1(i),
       generate_subscripts($1,2) g2(j);
$$ LANGUAGE sql IMMUTABLE;

panweidb=# SELECT * FROM unnest2(ARRAY[[1,2],[3,4]]);
```

```
 unnest2
---------
     1
     2
     3
     4
(4 rows)

--删除函数。
panweidb=# DROP FUNCTION unnest2;
```

4.1.4.26.条件表达式函数

条件表达式函数

- coalesce(expr1, expr2, ..., exprn)

  描述：

  返回参数列表中第一个非 NULL 的参数值。

  COALESCE(expr1, expr2) 等价于 CASE WHEN expr1 IS NOT NULL THEN expr1 ELSE expr2 END。

  示例：

```
panweidb=# SELECT coalesce(NULL,'hello');
 coalesce
----------
 hello
(1 row)
```

备注：

- 如果表达式列表中的所有表达式都等于 NULL，则本函数返回 NULL。
- 它常用于在显示数据时用缺省值替换 NULL。
- 和 CASE 表达式一样，COALESCE 不会计算不需要用来判断结果的参数；即在第一个非空参数右边的参数不会被计算。
- decode(base_expr, compare1, value1, Compare2,value2, … default)

  描述：把 base_expr 与后面的每个 compare(n) 进行比较，如果匹配返回相应的 value(n)。如果没有发生匹配，则返回 default。

示例：

```
panweidb=# SELECT decode('A','A',1,'B',2,0);
 case
------
  1
(1 row)
```

备注：不支持对 xml 数据类型的操作。

❖ nullif(expr1, expr2)

描述：当且仅当 expr1 和 expr2 相等时，NULLIF 才返回 NULL，否则它返回 expr1。

nullif(expr1, expr2) 逻辑上等价于 CASE WHEN expr1 = expr2 THEN NULL ELSE expr1 END。

示例：

```
panweidb=# SELECT nullif('hello','world');
   nullif
  --------
  hello
  (1 row)
```

备注：

不支持对 xml 数据类型的操作。、

如果两个参数的数据类型不同，则：

❖ 若两种数据类型之间存在隐式转换，则以其中优先级较高的数据类型为基准将另一个参数隐式转换成该类型，转换成功则进行计算，转换失败则返回错误。如：

```
panweidb=# SELECT nullif('1234'::VARCHAR,123::INT4);
     nullif
    --------
      1234
    (1 row)
panweidb=# SELECT nullif('1234'::VARCHAR,'2012-12-24'::DATE);
ERROR:  invalid input syntax for type timestamp: "1234"
```

❖ 若两种数据类型之间不存在隐式转换，则返回错误 。如：

```
panweidb=# SELECT nullif(TRUE::BOOLEAN,'2012-12-24'::DATE);
ERROR:  operator does not exist: boolean = timestamp without time zone
LINE 1: SELECT nullif(TRUE::BOOLEAN,'2012-12-24'::DATE) FROM sys_dummy;
HINT:  No operator matches the given name and argument type(s). You might nee
d to add explicit type casts.
```

❖ nvl(expr1 , expr2)

描述：

- 如果 expr1 为 NULL，则返回 expr2。
- 如果 expr1 非 NULL，则返回 expr1。

示例：

```
panweidb=# SELECT nvl('hello','world');
   nvl
-------
 hello
(1 row)
```

备注：参数 expr1 和 expr2 可以为任意类型，当 NVL 的两个参数不属于同类型时，看第二个参数是否可以向第一个参数进行隐式转换。如果可以则返回第一个参数类型，否则返回错误。

❖ greatest(expr1 [, ...])

描述：获取并返回参数列表中值最大的表达式的值。

返回值类型：

示例：

```
panweidb=# SELECT greatest(1*2,2-3,4-1);
 greatest
----------
        3
(1 row)
panweidb=# SELECT greatest('HARRY', 'HARRIOT', 'HAROLD');
 greatest
----------
 HARRY
```

```
(1 row)
```

备注：不支持对 xml 数据类型的操作。

❖ least(expr1 [, ...])

描述：获取并返回参数列表中值最小的表达式的值。

示例：

```
panweidb=# SELECT least(1*2,2-3,4-1);
 least
-------
    -1
(1 row)
panweidb=# SELECT least('HARRY','HARRIOT','HAROLD');
 least
--------
 HAROLD
(1 row)
```

不支持对 xml 数据类型的操作。

❖ EMPTY_BLOB()

描述：使用 EMPTY_BLOB 在 INSERT 或 UPDATE 语句中初始化一个 BLOB 变量，取值为 NULL。

返回值类型：BLOB

示例：

```
--新建表
panweidb=# CREATE TABLE blob_tb(b blob,id int);
--插入数据
panweidb=# INSERT INTO blob_tb VALUES (empty_blob(),1);
--删除表
panweidb=# DROP TABLE blob_tb;
```

4.1.4.27.触发器函数

❖ pg_get_triggerdef(oid)

描述：获取触发器的定义信息。

参数：待查触发器的 OID。

返回值类型：text

示例：

```
panweidb=# select pg_get_triggerdef(oid) from pg_trigger;
 pg_get_triggerdef
-----------------------------------------------------------------------
 CREATE TRIGGER tg1 BEFORE INSERT ON gtest26 FOR EACH STATEMENT EXECUTE PR
OCEDURE gtest_trigger_func()
 CREATE TRIGGER tg03 AFTER INSERT ON gtest26 FOR EACH ROW WHEN ((new.a IS N
OT NULL)) EXECUTE PROCEDURE gtest_trigger_func()
(2 rows)
```

- pg_get_triggerdef(oid, boolean)

  描述：获取触发器的定义信息。

  参数：待查触发器的 OID 及是否以 pretty 方式展示。

说明

仅在创建 trigger 时指定 WHEN 条件的情况下，布尔类型参数才生效。

返回值类型：text

示例：

```
panweidb=# select pg_get_triggerdef(oid,true) from pg_trigger;
 pg_get_triggerdef
  -----------------------------------------------------------------------
 CREATE TRIGGER tg1 BEFORE INSERT ON gtest26 FOR EACH STATEMENT EXECUTE PR
OCEDURE gtest_trigger_func()
 CREATE TRIGGER tg03 AFTER INSERT ON gtest26 FOR EACH ROW WHEN (new.a IS NO
T NULL) EXECUTE PROCEDURE gtest_trigger_func()
(2 rows)

panweidb=# select pg_get_triggerdef(oid,false) from pg_trigger;
 pg_get_triggerdef
  -----------------------------------------------------------------------
 CREATE TRIGGER tg1 BEFORE INSERT ON gtest26 FOR EACH STATEMENT EXECUTE PR
OCEDURE gtest_trigger_func()
```

```
CREATE TRIGGER tg03 AFTER INSERT ON gtest26 FOR EACH ROW WHEN ((new.a IS N
OT NULL)) EXECUTE PROCEDURE gtest_trigger_func()
(2 rows)
```

4.1.4.28.提示信息函数

- report_application_error

  描述：PL 执行过程中，可以使用此函数来抛 ERROR。

  返回值类型：void

表 1 report_application_error 参数说明

参数	类型	说明	是否必选
log	text	error 消息的内容。	是
code	int4	error 消息对应的 error code，范围为：-20999 ~ -20000。	否

4.1.4.29.统计信息函数

统计信息函数根据访问对象分为两种类型：针对某个数据库进行访问的函数，以数据库中每个表或索引的 OID 作为参数，标识需要报告的数据库；针对某个服务器进行访问的函数，以一个服务器进程号为参数，其范围从 1 到当前活跃服务器的数目。

- pg_stat_get_db_conflict_tablespace(oid)

  描述：由于恢复与数据库中删除的表空间发生冲突而取消的查询数。

  返回值类型：bigint

- pg_control_group_config

  描述：在当前节点上打印 cgroup 配置。

  返回值类型：record

- pg_stat_get_db_stat_reset_time(oid)

  描述：上次重置数据库统计信息的时间。首次连接到每个数据库期间初始化为系统时间。当您在数据库上调用 pg_stat_reset 以及针对其中的任何表或索引执行 pg_stat_reset_single_table_counters 时，重置时间都会更新。

  返回值类型：timestamptz

- pg_stat_get_function_total_time(oid)

  描述：该函数花费的总挂钟时间，以微秒为单位。包括花费在此函数调用其他函数上的时间。

  返回值类型：bigint

- pg_stat_get_xact_tuples_returned(oid)

  描述：当前事务中参数为表时通过顺序扫描读取的行数，或参数为索引时返回的索引条目数。

  返回值类型：bigint

- pg_lock_status()

  描述：查询打开事务所持有的锁信息，所有用户均可执行该函数。

  返回值类型：返回字段可参考 PG_LOCKS 视图返回字段，该视图是通过查询本函数得到的结果。

- pg_stat_get_xact_numscans(oid)

  描述：当前事务中参数为表时执行的顺序扫描次数，或参数为索引时执行的索引扫描次数。

  返回值类型：bigint

- pg_stat_get_xact_blocks_fetched(oid)

  描述：当前事务中对表或索引的磁盘块获取请求数。

  返回值类型：bigint

- pg_stat_get_xact_blocks_hit(oid)

  描述：当前事务中对缓存中找到的表或索引的磁盘块获取请求数。

  返回值类型：bigint

- pg_stat_get_xact_function_calls(oid)

  描述：在当前事务中调用该函数的次数。

  返回值类型：bigint

- pg_stat_get_xact_function_self_time(oid)

  描述：在当前事务中仅花费在此函数上的时间，不包括花费在此函数内部调用其他函数上的时间。

  返回值类型：bigint

- pg_stat_get_xact_function_total_time(oid)

描述：当前事务中该函数所花费的总挂钟时间（以微秒为单位），包括花费在此函数内部调用其他函数上的时间。

返回值类型：bigint

- pg_stat_get_wal_senders()

  描述：在主机端查询 walsender 信息。

  返回值类型：setofrecord

返回字段说明如下：

表 1 返回字段说明

字段名称	字段类型	字段说明
pid	bigint	walsender 的线程号。
sender_pid	integer	walsender 的 pid 相对的轻量级线程号。
local_role	text	主节点类型。
peer_role	text	备节点类型。
peer_state	text	备节点状态。
state	text	walsender 状态。
catchup_start	timestamp with time zone	catchup 启动时间。
catchup_end	timestamp with time zone	catchup 结束时间。
sender_sent_location	text	主节点发送位置。
sender_write_location	text	主节点落盘位置。
sender_flush_location	text	主节点 flush 磁盘位置。
sender_replay_loc	text	主节点 redo 位置。

字段名称	字段类型	字段说明
ation		
receiver_received_location	text	备节点接收位置。
receiver_write_location	text	备节点落盘位置。
receiver_flush_location	text	备节点 flush 磁盘位置。
receiver_replay_location	text	备节点 redo 磁盘位置。
sync_percent	text	同步百分比。
sync_state	text	同步状态。
sync_priority	text	同步复制的优先级。
sync_most_available	text	最大可用模式设置。
channel	text	walsender 信道信息。

- get_paxos_replication_info()

  描述：查询 Paxos 模式下主机或备机的复制状态。

  返回值类型：setofrecord

返回字段说明如下：

表 2 返回字段说明

字段名称	字段类型	字段说明
paxos_write_location	text	已经写入 DCF 的 XLog 位置。
paxos_commit_location	text	已经在 DCF 中达成一致的 XLog 位置。
local_write_location	text	节点的落盘位置。

字段名称	字段类型	字段说明
local_flush_location	text	节点的 flush 磁盘位置。
local_replay_location	text	节点的 redo 磁盘位置。
dcf_replication_info	text	节点的 DCF 模块信息。

- pg_stat_get_stream_replications()

  描述：查询主备复制状态。

  返回值类型：setofrecord

返回值说明如下。

表 3 返回值说明

返回参数	返回参数类型	返回参数说明
local_role	text	本地角色。
static_connections	integer	连接统计。
db_state	text	数据库状态。
detail_information	text	详细信息。

- pg_stat_get_db_numbackends(oid)

  描述：处理该数据库活跃的服务器进程数目。

  返回值类型：integer

- pg_stat_get_db_xact_commit(oid)

  描述：数据库中已提交事务的数量。

  返回值类型：bigint

- pg_stat_get_db_xact_rollback(oid)

  描述：数据库中回滚事务的数量。

  返回值类型：bigint

- pg_stat_get_db_blocks_fetched(oid)

  描述：数据库中磁盘块抓取请求的总数。

  返回值类型：bigint

- pg_stat_get_db_blocks_hit(oid)

描述：数据库在缓冲区中找到的磁盘块抓取请求的总数。

返回值类型：bigint

- pg_stat_get_db_tuples_returned(oid)

  描述：为数据库返回的 Tuple 数。

  返回值类型：bigint

- pg_stat_get_db_tuples_fetched(oid)

  描述：为数据库中获取的 Tuple 数。

  返回值类型：bigint

- pg_stat_get_db_tuples_inserted(oid)

  描述：在数据库中插入 Tuple 数。

  返回值类型：bigint

- pg_stat_get_db_tuples_updated(oid)

  描述：在数据库中更新的 Tuple 数。

  返回值类型：bigint

- pg_stat_get_db_tuples_deleted(oid)

  描述：数据库中删除 Tuple 数。

  返回值类型：bigint

- pg_stat_get_db_conflict_lock(oid)

  描述：数据库中锁冲突的数量。

  返回值类型：bigint

- pg_stat_get_db_deadlocks(oid)

  描述：数据库中死锁的数量。

  返回值类型：bigint

- pg_stat_get_numscans(oid)

  描述：如果参数是一个表，则顺序扫描读取的行数目。如果参数是一个索引，则返回索引行的数目。

  返回值类型：bigint

- pg_stat_get_role_name(oid)

  描述：根据用户 oid 获取用户名。仅 sysadmin 和 monitor admin 用户可以访问。

  返回值类型：text

示例：

```
panweidb=# select pg_stat_get_role_name(10);
 pg_stat_get_role_name
-----------------------
 aabbcc
(1 row)
```

- pg_stat_get_tuples_returned(oid)

  描述：如果参数是一个表，则顺序扫描读取的行数目。如果参数是一个索引，则返回的索引行的数目。

  返回值类型：bigint

- pg_stat_get_tuples_fetched(oid)

  描述：如果参数是一个表，则位图扫描抓取的行数目。如果参数是一个索引，则用简单索引扫描抓取的行数目。

  返回值类型：bigint

- pg_stat_get_tuples_inserted(oid)

  描述：插入表中行的数量。

  返回值类型：bigint

- pg_stat_get_tuples_updated(oid)

  描述：在表中已更新行的数量。

  返回值类型：bigint

- pg_stat_get_tuples_deleted(oid)

  描述：从表中删除行的数量。

  返回值类型：bigint

- pg_stat_get_tuples_changed(oid)

  描述：该表上一次 analyze 或 autoanalyze 之后插入、更新、删除行的总数量。

  返回值类型：bigint

- pg_stat_get_tuples_hot_updated(oid)

  描述：表热更新的行数。

  返回值类型：bigint

- pg_stat_get_live_tuples(oid)

描述：表活行数。

返回值类型：bigint

- pg_stat_get_dead_tuples(oid)

  描述：表死行数。

  返回值类型：bigint

- pg_stat_get_blocks_fetched(oid)

  描述：表或者索引的磁盘块抓取请求的数量。

  返回值类型：bigint

- pg_stat_get_blocks_hit(oid)

  描述：在缓冲区中找到的表或者索引的磁盘块请求数目。

  返回值类型：bigint

- pg_stat_get_partition_tuples_inserted(oid)

  描述：插入相应表分区中行的数量。

  返回值类型：bigint

- pg_stat_get_partition_tuples_updated(oid)

  描述：在相应表分区中已更新行的数量。

  返回值类型：bigint

- pg_stat_get_partition_tuples_deleted(oid)

  描述：从相应表分区中删除行的数量。

  返回值类型：bigint

- pg_stat_get_partition_tuples_changed(oid)

  描述：该表分区上一次 analyze 或 autoanalyze 之后插入、更新、删除行的总数量。

  返回值类型：bigint

- pg_stat_get_partition_live_tuples(oid)

  描述：分区表活行数。

  返回值类型：bigint

- pg_stat_get_partition_dead_tuples(oid)

  描述：分区表死行数。

  返回值类型：bigint

- pg_stat_get_xact_tuples_fetched(oid)

描述：事务中扫描的 tuple 行数。

返回值类型：bigint

- pg_stat_get_xact_tuples_inserted(oid)

  描述：表相关的活跃子事务中插入的 tuple 数。

  返回值类型：bigint

- pg_stat_get_xact_tuples_deleted(oid)

  描述：表相关的活跃子事务中删除的 tuple 数。

  返回值类型：bigint

- pg_stat_get_xact_tuples_hot_updated(oid)

  描述：表相关的活跃子事务中热更新的 tuple 数。

  返回值类型：bigint

- pg_stat_get_xact_tuples_updated(oid)

  描述：表相关的活跃子事务中更新的 tuple 数。

  返回值类型：bigint

- pg_stat_get_xact_partition_tuples_inserted(oid)

  描述：表分区相关的活跃子事务中插入的 tuple 数。

  返回值类型：bigint

- pg_stat_get_xact_partition_tuples_deleted(oid)

  描述：表分区相关的活跃子事务中删除的 tuple 数。

  返回值类型：bigint

- pg_stat_get_xact_partition_tuples_hot_updated(oid)

  描述：表分区相关的活跃子事务中热更新的 tuple 数。

  返回值类型：bigint

- pg_stat_get_xact_partition_tuples_updated(oid)

  描述：表分区相关的活跃子事务中更新的 tuple 数。

  返回值类型：bigint

- pg_stat_get_last_vacuum_time(oid)

  描述：用户在该表上最后一次手动启动清理或者 autovacuum 线程启动清理的时间。

  返回值类型：timestamptz

- pg_stat_get_last_autovacuum_time(oid)

描述：autovacuum 守护进程在该表上最后一次启动清理的时间。

返回值类型：timestamptz

- pg_stat_get_vacuum_count(oid)

  描述：用户在该表上启动清理的次数。

  返回值类型：bigint

- pg_stat_get_autovacuum_count(oid)

  描述：autovacuum 守护进程在该表上启动清理的次数。

  返回值类型：bigint

- pg_stat_get_last_analyze_time(oid)

  描述：用户在该表上最后一次手动启动分析或者 autovacuum 线程启动分析的时间。

  返回值类型：timestamptz

- pg_stat_get_last_autoanalyze_time(oid)

  描述：autovacuum 守护进程在该表上最后一次启动分析的时间。

  返回值类型：timestamptz

- pg_stat_get_analyze_count(oid)

  描述：用户在该表上启动分析的次数。

  返回值类型：bigint

- pg_stat_get_autoanalyze_count(oid)

  描述：autovacuum 守护进程在该表上启动分析的次数。

  返回值类型：bigint

- pg_total_autovac_tuples(bool,bool)

  描述：返回 total autovac 相关的 tuple 记录，如 nodename,nspname,relname 以及各类 tuple 的 IUD 信息，入参分别为：是否查询 relation 信息，是否查询 local 信息。

  返回值类型：setofrecord

返回参数说明如下。

表 4 返回参数说明

返回参数	返回参数类型	返回参数说明
nodename	name	节点名称。

返回参数	返回参数类型	返回参数说明
nspname	name	名称空间名称。
relname	name	表、索引、视图等对象名称。
partname	name	分区名称。
n_dead_tuples	bigint	表分区内的死行数。
n_live_tuples	bigint	表分区内的活行数。
changes_since_analyze	bigint	analyze 产生改变的数量。

- pg_autovac_status(oid)

  描述：返回和 autovac 状态相关的参数信息，如 nodename,nspname,relname,analyze,vacuum 设置，analyze/vacuum 阈值, analyze/vacuum tuple 数等。仅 sysadmin 可以使用该函数。

  返回值类型：setofrecord

返回值参数说明如下。

表 5 返回值参数说明

返回参数	返回参数类型	返回参数说明
nspname	text	名称空间名称。
relname	text	表、索引、视图等对象名称。
nodename	text	节点名称。
doanalyze	Boolean	是否执行 analyze。
anltuples	bigint	analyze tuple 数量。
anlthresh	bigint	analyze 阈值。
dovacuum	Boolean	是否执行 vacuum。
vactuples	bigint	vacuum tuple 数量。

返回参数	返回参数类型	返回参数说明
vacthresh	bigint	vacuum 阈值。

- pg_autovac_timeout(oid)

  描述：返回某个表做 autovac 连续超时的次数，表信息非法或 node 信息异常返回 NULL。

  返回值类型：bigint

- pg_stat_get_last_data_changed_time(oid)

  描述：insert/update/delete, exchange/truncate/drop partition 在该表上最后一次操作的时间，PG_STAT_ALL_TABLES(参考：开发者指南->系统表和系统视图->系统视图->PG_STAT_ALL_TABLES 章节)视图 last_data_changed 列的数据是通过该函数求值，在表数量很大的场景中，通过视图获取表数据最后修改时间的性能较差，建议直接使用该函数获取表数据的最后修改时间。入参可以是表的 oid 或分区的 oid，分别获取表或表的分区的最后修改时间。

  返回值类型：timestamptz

- pg_stat_set_last_data_changed_time(oid)

  描述：手动设置该表上最后一次 insert/update/delete, exchange/truncate/drop partition 操作的时间。

  返回值类型：void

- pg_backend_pid()

  描述：当前会话的服务器线程的线程 ID。

  返回值类型：bigint

- pg_stat_get_activity(integer)

  描述：返回一个关于带有特殊 PID 的后台进程的记录信息，当参数为 NULL 时，则返回每个活动的后台进程的记录。返回结果不包含 connection_info 列。初始用户、系统管理员和 monadmin 可以查看所有的数据，普通用户只能查询自己的结果。

  示例：

```
panweidb=# select * from pg_stat_get_activity(139881386280704);
```

```
 datid | pid | sessionid | usesysid | application_name | state | query  | waiting |       xact_start        |        query_start        |
   backend_start      |         state_change        | client_addr | client_hostname | client_port | enqueue |        query_id         |  srespool  | global_sessionid | unique_sql_id |  trace_id
-------+-----------------+----------+----------+------------------+-------+-------------------------------------------+---------+-------------------------------+-------------------------------+-----------------+-----------------+-------------+---------+-------------------------+------------+------------------+---------------+-----------
 16545 | 139881386280704 |     69 |     10 |  gsql     | active | select * from pg_stat_get_activity(139881386280704); | f      | 2022-01-18 19:43:05.167718+08 | 2022-01-18 19:43:05.167718+08 | 2022
-01-18 19:42:33.513507+08 | 2022-01-18 19:43:05.16773+08 |          |             |
 -1 |       | 72620543991624410 | default_pool | 1938253334#69#0 |   3751941862 |
(1 row)
```

返回值类型：setofrecord

返回参数说明如下。

表 6 返回参数说明

返回参数	返回参数类型	返回参数说明
datid	oid	用户会话在后台连接到的数据库 OID。
pid	bigint	后台线程 ID。
sessionid	bigint	会话 ID。
usesysid	oid	登录该后台的用户 OID。
application_name	text	连接到该后台的应用名。
state	text	该后台当前总体状态。
query	text	该后台的最新查询。如果 state 状态是 active（活跃的），此字段显示当前正在执行的查询。所有其他情况表示上一个查询。

返回参数	返回参数类型	返回参数说明
waiting	Boolean	如果后台当前正等待锁则为 true。
xact_start	timestamp with time zone	启动当前事务的时间，如果没有事务是活跃的，则为 null。 如果当前查询是首个事务，则这列等同于 query_start 列。
query_start	timestamp with time zone	开始当前活跃查询的时间，如果 state 的值不是 active，则这个值是上一个查询的开始时间。
backend_start	timestamp with time zone	该过程开始的时间，即当客户端连接服务器时。
state_change	timestamp with time zone	上次状态改变的时间。
client_addr	inet	连接到该后台的客户端的 IP 地址。如果此字段是 null，它表明通过服务器机器上 UNIX 套接字连接客户端或者这是内部进程，如 autovacuum。
client_hostname	text	客户端的主机名，这个字段是通过 client_addr 的反向 DNS 查找得到。这个字段只有在启动 log_hostname 且使用 IP 连接时才非空。
client_port	integer	客户端用于与后台通讯的 TCP 端口号，如果使用 Unix 套接字，则为-1。
enqueue	text	该字段暂不支持。

返回参数	返回参数类型	返回参数说明
query_id	bigint	查询语句的 ID。
srespool	name	资源池名字。
global_sessionid	text	全局会话 id。
unique_sql_id	bigint	语句的 unique sql id。
trace_id	text	驱动传入的 trace id，与应用的一次请求相关联。

- pg_stat_get_activity_with_conninfo(integer)

  描述：返回一个关于带有特殊 PID 的后台进程的记录信息，当参数为 NULL 时，则返回每个活动的后台进程的记录。初始用户、系统管理员和 monadmin 可以查看所有的数据，普通用户只能查询自己的结果。

  返回值类型：setofrecord

返回值说明如下。

表 7 返回值说明

返回值	返回值类型	返回值说明
datid	oid	用户会话在后台连接到的数据库 OID。
pid	bigint	后台线程 ID。
sessionid	bigint	会话 ID。
usesysid	oid	登录该后台的用户 OID。
application_name	text	连接到该后台的应用名。
state	text	改后台当前总体状态
query	text	该后台的最新查询。如

返回值	返回值类型	返回值说明
		果 state 状态是 active（活跃的），此字段显示当前正在执行的查询。所有其他情况表示上一个查询。
waiting	Boolean	如果后台当前正等待锁则为 true
xact_start	timestamp with time zone	启动当前事务的时间，如果没有事务是活跃的，则为 null。如果当前查询是首个事务，则这列等同于 query_start 列。
query_start	timestamp with time zone	开始当前活跃查询的时间，如果 state 的值不是 active，则这个值是上一个查询的开始时间。
backend_start	timestamp with time zone	该过程开始的时间，即当客户端连接服务器时。
state_change	timestamp with time zone	上次状态改变的时间。
client_addr	inet	连接到该后台的客户端的 IP 地址。如果此字段是 null，它表明通过服务器机器上 UNIX 套接字连接客户端或者这是内部进程，如 autovacuum

返回值	返回值类型	返回值说明
client_hostname	text	客户端的主机名，这个字段是通过 client_addr 的反向 DNS 查找得到。这个字段只有在启动 log_hostname 且使用 IP 连接时才非空。
client_port	integer	客户端用于与后台通讯的 TCP 端口号，如果使用 Unix 套接字，则为-1。
enqueue	text	该字段暂不支持。
query_id	bigint	查询语句的 ID。
connection_info	text	json 格式字符串，记录当前连接数据库的驱动类型、驱动版本号、当前驱动的部署路径、进程属主用户等信息。
srespool	name	资源池名字。
global_sessionid	text	全局会话 ID。
unique_sql_id	bigint	语句的 unique sql id。
trace_id	text	驱动传入的 trace id，与应用的一次请求相关联。

- pg_user_iostat(text)

  描述：显示和当前用户执行作业正在运行时的 IO 负载管理相关信息。

  返回值类型：record

函数返回字段说明如下：

名称	类型	描述
userid	oid	用户 id。
min_curr_iops	int4	当前该用户 io 在数据库节点中的最小值。对于行存，以万次/s 为单位；对于列存，以次/s 为单位。
max_curr_iops	int4	当前该用户 io 在数据库节点中的最大值。对于行存，以万次/s 为单位；对于列存，以次/s 为单位。
min_peak_iops	int4	该用户 io 峰值中，数据库节点的最小值。对于行存，以万次/s 为单位；对于列存，以次/s 为单位。
max_peak_iops	int4	该用户 io 峰值中，数据库节点的最大值。对于行存，以万次/s 为单位；对于列存，以次/s 为单位。
io_limits	int4	用户指定的资源池所设置的 io_limits。对于行存，以万次/s 为单位；对于列存，以次/s 为单位。
io_priority	text	该用户所设 io_priority。对于行存，以万次/s 为单位；对于列存，以次/s 为单位。
curr_io_limits	int4	使用 io_priority 管控 io 时的实时 io_limits 值。

- pg_stat_get_function_calls(oid)

  描述：函数已被调用次数。

  返回值类型：bigint

- pg_stat_get_function_self_time(oid)

  描述：只有在此函数上所花费的时间。此函数调用其他函数上花费的时间被排除在外。

  返回值类型：bigint

- pg_stat_get_backend_idset()

  描述：设置当前活动的服务器进程数（从 1 到活动服务器进程的数量）。

  返回值类型：setofinteger

- pg_stat_get_backend_pid(integer)

  描述：给定的服务器线程的线程 ID。

返回值类型：bigint

- pg_stat_get_backend_dbid(integer)

  描述：给定服务器进程的数据库 ID。

  返回值类型：oid

- pg_stat_get_backend_userid(integer)

  描述：给定服务器进程的用户 ID。

  返回值类型：oid

- pg_stat_get_backend_activity(integer)

  描述：给定服务器进程的当前活动查询，仅在调用者是系统管理员或被查询会话的用户，并且打开 track_activities 的时候才能获得结果。

  返回值类型：text

- pg_stat_get_backend_waiting(integer)

  描述：如果给定服务器进程在等待某个锁，并且调用者是系统管理员或被查询会话的用户，并且打开 track_activities 的时候才返回真。

  返回值类型：Boolean

- pg_stat_get_backend_activity_start(integer)

  描述：给定服务器进程当前正在执行的查询的起始时间，仅在调用者是系统管理员或被查询会话的用户，并且打开 track_activities 的时候才能获得结果。

  返回值类型：timestampwithtimezone

- pg_stat_get_backend_xact_start(integer)

  描述：给定服务器进程当前正在执行的事务的开始时间，但只有当前用户是系统管理员或被查询会话的用户，并且打开 track_activities 的时候才能获得结果。

  返回值类型：timestampwithtimezone

- pg_stat_get_backend_start(integer)

  描述：给定服务器进程启动的时间，如果当前用户不是系统管理员或被查询的后端的用户，则返回 NULL。

  返回值类型：timestampwithtimezone

- pg_stat_get_backend_client_addr(integer)

描述：连接到给定客户端后端的 IP 地址。如果是通过 Unix 域套接字连接的则返回 NULL；如果当前用户不是系统管理员或被查询会话的用户，也返回 NULL。

返回值类型：inet

- pg_stat_get_backend_client_port(integer)

  描述：连接到给定客户端后端的 TCP 端口。如果是通过 Unix 域套接字连接的则返回-1；如果当前用户不是系统管理员或被查询会话的用户，也返回 NULL。

  返回值类型：integer

- pg_stat_get_bgwriter_timed_checkpoints()

  描述：后台写进程开启定时检查点的时间（因为 checkpoint_timeout 时间已经过期了）。

  返回值类型：bigint

- pg_stat_get_bgwriter_requested_checkpoints()

  描述：后台写进程开启基于后端请求的检查点的时间，因为已经超过了 checkpoint_segments 或因为已经执行了 CHECKPOINT。

  返回值类型：bigint

- pg_stat_get_bgwriter_buf_written_checkpoints()

  描述：在检查点期间后台写进程写入的缓冲区数目。

  返回值类型：bigint

- pg_stat_get_bgwriter_buf_written_clean()

  描述：为日常清理脏块，后台写进程写入的缓冲区数目。

  返回值类型：bigint

- pg_stat_get_bgwriter_maxwritten_clean()

  描述：后台写进程停止清理扫描的时间，因为已经写入了更多的缓冲区（相比 bgwriter_lru_maxpages 参数声明的缓冲区数）。

  返回值类型：bigint

- pg_stat_get_buf_written_backend()

  描述：后端进程写入的缓冲区数，因为它们需要分配一个新的缓冲区。

  返回值类型：bigint

- pg_stat_get_buf_alloc()

描述：分配的总缓冲区数。

返回值类型：bigint

- pg_stat_clear_snapshot()

  描述：清理当前的统计快照。

  返回值类型：void

- pg_stat_reset()

  描述：为当前数据库重置统计计数器为 0（需要系统管理员权限）。

  返回值类型：void

- pg_stat_reset_shared(text)

  描述：重置 shared cluster 每个节点当前数据统计计数器为 0（需要系统管理员权限）。

  返回值类型：void

- pg_stat_reset_single_table_counters(oid)

  描述：为当前数据库中的一个表或索引重置统计为 0（需要系统管理员权限）。

  返回值类型：void

- pg_stat_reset_single_function_counters(oid)

  描述：为当前数据库中的一个函数重置统计为 0（需要系统管理员权限）。

  返回值类型：void

- pg_stat_session_cu(int, int, int)

  描述：获取当前节点所运行 session 的 CU 命中统计信息。

  返回值类型：record

- pg_stat_get_cu_mem_hit(oid)

  描述：获取当前节点当前数据库中一个列存表的 CU 内存命中次数。

  返回值类型：bigint

- pg_stat_get_cu_hdd_sync(oid)

  描述：获取当前节点当前数据库中一个列存表从磁盘同步读取 CU 次数。

  返回值类型：bigint

- pg_stat_get_cu_hdd_asyn(oid)

  描述：获取当前节点当前数据库中一个列存表从磁盘异步读取 CU 次数。

  返回值类型：bigint

- pg_stat_get_db_cu_mem_hit(oid)

  描述：获取当前节点一个数据库 CU 内存命中次数。

  返回值类型：bigint

- pg_stat_get_db_cu_hdd_sync(oid)

  描述：获取当前节点一个数据库从磁盘同步读取 CU 次数。

  返回值类型：bigint

- fenced_udf_process(integer)

  描述：查看本地 UDF Master 和 Work 进程数。入参为 1 时查看 master 进程数，入参为 2 时查看 worker 进程数，入参为 3 时杀死所有 worker 进程。

  返回值类型：text

- total_cpu()

  描述：获取当前节点使用的 CPU 时间，单位是 jiffies。

  返回值类型：bigint

- mot_global_memory_detail()

  描述：检查 MOT 全局内存的大小，主要包括数据和索引。

  返回值类型：record

- mot_local_memory_detail()

  描述：检查 MOT 局部内存的大小，主要包括数据和索引。

  返回值类型：record

- mot_session_memory_detail()

  描述：检查所有会话对 MOT 内存的使用情况。

  返回值类型：record

- total_memory()

  描述：获取当前节点使用的虚拟内存大小，单位 KB。

  返回值类型：bigint

- pg_stat_get_db_cu_hdd_asyn(oid)

  描述：获取当前节点一个数据库从磁盘异步读取 CU 次数。

  返回值类型：bigint

- pg_stat_bad_block(text, int, int, int, int, int, timestamp with time zone, timestamp with time zone)

描述：获取当前节点自启动后，读取出现 Page/CU 的损坏信息。

例: select * from pg_stat_bad_block();

返回值类型：record

- pg_stat_bad_block_clear()

  描述：清理节点记录的读取出现的 Page/CU 损坏信息（需要系统管理员权限）。

  返回值类型：void

- gs_respool_exception_info(pool text)

  描述：查看某个资源池关联的查询规则信息。

  返回值类型：record

- gs_control_group_info(pool text)

  描述：查看资源池关联的控制组信息

  返回值类型：record

返回信息如下：

属性	属性值	描述
name	class_a:workload_a1	class 和 workload 名称
class	class_a	Class 控制组名称
workload	workload_a1	Workload 控制组名称
type	DEFWD	控制组类型（Top、CLASS、BAKWD、DEFWD、TSWD）
gid	87	控制组 id
shares	30	占父节点 CPU 资源的百分比
limits	0	占父节点 CPU 核数的百分比
rate	0	Timeshare 中的分配比例
cpucores	0-3	CPU 核心数

- gs_all_control_group_info()

描述：查看数据库内所有的控制组信息。

返回值类型：record

- gs_get_control_group_info()

  描述：查看所有的控制组信息。

  返回值类型：record

- get_instr_workload_info(integer)

  描述：获取数据库主节点上事务量信息，事务时间信息。

  返回值类型：record

属性	属性值	描述
resourcepool_oid	10	资源池的 oid(逻辑同负载等价)
commit_counter	4	前端事务 commit 数量
rollback_counter	1	前端事务 rollback 数量
resp_min	949	前端事务最小响应时间（单位：微秒）
resp_max	201891	前端事务最大响应时间（单位：微秒）
resp_avg	43564	前端事务平均响应时间(单位：微秒)
resp_total	217822	前端事务总响应时间（单位：微秒）
bg_commit_counter	910	后端事务 commit 数量
bg_rollback_counter	0	后端事务 rollback 数量
bg_resp_min	97	后端事务最小响应时间（单位：微秒）
bg_resp_max	678080687	后端事务最大响应时间（单位：微秒）
bg_resp_avg	327847884	后端事务平均响应时间（单位：微秒）

属性	属性值	描述
bg_resp_total	298341575300	后端事务总响应时间（单位：微秒）

❖ pv_instance_time()

描述：获取当前节点上各个关键阶段的时间消耗。

返回值类型：record

Stat_name 属性	属性值	描述
DB_TIME	1062385	所有线程端到端的墙上时间（WALL TIME）消耗总和(单位： 微秒)
CPU_TIME	311777	所有线程 CPU 时间消耗总和(单位： 微秒)
EXECUTION_TIME	380037	消耗在执行器上的时间总和(单位： 微秒)
PARSE_TIME	6033	消耗在 SQL 解析上的时间总和(单位：微秒)
PLAN_TIME	173356	消耗在执行计划生成上的时间总和(单位： 微秒)
REWRITE_TIME	2274	消耗在查询重写上的时间总和(单位： 微秒)
PL_EXECUTION_TIME	0	消耗在 PL/SQL 执行上的时间总和(单位： 微秒)
PL_COMPILATION_TIME	557	消耗在 SQL 编译上的时间总和(单位：微秒)
NET_SEND_TIME	1673	消耗在网络发送上的时间总和(单位： 微秒)
DATA_IO_TIME	426622	消耗在数据读写上的时间总和(单位： 微秒)

- DBE_PERF.get_global_instance_time()

  描述：提供 panweidb 各个关键阶段的时间消耗，查询该函数必须具有 sysadmin 权限。

  返回值类型：record

- get_instr_unique_sql()

  描述：获取当前节点的执行语句（归一化 SQL）信息，查询该函数必须具有 sysadmin 权限。

  返回值类型：record

- reset_unique_sql(text, text, bigint)

  描述：重置系统执行语句（归一化 SQL）信息，执行该函数必须具有 sysadmin 权限。第一个参数取值范围“global/local”，global 表示清理所有节点上的信息，local 表示只清理当前节点；第二参数取值范围“ALL/BY_USERID/BY_CNID”，ALL 表示清理所有信息，BY_USERID 表示通过指定 USERID 清理只属于该用户的 sql 信息，BY_CNID 表示清理系统中涉及到该数据库主节点的 sql 信息；第三个参数表示具体的 CNID 和 USERID，如果第二个参数为 ALL，第三个参数不起作用，可以取任意值。

  返回值类型：boolean

📖 说明：

此函数中所说节点指分布式节点，当前 panweidb 为集中式数据库，global 与 local 功能一致，取值范围不支持 BY_CNID。

- get_instr_wait_event(NULL)

  描述：获取当前节点 event 等待的统计信息。

  返回值类型：record

- get_instr_user_login()

  描述：获取当前节点的用户登入登出次数信息，查询该函数必须具有 sysadmin 或者 monitor admin 权限。

  返回值类型：record

- get_instr_rt_percentile(integer)

  描述：获取数据库 SQL 响应时间 P80,P95 分布信息。

  返回值类型：record

- get_node_stat_reset_time()

  描述：获取当前节点的统计信息重置（重启，主备倒换，数据库删除）时间。

  返回值类型：record

- DBE_PERF.get_global_os_runtime()

  描述：显示当前操作系统运行的状态信息，查询该函数必须具有 sysadmin 权限。

  返回值类型：record

- DBE_PERF.get_global_os_threads()

  描述：提供 panweidb 中所有正常节点下的线程状态信息，查询该函数必须具有 sysadmin 权限。

  返回值类型：record

- DBE_PERF.get_summary_workload_sql_count()

  描述：提供 panweidb 中不同负载 SELECT，UPDATE，INSERT，DELETE，DDL，DML，DCL 计数信息，查询该函数必须具有 sysadmin 权限。

  返回值类型：record

- DBE_PERF.get_summary_workload_sql_elapse_time()

  描述：提供 panweidb 中不同负载 SELECT，UPDATE，INSERT，DELETE，响应时间信息（TOTAL,AVG, MIN, MAX），查询该函数必须具有 sysadmin 权限。

  返回值类型：record

- DBE_PERF.get_global_workload_transaction()

  描述：获取 panweidb 内所有节点上的事务量信息，事务时间信息，查询该函数必须具有 sysadmin 权限。

  返回值类型：record

- DBE_PERF.get_global_session_stat()

描述：获取 panweidb 节点上的会话状态信息，查询该函数必须具有 sysadmin 权限。

返回值类型：record

📖 说明：

状态信息有 17 项：commit、rollback、sql、table_scan、blocks_fetched、physical_read_operation、

shared_blocks_dirtied、local_blocks_dirtied、shared_blocks_read、local_blocks_read、

blocks_read_time、blocks_write_time、sort_imemory、sort_idisk、cu_mem_hit、

cu_hdd_sync_read、cu_hdd_asyread。

- DBE_PERF.get_global_session_time()

  描述：提供 panweidb 各节点各个关键阶段的时间消耗，查询该函数必须具有 sysadmin 权限。

  返回值类型：record

- DBE_PERF.get_global_session_memory()

  描述：汇聚各节点的 Session 级别的内存使用情况，包含执行作业在数据节点上 Postgres 线程和 Stream 线程分配的所有内存，单位为 MB，查询该函数必须具有 sysadmin 权限。

  返回值类型：record

- DBE_PERF.get_global_session_memory_detail()

  描述：汇聚各节点的线程的内存使用情况，以 MemoryContext 节点来统计，查询该函数必须具有 sysadmin 权限。

  返回值类型：record

- create_wlm_session_info(int flag)

  描述：将当前内存中记录的 TopSQL 查询语句级别相关统计信息清理。该函数只有管理员用户可以执行。

返回值类型：int

- pg_stat_get_wlm_session_info(int flag)

  描述：获取当前内存中记录的 TopSQL 查询语句级别相关统计信息，当传入的参数不为 0 时，会将这部分信息从内存中清理掉。该函数只有 system admin 和 monitor admin 用户可以执行。

  返回值类型：record

- gs_paxos_stat_replication()

  描述：在主机端查询备机信息。目前只支持集中式 DCF 模式。

  返回值类型：setofrecord

返回字段说明如下：

字段名称	字段类型	字段说明
local_role	text	发送日志节点的角色。
peer_role	text	接收日志节点的角色。
local_dcf_role	text	发送日志节点的 DCF 角色。
peer_dcf_role	text	接收日志节点的 DCF 角色。
peer_state	text	接收日志节点的状态。
sender_write_location	text	发送日志节点写到 xlog buffer 的位置。
sender_commit_location	text	发送日志节点 DCF 日志达成一致性点。
sender_flush_location	text	发送日志节点写到 xlog disk 的位置。
sender_replay_location	text	发送日志节点 replay 的位置。
receiver_write_location	text	接收日志节点写到 xlog buffer 的位置。

字段名称	字段类型	字段说明
receiver_commit_location	text	接收日志节点 DCF 日志达成一致性点。
receiver_flush_location	text	接收日志节点写到 xlog disk 的位置。
receiver_replay_location	text	接收日志节点重演 xlog 的位置。
sync_percent	text	同步百分比。
dcf_run_mode	int4	DCF 同步模式。
channel	text	信道信息。

- gs_wlm_get_resource_pool_info(int)

  描述：获取所有用户的资源使用统计信息，入参为 int 类型可以为可以为任意 int 值或 NULL。

  返回值类型：record

- gs_wlm_get_all_user_resource_info()

  描述：获取所有用户的资源使用统计信息。

  返回值类型：record

- gs_wlm_get_user_info(int)

  描述：获取所有用户的相关信息，入参为 int 类型可以为可以为任意 int 值或 NULL。该函数只有 sysadmin 权限的用户可以执行。

  返回值类型：record

- gs_wlm_get_workload_records()

  描述：获取动态负载管理下的所有作业信息，该函数只在动态负载管理开的情况下有效。

  返回值类型：record

- gs_wlm_readjust_user_space()

  描述：修正所有用户的存储空间使用情况。该函数只有管理员用户可以执行。

返回值类型：record

- gs_wlm_readjust_user_space_through_username(text name)

  描述：修正指定用户的存储空间使用情况。该函数普通用户只能修正自己的使用情况，只有管理员用户可以修正所有用户的使用情况。当 name 指定位“0000”，表示需要修正所有用户的使用情况。

  返回值类型：record

- gs_wlm_readjust_user_space_with_reset_flag(text name, boolean isfirst)

  描述：修正指定用户的存储空间使用情况。入参 isfirst 为 true 表示从 0 开始统计，否则从上一次结果继续统计。该函数普通用户只能修正自己的使用情况，只有管理员用户可以修正所有用户的使用情况。当 name 指定位“0000”，表示需要修正所有用户的使用情况。

  返回值类型：record

- gs_wlm_session_respool(bigint)

  描述：获取当前所有后台线程的 session resouce pool 相关信息，入参为 bigint 类型可以为可以为任意 bigint 值或 NULL。

  返回值类型：record

- gs_wlm_get_session_info()

  描述：目前该接口已废弃，暂不可用。

- gs_wlm_get_user_session_info()

  描述：目前该接口已废弃，暂不可用。

- gs_io_wait_status()

  描述：目前该接口不支持单机和集中式，暂不可用。

- global_stat_get_hotkeys_info()

  描述：获取整个数据库实例中热点 key 的统计情况。目前该接口不支持单机和集中式，暂不可用。

- global_stat_clean_hotkeys()

  描述：清理整个数据库实例中热点 key 的统计信息。目前该接口不支持单机和集中式，暂不可用。

- DBE_PERF.get_global_session_stat_activity()

描述：汇聚 panweidb 内各节点上正在运行的线程相关的信息，查询该函数必须具有 sysadmin 权限。

返回值类型：record

- DBE_PERF.get_global_thread_wait_status()

  描述：汇聚所有节点上工作线程（backend thread）以及辅助线程（auxiliary thread）的阻塞等待情况，查询该函数必须具有 sysadmin 和 monitoradmin 权限。

  返回值类型：record

- DBE_PERF.get_global_operator_history_table()

  描述：汇聚当前用户数据库主节点上执行作业结束后的算子相关记录（持久化），查询该函数必须具有 sysadmin 和 monitoradmin 权限。

  返回值类型：record

- DBE_PERF.get_global_operator_history()

  描述：汇聚当前用户数据库主节点上执行作业结束后的算子相关记录，查询该函数必须具有 sysadmin 和 monitoradmin 权限。

  返回值类型：record

- DBE_PERF.get_global_operator_runtime()

  描述：汇聚当前用户数据库主节点上执行作业实时的算子相关记录，查询该函数必须具有 sysadmin 和 monitoradmin 权限。

  返回值类型：record

- DBE_PERF.get_global_statement_complex_history()

  描述：汇聚当前用户数据库主节点上复杂查询的历史记录，查询该函数必须具有 monitoradmin 权限。

  返回值类型：record

- DBE_PERF.get_global_statement_complex_history_table()

  描述：汇聚当前用户数据库主节点上复杂查询的历史记录（持久化），查询该函数必须具有 monitoradmin 权限。

  返回值类型：record

- DBE_PERF.get_global_statement_complex_runtime()

  描述：汇聚当前用户数据库主节点上复杂查询的实时信息，查询该函数必须具有 sysadmin 和 monadmin 权限。

返回值类型：record

- DBE_PERF.get_global_memory_node_detail()

  描述：汇聚所有节点某个数据库节点内存使用情况，查询该函数必须具有 monitoradmin 权限。

  返回值类型：record

- DBE_PERF.get_global_shared_memory_detail()

  描述：汇聚所有节点已产生的共享内存上下文的使用信息，查询该函数必须具有 monitoradmin 权限。

  返回值类型：record

- DBE_PERF.get_global_statio_all_indexes()

  描述：汇聚所有节点当前数据库中的每个索引行，显示特定索引的 I/O 的统计，查询该函数必须具有 sysadmin 权限。

  返回值类型：record

- DBE_PERF.get_summary_stat_all_tables()

  描述：显示汇聚各节点数据中每个表（包括 TOAST 表）的一行的统计信息

  返回值类型：record

- DBE_PERF.get_global_stat_all_tables()

  描述：显示各节点数据中每个表（包括 TOAST 表）的一行的统计信息。

  返回值类型：record

- DBE_PERF.get_local_toastname_and_toastindexname()

  描述：提供本地 toast 表的 name 和 index 和其关联表的对应关系，查询该函数必须具有 sysadmin 权限。

  返回值类型：record

- DBE_PERF.get_summary_statio_all_indexes()

  描述：统计所有节点当前数据库中的每个索引行，显示特定索引的 I/O 的统计，查询该函数必须具有 sysadmin 权限。

  返回值类型：record

- DBE_PERF.get_global_statio_all_sequences()

  描述：提供命名空间中所有 sequences 的 IO 状态信息，查询该函数必须具有 sysadmin 权限。

返回值类型：record

- DBE_PERF.get_global_statio_all_tables()

  描述：汇聚各节点的数据库中每个表 I/O 的统计，查询该函数必须具有 sysadmin 权限。

  返回值类型：record

- DBE_PERF.get_summary_statio_all_tables()

  描述：统计 panweidb 内数据库中每个表 I/O 的统计，查询该函数必须具有 sysadmin 权限。

  返回值类型：record

- DBE_PERF.get_local_toast_relation()

  描述：提供本地 toast 表的 name 和其关联表的对应关系，查询该函数必须具有 sysadmin 权限

  返回值类型：record

- DBE_PERF.get_global_statio_sys_indexes()

  描述：汇聚各节点的命名空间中所有系统表索引的 IO 状态信息，查询该函数必须具有 sysadmin 权限。

  返回值类型：record

- DBE_PERF.get_summary_statio_sys_indexes()

  描述：统计各节点的命名空间中所有系统表索引的 IO 状态信息，查询该函数必须具有 sysadmin 权限。

  返回值类型：record

- DBE_PERF.get_global_statio_sys_sequences()

  描述：提供命名空间中所有系统表为 sequences 的 IO 状态信息，查询该函数必须具有 sysadmin 权限。

  返回值类型：record

- DBE_PERF.get_global_statio_sys_tables()

  描述：提供各节点的命名空间中所有系统表的 IO 状态信息，查询该函数必须具有 sysadmin 权限。

  返回值类型：record

- DBE_PERF.get_summary_statio_sys_tables()

描述：panweidb 内汇聚命名空间中所有系统表的 IO 状态信息，查询该函数必须具有 sysadmin 权限。

返回值类型：record

- DBE_PERF.get_global_statio_user_indexes()

  描述：各节点的命名空间中所有用户关系表索引的 IO 状态信息，查询该函数必须具有 sysadmin 权限。

  返回值类型：record

- DBE_PERF.get_summary_statio_user_indexes()

  描述：panweidb 内汇聚命名空间中所有用户关系表索引的 IO 状态信息，查询该函数必须具有 sysadmin 权限。

  返回值类型：record

- DBE_PERF.get_global_statio_user_sequences()

  描述：显示各节点的命名空间中所有用户的 sequences 的 IO 状态信息，查询该函数必须具有 sysadmin 权限。

  返回值类型：record

- DBE_PERF.get_global_statio_user_tables()

  描述：显示各节点的命名空间中所有用户关系表的 IO 状态信息，查询该函数必须具有 sysadmin 权限。

  返回值类型：record

- DBE_PERF.get_summary_statio_user_tables()

  描述：panweidb 内汇聚命名空间中所有用户关系表的 IO 状态信息，查询该函数必须具有 sysadmin 权限。

  返回值类型：record

- DBE_PERF.get_stat_db_cu()

  描述：视图查询 panweidb 各个节点，每个数据库的 CU 命中情况，查询该函数必须具有 sysadmin 权限。

  返回值类型：record

- DBE_PERF.get_global_stat_all_indexes()

  描述：汇聚所有节点数据库中每个索引的统计信息，查询该函数必须具有 sysadmin 权限。

  返回值类型：record

- DBE_PERF.get_summary_stat_all_indexes()

  描述：统计所有节点数据库中每个索引的统计信息，查询该函数必须具有 sysadmin 权限。

  返回值类型：record

- DBE_PERF.get_global_stat_sys_tables()

  描述：汇聚各节点 pg_catalog、information_schema 模式的所有命名空间中系统表的统计信息，查询该函数必须具有 sysadmin 权限。

  返回值类型：record

- DBE_PERF.get_summary_stat_sys_tables()

  描述：统计各节点 pg_catalog、information_schema 模式的所有命名空间中系统表的统计信息，查询该函数必须具有 sysadmin 权限。

  返回值类型：record

- DBE_PERF.get_global_stat_sys_indexes()

  描述：汇聚各节点 pg_catalog、information_schema 模式中所有系统表的索引状态信息，查询该函数必须具有 sysadmin 权限。

  返回值类型：record

- DBE_PERF.get_summary_stat_sys_indexes()

  描述：统计各节点 pg_catalog、information_schema 模式中所有系统表的索引状态信息，查询该函数必须具有 sysadmin 权限。

  返回值类型：record

- DBE_PERF.get_global_stat_user_tables()

  描述：汇聚所有命名空间中用户自定义普通表的状态信息，查询该函数必须具有 monitoradmin 权限。

  返回值类型：record

- DBE_PERF.get_summary_stat_user_tables()

  描述：统计所有命名空间中用户自定义普通表的状态信息，查询该函数必须具有 monitoradmin 权限。

  返回值类型：record

- DBE_PERF.get_global_stat_user_indexes()

  描述：汇聚所有数据库中用户自定义普通表的索引状态信息，查询该函数必须具有 sysadmin 权限。

返回值类型：record

- DBE_PERF.get_summary_stat_user_indexes()

  描述：统计所有数据库中用户自定义普通表的索引状态信息，查询该函数必须具有 sysadmin 权限。

  返回值类型：record

- DBE_PERF.get_global_stat_database()

  描述：汇聚所有节点数据库统计信息，查询该函数必须具有 sysadmin 权限。

  返回值类型：record

- DBE_PERF.get_global_stat_database_conflicts()

  描述：统计所有节点数据库统计信息，查询该函数必须具有 sysadmin 权限。

  返回值类型：record

- DBE_PERF.get_global_stat_xact_all_tables()

  描述：汇聚命名空间中所有普通表和 toast 表的事务状态信息，查询该函数必须具有 sysadmin 权限。

  返回值类型：record

- DBE_PERF.get_summary_stat_xact_all_tables()

  描述：统计命名空间中所有普通表和 toast 表的事务状态信息，查询该函数必须具有 sysadmin 权限。

  返回值类型：record

- DBE_PERF.get_global_stat_xact_sys_tables()

  描述：汇聚所有节点命名空间中系统表的事务状态信息，查询该函数必须具有 sysadmin 权限。

  返回值类型：record

- DBE_PERF.get_summary_stat_xact_sys_tables()

  描述：统计所有节点命名空间中系统表的事务状态信息，查询该函数必须具有 sysadmin 权限。

  返回值类型：record

- DBE_PERF.get_global_stat_xact_user_tables()

描述：汇聚所有节点命名空间中用户表的事务状态信息，查询该函数必须具有 sysadmin 权限。

返回值类型：record

- DBE_PERF.get_summary_stat_xact_user_tables()

  描述：统计所有节点命名空间中用户表的事务状态信息，查询该函数必须具有 sysadmin 权限。

  返回值类型：record

- DBE_PERF.get_global_stat_user_functions()

  描述：汇聚所有节点命名空间中用户定义函数的事务状态信息，查询该函数必须具有 sysadmin 权限。

  返回值类型：record

- DBE_PERF.get_global_stat_xact_user_functions()

  描述：统计所有节点命名空间中用户定义函数的事务状态信息，查询该函数必须具有 sysadmin 权限。

  返回值类型：record

- DBE_PERF.get_global_stat_bad_block()

  描述：汇聚所有节点表、索引等文件的读取失败信息，查询该函数必须具有 sysadmin 权限。

  返回值类型：record

- DBE_PERF.get_global_file_redo_iostat()

  描述：统计所有节点表、索引等文件的读取失败信息，查询该函数必须具有 sysadmin 权限。

  返回值类型：record

- DBE_PERF.get_global_file_iostat()

  描述：汇聚所有节点数据文件 IO 的统计，查询该函数必须具有 sysadmin 权限。

  返回值类型：record

- DBE_PERF.get_global_locks()

  描述：汇聚所有节点的锁信息，查询该函数必须具有 sysadmin 和 monadmin 权限。

  返回值类型：record

- DBE_PERF.get_global_replication_slots()

  描述：汇聚所有节点上逻辑复制信息，查询该函数必须具有 sysadmin 和 monadmin 权限。

  返回值类型：record

- DBE_PERF.get_global_bgwriter_stat()

  描述：汇聚所有节点后端写进程活动的统计信息，查询该函数必须具有 sysadmin 权限。

  返回值类型：record

- DBE_PERF.get_global_replication_stat()

  描述：汇聚各节点日志同步状态信息，如发起端发送日志位置，收端接收日志位置等，查询该函数必须具有 sysadmin 和 monadmin 权限。

  返回值类型：record

- DBE_PERF.get_global_transactions_running_xacts()

  描述：汇聚各节点运行事务的信息，查询该函数必须具有 sysadmin 权限。

  返回值类型：record

- DBE_PERF.get_summary_transactions_running_xacts()

  描述：统计各节点运行事务的信息，查询该函数必须具有 sysadmin 权限。

  返回值类型：record

- DBE_PERF.get_global_transactions_prepared_xacts()

  描述：汇聚各节点当前准备好进行两阶段提交的事务的信息，查询该函数必须具有 sysadmin 权限。

  返回值类型：record

- DBE_PERF.get_summary_transactions_prepared_xacts()

  描述：统计各节点当前准备好进行两阶段提交的事务的信息，查询该函数必须具有 sysadmin 权限。

  返回值类型：record

- DBE_PERF.get_summary_statement()

  描述：汇聚各节点历史执行语句状态信息，查询该函数必须具有 monitor admin 和 sysadmin 权限。

  返回值类型：record

- DBE_PERF.get_global_statement_count()

描述：汇聚各节点 SELECT，UPDATE，INSERT，DELETE，响应时间信息（TOTAL,AVG, MIN, MAX），查询该函数必须具有 sysadmin 权限。

返回值类型：record

- DBE_PERF.get_global_config_settings()

  描述：汇聚各节点 GUC 参数配置信息，查询该函数必须具有 sysadmin 权限。

  返回值类型：record

- DBE_PERF.get_global_wait_events()

  描述：汇聚各节点 wait events 状态信息，查询该函数必须具有 sysadmin 权限。

  返回值类型：record

- DBE_PERF.get_statement_responsetime_percentile()

  描述：获取 panweidbSQL 响应时间 P80，P95 分布信息，查询该函数必须具有 sysadmin 权限。

  返回值类型：record

- DBE_PERF.get_summary_user_login()

  描述：统计 panweidb 各节点用户登入登出次数信息，查询该函数必须具有 sysadmin 权限。

  返回值类型：record

- DBE_PERF.get_global_record_reset_time()

  描述：汇聚 panweidb 统计信息重置（重启，主备倒换，数据库删除）时间，查询该函数必须具有 sysadmin 权限。

  返回值类型：record

- DBE_PERF.track_memory_context(context_list text)

  描述：设置需要统计内存申请详细信息的内存上下文。入参为内存上下文的名称，使用“，”分隔，如“ThreadTopMemoryContext, SessionCacheMemoryContext”，注意该内存上下文名称是上下文敏感的。此外，单个内存上下文的长度为 63，超过的部分会被截断。而且一次能够统计的内存上下文上限为 16 个，设置超过 16 个内存上下文会设置失败。每一次调用该函数都会将上次统计的结果清空，

当入参指定为“”时，表示取消该统计功能。只有初始用户（super user）或者具有 monadmin 权限的用户可以执行该函数。

返回值类型：boolean

- DBE_PERF.track_memory_context_detail()

  描述：获取 DBE_PERF.track_memory_context 函数指定的内存上下文的内存申请详细信息。返回值的定义见视图 DBE_PERF.track_memory_context_detail。只有初始用户（super user）或者具有 monadmin 权限的用户可以执行该函数。

  返回值类型：record

- pg_stat_get_mem_mbytes_reserved(tid)

  描述：统计资源管理相关变量值，仅用于定位问题使用。

  参数：线程 id。

  返回值类型：text

- gs_wlm_user_resource_info(name text)

  描述：查询具体某个用户的资源限额和资源使用情况。

  返回值类型：record

- pg_stat_get_file_stat()

  描述：通过对数据文件 IO 的统计，反映数据的 IO 性能，用以发现 IO 操作异常等性能问题。

  返回值类型: record

- pg_stat_get_redo_stat()

  描述：用于统计会话线程日志回放情况。

  返回值类型: record

- pg_stat_get_status(int8)

  描述：可以检测当前实例中工作线程（backend thread）以及辅助线程（auxiliary thread）的阻塞等待情况。

  返回值类型：record

- get_local_rel_iostat()

  描述：查询当前节点的数据文件 IO 状态累计值。

  返回值类型：record

- DBE_PERF.get_global_rel_iostat()

描述：汇聚所有节点数据文件 IO 的统计，查询该函数必须具有 sysadmin 权限。

返回值类型：record

- DBE_PERF.global_threadpool_status()

  描述：显示在所有节点上的线程池中工作线程及会话的状态信息。函数返回信息具体字段参考：开发者指南->schema->DBE_PERF Schema->Session/Thread->GLOBAL_THREADPOOL_STATUS 章节。

  返回值类型：record

- remote_bgwriter_stat()

  描述：显示数据库所有实例的 bgwriter 线程刷页信息，候选 buffer 链中页面个数，buffer 淘汰信息（本节点除外、DN 上不可使用）。

  返回值类型：record

- pv_os_run_info

  描述：显示当前操作系统运行的状态信息，具体字段信息参考：开发者指南->系统表和系统视图->系统视图->GS_OS_RUN_INFO 章节。

  参数：nan

  返回值类型：setof record

- pv_session_stat

  描述：以会话线程或 AutoVacuum 线程为单位，统计会话状态信息，具体字段信息参考：开发者指南->系统表和系统视图->系统视图->GS_SESSION_STAT 章节。

  参数：nan

  返回值类型：setof record

- pv_session_time

  描述：用于统计会话线程的运行时间信息，及各执行阶段所消耗时间，具体字段信息参考：开发者指南->系统表和系统视图->系统视图->GS_SESSION_TIME 章节)。

  参数：nan

  返回值类型：setof record

- pg_stat_get_db_temp_bytes

描述：用于统计通过数据库查询写入临时文件的数据总量。计算所有临时文件，不论为什么创建临时文件，而且不管 log_temp_files 设置。

参数：oid

返回值类型：bigint

- pg_stat_get_db_temp_files

  描述：通过数据库查询创建的临时文件数量。计算所有临时文件，不论为什么创建临时文件（比如排序或者哈希），而且不管 log_temp_files 设置。

  参数：oid

  返回值类型：bigint

- remote_candidate_stat()

  描述：用于显示数据库所有实例的检查点信息和各类日志刷页情况（本节点除外），集中式不支持。

  返回值类型：record

- dbe_perf.gs_stat_activity_timeout(int)

  描述：获取当前节点上执行时间超过超时阈值的查询作业信息。需要 GUC 参数 track_activities 设置为 on 才能正确返回结果。超时阈值的取值范围是 0~2147483。

  返回值类型：setof record

名称	类型	描述
database	name	用户会话连接的数据库名称。
pid	bigint	后台线程 ID。
sessionid	bigint	会话 ID。
usesysid	oid	登录该后台的用户 OID。
application_name	text	连接到该后台的应用名。
query	text	该后台正在执行的查询。
xact_start	timesta	启动当前事务的时间。

名称	类型	描述
	mptz	
query_start	timesta mptz	开始当前查询的时间。
query_id	bigint	查询语句 ID。

- gs_wlm_user_resource_info(name text)

  描述：查询具体某个用户的资源限额和资源使用情况。普通用户只能查询到自己相关的信息，管理员权限的用户可以查看全部用户的信息。

  返回值类型：record

- create_wlm_instance_statistics_info

  描述：将当前实例的历史监控数据进行持久化保存。

  参数：nan

  返回值类型：integer

- gs_session_memory

  描述：统计 Session 级别的内存使用情况，包含执行作业在数据节点上 Postgres 线程和 Stream 线程分配的所有内存。

说明

若 GUC 参数 enable_memory_limit(参考：开发者指南->GUC 参数说明->资源消耗->内存章节的 enable_memory_limit 参数)=off，该函数不能使用。

返回值类型：record

表 8 返回值说明

名称	类型	描述
sessid	text	线程启动时间+线程标识。
init_mem	integer	当前正在执行作业进入执行器前已分配的内存，单位 MB。
used_mem	integer	当前正在执行作业已分配的内存，单位 MB。

名称	类型	描述
peak_mem	integer	当前正在执行作业已分配的内存峰值，单位 MB。

- gs_wlm_persistent_user_resource_info()

  描述：将当前所有的用户资源使用统计信息归档到 gs_wlm_user_resource_history 系统表中，只有 sysadmin 有权限查询。

  返回值类型：record

- create_wlm_operator_info(int flag)

  描述：将当前内存中记录的 TopSQL 算子级别相关统计信息清理，当传入的参数大于 0 时，会将这部分信息归档到 gs_wlm_operator_info 和 gs_wlm_ec_operator_info 中，否则不会归档。该函数只有 sysadmin 权限的用户可以执行。

  返回值类型：int

- GS_ALL_NODEGROUP_CONTROL_GROUP_INFO(text)

  描述：提供了所有逻辑数据库实例的控制组信息。该函数在调用的时候需要指定要查询逻辑数据库实例的名称。例如要查询'installation'逻辑数据库实例的控制组信息：

```
SELECT * FROM GS_ALL_NODEGROUP_CONTROL_GROUP_INFO('installation')
```

  返回值类型：record

  函数返回字段如下：

名称	类型	描述
name	text	控制组的名称。
type	text	控制组的类型。
gid	bigint	控制组 ID。
classgid	bigint	Workload 所属 Class 的控制组 ID。
class	text	Class 控制组。
workload	text	Workload 控制组。

名称	类型	描述
shares	bigint	控制组分配的 CPU 资源配额。
limits	bigint	控制组分配的 CPU 资源限额。
wdlevel	bigint	Workload 控制组层级。
cpucores	text	控制组使用的 CPU 核的信息。

- gs_total_nodegroup_memory_detail

  描述：返回当前数据库逻辑数据库使用内存的信息，单位为 MB 得到一个逻辑数据

  库。

  返回值类型：setof record

- local_redo_time_count()

  描述：返回本节点各个回放线程的各个流程的耗时统计（仅在备机上有有效数据）。

  返回值如下：

  local_redo_time_count 返回参数说明。

字段名	描述
thread_name	线程名字
step1_total	step1 的总时间，每个线程对应的流程如下： 极致 RTO： batch redo：从队列中获取一条日志 redo manager：从队列中获取一条日志 redo worker：从队列中获取一条日志 trxn manager：从队列中读取一条日志 trxn worker：从队列中读取一条日志 read worker：从文件中读取一次 xlog page（整体） read page worker：从队列中获取一个日志 startup：从队列中获取一个日志 并行回放： page redo：从队列中获取一条日志

字段名	描述
	startup：读取一条日志
step1_count	step1 的统计次数
step2_total	step2 的总时间，每个线程对应的流程如下： 极致 RTO： batch redo：处理日志（整体） redo manager：处理日志（整体） redo worker：处理日志（整体） trxn manager：处理日志（整体） trxn worker：处理日志（整体） redo worker：读取 xlog page 耗时 read page worker：生成和发送 lsn forwarder startup：check stop(是否回放到指定位置） 并行回放： page redo：处理日志（整体） startup：check stop(是否回放到指定位置）
step2_count	step2 的统计次数
step3_total	step3 的总时间，每个线程对应的流程如下： 极致 RTO： batch redo：更新 standbystate redo manager：数据日志处理 redo worker：回放 page 也日志（整体） trxn manager：更新 flush lsn trxn worker：回放日志处理 redo worker：推进 xlog segment read page worker：获取一个新的 item startup：redo delay（延迟回放特性等待时间） 并行回放： page redo：更新 standbystate startup：redo delay（延迟回放特性等待时间）
step3_count	step3 的统计次数

字段名	描述
step4_total	step4 的总时间，每个线程对应的流程如下： 极致 RTO： batch redo：解析 xlog redo manager：DDL 处理 redo worker：读取数据 page 页 trxn manager：同步等待时间 trxn worker：更新本线程 lsn read page worker：将日志放入分发线程 startup：分发（整体） 并行回放： page redo：undo 日志回放 startup：分发（整体）
step4_count	step4 的统计次数
step5_total	step5 的总时间，每个线程对应的流程如下： 极致 RTO： batch redo：分发给 redo manager redo manager：分发给 redo worker redo worker：回放数据 page 页的日志 trxn manager：分发给 trxn worker trxn worker：强同步 wait 时间 read page worker：更新本线程 lsn startup：日志 decode 并行回放： page redo：sharetrxn 日志回放 startup：日志回放
step5_count	step5 的统计次数
step6_total	step6 的总时间，每个线程对应的流程如下： 极致 RTO： redo worker：回放非数据页 page 日志 trxn manager：全局 lsn 更新 read page worker：日志 crc 校验

字段名	描述
	并行回放： page redo：synctrxn 日志回放 startup：强同步等待
step6_count	step6 的统计次数
step7_total	step7 的总时间，每个线程对应的流程如下： 极致 RTO： redo worker：fsm 更新 并行回放： page redo：single 日志回放
step7_count	step7 的统计次数
step8_total	step8 的总时间，每个线程对应的流程如下： 极致 RTO： redo worker：强同步等待 并行回放： page redo：all workers do 日志回放
step8_count	step8 的统计次数
step9_total	step9 的总时间，每个线程对应的流程如下： 极致 RTO： 无 并行回放： page redo：muliti workers do 日志回放
step9_count	step9 的统计次数

- local_xlog_redo_statics()

  描述：返回本节点已经回放的各个类型类型的日志统计信息（仅在备机上有有效数据）。

返回值如下：

表 9 local_xlog_redo_statics 返回参数说明

字段名	描述

字段名	描述
xlog_type	日志类型
rmid	resource manager id
info	xlog operation
num	日志个数
extra	针对 page 回放日志和 xact 日志有有效值。page 页回放日志标识从磁盘读取 page 的个数。xact 日志表示删除文件的个数。

❖ gs_get_shared_memctx_detail(text)

描述：返回指定内存上下文上的内存申请的详细信息，包含每一处内存申请所在的文件、行号和大小（同一文件同一行大小会做累加）。只支持查询通过 pg_shared_memory_detail 视图查询出来的内存上下文，入参为内存上下文名称（即 pg_shared_memory_detail 返回结果的 contextname 列）。查询该函数必须具有 sysadmin 权限或者 monitor admin 权限。

返回值类型：setof record

名称	类型	描述
file	text	申请内存所在文件的文件名。
line	int8	申请内存所在文件的代码行号。
size	int8	申请的内存大小，同一文件同一行多次申请会做累加。

说明

该视图不支持 release 版本小型化场景。

❖ gs_get_session_memctx_detail(text)

描述：返回指定内存上下文上的内存申请的详细信息，包含每一处内存申请所在的文件、行号和大小（同一文件同一行大小会做累加）。仅在

线程池模式下生效。只支持查询通过 gs_session_memory_context 视图查询出来的内存上下文，入参为内存上下文名称（即 gs_session_memory_context 返回结果的 contextname 列）。查询该函数必须具有 sysadmin 权限或者 monitor admin 权限。

返回值类型：setof record

名称	类型	描述
file	text	申请内存所在文件的文件名。
line	int8	申请内存所在文件的代码行号。
size	int8	申请的内存大小，同一文件同一行多次申请会做累加。

说明

该视图仅在线程池模式下生效，且该视图不支持 release 版本小型化场景。

- gs_get_thread_memctx_detail(tid,text)

  描述：返回指定内存上下文上的内存申请的详细信息，包含每一处内存申请所在的文件，行号和大小（同一文件同一行大小会做累加）。只支持查询通过 gs_thread_memory_context 视图查询出来的内存上下文，第一个入参为线程 id（即 gs_thread_memory_context 返回数据的 tid 列），第二个参数为内存上下文名称（即 gs_thread_memory_context 返回数据的 tid 列），第二个参数为内存上下文名称（即 pv_thread_memory_context 返回数据的 contextname 列）。查询该函数必须具有 sysadmin 权限或者 monitor admin 权限。

  返回值类型：setof record

名称	类型	描述
file	text	申请内存所在文件的文件名。
line	int8	申请内存所在文件的代码行号。

名称	类型	描述
size	int8	申请的内存大小，单位为 btye，同一文件同一行多次申请会做累加。

4.1.4.30.全局临时表函数

- pg_get_gtt_relstats(relOid)

  描述：显示当前会话指定的全局临时表的基本信息。

  参数：全局临时表的 OID。

  返回值类型：record

  示例：

```
panweidb=# select * from pg_get_gtt_relstats(74069);
 relfilenode | relpages | reltuples | relallvisible | relfrozenxid | relminmxid
-------------+----------+-----------+---------------+--------------+------------
       74069 |       58 |     13000 |             0 |        11151 |          0
(1 row)
```

- pg_get_gtt_statistics(relOid, attnum, ''::text)

  描述：显示当前会话指定的全局临时表的单列统计信息。

  参数：全局临时表的 OID 和属性 attnum。

  返回值类型：record

  示例：

```
panweidb=# select * from pg_get_gtt_statistics(74069,1,''::text);
 starelid | starelkind | staattnum | stainherit | stanullfrac | stawidth | stadistinct | staki
nd1 | stakind2 | stakind3 | stakind4 | stakind5 | staop1 | staop2 | staop3 | staop4 | stao
p5 | stanumbers1 | stanumbers2 | stanumbers3 | stanu
mbers4 | stanumbers5 |

                          stavalues1

                                                | stavalues2 | stavalues3 | stavalues4 | staval
ues5 | stadndistinct | staextinfo
----------+------------+-----------+------------+-------------+----------+-------------+
----------+----------+----------+----------+----------+--------+--------+--------+-----
---+--------+-------------+-------------+-------------+------
```

```
-------+------------+---------------------------------------------------------------------
---
-------------------------------------------------------------------------------------------
---
-----------------------------------------------------------------------+-----------+---
---------+-----------+-----------+--------------+-----------
  74069 | c       |      1 | f       |       0 |     4 |      -1 |     2 |     3 |     0 |     0 |
 0 |    97 |    97 |    0 |    0 |    0 |         | {1}       |         |
   |         | {1,130,260,390,520,650,780,910,1040,1170,1300,1430,1560,1690,1820,
1950,2080,2210,2340,2470,2600,2730,2860,2990,3120,3250,3380,3510,3640,3770,
3900,4030,4160,4290,4420,4550,4680,4810,4940,5070,5200,5330,5460,5590,57
  20,5850,5980,6110,6240,6370,6500,6630,6760,6890,7020,7150,7280,7410,7540,7
670,7800,7930,8060,8190,8320,8450,8580,8710,8840,8970,9100,9230,9360,9490,9
620,9750,9880,10010,10140,10270,10400,10530,10660,10790,10920,11050,11180,
11310,11440,1
  1570,11700,11830,11960,12090,12220,12350,12480,12610,12740,12870,13000} |
      |         |         |         |         0 |
(1 row)
```

❖ pg_gtt_attached_pid(relOid)

描述：显示正在使用指定全局临时表的所有线程 pid。

参数：全局临时表的 OID。

返回值类型：record

示例：

```
panweidb=# select * from pg_gtt_attached_pid(74069);
 relid |      pid
-------+-----------------
 74069 | 139648170456832
 74069 | 139648123270912
(2 rows)
```

❖ dbe_perf.get_global_full_sql_by_timestamp(start_timestamp timestamp, end_timestamp timestamp)

描述：获取实例级的全量 SQL(Full SQL)信息。

返回值类型：record

表 1 dbe_perf.get_global_full_sql_by_timestamp 参数说明

参数	类型	描述
start_timestamp	timestamp	SQL 启动时间范围的开始时间点。
end_timestamp	timestamp	SQL 启动时间范围的结束时间点。

- dbe_perf.get_global_slow_sql_by_timestamp(start_timestamp timestamp, end_timestamp timestamp)

  描述：获取实例级的慢 SQL(Slow SQL)信息。

  返回值类型：record

表 2 dbe_perf.get_global_slow_sql_by_timestamp 参数说明

参数	类型	描述
start_timestamp	timestamp	SQL 启动时间范围的开始时间点。
end_timestamp	timestamp	SQL 启动时间范围的结束时间点。

- statement_detail_decode(detail text, format text, pretty bool)

  解析全量/慢 SQL 语句中的 details 字段的信息。

表 3 statement_detail_decode 参数说明

参数	类型	描述
detail	text	SQL 语句产生的事件的集合（不可读）。
format	text	解析输出格式，取值为 plaintext 或 json。
pretty	bool	当 format 为 plaintext 时，是否以优雅的格式展示： true 表示通过 “\n” 分隔事件。 false 表示通过 “，” 分隔事件。

- pg_list_gtt_relfrozenxids()

  描述：显示各会话的冻结事务 xid。

  pid=0 的行，显示所有会话中最老的冻结事务 xid。

  参数：无。

  返回值类型：record

  示例：

```
panweidb=# select * from pg_list_gtt_relfrozenxids();
    pid     | relfrozenxid
```

```
----------------+-------------
139648123270912|     11151
139648170456832|     11155
              0|     11151
(3 rows)
```

4.1.4.31.故障注入系统函数

- gs_fault_inject(int64, text, text, text, text, text)

  描述：该函数不能调用，调用时会报 WARNING 信息："unsupported fault injection"，并不会对数据库产生任何影响和改变。

  参数：int64 注入故障类型（0：CLOG 扩展页面，1：读取 CLOG 页面，2：强制死锁）。

  - text 第二个入参在第一入参为 2 的模式下若为“1”则死锁，其余不死锁；第二个入参在第一入参为 0，1 时，表示 CLOG 开始扩展或读取的起始页面号。
  - text 第三个入参在第一入参为 0，1 时，表示扩展或读取的页面个数。
  - text 第四到六入参为预留参数。

  返回值类型：int64

4.1.4.32.AI 特性函数

- gs_index_advise(text)

  描述：针对单条查询语句推荐索引。

  参数：SQL 语句字符串

  返回值类型：record

  示例请参见：开发者指南->AI 特性->AI4DB 数据库自治运维->DBMIND 的 AI 子功能->INDEX-ADVISOR 索引推荐章节的单 query 索引推荐。

- hypopg_create_index(text)

  描述：创建虚拟索引。

  参数：创建索引语句的字符串

  返回值类型：record

示例请参见：开发者指南->AI 特性->AI4DB 数据库自治运维->DBMIND 的 AI 功能->INDEX-ADVISOR 索引推荐章节的虚拟索引。

- hypopg_display_index()

  描述：显示所有创建的虚拟索引信息。

  参数：无

  返回值类型：record

  示例请参见：开发者指南->AI 特性->AI4DB 数据库自治运维->DBMIND 的 AI 功能->INDEX-ADVISOR 索引推荐章节的虚拟索引。

- hypopg_drop_index(oid)

  描述：删除指定的虚拟索引。

  参数：索引的 oid

  返回值类型：bool

  示例请参见：开发者指南->AI 特性->AI4DB 数据库自治运维->DBMIND 的 AI 功能->INDEX-ADVISOR 索引推荐章节的虚拟索引。

- hypopg_reset_index()

  描述：清除所有虚拟索引。

  参数：无

  返回值类型：无

  示例请参见：开发者指南->AI 特性->AI4DB 数据库自治运维->DBMIND 的 AI 功能->INDEX-ADVISOR 索引推荐章节的虚拟索引。

- hypopg_estimate_size(oid)

  描述：估计指定索引创建所需的空间大小。

  参数：索引的 oid

  返回值类型：int8

  示例请参见：开发者指南->AI 特性->AI4DB 数据库自治运维->DBMIND 的 AI 功能->INDEX-ADVISOR 索引推荐章节的虚拟索引。

- check_engine_status(ip text, port text)

  描述：测试给定的 ip 和 port 上是否有 predictor engine 提供服务。

  参数：predictor engine 的 ip 地址和端口号。

  返回值类型：text

示例请参见：开发者指南->AI 特性->AI in DB：数据库内 AI 功能->智能 Explain：SQL 语句查询时间预测->使用指导。

- encode_plan_node(optname text, orientation text, strategy text, options text, dop int8, quals text, projection text)

  描述：对入参的计划算子信息进行编码。

  参数：计划算子信息。

  返回值类型：text。

  说明：

  该函数为内部功能调用函数，不建议用户直接使用。

- model_train_opt(template text, model text)

  描述：训练给定的查询性能预测模型。

  参数：性能预测模型的模板名和模型名。

  返回值类型：tartup_time_accuracy FLOAT8、 total_time_accuracy FLOAT8、 rows_accuracy FLOAT8、 peak_memory_accuracy FLOAT8

  示例请参见：开发者指南->AI 特性->AI in DB：数据库内 AI 功能->智能 Explain：SQL 语句查询时间预测->使用指导。

- track_model_train_opt(ip text, port text)

  描述：返回给定 ip 和 port predictor engine 的训练日志地址。

  参数：predictor engine 的 ip 地址和端口号。

  返回值类型：text

  示例请参见：开发者指南->AI 特性->AI in DB：数据库内 AI 功能->智能 Explain：SQL 语句查询时间预测->使用指导。

- encode_feature_perf_hist(datname text)

  描述：将目标数据库已收集的历史计划算子进行编码。

  参数：数据库名。

  返回值类型：queryid bigint、 plan_node_id int、 parent_node_id int、 left_child_id int、 right_child_id int, encode text、 startup_time bigint、 total_time bigint、 rows bigint、 peak_memory int

示例请参见：开发者指南->AI 特性->AI in DB：数据库内 AI 功能->智能 Explain：SQL 语句查询时间预测->使用指导。

- gather_encoding_info(datname text)

  描述：调用 encode_feature_perf_hist，将编码好的数据进行持久化保存。

  参数：数据库名。

  返回值类型：int

  示例请参见：开发者指南->AI 特性->AI in DB：数据库内 AI 功能->智能 Explain：SQL 语句查询时间预测->使用指导。

- db4ai_predict_by_bool (text, VARIADIC "any")

  描述：获取返回值为布尔型的模型进行模型推断任务。此函数为内部调用函数，建议直接使用语法 PREDICT BY 进行推断任务。

  参数：模型名称和推断任务的输入列。

  返回值类型：bool

- db4ai_predict_by_float4(text, VARIADIC "any")

  描述：获取返回值为 float4 的模型进行模型推断任务。此函数为内部调用函数，建议直接使用语法 PREDICT BY 进行推断任务。

  参数：模型名称和推断任务的输入列。

  返回值类型：float

- db4ai_predict_by_float8(text, VARIADIC "any")

  描述：获取返回值为 float8 的模型进行模型推断任务。此函数为内部调用函数，建议直接使用语法 PREDICT BY 进行推断任务。

  参数：模型名称和推断任务的输入列。

  返回值类型：float

- db4ai_predict_by_int32(text, VARIADIC "any")

  描述：获取返回值为 int32 的模型进行模型推断任务。此函数为内部调用函数，建议直接使用语法 PREDICT BY 进行推断任务。

  参数：模型名称和推断任务的输入列。

  返回值类型：int

- db4ai_predict_by_int64(text, VARIADIC "any")

描述：获取返回值为 int64 的模型进行模型推断任务。此函数为内部调用函数，建议直接使用语法 PREDICT BY 进行推断任务。

参数：模型名称和推断任务的输入列。

返回值类型：int

- db4ai_predict_by_numeric(text, VARIADIC "any")

  描述：获取返回值为 numeric 的模型进行模型推断任务。此函数为内部调用函数，建议直接使用语法 PREDICT BY 进行推断任务。

  参数：模型名称和推断任务的输入列。

  返回值类型：numeric

- db4ai_predict_by_text(text, VARIADIC "any")

  描述：获取返回值为字符型的模型进行模型推断任务。此函数为内部调用函数，建议直接使用语法 PREDICT BY 进行推断任务。

  参数：模型名称和推断任务的输入列。

  返回值类型：text

- db4ai_predict_by_float8_array(text, VARIADIC "any")

  描述：获取返回值为字符型的模型进行模型推断任务。此函数为内部调用函数，建议直接使用语法 PREDICT BY 进行推断任务。

  参数：模型名称和推断任务的输入列。

  返回值类型：text

- gs_explain_model(text)

  描述：获取返回值为字符型的模型进行模型解析文本化任务。

  参数：模型名称。

  返回值类型：text

4.1.4.33.动态数据脱敏函数

说明：

该函数为内部功能调用函数。

- creditcardmasking(col text, letter char default 'x')

  描述：将 col 字符串后四位之前的数字使用 letter 替换。

  参数：待替换的字符串、替换字符。

  返回值类型：text

- basicemailmasking(col text, letter char default 'x')

  描述：将 col 字符串中第一个'@'之前的字符使用 letter 替换。

  参数：待替换的字符串、替换字符。

  返回值类型：text

- fullemailmasking(col text, letter char default 'x')

  描述：将 col 字符串中出现最后一个'.'之前的字符(除'@'外)使用 letter 替换。

  参数：待替换的字符串、替换字符。

  返回值类型：text

- alldigitsmasking(col text, letter char default '0')

  描述：将 col 字符串中出现的数字使用 letter 替换。

  参数：待替换的字符串、替换字符。

  返回值类型：text

- shufflemasking(col text)

  描述：将 col 字符串中的字符乱序排列。

  参数：待替换的字符串、替换字符。

  返回值类型：text

- randommasking(col text)

  描述：将 col 字符串中的字符随机化。

  参数：待替换的字符串、替换字符。

  返回值类型：text

- regexpmasking

  描述：脱敏策略的内部函数，对字符进行正则表达式替换。

  参数：col text, reg text, replace_text text, pos INTEGER default 0, reg_len INTEGER default -1

  返回值类型：text

4.1.4.34.内部函数

PanWeiDB 中下列函数使用了内部数据类型，用户无法直接调用，在此章节列出。

❖ 选择率计算函数

areajoinsel	areasel	arraycontjoinsel	arraycontsel	contjoinsel	contsel	eqjoinsel
eqsel	iclikejoinsel	iclikesel	icnlikejoinsel	icnlikesel	icregexeqjoinsel	icregexeqsel
icregexnejoinsel	icregexnesel	likejoinsel	likesel	neqjoinsel	neqsel	nlikejoinsel
nlikesel	positionjoinsel	positionsel	regexeqjoinsel	regexeqsel	regexnejoinsel	regexnesel
scalargtjoinsel	scalargtsel	scalarltjoinsel	scalarltsel	tsmatchjoinsel	tsmatchsel	-

❖ 统计信息收集函数

array_typanalyze	range_typanalyze	ts_typanalyze
local_rto_stat		

❖ 排序内部功能函数

bpchar_sortsupport	bytea_sortsupport	date_sortsupport	numeric_sortsupport	timestamp_sortsupport

❖ 全文检索内部功能函数

dispell_init	dispell_lexize	dsimple_init	dsimple_lexize	dsnowball_init	dsnowball_lexize	dsynonym_init
dsynonym_lexize	gtsquery_compress	gtsquery_consistent	gtsquery_decompress	gtsquery_penalty	gtsquery_picksplit	gtsquery_same

gtsquery_union	ngram_end	ngram_lextype	ngram_start	pound_end	pound_lextype	pound_start
prsd_end	prsd_headline	prsd_lextype	prsd_start	thesaurus_init	thesaurus_lexize	zhprs_end
zhprs_getlexeme	zhprs_lextype	zhprs_start	-	-	-	-

❖ 内部类型处理函数

abstimerecv	euc_jis_2004_to_utf8	int2recv	line_recv	oidvectorrecv_extend	tidrecv	utf8_to_koi8u
anyarray_recv	euc_jp_to_mic	int2vectorrecv	lseg_recv	path_recv	time_recv	utf8_to_shift_jis_2004
array_recv	euc_jp_to_sjis	int4recv	macaddr_recv	pg_node_tree_recv	time_transform	utf8_to_sjis
ascii_to_mic	euc_jp_to_utf8	int8recv	mic_to_ascii	point_recv	timestamp_recv	utf8_to_uhc
ascii_to_utf8	euc_kr_to_mic	internal_out	mic_to_big5	poly_recv	timestamp_transform	utf8_to_win
big5_to_euc_tw	euc_kr_to_utf8	interval_recv	mic_to_euc_cn	pound_nexttoken	timestamptz_recv	uuid_recv

big5_to_mic	euc_tw_to_big5	interval_transform	mic_to_euc_jp	prsd_nexttoken	timetz_recv	varbit_recv
big5_to_utf8	euc_tw_to_mic	iso_to_koi8r	mic_to_euc_kr	range_recv	tinterval recv	varbit_transform
bit_recv	euc_tw_to_utf8	iso_to_mic	mic_to_euc_tw	rawrecv	tsqueryrecv	varchar_transform
boolrecv	float4recv	iso_to_win1251	mic_to_iso	record_recv	tsvectorrecv	varcharrecv
box_recv	float8recv	iso_to_win866	mic_to_koi8r	regclassrecv	txid_snapshot_recv	void_recv
bpcharrecv	gb18030_to_utf8	iso8859_1_to_utf8	mic_to_latin1	regconfigrecv	uhc_to_utf8	win_to_utf8
btoidsortsupport	gbk_to_utf8	iso8859_to_utf8	mic_to_latin2	regdictionaryrecv	unknownrecv	win1250_to_latin2
byteavecv	gin_extract_tsvector	johab_to_utf8	mic_to_latin3	regoperatorrecv	utf8_to_ascii	win1250_to_mic
byteawithoutorderwithequalcolre	gtsvector_compress	json_recv	mic_to_latin4	regoperrecv	utf8_to_big5	win1251_to_iso

cv						
cash_recv	gtsvector_consistent	koi8r_to_iso	mic_to_sjis	regprocedurerecv	utf8_to_euc_cn	win1251_to_koi8r
charrecv	gtsvector_decompress	koi8r_to_mic	mic_to_win1250	regprocrecv	utf8_to_euc_jis_2004	win1251_to_mic
cidr_recv	gtsvector_penalty	koi8r_to_utf8	mic_to_win1251	regtyperecv	utf8_to_euc_jp	win1251_to_win866
cidrecv	gtsvector_picksplit	koi8r_to_win1251	mic_to_win866	reltimerecv	utf8_to_euc_kr	win866_to_iso
circle_recv	gtsvector_same	koi8r_to_win866	namerecv	shift_jis_2004_to_euc_jis_2004	utf8_to_euc_tw	win866_to_koi8r
cstring_recv	gtsvector_union	koi8u_to_utf8	ngram_nexttoken	shift_jis_2004_to_utf8	utf8_to_gb18030	win866_to_mic
date_recv	hll_recv	latin1_to_mic	numeric_recv	sjis_to_euc_jp	utf8_to_gbk	win866_to_win1251
domain_recv	hll_trans_recv	latin2_to_mic	numeric_transform	sjis_to_mic	utf8_to_iso8859	xidrecv

euc_cn_to_mic	hstore_recv	latin2_to_win1250	nvarchar2recv	sjis_to_utf8	utf8_to_iso8859_1	xidrecv4
euc_cn_to_utf8	inet_recv	latin3_to_mic	oidrecv	smalldatetime_recv	utf8_to_johab	xml_recv
euc_jis_2004_to_shift_jis_2004	int1recv	latin4_to_mic	oidvectorrecv	textrecv	utf8_to_koi8r	cstore_tid_out
i16toi1	int16	int16_bool	int16eq	int16div	int16ge	int16gt
int16in	int16le	int16lt	int16mi	int16mul	int16ne	int16out
int16pl	int16recv	int16send	numeric_bool	int2vectorin_extend	int2vectorout_extend	int2vectorrecv_extend
int2vectorsend_extend	jsonb_in	jsonb_out	jsonb_recv	jsonb_send	complex_array_in	bool_int1
bool_int2	bool_int8	bpchar_float4	bpchar_float8	bpchar_int4	bpchar_int8	bpchar_numeric
bpchar_timestamp	f4toi1	f8toi1	float4_bpchar	float4_text	float4_varchar	float8_bpchar
float8_te	float8_v	i1tof4	i1tof8	i1toi2	i1toi4	i1toi8

xt	archar					
i2toi1	i4toi1	i8toi1	int1_avg_accum	int1_bool	int1_bpchar	int1_numeric
int1_nvarchar2	int1_text	int1_varchar	int1abs	int1and	int1cmp	int1div
int1eq	int1ge	int1gt	int1in	int1inc	int1larger	int1le
int1lt	int1mi	int1mod	int1mul	int1ne	int1not	int1or
int1out	int1pl	int1shl	int1shr	int1smaller	int1um	int1up
int1xor	int2_bool	int2_bpchar	int2_text	int2_varchar	int4_bpchar	int4_text
int4_varchar	int8_bool	int8_bpchar	int8_text	int8_varchar	job_submit	numeric_bpchar
numeric_int1	numeric_text	numeric_varchar	nvarchar2in	nvarchar2out	nvarchar2send	oidvectorin_extend
oidvectorout_extend	oidvectorsend_extend	rawcmp	raweq	rawge	rawgt	rawin
rawle	rawlike	rawlt	rawne	rawnlike	rawout	rawsend
text_float4	text_float8	text_int1	text_int2	text_int4	text_int8	text_numeric

timestamp_text	timestamp_varchar	varchar_float4	varchar_float8	varchar_int4	varchar_int8	varchar_numeric
xidout4	xidsend4	calculate_quantile_of	calculate_value_at	large_seq_rollback_ntree	large_seq_upgrade_ntree	-

❖ 聚合操作内部函数

array_agg_finalfn	array_agg_transfn	bytea_string_agg_finalfn	bytea_string_agg_transfn	date_list_agg_noarg2_transfn	date_list_agg_transfn	float4_list_agg_noarg2_transfn
float4_list_agg_transfn	float8_list_agg_noarg2_transfn	float8_list_agg_transfn	int2_list_agg_noarg2_transfn	int2_list_agg_transfn	int4_list_agg_noarg2_transfn	int4_list_agg_transfn
int8_list_agg_noarg2_transfn	int8_list_agg_transfn	interval_list_agg_noarg2_transfn	interval_list_agg_transfn	list_agg_finalfn	list_agg_noarg2_transfn	list_agg_transfn
median_float8_finalfn	median_interval_finalfn	median_transfn	mode_final	numeric_list_agg_noarg2_transfn	numeric_list_agg_transfn	ordered_set_transition
percentile_cont_float8_final	percentile_cont_interval_final	string_agg_finalfn	string_agg_transfn	timestamp_list_agg_noarg2_transfn	timestamp_list_agg_transfn	timestamptz_list_agg_noarg2_transfn

timestamptz_list_agg_transfn	checksumtext_agg_transfn	-	-	-	-	-
json_agg_finalfn	json_agg_transfn	json_object_agg_finalfn	json_object_agg_transfn	-	-	-

❖ 哈希内部功能函数

hashbeginscan	hashbuild	hashbuildempty	hashbulkdelete	hashcostestimate	hashendscan	hashgetbitmap
hashgettuple	hashinsert	hashmarkpos	hashmerge	hashrescan	hashrestrpos	hashvacuumcleanup
hashvarlena	-	-	-	-	-	-

❖ Btree 索引内部功能函数

cbtreebuild	cbtreecanreturn	cbtreecostestimate	cbtreegetbitmap	cbtreegettuple	btbeginscan	btbuild
btbuildempty	btbulkdelete	btcanreturn	btcostestimate	btendscan	btfloat4sortsupport	btfloat8sortsupport
btgetbitmap	btgettuple	btinsert	btint2sortsupport	btint4sortsupport	btint8sortsupport	btmarkpos
btmerge	btnamesortsupp	btrescan	btrestrp	bttextsor	btvacuumcleanu	cbtreeop

	ort		os	tsupport	p	tions

❖ GiST 索引内部功能函数

gist_box_compress	gist_box_consistent	gist_box_decompress	gist_box_penalty	gist_box_picksplit	gist_box_same	gist_box_union
gist_circle_compress	gist_circle_consistent	gist_point_compress	gist_point_consistent	gist_point_distance	gist_poly_compress	gist_poly_consistent
gistbeginscan	gistbuild	gistbuildempty	gistbulkdelete	gistcostestimate	gistendscan	gistgetbitmap
gistinsert	gistmarkpos	gistmerge	gistrescan	gistrestrpos	gistvacumcleanup	range_gist_compress
range_gist_decompress	range_gist_penalty	range_gist_picksplit	range_gist_same	range_gist_union	spg_kd_choose	spg_kd_config
spg_kd_picksplit	spg_quad_choose	spg_quad_config	spg_quad_inner_consistent	spg_quad_leaf_consistent	spg_quad_picksplit	spg_text_choose
spg_text_inner_consistent	spg_text_leaf_consistent	spg_text_picksplit	spgbeginscan	spgbuild	spgbuildempty	spgbulkdelete
spgcoste	spgends	spggetbi	spggettu	spginser	spgmark	spgmerg

stimate	can	tmap	ple	t	pos	e
spgrestrpos	spgvacuumcleanup	-	-	-	-	-

❖ Gin 索引内部功能函数

gin_cmp_prefix	gin_extract_tsquery	gin_tsquery_consistent	gin_tsquery_triconsistent	ginarrayconsistent	ginarrayextract	ginarraytriconsistent
ginbeginscan	ginbuild	ginbuildempty	ginbulkdelete	gincostestimate	ginendscan	gingetbitmap
gininsert	ginmarkpos	ginmerge	ginqueryarrayextract	ginrescan	ginrestrpos	ginvacuumcleanup
cginbuild	cgingetbitmap	-	-	-	-	-
gin_compare_jsonb	gin_consistent_jsonb	gin_consistent_jsonb_hash	gin_extract_jsonb gin_extract_jsonb_hash	cginbuild	gin_extract_jsonb_query	gin_triconsistent_jsonb

❖ Psort 索引内部函数

psortbuild	psortcanreturn	psortcostestimate	psortgetbitmap	psortgettuple

❖ Ubtree 索引内部函数

ubtbeginscan	ubtbuild	ubtbuildempty	ubtbulkdelete	ubtcanreturn
ubtcostestimate	ubtendscan	ubtgetbitmap	ubtgettuple	ubtinsert
ubtmarkpos	ubtmerge	ubtoptions	ubtrescan	ubtrestrpos
ubtvacuumcleanup				

❖ plpgsql 内部函数

plpgsql_inline_handler

❖ 集合相关内部函数

array_indexby_delete	array_indexby_length	array_integer_deleteidx	array_integer_exists	array_integer_first	array_integer_last
array_integer_next	array_integer_prior	array_varchar_deleteidx	array_varchar_exists	array_varchar_first	array_varchar_last
array_varchar_next	array_varchar_prior	-	-	-	-

❖ 外表相关内部函数

dist_fdw_handler	roach_handler	streaming_fdw_handler	dist_fdw_validator	file_fdw_handler	file_fdw_validator	log_fdw_handler

❖ 主数据库节点远程读取备数据库节点数据页辅助函数

gs_read_block_from_remote 用于读取非段页式表文件的页面。默认只有初始化用户可以查看，其余用户需要赋权后才可以使用。

pg_read_binary_file_blocks 用于读取数据，非压缩表返回实际数据，压缩表返回压缩后的数据。默认只有初始化用户可以查看，其余用户需要赋权后才可以使用。

gs_read_segment_block_from_remote 用于读取段页式表文件的页面。默认只有初始化用户可以查看，其余用户需要赋权后才可以使用。

❖ 主数据库节点远程读取备数据库节点数据文件辅助函数

gs_read_file_from_remote 用于读取指定的文件。gs_repair_file 利用 gs_read_file_size_from_remote 函数获取文件大小后，依赖这个函数将远端文件逐段读取。默认只有初始化用户可以查看，其余用户需要赋权后才可以使用。

gs_read_file_size_from_remote 用于读取指定文件的大小。用于读取指定文件的大小，gs_repair_file 函数修复文件时，要先获取远端关于这个文件的大小，用于校验本地文件缺失的文件信息，然后将缺失的文件逐个修复。默认只有初始化用户可以查看，其余用户需要赋权后才可以使用。

❖ 账本数据库函数

get_dn_hist_relhash

❖ AI 特性函数

create_snapshot	create_snapshot_internal	prepare_snapshot_internal	prepare_snapshot	manage_snapshot_internal	archive_snapshot	publish_snapshot
purge_snapshot_internal	purge_snapshot	sample_snapshot				

❖ PKG_SERVICE 函数

isubmit_on_nodes、submit_on_nodes

❖ 其他函数

to_tsvector_for_b	value_of_percent	disable_	bind_var	job_upd	job_canc	job_finis

atch	ile	conn	iable	ate	el	h
similar_escape	table_skewness（不可用）	timetz_text	time_text	reltime_text	abstime_text	_pg_keysequal
analyze_query (不可用)	analyze_workload（不可用）	ssign_table_type	gs_comm_proxy_thread_status	gs_txid_oldestxmin	pg_cancel_session	pg_stat_segment_space_info
remote_segment_space_info	set_cost_params	set_weight_params	start_collect_workload	tdigest_in	tdigest_merge	tdigest_merge_to_one
tdigest_mergep	tdigest_out	pg_get_delta_info	disable_conn	-	-	-

4.1.4.35.Global SysCache 特性函数

❖ gs_gsc_table_detail(database_id default NULL, rel_id default NULL)

描述：查看数据库里全局系统缓存的表元数据。调用该函数的用户需要具有 SYSADMIN 权限。

参数：指定需要查看全局系统缓存的数据库和表，database_id 默认值 NULL 或者-1 表示所有的数据库，0 表示共享表，其他数字表示指定数据库及共享表，rel_id 表示指定表的 oid，默认值 NULL 或者-1 表示所有的表，其他值表示指定的表，database_id 不存在会报错，rel_id 不存在结果为空。

返回值类型：Tuple

示例：

```
select * from gs_gsc_table_detail(-1) limit 1;
database_oid | database_name | reloid |       relname       | relnamespace | reltype |
reloftype | relowner | relam | relfilenode | reltablespace | relcmindex | relisshared | re
lkind | relnatts | relcmoids | relcmpkey | parttype | tdcmuids | attnames  | extinfo
-------------+---------------+--------+-----------------------+-------------+--------
--+----------+----------+-------+------------+--------------+------------+----------
---+---------+----------+-----------+-----------+----------+----------+----------+-
--------
0 |          |  2676 | pg_authid_rolname_index |       11 |     0 |      0 |     10 |  403 |
     0 |      1664 | f       | t       | i     |     1 | f       | f       | n     | f      | 'rolname' |
(1 row)
```

❖ gs_gsc_catalog_detail(database_id default NULL, rel_id default NULL)

描述：查看数据库里全局系统缓存的系统表行信息。调用该函数的用户需要具有 SYSADMIN 权限。

参数：指定需要查看全局系统缓存的数据库和表，database_id 默认值 NULL 或者-1 表示所有的数据库，0 表示共享表，其他数字表示指定数据库及共享表， rel_id 表示指定表的 id，仅包含所有有系统缓存的系统表，默认值 NULL 或者-1 表示所有的表，database_id 不存在会报错，rel_id 不存在结果为空。

返回值类型：Tuple

示例：

```
panweidb=#
select * from gs_gsc_catalog_detail(16574, 1260);
database_id | database_name | rel_id | rel_name  | cache_id |  self  |  ctid  | infomask |
 infomask2 | hash_value | refcount
-------------+---------------+--------+-----------+----------+--------+--------+--------
---+-----------+------------+----------
      0 |          |  1260 | pg_authid |     10 | (0, 9) | (0, 9) |  10507 |     26 |  53131156
8 |     10
        0 |          |  1260 | pg_authid |     11 | (0, 4) | (0, 4) |   2313 |     26 |  365368
336 |      1
        0 |          |  1260 | pg_authid |     11 | (0, 9) | (0, 9) |  10507 |     26 | 391151
7328 |     10
```

```
        0 |          | 1260 | pg_authid |    11 | (0, 7) | (0, 7) |   2313 |     26 | 131779
9983 |     1
        0 |          | 1260 | pg_authid |    11 | (0, 5) | (0, 5) |   2313 |     26 | 366434
7448 |     1
        0 |          | 1260 | pg_authid |    11 | (0, 1) | (0, 1) |   2313 |     26 |  276477
273 |     1
        0 |          | 1260 | pg_authid |    11 | (0, 3) | (0, 3) |   2313 |     26 | 246583
7659 |     1
        0 |          | 1260 | pg_authid |    11 | (0, 8) | (0, 8) |   2313 |     26 | 320528
8035 |     1
        0 |          | 1260 | pg_authid |    11 | (0, 6) | (0, 6) |   2313 |     26 |  131811
687 |     1
        0 |          | 1260 | pg_authid |    11 | (0, 2) | (0, 2) |   2313 |     26 | 122648
4587 |     1
(10 rows)
```

❖ gs_gsc_clean(database_id default NULL)

描述：清理 global syscache 的缓存，需要注意，正在使用中的数据不会被清理。调用该函数的用户需要具有 SYSADMIN 权限。

参数：指定需要清理全局系统缓存的数据库，默认值 NULL 或者-1 表示清理所有的数据库全局系统缓存，0 表示只淘汰共享表的全局系统缓存，其他数字表示淘汰清理指定数据库以及共享表的全局系统缓存，database_id 不存在会报错。

返回值类型：bool

示例：

```
panweidb=# select * from gs_gsc_clean();
gs_gsc_clean
--------------
t
(1 row)
```

❖ gs_gsc_dbstat_info(database_id default NULL)

描述：获取本地节点的 GSC 的内存统计信息，包括 tuple、relation、partition 的缓存查询，命中，加载、失效、占用空间信息，DB 级别的淘汰信息，线程引用信息，内存占用信息。可以用于定位性能问

题，例如当发现 hits/searches 数组远小于 1 时，可能是 global_syscache_threshold 设置太小，导致查询命中率下降。调用该函数的用户需要具有 SYSADMIN 权限。

参数：指定需要查看的数据库全局系统缓存统计信息，NULL 或者-1 表示查看所有的数据库，0 表示只查看共享表信息，其他数字表示查看指定的数据库和共享表的信息。不合法的输入值，database_id 不存在会报错。

返回值类型：Tuple

示例：

```
panweidb=# select * from gs_gsc_dbstat_info();
database_id | database_name | tup_searches | tup_hits | tup_miss | tup_count | tup_
dead | tup_memory | rel_searches | rel_hits | rel_mis
s | rel_count | rel_dead | rel_memory | part_searches | part_hits | part_miss | part_cou
nt | part_dead | part_memory | total_memory | swa
pout_count | refcount
-------------+---------------+-------------+----------+----------+-----------+---------
--+------------+--------------+----------+--------
--+-----------+----------+------------+---------------+-----------+-----------+---------
---+-----------+-------------+--------------+----
-----------+----------
      0 |          |      300 |    235 |     31 |     22 |      2 |    9752 |       598 |    108 |
    1
   8 |     18 |     0 |    77720 |        0 |      0 |      0 |      0 |      0 |       0 |    7529
12 |
       0 |      0
      16574 | postgres    |     3368 |   2289 |    329 |    273 |      0 |   92593 |
1113 |    524 |     4
   8 |     48 |     0 |   340456 |        0 |      0 |      0 |      0 |      0 |       0 |   4124
792 |
       0 |     10
(2 rows)
```

4.1.4.36.数据损坏检测修复函数

修复主备机文件页面的约束概述：

文件类型	文件页面级别	主备机	检测修复
普通行存表(astore,ustore，包括压缩表)，段页式表，不包括索引	文件&页面	主机	手动检测手动修复，不支持 dcf 模式。
undo(不包括 undo meta)	页面	主机	手动检测手动修复 (不包括 analyse verify)，不支持 dcf 模式。
unlogged 表的 init fork 文件	文件	主机	手动检测手动修复，不支持 dcf 模式。
普通行存表(astore.ustore，不包括压缩表)，索引 l(btree, ubtree)，undo(不包括 undo meta，undo 只支持 CRC 校验错误)	页面	备机	回放过程中自动检测自动修复。

备机修复支持备集群的首备和级联备

- gs_verify_data_file(verify_segment bool)

  描述：校验当前实例当前库是否存在文件丢失的情况。校验只包括数据表主文件是否有中间段丢失的情况。默认参数是 false，表示不校验段页式表数据文件。参数设置为 true 时仅校验段页式表文件。默认只有初始化用户、具有 sysadmin 属性的用户以及在运维模式下具有运维管理员属性的用户可以查看，其余用户需要赋权后才可以使用。

  返回的结果：

  - 非段页式表：rel_oid 和 rel_name 是对应文件的表 oid 和表名，miss_file_path 表示丢失文件的相对路径。
  - 段页式表：因所有表存放在相同文件中，所以 rel_oid 和 rel_name 无法显示具体表的信息。对于段页式表，如果第一个文件损坏，不会检查出后面的.1 .2 等文件。例如 3、3.1、3.2 损坏，只能检查出 3 损坏。当段页式文件不足 5 个时，使用函数检测时，未生成的文件也会校验出来，例如只有 1 和 2 文件，校验段页式时，也会检测出 3，4，5 文件。以下示例，第一个是校验非段页式表的示例，第二是校验段页式表的示例。

  参数说明：

➢ verify_segment

指定文件校验的范围。false 校验非段页式表；true 校验段页式表。

取值范围：true 和 false，默认是 false。

返回值类型：record

示例：

校验非段页式表

```
panweidb=# select * from gs_verify_data_file();
node_name      |rel_oid | rel_name   | miss_file_path
-----------------+---------+-------------+-----------------
dn_6001_6002_6003 |  16554 |    test    |base/16552/24745
```

校验段页式表

```
panweidb=# select * from gs_verify_data_file(true);
   node_name    |rel_oid | rel_name | miss_file_path
------------------+---------+----------+----------------
 dn_6001_6002_6003 |     0 | none    |base/16573/2
```

❖ gs_repair_file(tableoid Oid，path text, timeout int)

描述：根据传入的参数修复文件，仅支持有正常主备连接的主 DN 使用。参数依据 gs_verify_data_file 函数返回的 oid 和路径填写。段页式表 tableoid 赋值为 0 到 4,294,967,295 的任意值（内部校验根据文件路径判断是否是段页式表文件，段页式表文件则不使用 tableoid）。修复成功返回值为 true，修复失败会显示具体失败原因。默认只有在主 DN 节点上，使用初始化用户、具有 sysadmin 属性的用户以及在运维模式下具有运维管理员属性的用户可以查看，其余用户需要赋权后才可以使用。

⚠ **注意：**

1. 当 DN 实例上存在文件损坏时，进行升主会校验出错，报 PANIC 退出无法升主，为正常现象。可在其他 DN 升主后，通过备 DN 自动修复进行修复。
2. 当文件存在但是大小为 0 时，此时不会去修复该文件，若想要修复该文件，需要将为 0 的文件删除后再修复。

3. 删除文件需要等文件 fd 自动关闭后再修复，人工操作可以执行重启进程、主备切换命令。

参数说明：

- tableoid

  要修复的文件对应的表 oid，依据 gs_verify_data_file 函数返回的列表中 rel_oid 一列填写。

  取值范围： oid，0 - 4294967295。注意：输入负值等都会被强制转成非负整数类型。

- path

  需要修复的文件路径，依据 gs_verify_data_file 函数返回的列表中 miss_file_path 一列填写。

  取值范围：字符串。

- timeout

  等待备 DN 回放的时长，修复文件需要等待备 DN 回放到当前主 DN 对应的位置，根据备 DN 回放所需时长设定。

  取值范围：60s - 3600s。

返回值类型：bool

示例：

```
panweidb=# select * from gs_repair_file(16554,'base/16552/24745',360);
gs_repair_file
----------------
t
```

- local_bad_block_info()

  描述：显示本实例页面损坏的情况。从磁盘读取页面，发现页面 CRC 校验失败时进行记录。默认只有初始化用户、具有 sysadmin 属性的用户、具有监控管理员属性的用户以及在运维模式下具有运维管理员属性的用户、以及监控用户可以查看，其余用户需要赋权后才可以使用。

  显示信息：file_path 是损坏文件的相对路径，如果是段页式表，则显示的是逻辑信息，不是实际的物理文件信息。block_num 是该文件损坏的

具体页面号，页面号从 0 开始。check_time 表示发现页面损坏的时间。repair_time 表示修复页面的时间。

返回值类型：record

示例：

```
panweidb=# select * from local_bad_block_info();
node_name    |spc_node | db_node | rel_node| bucket_node | fork_num | block_num
|   file_path    | check_time          |  repair_time
-----------------+-------+--------+--------+-------------+----------+-----------+------
-----------+--------------------------+-------------------------------
dn_6001_6002_6003| 1663 |  16552 | 24745 |      -1   |   0   |0      |base/16552/247
45 |2022-01-13 20:19:08.385004+08 |2022-01-13 20:19:08.407314+08
```

❖ local_clear_bad_block_info()

描述：清理 local_bad_block_info 中已修复页面的数据，也就是 repair_time 不为空的信息。默认只有初始化用户、具有 sysadmin 属性的用户以及在运维模式下具有运维管理员属性的用户、以及监控用户可以查看，其余用户需要赋权后才可以使用。

返回值类型：bool

示例：

```
panweidb=# select * from local_clear_bad_block_info();
result
--------
t
```

❖ gs_verify_and_tryrepair_page (path text, blocknum oid, verify_mem bool, is_segment bool)

描述：校验本实例指定页面的情况。默认只有在主 DN 节点上，使用初始化用户、具有 sysadmin 属性的用户以及在运维模式下具有运维管理员属性的用户可以查看，其余用户需要赋权后才可以使用。

返回的结果信息：disk_page_res 表示磁盘上页面的校验结果；mem_page_res 表示内存中页面的校验结果；is_repair 表示在校验的过程中是否触发修复功能，t 表示已修复，f 表示未修复。

注意：当 DN 实例上存在页面损坏时，进行升主会校验出错，报 PANIC 退出无法升主，为正常现象。可在其他 DN 升主后，通过备 DN 自动修复进行修复。

参数说明：

- path

  损坏文件的路径。依据 local_bad_block_info 中 file_path 一列填写。

  取值范围：字符串。

- blocknum

  损坏文件的页号。依据 local_bad_block_info 中 block_num 一列填写。

  取值范围：Oid，0 - 4294967295。注意：输入负值等都会被强制转成非负整数类型。

- verify_mem

  指定是否校验内存中的指定页面。设定为 false 时，只校验磁盘上的页面。设置为 true 时，校验内存中的页面和磁盘上的页面。如果发现磁盘上页面损坏，会将内存中的页面做一个基本信息校验刷盘，修复磁盘上页面。如果校验内存页面时发现页面不在内存中，会经内存接口读取磁盘上的页面。此过程中如果磁盘页面有问题，则会触发远程读自动修复功能。

  取值范围：bool，true 和 false。

- is_segment

  是否是段页式表。根据 local_bad_block_info 中的 bucket_node 列值决定。如果 bucket_node 为-1 时，表示不是段页式表，将 i s_segment 设置为 false；非-1 的情况将 is_segment 设置为 true。

  取值范围：bool，true 和 false。

  返回值类型：record

示例：

```
panweidb=# select * from gs_verify_and_tryrepair_page('base/16552/24745',0,false,false);
```

```
node_name      |     path     | blocknum |      disk_page_res      | mem_page_res | i
s_repair
-----------------+-----------------+-----------+--------------------------+-------
--------+----------
dn_6001_6002_6003 | base/16552/24745 |    0     | page verification succeeded.|
    | f
```

❖ **gs_repair_page(path text, blocknum oid, is_segment bool, timeout int)**

描述：修复本实例指定页面，仅支持有正常主备连接的主 DN 使用。页面修复成功返回 true，修复过程中出错会有报错信息提示。默认只有在主 DN 节点上，使用初始化用户、具有 sysadmin 属性的用户以及在运维模式下具有运维管理员属性的用户可以查看，其余用户需要赋权后才可以使用。

注意：当 DN 实例上存在页面损坏时，进行升主会校验出错，报 PANIC 退出无法升主，为正常现象。可在其他 DN 升主后，通过备 DN 自动修复进行修复。

参数说明：

➢ path

损坏页面的路径。根据 local_bad_block_info 中 file_path 一列设置，或者是 gs_verify_and_tryrepair_page 函数中 path 一列设置。

取值范围：字符串。

➢ blocknum

损坏页面的页面号。根据 local_bad_block_info 中 block_num 一列设置，或者是 gs_verify_and_tryrepair_page 函数中 blocknum 一 列设置。

取值范围：int，0 - 2147483647。注意：输入负值等都会被强制转成非负整数类型。

➢ is_segment

是否是段页式表。根据 local_bad_block_info 中的 bucket_node 列值决定，如果 bucket_node 为-1 时，表示不是段页式表，将 is_segment 设置为 false；非-1 的情况将 is_segment 设置为 true。

取值范围：bool，true 或者 false。

- timeout

等待备 DN 回放的时长。修复页面需要等待备 DN 回放到当前主 DN 对应的位置，根据备 DN 回放所需时长设定。

取值范围：60s - 3600s。

返回值类型：bool

示例：

```
panweidb=# select * from gs_repair_page('base/16552/24745',0,false,60);
result
--------
t
```

4.1.4.37.废弃函数

PanWeiDB 中下列函数在最新版本中已废弃：

gs_wlm_get_session_info	gs_wlm_get_user_session_info	pgxc_get_csn
pgxc_get_stat_dirty_tables	pgxc_get_thread_wait_status	pgxc_gtm_snapshot_status
pgxc_is_committed	pgxc_lock_for_backup	pgxc_lock_for_sp_database
pgxc_lock_for_transfer	pgxc_log_comm_status	pgxc_max_datanode_size
pgxc_node_str	pgxc_pool_check	pgxc_pool_connection_status
pgxc_pool_reload	pgxc_prepared_xact	pgxc_snapshot_status
pgxc_stat_dirty_tables	pgxc_unlock_for_sp_database	pgxc_unlock_for_transfer
pgxc_version	array_extend	prepare_statement_status

remote_rto_stat	dbe_perf.global_slow_query_info	dbe_perf.global_slow_query_info_bytime
dbe_perf.global_slow_query_history	pg_stat_get_pooler_status	pg_stat_get_wlm_node_resource_info
pg_stat_get_wlm_session_info_internal	DBE_PERF.get_wlm_controlgroup_ng_config()	DBE_PERF.get_wlm_user_resource_runtime()
global_space_shrink	pg_pool_validate	gs_stat_ustore
table_skewness(text)	table_skewness(text, text, text)	-

4.1.4.38.XML 类型函数

4.1.4.38.1. 产生 XML 内容

下列函数可以用来从 SQL 数据产生 XML 内容。它们特别适合于将查询结果格式化成 XML 文档以便于在客户端应用中处理。

❖ **xmlparse ({ DOCUMENT | CONTENT } value)**

描述：使用函数 xmlparse, 来从字符数据产生 xml 类型的值。

参数：数据类型为 text

返回值类型：xml

示例：

```
SELECT XMLPARSE (DOCUMENT '<?xml version="1.0"?><book><title>Manual</tit
le><chapter>...</chapter></book>');
SELECT XMLPARSE (CONTENT 'abc<foo>bar</foo><bar>foo</bar>');
```

返回结果依次为：

```
             xmlparse
-----------------------------------------------------------
<book><title>Manual</title><chapter>...</chapter></book>
(1 row)

    xmlparse
--------------------------------
```

```
abc<foo>bar</foo><bar>foo</bar>
(1 row)
```

❖ **xmlserialize({ DOCUMENT | CONTENT } value AS type)**

描述：使用函数 xmlserialize, 来从 xml 产生一个字符串。

参数：类型可以是 character，character varying 或 text 或其中某个的变种。

返回值类型：xml

示例：

```
SELECT XMLSERIALIZE(CONTENT 'good' AS CHAR(10));
SELECT xmlserialize(DOCUMENT '<head>bad</head>' as text);
```

返回结果依次为：

```
 xmlserialize
--------------
good
(1 row)

xmlserialize
------------------
<head>bad</head>
(1 row)
```

【说明】

当一个字符串值在没有通过 XMLPARSE 或 XMLSERIALIZE 的情况下，与 xml 类型进行转换时，分别的选择 DOCUMENT 与 CONTENT 由”XML option” 会话配置参数决定，这个配置参数可以由标准命令来设置：

```
SET XML OPTION { DOCUMENT | CONTENT };
```

或使用类似的语法来设置：

```
SET xmloption TO { DOCUMENT | CONTENT };
```

❖ **xmlcomment(text)**

描述：创建一个 XML 值，并且它包含一个用指定文本作为内容的 XML 注释。该文本不包含“-”字符且不存在是“-”字符的结尾、符合 XML 注释的格式要求。且当参数为空时、结果也为空。

参数：数据类型为 text。

返回值类型：xml

示例：

```
SELECT xmlcomment('hello');
```

返回结果为：

```
 xmlcomment
--------------
<!--hello-->
(1 row)
```

- **xmlconcat(xml[, ...])**

描述：将由单个 XML 值组成的列表串接成一个单独的值，该值包含一个 XML 的内容片断。其中空值会被忽略，并且只有当所有参数都为空时结果才为空。

参数：数据类型为 xml。

返回值类型：xml

示例：

```
SELECT xmlconcat('<abc/>', '<bar>foo</bar>');
```

返回结果为：

```
    xmlconcat
----------------------
<abc/><bar>foo</bar>
(1 row)
```

- **xmlelement(name name [, xmlattributes(value [AS attname] [, ...])] [, content, ...])**

描述：使用给定名称、属性和内容产生一个 XML 元素。

返回值类型：xml

示例：

```
SELECT xmlelement(name foo);
```

返回结果为：

```
 xmlelement
------------
<foo/>
```

```
(1 row)
```

- **xmlforest(content [AS name] [, ...])**

  描述：使用给定名称和内容产生一个元素的 XML 序列。

  返回值类型：xml

  示例：

```
SELECT xmlforest('abc' AS foo, 123 AS bar);
SELECT xmlforest(table_name, column_name) FROM information_schema.colu
mns WHERE table_schema = 'pg_catalog';
```

  返回结果依次为：

```
    xmlforest
------------------------------
<foo>abc</foo><bar>123</bar>
(1 row)

                              xmlforest
--------------------------------------------------------------------------------
-------
<table_name>pg_authid</table_name><column_name>rolname</column_na
me>
<table_name>pg_authid</table_name><column_name>rolsuper</column_na
me>
...
```

- **xmlpi(name target [, content])**

  描述：创建一个 XML 处理指令。如果 content 内容不为空，则 content 内容不能包含字符序列?>。

【说明】

处理指令（Processing Instructions，PI）是用<? ?>包围的一种标签，用以描述特定应用程序信息。Xml 文档可以包含多个针对不同应用程序的处理指令。处理指令由两部分组成，target 和 content。target 的角色类似于“名称”，紧随 target 之后的字符串就是 content，content 可以包含多个标记。

  返回值类型：xml

  示例：

```
SELECT xmlpi(name php, 'echo "hello world";');
```

返回结果为：

```
        xmlpi
----------------------------
<?php echo "hello world";?>
(1 row)
```

❖ **xmlroot(xml, version text | no value [, standalone yes|no|no value])**

描述：修改一个 XML 值的根节点的属性。如果指定了一个版本，它会替换根节点的版本声明中的值、如果指定了一个独立设置、它会替换根节点的独立声明中的值。

返回值类型：xml

示例：

```
SELECT xmlroot('<?xml version="1.1"?><content>abc</content>',version '1.0', s
tandalone yes);
```

返回结果为：

```
                xmlroot
---------------------------------------------------------------
<?xml version="1.0" standalone="yes"?><content>abc</content>
(1 row)
```

❖ **xmlagg(xml)**

描述：该函数是一个聚集函数、它将聚集函数调用的输入值串接起来，且支持跨行串接。

参数：xml

返回值类型：xml

示例：

1、创建测试表并插入数据。

```
CREATE TABLE xmltest (id int,data xml);
INSERT INTO xmltest VALUES (1, '<value>one</value>');
INSERT INTO xmltest VALUES (2, '<value>two</value>');
```

2、调用 xmlagg 函数。

```
SELECT xmlagg(data) FROM xmltest;
```

返回结果为：

```
           xmlagg
-----------------------------------
```

```
<value>one</value><value>two</value>
(1 row)
```

4.1.4.38.2. XML 谓词

下列函数用于检查 xml 值的属性。

- **xmlexists(text passing [BY REF] xml [BY REF])**

  描述：评价一个 XPath 1.0 表达式(第一个参数)，以传递的 XML 值作为其上下文项。 如果评价的结果产生一个空节点集，该函数返回 false，如果产生任何其他值，则返回 true。 如果任何参数为空，则函数返回 null。作为上下文项传递的非空值必须是一个 XML 文档，而不是内容片段或任何非 XML 值。

  参数：xml

  返回值类型：bool

  示例：

```
SELECT xmlexists('//town[text() = ''Toronto'']' PASSING BY REF '<towns><town>
Toronto</town><town>Ottawa</town></towns>');
```

  返回结果为：

```
xmlexists
------------
t
(1 row)
```

- **xml_is_well_formed(text)**

  描述：检查 text 是不是正确的 XML 类型格式、返回值为布尔类型。

  参数：text

  返回值类型：bool

  示例：

```
SELECT xml_is_well_formed('<>');
```

  返回结果为：

```
 xml_is_well_formed
--------------------
f
(1 row)
```

❖ **xml_is_well_formed_document(text)**

描述：检查 text 是不是正确的 XML 类型格式、返回值为布尔类型。

参数：text

返回值类型：bool

示例：

```
SELECT xml_is_well_formed_document('<pg:foo xmlns:pg="http://postgresql.
org/stuff">bar</pg:foo>');
```

返回结果为：

```
xml_is_well_formed_document
----------------------------
t
(1 row)
```

❖ **xml_is_well_formed_content(text)**

描述：检查 text 是不是正确的 XML 类型格式、返回值为布尔类型。

参数：text

返回值类型：bool

示例：

```
select xml_is_well_formed_content('k');
```

返回结果为：

```
xml_is_well_formed_content
---------------------------
t
(1 row)
```

4.1.4.38.3. 处理 XML

PanWeiDB 提供了下列函数用于处理数据类型 xml 的值。

【须知】

xpath 相关函数仅支持 xpath() 和 xpath_exists()、由于其使用 xpath 语言查询 XML 文档，而这些函数都依赖于 libxml2 库，且这个库仅在 Xpath1.0 提供、所以对 XPath 的限制为 1.0。

❖ **xpath(xpath, xml [, nsarray])**

描述：在 XML 值 xml 上计算 XPath 1.0 表达式 xpath (a text value)。它返回一个 XML 值的数组，该数组对应于该 XPath 表达式产生的结点集合。如果该 XPath 表达式返回一个标量值而不是一个结点集合，将会返回一个单一元素的数组。

第二个参数必须是一个良构的 XML 文档。特殊地，它必须有一个单一根节点元素。

该函数可选的第三个参数是一个名字空间映射的数组。这个数组应该是一个二维 text 数组，其第二轴长度等于 2（即它应该是一个数组的数组，其中每一个都由刚好 2 个元素组成）。每个数组项的第一个元素是名字空间的名称（别名），第二个元素是名字空间的 URI。并不要求在这个数组中提供的别名和在 XML 文档本身中使用的那些名字空间相同（换句话说，在 XML 文档中和在 xpath 函数环境中，别名都是本地的）。

返回值类型：xml

示例：

```
SELECT xpath('/my:a/text()', '<my:a xmlns:my="http://example.com">test</my:a>',ARRAY[ARRAY['my', 'http://example.com']]);
```

返回结果为：

```
xpath
--------
{test}
(1 row)
```

- **xpath_exists(xpath, xml [, nsarray])**

描述：该函数是 xpath 函数的一种特殊形式。这个函数不是返回满足 XPath 1.0 表达式的单一 XML 值，它返回一个布尔值表示查询是否被满足(具体来说，它是否产生了空节点集以外的任何值)。这个函数等价于标准的 XMLEXISTS 谓词，不过它还提供了对一个名字空间映射参数的支持。

返回值类型：bool

示例：

```
SELECT xpath_exists('/my:a/text()', '<my:a xmlns:my="http://example.com">test</my:a>',ARRAY[ARRAY['my', 'http://example.com']]);
```

返回结果为：

```
xpath_exists
```

```
--------------
t
(1 row)
```

4.1.4.38.4. 将表映射到 XML

- **query_to_xml(query text, nulls boolean, tableforest boolean, targetns text)**

  描述：该函数将关系表的内容映射成 XML 值，可理解为 XML 的导出功能，由参数 query 传递的查询并且映射结果集。

  返回值类型：xml

  示例：示例中使用的 xmltest 表已创建，参考 xmlagg 示例。

```
select query_to_xml('select * from xmltest',true,true,'mydb');
```

  返回结果为：

```
                  query_to_xml
-----------------------------------------------------------------
<row xmlns:xsi="http://xxxxxx/2001/XMLSchema-instance" xmlns="mydb">+


        +
<id>1</id>                                                      +
<data><value>one</value></data>                                 +
</row>                                                                +
                                                               +
<row xmlns:xsi="http://xxxxxx/2001/XMLSchema-instance" xmlns="mydb">+


 +
<id>2</id>                                                        +
<data><value>two</value></data>                                  +
</row>                                                              +
                                                                +
(1 row)
```

- **query_to_xmlschema(query text, nulls boolean, tableforest boolean, targetns text)**

  描述：返回 XML 模式文档，这些文档描述上述对应函数所执行的映射。

  返回值：XML 模式文档

示例：把查询结果中的行转换成 xml 格式，示例中使用的 xmltest 表已创建，参考 xmlagg 示例。

```
select query_to_xmlschema('select * from xmltest',true,true,'mydb');
```

返回结果为：

```
                    query_to_xmlschema
------------------------------------------------------------------------------
------
<xsd:schema                                                +
  xmlns:xsd="http://xxxxxx/2001/XMLSchema"                          +
  targetNamespace="mydb"                                      +
  elementFormDefault="qualified">                               +
                                                         +
<xsd:simpleType name="INTEGER">                                 +
<xsd:restriction base="xsd:int">                              +
  <xsd:maxInclusive value="2147483647"/>                            +
  <xsd:minInclusive value="-2147483648"/>                           +
</xsd:restriction>                                           +
</xsd:simpleType>                                              +
                                                         +
<xsd:complexType mixed="true">                                  +
<xsd:sequence>                                               +
  <xsd:any name="element" minOccurs="0" maxOccurs="unbounded" process
Contents="skip"/>+
</xsd:sequence>                                              +
</xsd:complexType>                                             +
                                                         +
<xsd:complexType name="RowType">                                +
<xsd:sequence>                                               +
  <xsd:element name="id" type="INTEGER" nillable="true"></xsd:element>
     +
  <xsd:element name="data" type="XML" nillable="true"></xsd:element>
    +
</xsd:sequence>                                              +
</xsd:complexType>                                             +
                                                         +
```

```
<xsd:element name="row" type="RowType"/>                                +
                                                                        +
</xsd:schema>
(1 row)
```

- **query_to_xml_and_xmlschema(query text, nulls boolean, tableforest boolean, targetns text)**

  描述：产生 XML 数据映射和对应的 XML 模式，并把产生的结果链接在一起放在一个文档中。

  示例：把查询结果中行的定义和值转成 xml 格式。示例中使用的 xmltest 表已创建，参考 xmlagg 示例。

```
select query_to_xml_and_xmlschema('select * from xmltest',true,true,'mydb');
```

  返回结果为：

```
                  query_to_xml_and_xmlschema
------------------------------------------------------------------------------
------
<xsd:schema                                                 +
  xmlns:xsd="http://xxxxxx/2001/XMLSchema"                          +
  targetNamespace="mydb"                                      +
  elementFormDefault="qualified">                               +
                                                              +
<xsd:simpleType name="INTEGER">                                 +
<xsd:restriction base="xsd:int">                              +
  <xsd:maxInclusive value="2147483647"/>                          +
  <xsd:minInclusive value="-2147483648"/>                         +
</xsd:restriction>                                            +
</xsd:simpleType>                                               +
                                                              +
<xsd:complexType mixed="true">                                  +
<xsd:sequence>                                                +
  <xsd:any name="element" minOccurs="0" maxOccurs="unbounded" process
Contents="skip"/>+
</xsd:sequence>                                               +
</xsd:complexType>                                              +
                                                              +
```

```
<xsd:complexType name="RowType">                                    +
<xsd:sequence>                                                      +
  <xsd:element name="id" type="INTEGER" nillable="true"></xsd:element>
      +
  <xsd:element name="data" type="XML" nillable="true"></xsd:element>
      +
</xsd:sequence>                                                     +
</xsd:complexType>                                                  +
                                                                    +
<xsd:element name="row" type="RowType"/>                                    +
                                                                    +
</xsd:schema>                                                       +
                                                                    +
<row xmlns:xsi="http://xxxxxx/2001/XMLSchema-instance" xmlns="mydb">
          +
                                                                    +
<id>1</id>                                                          +
<data><value>one</value></data>                                             +
</row>                                                              +
                                                                    +
<row xmlns:xsi="http://xxxxxx/2001/XMLSchema-instance" xmlns="mydb">
          +
                                                                    +
<id>2</id>                                                          +
<data><value>two</value></data>                                             +
</row>                                                              +
                                                                    +
(1 row)
```

- **cursor_to_xml(cursor refcursor, count int, nulls boolean,tableforest boolean, targetns text)**

  描述：该函数将关系表的内容映射成 XML 值，可理解为 XML 的导出功能，从 cursor 指定的游标中取出指定数量的行。

  返回值类型：xml

  示例：

1、创建测试表并插入数据。

```
CREATE TABLE xmltbl (a int,b text);
INSERT INTO xmltbl VALUES (1,'one'),(2,'two'),(-1, null);
```

2、声明一个游标。

```
DECLARE XC CURSOR WITH HOLD FOR SELECT * FROM xmltbl ORDER BY 1,2;
```

3、调用 cursor_to_xml 函数。

```
SELECT cursor_to_xml('xc'::refcursor, 5, false, true,'');
```

返回结果为：

```
            cursor_to_xml
-----------------------------------------------------------
<row xmlns:xsi="http://xxxxxx/2001/XMLSchema-instance">+
                                +
<a>-1</a>                        +
</row>                          +
                                +
<row xmlns:xsi="http://xxxxxx/2001/XMLSchema-instance">+
                                +
<a>1</a>                        +
<b>one</b>                       +
</row>                          +
                                +
<row xmlns:xsi="http://xxxxxx/2001/XMLSchema-instance">+
                                +
<a>2</a>                        +
<b>two</b>                       +
</row>                          +
                                +
(1 row)
```

❖ **cursor_to_xmlschema(cursor refcursor, nulls boolean, tableforest boolean, targetns text)**

描述：返回 XML 模式文档，这些文档描述上述对应函数所执行的映射。

返回值类型：xml

示例：示例中使用的游标 xc 已创建，参考 cursor_to_xml 示例。

```
SELECT cursor_to_xmlschema('xc'::refcursor,  false, true,'');
```

返回结果为：

```
                        cursor_to_xmlschema
------------------------------------------------------------------------------
------
--
<xsd:schema
+
  xmlns:xsd="http://xxxxxx/2001/XMLSchema">
+

+
<xsd:simpleType name="INTEGER">
+
<xsd:restriction base="xsd:int">
+
  <xsd:maxInclusive value="2147483647"/>
+
  <xsd:minInclusive value="-2147483648"/>
+
</xsd:restriction>
+
</xsd:simpleType>
+

+
<xsd:simpleType name="UDT.panweidb.pg_catalog.text">
+
<xsd:restriction base="xsd:string">
+
</xsd:restriction>
+
</xsd:simpleType>
+

+
```

```
<xsd:complexType name="RowType">
+
<xsd:sequence>
+
  <xsd:element name="a" type="INTEGER" minOccurs="0"></xsd:element>
+
  <xsd:element name="b" type="UDT.panweidb.pg_catalog.text" minOccurs="
0"></xsd:element
>+
</xsd:sequence>
+
</xsd:complexType>
+


+
<xsd:element name="row" type="RowType"/>
+


+
</xsd:schema>
(1 row)
```

- **schema_to_xml(schema name, nulls boolean, tableforest boolean, targetns text)**

  描述：把模式中的表映射成 XML 值。

  返回值类型：xml

  示例：

```
SELECT schema_to_xml('public', true, true,'');
```

- **schema_to_xmlschema(schema name, nulls boolean, tableforest boolean, targetns text)**

  描述：把模式中的表映射成 XML 模式文档。

  返回值类型：xml

  示例：

```
select schema_to_xmlschema('public', true, true,'');
```

- **schema_to_xml_and_xmlschema(schema name, nulls boolean, tableforest boolean, targetns text)**

  描述：把模式中的表映射成 XML 值和模式文档。

  返回值类型：xml

  示例：

```
select schema_to_xml_and_xmlschema('public', true, true,'');
```

- **database_to_xml(nulls boolean, tableforest boolean, targetns text)**

  描述：把数据库的表映射成 XML 值。

  返回值类型：xml

- **database_to_xmlschema(nulls boolean, tableforest boolean, targetns text)**

  描述：把数据库的表映射成 XML 模式文档。

  返回值类型：xml

- **database_to_xml_and_xmlschema(nulls boolean, tableforest boolean, targetns text)**

  描述：把数据库的表映射成 XML 值和模式文档。

  返回值类型：xml

- **table_to_xml(tbl regclass, nulls boolean, tableforest boolean, targetns text)**

  描述：该函数将关系表的内容映射成 XML 值，可理解为 XML 的导出功能，由参数 tbl 传递的命名表的内容，regclass 类型接受使用常见标记标识表的字符串。

  返回值类型：xml

  示例：把表的内容转换成 xml 格式。示例中使用的 xmltest 表已创建，参考 xmlagg 示例。

```
select table_to_xml('xmltest'::regclass,true,true,'mydb');
```

  返回结果为：

```
                  table_to_xml
-------------------------------------------------------------------------------
<_x0078_mltest xmlns:xsi="http://xxxxxx/2001/XMLSchema-instance" xmlns=
"mydb">+
```

```
                                        +
<id>1</id>                                  +
<data><value>one</value></data>                     +
</_x0078_mltest>                                +
                                        +
<_x0078_mltest xmlns:xsi="http://xxxxxx/2001/XMLSchema-instance" xmlns=
"mydb">+
                                        +
<id>2</id>                                  +
<data><value>two</value></data>                     +
</_x0078_mltest>                                +
                                        +
(1 row)
```

- **table_to_xmlschema(tbl regclass, nulls boolean, tableforest boolean, targetns text)**

  描述：返回 XML 模式文档，这些文档描述上述对应函数所执行的映射。

  返回值类型：xml

  示例：把表的定义转换成 xml 格式。示例中使用的 xmltest 表已创建，参考 xmlagg 示例。

```
select table_to_xmlschema('xmltest'::regclass,true,true,'mydb');
```

返回结果为：

```
                    table_to_xmlschema
------------------------------------------------------------------------------
------
<xsd:schema                                         +
  xmlns:xsd="http://xxxxxx/2001/XMLSchema"                  +
  targetNamespace="mydb"                                +
  elementFormDefault="qualified">                           +
                                                +
<xsd:simpleType name="INTEGER">                             +
<xsd:restriction base="xsd:int">                            +
  <xsd:maxInclusive value="2147483647"/>                        +
  <xsd:minInclusive value="-2147483648"/>                       +
</xsd:restriction>                                  +
```

```
</xsd:simpleType>                                                +
                                                                 +
<xsd:complexType mixed="true">                                   +
<xsd:sequence>                                                   +
  <xsd:any name="element" minOccurs="0" maxOccurs="unbounded" process
Contents="skip"/>+
</xsd:sequence>                                                  +
</xsd:complexType>                                               +
                                                                 +
<xsd:complexType name="RowType.panweidb.public._x0078_mltest">
       +
<xsd:sequence>                                                   +
  <xsd:element name="id" type="INTEGER" nillable="true"></xsd:element>
      +
  <xsd:element name="data" type="XML" nillable="true"></xsd:element>
     +
</xsd:sequence>                                                  +
</xsd:complexType>                                               +
                                                                 +
<xsd:element name="_x0078_mltest" type="RowType.panweidb.public._x0078
_mltest"/>     +
                                                                 +
</xsd:schema>
(1 row)
```

- **table_to_xml_and_xmlschema(tbl regclass, nulls boolean, tableforest boolean, targetns text)**

  描述：产生 XML 数据映射和对应的 XML 模式，并把产生的结果链接在一起放在一个文档中。

  示例：把表的定义和表中的数据转换成 xml 格式。示例中使用的 xmltest 表已创建，参考 xmlagg 示例。

```
select table_to_xml_and_xmlschema('xmltest'::regclass,true,true,'mydb');
```

  返回结果为：

```
                  table_to_xml_and_xmlschema
-------------------------------------------------------------------------------
```

```
------
<xsd:schema                                                +
  xmlns:xsd="http://xxxxxx/2001/XMLSchema"                          +
  targetNamespace="mydb"                                    +
  elementFormDefault="qualified">                             +
                                                           +
<xsd:simpleType name="INTEGER">                               +
<xsd:restriction base="xsd:int">                             +
  <xsd:maxInclusive value="2147483647"/>                        +
  <xsd:minInclusive value="-2147483648"/>                       +
</xsd:restriction>                                          +
</xsd:simpleType>                                             +
                                                           +
<xsd:complexType mixed="true">                                +
<xsd:sequence>                                              +
  <xsd:any name="element" minOccurs="0" maxOccurs="unbounded" process
Contents="skip"/>+
</xsd:sequence>                                             +
</xsd:complexType>                                            +
                                                           +
<xsd:complexType name="RowType.panweidb.public._x0078_mltest">
       +
<xsd:sequence>                                              +
  <xsd:element name="id" type="INTEGER" nillable="true"></xsd:element>
      +
  <xsd:element name="data" type="XML" nillable="true"></xsd:element>
     +
</xsd:sequence>                                             +
</xsd:complexType>                                            +
                                                           +
<xsd:element name="_x0078_mltest" type="RowType.panweidb.public._x0078
_mltest"/>     +
                                                           +
</xsd:schema>                                                 +
                                                           +
```

```
<_x0078_mltest xmlns:xsi="http://xxxxxx/2001/XMLSchema-instance" xmlns=
"mydb">       +
                                                +
<id>1</id>                                          +
<data><value>one</value></data>                                 +
</_x0078_mltest>                                        +
                                                +
<_x0078_mltest xmlns:xsi="http://xxxxxx/2001/XMLSchema-instance" xmlns=
"mydb">       +
                                                +
<id>2</id>                                          +
<data><value>two</value></data>                                 +
</_x0078_mltest>                                        +
                                                +
(1 row)
```

4.1.4.39.HashFunc 函数

❖ **hash_array(anyarray)**

描述：数组哈希，将数组的元素通过哈希函数得到结果，并返回合并结果。

参数：数据类型为 anyarray。

返回值类型：integer

示例：

```
panweidb=# select hash_array(ARRAY[[1,2,3],[1,2,3]]);
 hash_array
------------
 -382888479
(1 row)
```

❖ **hash_numeric(numeric)**

描述：计算 Numeric 类型的数据的 hash 值。

参数：Numeric 类型的数据。

返回值类型：integer

示例：

```
panweidb=# select hash_numeric(30);
```

```
hash_numeric
--------------
-282860963
(1 row)
```

❖ **hash_range(anyrange)**

描述：计算 range 的哈希值。

参数：anyrange 类型的数据。

返回值类型：integer

示例：

```
panweidb=# select hash_range(numrange(1.1,2.2));
 hash_range
------------
 683508754
(1 row)
```

❖ **hashbpchar(character)**

描述：计算 bpchar 的哈希值。

参数：character 类型的数据。

返回值类型：integer

示例：

```
panweidb=# select hashbpchar('hello');
 hashbpchar
-------------
 -1870292951
(1 row)
```

❖ **hashchar(char)**

描述：char 和布尔数据转换为哈希值。

参数：char 类型的数据或者 bool 类型的数据。

返回值类型：integer

示例：

```
panweidb=# select hashbpchar('hello');
 hashbpchar
-------------
```

```
-1870292951
(1 row)

panweidb=# select hashchar('true');
 hashchar
------------
 1686226652
(1 row)
```

- **hashenum(anyenum)**

  描述：枚举类型转哈希值。

  参数：anyenum 类型的数据。

  返回值类型：integer

  示例：

```
panweidb=# CREATE TYPE b1 AS ENUM('good', 'bad', 'ugly');
CREATE TYPE
panweidb=# call hashenum('good'::b1);
 hashenum
------------
 1821213359
(1 row)
```

- **hashfloat4(real)**

  描述：float4 转哈希值。

  参数：real 类型的数据。

  返回值类型：integer

  示例：

```
panweidb=# select hashfloat4(12.1234);
 hashfloat4
------------
 1398514061
(1 row)
```

- **hashfloat8(double precision)**

  描述：float8 转哈希值。

参数：double precision 类型的数据。

返回值类型：integer

示例：

```
panweidb=# select hashfloat8(123456.1234);
 hashfloat8
------------
 1673665593
(1 row)
```

- **hashinet(inet)**

描述：支持 inet / cidr 上的哈希索引的功能。返回传入 inet 的 hash 值。

参数：int 类型的数据。

返回值类型：integer

示例：

```
panweidb=# select hashinet('127.0.0.1'::inet);
  hashinet
-------------
 -1435793109
(1 row)
```

- **hashint1(tinyint)**

描述：INT1 转哈希值。

参数：tinyint 类型的数据。

返回值类型：uint32

示例：

```
panweidb=# select hashint1(20);
  hashint1
-------------
 -2014641093
(1 row)
```

- **hashint2(smallint)**

描述：INT2 转哈希值。

参数：smallint 类型的数据。

返回值类型：uint32

示例：

```
panweidb=# select hashint2(20000);
 hashint2
------------
 -863179081
(1 row)
```

4.1.4.40.资源池化函数

❖ **ss_buffer_ctrl()**

描述：显示 buffer 对应的 buf_ctrl 上记录的信息。

返回值类型：record

返回的字段如下：

- bufferid：buffer 在数组的下标。
- is_remote_dirty：是否有远程脏页，即除本节点，其他节点是否持有这个页面的早期脏页版本(edp 页面)。0 代表没有远程脏页，1 代表有脏页版本。
- lock_mode：本地对页面持有的分布式锁模式。0 代表没有持有锁模式，1 代表以 S 锁的方式持有页面，2 代表以 X 锁的方式持有页面。
- is_edp：是否是 edp 页面（Early Dirty Page），当请求者 X 锁请求页面，本节点作为 owner 转移页面，且该页面还是脏页，等页面转移后，本地页面就是 edp 页面。0 代表不是 edp 页面，1 代表是 edp 页面。
- force_request：是否是是强制请求，0 代表不是，1 代表是。
- need_flush：reform 的 flush_copy 阶段，标记需要刷盘的页面。0 代表不需要刷盘，1 代表需要刷盘。
- buf_id：同 bufferid，buffer 在数组的下标。
- state：用于记录状态标签。
- pblk_relno：物理块的 relation 编号。
- pblk_blkno：物理块储的 block 编号。

- pblk_lsn：物理块的 lsn。
- seg_fileno：段页式存储的文件编号。
- seg_blockno：段页式存储的 block 编号。

示例 1：查询特定的 buffer 的 buf_ctrl 信息。

```
select * from ss_buffer_ctrl() where bufferid = 20;
```

返回结果如下：

```
bufferid | is_remote_dirty | lock_mode | is_edp | force_request | need_flush | buf
_id | state | pblk_relno | pblk_blkno | pblk_lsn | seg_fileno | seg_blockno
----------+-----------------+-----------+--------+---------------+------------+----
----+-------+------------+------------+----------+------------+-------------
20 | 0 | 1 | 0 | 0 | 0 | 19 | 32 | 0 | 4294967295 | 0 | 2 | 4375
(1 row)
```

示例 2：查询 DMS 页面分布式锁不为 NULL 的 buffer 数量。

```
select count(*) from ss_buffer_ctrl() where lock_mode > 0;
```

返回结果如下：

```
count
-------
258
(1 row)
```

❖ **ss_txnstatus_cache_stat()**

描述：返回事务信息缓存和事务信息获取的相关统计信息。注意相关 count 统计项并不是精确值，本统计函数的目的是性能调优分析，而非精确统计。

返回值类型：record

- vcache_gets：事务信息从变量缓存获取的次数。
- hcache_gets：事务信息从哈希缓存获取的次数。
- nio_gets：事务信息从网络 I/O 获取的次数。
- avg_hcache_gettime_us：从哈希缓存获取的平均耗时，单位 us。
- avg_nio_gettime_us：从网络 I/O 获取的平均耗时，单位 us。
- cache_hit_rate：缓存命中率。
- hcache_eviction：缓存淘汰计数。

- avg_eviction_refcnt：每个缓存条目被淘汰前的平均被引用次数。

示例：主备 TPCC 写/读，备机查询事务信息缓存统计项。

```
select * from ss_txnstatus_cache_stat();
```

返回结果如下：

```
vcache_gets | hcache_gets | nio_gets | avg_hcache_gettime_us | avg_nio_getti
me_us | cache_hit_rate | hcache_eviction | avg_eviction_refcnt
------------+-------------+----------+-----------------------+-------------------
+----------------+----------------+--------------------
263809 | 782159 | 275012 | 1.73883698838727 | 756.186410047562 | .791812139
47221 | 0 | 0
(1 row)
```

❖ **query_node_reform_info()**

描述：查询集群 reform 相关统计信息。

返回值类型：record

- reform_node_id：数据库节点编号，取值范围：[0, 63]
- reform_type：集群发生 reform 的类型，取值范围：[Normal reform、Failover、Switchover]
- reform_start_time：reform 开始的时间
- reform_end_time：reform 结束的时间
- is_reform_success：reform 是否成功
- redo_start_time：日志回放开始的时间
- redo_end_time：日志回放结束的时间
- xlog_total_bytes：总共回放的日志量
- hashmap_construct_time：hashmap 构造的时间
- action：节点的动作，取值范围：[kick off、join in、stable]

约束：该函数只能在数据库主节点进行查询；查询结果中备节点的信息仅支持查看 reform_node_id 和 action，其他信息无效

示例 1：备节点退出集群。

```
select * from query_node_reform_info();
```

返回结果如下：

```
reform_node_id | reform_type |   reform_start_time    |    reform_end_time     |
```

```
 is_reform_success | redo_start_time | redo_end_time | xlog_total_bytes | hash
map_construct_time |  action
---------------+--------------+-------------------------+----------------------
---+------------------+----------------+--------------+-----------------+------
------------------+----------
         0 | Normal reform | 2023-09-21 16:37:06.520  | 2023-09-21 16:37:06.568
| t             | -           | -          |           -1 | -                | stable
         1 | -          | -                 | -                 | t             | -           | -          |
          -1 | -                | kick off
(2 rows)
```

示例 2：备节点加入集群

```
select * from query_node_reform_info();
```

返回结果如下：

```
reform_node_id |  reform_type |    reform_start_time    |    reform_end_time    |
 is_reform_success | redo_start_time | redo_end_time | xlog_total_bytes | hash
map_construct_time | action
---------------+--------------+-------------------------+----------------------
---+------------------+----------------+--------------+-----------------+------
------------------+---------
           0 | Normal reform | 2023-09-21 16:37:46.414  | 2023-09-21 16:37:47.8
17  | t             | -           | -          |           -1 | -                | stable
           1 | -          | -                 | -                 | t             | -           | -
|           -1 | -                | join in
(2 rows)
```

示例 3：主节点故障，集群 failover

```
select * from query_node_reform_info();
```

返回结果如下：

```
reform_node_id | reform_type |    reform_start_time    |    reform_end_time    | i
s_reform_success |    redo_start_time    |    redo_end_time    | xlog_total_byt
es |  hashmap_construct_time  |  action
---------------+-------------+-------------------------+----------------------
--+------------------+-------------------------+-------------------------+-------
-----------+-------------------------+----------
           0 | -          | -                 | -                 | t             | -                | -
```

```
          |          -1 | -                | kick off
         1 | Failover   | 2023-09-21 16:56:45.893 | 2023-09-21 16:56:50.702 | t
        | 2023-09-21 16:56:50.282 | 2023-09-21 16:56:50.385 |          288 | 2
023-09-21 16:56:50.385 | stable
(2 rows)
```

示例 4：集群主动进行 switchover

```
select * from query_node_reform_info();
```

返回结果如下：

```
reform_node_id | reform_type |   reform_start_time   |   reform_end_time    | i
s_reform_success | redo_start_time | redo_end_time | xlog_total_bytes | hashm
ap_construct_time | action
---------------+-------------+-----------------------+----------------------
-+------------------+-----------------+---------------+------------------+--------
---------------+--------
         0 | Switchover | 2023-09-21 16:59:25.646 | 2023-09-21 16:59:29.182 |
t           | -          | -         |          -1 | -              | stable
         1 | -       | -              | -              | t           | -         |
        -1 | -             | stable
(2 rows)
```

- **query_page_distribution_info(relname TEXT, forkNum INT4, blockNum INT4)**

描述：根据`relname + forkNum + blockNum`查询页面在集群中的分布信息。

入参说明：

- relname：需要查询的表名
- forkNum：需要查询的文件名，取值范围为[0，3]的 int 值，其中数据文件为 0，FSM 文件为 1，visibility map 文件为 2，BCM 文件为 3
- blockNum：需要查询的数据页号，取值范围为[0, total_bolcks)的 int 值，其中 total_blocks 由表中相应文件的总页数决定

返回值类型：record

- instance_id：数据库节点 id，取值范围：[0, 63]
- is_master：是否是页面 master

- is_owner：是否是页面 owner
- is_copy：是否持有副本
- lock_mode：节点持有的锁模式
- mem_lsn：页面在内存中的 lsn
- disk_lsn：页面在磁盘上的 lsn
- is_dirty：内存中的页面是否是脏页

约束：该函数只能在数据库主节点进行查询

示例：向表中插入一行数据，然后立即查询数据页面的分布信息

```
create table tb(id int);
insert into tb values(0); select * from query_page_distribution_info('tb', 0, 0);
```

查询效果如下：

```
 instance_id | is_master | is_owner | is_copy |   lock_mode    | mem_lsn   | disk_ls
n  | is_dirty
-------------+-----------+----------+---------+----------------+-----------+------
------+----------
           0 | t         | t        | f       | Exclusive lock | 1975555208 | 1975555112 | t
(1 row)
```

❖ **dss_io_stat(duration INT4)**

描述：统计在给定时间间隔内 IO 读写速度和次数，通过`duration`指定时间间隔，单位秒。

返回值类型：record

❖ read_kilobyte_per_sec：给定时间内 DSS 读取数据的速度，单位 KB/s

❖ write_kilobyte_per_sec：给定时间内 DSS 写入数据的速度，单位 KB/s

❖ io_times：给定时间间隔内 DSS IO 的调用次数

约束：duration < 60

示例：主节点向表中写入数据，查询 2 秒内 DSS IO 的统计信息

```
select * from dss_io_stat(2);
```

返回结果如下：

```
 read_kilobyte_per_sec | write_kilobyte_per_sec | io_times
-----------------------+------------------------+----------
                   404 |                  25664 |     3158
```

```
(1 row)
```

❖ **query_node_reform_info_from_dms()**

描述：查询集群 reform 相关信息。

返回值类型：record

- NAME：TEXT
- DESCRIPTION：TEXT

其中 NAME 总共有 27 种：

(1) INSTANCE_ID：实例 ID

(2) DMS_ROLE：实例角色 reformer/partener

(3) PROC_STATUS：线程状态 runnung/finish

(4) REFORM_STATUS：执行进度 waiting/running/finished

(5) START_TIME：开始时间戳

(6) REFORM_TYPE：Reform 类型

(7) FULL_CLEAN：full clean 标志 true/false

(8) PROMOTE_ID：升主实例 ID 仅 switchover 类型

(9) DEMOTE_ID：降备实例 ID 仅 switchover 类型

(10) LAST_STEP：上一个步骤

(11) CURR_STEP：当前步骤

(12) NEXT_STEP：下一个步骤

(13) CURR_STEP_ELASPED：当前步骤耗时

(14) DDL_ENABLE：是否允许 DDL

(15) FILE_ENABLE：是否允许文件操作

(16) BITMAP_MES：建立 mes 的实例 bitmap

(17) BITMAP_CONNECT：允许广播的实例 bitmap

(18) BITMAP_STABLE：稳定节点 bitmap

(19) BITMAP_ONLINE：在线节点 bitmap

(20) BITMAP_RECONNECT：需要重新建联的实例 bitmap

(21) BITMAP_DISCONNECT：需要断连的实例 bitmap

(22) BITMAP_CLEAN：需要清理信息的实例 bitmap

(23) BITMAP_RECOVERY：需要日志回放的 bitmap

(24) BITMAP_IN：正常启动状态的实例 bitmap

(25) BITMAP_REBUILD：需要 rebuild 的 bitmap

(26) BITMAP_WITHDRAW：需要接管回事务信息的实例 bitmap

(27) BITMAP_ROLLBACK：需要被 reformer 接管事务信息的实例 bitmap

示例：查询 0 节点的 reform 信息

```
select * from query_node_reform_info_from_dms(0);
```

返回结果为：

```
  NAME      |     DESCRIPTION
------------------+------------------------
INSTANCE_ID     | 0
DMS_ROLE        | REFORMER
REFORM_STATUS    | FINISHED
START_TIME      | 2024-01-02 20:29:04.376
REFORM_TYPE     | OPENGAUSS_NORMAL
FULL_CLEAN      | FALSE
LAST_STEP       | PAGE_ACCESS
CURR_STEP       | DONE_CHECK
NEXT_STEP       | N/A
DDL_ENABLE      | TRUE
FILE_ENABLE     | TRUE
BITMAP_MES      | 3
BITMAP_CONNECT   | 1
BITMAP_STABLE    | 0
BITMAP_ONLINE    | 1
BITMAP_RECONNECT | 1
BITMAP_DISCONNECT| 0
BITMAP_CLEAN     | 0
BITMAP_RECOVERY  | 1
BITMAP_IN       | 0
BITMAP_REBUILD   | 0
```

```
BITMAP_WITHDRAW  | 0
BITMAP_ROLLBACK  | 0
(23 rows)
```

❖ **query_all_drc_info()**

描述：查询所有的 drc 相关信息。

返回值类型：record

- resource_id：资源标识符，对于页面，格式为 fileid-pageid；对于锁，格式为 type/uid/oid/index/part
- master_id：master 的节点 id
- copy_insts：副本持有节点 bitmap
- claimed_owner：owner 持有节点 id
- lock_mode：持有锁模式 0：NULL 1：SHARED 2：EXCLUSIVE
- last_edp：最近的早期脏页持有节点 id
- type：资源类型 1：page 2：lock
- in_recovery：是否处于 recovery 阶段
- copy_promote：reform 期间 DRC 是否从副本升级为 owner
- part_id：所属的 DRC 分区链表 id
- edp_map：EDP 持有节点 bitmap
- lsn：集群中本页面所有 EDP 中最大的 LSN
- len：DRC 缓存的 data 大小
- recovery_skip：reform 期间 DRC 是否跳过回放
- recycling：DRC 资源是否正在被回收
- CONVERTING_INST_ID：converting 节点 ID
- CONVERTING_CURR_MODE：converting 节点当前持有的锁模式： 0：NULL 1：SHARED 2：EXCLUSIVE
- CONVERTING_REQ_MODE：converting 节点申请的锁模式： 0：NULL 1：SHARED 2：EXCLUSIVE

约束：当有且仅有主节点启动后，该函数查询 drc 锁信息为空。

示例：查询 0 节点的 reform 信息

```
select * from query_all_drc_info(0) where RESOURCE_ID = 8/0/15199/1262/0;
```

返回结果为：

```
 RESOURCE_ID   | MASTER_ID | COPY_INSTS | CLAIMED_OWNER | LOCK_MODE | L
AST_EDP | TYPE | IN_RECOVERY | COPY_PROMOTE | PART_ID | EDP_MAP | LSN | LEN
 | RECOVERY_SKIP | RECYCLING | CONVERTING_INST_ID | CONVERTING_CURR_MO
DE | CONVERTING_REQ_MODE
------------------+-----------+------------+--------------+-----------+----------
+------+------------+-------------+---------+---------+-----+-----+------------
--+-----------+------------------+--------------------+--------------------
8/0/15199/1262/0 |      0|       0|         0|      2|   255|  2|       0|
0|   123|     0| 0| 20|        0|      0|         255|           0|          0
(1 row)
```

❖ **ondemand_recovery_status()**

描述：查询当前资源池化节点按需回放状态。

返回值类型：record

- primary_checkpoint_redo_lsn：当前节点读取到的主机 redo 点位置。未开启实时构建特性时，本字段值为'0/0'。
- realtime_build_replayed_lsn：当前节点完成构建的主机日志位置。未开启实时构建特性时，本字段值为'0/0'。
- hashmap_used_blocks：当前 hashmap 中已存入的 block-record 数量。
- hashmap_total_blocks：hashmap 中能够存入的 block-record 总数。
- trxn_queue_blocks：当前已存入的事务日志数量，未开启实时构建特性时，本字段为 0。
- seg_queue_blocks：当前已存入的段页式元数据日志数量，未开启实时构建特性时，本字段为 0。
- in_ondemand_recovery：当前节点是否处于按需回放中，取值范围：[t、f]。当节点正常运行时，无论是否开启实时构建特性，本字段值为'f'。
- ondemand_recovery_status：集群按需回放状态，取值范围及对应含义如下：

ondemand_recovery_status	集群按需回放状态
ONDEMAND_RECOVERY_BUILD	集群处于按需回放中，集群对外提供服

	务，且后台进行日志回放
ONDEMAND_RECOVERY_REDO	集群处于正常运行中，未进行按需回放
NORMAL	集群处于正常运行中，未进行按需回放

- realtime_build_status：当前节点实时构建状态，取值范围及对应含义如下：

realtime_build_status	当前节点实时构建状态
DISABLED	当前节点未处于实时构建状态
BUILD_NORMAL	当前节点处于实时构建状态，且集群状态正常
READY_TO_BUILD	当前节点正在启动实时构建
BUILD_TO_DISABLED	当前节点处于实时构建状态，且即将关闭实时构建，一般发生于 switchover、normal reform 和 failover 但是未升主的节点中
BUILD_TO_REDO	当前节点处于实时构建状态，且即将升主，一般发生于 failover 的升主节点中

- recovery_pause_status：当前节点实时构建暂停状态，取值范围及对应含义如下：

recovery_pause_status	当前节点实时构建暂停状态
NOT PAUSE	当前节点实时构建未暂停
PAUSE(for sync record)	当前节点实时构建暂停，暂停原因为构建到了 DDL 日志
PAUSE(for hashmap full)	当前节点实时构建暂停，暂停原因为 HashMap 空间满

PAUSE(for trxn queue full)	当前节点实时构建暂停，暂停原因为事务日志队列满
PAUSE(for seg queue full)	当前节点实时构建暂停，暂停原因为段页式元数据日志队列满

【说明】

- 当 recovery_pause_status 显示当前节点处于实时构建暂停状态时，会严重影响回放性能，无法保证 RTO 要求，应当考虑增大 ss_ondemand_recovery_mem_size，同时提高主机刷盘及 checkpoint 频率。
- 仅在开启了按需回放特性情况下，才允许查询该视图。

4.1.5.表达式

本章主要介绍 SQL 表达式。

SQL 表达式是构成 SQL 语句的全部或一部分的字符串。 表达式是计算值的一个或多个值，运算符和 SQL 函数的组合，它们计算结果为确定的值。 这些 SQL 表达式就像公式，用户可以使用它们在数据库中查询特定的数据集。

4.1.5.1.简单表达式

逻辑表达式

逻辑表达式的操作符和运算规则，请参见：开发者指南->SQL 语法参考->SQL 基本要素->函数和操作符->逻辑操作符。

比较表达式

常用的比较操作符，请参见：开发者指南->SQL 语法参考->SQL 基本要素->函数和操作符->比较操作符。

除比较操作符外，还可以使用以下句式结构：

- BETWEEN 操作符

  a BETWEEN x AND y 等效于 a >= x AND a <= y

  a NOT BETWEEN x AND y 等效于 a < x OR a > y

- 检查一个值是不是 null，可使用：

  expression IS NULL

  expression IS NOT NULL

  或者与之等价的句式结构，但不是标准的：

  expression ISNULL

  expression NOTNULL

  ⚠ **须知：**

  不要写 expression=NULL 或 expression<>(!=)NULL，因为 NULL 代表一个未知的值，不能通过该表达式判断两个未知值是否相等。

- is distinct from/is not distinct from
  - is distinct from
    - A 和 B 的数据类型、值不完全相同时为 true。
    - A 和 B 的数据类型、值完全相同时为 false。
    - 将空值视为相同。
  - is not distinct from
    - A 和 B 的数据类型、值不完全相同时为 false。
    - A 和 B 的数据类型、值完全相同时为 true。
    - 将空值视为相同。

伪列

ROWNUM

ROWNUM 是一个伪列，它返回一个数字，表示从查询中获取结果的行编号。第一行的 ROWNUM 为 1，第二行的为 2，依此类推。

ROWNUM 的返回类型为 numeric。ROWNUM 可以用于限制查询返回的总行数，例如下面语句限制查询从 Students 表中返回最多 10 条记录。

```
select * from Students where rownum <= 10;
```

示例

```
panweidb=# SELECT 2 BETWEEN 1 AND 3 AS RESULT;
 result
```

```
----------
 t
(1 row)

panweidb=# SELECT 2 >= 1 AND 2 <= 3 AS RESULT;
 result
----------
 t
(1 row)

panweidb=# SELECT 2 NOT BETWEEN 1 AND 3 AS RESULT;
 result
----------
 f
(1 row)

panweidb=# SELECT 2 < 1 OR 2 > 3 AS RESULT;
 result
----------
 f
(1 row)

panweidb=# SELECT 2+2 IS NULL AS RESULT;
 result
----------
 f
(1 row)

panweidb=# SELECT 2+2 IS NOT NULL AS RESULT;
 result
----------
 t
(1 row)

panweidb=# SELECT 2+2 ISNULL AS RESULT;
```

```
 result
----------
 f
(1 row)

panweidb=# SELECT 2+2 NOTNULL AS RESULT;
 result
----------
 t
(1 row)

panweidb=# SELECT 2+2 IS DISTINCT FROM NULL AS RESULT;
 result
----------
 t
(1 row)

panweidb=# SELECT 2+2 IS NOT DISTINCT FROM NULL AS RESULT;
 result
----------
 f
(1 row)
```

4.1.5.2.条件表达式

在执行 SQL 语句时，可通过条件表达式筛选出符合条件的数据。

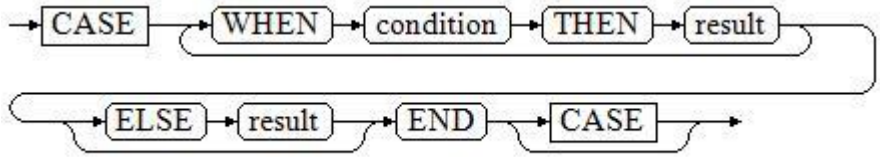
条件表达式主要有以下几种：

- CASE

  CASE 表达式是条件表达式，类似于其他编程语言中的 CASE 语句。

  CASE 表达式的语法图请参考图 1。

  图 1 case::=



CASE 子句可以用于合法的表达式中。condition 是一个返回 BOOLEAN 数据类型的表达式：

- 如果结果为真，CASE 表达式的结果就是符合该条件所对应的 result。
- 如果结果为假，则以相同方式处理随后的 WHEN 或 ELSE 子句。
- 如果各 WHEN condition 都不为真，表达式的结果就是在 ELSE 子句执行的 result。如果省略了 ELSE 子句且没有匹配的条件，结果为 NULL。

示例：

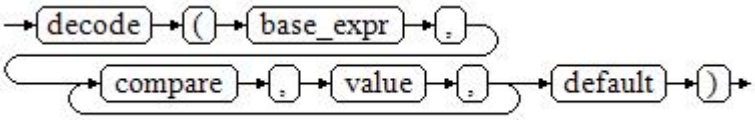
```
panweidb=#CREATE TABLE tpcds.case_when_t1(CW_COL1 INT);

panweidb=#INSERT INTO tpcds.case_when_t1 VALUES (1), (2), (3);

panweidb=#SELECT * FROM tpcds.case_when_t1;
cw_col1
---------
 1
 2
 3
(3 rows)

panweidb=#SELECT CW_COL1, CASE WHEN CW_COL1=1 THEN 'one' WHEN CW_COL1
=2 THEN 'two' ELSE 'other' END FROM tpcds.case_when_t1 ORDER BY 1;
 cw_col1 | case
---------+-------
       1 | one
       2 | two
       3 | other
(3 rows)
```

```
panweidb=#DROP TABLE tpcds.case_when_t1;
```

❖ DECODE

DECODE 的语法图请参考图 2.

图 2 decode::=



将表达式 base_expr 与后面的每个 compare(n) 进行比较，如果匹配返回相应的 value(n)。如果没有发生匹配，则返回 default。

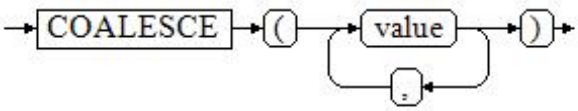
示例请参见条件表达式函数 (参见：开发者指南->SQL 语法参考->函数与操作符->条件表达式函数 章节)。

```
panweidb=#SELECT DECODE('A','A',1,'B',2,0);
 case
------
 1
(1 row)
```

❖ COALESCE

COALESCE 的语法图请参见图 3。

图 3 coalesce::=



COALESCE 返回它的第一个非 NULL 的参数值。如果参数都为 NULL，则返回 NULL。它常用于在显示数据时用缺省值替换 NULL。和 CASE 表达式一样，COALESCE 只计算用来判断结果的参数，即在第一个非空参数右边的参数不会被计算。

示例：

```
panweidb=#CREATE TABLE tpcds.c_tabl(description varchar(10), short_description
varchar(10), last_value varchar(10)) ;

panweidb=#INSERT INTO tpcds.c_tabl VALUES('abc', 'efg', '123');
panweidb=#INSERT INTO tpcds.c_tabl VALUES(NULL, 'efg', '123');
```

```
panweidb=#INSERT INTO tpcds.c_tabl VALUES(NULL, NULL, '123');

panweidb=#SELECT description, short_description, last_value, COALESCE(descriptio
n, short_description, last_value) FROM tpcds.c_tabl ORDER BY 1, 2, 3, 4;
 description | short_description | last_value | coalesce
-------------+-------------------+------------+----------
 abc         | efg               | 123        | abc
             | efg               | 123        | efg
             |                   | 123        | 123
(3 rows)

panweidb=#DROP TABLE tpcds.c_tabl;
```

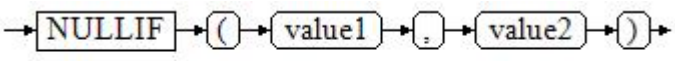
如果 description 不为 NULL，则返回 description 的值，否则计算下一个参数 short_description；如果 short_description 不为 NULL，则返回 short_description 的值，否则计算下一个参数 last_value；如果 last_value 不为 NULL，则返回 last_value 的值，否则返回（none）。

```
panweidb=#SELECT COALESCE(NULL,'Hello World');
  coalesce
---------------
 Hello World
(1 row)
```

❖ NULLIF

NULLIF 的语法图请参见图 4。

图 4 nullif::=



只有当 value1 和 value2 相等时，NULLIF 才返回 NULL。否则它返回 value1。

示例：

```
panweidb=#CREATE TABLE tpcds.null_if_t1 (
  NI_VALUE1 VARCHAR(10),
  NI_VALUE2 VARCHAR(10)
  );
```

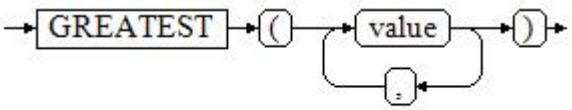
```
panweidb=#INSERT INTO tpcds.null_if_t1 VALUES('abc', 'abc');
panweidb=#INSERT INTO tpcds.null_if_t1 VALUES('abc', 'efg');

panweidb=#SELECT NI_VALUE1, NI_VALUE2, NULLIF(NI_VALUE1, NI_VALUE2) FROM t
pcds.null_if_t1 ORDER BY 1, 2, 3;

 ni_value1 | ni_value2 | nullif
-----------+-----------+--------
 abc       | abc       |
 abc       | efg       | abc
(2 rows)
panweidb=#DROP TABLE tpcds.null_if_t1;
如果 value1 等于 value2 则返回 NULL，否则返回 value1。
```

```
panweidb=#SELECT NULLIF('Hello','Hello World');
 nullif
--------
Hello
(1 row)
```

❖ GREATEST（最大值），LEAST（最小值）

GREATEST 的语法图请参见图 5。

图 5 greatest::=

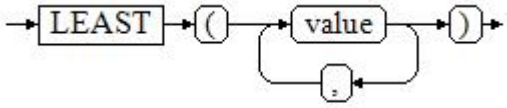


从一个任意数字表达式的列表里选取最大的数值。

```
panweidb=#SELECT greatest(9000,155555,2.01);
 greatest
----------
   155555
(1 row)
```

LEAST 的语法图请参见图 6。

图 6 least::=



从一个任意数字表达式的列表里选取最小的数值。

以上的数字表达式必须都可以转换成一个普通的数据类型，该数据类型将是结果类型。

列表中的 NULL 值将被忽略。只有所有表达式的结果都是 NULL 的时候，结果才是 NULL。

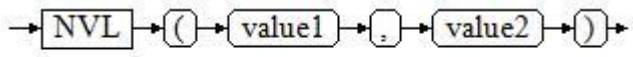
```
panweidb=#SELECT least(9000,2);
 least
-------
   2
(1 row)
```

示例请参见：开发者指南->SQL 语法参考->SQL 基本要素->函数和操作符->条件表达式函数章节。

❖ NVL

NVL 的语法图请参见图 7。

图 7 nvl::=



如果 value1 为 NULL 则返回 value2，如果 value1 非 NULL，则返回 value1。

示例：

```
panweidb=#SELECT nvl(null,1);
nvl
-----
 1
(1 row)
```

```
panweidb=#SELECT nvl ('Hello World' ,1);
    nvl
---------------
 Hello World
(1 row)
```

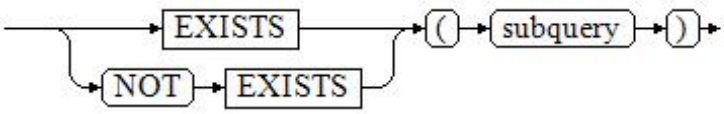
4.1.5.3.子查询表达式

子查询表达式主要有以下几种：

- EXISTS/NOT EXISTS

  EXISTS/NOT EXISTS 的语法图请参见图 1。

  图 1 EXISTS/NOT EXISTS::=

  EXISTS 的参数是一个任意的 SELECT 语句，或者说子查询。系统对子查询进行运算以判断它是否返回行。如果它至少返回一行，则 EXISTS 结果就为“真”；如果子查询没有返回任何行，EXISTS 的结果是“假”。

  这个子查询通常只是运行到能判断它是否可以生成至少一行为止，而不是等到全部结束。

  示例：

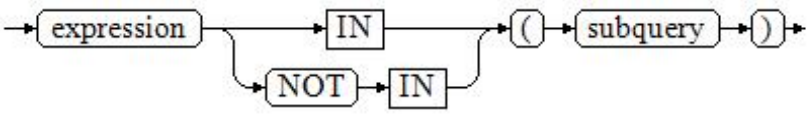
```
panweidb=# SELECT sr_reason_sk,sr_customer_sk FROM tpcds.store_returns WHERE
 EXISTS (SELECT d_dom FROM tpcds.date_dim WHERE d_dom = store_returns.sr_rea
son_sk and sr_customer_sk <10);
sr_reason_sk | sr_customer_sk
  -------------+----------------
         13 |              2
         22 |              5
         17 |              7
         25 |              7
          3 |              7
```

```
      31 |         5
       7 |         7
      14 |         6
      20 |         4
       5 |         6
      10 |         3
       1 |         5
      15 |         2
       4 |         1
      26 |         3
(15 rows)
```

❖ IN/NOT IN

IN/NOT IN 的语法请参见图 2。

图 2 IN/NOT IN::=



右边是一个圆括弧括起来的子查询，它必须只返回一个字段。左边表达式对子查询结果的每一行进行一次计算和比较。如果找到任何相等的子查询行，则 IN 结果为“真”。如果没有找到任何相等行，则结果为“假”（包括子查询没有返回任何行的情况）。

表达式或子查询行里的 NULL 遵照 SQL 处理布尔值和 NULL 组合时的规则。如果两个行对应的字段都相等且非空，则这两行相等；如果任意对应字段不等且非空，则这两行不等；否则结果是未知（NULL）。如果每一行的结果都是不等或 NULL，并且至少有一个 NULL，则 IN 的结果是 NULL。

示例：

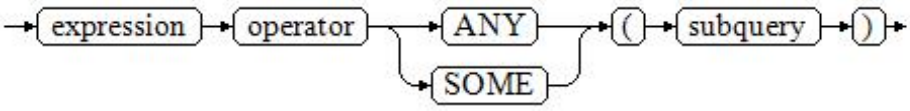
```
panweidb=# SELECT sr_reason_sk,sr_customer_sk  FROM tpcds.store_returns WHER
E sr_customer_sk IN (SELECT d_dom FROM tpcds.date_dim WHERE d_dom < 10);
 sr_reason_sk | sr_customer_sk
--------------+----------------
       10 |         3
       26 |         3
```

```
      22 |         5
      31 |         5
       1 |         5
      32 |         5
      32 |         5
       4 |         1
      15 |         2
      13 |         2
      33 |         4
      20 |         4
      33 |         8
       5 |         6
      14 |         6
      17 |         7
       3 |         7
      25 |         7
       7 |         7
(19 rows)
```

❖ ANY/SOME

ANY/SOME 的语法图请参见图 3。

图 3 any/some::=



右边是一个圆括弧括起来的子查询，它必须只返回一个字段。左边表达式使用 operator 对子查询结果的每一行进行一次计算和比较，其结果必须是布尔值。如果至少获得一个真值，则 ANY 结果为“真”。如果全部获得假值，则结果是“假”（包括子查询没有返回任何行的情况）。SOME 是 ANY 的同义词。IN 与 ANY 可以等效替换 。

示例：

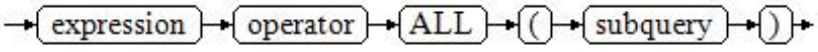
```
panweidb=# SELECT sr_reason_sk,sr_customer_sk  FROM tpcds.store_returns WHER
E sr_customer_sk < ANY (SELECT d_dom FROM tpcds.date_dim WHERE d_dom < 10);
 sr_reason_sk | sr_customer_sk
```

```
-------------+----------------
       26 |              3
       17 |              7
       32 |              5
       32 |              5
       13 |              2
       31 |              5
       25 |              7
        5 |              6
        7 |              7
       10 |              3
        1 |              5
       14 |              6
        4 |              1
        3 |              7
       22 |              5
       33 |              4
       20 |              4
       33 |              8
       15 |              2
(19 rows)
```

❖ ALL

ALL 的语法请参见图 4。

图 4 all::=



右边是一个圆括弧括起来的子查询，它必须只返回一个字段。左边表达式使用 operator 对子查询结果的每一行进行一次计算和比较，其结果必须是布尔值。如果全部获得真值，ALL 结果为“真”（包括子查询没有返回任何行的情况）。如果至少获得一个假值，则结果是“假”。

示例：

```
panweidb=# SELECT sr_reason_sk,sr_customer_sk  FROM tpcds.store_returns WHER
E sr_customer_sk < all(SELECT d_dom FROM tpcds.date_dim WHERE d_dom < 10);
 sr_reason_sk | sr_customer_sk
```

```
--------------+----------------
(0 rows)
```

4.1.5.4.数组表达式

IN

expression **IN** (value [, ...])

右侧括号中的是一个表达式列表。左侧表达式的结果与表达式列表的内容进行比较。如果列表中的内容符合左侧表达式的结果，则 IN 的结果为 true。如果没有相符的结果，则 IN 的结果为 false。

示例如下：

```
panweidb=# SELECT 8000+500 IN (10000, 9000) AS RESULT;
 result
----------
f
(1 row)
```

如果表达式结果为 null，或者表达式列表不符合表达式的条件且右侧表达式列表返回结果至少一处为空，则 IN 的返回结果为 null，而不是 false。这样的处理方式和 SQL 返回空值的布尔组合规则是一致的。

NOT IN

expression **NOT IN** (value [, ...])

右侧括号中的是一个表达式列表。左侧表达式的结果与表达式列表的内容进行比较。如果在列表中的内容没有符合左侧表达式结果的内容，则 NOT IN 的结果为 true。如果有符合的内容，则 NOT IN 的结果为 false。

示例如下：

```
panweidb=# SELECT 8000+500 NOT IN (10000, 9000) AS RESULT;
 result
----------
t
```

```
(1 row)
```

如果查询语句返回结果为空，或者表达式列表不符合表达式的条件且右侧表达式列表返回结果至少一处为空，则 NOT IN 的返回结果为 null，而不是 false。这样的处理方式和 SQL 返回空值的布尔组合规则是一致的。

说明

在所有情况下 X NOT IN Y 等价于 NOT(X IN Y)。

ANY/SOME (array)

expression operator **ANY** (array expression)

expression operator **SOME** (array expression)

```
panweidb=# SELECT 8000+500 < SOME (array[10000,9000]) AS RESULT;
 result
----------
 t
(1 row)
panweidb=# SELECT 8000+500 < ANY (array[10000,9000]) AS RESULT;
 result
----------
 t
(1 row)
```

右侧括号中的是一个数组表达式，它必须产生一个数组值。左侧表达式的结果使用操作符对数组表达式的每一行结果都进行计算和比较，比较结果必须是布尔值。

- 如果对比结果至少获取一个真值，则 ANY 的结果为 true。
- 如果对比结果没有真值，则 ANY 的结果为 false。
- 如果结果没有真值，并且数组表达式生成至少一个值为 null，则 ANY 的值为 NULL，而不是 false。这样的处理方式和 SQL 返回空值的布尔组合规则是一致的。

📖 说明

SOME 是 ANY 的同义词。

ALL (array)

expression operator **ALL** (array expression)

右侧括号中的是一个数组表达式，它必须产生一个数组值。左侧表达式的结果使用操作符对数组表达式的每一行结果都进行计算和比较，比较结果必须是布尔值。

- 如果所有的比较结果都为真值（包括数组不含任何元素的情况），则 ALL 的结果为 true。
- 如果存在一个或多个比较结果为假值，则 ALL 的结果为 false。
- 如果数组表达式产生一个 NULL 数组，则 ALL 的结果为 NULL。如果左边表达式的值为 NULL，则 ALL 的结果通常也为 NULL（某些不严格的比较操作符可能得到不同的结果）。另外，如果右边的数组表达式中包含 null 元素并且比较结果没有假值，则 ALL 的结果将是 NULL（某些不严格的比较操作符可能得到不同的结果），而不是真。这样的处理方式和 SQL 返回空值的布尔组合规则是一致的。

```
panweidb=# SELECT 8000+500 < ALL (array[10000,9000]) AS RESULT;
 result
----------
 t
(1 row)
```

4.1.5.5.行表达式

语法格式

```
row_constructor operator row_constructor
```

- 两边都是一个行构造器，两行值必须具有相同数目的字段，每一行都进行比较，行比较允许使用=、<>、<、<=、>=等操作符，或其中一个相似的语义符。

- =<>和别的操作符使用略有不同。如果两行值的所有字段都是非空并且相等，则认为两行是相等的；如果两行值的任意字段为非空并且不相等，则认为两行是不相等的；否则比较结果是未知的（null）。
- 对于<、<=、>、> =的情况下，行中元素从左到右依次比较，直到遇到一对不相等的元素或者一对为空的元素。如果这对元素中存在至少一个 null 值，则比较结果是未知的（null），否则这对元素的比较结果为最终的结果。

示例

```
panweidb=# SELECT ROW(1,2,NULL) < ROW(1,3,0) AS RESULT;
 result
----------
t
(1 row)
```

4.1.6.事务控制

事务是用户定义的一个数据库操作序列，这些操作要么全做要么全不做，是一个不可分割的工作单位。

启动事务

PanWeiDB 通过 START TRANSACTION 和 BEGIN 语法启动事务，请参见：开发者指南->SQL 语法参考->SQL 语法->START TRANSACTION 章节和 BEGIN 章节。

设置事务

PanWeiDB 通过 SET TRANSACTION 或者 SET LOCAL TRANSACTION 语法设置事务，请参见：开发者指南->SQL 语法参考->SQL 语法->SET TRANSACTION 章节。

提交事务

PanWeiDB 通过 COMMIT 或者 END 可完成提交事务的功能，即提交事务的所有操作，请参见：开发者指南->SQL 语法参考->SQL 语法->COMMIT | END 章节。

回滚事务

回滚是在事务运行的过程中发生了某种故障，事务不能继续执行，系统将事务中对数据库的所有已完成的操作全部撤销。请参见：开发者指南->SQL 语法参考->SQL 语法->ROLLBACK 章节。

说明

数据库中收到的一次执行请求（不在事务块中），如果含有多条语句，将会被打包成一个事务，如果其中有一个语句失败，那么整个请求都将会被回滚。

4.1.7.系统操作

PanWeiDB 通过 SQL 语句执行不同的系统操作，比如：设置变量、显示执行计划和垃圾收集等操作。

设置变量

设置会话或事务中需要使用的各种参数，请参见：开发者指南->SQL 语法参考->SQL 语法->SET 章节。

显示执行计划

显示 PanWeiDB 为 SQL 语句规划的执行计划，请参见：开发者指南->SQL 语法参考->SQL 语法->EXPLAIN 章节。

事务日志检查点

预写式日志（WAL）缺省时在事务日志中每隔一段时间放置一个检查点。CHECKPOINT 强制立即进行检查，而不是等到下一次调度时的检查点。请参见：开发者指南->SQL 语法参考->SQL 语法->CHECKPOINT 章节。

垃圾收集

进行垃圾收集以及可选择的对数据库进行分析。请参见：开发者指南->SQL 语法参考->SQL 语法->VACUUM 章节。

收集统计信息

收集与数据库中表内容相关的统计信息。请参见：开发者指南->SQL 语法参考->SQL 语法->ANALYZE 和 ANALYSE 章节。

设置当前事务的约束检查模式

设置当前事务里的约束检查的特性。请参见：开发者指南->SQL 语法参考->SQL 语法->SET CONSTRAINTS。

关闭当前数据库节点

关闭当前数据库节点，请参见：开发者指南->SQL 语法参考->SQL 语法->SHUTDOWN。

4.2.DML 语法一览表

功能描述

DML（Data Manipulation Language 数据操作语言），用于对数据库表中的数据进行操作。如：插入、更新、查询、删除。

插入数据

插入数据是往数据库表中添加一条或多条记录，请参见：开发者指南->SQL 语法参考->SQL 语法->INSERT。

修改数据

修改数据是修改数据库表中的一条或多条记录，请参见：开发者指南->SQL 语法参考->SQL 语法->UPDATE。

查询数据

数据库查询语句 SELECT 是用于在数据库中检索适合条件的信息，请参见：开发者指南->SQL 语法参考->SQL 语法->SELECT。

删除数据

PanWeiDB 提供了两种删除表数据的语句：删除表中指定条件的数据，请参见：开发者指南->SQL 语法参考->SQL 语法->DELETE；或删除表的所有数据，请参见：开发者指南->SQL 语法参考->SQL 语法->TRUNCATE。

TRUNCATE 快速地从表中删除所有行，它和在每个表上进行无条件的 DELETE 有同样的效果，不过因为它不做表扫描，因而快得多。在大表上最有用。

拷贝数据

PanWeiDB 提供了在表和文件之间拷贝数据的语句，请参见：开发者指南->SQL 语法参考->SQL 语法->COPY。

锁定表

PanWeiDB 提供了多种锁模式用于控制对表中数据的并发访问，请参见：开发者指南->SQL 语法参考->SQL 语法->LOCK。

调用函数

PanWeiDB 提供了三个用于调用函数的语句，它们在语法结构上没有差别，请参见：开发者指南->SQL 语法参考->SQL 语法->CALL。

操作会话

用户与数据库之间建立的连接称为会话，请参考下表：

会话相关 SQL

功能	相关 SQL
修改会话	ALTER SESSION(参见：开发者指南->SQL 语法参考->SQL 语法->ALTER SESSION 章节)
结束会话	ALTER SYSTEM KILL SESSION(参见：开发者指南->SQL 语法参考->SQL 语法->ALTER SYSTEM KILL SESSION)

4.3.DDL 语法一览表

功能描述

DDL（Data Definition Language 数据定义语言），用于定义或修改数据库中的对象。如：表、索引、视图等。

【说明】
PanWeiDB 不支持数据库主节点不完整时进行 DDL 操作。例如：PanWeiDB 中有 1 个数据库主节点故障时执行新建数据库、表等操作都会失败。

定义客户端加密主密钥

客户端加密主密钥主要用于密态数据库特性，用来加密列加密密钥(cek)。客户端加密主密钥定义主要包括创建客户端加密主密钥以及删除客户端加密主密钥。所涉及的 SQL 语句，请参考下表：

表-1 客户端加密主密钥定义相关 SQL

功能	相关 SQL
创建客户端加密主密钥	CREATE CLIENT MASTER KEY(参见：开发者指南->SQL 语法参考->SQL 语法->CREATE-CLIENT-MASTER-KEY 章节)
删除客户端加密主密钥	DROP CLIENT MASTER KEY(参见：开发者指南->SQL 语法参考->SQL 语法->DROP CLIENT MASTER KEY 章节)

定义列加密密钥

列加密密钥主要用于密态数据库特性中，用来加密数据。列加密密钥定义主要包括创建列加密密钥以及删除列加密密钥。所涉及的 SQL 语句，请参考下表：

表-2 列加密密钥定义相关 SQL

功能	相关 SQL
创建列加密密钥	CREATE COLUMN ENCRYPTION KEY(参见：开发者指南->SQL 语法参考->SQL 语法->CREATE COLUMN ENCRYPTION KEY 章节)
删列加密密钥	DROP COLUMN ENCRYPTION KEY(参见：开发者指南->SQL 语法参考->SQL 语法->DROP COLUMN ENCRYPTION KEY 章节)

定义数据库

数据库是组织、存储和管理数据的仓库，而数据库定义主要包括：创建数据库、修改数据库属性，以及删除数据库。所涉及的 SQL 语句，请参考下表：

表-3 数据库定义相关 SQL

功能	相关 SQL
创建数据库	CREATE DATABASE(参见：开发者指南->SQL 语法参考->SQL 语法->CREATE DATABASE 章节)
修改数据库属性	ALTER DATABASE(参见：开发者指南->SQL 语法参考->SQL 语法->ALTER DATABASE 章节)
删除数据	DROP DATABASE(参见：开发者指南->SQL 语法参考->SQL 语法->DROP

功能	相关 SQL
库	DATABASE 章节)

定义模式

模式是一组数据库对象的集合，主要用于控制对数据库对象的访问。所涉及的 SQL 语句，请参考下表：

表-4 模式定义相关 SQL

功能	相关 SQL
创建模式	CREATE SCHEMA(参见：开发者指南->SQL 语法参考->SQL 语法->CREATE-SCHEMA 章节)
修改模式属性	ALTER SCHEMA(参见：开发者指南->SQL 语法参考->SQL 语法->ALTER SCHEMA 章节)
删除模式	DROP SCHEMA(参见：开发者指南->SQL 语法参考->SQL 语法->DROP SCHEMA 章节)

定义表空间

表空间用于管理数据对象，与磁盘上的一个目录对应。所涉及的 SQL 语句，请参考下表：

表-5 表空间定义相关 SQL

功能	相关 SQL
创建表空间	CREATE TABLESPACE(参见：开发者指南->SQL 语法参考->SQL 语法->CREATE TABLESPACE 章节)
修改表空间属性	ALTER TABLESPACE(参见：开发者指南->SQL 语法参考->SQL 语法->ALTER TABLESPACE 章节)
删除表空间	DROP TABLESPACE(参见：开发者指南->SQL 语法参考->SQL 语法->DROP TABLESPACE 章节)

定义表

表是数据库中的一种特殊数据结构，用于存储数据对象以及对象之间的关系。所涉及的 SQL 语句，请参考下表：

表-6 表定义相关 SQL

功能	相关 SQL
创建表	CREATE TABLE(参见：开发者指南->SQL 语法参考->SQL 语法->CREATE TABLE 章节)
修改表属性	ALTER TABLE(参见：开发者指南->SQL 语法参考->SQL 语法->ALTER TABLE 章节)
删除表	DROP TABLE(参见：开发者指南->SQL 语法参考->SQL 语法->DROP TABLE 章节)

定义分区表

分区表是一种逻辑表，数据是由普通表存储的，主要用于提升查询性能。所涉及的 SQL 语句，请参考下表：

表-7 分区表定义相关 SQL

功能	相关 SQL
创建分区表	CREATE TABLE PARTITION(参见：开发者指南->SQL 语法参考->SQL 语法->CREATE TABLE PARTITION 章节)
创建分区	ALTER TABLE PARTITION(参见：开发者指南->SQL 语法参考->SQL 语法->ALTER TABLE PARTITION 章节)
修改分区表属性	ALTER TABLE PARTITION(参见：开发者指南->SQL 语法参考->SQL 语法->ALTER TABLE-PARTITION 章节)
删除分区	ALTER TABLE PARTITION(参见：开发者指南->SQL 语法参考->SQL 语法->ALTER TABLE PARTITION 章节)
删除分区表	DROP TABLE(参见：开发者指南->SQL 语法参考->SQL 语法->DROP TABLE 章节)

定义索引

索引是对数据库表中一列或多列的值进行排序的一种结构，使用索引可快速访问数据库表中的特定信息。所涉及的 SQL 语句，请参考下表：

表-8 索引定义相关 SQL

功能	相关 SQL
创建索引	CREATE INDEX(参见：开发者指南->SQL 语法参考->SQL 语法->CREATE INDEX 章节)
修改索引属	ALTER INDEX(参见：开发者指南->SQL 语法参考->SQL 语法->ALTER

功能	相关 SQL
性	INDEX 章节)
删除索引	DROP INDEX(参见：开发者指南->SQL 语法参考->SQL 语法->DROP-INDEX 章节)
重建索引	REINDEX(参见：开发者指南->SQL 语法参考->SQL 语法->REINDEX 章节)

定义存储过程

存储过程是一组为了完成特定功能的 SQL 语句集，经编译后存储在数据库中，用户通过指定存储过程的名称并给出参数（如果该存储过程带有参数）来执行它。所涉及的 SQL 语句，请参考下表：

表-9 存储过程定义相关 SQL

功能	相关 SQL
创建存储过程	CREATE PROCEDURE(参见：开发者指南->SQL 语法参考->SQL 语法->CREATE PROCEDURE 章节)
删除存储过程	DROP PROCEDURE(参见：开发者指南->SQL 语法参考->SQL 语法->DROP PROCEDURE 章节)

定义函数

在 PanWeiDB 中，它和存储过程类似，也是一组 SQL 语句集，使用上没有差别。所涉及的 SQL 语句，请参考下表：

表-10 函数定义相关 SQL

功能	相关 SQL
创建函数	CREATE FUNCTION(参见：开发者指南->SQL 语法参考->SQL 语法->CREATE FUNCTION 章节)
修改函数属性	ALTER FUNCTION(参见：开发者指南->SQL 语法参考->SQL 语法->ALTER FUNCTION 章节)
删除函数	DROP FUNCTION(参见：开发者指南->SQL 语法参考->SQL 语法->DROP FUNCTION 章节)

定义视图

视图是从一个或几个基本表中导出的虚表，可用于控制用户对数据访问，请参考下表：

表-11 视图定义相关 SQL

功能	相关 SQL
创建视图	CREATE VIEW(参见：开发者指南->SQL 语法参考->SQL 语法->CREATE VIEW 章节)
删除视图	DROP VIEW(参见：开发者指南->SQL 语法参考->SQL 语法->DROP VIEW 章节)

定义包

包是模块化的思想，由包头（package specification）和包体(package body)组成，用来分类管理存储过程和函数，类似于 Java、C++等语言中的类，请参考下表：

表-12 包定义相关 SQL

功能	相关 SQL
创建包	CREATE PACKAGE(参见：开发者指南->SQL 语法参考->SQL 语法->CREATE PACKAGE 章节)
删除包	DROP PACKAGE(参见：开发者指南->SQL 语法参考->SQL 语法->DROP PACKAGE 章节)
修改包属性	ALTER PACKAGE(参见：开发者指南->SQL 语法参考->SQL 语法->ALTER PACKAGE 章节)

定义游标

为了处理 SQL 语句，存储过程进程分配一段内存区域来保存上下文联系。游标是指向上下文区域的句柄或指针。借助游标，存储过程可以控制上下文区域的变化，请参考下表：

表-13 游标定义相关 SQL

功能	相关 SQL
创建游标	CURSOR(参见：开发者指南->SQL 语法参考->SQL 语法->CURSOR)
移动游标	MOVE(参见：开发者指南->SQL 语法参考->SQL 语法->MOVE)
从游标中提取数	FETCH(参见：开发者指南->SQL 语法参考->SQL 语法->FETCH 章

功能	相关 SQL
据	节)
关闭游标	CLOSE(参见：开发者指南->SQL 语法参考->SQL 语法->CLOSE 章节)

定义聚合函数

又称作汇总函数，可以对一组值执行计算并返回单个值，一般用于对数据集合进行汇总统计。

表-14 聚合函数定义相关 SQL

功能	相关 SQL
创建一个新的聚合函数	CREATE AGGREGATE(参见：开发者指南->SQL 语法参考->SQL 语法->CREATE AGGREGATE 章节)
修改聚合函数	ALTER AGGREGATE(参见：开发者指南->SQL 语法参考->SQL 语法->ALTER AGGREGATE 章节)
删除聚合函数	DROP AGGREGATE(参见：开发者指南->SQL 语法参考->SQL 语法->DROP-AGGREGATE 章节)

定义数据类型转换

将数据分类为不同类型时，都不可避免地需要从一种数据类型转换为另一种数据类型。将一种数据类型转换为另一种数据类型的原因包括将数据从一种数据库类型移植到另一种数据库类型、更改列的数据类型或在数据类型之间临时切换以进行计算。

表-15 数据类型转换定义相关 SQL

功能	相关 SQL
创建一个新的用户自定义数据类型转换	CREATE CAST(参见：开发者指南->SQL 语法参考->SQL 语法->CREATE CAST 章节)
删除用户自定义数据类型转换	DROP CAST(参见：开发者指南->SQL 语法参考->SQL 语法->DROP-CAST 章节)

定义插件扩展

开放的插件接口，为开发者开发自定义插件提供了便利。

表-16 插件扩展定义相关 SQL

功能	相关 SQL
创建一个新的插件扩展	CREATE EXTENSION(参见：开发者指南->SQL 语法参考->SQL 语法->CREATE EXTENSION 章节)
修改插件扩展	ALTER EXTENSION(参见：开发者指南->SQL 语法参考->SQL 语法->ALTER EXTENSION 章节)
删除插件扩展	DROP EXTENSION(参见：开发者指南->SQL 语法参考->SQL 语法->DROP EXTENSION 章节)

定义操作符

操作符/运算符大多用于在 SELECT 命令的 WHERE 字句中，表示表达式与其他元素之间的特定算数或逻辑关系，为返回的数据指定更明确的条件。SQL 里有多种操作符，能满足不同的查询需求。

表-17 操作符定义相关 SQL

功能	相关 SQL
创建一个新的操作符	CREATE OPERATOR(参见：开发者指南->SQL 语法参考->SQL 语法->CREATE OPERATOR 章节)
修改操作符	ALTER OPERATOR(参见：开发者指南->SQL 语法参考->SQL 语法->ALTER OPERATOR 章节)
删除操作符	DROP OPERATOR(参见：开发者指南->SQL 语法参考->SQL 语法->DROP OPERATOR 章节)

定义过程语言

表-18 过程语言定义相关 SQL

功能	相关 SQL
修改过程语言	ALTER LANGUAGE(参见：开发者指南->SQL 语法参考->SQL 语法->ALTER LANGUAGE 章节)

定义数据类型

数据类型定义列中存放的值的种类。 SQL 通用数据类型 数据库表中的每个列都要求有名称和数据类型。

表-19 数据类型定义相关 SQL

功能	相关 SQL

功能	相关 SQL
创建一个新的数据类型	CREATE TYPE(参见：开发者指南->SQL 语法参考->SQL 语法->CREATE TYPE 章节)
修改数据类型	ALTER TYPE(参见：开发者指南->SQL 语法参考->SQL 语法->ALTER TYPE 章节)
删除数据类型	DROP TYPE(参见：开发者指南->SQL 语法参考->SQL 语法->DROP TYPE 章节)

4.4.DCL 语法一览表

DCL（Data Control Language 数据控制语言），是用来创建用户角色、设置或更改数据库用户或角色权限的语句。

定义角色

角色是用来管理权限的，从数据库安全的角度考虑，可以把所有的管理和操作权限划分到不同的角色上。所涉及的 SQL 语句，请参考下表：

角色定义相关 SQL

功能	相关 SQL
创建角色	CREATE ROLE(参见：开发者指南->SQL 语法参考->SQL 语法->CREATE-ROLE 章节)
修改角色属性	ALTER ROLE(参见：开发者指南->SQL 语法参考->SQL 语法->ALTER ROLE 章节)
删除角色	DROP ROLE(参见：开发者指南->SQL 语法参考->SQL 语法->DROP ROLE 章节)

定义用户

用户是用来登录数据库的，通过对用户赋予不同的权限，可以方便地管理用户对数据库的访问及操作。所涉及的 SQL 语句，请参考下表：

用户定义相关 SQL

功能	相关 SQL
创建用户	CREATE USER(参见：开发者指南->SQL 语法参考->SQL 语法->CREATE USER 章节)
修改用户	ALTER USER(参见：开发者指南->SQL 语法参考->SQL 语法->ALTER-

功能	相关 SQL
属性	USER 章节)
删除用户	DROP USER(参见：开发者指南->SQL 语法参考->SQL 语法->DROP USER 章节)

授权

PanWeiDB 提供了针对数据对象和角色授权的语句，请参见：开发者指南->SQL 语法参考->SQL 语法->GRANT 章节。

收回权限

PanWeiDB 提供了收回权限的语句，请参见：开发者指南->SQL 语法参考->SQL 语法->REVOKE 章节。

设置默认权限

PanWeiDB 允许设置应用于将来创建的对象的权限，请参见：开发者指南->SQL 语法参考->SQL 语法->ALTER DEFAULT PRIVILEGES 章节。

关闭当前节点

PanWeiDB 支持使用 shutdown 命令关闭当前数据库节点，请参见：开发者指南->SQL 语法参考->SQL 语法->SHUTDOWN 章节。

4.5.SQL 语法

本章主要介绍 PanWeiDB 支持的 SQL 语法。

文档从功能描述、语法格式、参数说明、示例等各方面介绍了各 SQL 语法的详细信息，帮助用户使用它们。

4.5.1.ABORT

功能描述

回滚当前事务并且撤销所有当前事务中所做的更改。

作用等同于 ROLLBACK(参见：开发者指南->SQL 语法参考->SQL 语法->ROLLBACK 章节)，早期 SQL 有用 ABORT，现在推荐使用 ROLLBACK。

注意事项

在事务外部执行 ABORT 语句不会影响事务的执行，但是会抛出一个 NOTICE 信息。

语法格式

```
ABORT [ WORK | TRANSACTION ] ;
```

参数说明

WORK | TRANSACTION

可选关键字，除了增加可读性没有其他任何作用。

示例

```
--创建表 customer_demographics_t1。
panweidb=# CREATE TABLE customer_demographics_t1
(
   CD_DEMO_SK            INTEGER           NOT NULL,
   CD_GENDER             CHAR(1)                ,
   CD_MARITAL_STATUS       CHAR(1)                ,
   CD_EDUCATION_STATUS     CHAR(20)               ,
   CD_PURCCME_ESTIMATE     INTEGER                ,
   CD_CREDIT_RATING        CHAR(10)               ,
   CD_DEP_COUNT          INTEGER                ,
   CD_DEP_EMPLOYED_COUNT    INTEGER                ,
   CD_DEP_COLLEGE_COUNT     INTEGER
)
WITH (ORIENTATION = COLUMN,COMPRESSION=MIDDLE)
;

--插入记录。
panweidb=# INSERT INTO customer_demographics_t1 VALUES(1920801,'M', 'U', 'DO
CTOR DEGREE', 200, 'GOOD', 1, 0,0);

--开启事务。
panweidb=# START TRANSACTION;
```

```
--更新字段值。
panweidb=# UPDATE customer_demographics_t1 SET cd_education_status= 'Unkno
wn';

--终止事务，上面所执行的更新会被撤销掉。
panweidb=# ABORT;

--查询数据。
panweidb=# SELECT * FROM customer_demographics_t1 WHERE cd_demo_sk = 192
0801;
cd_demo_sk | cd_gender | cd_marital_status | cd_education_status  | cd_purccme_es
timate | cd_credit_rating | cd_dep_count | cd_dep_employed_count | cd_dep_colleg
e_count
------------+-----------+------------------+---------------------+------------------
--+-----------------+-------------+----------------------+---------------------
   1920801 | M       | U              | DOCTOR DEGREE       |             200 | GOOD          |
      1 |               0 |               0
(1 row)

--删除表。
panweidb=# DROP TABLE customer_demographics_t1;
```

相关链接

SET TRANSACTION(参见：开发者指南->SQL 语法参考->SQL 语法->SET TRANSACTION 章节)，COMMIT | END(参见：开发者指南->SQL 语法参考->SQL 语法->COMMIT END 章节)，ROLLBACK(参见：开发者指南->SQL 语法参考->SQL 语法->ROLLBACK 章节)

4.5.2.ALTER AUDIT POLICY

功能描述

修改统一审计策略。

注意事项

- 开启审计开关（audit_enabled=on）时，根据具体的审计策略决定需要记录的审计日志。
- 开启安全策略开关（enable_security_policy=on）时，审计策略才能生效。
- 未开启三权分立时，只有安全策略管理员（poladmin）、系统管理员（sysadmin）或初始用户能进行此操作。
- 开启三权分立（enable_Separation_Of_Duty）后，只有安全管理员才能进行此操作。

语法格式

- 修改审计策略设计的数据库操作。

```
ALTER AUDIT POLICY [ IF EXISTS ] policy_name { ADD | REMOVE } { [ privilege_audit_clause ] [ access_audit_clause ] };
```

其中权限审计子句 privilege_audit_clause：

```
PRIVILEGES { DDL | ALL }
```

其中访问审计子句 access_audit_clause：

```
ACCESS { DML | ALL }
```

  - 其中 DDL 可以是：

```
{ ( ALTER | ANALYZE | COMMENT | CREATE | DROP | GRANT | REVOKE | SET | SHOW | LOGIN_ACCESS | LOGIN_FAILURE | LOGOUT | LOGIN ) }
```

  - 其中 DML 可以是：

```
{ ( COPY | DEALLOCATE | DELETE_P | EXECUTE | REINDEX | INSERT | PREPARE | SELECT | TRUNCATE | UPDATE ) }
```

- 修改审计策略的过滤信息。

```
ALTER AUDIT POLICY [ IF EXISTS ] policy_name MODIFY ( filter_group_clause )
```

  - filter_group_clause：

```
FILTER ON { ( FILTER_TYPE ( filter_value [, ... ] ) ) [, ... ] }
```

- 删除审计策略的过滤器。

```
ALTER AUDIT POLICY [ IF EXISTS ] policy_name DROP FILTER;
```

- 修改审计策略的描述信息。

```
ALTER AUDIT POLICY [ IF EXISTS ] policy_name COMMENTS policy_comments;
```

- 开启或关闭审计策略。

```
ALTER AUDIT POLICY [ IF EXISTS ] policy_name { ENABLE | DISABLE };
```

- 向语句序列审计策略中追加 SQL。

```
ALTER AUDIT POLICY [ IF EXISTS ] policy_name ADD SEQUENCE BY sequence_audit_clause;
```

其中，审计语句序列的子句 sequence_audit_clause 可以是：

```
[ 'SQL' ]
```

参数说明

- **IF EXISTS**

  如果指定的审计策略不存在，则发出一个 notice 而不是 error。

- **policy_name**

  审计策略名称，需要唯一，不可重复。

  取值范围：字符串，要符合标识符的命名规范。

- **ADD | MOVE**

  给审计策略新增或移除数据库的操作。

- **DDL**

  指的是针对数据库执行如下操作时进行审计，目前支持：CREATE、ALTER、DROP、ANALYZE、COMMENT、GRANT、REVOKE、SET、SHOW、LOGIN_ANY、LOGIN_FAILURE、LOGIN_SUCCCESS、LOGOUT、LOGIN。

- **ALL**

  指的是上述 DDL 支持的所有对数据库的操作。

- **DML**

  指的是针对数据库执行如下操作时进行审计，目前支持：SELECT、COPY、DEALLOCATE、DELETE、EXECUTE、INSERT、PREPARE、REINDEX、TRUNCATE、UPDATE。

- **FILTER_TYPE**

  指定审计策略的过滤信息，过滤类型包括：IP、ROLES、APP、TIME_PERIOD，分别表示将 IP、访问的应用名、数据库系统用户以及重要时间作为审计的过滤条件。

- **filter_value**

  指具体过滤信息内容。

- **policy_comments**

  用于记录策略相关的描述信息。

- **ENABLE|DISABLE**

  可以打开或关闭统一审计策略。若不指定 ENABLE|DISABLE，语句默认为 ENABLE。

- **['SQL']**

  需要向审计策略中追加的 SQL。根据预先建立的语句序列审计规则，如果数据库中某个会话依次执行了序列中的 SQL 语句，就会触发审计。给出的 SQL 会追加到原 SQL 之后。

  - 每次仅能向审计策略中添加一条 SQL；SQL 语句应被单引号及方括号包围。
  - 一条 SQL 只能被一个审计策略所包含；不能出现两个语句序列完全相同的审计策略。
  - 被审计的 SQL 不能存在语法错误。

  【说明】

  在 MySQL 兼容模式下，执行的会话 SQL 必须于策略中定义的 SQL 大小写完全一致时，才会触发审计。

示例

对 CREATE AUDIT POLICY 示例中创建的统一审计策略进行修改。

- 修改审计策略 adt1，为审计策略新增所有的 DML 访问：

```
ALTER AUDIT POLICY adt1 ADD ACCESS(ALL);
```

- 修改审计策略 adt2 的描述信息：

```
ALTER AUDIT POLICY adt2 COMMENTS 'It was created to audit the select sql of t
he database';
```

- 删除审计策略 aud3 的过滤器：

```
ALTER AUDIT POLICY adt3 drop filter;
```

- 禁用审计策略 adt4：

```
ALTER AUDIT POLICY adt4 DISABLE;
```

- 修改审计策略 adt5，向策略中追加 SQL：

```
ALTER AUDIT POLICY adt5 add sequence by ['create table tb_123(id int);'];
```

相关链接

CREATE AUDIT POLICY(参见：开发者指南->SQL 语法参考->SQL 语法->CREATE-AUDIT-POLICY 章节)，DROP AUDIT POLICY(参见：开发者指南->SQL 语法参考->SQL 语法->DROP AUDIT POLICY 章节)。

4.5.3.ALTER AGGREGATE

功能描述

修改一个聚合函数的定义。

注意事项

要使用 ALTER AGGREGATE ，你必须是该聚合函数的所有者。 要改变一个聚合函数的模式，你必须在新模式上有 CREATE 权限。要改变所有者，你必须是新所有角色的一个直接或间接成员，并且该角色必须在聚合函数的模式上有 CREATE 权限。（这些限制强制了修改该所有者不会做任何通过删除和重建聚合函数不能做的事情。不过，具有 SYSADMIN 权限用户可以用任何方法任意更改聚合函数的所属关系）。

语法格式

```
ALTER AGGREGATE name ( argtype [ , ... ] ) RENAME TO new_name
ALTER AGGREGATE name ( argtype [ , ... ] ) OWNER TO new_owner
ALTER AGGREGATE name ( argtype [ , ... ] ) SET SCHEMA new_schema
```

参数说明

- **name**

  现有的聚合函数的名称（可以有模式修饰）。

- **argtype**

  聚合函数操作的输入数据类型。要引用一个零参数聚合函数，可以写入*代替输入数据类型列表。

- **new_name**

  聚合函数的新名字。

- **new_owner**

  聚合函数的新所有者。

❖ **new_schema**

聚合函数的新模式。

示例

把一个接受 integer 类型参数的聚合函数 myavg 重命名为 my_average：

```
ALTER AGGREGATE myavg(integer) RENAME TO my_average;
```

把一个接受 integer 类型参数的聚合函数 myavg 的所有者改为 joe：

```
ALTER AGGREGATE myavg(integer) OWNER TO joe;
```

把一个接受 integer 类型参数的聚合函数 myavg 移动到模式 myschema 里：

```
ALTER AGGREGATE myavg(integer) SET SCHEMA myschema;
```

兼容性

SQL 标准里没有 ALTER AGGREGATE 语句。

4.5.4.ALTER DATA SOURCE

功能描述

修改 Data Source 对象的属性和内容。

属性有：名称和属主；内容有：类型、版本和连接选项。

注意选项

❖ 只有初始用户、系统管理员和属主才拥有修改 Data Source 的权限。
❖ 修改属主时，新的属主用户必须是初始用户或系统管理员。
❖ 当在 OPTIONS 中出现 password 选项时，需要保证 PanWeiDB 每个节点的$GAUSSHOME/bin 目录下存在 datasource.key.cipher 和 datasource.key.rand 文件，如果不存在这两个文件，请使用 pw_guc 工具生成并发布到每个节点的$GAUSSHOME/bin 目录下。

语法格式

```
ALTER DATA SOURCE src_name
```

```
  [TYPE  'type_str']
  [VERSION  {'version_str' | NULL}]
  [OPTIONS ( {[ ADD | SET | DROP ] optname ['optvalue']} [, ...] )];
ALTER DATA SOURCE src_name RENAME TO src_new_name;
ALTER DATA SOURCE src_name OWNER TO new_owner;
```

参数说明

- **src_name**

  待修改的 Data Source 的名称。

  取值范围：字符串，需要符合标识符的命名规范。

- **TYPE**

  将 Data Source 原来的 TYPE 修改为指定值。

  取值范围：空串或非空字符串。

- **VERSION**

  将 Data Source 原来的 VERSION 修改为指定值。

  取值范围：空串或非空字符串或 NULL。

- **OPTIONS**

  修改 OPTIONS 中的字段：增加（ADD）、修改（SET）、删除（DROP），且字段名称 optname 需唯一，具体要求如下：

  增加字段：ADD 可以省略，待增加字段不能已经存在了；

  修改字段：SET 不可省略，待修改字段必须存在；

  删除字段：DROP 不可省略，待删除字段必须存在，且不能指定 optvalue；

- **src_new_name**

  新的 Data Source 名称。

  取值范围：字符串，需符合标识符命名规范。

- **new_user**

  对象的新属主。

  取值范围：字符串，有效的用户名。

示例

```
--创建一个空 Data Source 对象。
```

```
panweidb=# CREATE DATA SOURCE ds_test1;

--修改名称。
panweidb=# ALTER DATA SOURCE ds_test1 RENAME TO ds_test;

--修改属主。
panweidb=# CREATE USER user_test1 IDENTIFIED BY 'Gs@123456';
panweidb=# ALTER USER user_test1 WITH SYSADMIN;
panweidb=# ALTER DATA SOURCE ds_test OWNER TO user_test1;

--修改 TYPE 和 VERSION。
panweidb=# ALTER DATA SOURCE ds_test TYPE 'MPPDB_TYPE' VERSION 'XXX';

--添加字段。
panweidb=# ALTER DATA SOURCE ds_test OPTIONS (add dsn 'gaussdb', username 't
est_user');

--修改字段。
panweidb=# ALTER DATA SOURCE ds_test OPTIONS (set dsn 'unknown');

--删除字段。
panweidb=# ALTER DATA SOURCE ds_test OPTIONS (drop username);

--删除 Data Source 和 user 对象。
panweidb=# DROP DATA SOURCE ds_test;
panweidb=# DROP USER user_test1;
```

相关链接

CREATE DATA SOURCE (参见：开发者指南->SQL 语法参考->SQL 语法->CREATE DATA SOURCE 章节)，DROP DATA SOURCE(参见：开发者指南->SQL 语法参考->SQL 语法->DROP DATA SOURCE 章节)

4.5.5.ALTER DATABASE

功能描述

修改数据库的属性，包括它的名称、所有者、连接数限制、对象隔离属性等。

注意事项

- 只有数据库的所有者或者被授予了数据库 ALTER 权限的用户才能执行 ALTER DATABASE 命令，系统管理员默认拥有此权限。针对所要修改属性的不同，还有以下权限约束：
  - 修改数据库名称，必须拥有 CREATEDB 权限。
  - 修改数据库所有者，当前用户必须是该 database 的所有者或者系统管理员，必须拥有 CREATEDB 权限，且该用户是新所有者角色的成员。
  - 修改数据库默认表空间，必须拥有新表空间的 CREATE 权限。这个语句会从物理上将一个数据库原来缺省表空间上的表和索引移至新的表空间。注意不在缺省表空间的表和索引不受此影响。
- 不能重命名当前使用的数据库，如果需要重新命名，须连接至其他数据库上。

语法格式

- 修改数据库的最大连接数。

```
ALTER DATABASE database_name
  [ [ WITH ] CONNECTION LIMIT connlimit ];
```

- 修改数据库名称。

```
ALTER DATABASE database_name
  RENAME TO new_name;
```

- 修改数据库所属者。

```
ALTER DATABASE database_name
  OWNER TO new_owner;
```

- 修改数据库默认表空间。

```
ALTER DATABASE database_name
  SET TABLESPACE new_tablespace;
```

【说明】

如果该数据库中的某些表或对象已经创建在 new_tablespace 下，则无法将该数据库的默认表空间修改为 new_tablespace，执行会报错。

❖ 修改数据库指定会话参数值。

```
ALTER DATABASE database_name
  SET configuration_parameter { { TO | = } { value | DEFAULT } | FROM CURRENT };
```

❖ 数据库配置参数重置。

```
ALTER DATABASE database_name RESET
  { configuration_parameter | ALL };
```

❖ 修改数据库对象隔离属性。

```
ALTER DATABASE database_name [ WITH ] { ENABLE | DISABLE } PRIVATE OBJECT;
```

【说明】

❖ 修改数据库的对象隔离属性时须连接至该数据库，否则无法更改。

❖ 新创建的数据库，对象隔离属性默认是关闭的。当开启数据库对象隔离属性后，普通用户只能查看有权访问的对象（表、函数、视图、字段等）。对象隔离特性对管理员用户不生效，当开启对象隔离特性后，管理员也可以查看到全量的数据库对象。

参数说明

❖ **database_name**

需要修改属性的数据库名称。

取值范围：字符串，要符合标识符的命名规范。

❖ **connlimit**

数据库可以接收的最大并发连接数（管理员用户连接除外）。

取值范围：整数，建议填写 1~50 的整数。-1（缺省）表示没有限制。

❖ **new_name**

数据库的新名称。

取值范围：字符串，要符合标识符的命名规范。

❖ **new_owner**

数据库的新所有者。

取值范围：字符串，有效的用户名。

❖ **new_tablespace**

数据库新的默认表空间，该表空间为数据库中已经存在的表空间。默认的表空间为 pg_default。

取值范围：字符串，有效的表空间名。

- **configuration_parameter**

  value

  把指定的数据库会话参数值设置为给定的值。如果 value 是 DEFAULT 或者 RESET，则在新的会话中使用系统的缺省设置。OFF 关闭设置。

  取值范围：字符串

  - DEFAULT
  - OFF
  - RESET

- **FROM CURRENT**

  根据当前会话连接的数据库设置该参数的值。

- **RESET configuration_parameter**

  重置指定的数据库会话参数值。

- **RESET ALL**

  重置全部的数据库会话参数值。

【说明】

- 修改数据库默认表空间，会将旧表空间中的所有表和索引转移到新表空间中，该操作不会影响其他非默认表空间中的表和索引。
- 修改的数据库会话参数值，将在下一次会话中生效。

示例

请参考 CREATE DATABASE (参见：开发者指南->SQL 语法参考->SQL 语法->CREATE DATABASE 章节)的示例。

相关链接

CREATE DATABASE (参见：开发者指南->SQL 语法参考->SQL 语法->CREATE DATABASE 章节)，DROP DATABASE (参见：开发者指南->SQL 语法参考->SQL 语法->DROP DATABASE 章节)

4.5.6.ALTER DEFAULT PRIVILEGES

功能描述

设置应用于将来创建的对象的权限（这不会影响分配到已有对象中的权限）。

注意事项

目前只支持表（包括视图）、序列、函数、类型、密态数据库客户端主密钥和列加密密钥的权限更改。

语法格式

```
ALTER DEFAULT PRIVILEGES
  [ FOR { ROLE | USER } target_role [, ...] ]
  [ IN SCHEMA schema_name [, ...] ]
  abbreviated_grant_or_revoke;
```

- 其中 abbreviated_grant_or_revoke 子句用于指定对哪些对象进行授权或回收权限。

```
grant_on_tables_clause
    | grant_on_sequences_clause
    | grant_on_functions_clause
    | grant_on_types_clause
    | grant_on_client_master_keys_clause
    | grant_on_column_encryption_keys_clause
    | revoke_on_tables_clause
    | revoke_on_sequences_clause
    | revoke_on_functions_clause
    | revoke_on_types_clause
    | revoke_on_client_master_keys_clause
    | revoke_on_column_encryption_keys_clause
```

- 其中 grant_on_tables_clause 子句用于对表授权。

```
GRANT { { SELECT | INSERT | UPDATE | DELETE | TRUNCATE | REFERENCES | ALTER | DROP
| COMMENT | INDEX | VACUUM }
    [, ...] | ALL [ PRIVILEGES ] }
    ON TABLES
    TO { [ GROUP ] role_name | PUBLIC } [, ...]
```

```
    [ WITH GRANT OPTION ]
```

❖ 其中 grant_on_sequences_clause 子句用于对序列授权。

```
GRANT { { SELECT | UPDATE | USAGE | ALTER | DROP | COMMENT }
    [, ...] | ALL [ PRIVILEGES ] }
    ON SEQUENCES
    TO { [ GROUP ] role_name | PUBLIC } [, ...]
    [ WITH GRANT OPTION ]
```

❖ 其中 grant_on_functions_clause 子句用于对函数授权。

```
GRANT { { EXECUTE | ALTER | DROP | COMMENT } [, ...] | ALL [ PRIVILEGES ] }
    ON FUNCTIONS
    TO { [ GROUP ] role_name | PUBLIC } [, ...]
    [ WITH GRANT OPTION ]
```

❖ 其中 grant_on_types_clause 子句用于对类型授权。

```
GRANT { { USAGE | ALTER | DROP | COMMENT } [, ...] | ALL [ PRIVILEGES ] }
    ON TYPES
    TO { [ GROUP ] role_name | PUBLIC } [, ...]
    [ WITH GRANT OPTION ]
```

❖ 其中 grant_on_client_master_keys_clause 子句用于对客户端主密钥授权。

```
GRANT { { USAGE  | DROP } [, ...] | ALL [ PRIVILEGES ] }
    ON CLIENT_MASTER_KEYS
    TO { [ GROUP ] role_name | PUBLIC } [, ...]
    [ WITH GRANT OPTION ]
```

❖ 其中 grant_on_column_encryption_keys_clause 子句用于对列加密密钥授权。

```
GRANT { { USAGE | DROP } [, ...] | ALL [ PRIVILEGES ] }
    ON COLUMN_ENCRYPTION_KEYS
    TO { [ GROUP ] role_name | PUBLIC } [, ...]
    [ WITH GRANT OPTION ]
```

❖ 其中 revoke_on_tables_clause 子句用于回收表对象的权限。

```
REVOKE [ GRANT OPTION FOR ]
    { { SELECT | INSERT | UPDATE | DELETE | TRUNCATE | REFERENCES | ALTER | DROP | C
OMMENT | INDEX | VACUUM }
    [, ...] | ALL [ PRIVILEGES ] }
```

```
    ON TABLES
    FROM { [ GROUP ] role_name | PUBLIC } [, ...]
    [ CASCADE | RESTRICT | CASCADE CONSTRAINTS ]
```

❖ 其中 revoke_on_sequences_clause 子句用于回收序列的权限。

```
REVOKE [ GRANT OPTION FOR ]
    { { SELECT | UPDATE | USAGE | ALTER | DROP | COMMENT }
    [, ...] | ALL [ PRIVILEGES ] }
    ON SEQUENCES
    FROM { [ GROUP ] role_name | PUBLIC } [, ...]
    [ CASCADE | RESTRICT | CASCADE CONSTRAINTS ]
```

❖ 其中 revoke_on_functions_clause 子句用于回收函数的权限。

```
REVOKE [ GRANT OPTION FOR ]
    { {EXECUTE | ALTER | DROP | COMMENT } [, ...] | ALL [ PRIVILEGES ] }
    ON FUNCTIONS
    FROM { [ GROUP ] role_name | PUBLIC } [, ...]
    [ CASCADE | RESTRICT | CASCADE CONSTRAINTS ]
```

❖ 其中 revoke_on_types_clause 子句用于回收类型的权限。

```
REVOKE [ GRANT OPTION FOR ]
    { { USAGE | ALTER | DROP | COMMENT } [, ...] | ALL [ PRIVILEGES ] }
    ON TYPES
    FROM { [ GROUP ] role_name | PUBLIC } [, ...]
    [ CASCADE | RESTRICT | CASCADE CONSTRAINTS ]
```

❖ 其中 revoke_on_client_master_keys_clause 子句用于回收客户端主密钥的权限。

```
REVOKE [ GRANT OPTION FOR ]
    { { USAGE | DROP } [, ...] | ALL [ PRIVILEGES ] }
    ON CLIENT_MASTER_KEYS
    FROM { [ GROUP ] role_name | PUBLIC } [, ...]
    [ CASCADE | RESTRICT | CASCADE CONSTRAINTS ]
```

❖ 其中 revoke_on_column_encryption_keys_clause 子句用于回收列加密密钥的权限。

```
REVOKE [ GRANT OPTION FOR ]
    { { USAGE | DROP } [, ...] | ALL [ PRIVILEGES ] }
    ON COLUMN_ENCRYPTION_KEYS
```

```
    FROM { [ GROUP ] role_name | PUBLIC } [, ...]
    [ CASCADE | RESTRICT | CASCADE CONSTRAINTS ]
```

参数说明

- **target_role**

  已有角色的名称。如果省略 FOR ROLE/USER，则缺省值为当前角色/用户。

  取值范围：已有角色的名称。

- **schema_name**

  现有模式的名称。

  target_role 必须有 schema_name 的 CREATE 权限。

  取值范围：现有模式的名称。

- **role_name**

  被授予或者取消权限角色的名称。

  取值范围：已存在的角色名称。

  ⚠ **须知：**

  如果想删除一个被赋予了默认权限的角色，有必要恢复改变的缺省权限或者使用 DROP OWNED BY 来为角色脱离缺省的权限记录。

示例

```
--创建模式 tpcds
panweidb=# CREATE SCHEMA tpcds;
--将创建在模式 tpcds 里的所有表（和视图）的 SELECT 权限授予每一个用户。
panweidb=# ALTER DEFAULT PRIVILEGES IN SCHEMA tpcds GRANT SELECT ON TABLE
S TO PUBLIC;

--创建用户普通用户 jack。
panweidb=# CREATE USER jack PASSWORD 'Bigdata@123';

--将 tpcds 下的所有表的插入权限授予用户 jack。
panweidb=# ALTER DEFAULT PRIVILEGES IN SCHEMA tpcds GRANT INSERT ON TABLE
S TO jack;
```

```
--撤销上述权限。
panweidb=# ALTER DEFAULT PRIVILEGES IN SCHEMA tpcds REVOKE SELECT ON TABL
ES FROM PUBLIC;
panweidb=# ALTER DEFAULT PRIVILEGES IN SCHEMA tpcds REVOKE INSERT ON TABL
ES FROM jack;

--删除用户 jack。
panweidb=# DROP USER jack;
```

相关链接

GRANT (参见：开发者指南->SQL 语法参考->SQL 语法->GRANT 章节)，REVOKE(参见：开发者指南->SQL 语法参考->SQL 语法->REVOKE 章节)

4.5.7.ALTER DIRECTORY

功能描述

对 directory 属性进行修改。

注意事项

- 目前只支持修改 directory 属主。
- 当 enable_access_server_directory=off 时，只允许初始用户修改 directory 属主；当 enable_access_server_directory=on 时，具有 SYSADMIN 权限的用户和 directory 对象的属主可以修改 directory，且要求该用户是新属主的成员。

语法格式

```
ALTER DIRECTORY directory_name
  OWNER TO new_owner;
```

参数描述

directory_name

需要修改的目录名称，范围为已经存在的目录名称。

示例

```
--创建目录。
panweidb=# CREATE OR REPLACE DIRECTORY  dir  as '/tmp/';

--修改目录的  owner。
panweidb=# ALTER DIRECTORY dir OWNER TO system;

--删除目录。
panweidb=# DROP DIRECTORY dir;
```

相关链接

CREATE DIRECTORY (参见：开发者指南->SQL 语法参考->SQL 语法->CREATE DIRECTORY 章节)，DROP DIRECTORY(参见：开发者指南->SQL 语法参考->SQL 语法->DROP DIRECTORY 章节)

4.5.8.ALTER EVENT TRIGGER

功能描述

ALTER EVENT TRIGGER 用于修改 DDL 触发器。

注意事项

ALTER EVENT TRIGGER 更改现有 DDL 触发器的属性。

只有超级用户才能更改 DDL 触发器。

语法格式

```
ALTER EVENT TRIGGER name DISABLE
ALTER EVENT TRIGGER name ENABLE [REPLICA | ALWAYS ]
ALTER EVENT TRIGGER name OWNER TO  {new_owner | CURRENT_USER |SESSION_US
ER}
ALTER EVENT TRIGGER name RENAME TO new name
```

参数说明

- name

  要修改的现有触发器的名称。

- new_owner

该 DDL 触发器的新拥有者的用户名。

- new_name

  该 DDL 触发器的新名称。

- DISABLE | ENABLE

  该参数用于配置事件 DDL 触发器是否被触发。一个被禁用的触发器对系统来说仍然是可知的，但是当其触发事件发生时却不会执行它。

- REPLICA

  触发器触发机制受配置变量 session_replication_role 的影响，当复制角色为“origin”（默认值）或“local”时，将触发简单启用的触发器。

  配置为 ENABLE REPLICA 的触发器仅在会话处于“replica”模式时触发。

- ALWAYS

  无论当前复制模式如何，配置为 ENABLE ALWAYS 的触发器都将触发。

相关链接

CREATE EVENT TRIGGER(参见：开发者指南->SQL 语法参考->SQL 语法->CREATE EVENT TRIGGER 章节),DROP EVENT TRIGGER(参见：开发者指南->SQL 语法参考->SQL 语法->DROP EVENT TRIGGER 章节)

4.5.9.ALTER EXTENSION

功能描述

修改插件扩展。

注意事项

ALTER EXTENSION 修改一个已安装的扩展的定义。这里有几种方式：

- UPDATE

  这种方式更新这个扩展到一个新的版本。这个扩展必须满足一个适用的更新脚本（或者一系列脚本）这样就能修改当前安装版本到一个要求的版本。

- SET SCHEMA

  这种方式移动扩展对象到另一个模式。这个扩展必须 relocatable 才能使命令成功。

- ADD member_object

  这种方式添加一个已存在对象到扩展。这主要在扩展更新脚本上有用。 这个对象接着会被视为扩展的成员; 显而易见,该对象只能通过取消扩展来取消 。

- DROP member_object

  这个方式从扩展上移除一个成员对象。 主要在扩展更新脚本上有用。这个对象没有被取消，只是从扩展里分开了。

  您必须拥有扩展来使用 ALTER EXTENSION。 这个 ADD/DROP 方式要求添加/删除对象的所有权。

语法格式

```
ALTER EXTENSION name UPDATE [ TO new_version ];
ALTER EXTENSION name SET SCHEMA new_schema;
ALTER EXTENSION name ADD member_object;
ALTER EXTENSION name DROP member_object;

where member_object is:

 FOREIGN TABLE object_name |
 FUNCTION function_name ( [ [ argmode ] [ argname ] argtype [, ...] ] ) |
 [ PROCEDURAL ] LANGUAGE object_name |
 SCHEMA object_name |
 SERVER object_name |
 TABLE object_name |
 TEXT SEARCH CONFIGURATION object_name |
 TYPE object_name |
 VIEW object_name
```

参数说明

- **name**

  已安装扩展的名称。

- **new_version**

扩展的新版本。可以通过被标识符和字面字符重写。如果不指定的扩展的新版本，ALTER EXTENSION UPDATE 会更新到扩展的控制文件中显示的默认版本。

- **new_schema**

  扩展的新模式。

- **object_name**

  function_name

  从扩展里被添加或移除的对象的名称。包含表、聚合 、域、外链表、函数、操作符、操作符类、操作符族、序列、文本搜索对象、类型和能被模式合格的视图的名称。

- **argmode**

  这个函数参数的模型：IN、OUT、INOUT 或者 VARIADIC。如果省略的话，默认值为 IN。ALTER EXTENSION 不关心 OUT 参数，因为确认函数的一致性只需要输入参数，因此列出 IN、INOUT 和 VARIADIC 参数就足够了。

- **argname**

  函数参数的名称。ALTER EXTENSION 不关心参数名称，确认函数的一致性只需要参数数据类型。

- **argtype**

  函数参数的数据类型（可以有模式修饰）。

示例

1、创建扩展

```
create extension "uuid-ossp";
```

2、更新 "uuid-ossp" 扩展到版本 2.0：

```
ALTER EXTENSION "uuid-ossp" UPDATE TO '2.0';
```

3、创建模式 utils

```
CREATE SCHEMA utils;
```

4、更新 "uuid-ossp"扩展的模式为 utils：

```
ALTER EXTENSION "uuid-ossp" SET SCHEMA utils;
```

5、创建函数

```
create function length(p_one text, p_other text)
returns text
as
$$
select case
when length(p_one) >= length(p_other) then p_one
else p_other
end
$$
language sql
immutable;
```

6、添加 length 函数给 "uuid-ossp" 扩展：

```
ALTER EXTENSION "uuid-ossp" ADD FUNCTION length(p_one text, p_other text);
```

7、从"uuid-ossp" 扩展扩展删除 length 函数：

```
ALTER EXTENSION "uuid-ossp" DROP FUNCTION length(p_one text, p_other text);
```

4.5.10.ALTER FOREIGN DATA WRAPPER

功能描述

修改外部数据包装器的定义。

语法格式

```
ALTER FOREIGN DATA WRAPPER name
  [ HANDLER handler_function | NO HANDLER ]
  [ VALIDATOR validator_function | NO VALIDATOR ]
  [ OPTIONS ( [ ADD | SET | DROP ] option ['value'] [,...] ) ]
```

参数说明

- **name**

  要修改的外部数据包装器名。

- **handler_function**

为外部数据包装器指定一个新的处理器函数。

❖ **NO HANDLER**

指定外部数据包装器不再具有处理器函数。

【须知】

不能访问使用没有处理器的外部数据包装器的外部表。

❖ **validator_function**

为外部数据包装器指定一个新的验证器函数。

【须知】

在修改验证器函数后，外部数据包装器，服务器和用户映射的选项可能会失效。在使用外部数据包装器之前，用户应确保这些选项是正确的。

❖ **NO VALIDATOR**

指定外部数据包装器不再具有验证器函数。

❖ **OPTIONS ([ADD | SET | DROP] option ['value'] [,...])**

外部数据包装器的修改选项。添加，设置和删除指定要执行的操作。如果未明确指定操作，则假定添加。选项名称不许是唯一的；如果有的话，使用外部数据包装器的验证器函数验证名称和值。

示例

1、创建外部包装器 dbi。

```
CREATE FOREIGN DATA WRAPPER dbi OPTIONS (debug 'true');
```

2、修改外部包装器 dbi，添加选项 foo，删除选项 debug。

```
ALTER FOREIGN DATA WRAPPER dbi OPTIONS (ADD foo '1', DROP debug);
```

3、修改外部数据包装器 dbi 的验证器为 myvalidator。

```
ALTER FOREIGN DATA WRAPPER dbi VALIDATOR file_fdw_validator;
```

4.5.11.ALTER FOREIGN TABLE

功能描述

对外表进行修改。

语法格式

```
ALTER FOREIGN TABLE [ IF EXISTS ]  table_name
   OPTIONS ( {[ ADD | SET | DROP ] option ['value']}[, ... ]);
```

```
ALTER FOREIGN TABLE [ IF EXISTS ] table_name
  ALTER column_name OPTIONS;
```

参数说明

- **table_name**

  需要修改的外表名称。

  取值范围：已存在的外表名。

- **option**

  改变外表或者外表字段的选项。ADD、SET 和 DROP 指定执行的操作。如果没有显式设置，那么默认为 ADD。选项的名字不允许重复（尽管表选项和表字段选项可以有相同的名字）。选项的名称和值也会通过外部数据封装器的类库进行校验。

  - **oracle_fdw 支持的 options 包括**：
    - **table**：oracle server 侧的表名。需要同 oracle 系统表中记录的表名完全一致，通常是由大写字符组成。
    - **schema**：表所对应的 schema（或 owner）。需要与 oracle 系统表中记录的表名完全一致，通常是由大写字符组成。
  - mysql_fdw 支持的 options 包括：
    - **dbname**：MySQL 的 database 名称。
    - **table_name**：MySQL 侧的表名。
  - **postgres_fdw 支持的 options 包括：**
    - **schema_name**：远端 server 的 schema 名称。如果不指定的话，将使用外表自身的 schema 名称作为远端的 schema 名称。
    - **table_name**：远端 server 的表名。如果不指定的话，将使用外表自身的表名作为远端的表名。
    - **column_name**：远端 server 的表的列名。如果不指定的话，将使用外表自身的列名作为远端的的表的列名。
  - **file_fdw 支持的 options 包括：**
    - filename：指定要读取的文件，必需的参数，且必须是一个绝对路径名。

- ✧ format：远端 server 的文件格式，支持 text/csv/binary/fixed 四种格式，和 COPY 语句的 FORMAT 选项相同。
- ✧ header “” 指定的文件是否有标题行，与 COPY 语句的 HEADER 选项相同。

  -delimiter：指定文件的分隔符，与 COPY 的 DELIMITER 选项相同。

  -quote：指定文件的引用字符，与 COPY 的 QUOTE 选项相同。

  -escape：指定文件的转义字符，与 COPY 的 ESCAPE 选项相同。

  -null：指定文件的 null 字符串，与 COPY 的 NULL 选项相同。

  -encoding：指定文件的编码，与 COPY 的 ENCODING 选项相同。

  -force_not_null：这是一个布尔选项。如果为真，则声明字段的值不应该匹配空字符串（也就是，文件级别 null 选项）。与 COPY 的 FORCE_NOT_NULL 选项里的字段相同。

❖ **value**

option 的新值。

相关链接

CREATE FOREIGN TABLE(参见：开发者指南->SQL 语法参考->SQL 语法->CREATE FOREIGN TABLE 章节)，DROP FOREIGN TABLE(参见：开发者指南->SQL 语法参考->SQL 语法->DROP FOREIGN TABLE 章节)

4.5.12.ALTER FUNCTION

功能描述

修改自定义函数的属性。

注意事项

只有函数的所有者或者被授予了函数 ALTER 权限的用户才能执行 ALTER FUNCTION 命令，系统管理员默认拥有该权限。针对所要修改属性的不同，还有以下权限约束：

- 如果函数中涉及对临时表相关的操作，则无法使用 ALTER FUNCTION。
- 修改函数的所有者或修改函数的模式，当前用户必须是该函数的所有者或者系统管理员，且该用户是新所有者角色的成员。
- 只有系统管理员和初始化用户可以将 function 的 schema 修改成 public。
- 重命名函数时，不能与当前模式下已存在的 synonym 产生命名冲突。
- 修改函数的模式时，不能与新模式下已存在的 synonym 产生命名冲突。
- 仅有初始化用户或者创建该存储过程的用户可以修改存储过程为定义者权限的存储过程。
- 打开三权分立时，即使是 sysadmin 权限用户，也需要校验用户的组权限。

语法格式

- 修改自定义函数的附加参数。

```
ALTER FUNCTION function_name ( [ { [ argname ] [ argmode ] argtype} [, ...] ] )
  action [ ... ] [ RESTRICT ];
```

其中附加参数 action 子句语法为。

```
{CALLED ON NULL INPUT | RETURNS NULL ON NULL INPUT | STRICT}
 | {IMMUTABLE | STABLE | VOLATILE}
 | {NOT FENCED | FENCED}
 | [ NOT ] LEAKPROOF
 | {[ EXTERNAL ] SECURITY INVOKER | [ EXTERNAL ] SECURITY DEFINER}
 | AUTHID { DEFINER | CURRENT_USER }
 | COST execution_cost
 | ROWS result_rows
 | SET configuration_parameter {{ TO | = } { value | DEFAULT }| FROM CURRENT}
 | RESET {configuration_parameter| ALL}
```

- 修改自定义函数的名称。

```
ALTER FUNCTION funname ( [ { [ argname ] [ argmode ] argtype} [, ...] ] )
 RENAME TO new_name;
```

❖ 修改自定义函数的所属者。

```
ALTER FUNCTION funname ( [ { [ argname ] [ argmode ] argtype} [, ...] ] )
OWNER TO new_owner;
```

❖ 修改自定义函数的模式。

```
ALTER FUNCTION funname ( [ { [ argname ] [ argmode ] argtype} [, ...] ] )
SET SCHEMA new_schema;
```

参数说明

❖ **function_name**

要修改的函数名称。

取值范围：已存在的函数名。

❖ **argmode**

标识该参数是输入、输出参数。

取值范围：IN/OUT/INOUT/VARIADIC。

❖ **argname**

参数名称。

取值范围：字符串，符合标识符命名规范。

❖ **argtype**

函数参数的类型。

❖ **CALLED ON NULL INPUT**

表明该函数的某些参数是 NULL 的时候可以按照正常的方式调用。缺省时与指定此参数的作用相同。

❖ **RETURNS NULL ON NULL INPUT**

STRICT

STRICT 用于指定如果函数的某个参数是 NULL，此函数总是返回 NULL。如果声明了这个参数，则如果存在 NULL 参数时不会执行该函数；而只是自动假设一个 NULL 结果。

RETURNS NULL ON NULL INPUT 和 STRICT 的功能相同。

❖ **IMMUTABLE**

表示该函数在给出同样的参数值时总是返回同样的结果。

❖ **STABLE**

表示该函数不能修改数据库，对相同参数值，在同一次表扫描里，该函数的返回值不变，但是返回值可能在不同 SQL 语句之间变化。

- **VOLATILE**

表示该函数值可以在一次表扫描内改变，不会做任何优化。

- **LEAKPROOF**

表示该函数没有副作用，指出参数只包括返回值。LEAKPROOF 只能由系统管理员设置。

- **EXTERNAL**

（可选）目的是和 SQL 兼容，这个特性适合于所有函数，而不仅是外部函数。

- **SECURITY INVOKER**

AUTHID CURRENT_USER

表明该函数将以调用它的用户的权限执行。缺省时与指定此参数的作用相同。

SECURITY INVOKER 和 AUTHID CURRENT_USER 的功能相同。

- **SECURITY DEFINER**

AUTHID DEFINER

声明该函数将以创建它的用户的权限执行。

AUTHID DEFINER 和 SECURITY DEFINER 的功能相同。

- **COST execution_cost**

用来估计函数的执行成本。

execution_cost 以 cpu_operator_cost 为单位。

取值范围：正数

- **ROWS result_rows**

估计函数返回的行数。用于函数返回的是一个集合。

取值范围：正数，默认值是 1000 行。

- **configuration_parameter**
  - **value**

    把指定的数据库会话参数值设置为给定的值。如果 value 是 DEFAULT 或者 RESET，则在新的会话中使用系统的缺省设置。

    OFF 关闭设置。

    取值范围：字符串

- DEFAULT
- OFF
- RESET

  指定默认值。

  - **from current**

    取当前会话中的值设置为 configuration_parameter 的值。

- **new_name**

  函数的新名称。要修改函数的所属模式，必须拥有新模式的 CREATE 权限。

  取值范围：字符串，符合标识符命名规范。

- **new_owner**

  函数的新所有者。要修改函数的所有者，新所有者必须拥有该函数所属模式的 CREATE 权限。注意：仅有数据库初始化用户才可将函数的 owner 设置为初始化用户。

  取值范围：已存在的用户角色。

- **new_schema**

  函数的新模式。

  取值范围：已存在的模式。

示例

请参见 CREATE FUNCTION 的示例。

相关链接

CREATE FUNCTION (参见：开发者指南->SQL 语法参考->SQL 语法->CREATE FUNCTION 章节)，DROP FUNCTION (参见：开发者指南->SQL 语法参考->SQL 语法->DROP FUNCTION 章节)

4.5.13.ALTER GLOBAL CONFIGURATION

功能描述

新增、修改系统表 gs_global_config，增加 key-value 值。

注意事项

- 仅支持数据库初始用户运行此命令。
- 不支持创建修改关键字为 weak_password。

语法格式

```
ALTER GLOBAL CONFIGURATION with(paraname=value,paraname=value...);
```

参数说明

- **paraname**

  参数名称，text 类型。

- **value**

  参数值，text 类型。

4.5.14.ALTER GROUP

功能描述

修改一个用户组的属性。

注意事项

ALTER GROUP 是 ALTER ROLE 的别名，非 SQL 标准语法，不推荐使用，建议用户直接使用 ALTER ROLE 替代。

语法格式

- 向用户组中添加用户。

```
ALTER GROUP group_name
  ADD USER user_name [, ... ];
```

- 从用户组中删除用户。

```
ALTER GROUP group_name
  DROP USER user_name [, ... ];
```

- 修改用户组的名称。

```
ALTER GROUP group_name
  RENAME TO new_name;
```

参数说明

请参见：开发者指南->SQL 语法参考->SQL 语法->ALTER ROLE 的参数说明。

示例

```
--创建用户 lche、jim
CREATE USER lche PASSWORD 'Aa@12345';
CREATE USER jim PASSWORD 'Bb@12345';

--创建 GROUP
CREATE GROUP super_users IDENTIFIED BY 'Aa@12345';

--向用户组中添加用户。
ALTER GROUP super_users ADD USER lche, jim;

--从用户组中删除用户。
ALTER GROUP super_users DROP USER jim;

--修改用户组的名称。
ALTER GROUP super_users RENAME TO normal_users;
```

相关链接

ALTER GROUP(参见：开发者指南->SQL 语法参考->SQL 语法->ALTER GROUP 章节)，DROP GROUP(参见：开发者指南->SQL 语法参考->SQL 语法->DROP GROUP 章节)，ALTER ROLE(参见：开发者指南->SQL 语法参考->SQL 语法->ALTER ROLE 章节)

4.5.15.ALTER INDEX

功能描述

ALTER INDEX 用于修改现有索引的定义。

它有几种子形式：

- **IF EXISTS**

  如果指定的索引不存在，则发出一个 notice 而不是 error。

- **RENAME TO**

只改变索引的名称。对存储的数据没有影响。

- **SET TABLESPACE**

  这个选项会改变索引的表空间为指定表空间，并且把索引相关的数据文件移动到新的表空间里。

- **SET ({ STORAGE_PARAMETER = value } [, ...])**

  改变索引的一个或多个索引方法特定的存储参数。 需要注意的是索引内容不会被这个命令立即修改，根据参数的不同，可能需要使用 REINDEX 重建索引来获得期望的效果。

- **RESET ({ storage_parameter } [, ...])**

  重置索引的一个或多个索引方法特定的存储参数为缺省值。与 SET 一样，可能需要使用 REINDEX 来完全更新索引。

- **[MODIFY PARTITION index_partition_name] UNUSABLE**

  用于设置表或者索引分区上的索引不可用。

- **REBUILD [PARTITION index_partition_name]**

  用于重建表或者索引分区上的索引。

- **RENAME PARTITION**

  用于重命名索引分区。

- **MOVE PARTITION**

  用于修改索引分区的所属表空间。

注意事项

索引的所有者，拥有索引所在表的 INDEX 权限的用户，或者被授予了 ALTER ANY INDEX 权限的用户有权限执行此命令，系统管理员默认拥有此权限。

语法格式

- 重命名表索引的名称。

```
ALTER INDEX [ IF EXISTS ] index_name
```

```
    RENAME TO new_name;
```

❖ 修改表索引的所属空间。

```
ALTER INDEX [ IF EXISTS ] index_name
    SET TABLESPACE tablespace_name;
```

❖ 修改表索引的存储参数。

```
ALTER INDEX [ IF EXISTS ] index_name
    SET ( {storage_parameter = value} [, ... ] );
```

❖ 重置表索引的存储参数。

```
ALTER INDEX [ IF EXISTS ] index_name
    RESET ( storage_parameter [, ... ] ) ;
```

❖ 设置表索引或索引分区不可用。

```
ALTER INDEX [ IF EXISTS ] index_name
    [ MODIFY PARTITION index_partition_name ] UNUSABLE;
```

【须知】

列存表不支持该语法。

❖ 重建表索引或索引分区。

```
ALTER INDEX index_name
    REBUILD [ PARTITION index_partition_name ];
```

❖ 重命名索引分区。

```
ALTER INDEX [ IF EXISTS ] index_name
  RENAME PARTITION index_partition_name TO new_index_partition_name;
```

❖ 修改索引分区的所属表空间。

```
ALTER INDEX [ IF EXISTS ] index_name
    MOVE PARTITION index_partition_name TABLESPACE new_tablespace;
```

- 在 PostgreSQL 兼容模式下，支持使用 ALTER INDEX ATTACH PARTITION 语法把索引加入到分区索引中。详见《PanWeiDB V2.0-S3.0.1_B01 兼容性手册》-> PostgreSQL 兼容性->ALTER INDEX。

```
ALTER INDEX [ IF EXISTS ] index_name ATTACH PARTITION index_name
```

参数说明

- **index_name**

  要修改的索引名。

- **new_name**

  新的索引名。

  取值范围：字符串，且符合标识符命名规范。

- **tablespace_name**

  表空间的名称。

  取值范围：已存在的表空间。

- **storage_parameter**

  索引方法特定的参数名。

- **value**

  索引方法特定的存储参数的新值。根据参数的不同，这可能是一个数字或单词。

- **new_index_partition_name**

  新索引分区名。

- **index_partition_name**

  索引分区名。

- **new_tablespace**

  新表空间。

示例

请参考 CREATE INDEX 的示例。

4.5.16.ALTER LANGUAGE

功能描述

修改一个过程语言的定义。单机和集中式暂不支持创建过程语言，所以暂不支持修改过程语言。

语法格式

```
ALTER [ PROCEDURAL ] LANGUAGE name RENAME TO new_name ALTER [ PROCEDURAL ] LANGUAGE name OWNER TO new_owner
```

参数说明

- **name**

  语言的名字。

- **new_name**

  语言的新名字。

- **new_owner**

  语言的新的所有者。

兼容性

SQL 标准里没有 ALTER LANGUAGE 语句。

4.5.17.ALTER LARGE OBJECT

功能描述

ALTER LARGE OBJECT 用于更改一个 large object 的定义。它的唯一的功能是分配一个新的所有者。

注意事项

使用 ALTER LARGE OBJECT 必须是系统管理员或者是其所有者。

语法格式

```
ALTER LARGE OBJECT large_object_oid
  OWNER TO new_owner;
```

参数说明

- **large_object_oid**

  要被变 large object 的 OID 。

  取值范围：已存在的大对象名。

- **OWNER TO new_owner**

  large object 新的所有者。

  取值范围：已存在的用户名/角色名。

示例

1、创建 LARGE OBJECT。

```
SELECT pg_catalog.lo_create('100001');
```

2、将 LARGE OBJECT 所有者改为 panweidb。

```
ALTER LARGE OBJECT 100001 OWNER TO panweidb;
```

4.5.18.ALTER MASKING POLICY

功能描述

修改脱敏策略。

注意事项

- 只有 poladmin、sysadmin 或初始用户才能执行此操作。
- 需要打开 enable_security_policy 开关脱敏策略才可以生效。

语法格式

- 修改策略描述：

```
ALTER MASKING POLICY policy_name COMMENTS policy_comments;
```

- 修改脱敏方式：

```
ALTER MASKING POLICY policy_name [ADD | REMOVE | MODIFY] masking_actions[, ...]
*;
```

```
其中 masking_action:
  masking_function ON LABEL(label_name[, ...]*)
```

❖ 修改脱敏策略生效场景：

```
ALTER MASKING POLICY policy_name MODIFY(FILTER ON FILTER_TYPE(filter_value[, ...]*)[, ...]*);
```

❖ 移除脱敏策略生效场景，使策略对所用场景生效：

```
ALTER MASKING POLICY policy_name DROP FILTER;
```

❖ 修改脱敏策略开启/关闭：

```
ALTER MASKING POLICY policy_name [ENABLE | DISABLE];
```

参数说明

❖ **policy_name**

脱敏策略名称，需要唯一，不可重复。

取值范围：字符串，要符合标识符的命名规范。

❖ **policy_comments**

需要为脱敏策略添加或修改的描述信息。

❖ **masking_function**

指的是预置的八种脱敏方式或者用户自定义的函数，支持模式。

maskall 不是预置函数，硬编码在代码中，不支持 df 展示。

预置时脱敏方式如下：

```
maskall | randommasking | creditcardmasking | basicemailmasking | fullemailmasking | shufflemasking | alldigitsmasking | regexpmasking
```

❖ **label_name**

资源标签名称。

❖ **FILTER_TYPE**

指定脱敏策略的过滤信息，过滤类型包括：IP、ROLES、APP。

❖ **filter_value**

指具体过滤信息内容，例如具体的 IP、具体的 APP 名称、具体的用户名。

❖ **ENABLE|DISABLE**

可以打开或关闭脱敏策略。若不指定 ENABLE|DISABLE，语句默认为 ENABLE。

示例

```
--创建 dev_mask 和 bob_mask 用户。
panweidb=# CREATE USER dev_mask PASSWORD 'dev@1234';
panweidb=# CREATE USER bob_mask PASSWORD 'bob@1234';

--创建一个表 tb_for_masking
panweidb=# CREATE TABLE tb_for_masking(col1 text, col2 text, col3 text);

--创建资源标签标记敏感列 col1
panweidb=# CREATE RESOURCE LABEL mask_lb1 ADD COLUMN(tb_for_masking.col
1);

--创建资源标签标记敏感列 col2
panweidb=# CREATE RESOURCE LABEL mask_lb2 ADD COLUMN(tb_for_masking.col
2);

--对访问敏感列 col1 的操作创建脱敏策略
panweidb=# CREATE MASKING POLICY maskpol1 maskall ON LABEL(mask_lb1);

--为脱敏策略 maskpol1 添加描述
panweidb=# ALTER MASKING POLICY maskpol1 COMMENTS 'masking policy for tb_f
or_masking.col1';

--修改脱敏策略 maskpol1，新增一项脱敏方式
panweidb=# ALTER MASKING POLICY maskpol1 ADD randommasking ON LABEL(ma
sk_lb2);

--修改脱敏策略 maskpol1，移除一项脱敏方式
panweidb=# ALTER MASKING POLICY maskpol1 REMOVE randommasking ON LABEL
(mask_lb2);

--修改脱敏策略 maskpol1，修改一项脱敏方式
panweidb=# ALTER MASKING POLICY maskpol1 MODIFY randommasking ON LABEL
(mask_lb1);
```

```
--修改脱敏策略 maskpol1 使之仅对用户 dev_mask 和 bob_mask，客户端工具为 psql 和 gsql，IP 地址为'10.20.30.40', '127.0.0.0/24'场景生效。
panweidb=# ALTER MASKING POLICY maskpol1 MODIFY (FILTER ON ROLES(dev_mask, bob_mask), APP(psql, gsql), IP('10.20.30.40', '127.0.0.0/24'));

--修改脱敏策略 maskpol1，使之对所有用户场景生效
panweidb=# ALTER MASKING POLICY maskpol1 DROP FILTER;

--禁用脱敏策略 maskpol1
panweidb=# ALTER MASKING POLICY maskpol1 DISABLE;
```

相关链接

CREATE MASKING POLICY (参见：开发者指南->SQL 语法参考->SQL 语法->CREATE MASKING POLICY 章节),DROP MASKING POLICY(参见：开发者指南->SQL 语法参考->SQL 语法->DROP MASKING POLICY 章节)

4.5.19.ALTER MATERIALIZED VIEW

功能描述

更改一个现有物化视图的多个辅助属性。

可用于 ALTER MATERIALIZED VIEW 的语句形式和动作是 ALTER TABLE 的一个子集，并且在用于物化视图时具有相同的含义。详见开发者指南->SQL 语法参考->SQL 语法->ALTER TABLE 章节。

注意事项

- 只有物化视图的所有者有权限执行 ALTER TMATERIALIZED VIEW 命令，系统管理员默认拥有此权限。
- 不支持更改物化视图结构。

语法格式

- 修改物化视图的所属用户。

```
ALTER MATERIALIZED VIEW [ IF EXISTS ] mv_name
  OWNER TO new_owner;
```

❖ 修改物化视图的列。

```
ALTER MATERIALIZED VIEW [ IF EXISTS ] mv_name
  RENAME [ COLUMN ] column_name TO new_column_name;
```

❖ 重命名物化视图。

```
ALTER MATERIALIZED VIEW [ IF EXISTS ] mv_name

  RENAME TO new_name;
```

参数说明

❖ **mv_name**

一个现有物化视图的名称，可以用模式修饰。

取值范围：字符串，符合标识符命名规范。

❖ **column_name**

一个新的或者现有的列的名称。

取值范围：字符串，符合标识符命名规范。

❖ **new_column_name**

一个现有列的新名称。

❖ **new_owner**

该物化视图的新拥有者的用户名。

❖ **new_name**

该物化视图的新名称。

示例

```
--把物化视图 foo 重命名为 bar。
panweidb=# ALTER MATERIALIZED VIEW foo RENAME TO bar;
```

相关链接

CREATE MATERIALIZED VIEW (参见：开发者指南->SQL 语法参考->SQL 语法->CREATE MATERIALIZED VIEW 章节)、CREATE INCREMENTAL MATERIALIZED VIEW (参见：开发者指南->SQL 语法参考->SQL 语法->CREATE INCREMENTAL MATERIALIZED VIEW 章节)、DROP MATERIALIZED VIEW (参见：开发者指南->SQL 语法参考->SQL 语法->DROP MATERIALIZED VIEW 章节)、REFRESH INCREMENTAL MATERIALIZED VIEW(参见：开发者指

南->SQL 语法参考->SQL 语法->REFRESH INCREMENTAL MATERIALIZED VIEW 章节）、REFRESH MATERIALIZED VIEW (参见：开发者指南->SQL 语法参考->SQL 语法->REFRESH MATERIALIZED VIEW 章节)

4.5.20.ALTER OPERATOR

功能描述

修改一个操作符的定义。

注意事项

ALTER OPERATOR 改变一个操作符的定义。目前唯一能用的功能是改变操作符的所有者。

要使用 ALTER OPERATOR，你必须是该操作符的所有者。要修改所有者，你还必须是新的所有角色的直接或间接成员，并且该成员必须在此操作符的模式上有 CREATE 权限。（这些限制强制了修改该所有者不会做任何通过删除和重建操作符不能做的事情。不过，具有 SYSADMIN 权限用户可以以任何方式修改任意操作符的所有权。）

语法格式

```
ALTER OPERATOR name ( { left_type | NONE } , { right_type | NONE } ) OWNER TO new_owner
ALTER OPERATOR name ( { left_type | NONE } , { right_type | NONE } ) SET SCHEMA new_schema
```

参数说明

- **name**

  一个现有操作符的名字。

- **left_type**

  操作符的左操作数的数据类型；如果没有左操作数，那么写 NONE。

- **right_type**

  操作符的右操作数的数据类型；如果没有右操作数，那么写 NONE。

- **new_owner**

  操作符的新所有者。

- **new_schema**

  操作符的新模式名。

示例

改变一个用于 text 的用户定义操作符 a @@ b：

```
ALTER OPERATOR @@ (text, text) OWNER TO joe;
```

兼容性

SQL 标准里没有 ALTER OPERATOR 语句。

4.5.21.ALTER PACKAGE

功能描述

修改 PACKAGE 的属性。

注意事项

目前仅支持 ALTER PACKAGE OWNER 功能，系统管理员默认拥有该权限，有以下权限约束：

- 当前用户必须是该 PACKAGE 的所有者或者系统管理员，且该用户是新所有者角色的成员。
- 仅有初始化用户可以修改定义者权限的 package 的 owner。

语法格式

修改 PACKAGE 的所属者。

```
ALTER PACKAGE package_name OWNER TO new_owner;
```

参数说明

- **package_name**

要修改的 PACKAGE 名称。

取值范围：已存在的 PACKAGE 名，仅支持修改单个 PACKAGE。

❖ **new_owner**

PACKAGE 的新所有者。要修改函数的所有者，新所有者必须拥有该 PACKAGE 所属模式的 CREATE 权限。

取值范围：已存在的用户角色。

示例

请参见 CREATE PACKAGE 中示例。（参见：开发者指南->SQL 语法参考->SQL 语法->CREATE PACKAGE 章节）

4.5.22.ALTER PUBLICATION

功能描述

更改发布 PUBLICATION 的属性。

注意事项

发布的属主和系统管理员才能执行 ALTER PUBLICATION。新所有者角色的直接或间接成员才可以改变所有者。新的所有者必须在当前数据库上拥有 CREATE 权限。此外，FOR ALL TABLES 发布的新所有者必须是系统管理员。但是，系统管理员可以在避开这些限制的情况下更改发布的所有权。

语法格式

❖ 用指定的表替换当前发布的表。

```
ALTER PUBLICATION name SET TABLE table_name [, ...]
```

❖ 从发布中添加一个或多个表。

```
ALTER PUBLICATION name ADD TABLE table_name [, ...]
```

❖ 从发布中删除一个或多个表。

```
ALTER PUBLICATION name DROP TABLE table_name [, ...]
```

❖ 改变在 CREATE PUBLICATION 中指定的所有发布属性，未提及的属性保留其之前的设置。

```
ALTER PUBLICATION name SET ( publication_parameter [= value] [, ... ] )
```

❖ 更改发布的所有者。

```
ALTER PUBLICATION name OWNER TO { new_owner | CURRENT_USER | SESSION_USE
R }
```

❖ 更改发布的名称。

```
ALTER PUBLICATION name RENAME TO new_name
```

参数说明

❖ **name**

待修改的发布的名称。

❖ **table_name**

现有表的名称。

❖ **SET (publication_parameter [= value] [, ...])。**

该子句修改最初由 CREATE PUBLICATION 设置的发布参数。

❖ **new_owner**

发布的新所有者的用户名。

❖ **new_name**

发布的新名称。

示例

详情请参见 CREATE-PUBLICATION 的示例。

相关链接

CREATE PUBLICATION （参见：开发者指南->SQL 语法参考->SQL 语法->CREATE PUBLICATION 章节），DROP PUBLICATION （参见：开发者指南->SQL 语法参考->SQL 语法->DROP PUBLICATION 章节）。

4.5.23.ALTER RESOURCE LABEL

功能描述

修改资源标签。

注意事项

只有 poladmin、 sysadmin 或初始用户才能执行此操作。

语法格式

```
ALTER RESOURCE LABEL label_name (ADD|REMOVE)
 label_item_list[, ...]*;
```

- label_item_list:

```
resource_type(resource_path[, ...]*)
```

- resource_type:

```
TABLE | COLUMN | SCHEMA | VIEW | FUNCTION
```

参数说明

- **label_name**

  资源标签名称。

  取值范围：字符串，要符合标识符的命名规范。

- **resource_type**

  指的是要标记的数据库资源类型。

- **resource_path**

  指的是描述具体的数据库资源的路径。

示例

```
--创建基本表 table_for_label。
panweidb=# CREATE TABLE table_for_label(col1 int, col2 text);

--创建资源标签 table_label。
panweidb=# CREATE RESOURCE LABEL table_label ADD COLUMN(table_for_label.co
l1);

--将 col2 添加至资源标签 table_label 中
panweidb=# ALTER RESOURCE LABEL table_label ADD COLUMN(table_for_label.col
2);

--将资源标签 table_label 中的一项移除
panweidb=# ALTER RESOURCE LABEL table_label REMOVE COLUMN(table_for_label.
col1);
```

相关链接

CREATE RESOURCE LABEL(参见：开发者指南->SQL 语法参考->SQL 语法->CREATE RESOURCE LABEL 章节)，DROP RESOURCE LABEL(参见：开发者指南->SQL 语法参考->SQL 语法->DROP RESOURCE LABEL 章节)。

4.5.24.ALTER ROLE

功能描述

修改角色属性。

语法格式

❖ 修改角色的权限。

```
ALTER ROLE role_name [ [ WITH ] option [ ... ] ];
```

其中权限项子句 option 为：

```
{CREATEDB | NOCREATEDB}
    | {CREATEROLE | NOCREATEROLE}
    | {INHERIT | NOINHERIT}
    | {AUDITADMIN | NOAUDITADMIN}
    | {SYSADMIN | NOSYSADMIN}
    | {MONADMIN | NOMONADMIN}
    | {OPRADMIN | NOOPRADMIN}
    | {POLADMIN | NOPOLADMIN}
    | {USEFT | NOUSEFT}
    | {LOGIN | NOLOGIN}
    | {REPLICATION | NOREPLICATION}
    | {INDEPENDENT | NOINDEPENDENT}
    | {VCADMIN | NOVCADMIN}
    | {PERSISTENCE | NOPERSISTENCE}
    | CONNECTION LIMIT connlimit
    | [ ENCRYPTED | UNENCRYPTED ] PASSWORD 'password' [EXPIRED]
    | [ ENCRYPTED | UNENCRYPTED ] IDENTIFIED BY 'password' [ REPLACE 'old_passw
ord' | EXPIRED ]
    | VALID BEGIN 'timestamp'
    | VALID UNTIL 'timestamp'
    | RESOURCE POOL 'respool'
```

```
    | USER GROUP 'groupuser'
    | PERM SPACE 'spacelimit'
    | TEMP SPACE 'tmpspacelimit'
    | SPILL SPACE 'spillspacelimit'
    | NODE GROUP logic_cluster_name
    | ACCOUNT { LOCK | UNLOCK }
    | PGUSER
```

❖ 修改角色的名称。

```
ALTER ROLE role_name
  RENAME TO new_name;
```

❖ 锁定或解锁。

```
ALTER ROLE role_name
  ACCOUNT { LOCK | UNLOCK };
```

❖ 设置角色的配置参数。

```
ALTER ROLE role_name [ IN DATABASE database_name ]
 SET configuration_parameter {{ TO | = } { value | DEFAULT } | FROM CURRENT};
```

❖ 重置角色的配置参数。

```
ALTER ROLE role_name
  [ IN DATABASE database_name ] RESET {configuration_parameter|ALL};
```

参数说明

❖ **role_name**

现有角色名。

取值范围：已存在的用户名。

❖ **IN DATABASE database_name**

表示修改角色在指定数据库上的参数。

❖ **SET configuration_parameter**

设置角色的参数。ALTER ROLE 中修改的会话参数只针对指定的角色，且在下一次该角色启动的会话中有效。

取值范围：configuration_parameter 和 value 的取值请参见：开发者指南->SQL 语法参考->SQL 语法->SET 章节。

- DEFAULT：表示清除 configuration_parameter 参数的值，configuration_parameter 参数的值将继承本角色新产生的 SESSION 的默认值。
- FROM CURRENT：取当前会话中的值设置为 configuration_parameter 参数的值。

❖ **RESET configuration_parameter/ALL**

清除 configuration_parameter 参数的值。与 SETconfiguration_parameter TO DEFAULT 的效果相同。

取值范围：ALL 表示清除所有参数的值。

❖ **ACCOUNT LOCK | ACCOUNT UNLOCK**

- ACCOUNT LOCK：锁定帐户，禁止登录数据库。
- ACCOUNT UNLOCK：解锁帐户，允许登录数据库。

❖ **PGUSER**

当前版本不允许修改角色的 PGUSER 属性。

❖ **PASSWORD/IDENTIFIED BY 'password'**

重置或修改用户密码。除了初始用户外其他管理员或普通用户修改自己的密码需要输入正确的旧密码。只有初始用户、系统管理员（sysadmin）或拥有创建用户（CREATEROLE）权限的用户才可以重置普通用户密码，无需输入旧密码。初始用户可以重置系统管理员的密码， 系统管理员不允许重置其他系统管理员的密码。

❖ **EXPIRED**

设置密码失效。只有初始用户、系统管理员（sysadmin）或拥有创建用户（CREATEROLE）权限的用户才可以设置用户密码失效，其中系统管理员也可以设置自己或其他系统管理员密码失效。任何用户都不允许设置初始用户密码失效。密码失效的用户可以登录数据库但不能执行查询操作，只有 修改密码或由管理员重置密码后才可以恢复正常查询操作。

其他参数请参见“CREATE ROLE”的参数说明部分。

注意事项

只允许修改用户的标识，但是不允许修改属性标识。

示例

❖ 修改角色 manager 的密码为 abcd@123。

```
ALTER ROLE manager IDENTIFIED BY 'abcd@123' REPLACE 'Bigdata@123';
```

❖ 修改角色 manager 为系统管理员。

```
ALTER ROLE manager SYSADMIN;
```

4.5.25.ALTER ROW LEVEL SECURITY POLICY

功能描述

对已存在的行访问控制策略（包括行访问控制策略的名称、行访问控制指定的用户、行访问控制的策略表达式）进行修改。

注意事项

表的所有者或管理员用户才能进行此操作。

语法格式

```
ALTER [ ROW LEVEL SECURITY ] POLICY [ IF EXISTS ] policy_name ON table_name REN
AME TO new_policy_name;

ALTER [ ROW LEVEL SECURITY ] POLICY policy_name ON table_name
  [ TO { role_name | PUBLIC } [, ...] ]
  [ USING ( using_expression ) ];
```

参数说明

❖ **policy_name**

行访问控制策略名称。

❖ **table_name**

行访问控制策略的表名。

❖ **new_policy_name**

新的行访问控制策略名称。

❖ **role_name**

行访问控制策略应用的数据库用户，可以指定多个用户，PUBLIC 表示应用到所有用户。

❖ **using_expression**

行访问控制的表达式，返回值为 boolean 类型。

示例

```
--创建数据表 all_data
panweidb=# CREATE TABLE all_data(id int, role varchar(100), data varchar(100));

--创建行访问控制策略，当前用户只能查看用户自身的数据
panweidb=# CREATE ROW LEVEL SECURITY POLICY all_data_rls ON all_data USING(r
ole = CURRENT_USER);
panweidb=# \d+ all_data
                    Table "public.all_data"
 Column |        Type         | Modifiers | Storage  | Stats target | Description
--------+----------------------+-----------+----------+--------------+-------------
 id     | integer              |           | plain    |              |
 role   | character varying(100) |         | extended |              |
 data   | character varying(100) |         | extended |              |
Row Level Security Policies:
   POLICY "all_data_rls" FOR ALL
     TO public
     USING (((role)::name = "current_user"()))
Has OIDs: no
Options: orientation=row, compression=no

--修改行访问控制 all_data_rls 的名称
panweidb=# ALTER ROW LEVEL SECURITY POLICY all_data_rls ON all_data RENAME T
O all_data_new_rls;

--修改行访问控制策略影响的用户
panweidb=# ALTER ROW LEVEL SECURITY POLICY all_data_new_rls ON all_data TO a
lice, bob;
panweidb=# \d+ all_data
                    Table "public.all_data"
 Column |        Type         | Modifiers | Storage  | Stats target | Description
--------+----------------------+-----------+----------+--------------+-------------
```

```
 id    | integer               |         | plain    |              |
 role  | character varying(100) |         | extended |              |
 data  | character varying(100) |         | extended |              |
Row Level Security Policies:
   POLICY "all_data_new_rls" FOR ALL
     TO alice,bob
     USING (((role)::name = "current_user"()))
Has OIDs: no
Options: orientation=row, compression=no

--修改行访问控制策略表达式
panweidb=# ALTER ROW LEVEL SECURITY POLICY all_data_new_rls ON all_data USI
NG (id > 100 AND role = current_user);
panweidb=# \d+ all_data
                        Table "public.all_data"
 Column |         Type          | Modifiers | Storage  | Stats target | Description
--------+-----------------------+-----------+----------+--------------+-------------
 id     | integer               |         | plain    |              |
 role   | character varying(100) |         | extended |              |
 data   | character varying(100) |         | extended |              |
Row Level Security Policies:
   POLICY "all_data_new_rls" FOR ALL
     TO alice,bob
     USING (((id > 100) AND ((role)::name = "current_user"())))
Has OIDs: no
Options: orientation=row, compression=no
```

相关链接

CREATE ROW LEVEL SECURITY POLICY(参见：开发者指南->SQL 语法参考->SQL 语法->CREATE ROW LEVEL SECURITY POLICY 章节)，DROP ROW LEVEL SECURITY POLICY(参见：开发者指南->SQL 语法参考->SQL 语法->DROP ROW LEVEL SECURITY POLICY 章节)

4.5.26.ALTER PROCEDURE

功能描述

修改自定义存储过程的属性。

注意事项

只有存储过程的所有者或者被授予了存储过程 ALTER 权限的用户才能执行 ALTER PROCEDURE 命令，系统管理员默认拥有该权限。针对所要修改属性的不同，还有以下权限约束：

- 如果存储过程中涉及对临时表相关的操作，则无法使用 ALTER PROCEDURE。
- 修改存储过程的所有者或修改存储过程的模式，当前用户必须是该存储过程的所有者或者系统管理员，且该用户是新所有者角色的成员。
- 只有系统管理员和初始化用户可以将 procedure 的 schema 修改成 public。
- 重命名存储过程时，不能与当前模式下已经存在的 synonym 产生命名冲突。
- 修改存储过程的模式时，不能与新模式下已经存在的 synonym 产生命名冲突。

语法格式

- 修改自定义存储过程的附加参数。

```
ALTER PROCEDURE procedure_name ( [ { [ argname ] [ argmode ] argtype} [, ...] ] )
    action [ ... ] [ RESTRICT ];
```

其中自定义存储过程的附加参数 action 子句语法为：

```
 [ EXTERNAL ] SECURITY INVOKER | [ EXTERNAL ] SECURITY DEFINER
SET configuration_parameter { TO | = } { value | DEFAULT }
SET configuration_parameter FROM CURRENT
RESET configuration_parameter
RESET ALL
```

- 修改自定义存储过程的名称。

```
ALTER PROCEDURE proname ( [ { [ argname ] [ argmode ] argtype} [, ...] ] )
    RENAME TO new_name;
```

❖ 修改自定义存储过程的所属者。

```
ALTER PROCEDURE proname ( [ { [ argname ] [ argmode ] argtype} [, ...] ] )
    OWNER TO new_owner;
```

❖ 修改自定义存储过程的模式。

```
ALTER PROCEDURE proname ( [ { [ argname ] [ argmode ] argtype} [, ...] ] )
    SET SCHEMA new_schema;
```

❖ 修改自定义存储过程的插件。

```
ALTER PROCEDURE name [ ( [ [ argmode ] [ argname ] argtype [, ...] ] ) ]
  [ NO ] DEPENDS ON EXTENSION extension_name;
```

参数说明

❖ **procedure_name**

要修改的存储过程名称。

取值范围：已存在的存储过程名。

❖ **argmode**

标识该参数是输入、输出参数。

取值范围：IN/OUT/INOUT/VARIADIC。

❖ **argname**

参数名称。

取值范围：字符串，符合标识符命名规范。

❖ **argtype**

存储过程参数的类型。

❖ **new_name**

存储过程的新名称。要修改存储过程的所属模式，必须拥有新模式的 CREATE 权限。

取值范围：字符串，符合标识符命名规范。

❖ **new_owner**

存储过程的新所有者。要修改存储过程的所有者，新所有者必须拥有该存储过程所属模式的 CREATE 权限。

取值范围：已存在的用户角色。

- **new_schema**

  存储过程的新模式。

  取值范围：已存在的模式。

- **extension_name**

  已安装扩展的名称。

示例

1、定义存储过程为 SQL 查询。

```
CREATE PROCEDURE func_add_sql(integer, integer) RETURNS integer
  AS 'select $1 + $2;'
  LANGUAGE SQL
  IMMUTABLE
  RETURNS NULL ON NULL INPUT;
```

2、利用参数名用 PL/pgSQL 自增一个整数。

```
CREATE OR REPLACE PROCEDURE func_increment_plsql(i integer) RETURNS integer
AS $$
    BEGIN
        RETURN i + 1;
    END;
$$ LANGUAGE plpgsql;
```

3、返回 RECORD 类型

```
CREATE OR REPLACE PROCEDURE func_increment_sql(i int, out result_1 bigint, out re
sult_2 bigint)
returns SETOF RECORD
as $$
begin
  result_1 = i + 1;
  result_2 = i * 10;
return next;
end;
$$language plpgsql;
```

4、返回一个包含多个输出参数的记录。

```
CREATE PROCEDURE func_dup_sql(in int, out f1 int, out f2 text)
   AS $$ SELECT $1, CAST($1 AS text) || ' is text' $$
   LANGUAGE SQL;

SELECT * FROM func_dup_sql(42);
```

返回结果为：

```
 f1 |    f2
----+------------
 42 | 42 is text
(1 row)
```

5、计算两个整数的和，并返回结果。如果输入为 null，则返回 null。

```
CREATE PROCEDURE func_add_sql2(num1 integer, num2 integer) RETURN integer
AS
BEGIN
RETURN num1 + num2;
END;
/
```

6、修改函数 func_add_sql2 的执行规则为 IMMUTABLE，即参数不变时返回相同结果。

```
ALTER PROCEDURE func_add_sql2(INTEGER, INTEGER) IMMUTABLE;
```

7、将函数 func_add_sql2 的名称修改为 add_two_number。

```
 ALTER PROCEDURE func_add_sql2(INTEGER, INTEGER) RENAME TO add_two_numbe
r
```

8、将函数 add_two_number 的属者改为 panweidb。

```
ALTER PROCEDURE add_two_number(INTEGER, INTEGER) OWNER TO panweidb;
```

9、删除函数。

```
DROP PROCEDURE add_two_number;
```

```
DROP PROCEDURE func_increment_sql;
DROP PROCEDURE func_dup_sql;
DROP PROCEDURE func_increment_plsql;
DROP PROCEDUREfunc_add_sql;
```

相关链接

CREATE PROCEDURE(参见：开发者指南->SQL 语法参考->SQL 语法->CREATE PROCEDURE 章节)，DROP PROCEDURE(参见：开发者指南->SQL 语法参考->SQL 语法->DROP PROCEDURE 章节)

4.5.27.ALTER SCHEMA

功能描述

修改模式属性。

注意事项

- 只有模式的所有者或者被授予了模式 ALTER 权限的用户有权限执行 ALTER SCHEMA 命令，系统管理员默认拥有此权限。但要修改模式的所有者，当前用户必须是该模式的所有者或者系统管理员，且该用户是新所有者角色的成员。
- 对于系统模式 pg_catalog，只允许初始用户修改模式的所有者。

语法格式

- 修改模式的防篡改属性。

```
ALTER SCHEMA schema_name { WITH | WITHOUT } BLOCKCHAIN
```

- 修改模式的名称。

```
ALTER SCHEMA schema_name
  RENAME TO new_name;
```

- 修改模式的所有者。

```
ALTER SCHEMA schema_name
  OWNER TO new_owner;
```

参数说明

- **schema_name**

  现有模式的名称。

  取值范围：已存在的模式名。

- **RENAME TO new_name**

  修改模式的名称。非系统管理员要改变模式的名称，则该用户必须在此数据库上有 CREATE 权限。

  new_name：模式的新名称。

  取值范围：字符串，要符合标识符命名规范。

- **OWNER TO new_owner**

  修改模式的所有者。非系统管理员要改变模式的所有者，该用户还必须是新的所有角色的直接或间接成员， 并且该成员必须在此数据库上有 CREATE 权限。

  new_owner：模式的新所有者。

  取值范围：已存在的用户名/角色名。

- **{ WITH | WITHOUT } BLOCKCHAIN**

  修改模式的防篡改属性。具有防篡改属性模式下的普通行存表均为防篡改历史表，不包括外表、临时表、系统表。当该模式下不包含任何表时才可修改防篡改属性。另外，不支持临时表模式、toast 表模式、dbe_perf 模式、blockchain 模式修改防篡改属性。

示例

```
--创建模式 ds。
panweidb=# CREATE SCHEMA ds;

--将当前模式 ds 更名为 ds_new。
panweidb=# ALTER SCHEMA ds RENAME TO ds_new;

--创建用户 jack。
panweidb=# CREATE USER jack PASSWORD 'xxxxxxxxx';

--将 DS_NEW 的所有者修改为 jack。
panweidb=# ALTER SCHEMA ds_new OWNER TO jack;
```

```
--删除用户 jack 和模式 ds_new。
panweidb=# DROP SCHEMA ds_new;
panweidb=# DROP USER jack;
```

相关链接

CREATE SCHEMA (参见：开发者指南->SQL 语法参考->SQL 语法->CREATE SCHEMA 章节)，DROP SCHEMA (参见：开发者指南->SQL 语法参考->SQL 语法->DROP SCHEMA 章节)

4.5.28.ALTER SEQUENCE

功能描述

修改一个现有的序列的参数。

注意事项

- 序列的所有者或者被授予了序列 ALTER 权限的用户或者被授予了 ALTER ANY SEQUENCE 权限的用户才能执行 ALTER SEQUENCE 命令，系统管理员默认拥有该权限。但要修改序列的所有者，当前用户必须是该序列的所有者或者系统管理员，且该用户是新所有者角色的成员。
- 当前版本仅支持修改步长、最大值、最小值、起始值、缓冲值、是否循环、重新开始、归属列和拥有者。若要修改其他参数，可以删除重建，并用 Setval 函数恢复当前值。
- ALTER SEQUENCE MAXVALUE 不支持在事务、函数和存储过程中使用。
- 修改序列的最大值后，会清空该序列在所有会话的 cache。
- 如果 Sequence 被创建时使用了 LARGE 标识，则 ALTER 时也需要使用 LARGE 标识。
- ALTER SEQUENCE 会阻塞 nextval、setval、currval 和 lastval 的调用。

语法格式

- 修改序列归属列

```
ALTER [ LARGE ] SEQUENCE [ IF EXISTS ] name [ INCREMENT [ BY ] increment ]
```

```
    [ MINVALUE minvalue | NO MINVALUE | NOMINVALUE ] [MAXVALUE maxvalue | NO MAXVALUE | NOMAXVALUE]
    [ START [ WITH ] start ] [ CACHE cache ] [ [ NO ] CYCLE | NOCYCLE ] [ RESTART [ WITH ] restart ]
    [ OWNED BY { table_name.column_name | NONE } ] ;
```

- 修改序列的拥有者

```
ALTER SEQUENCE [ IF EXISTS ] name OWNER TO new_owner;
```

参数说明

- **name**

  将要修改的序列名称。

- **IF EXISTS**

  当序列不存在时使用该选项不会出现错误消息，仅有一个通知。

- **INCREMENT**

  指定序列的步长。

- **MINVALUE minvalue | NO MINVALUE| NOMINVALUE**

  指定序列的最小值。如果没有声明 minvalue 或者声明了 NO MINVALUE，则递增序列的缺省值为 1，递减序列的缺省值为$-2^{63}+1$（Large 序列为$-2^{127}+1$）。NOMINVALUE 等价于 NO MINVALUE

- **MAXVALUE maxvalue | NO MAXVALUE| NOMAXVALUE**

  指定序列的最大值。如果没有声明 maxvalue 或者声明了 NO MAXVALUE，则递增序列的缺省值为 $2^{63}-1$（Large 序列为 $2^{127}-1$），递减序列的缺省值为-1。NOMAXVALUE 等价于 NO MAXVALUE

- **START**

  指定序列的起始值。

- **CACHE**

  为了快速访问，而在内存中预先存储序列号的个数。如果没有指定，将保持旧的缓冲值。

- **CYCLE**

  用于使序列达到 maxvalue 或者 minvalue 后可循环并继续下去。

如果声明了 NO CYCLE，则在序列达到其最大值后任何对 nextval 的调用都会返回一个错误。

NOCYCLE 的作用等价于 NO CYCLE。

若修改序列为 CYCLE，则不能保证序列的唯一性。

- **RESTART**

  用于更改序列的当前值，指定的当前值将作为下次调用 nextval 的结果返回。缺省值为序列的起始值。

- **OWNED BY**

  将序列和一个表的指定字段进行关联。这样，在删除那个字段或其所在表的时候会自动删除已关联的序列。

  如果序列已经和表有关联后，使用这个选项后新的关联关系会覆盖旧的关联。

  关联的表和序列的所有者必须是同一个用户，并且在同一个模式中。

  使用 OWNED BY NONE 将删除任何已经存在的关联。

- **new_owner**

  序列新所有者的用户名。用户要修改序列的所有者，必须是新角色的直接或者间接成员，并且那个角色必须有序列所在模式上的 CREATE 权限。

- **number**

  序列的起始值。

示例

```
--创建一个名为 serial 的递增序列，从 101 开始。
panweidb=# CREATE SEQUENCE serial START 101;
--创建一个表,定义默认值。
panweidb=# CREATE TABLE T1(C1 bigint default nextval('serial'));
--将序列 serial 的归属列变为 T1.C1。
panweidb=# ALTER SEQUENCE serial OWNED BY T1.C1;
--删除序列和表。
panweidb=# DROP SEQUENCE serial cascade;
panweidb=# DROP TABLE T1;
```

相关链接

CREATE SEQUENCE(参见：开发者指南->SQL 语法参考->SQL 语法->CREATE SEQUENCE 章节)，DROP SEQUENCE(参见：开发者指南->SQL 语法参考->SQL 语法->DROP SEQUENCE 章节)

4.5.29.ALTER SERVER

功能描述

增加、修改和删除一个现有 server 的参数。已有 server 可以从 pg_foreign_server 系统表中查询。

注意事项

只有 SERVER 的所有者或者被授予了 SERVER 的 ALTER 权限的用户才可以执行 ALTER SERVER 命令，系统管理员默认拥有该权限。但要修改 SERVER 的所有者，当前用户必须是该 SERVER 的所有者或者系统管理员，且该用户是新所有者角色的成员。

语法格式

❖ 修改外部服务的参数。

```
ALTER SERVER server_name [ VERSION 'new_version' ]
    [ OPTIONS ( {[ ADD | SET | DROP ] option ['value']} [, ... ] ) ];
```

在 OPTIONS 选项里，ADD、SET 和 DROP 指定要执行的操作，未指定时默认为 ADD 操作。option 和 value 为对应操作的参数。

❖ 修改外部服务的名称。

```
ALTER SERVER server_name
 RENAME TO new_name;
```

参数说明

❖ **server_name**

所修改的 server 的名称。

❖ **new_version**

修改后 server 的新版本名称。

❖ **OPTIONS**

更改该服务器的选项。ADD、SET 和 DROP 指定要执行的动作。如果没有显式地指定操作，将会假定为 ADD。选项名称必须唯一，名称和值也会使用该服务器的外部数据包装器库进行验证。

- oracle_fdw 支持的 options 包括：
  - **dbserver**

    远端 oracle 数据库的连接字符串。
  - **isolation_level**（默认值为 serializable）

    oracle 数据库的事务隔离级别。

    取值范围：serializable、read_committed、read_only
- mysql_fdw 支持的 options 包括：
  - **host**（默认值为 127.0.0.1）

    MySQL Server/MariaDB 的地址。
  - **port**（默认值为 3306）

    MySQL Server/MariaDB 侦听的端口号。
- postgres_fdw 支持的 options 同 libpq 支持的连接参数一致。需要注意的是，以下几个 options 不支持修改：
  - **user** 和 **password**

    用户名和密码将在创建 user mapping 时指定。
  - **client_encoding**

    将自动获取本地 server 的编码方式并设置该值。
  - **application_name**

    总是设置成 postgres_fdw。
- 除了 libpq 支持的连接参数外，还额外提供 3 个 options：
  - **use_remote_estimate**

    控制 postgres_fdw 是否发出 EXPLAIN 命令以获取运行消耗估算。默认值为 false。
  - **fdw_startup_cost**

    执行一个外表扫描时的启动耗时估算。这个值通常包含建立连接、远端对请求的分析和生成计划的耗时。默认值为 100。
  - **fdw_typle_cost**

在远端服务器上对每一个元组进行扫描时的额外消耗。这个值通常表示数据在 server 间传输的额外消耗。默认值为 0.01。

- **new_name**

  修改后 server 的新名称。

相关链接

CREATE SERVER (参见：开发者指南->SQL 语法参考->SQL 语法->CREATE SERVER 章节)，DROP SERVER (参见：开发者指南->SQL 语法参考->SQL 语法->DROP SERVER 章节)

4.5.30.ALTER SESSION

功能描述

ALTER SESSION 命令用于定义或修改那些对当前会话有影响的条件或参数。修改后的会话参数会一直保持，直到断开当前会话。

注意事项

- 如果执行 SET TRANSACTION 之前没有执行 START TRANSACTION，则事务立即结束，命令无法显示效果。
- 可以用 START TRANSACTION 里面声明所需要的 transaction_mode(s) 的方法来避免使用 SET TRANSACTION。

语法格式

- 设置会话的事务参数。

```
ALTER SESSION SET [ SESSION CHARACTERISTICS AS ] TRANSACTION
  { ISOLATION LEVEL { READ COMMITTED } | { READ ONLY | READ WRITE } } [, ...] ;
```

- 设置会话的其他运行时参数。

```
ALTER SESSION SET
  {{config_parameter { { TO | = } { value | DEFAULT }
    | FROM CURRENT }}
    | TIME ZONE time_zone
    | CURRENT_SCHEMA schema
    | NAMES encoding_name
```

```
    | ROLE role_name PASSWORD 'password'
    | SESSION AUTHORIZATION { role_name PASSWORD 'password' | DEFAULT      | X
ML OPTION { DOCUMENT | CONTENT }
  } ;
```

参数说明

修改会话涉及到的参数说明请参见 SET 语法 (参见：开发者指南->SQL 语法参考->SQL 语法->SET 章节)中的参数说明。

示例

```
-- 创建模式 ds。
panweidb=# CREATE SCHEMA ds;

--设置模式搜索路径。
panweidb=# SET SEARCH_PATH TO ds, public;

--设置日期时间风格为传统的 POSTGRES 风格（日在月前）。
panweidb=# SET DATESTYLE TO postgres, dmy;

--设置当前会话的字符编码为 UTF8。
panweidb=# ALTER SESSION SET NAMES 'UTF8';

--设置时区为加州伯克利。
panweidb=# SET TIME ZONE 'PST8PDT';

--设置时区为意大利。
panweidb=# SET TIME ZONE 'Europe/Rome';

--设置当前模式。
panweidb=# ALTER SESSION SET CURRENT_SCHEMA TO tpcds;

--设置 XML OPTION 为 DOCUMENT。
panweidb=# ALTER SESSION SET XML OPTION DOCUMENT;

--创建角色 joe，并设置会话的角色为 joe。
```

```
panweidb=# CREATE ROLE joe WITH PASSWORD 'xxxxxxxxx';
panweidb=# ALTER SESSION SET SESSION AUTHORIZATION joe PASSWORD 'xxxxxxx
xx';

--切换到默认用户。
panweidb=> ALTER SESSION SET SESSION AUTHORIZATION default;

--删除 ds 模式。
panweidb=# DROP SCHEMA ds;

--删除 joe。
panweidb=# DROP ROLE joe;
```

相关链接

SET (参见：开发者指南->SQL 语法参考->SQL 语法->SET 章节)

4.5.31.ALTER SUBSCRIPTION

功能描述

ALTER SUBSCRIPTION 可以修改在 CREATE SUBSCRIPTION 中指定的订阅属性。

注意事项

订阅的所有者才能执行 ALTER SUBSCRIPTION，并且新的所有者必须是系统管理员。

语法格式

❖ 更新订阅的连接信息。

```
ALTER SUBSCRIPTION name CONNECTION 'conninfo'
```

❖ 更新订阅的发布端的发布名称。

```
ALTER SUBSCRIPTION name SET PUBLICATION publication_name [, ...]
```

❖ 更新订阅的订阅列表。

```
ALTER SUBSCRIPTION name REFRESH PUBLICATION [ WITH ( refresh_option [= value]
[, ... ] ) ]
```

❖ 激活订阅。

```
ALTER SUBSCRIPTION name ENABLE
```

❖ 更新 CREATE SUBSCRIPTION 中定义的属性。

```
ALTER SUBSCRIPTION name SET ( subscription_parameter [= value] [, ... ] )
```

❖ 更新订阅的属主。

```
ALTER SUBSCRIPTION name OWNER TO { new_owner | CURRENT_USER | SESSION_USER }
```

❖ 修改订阅的名称。

```
ALTER SUBSCRIPTION name RENAME TO new_name
```

参数说明

❖ **name**

要修改属性的订阅的名称。

❖ **CONNECTION 'conninfo'**

该子句修改最初由 CREATE SUBSCRIPTION 设置的连接属性。

❖ **ENABLE (boolean)**

指定订阅是否应该主动复制，或者是否应该只是设置，但尚未启动。默认值是 true。

❖ **SET (subscription_parameter [= value] [, ...])**

该子句修改原先由 CREATE SUBSCRIPTION 设置的参数。允许的选项是 slot_name 和 synchronous_commit。

- 如果创建订阅时设置 enabled 为 false，则 slot_name 将被强制设置为 NONE，即空值，即使用户指定了 slot_name 的值，复制槽也不存在。
- 将 enabled 参数的值由 false 改为 true，即启用订阅时，将会连接发布端创建复制槽，此时如果用户未指定 slot_name 参数的值，则会使用默认值，即对应的订阅的名称。
- 当 enabled 为 true，即订阅处于正常使用状态，不能修改 slot_name 为空，但可以修改复制槽的名称为其他非空合法名称。

❖ **REFRESH PUBLICATION**

从发布端获取缺少的表信息。这将开始复制自上次调用 REFRESH PUBLICATION 或从 CREATE SUBSCRIPTION 以来添加到订阅发布中的表。

refresh_option 指定了刷新操作的附加选项。支持的选项有：

copy_data (boolean)

指定在复制启动后是否应复制正在订阅的发布中的现有数据。默认值是 true。（以前订阅的表不会被复制）

- **new_owner**

订阅的新所有者的用户名。

- **new_name**

订阅的新名称。

示例

请参见 CREATE SUBSCRIPTION 的示例。

相关链接

CREATE SUBSCRIPTION (参见：开发者指南->SQL 语法参考->SQL 语法->CREATE SUBSCRIPTION 章节)，DROP SUBSCRIPTION(参见：开发者指南->SQL 语法参考->SQL 语法->DROP SUBSCRIPTION 章节)

4.5.32.ALTER SYNONYM

功能描述

修改 SYNONYM 对象的属性。

注意事项

- 目前仅支持修改 SYNONYM 对象的属主。
- 只有系统管理员有权限修改 SYNONYM 对象的属主信息。
- 新属主必须具有 SYNONYM 对象所在模式的 CREATE 权限。

语法格式

```
ALTER SYNONYM synonym_name OWNER TO new_owner;
```

参数描述

- **synonym**

  待修改的同义词名字，可以带模式名。

  取值范围：字符串，需要符合标识符的命名规范。

- **new_owner**

  同义词对象的新所有者。

  取值范围：字符串，有效的用户名。

示例

1、创建同义词 t1。

```
CREATE OR REPLACE SYNONYM t1 FOR t1;
```

2、创建新用户 u1，将授予 public 模式的 CREATE 权限。

```
CREATE USER u1 PASSWORD 'user@111';
GRANT CREATE ON SCHEMA public to u1;
```

3、修改同义词 t1 的 owner 为 u1。

```
ALTER SYNONYM t1 OWNER TO u1;
```

4、删除同义词 t1。

```
DROP SYNONYM t1;
```

5、删除用户 u1。

```
DROP USER u1 CASCADE;
```

相关链接

CREATE SYNONYM (参见：开发者指南->SQL 语法参考->SQL 语法->CREATE SYNONYM 章节)，DROP SYNONYM(参见：开发者指南->SQL 语法参考->SQL 语法->DROP SYNONYM 章节)

4.5.33.ALTER SYSTEM KILL SESSION

功能描述

ALTER SYSTEM KILL SESSION 命令用于结束一个会话。

注意事项

无。

语法格式

```
ALTER SYSTEM KILL SESSION 'session_sid, serial' [ IMMEDIATE ];
```

参数说明

- **session_sid, serial**

  会话的 SID 和 SERIAL（获取方法请参考示例）。

- **IMMEDIATE**

  表明会话将在命令执行后立即结束。

示例

```
--查询会话信息。
panweidb=#
SELECT sa.sessionid AS sid,0::integer AS serial#,ad.rolname AS username FROM pg
_stat_get_activity(NULL) AS sa
LEFT JOIN pg_authid ad ON(sa.usesysid = ad.oid)WHERE sa.application_name <> 'Jo
bScheduler';
     sid      | serial# | username
----------------+---------+----------
 140131075880720 |       0 | omm
 140131025549072 |       0 | omm
 140131073779472 |       0 | omm
 140131071678224 |       0 | omm
 140131125774096 |       0 |
 140131127875344 |       0 |
 140131113629456 |       0 |
 140131094742800 |       0 |
```

```
(8 rows)

--结束 SID 为 140131075880720 的会话。
panweidb=# ALTER SYSTEM KILL SESSION '140131075880720,0' IMMEDIATE;
```

4.5.34.ALTER SYSTEM SET

功能描述

ALTER SYSTEM SET 命令用于设置 POSTMASTER、SIGHUP、BACKEND 级别的 GUC 参数。此命令会将参数写入配置文件，不同级别生效方式有所不同。

注意事项

- 此命令仅限初始用户和拥有 sysadmin 权限的用户才可使用。
- 不同级别 GUC 参数生效时间如下：
  - POSTMASTER 级别的 GUC 参数需要重启后才生效。
  - BACKEND 级别的 GUC 参数需要会话重新连接后才生效。
  - SIGHUP 级别的 GUC 参数立即生效（需要等待线程重新加载参数，实际略微有延迟）。
- 通过配置 audit_set_parameter 参数 (参见：开发者指南->GUC 参数说明->审计->操作审计章节的 audit_set_parameter 参数)，可以配置此操作是否被审计。
- 操作可被备机同步。
- 同 pw_guc 一致，并不关注数据库是主或备节点、是否只读。
- 不可在事务中执行，因为此操作无法被回滚。
- 部分参数只能由初始用户修改，具体如下：

```
audit_copy_exec, audit_data_format, audit_database_process, audit_directory, au
dit_dml_state,audit_dml_state_select, audit_enabled, audit_file_remain_threshold,
 audit_file_remain_time,audit_function_exec, audit_grant_revoke, audit_login_logo
ut, audit_resource_policy,audit_rotation_interval, audit_rotation_size, audit_set_pa
rameter, audit_space_limit,audit_system_object, audit_user_locked, audit_user_vio
lation,enable_access_server_directoty,enable_copy_server_files,modify_initial_pas
sword
```

语法格式

```
ALTER SYSTEM SET parameter TO value;
```

参数说明

- **parameter**

  GUC 参数名。

- **value**

  GUC 参数值。

示例

```
--设置 SIGHUP 级别参数 audit_enabled。
panweidb=# alter system set audit_enabled to off;
ALTER SYSTEM SET
panweidb=# show audit_enabled;
 audit_enabled
---------------
 off
(1 row)

--设置 POSTMASTER 级别参数 enable_thread_pool，将在重启之后生效。
panweidb=# alter system set enable_thread_pool to on;
NOTICE:  please restart the database for the POSTMASTER level parameter to take e
ffect.
ALTER SYSTEM SET
```

4.5.35.ALTER TABLE

功能描述

修改表，包括修改表的定义、重命名表、重命名表中指定的列、重命名表的约束、设置表的所属模式、添加或更新多个列、打开或关闭行访问控制开关。

注意事项

- 表的所有者被授予了表 ALTER 权限的用户或被授予 ALTER ANY TABLE 的用户有权限执行 ALTER TABLE 命令，系统管理员默认拥有此权限。但要修改表的所有者或者修改表的模式，当前用户必须是该表的所有者或者系统管理员，且该用户是新所有者角色的成员。
- 不能修改分区表的 tablespace，但可以修改分区的 tablespace。
- 不支持修改存储参数 ORIENTATION。
- SET SCHEMA 操作不支持修改为系统内部模式，当前仅支持用户模式之间的修改。
- 列存表只支持 PARTIAL CLUSTER KEY、UNIQUE、PRIMARY KEY 表级约束，不支持外键等表级约束。
- 列存表只支持添加字段 ADD COLUMN、修改字段的数据类型 ALTER TYPE、设置单个字段的收集目标 SET STATISTICS、支持更改表名称、支持更改表空间、支持删除字段 DROP COLUMN。对于添加的字段和修改的字段类型要求是列存表支持的数据类型（参考《PanWeiDB V2.0-S3.0.1_B01 开发者指南》->列存表支持的数据类型）。ALTER TYPE 的 USING 选项只支持常量表达式和涉及本字段的表达式，暂不支持涉及其他字段的表达式。
- 列存表支持的字段约束包括 NULL、NOT NULL、DEFAULT 常量值、UNIQUE 和 PRIMARY KEY；对字段约束的修改当前只支持对 DEFAULT 值的修改（SET DEFAULT）和删除（DROP DEFAULT），暂不支持对非空约束 NULL/NOT NULL 的修改。
- 不支持增加自增列，或者增加 DEFAULT 值中包含 nextval()表达式的列。
- 不支持对外表、临时表开启行访问控制开关。
- 通过约束名删除 PRIMARY KEY 约束时，不会删除 NOT NULL 约束，如果有需要，请手动删除 NOT NULL 约束。
- 使用 JDBC 时，支持通过 PrepareStatement 对 DEFAULT 值进行参数化设置。
- 修改表时字段前的可选关键字[COLUMN]仅能修饰单个字段，不支持使用一个 COLUMN 关键字修饰多个字段的聚合，即以下用法会报错：

```
ADD COLUMN( col1 ... , col2 ...)
```

若同时修饰多个字段，则需使用如下写法：

```
ADD/MODIFY COLUMN col1 ... , ADD/MODIFY COLUMN col2 ...
```

- 重命名时，不能与当前命名空间的 synonym 产生命名冲突。
- 设置命名空间时，不能与当前命名空间的 synonym 产生命名冲突。
- 使用 FIRST | AFTER column_name 新增列或修改列，或修改字段的字符集，会带来全表更新开销，影响在线业务。向已有的字段之间新插入列时，需要保证引用了字段的视图对象有效。
- 删除被视图引用的表字段或修改表字段类型以及字段长度时，将引用视图和物化视图置为无效状态，在查询无效视图或通过无效视图更新、删除和新增表记录以及全量更新物化视图时，检查无效的视图和物化视图引用的表字段是否全部存在，如果存在恢复视图和物化视图的有效状态并返回查询结果，否则报错提示查询无效视图。

语法格式

修改表的定义

```
ALTER TABLE [ IF EXISTS ] { table_name [*] | ONLY table_name | ONLY
( table_name ) }
    action [, ... ];
```

- 其中具体表操作 action 可以是以下子句之一：

```
column_clause
  | ADD table_constraint [ NOT VALID ]
  | ADD table_constraint_using_index
  | VALIDATE CONSTRAINT constraint_name
  | DROP CONSTRAINT [ IF EXISTS ]  constraint_name [ RESTRICT | CASCADE ]
  | CLUSTER ON index_name
  | SET WITHOUT CLUSTER
```

```
    | SET ( {storage_parameter = value} [, ... ] )
    | RESET ( storage_parameter [, ... ] )
    | OWNER TO new_owner
    | SET TABLESPACE new_tablespace
    | SET {COMPRESS|NOCOMPRESS}
    | TO { GROUP groupname | NODE ( nodename [, ... ] ) }
    | ADD NODE ( nodename [, ... ] )
    | DELETE NODE ( nodename [, ... ] )
    | DISABLE TRIGGER [ trigger_name | ALL | USER ]
    | ENABLE TRIGGER [ trigger_name | ALL | USER ]
    | ENABLE REPLICA TRIGGER trigger_name
    | ENABLE ALWAYS TRIGGER trigger_name
    | DISABLE/ENABLE [ REPLICA | ALWAYS ] RULE
    | DISABLE ROW LEVEL SECURITY
    | ENABLE ROW LEVEL SECURITY
    | FORCE ROW LEVEL SECURITY
    | NO FORCE ROW LEVEL SECURITY
    | ENCRYPTION KEY ROTATION
    | AUTO_INCREMENT [ = ] value
    | INHERIT parents
    | NO INHERIT parents
    | OF type_name
    | NOT OF
| REPLICA IDENTITY { DEFAULT | USING INDEX index_name | FULL | NOTHING }
```

以下是对于各种 action 子句含义的说明：

- **ADD table_constraint [NOT VALID]**

  给表增加一个新的约束。

- **ADD table_constraint_using_index**

  根据已有唯一索引为表增加主键约束或唯一约束。

- **VALIDATE CONSTRAINT constraint_name**

  验证一个使用 NOT VALID 选项创建的检查类约束，通过扫描全表来保证所有记录都符合约束条件。如果约束已标记为有效时，什么操作也不会发生。

- **DROP CONSTRAINT [IF EXISTS] constraint_name [RESTRICT | CASCADE]**

  删除一个表上的约束。

- **CLUSTER ON index_name**

  为将来的 CLUSTER（聚簇）操作选择默认索引。实际上并没有重新盘簇化处理该表。

- **SET WITHOUT CLUSTER**

  从表中删除最新使用的 CLUSTER 索引。这样会影响将来那些没有声明索引的 CLUSTER（聚簇）操作。

- **SET ({storage_parameter = value} [, ...])**

  修改表的一个或多个存储参数。

- **RESET (storage_parameter [, ...])**

  重置表的一个或多个存储参数。与 SET 一样，根据参数的不同可能需要重写表才能获得想要的效果。

- **OWNER TO new_owner**

  将表、序列、视图的属主改变成指定的用户。

- **SET TABLESPACE new_tablespace**

  这种形式将表空间修改为指定的表空间并将相关的数据文件移动到新的表空间。但是表上的所有索引都不会被移动，索引可以通过 ALTER INDEX 语法的 SET TABLESPACE 选项来修改索引的表空间。

- **SET {COMPRESS|NOCOMPRESS}**

  修改表的压缩特性。表压缩特性的改变只会影响后续批量插入的数据的存储方式，对已有数据的存储毫无影响。也就是说，表压缩特性的修改会导致该表中同时存在着已压缩和未压缩的数据。

  【须知】

  行存表不支持压缩。

- **TO { GROUP groupname | NODE (nodename [, ...]) }**

  此语法仅在扩展模式（GUC 参数 support_extended_features 为 on 时）下可用。该模式谨慎打开，主要供内部扩容工具使用，一般用户不应使用该模式。

- **ADD NODE (nodename [, ...])**

  此语法主要供内部扩容工具使用，一般用户不建议使用。

- **DELETE NODE (nodename [, ...])**

  此语法主要供内部缩容工具使用，一般用户不建议使用。

- **DISABLE TRIGGER [trigger_name | ALL | USER]**

  禁用 trigger_name 所表示的单个触发器，或禁用所有触发器，或仅禁用用户触发器。

  【须知】

  ❖ 此选项不包括内部生成的约束触发器，例如，可延迟唯一性和排除约束的约束触发器。

  ❖ 应谨慎使用此功能，因为如果不执行触发器，则无法保证原先期望的约束的完整性。

- **ENABLE TRIGGER [trigger_name | ALL | USER]**

  启用 trigger_name 所表示的单个触发器，或启用所有触发器，或仅启用用户触发器。

- **ENABLE REPLICA TRIGGER trigger_name**

  触发器触发机制受配置变量 session_replication_role（参考《PanWeiDB V2.0-S3.0.1_B01 开发者指南》）的影响，当复制角色为“origin”（默认值）或“local”时，将触发简单启用的触发器。

  配置为 ENABLE REPLICA 的触发器仅在会话处于“replica”模式时触发。

- **ENABLE ALWAYS TRIGGER trigger_name**

  无论当前复制模式如何，配置为 ENABLE ALWAYS 的触发器都将触发。

- DISABLE/ENABLE [REPLICA | ALWAYS] RULE

  配置属于表的重写规则，已禁用的规则对系统来说仍然是可见的，只是在查询重写期间不被应用。语义为关闭/启动规则。由于关系到视图的实现，ON SELECT 规则不可禁用。 配置为 ENABLE REPLICA 的规则将会仅在会话为"replica" 模式时启动，而配置为 ENABLE ALWAYS 的触发器将总是会启动，不考虑当前复制模式。规则触发机制也受配置变量 session_replication_role 的影响，类似于上述触发器。

  DISABLE/ENABLE ROW LEVEL SECURITY 开启或关闭表的行访问控制开关。

  当开启行访问控制开关时，如果未在该数据表定义相关行访问控制策略，数据表的行级访问将不受影响；如果关闭表的行访问控制开关，即使定义了行访问控制策略，数据表的行访问也不受影响。详细信息参见《PanWeiDB V2.0-S3.0.1_B01 开发者指南》->SQL 语法参考->CREATE ROW LEVEL SECURITY POLICY。

- **NO FORCE/FORCE ROW LEVEL SECURITY**

  强制开启或关闭表的行访问控制开关。

  默认情况，表所有者不受行访问控制特性影响，但当强制开启表的行访问控制开关时，表的所有者（不包含系统管理员用户）会受影响。系统管理员可以绕过所有的行访问控制策略，不受影响。

- **ENCRYPTION KEY ROTATION**

  透明数据加密密钥轮转。只有在数据库开启透明加密功能，并且表的 enable_tde 选项为 on 时才可以进行表的数据加密密钥轮转。执行密钥轮转操作后，系统会自动向 KMS 申请创建新的密钥。密钥轮转后，使用旧密钥加密的数据仍使用旧密钥解密，新写入的数据使用新密钥加密。为保证加密数据安全，用户可根据加密表的新增数据量大小定期更新密钥，建议更新周期为两到三年。

- **AUTO_INCREMENT [=] value**

  value 表示修改后自增字段的下一个值。仅当 value 的值大于当前自增字段的最大值时，修改才真正生效。

【须知】

该字段仅在数据库兼容模式为 MySQL 时支持（即数据库初始化时指定 DBCOMPATIBILITY='B'）。详见 ALTER TABLE（参考《PanWeiDB V2.0-S3.0.1_B01 兼容性手册》->MySQL 兼容性->ALTER TABLE。

- **INHERIT parent_table**

  将目标资料表加到指定的父资料表中成为新的子资料表。之后，针对父资料表的查询将会包含目标资料表的资料。要作为子资料表加入前，目标资料表必须已经包含父资料表的所有栏位。这些栏位必须具有可匹配的资料类别，并且如果他们在父资料表中具有 NOT NULL 的限制条件，那么他们必须在子资料表中也具有 NOT NULL 的限制条件。对于父资料表的所有 CHECK 限制条件，必须还有相对应的子资料表限制条件，除非父资料表中标记为不可继承。

- **NO INHERIT parent_table**

  从指定的父资料表的子资料表中产出目标资料表。针对父资料表的查询将不再包含从目标资料表中所产生的记录。

- **OF type_name**

  将表连接至一种复合类型，与 CREATE TABLE OF 选项创建表一样。表的字段的名称和类型必须精确匹配复合类型中的定义，不过 oid 系统字段允许不一样。表不能是从任何其他表继承的。这些限制确保 CREATE TABLE OF 选项允许一个相同的表定义。

- **NOT OF**

将一个与某类型进行关联的表进行关联的解除。

- **REPLICA IDENTITY { DEFAULT | USING INDEX index_name | FULL | NOTHING }**

  在逻辑复制场景下，指定该表的 UPDATE 和 DELETE 操作中旧元组的记录级别。

  - DEFAULT 记录主键的列的旧值。
  - USING INDEX 记录命名索引覆盖的列的旧值，这些值必须是唯一的，不局部的，不可延迟的，并且仅包括标记为 NOT NULL 的列。
  - FULL 记录该行中所有列的旧值。
  - NOTHING 不记录有关旧行的信息。

在逻辑复制场景，解析该表的 UPDATE 和 DELETE 操作语句时，解析出的旧元组由以此方法记录的信息组成。对于有主键表该选项可设置为 DEFAULT 或 FULL。对于无主键表该选项需设置为 FULL，否则解码时旧元组将解析为空。一般场景不建议设置为 NOTHING，旧元组会始终解析为空。

即使指定 DEFAULT 或 USING INDEX，当前 ustore 表列的旧值中也可能包含该行所有列的旧值，只有旧值涉及 toast 该配置选项才会生效。另外针对 ustore 表，选项 NOTHING 无效，实际效果等同于 FULL。

❖ 其中列相关的操作 column_clause 可以是以下子句之一 ：

```
    ADD [ COLUMN ] [ IF NOT EXISTS ] column_name data_type [ compress_mode ] [ COLLATE collation ] [ column_constraint [ ... ] ]

    | MODIFY column_name data_type

    | MODIFY column_name [ CONSTRAINT constraint_name ] NOT NULL [ ENABLE ]

    | MODIFY column_name [ CONSTRAINT constraint_name ] NULL

    | DROP [ COLUMN ] [ IF EXISTS ] column_name [ RESTRICT | CASCADE ]

    | ALTER [ COLUMN ] column_name [ SET DATA ] TYPE data_type [ COLLATE collation ] [ USING expression ]

    | ALTER [ COLUMN ] column_name { SET DEFAULT expression | DROP DEFAULT }

    | ALTER [ COLUMN ] column_name { SET | DROP } NOT NULL

    | ALTER [ COLUMN ] column_name SET STATISTICS [PERCENT] integer

    | ADD STATISTICS (( column_1_name, column_2_name [, ...] ))

    | DELETE STATISTICS (( column_1_name, column_2_name [, ...] ))

    | ALTER [ COLUMN ] column_name SET ( {attribute_option = value} [, ... ] )

    | ALTER [ COLUMN ] column_name RESET ( attribute_option [, ... ] )

    | ALTER [ COLUMN ] column_name SET STORAGE { PLAIN | EXTERNAL | EXTENDED | MAIN }
```

以下是对于各种 column_clause 子句含义的说明：

- **ADD [COLUMN] [IF NOT EXISTS] column_name data_type [CHARACTER SET | CHARSET charset] [compress_mode] [COLLATE collation] [column_constraint [...]] [FIRST | AFTER column_name]**

  向表中增加一个新的字段。用 ADD COLUMN 增加一个字段，所有表中现有行都初始化为该字段的缺省值（如果没有声明 DEFAULT 子句，值为 NULL）。

  【须知】

  ❖ 新增字段时使用[IF NOT EXISTS]选项的功能仅在 PostgreSQL 兼容模式下支持，表示如果已经存在相同名称的字段，不会抛出一个错误，而会发出一个通知，告知字段已存在。详见 PostgreSQL 兼容性->ALTER TABLE。

  ❖ 新增选项 FIRST 或 AFTER 用于指定修改或新增的字段位置，该选项仅在数据库兼容模式为 MySQL 时支持（即数据库初始化时指定 DBCOMPATIBILITY='B'）。详见 MySQL 兼容性->ALTER TABLE。

- **ADD ({ column_name data_type [compress_mode] } [, ...])**

  向表中增加多列。

- **MODIFY ({ column_name data_type | column_name [CONSTRAINT constraint_name] NOT NULL [ENABLE] | column_name [CONSTRAINT constraint_name] NULL } [, ...])**

  修改表已存在字段的数据类型。

- **DROP [COLUMN] [IF EXISTS] column_name [RESTRICT | CASCADE]**

  从表中删除一个字段，和这个字段相关的索引和表约束也会被自动删除。如果任何表之外的对象依赖于这个字段，必须声明 CASCADE，比如视图。DROP COLUMN 命令并不是物理上把字段删除，而只是简单地把它标记为对 SQL 操作不可见。随后对该表的插入和更新将在该字段存储一个 NULL。因此，删除一个字段是很快的，但是它不会立即释放表在磁盘上的

空间，因为被删除了的字段占据的空间还没有回收。这些空间将在执行 VACUUM 时而得到回收。

- **ALTER [COLUMN] column_name [SET DATA] TYPE data_type [COLLATE collation] [USING expression]**

  改变表字段的数据类型。该字段涉及的索引和简单的表约束将被自动地转换为使用新的字段类型，方法是重新分析最初提供的表达式。ALTER TYPE 要求重写整个表的特性有时候是一个优点，因为重写的过程消除了表中没用的空间。比如，要想立刻回收被一个已经删除的字段占据的空间，最快的方法是：

  ```
  ALTER TABLE table ALTER COLUMN anycol TYPE anytype;
  ```

  这里的 anycol 是任何在表中还存在的字段，而 anytype 是和该字段的原类型一样的类型。这样的结果是在表上没有任何可见的语意的变化，但是这个命令强制重写，这样就删除了不再使用的数据。

- **ALTER [COLUMN] column_name { SET DEFAULT expression | DROP DEFAULT }**

  为一个字段设置或者删除缺省值。请注意缺省值只应用于随后的 INSERT 命令，它们不会修改表中已经存在的行。也可以为视图创建缺省，这个时候它们是在视图的 ON INSERT 规则应用之前插入到 INSERT 句中的。

- **ALTER [COLUMN] column_name { SET | DROP } NOT NULL**

  修改一个字段是否允许 NULL 值或者拒绝 NULL 值。如果表在字段中包含非 NULL，则只能使用 SET NOT NULL。

- **ALTER [COLUMN] column_name SET STATISTICS [PERCENT] integer**

  为随后的 ANALYZE 操作设置针对每个字段的统计收集目标。目标的范围可以在 0 到 10000 之内设置。设置为-1 时表示重新恢复到使用系统缺省的统计目标。

- **{ADD | DELETE} STATISTICS ((column_1_name, column_2_name [, ...]))**

  用于添加和删除多列统计信息声明（不实际进行多列统计信息收集），以便在后续进行全表或全库 analyze 时进行多列统计信息收集。如果关闭 GUC 参数 enable_functional_dependency，每组多列统计信息最多支持 32 列；如果开启 GUC 参数 enable_functional_dependency，每组多列统计信息最多支持 4 列。不支持添加/删除多列统计信息声明的表：系统表、外表。

- **ALTER [COLUMN] column_name SET ({attribute_option = value} [, ...])**

  ALTER [COLUMN] column_name RESET (attribute_option [, ...])

  设置/重置属性选项。

  目前，属性选项只定义了 n_distinct 和 n_distinct_inherited。

  n_distinct 影响表本身的统计值，而 n_distinct_inherited 影响表及其继承子表的统计。目前，只支持 SET/RESET n_distinct 参数，禁止 SET/RESET n_distinct_inherited 参数。

- **ALTER [COLUMN] column_name SET STORAGE { PLAIN | EXTERNAL | EXTENDED | MAIN }**

  为一个字段设置存储模式。这个设置控制这个字段是内联保存还是保存在一个附属的表里，以及数据是否要压缩。仅支持对行存表的设置；对列存表没有意义，执行时报错。SET STORAGE 本身并不改变表上的任何东西，只是设置将来的表操作时，建议使用的策略。

❖ 其中列约束 column_constraint 为：

```
[ CONSTRAINT constraint_name ]
  { NOT NULL |
    NULL |
```

```
        CHECK ( expression ) |

        DEFAULT default_expr  |

        GENERATED ALWAYS AS ( generation_expr ) [STORED] |

        UNIQUE index_parameters |

        PRIMARY KEY index_parameters |

        ENCRYPTED WITH ( COLUMN_ENCRYPTION_KEY =  column_encryption_key,
ENCRYPTION_TYPE = encryption_type_value ) |

        REFERENCES reftable [ ( refcolumn ) ] [ MATCH FULL | MATCH PARTIAL |
MATCH SIMPLE ]

          [ ON DELETE action ] [ ON UPDATE action ] }  [ DEFERRABLE | NOT
DEFERRABLE | INITIALLY DEFERRED | INITIALLY IMMEDIATE ]
```

【须知】

GENERATED ALWAYS AS (generation_expr) [STORED]语法仅在 PostgreSQL 兼容模式下可用，更多内容可参考 PostgreSQL 兼容性手册下的 ALTER TABLE。

❖ 其中列的压缩可选项 compress_mode 为：

```
    [ DELTA | PREFIX | DICTIONARY | NUMSTR | NOCOMPRESS ]
```

❖ 其中根据已有唯一索引为表增加主键约束或唯一约束 table_constraint_using_index 为：

```
      [ CONSTRAINT constraint_name ]

        { UNIQUE | PRIMARY KEY } USING INDEX index_name

        [ DEFERRABLE | NOT DEFERRABLE | INITIALLY      DEFERRED | INITIALLY
IMMEDIATE ]
```

❖ 其中表约束 table_constraint 为：

```
    [ CONSTRAINT constraint_name ]
      { CHECK ( expression ) |
        UNIQUE ( column_name [, ... ] ) index_parameters |
        PRIMARY KEY ( column_name [, ... ] ) index_parameters |
        PARTIAL CLUSTER KEY ( column_name [, ... ] ) }
        FOREIGN KEY ( column_name [, ... ] ) REFERENCES reftable [ ( refcolumn
[, ... ] ) ]
          [ MATCH FULL | MATCH PARTIAL | MATCH SIMPLE ] [ ON DELETE action ]
[ ON UPDATE action ] }
      [ DEFERRABLE | NOT DEFERRABLE | INITIALLY DEFERRED | INITIALLY
IMMEDIATE ]
```

❖ 其中索引参数 index_parameters 为：

```
  [ WITH ( {storage_parameter = value} [, ... ] ) ]
      [ USING INDEX TABLESPACE tablespace_name ]
```

❖ 重命名表。对名称的修改不会影响所存储的数据。

```
  ALTER TABLE [ IF EXISTS ] table_name
      RENAME TO new_table_name;
```

❖ 重命名表中指定的列。

```
  ALTER TABLE [ IF EXISTS ] { table_name [*] | ONLY table_name | ONLY
( table_name )}
  RENAME [ COLUMN ] column_name TO new_column_name;
```

❖ 重命名表的约束。

```
ALTER TABLE [ IF EXISTS ] { table_name [*] | ONLY table_name | ONLY
( table_name ) }
RENAME CONSTRAINT constraint_name TO new_constraint_name;
```

❖ 设置表的所属模式。

```
ALTER TABLE [ IF EXISTS ] table_name
SET SCHEMA new_schema;
```

【须知】

❖ 这种形式把表移动到另外一个模式。相关的索引、约束都跟着移动。目前序列不支持改变 schema。若该表拥有序列，需要将序列删除，重建，或者取消拥有关系，才能将表 schema 更改成功。

❖ 要修改一个表的模式，用户必须在新模式上拥有 CREATE 权限。要把该表添加为一个父表的新子表，用户必须同时又是父表的所有者。要修改所有者，用户还必须是新的所有角色的直接或间接成员，并且该成员必须在此表的模式上有 CREATE 权限。这些限制规定了该用户不能做出了重建和删除表之外的事情。不过，系统管理员可以以任何方式修改任意表的所有权限。

❖ 除了 RENAME 和 SET SCHEMA 之外所有动作都可以捆绑在一个经过多次修改的列表中并行使用。比如，可以在一个命令里增加几个字段或修改几个字段的类型。对于大表，此种操作带来的效率提升更明显，原因在于只需要对该大表做一次处理。

❖ 增加一个 CHECK 或 NOT NULL 约束将会扫描该表，以保证现有的行符合约束要求。

❖ 用一个非空缺省值增加一个字段或者改变一个字段的现有类型会重写整个表。对于大表来说，这个操作可能会花很长时间，并且它还临时需要两倍的磁盘空间。

❖ 添加多个列。

```
ALTER TABLE [ IF EXISTS ] table_name
ADD ( { column_name data_type [ compress_mode ] [ COLLATE collation ] [ column_constraint [ ... ] ]} [, ...] );
```

❖ 更新多个列。

```
ALTER TABLE [ IF EXISTS ] table_name
MODIFY ( { column_name data_type | column_name [ CONSTRAINT constraint_name ] NOT NULL [ ENABLE ] | column_name [ CONSTRAINT constraint_name ] NULL } [, ...] );
```

参数说明

❖ **IF EXISTS**

如果不存在相同名称的表，不会抛出一个错误，而会发出一个通知，告知表不存在。

❖ **table_name [*] | ONLY table_name | ONLY (table_name)**

table_name 是需要修改的表名。

若声明了 ONLY 选项，则只有那个表被更改。若未声明 ONLY，该表及其所有子表都将会被更改。另外，可以在表名称后面显示地增加*选项来指定包括子表，即表示所有后代表都被扫描，这是默认行为。

❖ **constraint_name**

要删除的现有约束的名称。

❖ **index_name**

索引名称。

❖ **storage_parameter**

表的存储参数的名称。

创建索引新增一个选项：

- parallel_workers（int 类型）

  取值范围：[1,32]，1 表示关闭并发。

  表示创建索引时启动的 bgworker 线程数量，例如 2 就表示将会启动两个 bgworker 线程并发创建索引。

  如果未设置，启动 bgworker 线程数量与表大小相关，一般不超过 4 个线程。

- cmuids（bool 类型）

  默认值：off

  参数开启：更新表元组时，为元组分配表级唯一标识 id。

❖ **new_owner**

表新拥有者的名称。

❖ **new_tablespace**

表所属新的表空间名称。

❖ **column_name、column_1_name、 column_2_name**

现存的或新字段的名称。

❖ **data_type**

新字段的类型，或者现存字段的新类型。

❖ **compress_mode**

表字段的压缩可选项。该子句指定该字段优先使用的压缩算法。

【须知】

行存表不支持压缩。

❖ **collation**

字段排序规则名称。可选字段 COLLATE 指定了新字段的排序规则，如果省略，排序规则为新字段的默认类型。排序规则可以使用 select * from pg_collation;命令从 pg_collation 系统表中查询，默认的排序规则为查询结果中以 default 开始的行。

❖ **USING expression**

SING 子句声明如何从旧的字段值里计算新的字段值；如果省略，缺省从旧类型向新类型的赋值转换。如果从旧数据类型到新类型没有隐含或者赋值的转换，则必须提供一个 USING 子句。

【须知】

ALTER TYPE 的 USING 选项实际上可以声明涉及该行旧值的任何表达式，即它可以引用除了正在被转换的字段之外其他的字段。这样，就可以用 ALTER TYPE 语法做非常普遍性的转换。因为这个灵活性，USING 表达式并没有作用于该字段的缺省值（如果有的话），结果可能不是缺省表达式要求的常量表达式。这就意味着如果从旧类型到新类型没有隐含或者赋值转换的话，即使存在 USING 子句，ALTER TYPE 也可能无法把缺省值转换成新的类型。在这种情况下，应该用 DROP DEFAULT 先删除缺省，执行 ALTER TYPE，然后使用 SET DEFAULT 增加一个合适的新缺省值。类似的考虑也适用于涉及该字段的索引和约束。

- **NOT NULL | NULL**

  设置列是否允许空值。

- **integer**

  带符号的整数常值。当使用 PERCENT 时表示按照表数据的百分比收集统计信息，integer 的取值范围为 0-100。

- **attribute_option**

  属性选项。

- **PLAIN | EXTERNAL | EXTENDED | MAIN**

  字段存储模式。

  - PLAIN 必需用于定长的数值（比如 integer）并且是内联的、不压缩的。
  - MAIN 用于内联、可压缩的数据。
  - EXTERNAL 用于外部保存、不压缩的数据。使用 EXTERNAL 将令在 text 和 bytea 字段上的子字符串操作更快，但付出的代价是增加了存储空间。
  - EXTENDED 用于外部的压缩数据，EXTENDED 是大多数支持非 PLAIN 存储的数据的缺省。

- **CHECK (expression)**

  每次将要插入的新行或者将要被更新的行必须使表达式结果为真才能成功，否则会抛出一个异常并且不会修改数据库。

  声明为字段约束的检查约束应该只引用该字段的数值，而在表约束里出现的表达式可以引用多个字段。

目前，CHECK 表达式不能包含子查询也不能引用除当前行字段之外的变量。

- **DEFAULT default_expr**

  给字段指定缺省值。

  缺省表达式的数据类型必须和字段类型匹配。

  缺省表达式将被用于任何未声明该字段数值的插入操作。如果没有指定缺省值则缺省值为 NULL 。

- **UNIQUE index_parameters**

  UNIQUE (column_name [, ...]) index_parameters

  UNIQUE 约束表示表里的一个或多个字段的组合必须在全表范围内唯一。

- **PRIMARY KEY index_parameters**

  PRIMARY KEY (column_name [, ...]) index_parameters

  主键约束表明表中的一个或者一些字段只能包含唯一（不重复）的非 NULL 值。

- **REFERENCES reftable [(refcolum)] [MATCH matchtype] [ON DELETE action] [ON UPDATE action] \(column constraint)**

  FOREIGN KEY (column_name [, ...]) REFERENCES reftable [(refcolumn [, ...])] [MATCH matchtype] [ON DELETE action] [ON UPDATE action] \(table constraint)

  外键约束要求新表中一列或多列构成的组应该只包含、匹配被参考表中被参考字段值。若省略 refcolum，则将使用 reftable 的主键。被参考列应该是被参考表中的唯一字段或主键。外键约束不能被定义在临时表和永久表之间。

  参考字段与被参考字段之间存在三种类型匹配，分别是：

  - MATCH FULL：不允许一个多字段外键的字段为 NULL，除非全部外键字段都是 NULL。
  - MATCH SIMPLE（缺省）：允许任意外键字段为 NULL。
  - MATCH PARTIAL：目前暂不支持。

  另外，当被参考表中的数据发生改变时，某些操作也会在新表对应字段的数据上执行。ON DELETE 子句声明当被参考表中的被参考行被删除时要执

行的操作。ON UPDATE 子句声明当被参考表中的被参考字段数据更新时要执行的操作。对于 ON DELETE 子句、ON UPDATE 子句的可能动作：

- NO ACTION（缺省）：删除或更新时，创建一个表明违反外键约束的错误。若约束可推迟，且若仍存在任何引用行，那这个错误将会在检查约束的时候产生。
- RESTRICT：删除或更新时，创建一个表明违反外键约束的错误。与 NO ACTION 相同，只是动作不可推迟。
- CASCADE：删除新表中任何引用了被删除行的行，或更新新表中引用行的字段值为被参考字段的新值。
- SET NULL：设置引用字段为 NULL。
- SET DEFAULT：设置引用字段为它们的缺省值。

❖ **DEFERRABLE | NOT DEFERRABLE | INITIALLY DEFERRED | INITIALLY IMMEDIATE**

设置该约束是否可推迟。

- DEFERRABLE：可以推迟到事务结尾使用 SET CONSTRAINTS 命令检查。
- NOT DEFERRABLE：在每条命令之后马上检查。
- INITIALLY IMMEDIATE：那么每条语句之后就立即检查它。
- INITIALLY DEFERRED：只有在事务结尾才检查它。

【须知】

Ustore 表不支持新增 DEFERRABLE 以及 INITIALLY DEFERRED 约束。

❖ **PARTIAL CLUSTER KEY**

局部聚簇存储，列存表导入数据时按照指定的列(单列或多列)，进行局部排序。

❖ **WITH ({storage_parameter = value} [, ...])**

为表或索引指定一个可选的存储参数。

❖ **tablespace_name**

索引所在表空间的名称。

❖ **COMPRESS|NOCOMPRESS**

- NOCOMPRESS：如果指定关键字 NOCOMPRESS 则不会修改表的现有压缩特性。

➢ COMPRESS：如果指定 COMPRESS 关键字，则对该表进行批量插入元组时触发该特性。行存表不支持压缩。

❖ **new_table_name**

修改后新的表名称。

❖ **new_column_name**

表中指定列修改后新的列名称。

❖ **new_constraint_name**

修改后表约束的新名称。

❖ **new_schema**

修改后新的模式名称。

❖ **CASCADE**

级联删除依赖于被依赖字段或者约束的对象（比如引用该字段的视图）。

❖ **RESTRICT**

如果字段或者约束还有任何依赖的对象，则拒绝删除该字段。这是缺省行为。

❖ **schema_name**

表所在的模式名称。

示例

请参考 CREATE TABLE 的示例 20。

4.5.36.ALTER TABLESPACE

功能描述

修改表空间的属性。

注意事项

❖ 只有表空间的所有者或者被授予了表空间 ALTER 权限的用户有权限执行 ALTER TABLESPACE 命令，系统管理员默认拥有此权限。但要修改表空间的所有者，当前用户必须是该表空间的所有者或系统管理员，且该用户是新所有者角色的成员。

- 要修改表空间的所有者 A 为 B，则 A 必须是 B 的直接或者间接成员。

  说明：

  如果 new_owner 与 old_owner 一致，此处不再校验当前执行操作的用户是否具有修改权限，而直接显示 ALTER 成功。

语法格式

- 重命名表空间的语法。

```
ALTER TABLESPACE tablespace_name
  RENAME TO new_tablespace_name;
```

- 设置表空间所有者的语法。

```
ALTER TABLESPACE tablespace_name
  OWNER TO new_owner;
```

- 设置表空间属性的语法。

```
ALTER TABLESPACE tablespace_name
  SET ( {tablespace_option = value} [, ... ] );
```

- 重置表空间属性的语法。

```
ALTER TABLESPACE tablespace_name
  RESET ( { tablespace_option } [, ...] );
```

- 设置表空间限额的语法。

```
ALTER TABLESPACE tablespace_name
  RESIZE MAXSIZE { UNLIMITED | 'space_size'};
```

参数说明

- **tablespace_name**

  要修改的表空间。

  取值范围：已存在的表空间名。

- **new_tablespace_name**

  表空间的新名称。

  新名称不能以“PG_”开头。

  取值范围：字符串，符合标识符命名规范。

- **new_owner**

  表空间的新所有者。

取值范围：已存在的用户名。

- **tablespace_option**

设置或者重置表空间的参数。

取值范围：

- seq_page_cost：设置优化器计算一次顺序获取磁盘页面的开销。缺省为 1.0。
- random_page_cost：设置优化器计算一次非顺序获取磁盘页面的开销。缺省为 4.0。

说明：

- random_page_cost 是相对于 seq_page_cost 的取值，等于或者小于 seq_page_cost 时毫无意义。
- 默认值为 4.0 的前提条件是，优化器采用索引来扫描表数据，并且表数据在 cache 中命中率可以 90%左右。
- 如果表数据空间要比物理内存小，那么减小该值到一个适当水平；相反地，如果表数据在 cache 中命中率要低于 90%，那么适当增大该值。
- 如果采用了类似于 SSD 的随机访问代价较小的存储器，可以适当减小该值，以反映真正的随机扫描代价。

value 的取值范围：正的浮点类型。

- **RESIZE MAXSIZE**

重新设置表空间限额的数值。

取值范围：

- UNLIMITED，该表空间不设置限额。
- 由 space_size 来确定，其格式参考 CREATE TABLESPACE(参见：开发者指南->SQL 语法参考->SQL 语法->CREATE TABLESPACE 章节)。

说明：

- 若调整后的限额值比当前表空间实际使用的值要小，调整操作可以执行成功，后续用户需要将该表空间的使用值降低到新限额值之下，才能继续往该表空间中写入数据。

❖ 修改参数 MAXSIZE 时也可使用：

```
ALTER TABLESPACE tablespace_name RESIZE MAXSIZE
{ 'UNLIMITED' | 'space_size'};
```

示例

请参考 CREATE TABLESPACE （参见：开发者指南->SQL 语法参考->SQL 语法->CREATE TABLESPACE 章节）的示例。

相关链接

CREATE TABLESPACE (参见：开发者指南->SQL 语法参考->SQL 语法->CREATE TABLESPACE 章节)，DROP TABLESPACE (参见：开发者指南->SQL 语法参考->SQL 语法->DROP TABLESPACE 章节)

4.5.37.ALTER TABLE PARTITION

功能描述

修改表分区，包括增加/删除分区、切割/合并分区、清空分区、移动分区表空间、交换分区、重命名分区，以及修改分区属性等。

语法格式

修改表分区主语法如下：

```
    ALTER TABLE [ IF EXISTS ] { table_name  [*] | ONLY table_name | ONLY
( table_name  )}
       action [, ... ];
```

其中 action 统指如下分区维护子语法。当存在多个分区维护子句时，保证了分区的连续性，无论这些子句的排序如何，PanWeiDB 总会先执行 DROP PARTITION 再执行 ADD PARTITION 操作，最后顺序执行其他分区维护操作。

```
    move_clause |
    exchange_clause |
    row_clause |
```

```
    merge_clause |
    modify_clause |
    split_clause |
    add_clause |
    drop_clause |
    truncate_clause
```

- move_clause 子语法用于移动分区到新的表空间。

```
    MOVE PARTITION { partion_name | FOR ( partition_value [, ...] ) } TABLESPACE
tablespacename
```

- exchange_clause 子语法用于把普通表的数据迁移到指定的分区。

```
    EXCHANGE PARTITION|SUBPARTITION { ( partition_name ) | FOR
( partition_value [, ...] ) }
    WITH TABLE {[ ONLY ] ordinary_table_name | ordinary_table_name * | ONLY
( ordinary_table_name )}
    [ { WITH | WITHOUT } VALIDATION ] [ VERBOSE ]
```

进行交换的普通表和分区必须满足如下条件：

- 普通表和分区的列数目相同，对应列的信息严格一致，包括：列名、列的数据类型、列约束、列的 Collation 信息、列的存储参数、列的压缩信息等。
- 普通表和分区的表压缩信息严格一致。
- 普通表和分区的索引个数相同，且对应索引的信息严格一致。
- 普通表和分区的表约束个数相同，且对应表约束的信息严格一致。
- 普通表不可以是临时表，分区表只能是范围分区表，列表分区表，哈希分区表。

- 普通表和分区表上不可以有动态数据脱敏，行访问控制约束。
- 列表分区表，哈希分区表不能是列存储。
- List/Hash/Range 类型分区表支持 exchange_clause。

【须知】

- 完成交换后，普通表和分区的数据被置换，同时普通表和分区的表空间信息被置换。此时，普通表和分区的统计信息变得不可靠，需要对普通表和分区重新执行 analyze。
- 由于非分区键不能建立本地唯一索引，只能建立全局唯一索引，所以如果普通表含有唯一索引时，会导致不能交换数据。
- 如果在普通表/分区表上进行了 drop column 操作，被删除的列依然物理存在，所以需要保证普通表和分区的被删除列也严格对齐才能交换成功。
- 支持增删列之后的表进行分区交换。

- row_clause 子语法用于设置分区表的行迁移开关。

```
{ ENABLE | DISABLE } ROW MOVEMENT
```

- merge_clause 子语法用于把多个分区合并成一个分区。

```
MERGE PARTITIONS { partition_name } [, ...] INTO PARTITION partition_name
[ TABLESPACE tablespacename ] [ UPDATE GLOBAL INDEX ]
```

- modify_clause 子语法用于设置分区索引是否可用。

```
MODIFY PARTITION partition_name { UNUSABLE LOCAL INDEXES | REBUILD
UNUSABLE LOCAL INDEXES }
```

- split_clause 子语法用于把一个分区切割成多个分区。

```
SPLIT PARTITION { partition_name | FOR ( partition_value [, ...] ) }
{ split_point_clause | no_split_point_clause } [ UPDATE GLOBAL INDEX ]
```

❖ 指定切割点 split_point_clause 的语法为。

```
AT ( partition_value ) INTO ( PARTITION partition_name [ TABLESPACE tablespacename ] , PARTITION partition_name [ TABLESPACE tablespacename ] )
```

【须知】

❖ 列存分区表不支持切割分区。

❖ 切割点的大小要位于正在被切割的分区的分区键范围内，指定切割点的方式只能把一个分区切割成两个新分区。

❖ 不指定切割点 no_split_point_clause 的语法为：

```
INTO { ( partition_less_than_item [, ...] ) | ( partition_start_end_item [, ...] ) }
```

【须知】

❖ 不指定切割点的方式，partition_less_than_item 指定的第一个新分区的分区键要大于正在被切割的分区的前一个分区（如果存在的话）的分区键，partition_less_than_item 指定的最后一个分区的分区键要等于正在被切割的分区的分区键大小。

❖ 不指定切割点的方式，partition_start_end_item 指定的第一个新分区的起始点（如果存在的话）必须等于正在被切割的分区的前一个分区（如果存在的话）的分区键，partition_start_end_item 指定的最后一个分区的终止点（如果存在的话）必须等于正在被切割的分区的分区键。

❖ partition_less_than_item 支持的分区键个数最多为 4，而 partition_start_end_item 仅支持 1 个分区键，其支持的数据类型参见《PanWeiDB V2.0-S3.0.1_B01 兼容性手册》->PARTITION BY RANGE。

❖ 在同一语句中 partition_less_than_item 和 partition_start_end_item 两者不可同时使用；不同 split 语句之间没有限制。

❖ 分区项 partition_less_than_item 的语法为：

```
PARTITION partition_name VALUES LESS THAN ( { partition_value | MAXVALUE } [, ...] ) [ TABLESPACE tablespacename ]
```

❖ 分区项 partition_start_end_item 的语法为，其约束参见《PanWeiDB V2.0-S3.0.1_B01 开发者指南》->START END 语法描述。

```
PARTITION partition_name {{START(partition_value) END (partition_value) EVERY (interval_value)} |{START(partition_value) END ({partition_value | MAXVALUE})} |{START(partition_value)} |{END({partition_value |MAXVALUE})}} [TABLESPACE tablespace_name]
```

❖ add_clause 子语法用于为指定的分区表添加一个或多个分区。

```
ADD PARTITION ( partition_col1_name = partition_col1_value [, partition_col2_name = partition_col2_value ] [, ...] )
    [ LOCATION 'location1' ]
    [ PARTITION (partition_colA_name = partition_colA_value [, partition_colB_name = partition_colB_value ] [, ...] ) ]
    [ LOCATION 'location2' ]

ADD {partition_less_than_item | partition_start_end_item| partition_list_item }
```

❖ 分区项 partition_list_item 的语法如下。

```
PARTITION partition_name VALUES (list_values_clause) [ TABLESPACE tablespacename ]
```

【须知】

❖ partition_list_item 仅支持的 1 个分区键，其支持的数据类型参见《PanWeiDB V2.0-S3.0.1_B01 开发者指南》->PARTITION BY LIST(partition_key)。

❖ 间隔/哈希分区表不支持添加分区。

❖ drop_clause 子语法用于删除分区表中的指定分区。

【须知】
哈希分区表不支持删除分区。

❖ truncate_clause 子语法用于清空分区表中的指定分区。

```
TRUNCATE PARTITION { partition_name | FOR ( partition_value [, ...] ) }
[ UPDATE GLOBAL INDEX ]
```

❖ 修改表分区名称的语法。

```
ALTER TABLE [ IF EXISTS ] { table_name [*] | ONLY table_name | ONLY
( table_name )}

RENAME PARTITION { partion_name | FOR ( partition_value [, ...] ) } TO
partition_new_name;
```

❖ 在 PostgreSQL 兼容模式下，支持使用 ALTER TABLE ATTACH PARTITION 把普通表加入到分区表中。详见 PostgreSQL 兼容性手册中的 ALTER TABLE PRATITION。

```
ALTER TABLE table_name attach_partiotion_cmd

where attach_partiotion_cmd can be:

ATTACH PARTITION table_name FOR VALUES

{ IN ( partition_value [, ...] ) |

 FROM ( { partition_value | MINVALUE | MAXVALUE } [, ...] )

 TO ( { partition_value | MINVALUE | MAXVALUE } [, ...] ) |

 WITH ( MODULUS numeric_literal, REMAINDER numeric_literal ) }
```

参数说明

- **table_name**

  分区表名。

  取值范围：已存在的分区表名。

- **partition_name**

  分区名。

  取值范围：已存在的分区名。

- **tablespacename**

  指定分区要移动到哪个表空间。

  取值范围：已存在的表空间名。

- **partition_value**

  分区键值。

  通过 PARTITION FOR (partition_value [, ...])子句指定的这一组值，可以唯一确定一个分区。

  取值范围：需要进行重命名的分区的分区键的取值范围。

- **UNUSABLE LOCAL INDEXES**

  设置该分区上的所有索引不可用。

- **REBUILD UNUSABLE LOCAL INDEXES**

  重建该分区上的所有索引。

- **ENABLE/DISABLE ROW MOVEMET**

  行迁移开关。

  如果进行 UPDATE 操作时，更新了元组在分区键上的值，造成了该元组所在分区发生变化，就会根据该开关给出报错信息，或者进行元组在分区间的转移。

  取值范围：

  - ENABLE：打开行迁移开关。
  - DISABLE：关闭行迁移开关。

  默认是打开状态。

- **ordinary_table_name**

进行迁移的普通表的名称。

取值范围：已存在的普通表名。

- **{ WITH | WITHOUT } VALIDATION**

  在进行数据迁移时，是否检查普通表中的数据满足指定分区的分区键范围。

  取值范围：

  - WITH：对于普通表中的数据要检查是否满足分区的分区键范围，如果有数据不满足，则报错。
  - WITHOUT：对于普通表中的数据不检查是否满足分区的分区键范围。

  默认是 WITH 状态。

  由于检查比较耗时，特别是当数据量很大的情况下更甚。所以在保证当前普通表中的数据满足分区的分区键范围时，可以加上 WITHOUT 来指明不进行检查。

- **VERBOSE**

  在 VALIDATION 是 WITH 状态时，如果检查出普通表有不满足要交换分区的分区键范围的数据，那么把这些数据插入到正确的分区，如果路由不到任何分区，再报错。

  【须知】

  只有在 VALIDATION 是 WITH 状态时，才可以指定 VERBOSE。

- **partition_new_name**

  分区的新名称。

  取值范围：字符串，要符合标识符的命名规范。

注意事项

- 添加分区的表空间不能是 PG_GLOBAL。
- 添加分区的名称不能与该分区表已有分区的名称相同。
- 添加分区的分区键值要和分区表的分区键的类型一致。
- 若添加 RANGE 分区，添加分区键值要大于分区表中最后一个范围分区的上边界。
- 若添加 LIST 分区，添加分区键值不能与现有分区键值重复。
- 不支持添加 HASH 分区。

- 如果目标分区表中已有分区数达到了最大值 1048575，则不能继续添加分区。
- 当分区表只有一个分区时，不能删除该分区。
- 选择分区使用 PARTITION FOR()，括号里指定值个数应该与定义分区时使用的列个数相同，并且一一对应。
- Value 分区表不支持相应的 Alter Partition 操作。
- 列存分区表不支持切割分区。
- 间隔分区表不支持添加分区。
- 哈希分区表不支持切割分区，不支持合成分区，不支持添加和删除分区。
- 列表分区表不支持切割分区，不支持合成分区。
- 只有分区表的所有者或者被授予了分区表 ALTER 权限的用户有权限执行 ALTER TABLE PARTITION 命令，系统管理员默认拥有此权限。
- 删除、切割、合并、清空、交换分区的操作会使 Global 索引失效，可以申明 UPDATE GLOBAL INDEX 子句同步更新索引。
- 如果删除、切割、合并、清空、交换分区操作不申明 UPDATE GLOBAL INDEX 子句，并发的 DML 业务有可能因为索引不可用而报错。

示例

【须知】

此处仅展示 exchange_clause 子语法的示例，其余更多示例请参考 CREATE TABLE PARTITION 的示例。

示例 1：分区交换。

1、创建分区表。

```
create table t_range_list
( id number,
partition_key int,
subpartition_key int,
```

```
col2 varchar2(10) )
partition by range(partition_key)
subpartition by list(subpartition_key)(
partition p1 values less than (100)
(subpartition sub_1_1 values (10),
subpartition sub_1_2 values (20)),
partition p2 values less than(200)
(subpartition sub_2_1 values (10),
subpartition sub_2_2 values (20)),
partition p3 values less than (300)
(subpartition sub_3_1 values (10),
subpartition sub_3_2 values (20)));
```

2、插入测试数据。

```
INSERT INTO t_range_list VALUES(1,50,10,'sub_1_1');
INSERT INTO t_range_list VALUES(2,150,20,'sub_2_2');
INSERT INTO t_range_list VALUES(3,250,10,'sub_3_1');
```

3、创建用于交换的普通表。

```
create table t_exchange
( id number,
partition_key int,
subpartition_key int,
col2 varchar2(10) );
```

4、为普通表插入数据。

```
INSERT INTO t_exchange VALUES(1,40,10,'sub_1_1');
```

5、将分区 sub_1_1 的数据与普通表的数据进行交换。

```
ALTER TABLE t_range_list exchange subpartition sub_1_1 with table
t_exchange;
```

6、查看交换后的普通表数据。

```
select * from t_exchange;
```

返回结果为：

```
 id | partition_key | subpartition_key | col2
----+---------------+------------------+---------
  1 |            50 |               10 | sub_1_1
(1 row)
```

7、查看交换后分区表的数据。

```
select * from t_range_list;
```

返回结果为

```
 id | partition_key | subpartition_key | col2
----+---------------+------------------+---------
  1 |            40 |               10 | sub_1_1
  2 |           150 |               20 | sub_2_2
```

```
 3 |       250 |          10 | sub_3_1
(3 rows)
```

示例 2：进行增删列操作之后的分区交换。

1、创建 range 分区表。

```
CREATE TABLE e (
id INT NOT NULL,
fname VARCHAR(30),
lname VARCHAR(30) )
PARTITION BY RANGE (id)
( PARTITION p0 VALUES LESS THAN (50),
PARTITION p1 VALUES LESS THAN (100),
PARTITION p2 VALUES LESS THAN (150),
PARTITION p3 VALUES LESS THAN (MAXVALUE) );
```

2、向分区表中插入测试数据。

```
INSERT INTO e VALUES (1669, 'Jim', 'Smith'), (337, 'Mary', 'Jones'), (16, 'Frank', 'White'), (2005, 'Linda', 'Black');
```

3、创建普通表，比步骤 1 中创建的分区表少一列 lname。

```
CREATE TABLE e2 (
id INT NOT NULL,
fname VARCHAR(30));
```

4、为普通表新增一列 lname，但数据类型和分区表中的 lname 列不同。

```
alter table e2 add column lname text;
```

5、交换分区。

```
alter table e exchange partition p3 with table e2;
```

返回结果如下，交换失败（同名列 lname 在普通表和分区表中数据类型不同）。

```
ERROR:  column type or size mismatch in ALTER TABLE EXCHANGE PARTITION
```

6、删除普通表中的列 lname。

```
alter table e2 drop lname;
```

7、为普通表再次新增列 lname，数据类型与分区表中的列 lname 相同。

```
alter table e2 add column lname VARCHAR(30);
```

8、执行分区交换，交换成功。

```
alter table e exchange partition p3 with table e2;
```

9、查看此时的分区表 e 和普通表 e2 的数据。

```
select * from e;

select * from e2;
```

返回结果为：

```
 id | fname | lname

----+-------+-------
```

```
 16 | Frank | White
(1 row)

 id  | fname | lname
------+-------+-------
 1669 | Jim   | Smith
 337 | Mary  | Jones
 2005 | Linda | Black
(3 rows)
```

4.5.38.ALTER TABLE SUBPARTITION

功能描述

修改二级分区表分区，包括增删分区、清空分区、切割分区、移动分区表空间、交换分区、重命名分区，以及修改分区属性等。

注意事项

- 目前二级分区表只支持增删分区、清空分区、切割分区。
- 添加分区的表空间不能是 PG_GLOBAL。
- 添加分区的名称不能与该分区表已有一级分区和二级分区的名称相同。
- 添加分区的分区键值要和分区表的分区键的类型一致。
- 若添加 Range 分区，添加分区键值要大于分区表中最后一个范围分区的上边界。若需要在有 MAXVALUE 分区的表上新增分区，建议使用 SPLIT 语法。
- 若添加 List 分区，添加分区键值不能与现有分区键值重复。若需要在有 DEFAULT 分区的表上新增分区，建议使用 SPLIT 语法。
- 不支持添加 Hash 分区。只有一种情况例外，二级分区表的二级分区方式为 Hash 且一级分区方式不是 Hash，此时支持新增一级分区并创建对应的二级分区。

- 如果目标分区表中已有分区数达到了最大值 1048575，则不能继续添加分区。
- 当分区表只有一个一级分区或二级分区时，不能删除该分区。
- 不支持删除 Hash 分区。
- 选择分区使用 PARTITION FOR()或 SUBPARTITION FOR()，括号里指定值个数应该与定义分区时使用的列个数相同，并且一一对应。
- 切割分区只能对二级分区（叶子节点）进行切割，被切割分区只能是 Range、List 分区策略，不支持切割 Hash 分区策略。List 分区策略只能是 Default 分区才能被切割。
- 只有分区表的所有者或者被授予了分区表 ALTER 权限的用户有权限执行 ALTER TABLE PARTITION 命令，系统管理员默认拥有此权限。
- 删除、切割、合并、清空、交换分区的操作会使 Global 索引失效，可以申明 UPDATE GLOBAL INDEX 子句同步更新索引
- 如果删除、切割、合并、清空、交换分区操作不申明 UPDATE GLOBAL INDEX 子句，并发的 DML 业务有可能因为索引不可用而报错。

语法格式

修改表分区主语法：

```
ALTER TABLE [ IF EXISTS ] { table_name [*] | ONLY table_name | ONLY ( table_name ) }
  action [, ... ];
```

其中 action 统指如下分区维护子语法：

```
dd_clause    |
drop_clause  |
split_clause |
truncate_clause
```

add_clause 子语法用于为指定的分区表添加一个或多个分区。

语法可以作用在一级分区上：

```
ADD {partition_less_than_item | partition_list_item } [ ( subpartition_definition_list ) ]
```

也可以作用在二级分区上：

```
MODIFY PARTITION partition_name ADD subpartition_definition
```

其中，分区项 partition_less_than_item 为 Range 分区定义语法，具体语法如下：

```
PARTITION partition_name VALUES LESS THAN ( partition_value | MAXVALUE ) [ TABL
ESPACE tablespacename ]
```

分区项 partition_list_item 为 List 分区定义语法，具体语法如下：

```
PARTITION partition_name VALUES ( partition_value [, ...] | DEFAULT ) [ TABLESPACE t
ablespacename ]
```

subpartition_definition_list 为 1 到多个二级分区 subpartition_definition 对象，subpartition_definition 具体语法如下：

```
SUBPARTITION subpartition_name { VALUES LESS THAN ( partition_value | MAXVALU
E ) | VALUES ( partition_value [, ...] | DEFAULT )} [ TABLESPACE tablespace ]
```

【须知】

若一级分区为 Hash 分区，不支持以 ADD 形式新增一级分区；若二级分区为 Hash 分区，不支持以 MODIFY 形式新增二级分区。

drop_clause 子语法用于删除分区表中的指定分区。

语法可以作用在一级分区上：

```
DROP PARTITION { partition_name | FOR ( partition_value ) } [ UPDATE GLOBAL IND
EX ]
```

也可以作用在二级分区上：

```
DROP SUBPARTITION { subpartition_name | FOR ( partition_value, subpartition_val
ue ) } [ UPDATE GLOBAL INDEX ]
```

【须知】

- 若一级分区为 Hash 分区，不支持删除一级分区；若二级分区为 Hash 分区，不支持删除二级分区。
- 不支持删除唯一子分区。

split_clause 子语法用于把一个分区切割成多个分区：

```
SPLIT SUBPARTITION { subpartition_name} { split_point_clause } [ UPDATE GLOBAL I
NDEX ]
```

指定切割点 split_point_clause 的语法为：

```
AT ( subpartition_value ) INTO ( SUBPARTITION subpartition_name [ TABLESPACE ta
blespacename ] , SUBPARTITION subpartition_name [ TABLESPACE tablespacename
] )
```

【须知】

- 切割点的大小要位于正在被切割的分区的分区键范围内。
- 只能把一个分区切割成两个新分区。

指定 Range 分区策略切割点 split_point_clause 的语法为：

```
AT ( subpartition_value ) INTO ( SUBPARTITION subpartition_name [ TABLESPACE tablespacename ] , SUBPARTITION subpartition_name [ TABLESPACE tablespacename ] )
```

指定 List 分区策略切割点 split_point_clause 的语法为：

```
VALUES ( subpartition_value ) INTO ( SUBPARTITION subpartition_name [ TABLESPACE tablespacename ] , SUBPARTITION subpartition_name [ TABLESPACE tablespacename ] )
```

【须知】

- 切割点的大小要位于正在被切割的分区的分区键范围内。
- 只能把一个分区切割成两个新分区。
- Range 分区策略切割点是把当前分区以此切割点分割为两个分区（小于此分割点为一个分区，大于此分割点为另一个分区），所以 Range 分区策略切割点只能为一个。List 分区策略切割点可以为多个，但不超过 64 个，即把这些切割点从当前分区的边界值提取出来作为一个新分区，当前分区剩余边界值作为另一个新分区。

truncate_clause 子语法用于清空分区表中的指定分区：

```
TRUNCATE SUBPARTITION { subpartition_name } [ UPDATE GLOBAL INDEX ]
```

参数说明

- **table_name**

  分区表名。

  取值范围：已存在的分区表名。

- **subpartition_name**

  二级分区名。

  取值范围：已存在的二级分区名。

- **tablespacename**

  指定分区要移动到哪个表空间。

  取值范围：已存在的表空间名。

示例

1、创建一个分区表并插入数据。

```
create table t_part_list_range
( id number not null,
 partition_key int,
 subpartition_key int,
 col2 varchar2(10)
)
partition by list(partition_key)
subpartition by range(subpartition_key)
(
 partition t_partition_01 values (100)
 (subpartition sub_1_1 values less than (10),
 subpartition sub_1_2 values less than (20)
 ),
 partition t_partition_02 values (200)
 (subpartition sub_2_1 values less than (10),
 subpartition sub_2_2 values less than (20)
 )
);
insert into t_part_list_range values(1,100,5,'sub_1_1');
insert into t_part_list_range values(2,100,15,'sub_1_2');
insert into t_part_list_range values(3,200,5,'sub_2_1');
insert into t_part_list_range values(4,200,15,'sub_2_2');
insert into t_part_list_range values(5,200,16,'sub_2_2');
```

2、查询数据。

```
select * from t_part_list_range subpartition for (100,5);
```

返回结果为：

```
 id | partition_key | subpartition_key |  col2
----+---------------+------------------+---------
  1 |           100 |                5 | sub_1_1
(1 row)
```

3、新增一级与二级分区。

```
alter table t_part_list_range add partition t_partition_03 values (300)
( subpartition sub_3_1 values less than (10),
 subpartition sub_3_2 values less than (20)
);
```

4、删除指定一级分区包括属于它的所有二级分区。

```
alter table t_part_list_range drop partition t_partition_02;
```

5、为指定一级分区新增二级分区。

```
alter table t_part_list_range modify partition t_partition_01 add subpartition sub_1
_3 values less than (30);
```

6、删除指定二级分区。

```
alter table t_part_list_range drop subpartition sub_1_3;
```

4.5.39.ALTER TEXT SEARCH CONFIGURATION

功能描述

更改文本搜索配置的定义。用户可以将映射从字串类型调整为字典，或者改变配置的名称或者所有者，或者修改搜索配置的配置参数。

ADD MAPPING FOR 选项为文本搜索配置增加字串类型映射；如果 ADD MAPPING FOR 后面任何一个字串类型的映射已经存在于此文本搜索配置中，那么系统将会报错。

ALTER MAPPING FOR 选项会首先清除已有的字串类型映射，然后添加指定的字串类型映射。

ALTER MAPPING REPLACE ... WITH ... 与 ALTER MAPPING FOR ... REPLACE ... WITH ...选项会直接使用 new_dictionary 替换 old_dictionary。需要注意的是，只有 pg_ts_config_map 系统表中存在 maptokentype 与 old_dictionary 对应关系的元组时，才能更新成功，否则不会成功，也不会有任何提示信息返回。

DROP MAPPING FOR 选项会删除当前文本搜索配置中指定的字串类型映射。如果没有指定 IF EXISTS 选项，当 DROP MAPPING FOR 选项指定的字串类型映射在文本搜索配置中不存在时，数据库会报错。

注意事项

- 当一个搜索配置已经被引用（如被用来创建索引），则不允许用户修改此文本搜索配置。
- 要使用 ALTER TEXT SEARCH CONFIGURATION，用户必须是配置的所有者。

语法格式

- 增加文本搜索配置字串类型映射语法

```
ALTER TEXT SEARCH CONFIGURATION name
  ADD MAPPING FOR token_type [, ... ] WITH dictionary_name [, ... ];
```

- 修改文本搜索配置字典语法

```
ALTER TEXT SEARCH CONFIGURATION name
  ALTER MAPPING FOR token_type [, ... ] REPLACE old_dictionary WITH new_dictiona
ry;
```

- 修改文本搜索配置字串类型语法

```
ALTER TEXT SEARCH CONFIGURATION name
  ALTER MAPPING FOR token_type [, ... ] WITH dictionary_name [, ... ];
```

- 更改文本搜索配置字典语法

```
ALTER TEXT SEARCH CONFIGURATION name
  ALTER MAPPING REPLACE old_dictionary WITH new_dictionary;
```

- 删除文本搜索配置字串类型映射语法

```
ALTER TEXT SEARCH CONFIGURATION name
  DROP MAPPING [ IF EXISTS ] FOR token_type [, ... ];
```

- 重命名文本搜索配置所有者语法

```
ALTER TEXT SEARCH CONFIGURATION name OWNER TO new_owner;
```

- 重命名文本搜索配置名称语法

```
ALTER TEXT SEARCH CONFIGURATION name RENAME TO new_name;
```

- 重命名文本搜索配置命名空间语法

```
ALTER TEXT SEARCH CONFIGURATION name SET SCHEMA new_schema;
```

- 修改文本搜索配置属性语法

```
ALTER TEXT SEARCH CONFIGURATION name SET ( { configuration_option = value } [,
...] );
```

- 重置文本搜索配置属性语法

```
ALTER TEXT SEARCH CONFIGURATION name RESET ( {configuration_option} [, ...] );
```

参数说明

- **name**

  已有文本搜索配置的名称（可以有模式修饰）。

- **token_type**

  与配置的语法解析器关联的字串类型的名称。详细信息参见解析器 (参见：开发者指南->SQL 语法参考->全文检索->解析器章节)。

- **dictionary_name**

  文本搜索字典名称。 如果有多个字典，则它们会按指定的顺序搜索。

- **old_dictionary**

  映身中拟被替换的文本搜索字典名称。

- **new_dictionary**

  替换 old_dictionary 的文本搜索字典的名称。

- **new_owner**

  文本搜索配置的新所有者。

- **new_name**

  文本搜索配置的新名称。

- **new_schema**

  文本搜索配置的新模式名。

- **configuration_option**

  文本搜索配置项。详细信息参见 CREATE TEXT SEARCH CONFIGURATION (参见：开发者指南->SQL 语法参考->SQL 语法->CREATE TEXT SEARCH CONFIGURATION 章节)。

- **value**

  文本搜索配置项的值。

示例

```
--创建文本搜索配置。
panweidb=# CREATE TEXT SEARCH CONFIGURATION english_1 (parser=default);
CREATE TEXT SEARCH CONFIGURATION

--增加文本搜索配置字串类型映射语法。
```

```
panweidb=# ALTER TEXT SEARCH CONFIGURATION english_1 ADD MAPPING FOR wo
rd WITH simple,english_stem;
ALTER TEXT SEARCH CONFIGURATION

--增加文本搜索配置字串类型映射语法。
panweidb=# ALTER TEXT SEARCH CONFIGURATION english_1 ADD MAPPING FOR em
ail WITH english_stem, french_stem;
ALTER TEXT SEARCH CONFIGURATION

--查询文本搜索配置相关信息。
panweidb=# SELECT b.cfgname,a.maptokentype,a.mapseqno,a.mapdict,c.dictnam
e FROM pg_ts_config_map a,pg_ts_config b, pg_ts_dict c WHERE a.mapcfg=b.oid A
ND a.mapdict=c.oid AND b.cfgname='english_1' ORDER BY 1,2,3,4,5;
 cfgname  | maptokentype | mapseqno | mapdict |   dictname
-----------+--------------+----------+---------+--------------
 english_1 |            2 |        1 |    3765 | simple
 english_1 |            2 |        2 |   12960 | english_stem
 english_1 |            4 |        1 |   12960 | english_stem
 english_1 |            4 |        2 |   12964 | french_stem
(4 rows)

--增加文本搜索配置字串类型映射语法。
panweidb=# ALTER TEXT SEARCH CONFIGURATION english_1 ALTER MAPPING REPLA
CE french_stem with german_stem;
ALTER TEXT SEARCH CONFIGURATION

--查询文本搜索配置相关信息。
panweidb=# SELECT b.cfgname,a.maptokentype,a.mapseqno,a.mapdict,c.dictnam
e FROM pg_ts_config_map a,pg_ts_config b, pg_ts_dict c WHERE a.mapcfg=b.oid A
ND a.mapdict=c.oid AND b.cfgname='english_1' ORDER BY 1,2,3,4,5;
 cfgname  | maptokentype | mapseqno | mapdict |   dictname
-----------+--------------+----------+---------+--------------
 english_1 |            2 |        1 |    3765 | simple
 english_1 |            2 |        2 |   12960 | english_stem
 english_1 |            4 |        1 |   12960 | english_stem
```

```
english_1 |          4 |      2 |  12966 | german_stem
(4 rows)
```

请参见 CREATE TEXT SEARCH CONFIGURATION (参见：开发者指南->SQL 语法参考->SQL 语法->CREATE TEXT SEARCH CONFIGURATION 章节)的示例。

相关链接

CREATE TEXT SEARCH CONFIGURATION(参见：开发者指南->SQL 语法参考->SQL 语法->CREATE TEXT SEARCH CONFIGURATION 章节)， DROP TEXT SEARCH CONFIGURATION(参见：开发者指南->SQL 语法参考->SQL 语法->DROP TEXT SEARCH CONFIGURATION 章节)

4.5.40.ALTER TEXT SEARCH DICTIONARY

功能描述

修改全文检索词典的相关定义，包括参数、名称、所有者以及模式等。

注意事项

- 预定义词典不支持 ALTER 操作。
- 只有词典的所有者可以执行 ALTER 操作，系统管理员默认拥有此权限。
- 创建或修改词典之后，任何对于 filepath 路径下用户自定义的词典定义文件的修改，将不会影响到数据库中的词典。如果需要在数据库中使用这些修改，需使用 ALTER TEXT SEARCH DICTIONARY 语句更新对应词典的定义文件。

语法格式

- 修改词典定义。

```
ALTER TEXT SEARCH DICTIONARY name (
    option [ = value ] [, ... ]
);
```

- 重命名词典。

```
ALTER TEXT SEARCH DICTIONARY name RENAME TO new_name;
```

- 设置词典的所属模式。

```
ALTER TEXT SEARCH DICTIONARY name SET SCHEMA new_schema;
```

❖ 修改词典的所属者。

```
ALTER TEXT SEARCH DICTIONARY name OWNER TO new_owner;
```

参数说明

❖ **name**

已存在的词典名（可指定模式名，否则默认在当前模式下）。

取值范围：已存在的词典名。

❖ **option**

要修改的参数名。与 template 对应，不同的词典类型具有不同的参数列表，且与指定顺序无关。详细参数说明请见 option(参见：开发者指南->SQL 语法参考->SQL 语法->CREATE TEXT SEARCH CONFIGURATION 章节的 option)。

说明

❖ 不支持修改词典的 TEMPLATE 参数值。

❖ 不支持仅修改 FILEPATH 参数而不修改对应的词典定义文件参数。

❖ 词典定义文件的文件名仅支持小写字母、数据、下划线混合。

❖ **value**

要修改的参数值。如果省略等号（=）和 value，则表示删除该 option 的先前设置，使用默认值。

取值范围：对应 option 定义。

❖ **new_name**

词典的新名称。

取值范围：符合标识符命名规范的字符串，且最大长度不超过 63 个字符。

❖ **new_owner**

词典新的所有者。

取值范围：已存在的用户。

❖ **new_schema**

词典的新模式。

取值范围：已存在的模式。

示例

```
--更改 Snowball 类型字典的停用词定义，其他参数保持不变。
panweidb=# ALTER TEXT SEARCH DICTIONARY my_dict ( StopWords = newrussian, FilePath = 'file:///home/dicts' );

--更改 Snowball 类型字典的 Language 参数，并删除停用词定义。
panweidb=# ALTER TEXT SEARCH DICTIONARY my_dict ( Language = dutch,StopWords );

--更新词典定义，不实际更改任何内容。
panweidb=# ALTER TEXT SEARCH DICTIONARY my_dict ( dummy );
```

相关链接

CREATE TEXT SEARCH DICTIONARY（参见：开发者指南->SQL 语法参考->SQL 语法->CREATE TEXT SEARCH CONFIGURATION 章节），DROP TEXT SEARCH DICTIONARY（参见：开发者指南->SQL 语法参考->SQL 语法->DROP TEXT SEARCH CONFIGURATION 章节）

4.5.41.ALTER TRIGGER

功能描述

修改触发器名称。

说明

目前只支持修改名称。

注意事项

触发器所在表的所有者或者被授予了 ALTER ANY SEQUENCE 权限的用户可以执行 ALTER TRIGGER 操作，系统管理员默认拥有此权限。

语法格式

```
ALTER TRIGGER trigger_name ON table_name RENAME TO new_name;
```

参数说明

- **trigger_name**

  要修改的触发器名称。

  取值范围：已存在的触发器。

- **table_name**

  要修改的触发器所在的表名称。

  取值范围：已存在的含触发器的表。

- **new_name**

  修改后的新名称。

  取值范围：符合标识符命名规范的字符串，最大长度不超过 63 个字符，且不能与所在表上其他触发器同名。

示例

请参见 CREATE TRIGGER (参见：开发者指南->SQL 语法参考->SQL 语法->CREATE TRIGGER 章节)的示例。

相关链接

CREATE TRIGGER (参见：开发者指南->SQL 语法参考->SQL 语法->CREATE TRIGGER 章节)，DROP TRIGGER (参见：开发者指南->SQL 语法参考->SQL 语法->DROP TRIGGER 章节)，ALTER TABLE(参见：开发者指南->SQL 语法参考->SQL 语法->ALTER TABLE 章节)

4.5.42.ALTER TYPE

功能描述

修改一个类型的定义。

注意事项

类型的所有者或者被授予了类型 ALTER 权限的用户或者被授予了 ALTER ANY TYPE 权限的用户可以执行 ALTER TYPE 命令，系统管理员默认拥有此权限。

但要修改类型的所有者或者修改类型的模式，当前用户必须是该类型的所有者或者系统管理员，且该用户是新所有者角色的成员。

语法格式

❖ 修改类型。

```
ALTER TYPE name action [, ... ]
ALTER TYPE name OWNER TO { new_owner | CURRENT_USER | SESSION_USER }
ALTER TYPE name RENAME ATTRIBUTE attribute_name TO new_attribute_name [ CA
SCADE | RESTRICT ]
ALTER TYPE name RENAME TO new_name
ALTER TYPE name SET SCHEMA new_schema
ALTER TYPE name ADD VALUE [ IF NOT EXISTS ] new_enum_value [ { BEFORE | AFTER }
 neighbor_enum_value ]
ALTER TYPE name RENAME VALUE existing_enum_value TO new_enum_value

where action is one of:
   ADD ATTRIBUTE attribute_name data_type [ COLLATE collation ] [ CASCADE | REST
RICT ]
   DROP ATTRIBUTE [ IF EXISTS ] attribute_name [ CASCADE | RESTRICT ]
   ALTER ATTRIBUTE attribute_name [ SET DATA ] TYPE data_type [ COLLATE collation
] [ CASCADE | RESTRICT ]
```

❖ 给复合类型增加新的属性。

```
ALTER TYPE name ADD ATTRIBUTE attribute_name data_type [ COLLATE collation ] [
CASCADE | RESTRICT ]
```

❖ 从复合类型删除一个属性。

```
ALTER TYPE name DROP ATTRIBUTE [ IF EXISTS ] attribute_name [ CASCADE | RESTRIC
T ]
```

❖ 改变一种复合类型中某个属性的类型。

```
ALTER TYPE name ALTER ATTRIBUTE attribute_name [ SET DATA ] TYPE data_type [ C
OLLATE collation ] [ CASCADE | RESTRICT ]
```

❖ 改变类型的所有者。

```
ALTER TYPE name OWNER TO { new_owner | CURRENT_USER | SESSION_USER }
```

❖ 改变类型的名称或是一个复合类型中的一个属性的名称。

```
ALTER TYPE name RENAME TO new_name
```

```
ALTER TYPE name RENAME ATTRIBUTE attribute_name TO new_attribute_name [ CA
SCADE | RESTRICT ]
```

❖ 将类型移至一个新的模式中。

```
ALTER TYPE name SET SCHEMA new_schema
```

❖ 为枚举类型增加一个新值。

```
ALTER TYPE name ADD VALUE [ IF NOT EXISTS ] new_enum_value [ { BEFORE | AFTER }
 neighbor_enum_value ]
```

❖ 重命名枚举类型的一个标签值。

```
ALTER TYPE name RENAME VALUE existing_enum_value TO new_enum_value
```

参数说明

❖ **name**

一个需要修改的现有的类型的名称(可以有模式修饰) 。

❖ **new_name**

该类型的新名称。

❖ **new_owner**

新所有者的用户名 。

❖ **new_schema**

该类型的新模式 。

❖ **attribute_name**

拟增加、更改或删除的属性的名称。

❖ **new_attribute_name**

拟改名的属性的新名称。

❖ **data_type**

拟新增属性的数据类型或是拟更改的属性的新类型名。

❖ **new_enum_value**

枚举类型新增加的标签值，是一个非空的长度不超过 63 个字节的字符串。

❖ **neighbor_enum_value**

一个已有枚举标签值，新值应该被增加在紧接着该枚举值之前或者之后的位置上。

❖ **existing_enum_value**

现有的要重命名的枚举值，是一个非空的长度不超过 63 个字节的字符串

- **CASCADE**

  自动级联更新需更新类型以及相关联的记录和继承它们的子表。

- **RESTRICT**

  如果需联动更新类型是已更新类型的关联记录，则拒绝更新。这是缺省选项。

须知

- ADD ATTRIBUTE、DROP ATTRIBUTE 和 ALTER ATTRIBUTE 选项可以组合成一个列表同时处理。 例如，在一条命令中同时增加几个属性或是更改几个属性的类型是可以实现的。

- 要修改一个类型的模式，必须在新模式上拥有 CREATE 权限。 要修改所有者，必须是新的所有角色的直接或间接成员， 并且该成员必须在此类型的模式上有 CREATE 权限。 （这些限制强制了修改所有者不会做任何通过删除和重建类型不能做的事情。 不过，系统管理员可以以任何方式修改任意类型的所有权。） 要增加一个属性或是修改一个属性的类型，也必须有该类型的 USAGE 权限。

示例

请参考 CREATE TYPE (参见：开发者指南->SQL 语法参考->SQL 语法->CREATE TYPE 章节)的示例。

相关链接

CREATE TYPE(参见：开发者指南->SQL 语法参考->SQL 语法->CREATE TYPE 章节)，DROP TYPE(参见：开发者指南->SQL 语法参考->SQL 语法->DROP TYPE 章节)

4.5.43.ALTER USER

功能描述

修改数据库用户的属性。

注意事项

ALTER USER 中修改的会话参数只针对指定的用户，且在下一次会话中有效。

语法格式

❖ 修改用户的权限等信息。

```
ALTER USER user_name [ [ WITH ] option [ ... ] ];
```

其中 option 子句为：

```
{ CREATEDB | NOCREATEDB }
    | { CREATEROLE | NOCREATEROLE }
    | { INHERIT | NOINHERIT }
    | { AUDITADMIN | NOAUDITADMIN }
    | { SYSADMIN | NOSYSADMIN }
    | {MONADMIN | NOMONADMIN}
    | {OPRADMIN | NOOPRADMIN}
    | {POLADMIN | NOPOLADMIN}
    | { USEFT | NOUSEFT }
    | { LOGIN | NOLOGIN }
    | { REPLICATION | NOREPLICATION }
    | {INDEPENDENT | NOINDEPENDENT}
    | {VCADMIN | NOVCADMIN}
    | {PERSISTENCE | NOPERSISTENCE}
    | CONNECTION LIMIT connlimit
    | [ ENCRYPTED | UNENCRYPTED ] PASSWORD { 'password' [EXPIRED] | DISABLE | EX
PIRED }
    | [ ENCRYPTED | UNENCRYPTED ] IDENTIFIED BY { 'password' [ REPLACE 'old_passw
ord' | EXPIRED ] | DISABLE }
    | VALID BEGIN 'timestamp'
    | VALID UNTIL 'timestamp'
    | RESOURCE POOL 'respool'
    | PERM SPACE 'spacelimit'
    | PGUSER
```

❖ 修改用户名。

```
ALTER USER user_name
```

```
  RENAME TO new_name;
```

❖ 锁定或解锁。

```
ALTER USER user_name
  ACCOUNT { LOCK | UNLOCK };
```

❖ 修改与用户关联的指定会话参数值。

```
ALTER USER user_name
  SET configuration_parameter { { TO | = } { value | DEFAULT } | FROM CURRENT };
```

❖ 重置与用户关联的指定会话参数值。

```
ALTER USER user_name
  RESET { configuration_parameter | ALL };
```

参数说明

❖ **user_name**

现有用户名。

取值范围：已存在的用户名。

❖ **new_password**

新密码。

密码规则如下：

- 不能与当前密码相同。
- 密码默认不少于 8 个字符。
- 不能与用户名及用户名倒序相同。
- 至少包含大写字母（A-Z）、小写字母（a-z）、数字（0-9）、非字母数字字符（限定为~!@#$%^&*()-_=+|[{}];:,<.>/?）四类字符中的三类字符。

取值范围：字符串。

❖ **old_password**

旧密码。

❖ **ACCOUNT LOCK | ACCOUNT UNLOCK**

- ACCOUNT LOCK：锁定帐户，禁止登录数据库。
- ACCOUNT UNLOCK：解锁帐户，允许登录数据库。

❖ **PGUSER**

当前版本不允许修改用户的 PGUSER 属性。

其他参数请参见 CREATE ROLE (参见：开发者指南->SQL 语法参考->SQL 语法->CREATE ROLE 章节)和 ALTER ROLE (参见：开发者指南->SQL 语法参考->SQL 语法->ALTER ROLE 章节)的参数说明。

示例

请参考 CREATE USER(参见：开发者指南->SQL 语法参考->SQL 语法->CREATE USER 章节)的示例。

相关链接

CREATE ROLE (参见：开发者指南->SQL 语法参考->SQL 语法->CREATE ROLE 章节)，CREATE USER(参见：开发者指南->SQL 语法参考->SQL 语法->CREATE USER 章节)，DROP USER(参见：开发者指南->SQL 语法参考->SQL 语法->DROP USER 章节)

4.5.44.ALTER USER MAPPING

功能描述

更改一个用户映射的定义。

注意事项

当在 OPTIONS 中出现 password 选项时，需要保证 panweidb 每个节点的 $GAUSSHOME/bin 目录下存在 usermapping.key.cipher 和 usermapping.key.rand 文件，如果不存在这两个文件，请使用 pw_guc 工具生成并发布到每个节点的$GAUSSHOME/bin 目录下。

语法格式

```
ALTER USER MAPPING FOR { user_name | USER | CURRENT_USER | PUBLIC }
  SERVER server_name
  OPTIONS ( [ ADD | SET | DROP ] option ['value'] [, ... ] )
```

在 OPTIONS 选项里，ADD、SET 和 DROP 指定要执行的操作，未指定时默认为 ADD 操作。option 和 value 为对应操作的参数及参数值。

参数说明

- **user_name**

该映射的用户名。

CURRENT_USER 和 USER 匹配当前用户的名称。PUBLIC 被用来匹配系统中所有当前以及未来的用户名。

- **server_name**

该用户映射的服务器名。

- **OPTIONS**

为该用户映射更改选项。新选项会覆盖任何之前指定的选项。ADD、SET 和 DROP 指定要被执行的动作。如果没有显式地指定操作，将假定为 ADD。选项名称必须为唯一，该服务器的外部数据包装器也会验证选项。

  - oracle_fdw 支持的 options 包括：
    - **user**

      oracle server 的用户名。
    - **password**

      oracle 用户对应的密码。
  - mysql_fdw 支持的 options 包括：
    - **username**

      MySQL Server/MariaDB 的用户名。
    - **password**

      MySQL Server/MariaDB 用户对应的密码。
  - postgres_fdw 支持的 options 包括：
    - **user**

      远端 panweidb 数据库用户的用户名。
    - **password**

      远端 panweidb 数据库用户对应的密码。

相关链接

CREATE USER MAPPING (参见：开发者指南->SQL 语法参考->SQL 语法->CREATE USER MAPPING 章节)，DROP USER MAPPING (参见：开发者指南->SQL 语法参考->SQL 语法->DROP USER MAPPING 章节)

4.5.45.ALTER VIEW

功能描述

ALTER VIEW 更改视图的各种辅助属性。（如果用户是更改视图的查询定义，要使用 CREATE OR REPLACE VIEW。）

注意事项

只有视图的所有者或者被授予了视图 ALTER 权限的用户才可以执行 ALTER VIEW 命令，系统管理员默认拥有该权限。针对所要修改属性的不同，对其还有以下权限约束：

- 修改视图的模式，当前用户必须是视图的所有者或者系统管理员，且要有新模式的 CREATE 权限，且不能与新模式中已存在的 synonym 产生命名冲突。
- 修改视图的所有者，当前用户必须是视图的所有者或者系统管理员，且该用户必须是新所有者角色的成员，并且此角色必须有视图所在模式的 CREATE 权限。
- 修改视图的命名，不能与当前模式中已存在的 synonym 产生命名冲突。

语法格式

- 设置视图列的默认值。

```
ALTER VIEW [ IF EXISTS ] view_name
  ALTER [ COLUMN ] column_name SET DEFAULT expression;
```

- 取消列视图列的默认值。

```
ALTER VIEW [ IF EXISTS ] view_name
  ALTER [ COLUMN ] column_name DROP DEFAULT;
```

- 修改视图的所有者。

```
ALTER VIEW [ IF EXISTS ] view_name
  OWNER TO new_owner;
```

- 重命名视图。

```
ALTER VIEW [ IF EXISTS ] view_name
  RENAME TO new_name;
```

- 设置视图的所属模式。

```
ALTER VIEW [ IF EXISTS ] view_name
  SET SCHEMA new_schema;
```

❖ 设置视图的选项。

```
ALTER VIEW [ IF EXISTS ] view_name
  SET ( { view_option_name [ = view_option_value ] } [, ... ] );
```

❖ 重置视图的选项。

```
ALTER VIEW [ IF EXISTS ] view_name
  RESET ( view_option_name [, ... ] );
```

参数说明

❖ **IF EXISTS**

使用这个选项，如果视图不存在时不会产生错误，仅有会有一个提示信息。

❖ **view_name**

视图名称，可以用模式修饰。

取值范围：字符串，符合标识符命名规范。

❖ **column_name**

可选的名称列表，视图的字段名。如果没有给出，字段名取自查询中的字段名。

取值范围：字符串，符合标识符命名规范。

❖ **SET/DROP DEFAULT**

设置或删除一个列的缺省值，该参数暂无实际意义。

❖ **new_owner**

视图新所有者的用户名称。

❖ **new_name**

视图的新名称。

❖ **new_schema**

视图的新模式。

❖ **view_option_name [= view_option_value]**

该子句为视图指定一个可选的参数。

目前 view_option_name 支持的参数仅有 security_barrier，当 VIEW 试图提供行级安全时，应使用该参数。

取值范围：Boolean 类型，TRUE、FALSE。

示例

```
--创建测试 schema。
panweidb=# CREATE SCHEMA stest;

--创建测试表 stest.customer。
panweidb=# CREATE TABLE stest.customer(c_customer_sk int);

--插入测试数据。
panweidb=# INSERT INTO stest.customer values(1);
panweidb=# INSERT INTO stest.customer values(2);
panweidb=# INSERT INTO stest.customer values(3);

--创建一个由 c_customer_sk 小于 10 的内容组成的视图。
panweidb=# CREATE VIEW stest.customer_details_view_v1 AS
   SELECT * FROM stest.customer
   WHERE c_customer_sk < 10;

--修改视图名称。
panweidb=# ALTER VIEW stest.customer_details_view_v1 RENAME TO customer_det
ails_view_v2;

--修改视图所属 schema。
panweidb=# ALTER VIEW stest.customer_details_view_v2 SET schema public;

--删除视图。
panweidb=# DROP VIEW public.customer_details_view_v2;
```

相关链接

CREATE VIEW(参考：开发者指南->SQL 语法参考->SQL 语法->CREATE-VIEW 章节)，DROP VIEW(参考：开发者指南->SQL 语法参考->SQL 语法->DROP-VIEW 章节)

4.5.46.ANALYZE 和 ANALYSE

功能描述

ANALYZE 和 ANALYSE 用于收集与数据库中普通表内容相关的统计信息，统计结果存储在系统表 PG_STATISTIC 下。执行计划生成器会使用这些统计数据，以确定最有效的执行计划。

如果没有指定参数，ANALYZE 会分析当前数据库中的每个表和分区表。同时也可以通过指定 table_name、column 和 partition_name 参数把分析限定在特定的表、列或分区表中。

ANALYZE VERIFY 和 ANALYSE VERIFY 用于检测数据库中普通表（行存表、列存表）的数据文件是否损坏。

注意事项

- ANALYZE 非临时表不能在一个匿名块、事务块、函数或存储过程内被执行。支持存储过程中 ANALYZE 临时表，不支持统计信息回滚操作。
- ANALYZE VERIFY 操作处理的大多为异常场景检测需要使用 RELEASE 版本。
- ANALYZE VERIFY 场景不触发远程读，因此远程读参数不生效。对于关键系统表出现错误被系统检测出页面损坏时，将直接报错不再继续检测。

语法格式

- 收集表的统计信息。

```
{ ANALYZE | ANALYSE } [ VERBOSE ]
 [ table_name [ ( column_name [, ...] ) ] ];
```

- 收集分区表的统计信息。

```
{ ANALYZE | ANALYSE } [ VERBOSE ]
[ table_name [ ( column_name [, ...] ) ] ]
PARTITION ( patrition_name ) ;
```

普通分区表目前支持针对某个分区的统计信息的语法，在功能上不支持针对某个分区的统计信息收集。

- 收集多列统计信息。

```
{ANALYZE | ANALYSE} [ VERBOSE ]
 table_name (( column_1_name, column_2_name [, ...] ));
```

说明：

- 收集多列统计信息时，请设置 GUC 参数 default_statistics_target 为负数， 以使用百分比采样方式。
- 每组多列统计信息最多支持 32 列。
- 如果关闭 GUC 参数 enable_functional_dependency，每组多列统计信息最多支持 32 列；如果开启 GUC 参数 enable_functional_dependency，每组多列统计信息最多支持 4 列。
- 不支持收集多列统计信息的表：系统表。

- 检测当前库的数据文件。

```
{ANALYZE | ANALYSE} VERIFY {FAST|COMPLETE};
```

说明：

- Fast 模式校验时，需要对校验的表有并发的 DML 操作，会导致校验过程中有误报的问题，因为当前 Fast 模式是直接从磁盘上读取，并发有其他线程修改文件时，会导致获取的数据不准确，建议离线操作。
- 支持对全库进行操作，由于涉及的表较多，建议以重定向保存结果，命令如下所示：

```
gsql -d database -p port -f "verify.sql"> verify_warning.txt 2>&1
```

- 对外提示 NOTICE 只核对外可见的表，内部表的检测会包含在它所依赖的外部表，不对外显示和呈现。
- 此命令的处理可容错 ERROR 级别的处理。由于 debug 版本的 Assert 可能会导致 core 无法继续执行命令，建议在 release 模式下操作。
- 对于全库操作时，当关键系统表出现损坏则直接报错，不再继续执行。
- 不支持临时表和 unlog 表。

- 检测表和索引的数据文件。

```
{ANALYZE | ANALYSE} VERIFY {FAST|COMPLETE} table_name|index_name [CASCADE];
```

说明：

- 支持对普通表的操作和索引表的操作，但不支持对索引表 index 使用 CASCADE 操作。原因是由于 CASCADE 模式用于处理主表的所有索引表，当单独对索引表进行检测时，无需使用 CASCADE 模式。
- 不支持临时表和 unlog 表。
- 对于主表的检测会同步检测主表的内部表，例如 toast 表、cudesc 表等。
- 当提示索引表损坏时，建议使用 reindex 命令进行重建索引操作。
- 检测表分区的数据文件。

```
{ANALYZE | ANALYSE} VERIFY {FAST|COMPLETE} table_name PARTITION {(patrition_name)}[CASCADE];
```

说明：

- 支持对表的单独分区进行检测操作，但不支持对索引表 index 使用 CASCADE 操作。
- 不支持临时表和 unlog 表。
- 全局分区索引适配支持 ANALYZE VERIFY FAST。

参数说明

- VERBOSE：启用显示进度信息，如果指定了 VERBOSE，ANALYZE 发出进度信息，表明目前正在处理的表。各种有关表的统计信息也会打印出来。
- FAST|COMPLETE：对于行存表，FAST 模式下主要对于行存表的 CRC 和 page header 进行校验，如果校验失败则会告警; 而 COMPLETE 模式下，则主要对行存表的指针、tuple 进行解析校验。 对于列存表，FAST 模式下主要对于列存表的 CRC 和 magic 进行校验，如果校验失败则会告警; 而 COMPLETE 模式下，则主要对列存表的 CU 进行解析校验。
- table_name：需要分析的特定表的表名（可能会带模式名），如果省略，将对数据库中的所有表（非外部表）进行分析。对于 ANALYZE 收集统计信息，目前仅支持行存表、列存表。

  取值范围：已有的表名。
- index_name 需要分析的特定索引表的表名（可能会带模式名）。

取值范围：已有的表名。

- CASCADE：CASCADE 模式下会对当前表的所有索引进行检测处理。
- partition_name：如果 table 为分区表，在关键字 PARTITION 后面指定分区名 partition_name 表示分析该分区表的统计信息。目前语法上支持分区表做 ANALYZE，但功能实现上暂不支持对指定分区统计信息的分析。

取值范围：表的某一个分区名。

示例

1、创建测试表。

```
CREATE TABLE customer_info
(
WR_RETURNED_DATE_SK   INTEGER         ,
WR_RETURNED_TIME_SK   INTEGER         ,
WR_ITEM_SK       INTEGER       NOT NULL,
WR_REFUNDED_CUSTOMER_SK  INTEGER
) ;
```

2、创建分区表。

```
CREATE TABLE customer_par
(
WR_RETURNED_DATE_SK   INTEGER         ,
WR_RETURNED_TIME_SK   INTEGER         ,
WR_ITEM_SK       INTEGER       NOT NULL,
WR_REFUNDED_CUSTOMER_SK  INTEGER
)
PARTITION BY RANGE(WR_RETURNED_DATE_SK)
(
PARTITION P1 VALUES LESS THAN(2452275),
PARTITION P2 VALUES LESS THAN(2452640),
PARTITION P3 VALUES LESS THAN(2453000),
PARTITION P4 VALUES LESS THAN(MAXVALUE)
)
ENABLE ROW MOVEMENT;
```

3、使用 ANALYZE 语句更新统计信息。

```
ANALYZE customer_info;
```

4、使用 ANALYZE VERBOSE 语句更新统计信息，并输出表的相关信息。

```
ANALYZE VERBOSE customer_info;
```

返回结果为：

```
INFO:  analyzing "public.customer_info"(node1 pid=336454)
INFO:  ANALYZE INFO : "customer_info": scanned 0 of 0 pages, containing 0 live ro
ws and 0 dead rows; 0 rows in sample, 0 estimated total rows(node1 pid=336454)
ANALYZE
```

5、删除表。

```
DROP TABLE customer_info;
DROP TABLE customer_par;
```

4.5.47.ANONYMOUS BLOCK

功能描述

定义一个新的匿名块。

注意事项

无

语法格式

```
[DECLARE [declare_statements]]
    BEGIN
    execution_staements
    END;
    /
```

参数说明

- declare_statements

声明变量，包括变量名和变量类型，如“sales_cnt int”。

❖ execution_statements

匿名块中要执行的语句。

取值范围：DML 操作(数据操纵操作：select、insert、delete、update)或系统表中已注册的函数名称。

示例

1、创建测试表 test。

```
CREATE TABLE test(id int);
```

2、 执行匿名块

```
BEGIN
for i in 0..10 LOOP
INSERT INTO test(id) values (i);
END LOOP;
END;
/
```

3、 验证结果

```
select * from test;
```

当结果显示如下信息，则表示匿名块执行完成。

```
 id
----------------
 0
 1
 2
 3
 4
 5
 6
 7
 8
 9
```

```
 10
(11 rows)
```

相关链接

BEGIN（参考：开发者指南->SQL 语法参考->SQL 语法->BEGIN 章节）

4.5.48.BEGIN

功能描述

BEGIN 可以用于开始一个匿名块，也可以用于开始一个事务。本节描述用 BEGIN 开始匿名块的语法，以 BEGIN 开始事务的语法见章节“START TRANSACTION”。

语法格式

开启匿名块：

```
[DECLARE [declare_statements]]
BEGIN
execution_statements
END;
/
```

参数说明

- declare_statements：声明变量，包括变量名和变量类型，如“sales_cnt int”。
- execution_statements：匿名块中要执行的语句。

  取值范围：DML 操作(数据操纵操作：select、insert、delete、update)或系统表中已注册的函数名称。

注意事项

在 gsql 中使用 BEGIN 开启事务执行 SQL 语句时，请勿将 BEGIN 与需要执行的 SQL 语句写在一行，因为与存储过程、匿名块等语法相关，可能会出现获取结果次数异常的情况。

示例

1、创建测试表 test。

```
CREATE TABLE test(id int);
```

2、 执行匿名块

```
BEGIN
for i in 0..10 LOOP
INSERT INTO test(id) values (i);
END LOOP;
END;
/
```

3、 验证结果

```
select * from test;
```

当结果显示如下信息，则表示匿名块执行完成。

```
 id
------------------
 0
 1
 2
 3
 4
 5
 6
 7
 8
 9
 10
(11 rows)
```

4、 删除测试表

```
DROP TABLE test;
```

4.5.49.CALL

功能描述

使用 CALL 命令可以调用已定义的函数和存储过程。

注意事项

- 函数或存储过程的所有者、被授予了函数或存储过程 EXECUTE 权限的用户或被授予 EXECUTE ANY FUNCTION 权限的用户有权调用函数或存储过程，系统管理员默认拥有此权限。
- 在使用 CALL 语法调用时，允许使用变量对应作为 out、inout 类型的参数，可参考《PanWeiDB V2.0.2 Build01 MySQL 兼容性手册》 -> SQL 语法 -> CALL。

语法格式

```
CALL [schema.|package.] {func_name| procedure_name} ( param_expr );
```

参数说明

- **schema**

  函数或存储过程所在的模式名称。

- **package**

  函数或存储过程所在的 package 名称。

- **func_name**

  所调用函数或存储过程的名称。

  取值范围：已存在的函数名称。

- **param_expr**

  参数列表可以用符号“:=”或者“=>”将参数名和参数值隔开，这种方法的好处是参数可以以任意顺序排列。若参数列表中仅出现参数值，则参数值的排列顺序必须和函数或存储过程定义时的相同。

  取值范围：已存在的函数参数名称或存储过程参数名称。

【说明】

参数可以包含入参（参数名和类型之间指定“IN”关键字）和出参（参数名和类型之间指定“OUT”关键字），使用 CALL 命令调用函数或存储过程时，对于

非重载的函数，参数列表必须包含出参，出参可以传入一个变量或者任一常量，详见示例。对于重载的 package 函数，参数列表里可以忽略出参，忽略出参时可能会导致函数找不到。包含出参时，出参只能是常量。

示例

❖ **示例 1：**

1、创建一个函数 func_add_sql，计算两个整数的和，并返回结果。

```
panweidb=# CREATE FUNCTION func_add_sql(num1 integer, num2 integer) RETURN
 integer
AS
BEGIN
RETURN num1 + num2;
END;
/
```

2、按参数值传递。

```
CALL func_add_sql(1, 3);
```

结果显示如下：

```
 func_add_sql
--------------
            4
(1 row)
```

3、使用命名标记法传参。

```
CALL func_add_sql(num1 => 1,num2 => 3);
```

结果显示如下：

```
 func_add_sql
--------------
            4
(1 row)
```

4、删除函数。

```
DROP FUNCTION func_add_sql;
```

❖ **示例 2:**

1、创建带出参的函数。

```
CREATE FUNCTION func_increment_sql(num1 IN integer, num2 IN integer, res OUT i
nteger)
RETURN integer
AS
BEGIN
res := num1 + num2;
END;
/
```

2、出参传入常量。

```
CALL func_increment_sql(1,2,1);
```

结果显示如下：

```
 res
-----
   3
(1 row)
```

3、删除函数。

```
DROP FUNCTION func_increment_sql;
```

❖ **示例 3:**

【说明】
仅在 MySQL 兼容模式下，针对无参的存储过程可不使用括号。如下示例要求在 MySQL 兼容模式下执行。

1、创建无参存储过程。

```
CREATE PROCEDURE ptest()
AS
BEGIN
raise notice 'test';
```

```
END;
/
```

2、不加括号调用存储过程。

```
call ptest;
```

结果显示如下：

```
 ptest
-------


(1 row)
```

❖ **示例 4**

1、设置参数设置 behavior_compat_options = “proc_outparam_override”，控制存储过程出参的重载行为。

```
set behavior_compat_options ="proc_outparam_override";
```

2、创建 package 属性的重载函数。

```
CREATE OR REPLACE PACKAGE test_overload IS
function testp(a int) return int;
function testp(a int, b out int) return int;
end test_overload;
/

create or replace package body test_overload --创建 package body
is
function testp(a int) return int
is
Begin
raise notice 'func without out_arg';
return a;
end;
function testp(a int, b out int)
return int
is
begin
```

```
b:=1;
Raise notice 'func with out_arg';
return 2;
end;
end test_overload;
/
```

3、使用 call 调用 package 属性的重载函数。

```
call test_overload.testp(1);--调用忽略出参
call test_overload.testp(1,2);--调用包含出参
```

返回结果为如下：

```
NOTICE:  func without out_arg
 testp
-------
     1
(1 row)

NOTICE:  func with out_arg
 testp | b
-------+---
     2 | 1
(1 row)
```

4.5.50.CHECKPOINT

功能描述

检查点（CHECKPOINT）是一个事务日志中的点，所有数据文件都在该点被更新以反映日志中的信息，所有数据文件都将被刷新到磁盘。

设置事务日志检查点。预写式日志（WAL）缺省时在事务日志中每隔一段时间放置一个检查点。可以使用 pw_guc 命令设置相关运行时参数（checkpoint_segments、checkpoint_timeout 和 incremental_checkpoint_timeout）来调整这个原子化检查点的间隔。

注意事项

- 只有系统管理员和运维管理员可以调用 CHECKPOINT。

❖ CHECKPOINT 强制立即进行检查，而不是等到下一次调度时的检查点。

语法格式

```
CHECKPOINT;
```

参数说明

无。

示例

```
--设置检查点。
panweidb=# CHECKPOINT;
```

4.5.51.CLEAN CONNECTION

功能描述

用来清理数据库连接。允许在节点上清理指定数据库的指定用户的相关连接。

注意事项

❖ PanWeiDB 下不支持指定节点，仅支持 TO ALL。

❖ 该功能仅在 force 模式下，可以清理正在使用的正常连接。

语法格式

```
CLEAN CONNECTION
    TO { COORDINATOR ( nodename [, ... ] ) | NODE ( nodename [, ... ] )| ALL [ CHECK ]
[ FORCE ] }
    [ FOR DATABASE dbname ]
    [ TO USER username ];
```

参数说明

❖ **CHECK**

仅在节点列表为 TO ALL 时可以指定。如果指定该参数，会在清理连接之前检查数据库是否被其他会话连接访问。此参数主要用于 DROP

DATABASE 之前的连接访问检查，如果发现有其他会话连接，则将报错并停止删除数据库。

- **FORCE**

  仅在节点列表为 TO ALL 时可以指定，如果指定该参数，所有和指定 dbname 和 username 相关的线程都会收到 SIGTERM 信号，然后被强制关闭。

- **COORDINATOR (nodename [, ...]) | NODE (nodename [, ...]) | ALL**

  删除指定节点上的连接。有三种场景：

  - 删除指定 CN 上的连接，PanWeiDB 不支持。
  - 删除指定 DN 上的连接，PanWeiDB 不支持。
  - 删除所有节点上的连接(TO ALL)，PanWeiDB 仅支持该场景。

- **dbname**

  删除指定数据库上的连接。如果不指定，则删除所有数据库的连接。

  取值范围：已存在数据库名。

- **username**

  删除指定用户上的连接。如果不指定，则删除所有用户的连接。

  取值范围：已存在的用户。

示例

```
--创建 jack 用户。
CREATE USER jack PASSWORD 'Bigdata123@';

--删除用户 jack 在数据库 template1 上的所有连接。
CLEAN CONNECTION TO ALL FOR DATABASE template1 TO USER jack;

--删除用户 jack 的所有连接。
CLEAN CONNECTION TO ALL TO USER jack;

--删除在数据库 gaussdb 上的所有连接。
CLEAN CONNECTION TO ALL FORCE FOR DATABASE gaussdb;

--删除用户 jack。
DROP USER jack;
```

4.5.52.CLOSE

功能描述

CLOSE 释放和一个游标关联的所有资源。

注意事项

- 不允许对一个已关闭的游标再做任何操作。
- 一个不再使用的游标应该尽早关闭。
- 当创建游标的事务用 COMMIT 或 ROLLBACK 终止之后，每个不可保持的已打开游标都隐含关闭。
- 当创建游标的事务通过 ROLLBACK 退出之后，每个可以保持的游标都将隐含关闭。
- 当创建游标的事务成功提交，可保持的游标将保持打开，直到执行一个明确的 CLOSE 或者客户端断开。
- PanWeiDB 没有明确打开游标的 OPEN 语句，因为一个游标在使用 CURSOR 命令定义的时候就打开了。可以通过查询系统视图 pg_cursors 看到所有可用的游标。

语法格式

```
CLOSE { cursor_name | ALL };
```

参数说明

- **cursor_name**

  一个待关闭的游标名称。

- **ALL**

  关闭所有已打开的游标。

示例

参考：SQL 语法参考->SQL 语法->FETCH 的示例。

相关链接

参考：SQL 语法参考->SQL 语法中的 FETCH、MOVE 章节。

4.5.53.CLUSTER

功能描述

根据一个索引对表进行聚簇排序。

CLUSTER 指定 PanWeiDB 通过索引名指定的索引聚簇由表名指定的表。表名上必须已经定义该索引。

当对一个表聚集后，该表将基于索引信息进行物理存储。聚集是一次性操作：当表被更新之后，更改的内容不会被聚集。也就是说，系统不会试图按照索引顺序对新的存储内容及更新记录进行重新聚集。

在对一个表聚簇之后，PanWeiDB 会记录在哪个索引上建立了聚集。CLUSTER table_name 的聚集形式在之前的同一个索引的表上重新聚集。用户也可以用 ALTER TABLE 的 CLUSTER 或 SET WITHOUT CLUSTER 形式来设置索引来用于后续的聚集操作或清除任何之前的设置。

不含参数的 CLUSTER 会将当前用户所拥有的数据库中的先前做过聚簇的所有表重新处理，或者系统管理员调用的这些表。

在对一个表进行聚簇的时候，会在其上请求一个 ACCESS EXCLUSIVE 锁。这样就避免了在 CLUSTER 完成之前对此表执行其他的操作(包括读写)。

注意事项

- 只有行存 B-tree 索引支持 CLUSTER 操作。
- 如果用户只是随机访问表中的行，那么表中数据的实际存储顺序是无关紧要的。但是，如果对某些数据的访问多于其他数据，而且有一个索引将这些数据分组，那么将使用 CLUSTER 中会有所帮助。如果从一个表中请求一定索引范围的值，或者是一个索引值对应多行，CLUSTER 也会有助于应用，因为如果索引标识出第一匹配行所在的存储页，所有其他行也可能已经在同一个存储页里了，这样便节省了磁盘访问的时间，加速了查询。
- 在聚簇过程中，系统先创建一个按照索引顺序建立的表的临时拷贝。同时也建立表上的每个索引的临时拷贝。因此，需要磁盘上有足够的剩余空间，至少是表大小和索引大小的和。

- 因为 CLUSTER 记忆聚集信息，可以在第一次的时候手工对表进行聚簇，然后设置一个类似 VACUUM 的时间，这样就可以周期地自动对表进行聚簇操作。
- 因为优化器记录着有关表的排序的统计，所以建议在新近聚簇的表上运行 ANALYZE。否则，优化器可能会选择很差劲的查询规划。
- CLUSTER 不允许在事务中执行。
- 如果没有打开 xc_maintenance_mode 参数，那么 CLUSTER 操作将跳过所有系统表。
- 段页式表不支持 CLUSTER 操作。

语法格式

- 对一个表进行聚簇排序。

```
CLUSTER [ VERBOSE ] table_name [ USING index_name ];
```

- 对一个分区进行聚簇排序。

```
CLUSTER [ VERBOSE ] table_name PARTITION ( partition_name ) [ USING index_name ];
```

- 对已做过聚簇的表重新进行聚簇。

```
CLUSTER [ VERBOSE ];
```

参数说明

- **VERBOSE**

  启用显示进度信息。

- **table_name**

  表名称。

  取值范围：已存在的表名称。

- **index_name**

  索引名称。

  取值范围：已存在的索引名称。

- **partition_name**

  分区名称。

  取值范围：已存在的分区名称。

示例

1、创建一个分区表。

```
CREATE TABLE inventory_p1
(
   INV_DATE_SK            INTEGER            NOT NULL,
   INV_ITEM_SK            INTEGER            NOT NULL,
   INV_WAREHOUSE_SK       INTEGER            NOT NULL,
   INV_QUANTITY_ON_HAND   INTEGER
)
PARTITION BY RANGE(INV_DATE_SK)
(
      PARTITION P1 VALUES LESS THAN(2451179),
      PARTITION P2 VALUES LESS THAN(2451544),
      PARTITION P3 VALUES LESS THAN(2451910),
      PARTITION P4 VALUES LESS THAN(2452275),
      PARTITION P5 VALUES LESS THAN(2452640),
      PARTITION P6 VALUES LESS THAN(2453005),
      PARTITION P7 VALUES LESS THAN(MAXVALUE)
);
```

2、 创建索引 ds_inventory_p1_index1。

```
CREATE INDEX ds_inventory_p1_index1 ON inventory_p1 (INV_ITEM_SK) LOCAL;
```

3、对表 inventory_p1 进行聚集。

```
CLUSTER inventory_p1 USING ds_inventory_p1_index1;
```

4、对分区 p3 进行聚集。

```
CLUSTER inventory_p1 PARTITION (p3) USING ds_inventory_p1_index1;
```

5、对数据库中可以进行聚集的表进行聚集。

```
CLUSTER;
```

6、删除索引。

```
DROP INDEX ds_inventory_p1_index1;
```

7、删除分区表。

```
DROP TABLE inventory_p1;
```

4.5.54.COMMENT

功能描述

定义或修改一个对象的注释。

语法格式

```
COMMENT ON
{
 AGGREGATE agg_name (agg_type [, ...] ) |
 CAST (source_type AS target_type) |
 COLLATION object_name |
 COLUMN { table_name.column_name | view_name.column_name } |
 CONSTRAINT constraint_name ON table_name |
 CONVERSION object_name |
 DATABASE object_name |
 DOMAIN object_name |
 EXTENSION object_name |
 FOREIGN DATA WRAPPER object_name |
 FOREIGN TABLE object_name |
 FUNCTION function_name ( [ {[ argname ] [ argmode ] argtype} [, ...] ] ) |
 INDEX object_name |
 LARGE OBJECT large_object_oid |
 OPERATOR operator_name (left_type, right_type) |
 OPERATOR CLASS object_name USING index_method |
 OPERATOR FAMILY object_name USING index_method |
 [ PROCEDURAL ] LANGUAGE object_name |
 ROLE object_name |
 SCHEMA object_name |
 SEQUENCE object_name |
 SERVER object_name |
 TABLE object_name |
 TABLESPACE object_name |
 TEXT SEARCH CONFIGURATION object_name |
 TEXT SEARCH DICTIONARY object_name |
 TEXT SEARCH PARSER object_name |
```

```
TEXT SEARCH TEMPLATE object_name |
TYPE object_name |
VIEW object_name |
TRIGGER trigger_name ON table_name
}
 IS 'text';
```

参数说明

- **agg_name**

  聚集函数的名称。

- **agg_type**

  聚集函数参数的类型。

- **source_type**

  类型转换的源数据类型。

- **target_type**

  类型转换的目标数据类型。

- **object_name**

  对象名。

- **table_name.column_name**

  view_name.column_name

  定义/修改注释的列名称。前缀可加表名称或者视图名称。

- **constraint_name**

  定义/修改注释的表约束的名称。

- **table_name**

  表的名称。

- **function_name**

  定义/修改注释的函数名称。

- **argname,argmode,argtype**

  函数参数的模式、名称、类型。

- **large_object_oid**

  定义/修改注释的大对象的 OID 值。

- **operator_name**

操作符名称。

- **left_type,right_type**

  操作参数的数据类型（可以用模式修饰）。当前置或者后置操作符不存在时，可以增加 NONE 选项。

- **trigger_name**

  触发器名称。

- **text**

  注释。

注意事项

- 每个对象只存储一条注释，因此要修改一个注释，对同一个对象发出一条新的 COMMENT 命令即可。要删除注释，在文本字符串的位置写上 NULL 即可。当删除对象时，注释自动被删除掉。
- 目前注释浏览没有安全机制：任何连接到某数据库上的用户都可以看到所有该数据库对象的注释。共享对象（比如数据库、角色、表空间）的注释是全局存储的，连接到任何数据库的任何用户都可以看到它们。因此，不要在注释里存放与安全有关的敏感信息。
- 对大多数对象，只有对象的所有者或者被授予了对象 COMMENT 权限的用户可以设置注释，系统管理员默认拥有该权限。
- 角色没有所有者，所以 COMMENT ON ROLE 命令仅可以由系统管理员对系统管理员角色执行，有 CREATEROLE 权限的角色也可以为非系统管理员角色设置注释。系统管理员可以对所有对象进行注释。

示例

1、创建测试表。

```
CREATE TABLE customer_demographics_t2
(
CD_DEMO_SK            INTEGER          NOT NULL,
CD_GENDER             CHAR(1)                  ,
CD_MARITAL_STATUS     CHAR(1)                  ,
CD_EDUCATION_STATUS   CHAR(20)                 ,
CD_PURCCME_ESTIMATE   INTEGER                  ,
```

```
CD_CREDIT_RATING       CHAR(10)              ,
CD_DEP_COUNT          INTEGER               ,
CD_DEP_EMPLOYED_COUNT    INTEGER               ,
CD_DEP_COLLEGE_COUNT     INTEGER
)
WITH (ORIENTATION = COLUMN,COMPRESSION=MIDDLE) ;
```

2、为 customer_demographics_t2.cd_demo_sk 列加注释。

```
COMMENT ON COLUMN customer_demographics_t2.cd_demo_sk IS 'Primary key of
customer demographics table.';
```

3、创建一个视图。

```
CREATE VIEW customer_details_view_v2 AS
SELECT *
FROM customer_demographics_t2;
```

4、为 customer_details_view_v2 视图加注释。

```
COMMENT ON VIEW customer_details_view_v2 IS 'View of customer detail';
```

5、删除 view 和 customer_demographics_t2。

```
DROP VIEW customer_details_view_v2;
DROP TABLE customer_demographics_t2;
```

4.5.55.COMMIT | END

功能描述

通过 COMMIT 或者 END 可完成提交事务的功能，即提交事务的所有操作。

语法格式

```
{ COMMIT | END } [ WORK | TRANSACTION ] ;
```

参数说明

❖ COMMIT | END：提交当前事务，让所有当前事务的更改为其他事务可见。

- WORK | TRANSACTION：可选关键字，除了增加可读性没有其他任何作用。

注意事项

无。

示例

1、创建表。

```
CREATE TABLE customer_demographics_t2
  (
    CD_DEMO_SK        INTEGER       NOT NULL,
    CD_GENDER         CHAR(1),
    CD_MARITAL_STATUS     CHAR(1),
    CD_EDUCATION_STATUS    CHAR(20),
    CD_PURCCME_ESTIMATE    INTEGER,
    CD_CREDIT_RATING      CHAR(10),
    CD_DEP_COUNT        INTEGER,
    CD_DEP_EMPLOYED_COUNT   INTEGER,
    CD_DEP_COLLEGE_COUNT    INTEGER
  )
  WITH (ORIENTATION = COLUMN,COMPRESSION=MIDDLE)
  ;
```

2、 开启事务。

```
START TRANSACTION;
```

3、插入数据。

```
INSERT INTO customer_demographics_t2 VALUES(1,'M', 'U', 'DOCTOR DEGREE', 1200,
 'GOOD', 1, 0, 0);
INSERT INTO customer_demographics_t2 VALUES(2,'F', 'U', 'MASTER DEGREE', 300, '
BAD', 1, 0, 0);
```

4、 提交事务，让所有更改永久化。

```
COMMIT;
```

5、 新建连接查询数据。

```
SELECT * FROM customer_demographics_t2;
```

当结果显示如下信息时，则表示事务提交成功。

```
cd_demo_sk | cd_gender | cd_marital_status | cd_education_status |    cd_purccme
_estimate | cd_credi
t_rating | cd_dep_count | cd_dep_employed_count |      cd_dep_college_count
-----------+----------+------------------+--------------------+------------------
--+---------
---------+-------------+---------------------+---------------------
      1 | M      | U           | DOCTOR DEGREE      |            1200 | GOOD
     |         1 |             0 |              0
      2 | F      | U           | MASTER DEGREE      |             300 | BAD
     |         1 |             0 |              0
(2 rows)
```

6、 删除表 customer_demographics_t2。

```
DROP TABLE customer_demographics_t2;
```

4.5.56.COMMIT PREPARED

功能描述

提交一个早先为两阶段提交准备好的事务。

注意事项

- 该功能仅在维护模式（GUC 参数 xc_maintenance_mode 为 on 时）下可用。该模式谨慎打开，一般供维护人员排查问题使用，一般用户不应使用该模式。
- 命令执行者必须是该事务的创建者或系统管理员，且创建和提交操作只能在同一个会话中。
- 事务功能由数据库自动维护，不应显式使用事务功能。

语法格式

```
COMMIT PREPARED transaction_id ;
COMMIT PREPARED transaction_id WITH CSN;
```

参数说明

- **transaction_id**

  待提交事务的标识符。它不能和任何当前预备事务已经使用了的标识符同名。

- **CSN（commit sequence number）**

  待提交事务的序列号。它是一个 64 位递增无符号数。

示例

```
--提交标识符为的 trans_test 的事务。
panweidb=# COMMIT PREPARED 'trans_test';
```

相关链接

参考：SQL 语法参考->SQL 语法中的 PREPARE TRANSACTION、ROLLBACK PREPARED 章节。

4.5.57.COPY

功能描述

通过 COPY 命令实现在表和文件之间拷贝数据。

COPY FROM 从一个文件拷贝数据到一个表，COPY TO 把一个表的数据拷贝到一个文件。

注意事项

- 当参数 enable_copy_server_files 关闭时，只允许初始用户执行 COPY FROM FILENAME 或 COPY TO FILENAME 命令，当参数 enable_copy_server_files 打开，允许具有 SYSADMIN 权限的用户执行，但默认禁止对数据库配置文件、密钥文件、证书文件和审计日志执行 COPY FROM FILENAME 或 COPY TO FILENAME，以防止用户越权查看或修改敏感文件。
- COPY 只能用于表，不能用于视图。

- COPY TO 需要读取的表的 select 权限，copy from 需要插入的表的 insert 权限。
- 如果声明了一个字段列表，COPY 将只在文件和表之间拷贝已声明字段的数据。如果表中有任何不在字段列表里的字段，COPY FROM 将为那些字段插入缺省值。
- 如果声明了数据源文件，服务器必须可以访问该文件；如果指定了 STDIN，数据将在客户前端和服务器之间流动，输入时，表的列与列之间使用 TAB 键分隔，在新的一行中以反斜杠和句点（.）表示输入结束。
- 如果数据文件的任意行包含比预期多或者少的字段，COPY FROM 将抛出一个错误。
- 数据的结束可以用一个只包含反斜杠和句点（.）的行表示。如果从文件中读取数据，数据结束的标记是不必要的；如果在客户端应用之间拷贝数据，必须要有结束标记。
- COPY FROM 中\N 为空字符串，如果要输入实际数据值\N ，使用\N。
- COPY FROM 不支持在导入过程中对数据做预处理（比如说表达式运算、填充指定默认值等）。如果需要在导入过程中对数据做预处理，用户需先把数据导入到临时表中，然后执行 SQL 语句通过运算插入到表中，但此方法会导致 I/O 膨胀，降低导入性能。
- COPY FROM 在遇到数据格式错误时会回滚事务，但没有足够的错误信息，不方便用户从大量的原始数据中定位错误数据。
- 目标表存在 trigger，支持 COPY 操作。
- COPY 命令中，生成列不能出现在指定列的列表中。使用 COPY... TO 导出数据时，如果没有指定列的列表，则该表的所有列除了生成列都会被导出。COPY... FROM 导入数据时，生成列会自动更新，并像普通列一样保存。

语法格式

- 从一个文件拷贝数据到一个表。

```
COPY table_name [ ( column_name [, ...] ) ]
  FROM { 'filename' | STDIN }
  [ [ USING ] DELIMITERS 'delimiters' ]
```

```
  [ WITHOUT ESCAPING ]
  [ LOG ERRORS ]
  [ LOG ERRORS DATA ]
  [ REJECT LIMIT 'limit' ]
  [ [ WITH ] ( option [, ...] ) ]
  | copy_option
  | [ FIXED FORMATTER ( { column_name( offset, length ) } [, ...] ) ]
  | [ TRANSFORM ( { column_name [ data_type ] [ AS transform_expr ] } [, ...] ) ];
```

【说明】

- fixed formatter 与 copy_option 语法兼容，与 option 语法不兼容。
- copy_option 和 option 语法不兼容。
- transfrom 可以和 copy_option、fix ed formatter 配合使用。

- 把一个表的数据拷贝到一个文件。

```
COPY table_name [ ( column_name [, ...] ) ]
  TO { 'filename' | STDOUT }
  [ [ USING ] DELIMITERS 'delimiters' ]
  [ WITHOUT ESCAPING ]
  [ [WITH] ( option [, ...] ) ]
  | copy_option
 | [ FIXED FORMATTER ( { column_name( offset, length ) } [, ...] ) ];

COPY query
  TO { 'filename' | STDOUT }
  [ WITHOUT ESCAPING ]
  [ [WITH] ( option [, ...] ) ]
  | copy_option
 | [ FIXED FORMATTER ( { column_name( offset, length ) } [, ...] ) ];
```

【说明】

1、COPY TO 语法形式约束如下：(query)与[USING] DELIMITER 不兼容，即若 COPY TO 的数据来自于一个 query 的查询结果，那么 COPY TO 语法不能再指定[USING] DELIMITERS 语法子句。

2、对于 FIXED FORMATTTER 语法后面跟随的 copy_option 是以空格进行分隔的。
3、copy_option 是指 COPY 原生的参数形式，而 option 是兼容外表导入的参数形式。
4、语法中的 FIXED FORMATTER ({ column_name(offset, length) } [, ...]) 以及 [(option [, ...]) | copy_option [...]] 可以任意排列组合。

其中可选参数 option 子句语法为：

```
FORMAT format_name
| OIDS [ boolean ]
| DELIMITER 'delimiter_character'
| NULL 'null_string'
| HEADER [ boolean ]
| FILEHEADER 'header_file_string'
| FREEZE [ boolean ]
| QUOTE 'quote_coolean ]
| FORCE_QUOTE { ( column_name [, ...] ) | * }
| FORCE_NOT_NULL ( column_name [, ...] )
| FORCE_NULL ( column_name [, ...] )
| ENCODING 'encoding_name'
| IGNORE_EXTRA_DATA [ boolean ]
| FILL_MISSING_FIELDS [ boolean ]
| COMPATIBLE_ILLEGAL_CHARS [ boolean ]
| DATE_FORMAT 'date_format_string'
| TIME_FORMAT 'time_format_string'
| TIMESTAMP_FORMAT 'timestamp_format_string'
| SMALLDATETIME_FORMAT 'smalldatetime_format_string'
```

其中可选参数 copy_option 子句语法为：

```
OIDS
| NULL 'null_string'
| HEADER
| FILEHEADER 'header_file_string'
| FREEZE
```

```
| FORCE NOT NULL column_name [, ...]
| FORCE_NULL ( column_name [, ...] )
| FORCE QUOTE { column_name [, ...] | * }
| BINARY
| CSV
| QUOTE [ AS ] 'quote_character'
| ESCAPE [ AS ] 'escape_character'
| EOL 'newline_character'
| ENCODING 'encoding_name'
| IGNORE_EXTRA_DATA
| FILL_MISSING_FIELDS
| COMPATIBLE_ILLEGAL_CHARS
| DATE_FORMAT 'date_format_string'
| TIME_FORMAT 'time_format_string'
| TIMESTAMP_FORMAT 'timestamp_format_string'
| SMALLDATETIME_FORMAT 'smalldatetime_format_string'
```

参数说明

- **query**

  其结果将被拷贝。

  取值范围：一个必须用圆括弧包围的 SELECT 或 VALUES 命令。

- **table_name**

  表的名称（可以有模式修饰）。

  取值范围：已存在的表名。

- **column_name**

  可选的待拷贝字段列表。

  取值范围：如果没有声明字段列表，将使用所有字段。

- **STDIN**

  声明输入是来自标准输入。

- **STDOUT**

  声明输出打印到标准输出。

- **FIXED**

打开字段固定长度模式。在字段固定长度模式下，不能声明 DELIMITER、NULL、CSV 选项。指定 FIXED 类型后，不能再通过 option 或 copy_option 指定 BINARY、CSV、TEXT 等类型。

【说明】

定长格式定义如下：

1、每条记录的每个字段长度相同。

2、长度不足的字段以空格填充，数字类型字段左对齐，字符字段右对齐。

3、字段和字段之间没有分隔符。

- **[USING] DELIMITER 'delimiters'**

  在文件中分隔各个字段的字符串，分隔符最大长度不超过 10 个字节。

  取值范围：不允许包含.abcdefghijklmnopqrstuvwxyz0123456789 中的任何一个字符。

  缺省值：在文本模式下，缺省是水平制表符，在 CSV 模式下是一个逗号。

- **WITHOUT ESCAPING**

  在 TEXT 格式中，不对''和后面的字符进行转义。

  取值范围：仅支持 TEXT 格式。

- **LOG ERRORS**

  若指定，则开启对于 COPY FROM 语句中数据类型错误的容错机制。

  取值范围：仅支持导入（即 COPY FROM）时指定。

【说明】

- 此容错选项的使用限制如下：
- 此容错机制仅捕捉 COPY FROM 过程中数据库主节点上数据解析过程中相关的数据类型错误（DATA_EXCEPTION）。
- COPY 已有的容错选项（如 IGNORE_EXTRA_DATA）开启时，对应类型的错误会按照已有的方式处理而不会报出异常，因此错误表也不会有相应数据。

- **LOG ERRORS DATA**

  LOG ERRORS DATA 和 LOG ERRORS 的区别：

  - LOG ERRORS DATA 会填充容错表的 rawrecord 字段。

➢ 只有 supper 权限的用户才能使用 LOG ERRORS DATA 参数选项。

【说明】

使用“LOG ERRORS DATA”时，若错误内容过于复杂可能存在写入容错表失败的风险，导致任务失败。

❖ **REJECT LIMIT 'imit'**

与 LOG ERROR 选项共同使用，对 COPY FROM 的容错机制设置数值上限，一旦此 COPY FROM 语句错误数据超过选项指定条数，则会按照原有机制报错。

取值范围：正整数（1-INTMAX），'unlimited'（无最大值限制）

缺省值：若未指定 LOG ERRORS，则会报错；若指定 LOG ERRORS，则默认为 0。

【说明】

如上述 LOG ERRORS 中描述的容错机制，REJECT LIMIT 的计数也是按照执行 COPY FROM 的数据库主节点上遇到的解析错误数量计算，而不是数据库节点的错误数量。

❖ **FORMATTER**

在固定长度模式中，定义每一个字段在数据文件中的位置。按照 column(offset,length)格式定义每一列在数据文件中的位置。

取值范围：

- offset 取值不能小于 0，以字节为单位。
- length 取值不能小于 0，以字节为单位。

所有列的总长度和不能大于 1GB。

文件中没有出现的列默认以空值代替。

❖ **OPTION { option_name ' value ' }**

用于指定兼容外表的各类参数。

➢ FORMAT

数据源文件的格式。

取值范围：CSV、TEXT、FIXED、BINARY。

✧ CSV 格式的文件，可以有效处理数据列中的换行符，但对一些特殊字符处理有欠缺。

- ✧ TEXT 格式的文件，可以有效处理一些特殊字符，但无法正确处理数据列中的换行符。
- ✧ FIXED 格式的文件，适用于每条数据的数据列都比较固定的数据，长度不足的列会添加空格补齐，过长的列则会自动截断。
- ✧ BINARY 形式的选项会使得所有的数据被存储/读作二进制格式而不是文本。 这比 TEXT 和 CSV 格式的要快一些，但是一个 BINARY 格式文件可移植性比较差。

缺省值：TEXT

➢ DELIMITER

指定数据文件行数据的字段分隔符。

【说明】

- ❖ 分隔符不能是\r 和\n。
- ❖ 分隔符不能和 null 参数相同，CSV 格式数据的分隔符不能和 quote 参数相同。
- ❖ TEXT 格式数据的分隔符不能包含：小写字母、数字和特殊字符.\。
- ❖ 数据文件中单行数据长度需<1GB，如果分隔符较长且数据列较多的情况下，会影响导出有效数据的长度。
- ❖ 分隔符推荐使用多字符和不可见字符。多字符例如'$^&'；不可见字符例如 0x07、0x08、0x1b 等。

取值范围：支持多字符分隔符，但分隔符不能超过 10 个字节。

缺省值：

- ✧ TEXT 格式的默认分隔符是水平制表符（tab）。
- ✧ CSV 格式的默认分隔符为“,”。
- ✧ FIXED 格式没有分隔符。

➢ NULL

用来指定数据文件中空值的表示。

取值范围：

- ✧ null 值不能是\r 和\n，最大为 100 个字符。
- ✧ null 值不能和分隔符、quote 参数相同。

缺省值：

- ✧ CSV 格式下默认值是一个没有引号的空字符串。

- 在 TEXT 格式下默认值是\N。

➢ HEADER

指定导出数据文件是否包含标题行，标题行一般用来描述表中每个字段的信息。header 只能用于 CSV、FIXED 格式的文件中。

在导入数据时，如果 header 选项为 on，则数据文本第一行会被识别为标题行，会忽略此行。如果 header 为 off，而数据文件中第一行会被识别为数据。

在导出数据时，如果 header 选项为 on，且未指定 fileheader，则将把表的列名信息导出到文件的第一行；如果指定了 fileheader，则将把 fileheader 文件的第一行导出到最终输出文件的第一行。如果 header 为 off，则导出数据文件不包含标题行。

取值范围：true/on、false/off。

缺省值：false

➢ QUOTE

CSV 格式文件下的引号字符。

缺省值：双引号

【说明】

- quote 参数不能和分隔符、null 参数相同。
- quote 参数只能是单字节的字符。
- 推荐不可见字符作为 quote，例如 0x07、0x08、0x1b 等。

例如 quote e'\x22'：使用 ASCII 编码为 16 进制 22 的字符。

- **ESCAPE**

CSV 格式下，用来指定逃逸字符，逃逸字符只能指定为单字节字符。

缺省值：双引号。当与 quote 值相同时，会被替换为'\0'。

- **EOL 'newline_character'**

指定导入导出数据文件换行符样式。

取值范围：支持多字符换行符，但换行符不能超过 10 个字节。常见的换行符，如\r、\n、\r\n（设成 0x0D、0x0A、0x0D0A 效果是相同的），其他字符或字符串，如$、#。

【说明】

- EOL 参数只能用于 TEXT 格式的导入导出，不支持 CSV 格式和 FIXED 格式导入。为了兼容原有 EOL 参数，仍然支持导出 CSV 格式和 FIXED 格式时指定 EOL 参数为 0x0D 或 0x0D0A。
- EOL 参数不能和分隔符、null 参数相同。
- EOL 参数不能包含：.abcdefghijklmnopqrstuvwxyz0123456789。

- **FORCE_QUOTE { (column_name [, ...]) | * }**

  在 CSV COPY TO 模式下，强制在每个声明的字段周围对所有非 NULL 值都使用引号包围。NULL 输出不会被引号包围。

  取值范围：已存在的字段。

- **FORCE_NOT_NULL (column_name [, ...])**

  此函数指定的列将不会把输入匹配（识别）为 null 字符串。null 值字符串将被默认为空字符串，即长度为零的字符串而不是 null，即使它们没有用引号引起来。

  此选项仅允许在“COPY FROM”语句中，并且仅在指定为 CSV 格式时被允许使用。

  取值范围：已存在的字段。

- **FORCE NULL column_name \[, ...\]**

  在 CSV COPY FROM 模式下，将指定的字段表示空值的字符串设置为 NULL，包括加了引号的空值字符串。

  取值范围：已存在的字段。

- **ENCODING**

  指定数据文件的编码格式名称，缺省为当前数据库编码格式。

- **IGNORE_EXTRA_DATA**

  若数据源文件比外表定义列数多，是否会忽略对多出的列。该参数只在数据导入过程中使用。

  缺省值：false。

【须知】

如果行尾换行符丢失，使两行变成一行时，设置此参数为 true 将导致后一行数据被忽略掉。

取值范围：true/on、false/off。

- 参数为 true/on，若数据源文件比外表定义列数多，则忽略行尾多出来的列。
- 参数为 false/off，若数据源文件比外表定义列数多，会显示如下错误信息。

```
extra data after last expected column
```

❖ **COMPATIBLE_ILLEGAL_CHARS**

导入非法字符容错参数。此语法仅对 COPY FROM 导入有效。

取值范围：true/on、false/off

- 参数为 true/on，则导入时遇到非法字符进行容错处理，非法字符转换后入库，不报错，不中断导入。
- 参数为 false/off，导入时遇到非法字符进行报错，中断导入。

缺省值：false/off

【说明】

导入非法字符容错规则如下：

（1）对于'\0'，容错后转换为空格；

（2）对于其他非法字符，容错后转换为问号；

（3）若 compatible_illegal_chars 为 true/on 标识导入时对于非法字符进行容错处理，则若 NULL、DELIMITER、QUOTE、ESCAPE 设置为空格或问号则会通过如“illegal chars conversion may confuse COPY escape 0x20”等报错信息提示用户修改可能引起混淆的参数以避免导入错误。

❖ **FILL_MISSING_FIELDS**

当数据加载时，若数据源文件中一行的最后一个字段缺失的处理方式。

取值范围：true/on、false/off。

缺省值：false/off

❖ **DATE_FORMAT**

导入对于 DATE 类型指定格式。此参数不支持 BINARY 格式，会报“cannot specify bulkload compatibility options in BINARY mode”错误信息。此参数仅对 COPY FROM 导入有效。

取值范围：合法 DATE 格式。可参考：SQL 语法参考->SQL 基本要素->函数和操作符->时间和日期处理函数和操作符章节。

【说明】

对于 DATE 类型内建为 TIMESTAMP 类型的数据库，在导入的时候，若需指定格式，可以参考下面的 timestamp_format 参数。

- **TIME_FORMAT**

  导入对于 TIME 类型指定格式。此参数不支持 BINARY 格式，会报“cannot specify bulkload compatibility options in BINARY mode”错误信息。此参数仅对 COPY FROM 导入有效。

  取值范围：合法 TIME 格式，不支持时区。可参考：SQL 语法参考->SQL 基本要素->函数和操作符->时间和日期处理函数和操作符章节。

- **TIMESTAMP_FORMAT**

  导入对于 TIMESTAMP 类型指定格式。此参数不支持 BINARY 格式，会报“cannot specify bulkload compatibility options in BINARY mode”错误信息。此参数仅对 COPY FROM 导入有效。

  取值范围：合法 TIMESTAMP 格式，不支持时区。可参考：SQL 语法参考->SQL 基本要素->函数和操作符->时间和日期处理函数和操作符章节。

- **SMALLDATETIME_FORMAT**

  导入对于 SMALLDATETIME 类型指定格式。此参数不支持 BINARY 格式，会报“cannot specify bulkload compatibility options in BINARY mode”错误信息。此参数仅对 COPY FROM 导入有效。

  取值范围：合法 SMALLDATETIME 格式。可参考：SQL 语法参考->SQL 基本要素->函数和操作符->时间和日期处理函数和操作符章节。

- **COPY_OPTION { option_name ' value ' }**

  用于指定 COPY 原生的各类参数。

  - NULL null_string

    用来指定数据文件中空值的表示。

【说明】

在使用 COPY FROM 的时候，任何匹配这个字符串的字符串将被存储为 NULL 值，所以应该确保指定的字符串和 COPY TO 相同。

取值范围：

- null 值不能是\r 和\n，最大为 100 个字符。

✧ null 值不能和分隔符、quote 参数相同。

缺省值：

✧ 在 TEXT 格式下默认值是\N。

✧ CSV 格式下默认值是一个没有引号的空字符串。

➢ HEADER

指定导出数据文件是否包含标题行，标题行一般用来描述表中每个字段的信息。header 只能用于 CSV、FIXED 格式的文件中。

在导入数据时，如果 header 选项为 on，则数据文本第一行会被识别为标题行，会忽略此行。如果 header 为 off，而数据文件中第一行会被识别为数据。

在导出数据时，如果 header 选项为 on，且未指定 fileheader，则将把表的列名信息导出到文件的第一行；如果指定了 fileheader，则将把 fileheader 文件的第一行导出到最终输出文件的第一行。如果 header 为 off，则导出数据文件不包含标题行。

➢ FILEHEADER

导出数据时用于定义标题行的文件，一般用来描述每一列的数据信息。

【须知】

❖ 仅在 header 为 on 或 true 的情况下有效。

❖ fileheader 指定的是绝对路径。

❖ 该文件只能包含一行标题信息，并以换行符结尾，多余的行将被丢弃（标题信息不能包含换行符）。

❖ 该文件包括换行符在内长度不超过 1M。

❖ **FREEZE**

将 COPY 加载的数据行设置为已经被 frozen，就像这些数据行执行过 VACUUM FREEZE。

这是一个初始数据加载的性能选项。仅当以下三个条件同时满足时，数据行会被 frozen：

➢ 在同一事务中 create 或 truncate 这张表之后执行 COPY。

➢ 当前事务中没有打开的游标。

➢ 当前事务中没有原有的快照。

【说明】

❖ COPY 完成后，所有其他会话将会立刻看到这些数据。但是这违反了 MVCC 可见性的一般原则，用户应当了解这样会导致潜在的风险。

❖ 当条件不满足时，FREEZE 选项将会被忽略而不会报错。

❖ **FORCE NOT NULL column_name [, ...]**

在 CSV COPY FROM 模式下，指定的字段不为空。若输入为空，则将视为长度为 0 的字符串。

取值范围：已存在的字段。

❖ **FORCE QUOTE { column_name [, ...] | * }**

在 CSV COPY TO 模式下，强制在每个声明的字段周围对所有非 NULL 值都使用引号包围。NULL 输出不会被引号包围。

取值范围：已存在的字段。

❖ **BINARY**

使用二进制格式存储和读取，而不是以文本的方式。在二进制模式下，不能声明 DELIMITER、NULL、CSV 选项。指定 BINARY 类型后，不能再通过 option 或 copy_option 指定 CSV、FIXED、TEXT 等类型。

❖ **CSV**

打开逗号分隔变量（CSV）模式。指定 CSV 类型后，不能再通过 option 或 copy_option 指定 BINARY、FIXED、TEXT 等类型。

QUOTE [AS] 'quote_character'

CSV 格式文件下的引号字符。

缺省值：双引号。

【说明】

❖ quote 参数不能和分隔符、null 参数相同。

❖ quote 参数只能是单字节的字符。

❖ 推荐不可见字符作为 quote，例如 0x07、0x08、0x1b 等。

例如 quote e'\x22'：使用 ASCII 编码为 16 进制 22 的字符。

❖ **ESCAPE [AS] 'escape_character'**

CSV 格式下，用来指定逃逸字符，逃逸字符只能指定为单字节字符。

默认值为双引号。当与 quote 值相同时，会被替换为'\0'。

- **EOL 'newline_character'**

  指定导入导出数据文件换行符样式。

  取值范围：支持多字符换行符，但换行符不能超过 10 个字节。常见的换行符，如\r、\n、\r\n（设成 0x0D、0x0A、0x0D0A 效果是相同的），其他字符或字符串，如$、#。

【说明】

- EOL 参数只能用于 TEXT 格式的导入导出，不支持 CSV 格式和 FIXED 格式。为了兼容原有 EOL 参数，仍然支持导出 CSV 格式和 FIXED 格式时指定 EOL 参数为 0x0D 或 0x0D0A。
- EOL 参数不能和分隔符、null 参数相同。
- EOL 参数不能包含：.abcdefghijklmnopqrstuvwxyz0123456789。

- **ENCODING 'encoding_name'**

  指定文件编码格式名称。

  取值范围：有效的编码格式。

  缺省值：当前编码格式。

- **IGNORE_EXTRA_DATA**

  指定当数据源文件比外表定义列数多时，忽略行尾多出来的列。该参数只在数据导入过程中使用。

  若不使用该参数，在数据源文件比外表定义列数多，会显示如下错误信息。

```
extra data after last expected column
```

- **COMPATIBLE_ILLEGAL_CHARS**

  指定导入时对非法字符进行容错处理，非法字符转换后入库。不报错，不中断导入。此参数不支持 BINARY 格式，会报“cannot specify bulkload compatibility options in BINARY mode”错误信息。此参数仅对 COPY FROM 导入有效。

  若不使用该参数，导入时遇到非法字符进行报错，中断导入。

【说明】

导入非法字符容错规则如下：

（1）对于'\0'，容错后转换为空格；

（2）对于其他非法字符，容错后转换为问号；

（3）若 compatible_illegal_chars 为 true/on 标识，导入时对于非法字符进行容错处理，则若 NULL、DELIMITER、QUOTE、ESCAPE 设置为空格或问号则会通过如“illegal chars conversion may confuse COPY escape 0x20”等报错信息提示用户修改可能引起混淆的参数以避免导入错误。

- **FILL_MISSING_FIELDS**

  当数据加载时，若数据源文件中一行的最后一个字段缺失的处理方式。

  取值范围：true/on、false/off。

  缺省值：false/off。

【须知】

目前 COPY 指定此 Option 实际不会生效，即不会有相应的容错处理效果（不生效）。需要额外注意的是，打开此选项会导致解析器在数据库主节点数据解析阶段（即 COPY 错误表容错的涵盖范围）忽略此数据问题，而到数据库节点重新报错，从而使得 COPY 错误表（打开 LOG ERRORS REJECT LIMIT）在此选项打开的情况下无法成功捕获这类少列的数据异常。因此请不要指定此选项。

- **DATE_FORMAT 'date_format_string'**

  导入对于 DATE 类型指定格式。此参数不支持 BINARY 格式，会报“cannot specify bulkload compatibility options in BINARY mode”错误信息。此参数仅对 COPY FROM 导入有效。

  取值范围：合法 DATE 格式。可参考：SQL 语法参考->SQL 基本要素->函数和操作符->时间和日期处理函数和操作符章节。

【说明】

对于 DATE 类型内建为 TIMESTAMP 类型的数据库，在导入的时候，若需指定格式，可以参考下面的 timestamp_format 参数。

- **TIME_FORMAT 'time_format_string'**

  导入对于 TIME 类型指定格式。此参数不支持 BINARY 格式，会报“cannot specify bulkload compatibility options in BINARY mode”错误信息。此参数仅对 COPY FROM 导入有效。

  取值范围：合法 TIME 格式，不支持时区。可参考：SQL 语法参考->SQL 基本要素->函数和操作符->时间和日期处理函数和操作符章节。

- **TIMESTAMP_FORMAT 'timestamp_format_string'**

  导入对于 TIMESTAMP 类型指定格式。此参数不支持 BINARY 格式，会报“cannot specify bulkload compatibility options in BINARY mode”错误信息。此参数仅对 COPY FROM 导入有效。

  取值范围：合法 TIMESTAMP 格式，不支持时区。可参考：SQL 语法参考->SQL 基本要素->函数和操作符->时间和日期处理函数和操作符章节。

- **SMALLDATETIME_FORMAT 'smalldatetime_format_string'**

  导入对于 SMALLDATETIME 类型指定格式。此参数不支持 BINARY 格式，会报“cannot specify bulkload compatibility options in BINARY mode”错误信息。此参数仅对 COPY FROM 导入有效。

  取值范围：合法 SMALLDATETIME 格式。可参考：SQL 语法参考->SQL 基本要素->函数和操作符->时间和日期处理函数和操作符章节。

- **TRANSFORM ({ column_name [data_type] [AS transform_expr] } [, ...])**

  指定表中各个列的转换表达式；其中 data_type 指定该列在表达式参数中的数据类型；transform_expr 为目标表达式，返回与表中目标列数据类型一致的结果值，表达式可参考：SQL 语法参考->SQL 基本要素->表达式章节。

  COPY FROM 能够识别的特殊反斜杠序列如下所示。

  - \b：反斜杠 （ASCII 8）
  - \f：换页（ASCII 12）
  - \n：换行符 （ASCII 10）
  - \r：回车符 （ASCII 13）
  - \t：水平制表符 （ASCII 9）
  - \v：垂直制表符 （ASCII 11）
  - \digits：反斜杠后面跟着一到三个八进制数，表示 ASCII 值为该数的字符。
  - \xdigits：反斜杠 x 后面跟着一个或两个十六进制位声明指定数值编码的字符。

示例

```
--将 ship_mode 中的数据拷贝到/home/omm/ds_ship_mode.dat 文件中。
panweidb=# COPY ship_mode TO '/home/omm/ds_ship_mode.dat';

--将 ship_mode 输出到 stdout。
panweidb=# COPY ship_mode TO stdout;

--创建 ship_mode_t1 表。
panweidb=# CREATE TABLE ship_mode_t1
(
  SM_SHIP_MODE_SK        INTEGER           NOT NULL,
  SM_SHIP_MODE_ID        CHAR(16)          NOT NULL,
  SM_TYPE              CHAR(30)                ,
  SM_CODE               CHAR(10)               ,
  SM_CARRIER             CHAR(20)                ,
  SM_CONTRACT            CHAR(20)
)
WITH (ORIENTATION = COLUMN,COMPRESSION=MIDDLE)
;

--从 stdin 拷贝数据到表 ship_mode_t1。
panweidb=# COPY ship_mode_t1 FROM stdin;

--从/home/omm/ds_ship_mode.dat 文件拷贝数据到表 ship_mode_t1。
panweidb=# COPY ship_mode_t1 FROM '/home/omm/ds_ship_mode.dat';

--从/home/omm/ds_ship_mode.dat 文件拷贝数据到表 ship_mode_t1，应用 TRANS
FORM 表达式转换，取 SM_TYPE 列左边 10 个字符插入到表中。
panweidb=# COPY ship_mode_t1 FROM '/home/omm/ds_ship_mode.dat' TRANSFO
RM (SM_TYPE AS LEFT(SM_TYPE, 10));

--从/home/omm/ds_ship_mode.dat 文件拷贝数据到表 ship_mode_t1，使用参数如
下：导入格式为 TEXT（format 'text'），分隔符为'\t'（delimiter E'\t'），忽略多余列（i
gnore_extra_data 'true'），不指定转义（noescaping 'true'）。
panweidb=# COPY ship_mode_t1 FROM '/home/omm/ds_ship_mode.dat' WITH(for
mat 'text', delimiter E'\t', ignore_extra_data 'true', noescaping 'true');
```

```
--从/home/omm/ds_ship_mode.dat 文件拷贝数据到表 ship_mode_t1，使用参数如下：导入格式为 FIXED（FIXED），指定定长格式（FORMATTER(SM_SHIP_MODE_SK(0, 2), SM_SHIP_MODE_ID(2,16), SM_TYPE(18,30), SM_CODE(50,10), SM_CARRIER(61,20), SM_CONTRACT(82,20)))，忽略多余列（ignore_extra_data），有数据头（header）。
panweidb=# COPY ship_mode_t1 FROM '/home/omm/ds_ship_mode.dat' FIXED FORMATTER(SM_SHIP_MODE_SK(0, 2), SM_SHIP_MODE_ID(2,16), SM_TYPE(18,30), SM_CODE(50,10), SM_CARRIER(61,20), SM_CONTRACT(82,20)) header ignore_extra_data;

--删除 ship_mode_t1。
panweidb=# DROP TABLE ship_mode_t1;
```

4.5.58.CREATE AGGREGATE

功能描述

定义一个新的聚合函数。

语法格式

```
CREATE AGGREGATE name ( input_data_type [ , ... ] ) (
    SFUNC = sfunc,
    STYPE = state_data_type
    [ , FINALFUNC = ffunc ]
    [ , INITCOND = initial_condition ]
    [ , SORTOP = sort_operator ]
)

or the old syntax

CREATE AGGREGATE name (
    BASETYPE = base_type,
    SFUNC = sfunc,
    STYPE = state_data_type
    [ , FINALFUNC = ffunc ]
    [ , INITCOND = initial_condition ]
```

```
  [ , SORTOP = sort_operator ]
)
```

参数说明

- **name**

  要创建的聚合函数名(可以有模式修饰) 。

- **input_data_type**

  该聚合函数要处理的输入数据类型。要创建一个零参数聚合函数，可以使用代替输入数据类型列表。（count()就是这种聚合函数的一个实例。）

- **base_type**

  在以前的 CREATE AGGREGATE 语法中，输入数据类型是通过 basetype 参数指定的，而不是写在聚合的名称之后。 需要注意的是这种以前语法仅允许一个输入参数。 要创建一个零参数聚合函数，可以将 basetype 指定为 “ANY” (而不是*)。

- **sfunc**

  将在每一个输入行上调用的状态转换函数的名称。 对于有 N 个参数的聚合函数，sfunc 必须有 +1 个参数，其中的第一个参数类型为 state_data_type，其余的匹配已声明的输入数据类型。 函数必须返回一个 state_data_type 类型的值。 这个函数接受当前状态值和当前输入数据，并返回下个状态值。

- **state_data_type**

  聚合的状态值的数据类型。

- **ffunc**

  在转换完所有输入行后调用的最终处理函数，它计算聚合的结果。 此函数必须接受一个类型为 state_data_type 的参数。 聚合的输出数据 类型被定义为此函数的返回类型。 如果没有声明 ffunc 则使用聚合结果的状态值作为聚合的结果，且输出类型为 state_data_type。

- **initial_condition**

  状态值的初始设置(值)。 它必须是一个 state_data_type 类型可以接受的文本常量值。 如果没有声明，状态值初始为 NULL 。

- **sort_operator**

用于 MIN 或 MAX 类型聚合的排序操作符。 这个只是一个操作符名 (可以有模式修饰)。这个操作符假设接受和聚合一样的输入数据类型。

示例

```
CREATE AGGREGATE array_accum (anyelement)
(
  sfunc = array_append,
  stype = anyarray,
  initcond = '{}'
);
```

4.5.59.CREATE AUDIT POLICY

功能描述

创建统一审计策略。

注意事项

- 开启审计开关（audit_enabled=on）时，根据具体的审计策略决定需要记录的审计日志。
- 开启安全策略开关（enable_security_policy=on）时，审计策略才能生效。
- 未开启三权分立时，只有安全策略管理员（poladmin）、系统管理员（sysadmin）或初始用户能进行此操作。
- 开启三权分立（enable_Separation_Of_Duty）后，只有安全管理员才能进行此操作。

语法格式

```
CREATE AUDIT POLICY [ IF NOT EXISTS ] policy_name { { { privilege_audit_clause | access_audit_clause } [ filter_group_clause ] [ ENABLE | DISABLE ] } | [ sequence_audit_clause ] };
```

- 其中权限审计子句 privilege_audit_clause 可以是：

  ```
  PRIVILEGES { DDL | ALL } [ ON LABEL ( resource_label_name [, ... ] ) ]
  ```

  - 其中 DDL 可以是：

    ```
    { ( ALTER | ANALYZE | COMMENT | CREATE | DROP | GRANT | REVOKE | SET | SH
    ```

```
OW | LOGIN_ACCESS | LOGIN_FAILURE | LOGOUT | LOGIN ) }
```

- 其中 DML 可以是：

```
{ ( COPY | DEALLOCATE | DELETE_P | EXECUTE | REINDEX | INSERT | PREPARE | S
ELECT | TRUNCATE | UPDATE ) }
```

- 其中访问审计子句 access_audit_clause 可以是：

```
ACCESS { DML | ALL } [ ON LABEL ( resource_label_name [, ... ] ) ]
```

- 其中，定义审计策略过滤信息的子句 filter_group_clause 可以是：

```
FILTER ON { ( FILTER_TYPE ( filter_value [, ... ] ) ) [, ... ] }
```

- 其中 FILTER_TYPE 可以是：

```
{ APP | ROLES | IP | TIME_PERIOD }
```

- 其中，审计语句序列的子句 sequence_audit_clause 可以是：

```
SEQUENCE BY [ 'SQL' ]
```

参数说明

- **[IF NOT EXISTS]**

  如果指定了 IF NOT EXISTS，当数据库中存在同名审计策略时不会报错，而会发出通知，告知同名审计策略已存在。

- **policy_name**

  审计策略名称，需要唯一，不可重复；

  取值范围：字符串，要符合标识符的命名规范。

- **DDL**

  指的是针对数据库执行如下操作时进行审计，目前支持：CREATE、ALTER、DROP、ANALYZE、COMMENT、GRANT、REVOKE、SET、SHOW、LOGIN_ANY、LOGIN_FAILURE、LOGIN_SUCCESS、LOGOUT。

- **ALL**

  指的是上述 DDL 支持的所有对数据库的操作。

- **resource_label_name**

  资源标签名称。

- DDL

指的是针对数据库执行如下操作时进行审计，目前支持：CREATE、ALTER、DROP、ANALYZE、COMMENT、GRANT、REVOKE、SET、SHOW、LOGIN_ANY、LOGIN_FAILURE、LOGIN_SUCCESS、LOGOUT。

- **DML**

  指的是针对数据库执行如下操作时进行审计，目前支持：SELECT、COPY、DEALLOCATE、DELETE、EXECUTE、INSERT、PREPARE、REINDEX、TRUNCATE、UPDATE。

- **FILTER_TYPE**

  描述策略过滤的条件类型，包括 IP、APP、ROLES、TIME_PERIOD，分别表示将 IP、访问的应用名、数据库系统用户、以及重要时间段作为审计的过滤条件。

- **filter_value**

  指具体过滤信息内容。

- **ENABLE | DISABLE**

  可以打开或关闭统一审计策略。若不指定 ENABLE、DISABLE，语句默认为 ENABLE。

- **['SQL']**

  使用基于语句序列的审计功能：预先建立的语句序列审计规则，如果数据库中某个会话依次执行了预先定义的 SQL 语句，就会触发审计。

  - 创建语句审计策略时只能指定一条 SQL，通过 ALTER AUDIT POLICY 命令可向审计策略中追加 SQL。
  - 一条 SQL 只能被一个审计策略所包含；不能出现两个语句序列完全相同的审计策略。
  - 被审计的 SQL 不能存在语法错误；单条 SQL 语句应被单引号及方括号包围。

示例

1、创建 dev_audit 和 bob_audit 用户。

```
CREATE USER dev_audit PASSWORD 'dev@1234';
CREATE USER bob_audit password 'bob@1234';
```

2、创建一个表 tb_for_audit

```
CREATE TABLE tb_for_audit(col1 text, col2 text, col3 text);
```

3、创建资源标签

```
CREATE RESOURCE LABEL adt_lb0 add TABLE(tb_for_audit);
```

4、创建审计策略。

- 对数据库执行 create 操作创建审计策略。

```
CREATE AUDIT POLICY adt1 PRIVILEGES CREATE;
```

- 对数据库执行 select 操作创建审计策略。

```
CREATE AUDIT POLICY adt2 ACCESS SELECT;
```

- 仅审计记录用户 dev_audit 和 bob_audit 在执行针对 adt_lb0 资源进行的 create 操作数据库创建审计策略。

```
CREATE AUDIT POLICY adt3 PRIVILEGES CREATE ON LABEL(adt_lb0) FILTER O
N ROLES(dev_audit, bob_audit);
```

- 仅审计记录用户 dev_audit 和 bob_audit,客户端工具为 psql 和 gsql，IP 地址为'10.20.30.40', '127.0.0.0/24'，在执行针对 adt_lb0 资源进行的 select、insert、delete 操作数据库创建审计策略。

```
CREATE AUDIT POLICY adt4 ACCESS SELECT ON LABEL(adt_lb0), INSERT ON L
ABEL(adt_lb0), DELETE FILTER ON ROLES(dev_audit, bob_audit), APP(psql,
gsql), IP('10.20.30.40', '127.0.0.0/24');
```

- 审计通过 gsql 客户端工具或者“从 127.0.0.1 发起且用户为 panweidb”的对资源池 adt_lb0 中对象进行 insert 或 update 的操作。

```
CREATE AUDIT POLICY p_tbl ACCESS insert ON LABEL (adt_lb0), update ON L
ABEL (adt_lb0) FILTER ON IP ('127.0.0.1') AND ROLES (panweidb) OR APP (g
sql);
```

- 审计指定时间范围内用户 panweidb 的所有 drop 操作。

```
CREATE AUDIT POLICY adt_test privileges drop filter on time_period('2023-1
1-28 00:00:00','2023-12-07 00:00:00') and roles(panweidb);
```

- 预先定义语句序列审计规则，如果数据库中某个会话执行了该 SQL，就会触发审计。

```
CREATE AUDIT POLICY adt5 sequence by ['create table tb_123456(id int,na
me text);']
```

5、删除创建的审计策略 adt1。

```
DROP AUDIT POLICY adt1;
```

相关链接

参考：SQL 语法参考->SQL 语法中的 ALTER AUDIT POLICY、DROP AUDIT POLICY 章节。

4.5.60.CREATE CAST

功能描述

定义一个用户自定义的转换。

语法格式

```
CREATE CAST (source_type AS target_type)
  WITH FUNCTION function_name (argument_type [, ...])
  [ AS ASSIGNMENT | AS IMPLICIT ]

CREATE CAST (source_type AS target_type)
  WITHOUT FUNCTION
  [ AS ASSIGNMENT | AS IMPLICIT ]

CREATE CAST (source_type AS target_type)
  WITH INOUT
  [ AS ASSIGNMENT | AS IMPLICIT ]
```

参数说明

- **source_type**

  转换的源数据类型。

- **target_type**

  转换的目标数据类型。

- **function_name(argument_type [, ...])**

  用于执行转换的函数。这个函数名可以是用模式名修饰的。如果它没有用模式名修饰，那么该函数将从模式搜索路径中找出来。函数的结果数据类型必须匹配转换的目标类型。 它的参数在下面讨论。

- **WITHOUT FUNCTION**

表明源类型是对目标类型是二进制可强制转换的，所以没有函数需要执行此转换。

- **WITH INOUT**

  表明转换是 I/O 转换，通过调用源数据类型的输出函数来执行，并将结果传给目标数据类型的输入函数。

- **AS ASSIGNMENT**

  表示转换可以在赋值模式下隐含调用。

- **AS IMPLICIT**

  表示转换可以在任何环境里隐含调用。

转换实现函数可以有一到三个参数。第一个参数的类型必须与转换的源类型相同的，或可以从转换的源类型二进制可强制转换的。第二个参数，如果存在，必须是 integer 类型；它接收这些与目标类型相关联的类型修饰符，或者若什么都没有则是-1。第三个参数，如果存在，必须是 boolean 类型；若转换是一个显式类型转换则会收到 true，否则是 false。

一个转换函数的返回类型必须是与转换的目标类型相同或者对转换的目标类型二进制可强制转换 。

通常，一个转换必须有不同的源和目标数据类型。然而，若有多于一个参数的转换实现函数，则允许声明一个有相同的源和目标类型的转换。这用于表示系统目录中的特定类型的长度强制函数。命名的函数用于强制一个该类型的值为第二个参数给出的类型修饰符值。

如果一个类型转换的源类型和目标类型不同，并且接收多于一个参数，它就表示从一种类型转换成另外一种类型只用一个步骤，并且同时实施长度转换。如果没有这样的项可用， 那么转换成一个使用了类型修饰词的类型将涉及两个步骤，一个是在数据类型之间转换， 另外一个是施加修饰词指定的转换。

对域类型的转换目前没有作用。转换一般是针对域相关的所属数据类型。

示例

为了从类型 bigint 到类型 int4 创建一个指派映射要通过使用函数 int4(bigint):

```
CREATE CAST (bigint AS int4) WITH FUNCTION int4(bigint) AS ASSIGNMENT;
```

（这个转换在系统中已经预先定义了。）

兼容性

CREATE CAST 指令符合 SQL 标准，除了 SQL 没有为二进制可强制转换类型或者实现函数的额外参数来实现功能。

4.5.61.CREATE CLIENT MASTER KEY

功能描述

创建一个客户端主密钥对象，该对象可用于加密 Column Encryption Key 对象。

注意事项

本语法属于全密态数据库特有语法。

当使用 vql 连接数据库服务器时，需使用‘-C’参数，打开全密态数据库的开关，才能使用本语法。

由本语法创建的 CMK 对象中，仅存储从独立的密钥管理工具/服务/组件中读取密钥的方法，而不存储密钥本身。

语法格式

```
CREATE CLIENT MASTER KEY client_master_key_name WITH （KEY_STORE = key_store_name, KEY_PATH = "key_path_value", ALGORITHM = algorithm_type）
```

参数说明

- **client_master_key_name**

  该参数作为密钥对象名，在同一命名空间下，需满足命名唯一性约束。

  取值范围：字符串，需符合标识符的命名规范。

- **KEY_STORE**

  指定管理 CMK 的密钥工具或组件；取值：目前仅支持 localkms。

- **KEY_PATH**

KEY_STORE 负责管理多个 CMK 密钥，KEY_PATH 选项用于在 KEY_STORE 中唯一标识 CMK。取值类似：“key_path_value”。

- **ALGORITHM**

  由本语法创建的用于加密 COLUMN ENCRYPTION KEY，该参数用于指定加密算法的类型。取值范围：RSA_2048、RSA_3072 和 SM2。

【说明】

- **密钥存储路径：**默认情况下，localkms 将在$LOCALKMS_FILE_PATH 路径下生成/读取/删除密钥文件，用户可手动配置该环境变量。但是，用户也可以不用单独配置该环境变量，在尝试获取$LOCALKMS_FILE_PATH 失败时，localkms 会尝试获取$GAUSSHOME/etc/localkms/路径，如果该路径存在，则将其作为密钥存储路径。
- **密钥相关文件名：**使用 CREATE CMK 语法时，localkms 将会创建四个与存储密钥相关的文件。示例：当 KEY_PATH = "key_path_value", 四个文件的名称分别为 key_path_value.pub、key_path_value.pub.rand、 key_path_value.priv、 key_path_value.priv.rand。

所以，为了能够成功创建密钥相关文件，在密钥存储路径下，应该保证没有已存在的与密钥相关文件名同名的文件。

示例

使用普通账户 alice，连接全密态数据库。

```
[cmd] vql -U alice -h $host -p $port -d $database -C -r
```

使用本语法创建客户端加密主密钥(CMK)对象

```
panweidb=> CREATE CLIENT MASTER KEY a_cmk WITH (KEY_STORE = localkms, KEY_
PATH = "key_path_value", ALGORITHM = RSA_2048);
panweidb=> CREATE CLIENT MASTER KEY another_cmk WITH (KEY_STORE = localkms,
 KEY_PATH = "another_path_value", ALGORITHM = SM2);
```

4.5.62.CREATE COLUMN ENCRYPTION KEY

功能描述

创建一个列加密密钥，该密钥可用于加密表中指定列。

注意事项

本语法属于全密态数据库特有语法。

当使用 gsql 连接数据库服务器时，需使用‘-C’参数，打开全密态数据库的开关，才能使用本语法。

由该语法创建 CEK 对象可用于列级加密。在定义表中列字段时，可指定一个 CEK 对象，用于加密该列。

语法格式

```
CREATE COLUMN ENCRYPTION KEY column_encryption_key_name WITH VALUES(CLIENT_MASTER_KEY = client_master_key_name, ALGORITHM = algorithm_type, ENCRYPTED_VALUE = encrypted_value);
```

参数说明

- **column_encryption_key_name**

  该参数作为密钥对象名，在同一命名空间下，需满足命名唯一性约束。

  取值范围：字符串，要符合标识符的命名规范。

- **CLIENT_MASTER_KEY**

  指定用于加密本 CEK 的 CMK，取值为：CMK 对象名，该 CMK 对象由 CREATE CLIENT MASTER KEY 语法创建。

- **ALGORITHM**

  指定该 CEK 将用于何种加密算法，取值范围为：

  AEAD_AES_256_CBC_HMAC_SHA256、

  AEAD_AES_128_CBC_HMAC_SHA256 和 SM4_SM3；

- **ENCRYPTED_VALUE（可选项）**

  该值为用户指定的密钥口令，密钥口令长度范围为 28～256 位，28 位派生出来的密钥安全强度满足 AES128，若用户需要用 AES256，密钥口令的长度需要 39 位，如果不指定，则会自动生成 256 字符的密钥。

须知

国密算法约束：由于 SM2、SM3、SM4 等算法属于中国国家密码标准算法，为规避法律风险，需配套使用。如果创建 CMK 时指定 SM2 算法来加密 CEK，则创建 CEK 时必须指定 SM4_SM3 算法来加密数据。

示例

```
--创建列加密密钥(CEK)
panweidb=> CREATE COLUMN ENCRYPTION KEY a_cek WITH VALUES (CLIENT_MASTE
R_KEY = a_cmk, ALGORITHM  = AEAD_AES_256_CBC_HMAC_SHA256);
CREATE COLUMN ENCRYPTION KEY
panweidb=> CREATE COLUMN ENCRYPTION KEY another_cek WITH VALUES (CLIENT_
MASTER_KEY = a_cmk, ALGORITHM  = SM4_SM3);
CREATE COLUMN ENCRYPTION KEY
```

4.5.63.CREATE DATA SOURCE

功能描述

创建一个新的外部数据源对象，该对象用于定义 PanWeiDB 要连接的目标库信息。

注意事项

- Data Source 名称在数据库中需唯一，遵循标识符命名规范，长度限制为 63 字节，过长则会被截断。
- 只有系统管理员或初始用户才有权限创建 Data Source 对象。且创建该对象的用户为其默认属主。
- 当在 OPTIONS 中出现 password 选项时，需要保证 PanWeiDB 每个节点的$GAUSSHOME/bin 目录下存在 datasource.key.cipher 和 datasource.key.rand 文件，如果不存在这两个文件，请使用 pw_guc 工具生成并发布到 PanWeiDB 每个节点的$GAUSSHOME/bin 目录下。

语法格式

```
CREATE DATA SOURCE src_name
  [TYPE 'type_str']
```

```
[VERSION {'version_str' | NULL}]
[OPTIONS (optname 'optvalue' [, ...])];
```

参数说明

- **src_name**

  新建 Data Source 对象的名称，需在数据库内部唯一。

  取值范围：字符串，要符标识符的命名规范。

- **TYPE**

  新建 Data Source 对象的类型，可缺省。

  取值范围：空串或非空字符串。

- **VERSION**

  新建 Data Source 对象的版本号，可缺省或 NULL 值。

  取值范围：空串或非空字符串或 NULL。

- **OPTIONS**

  Data Source 对象的选项字段，创建时可省略，如若指定，其关键字如下：

  - optname
  - 选项名称。
  - 取值范围：dsn、 username、 password、 encoding。不区分大小写。
    - dsn 对应 odbc 配置文件中的 DSN。
    - username/password 对应连接目标库的用户名和密码。
    - PanWeiDB 在后台会对用户输入的 username/password 加密以保证安全性。该加密所需密钥文件需要使用 pw_guc 工具生成并发布到 PanWeiDB 每个节点的$GAUSSHOME/bin 目录下。username/password 不应当包含'encryptOpt'前缀，否则会被认为是加密后的密文。
    - encoding 表示与目标库交互的字符串编码方式（含发送的 SQL 语句和返回的字符类型数据），此处创建对象时不检查 encoding 取值的合法性，能否正确编解码取决于用户提供的编码方式是否在数据库本身支持的字符编码范围内。
  - optvalue

➢选项值。

➢取值范围：空或者非空字符串。

示例

```
--创建一个空的 Data Source 对象，不含任何信息。
panweidb=# CREATE DATA SOURCE ds_test1;

--创建一个 Data Source 对象，含 TYPE 信息，VERSION 为 NULL。
panweidb=# CREATE DATA SOURCE ds_test2 TYPE 'MPPDB' VERSION NULL;

--创建一个 Data Source 对象，仅含 OPTIONS。
panweidb=#  CREATE DATA SOURCE ds_test3 OPTIONS (dsn 'PanWeiDB', encoding '
utf8');

--创建一个 Data Source 对象，含 TYPE, VERSION, OPTIONS。
panweidb=# CREATE DATA SOURCE ds_test4 TYPE 'unknown' VERSION '11.2.3' OPTI
ONS (dsn 'PanWeiDB', username 'userid', password 'pwd@123456', encoding '');

--删除 Data Source 对象。
panweidb=# DROP DATA SOURCE ds_test1;
panweidb=# DROP DATA SOURCE ds_test2;
panweidb=# DROP DATA SOURCE ds_test3;
panweidb=# DROP DATA SOURCE ds_test4;
```

相关链接

参考：SQL 语法参考->SQL 语法中的 ALTER DATA SOURCE、DROP DATA SOURCE 章节。

4.5.64.CREATE DATABASE

功能描述

CREATE DATABASE 语法用于创建一个新的数据库。缺省情况下新数据库将通过复制标准系统数据库 template0 来创建。支持创建数据库时自定义指定模板。

注意事项

- 只有拥有 CREATEDB 权限的用户才可以创建新数据库，系统管理员默认拥有此权限。
- 不能在事务块中执行创建数据库语句。
- 在创建数据库过程中，出现类似“Permission denied”的错误提示，可能是由于文件系统上数据目录的权限不足。出现类似“No space left on device”的错误提示，可能是由于磁盘满引起的。
- 事务中不支持创建 database。
- 当新建数据库 Encoding、LC-Collate 或 LC_Ctype 与模板数据库（SQL_ASCII）不匹配（为'GBK' /'UTF8'/'LATIN1'）时，必须指定 template [=] template0。

语法格式

```
CREATE DATABASE [ IF NOT EXISTS ] database_name
  [ [ WITH ] {[ OWNER [=] user_name ]|
      [ TEMPLATE [=] template ]|
      [ ENCODING [=] encoding ]|
      [ LC_COLLATE [=] lc_collate ]|
      [ LC_CTYPE [=] lc_ctype ]|
      [ TABLESPACE [=] tablespace_name ]|
      [ MAXSIZE [=] tablespace_maxsize ]|
      [ THRESHOLD [=] warn_threshold ]|
      [ CONNECTION LIMIT [=] connlimit ]}[...] ]|
      [ PAD_ATTRIBUTE [=] pad_attribute_type ];
```

参数说明

- **database_name**

  数据库名称。支持使用中文名称

  取值范围：字符串，要符合标识符的命名规范。

- **OWNER [=] user_name**

  数据库所有者。

  取值范围：已存在的用户名。

  默认值：当前用户。

- **TEMPLATE [=] template**

  模板名。即从哪个模板创建新数据库。PanWeiDB 采用从模板数据库复制的方式来创建新的数据库。初始时，PanWeiDB 包含两个模板数据库 template0、template1，以及一个默认的用户数据库 postgres。

  取值范围：template0，用户自定义数据库。

- **ENCODING [=] encoding**

  指定数据库使用的字符编码，可以是字符串（如'SQL_ASCII'）、整数编号。

  不指定时，默认使用模版数据库的编码。模板数据库 template0 和 template1 的编码默认与操作系统环境相关。template1 不允许修改字符编码，因此若要变更编码，请使用 template0 创建数据库。

  常用取值：GBK、UTF8、Latin1。

表 1 PanWeiDB 字符集

名称	描述	语言	是否服务器端	ICU	字节/字符	别名
BIG5	Big Five	繁体中文	否	否	1-2	WIN950, Windows950
EUC_CN	扩展 UNIX 编码-中国	简体中文	是	是	1-3	-
EUC_JP	扩展 UNIX 编	日文	是	是	1-3	-

名称	描述	语言	是否服务器端	ICU	字节/字符	别名
	码-日本					
EUC_JIS_2004	扩展 UNIX 编码-日本, JIS X 0213	日文	是	否	1-3	-
EUC_KR	扩展 UNIX 编码-韩国	韩文	是	是	1-3	-
EUC_TW	扩展 UNIX 编码-中国台湾	繁体中文	是	是	1-3	-
GB18030	国家标准	中文	是	否	1-4	-
GBK	扩展国家标准	简体中文	是	否	1-2	WIN936, Windows936
ISO_8859_5	ISO 8859-5, ECMA 113	拉丁语/西里尔语	是	是	1	-
ISO_8859_6	ISO 8859-6, ECMA 114	拉丁语/阿拉伯语	是	是	1	-
ISO_8859_7	ISO 8859-7, ECMA 118	拉丁语/希腊语	是	是	1	-
ISO_8	ISO 8859-	拉丁语/	是	是	1	-

名称	描述	语言	是否服务器端	ICU	字节/字符	别名
859_8	8, ECMA 121	希伯来语				
JOHAB	JOHAB	韩语	否	否	1-3	-
KOI8R	KOI8-R	西里尔语（俄语）	是	是	1	KOI8
KOI8U	KOI8-U	西里尔语（乌克兰语）	是	是	1	-
LATIN1	ISO 8859-1, ECMA 94	西欧	是	是	1	ISO88591
LATIN2	ISO 8859-2, ECMA 94	中欧	是	是	1	ISO88592
LATIN3	ISO 8859-3, ECMA 94	南欧	是	是	1	ISO88593
LATIN4	ISO 8859-4, ECMA 94	北欧	是	是	1	ISO88594
LATIN5	ISO 8859-9, ECMA 128	土耳其语	是	是	1	ISO88599
LATIN6	ISO 8859-10, ECMA 144	日耳曼语	是	是	1	ISO885910

名称	描述	语言	是否服务器端	ICU	字节/字符	别名
LATIN7	ISO 8859-13	波罗的海	是	是	1	ISO885913
LATIN8	ISO 8859-14	凯尔特语	是	是	1	ISO885914
LATIN9	ISO 8859-15	带欧罗巴和口音的LATIN1	是	是	1	ISO885915
LATIN10	ISO 8859-16, ASRO SR 14111	罗马尼亚语	是	否	1	ISO885916
MULE_INTERNAL	Mule 内部编码	多语种编辑器	是	否	1-4	-
SJIS	Shift JIS	日语	否	否	1-2	Mskanji, ShiftJIS, WIN932, Windows932
SHIFT_JIS_2004	Shift JIS, JIS X 0213	日语	否	否	1-2	-
SQL_ASCII	未指定（见文本）	任意	是	否	1	-
UHC	统一韩语	韩语	否	否	1-2	WIN949

名称	描述	语言	是否服务器端	ICU	字节/字符	别名
	编码					，Windows949
UTF8	Unicode, 8-bit	所有	是	是	1-4	Unicode
WIN866	Windows CP866	西里尔语	是	是	1	ALT
WIN874	Windows CP874	泰语	是	否	1	-
WIN1250	Windows CP1250	中欧	是	是	1	-
WIN1251	Windows CP1251	西里尔语	是	是	1	WIN
WIN1252	Windows CP1252	西欧	是	是	1	-
WIN1253	Windows CP1253	希腊语	是	是	1	-
WIN1254	Windows CP1254	土耳其语	是	是	1	-
WIN1255	Windows CP1255	希伯来语	是	是	1	-
WIN1256	Windows CP1256	阿拉伯语	是	是	1	-
WIN1257	Windows CP1257	波罗的海	是	是	1	-
WIN1	Windows	越南语	是			

名称	描述	语言	是否服务器端	ICU	字节/字符	别名
258	CP1258					

【注意】

- 并非所有的客户端 API 都支持上面列出的字符集。
- SQL_ASCII 设置与其他设置表现得相当不同。如果服务器字符集是 SQL_ASCII，服务器把字节值 0-127 根据 ASCII 标准解释，而字节值 128-255 则当作无法解析的字符。如果设置为 SQL_ASCII，就不会有编码转换。因此，这个设置基本不是用来声明所使用的指定编码， 因为这个声明会忽略编码。在大多数情况下，如果你使用了任何非 ASCII 数据，那么使用 SQL_ASCII 设置都是不明智的，因为 PanWeiDB 将无法帮助你转换或者校验非 ASCII 字符。

【须知】

- 指定新的数据库字符集编码必须与所选择的本地环境中（LC_COLLATE 和 LC_CTYPE）的设置兼容。
- 当指定的字符编码集为 GBK 时，部分中文生僻字无法直接作为对象名。这是因为 GBK 第二个字节的编码范围在 0x40-0x7E 之间时，字节编码与 ASCII 字符@A-Z[\]^_a-z{|}重叠。其中@[]^_{|}是数据库中的操作符，直接作为对象名时，会语法报错。例如“俦”字，GBK16 进制编码为 0x8240，第二个字节为 0x40，与 ASCII“@”符号编码相同，因此无法直接作为对象名使用。如果确实要使用，可以在创建和访问对象时，通过增加双引号来规避这个问题。
- 若客户端编码为 A，服务器端编码为 B，则需要满足数据库中存在编码格式 A 与 B 的转换。数据库能够支持的所有的编码格式转换详见系统表 PG_CONVERSION（参见《PanWeiDB V2.0-S3.0.1_B01 开发者指南》->系统表->PG_CONVERSION）（若无法转换，则建议客户端编码与服务器端编码保持一致，客户端编码可通过 GUC 参数 client_encoding 修改）。

- **LC_COLLATE [=] lc_collate**

指定新数据库使用的字符集。例如，通过 lc_collate = 'zh_CN.gbk'设定该参数。

该参数的使用会影响到对字符串的排序顺序（如使用 ORDER BY 执行，以及在文本列上使用索引的顺序）。默认是使用模板数据库的排序顺序。

取值范围：操作系统支持的字符集。

- **LC_CTYPE [=] lc_ctype**

  指定新数据库使用的字符分类。例如，通过 lc_ctype = 'zh_CN.gbk'设定该参数。该参数的使用会影响到字符的分类，如大写、小写和数字。默认是使用模板数据库的字符分类。

  取值范围：操作系统支持的字符分类。

【须知】

对于 lc_collate 和 lc_ctype 参数的取值范围，取决于本地环境支持的字符集。

例如：在 Linux 操作系统上，可通过 locale -a 命令获取操作系统支持的字符集列表，在应用 lc_collate 和 lc_ctype 参数时可从中选择用户需要的字符集和字符分类。

- **TABLESPACE [=] tablespace_name**

  指定数据库对应的表空间。

  取值范围：已存在表空间名。

- **CONNECTION LIMIT [=] connlimit**

  数据库可以接受的并发连接数。

  取值范围：>=-1 的整数。

  默认值：-1，表示没有限制。

【须知】

系统管理员不受此参数的限制。

connlimit 数据库主节点单独统计，PanWeiDB 整体的连接数 = connlimit * 当前正常数据库主节点个数。

- **[PAD_ATTRIBUTE [=] pad_attribute_type]**

  为新数据库设置默认的列校对规则。

【须知】

该功能仅在数据库兼容模式为 MySQL 时能够使用（即数据库初始化时指定 DBCOMPATIBILITY='B'）。详情参考 MySQL 兼容性手册中的 CREATE DATABASE。

有关字符编码的一些限制：

- 若区域设置为 C（或 POSIX），则允许所有的编码类型，但是对于其他的区域设置，字符编码必须和区域设置相同。
- 若字符编码方式是 SQL_ASCII，并且修改者为管理员用户时，则字符编码可以和区域设置不相同。
- 编码和区域设置必须匹配模板数据库，除了将 template0 当作模板。因为其他数据库可能会包含不匹配指定编码的数据，或者可能包含排序顺序受 LC_COLLATE 和 LC_CTYPE 影响的索引。复制这些数据会导致在新数据库中的索引失效。template0 是不包含任何会受到影响的数据或者索引。

示例

示例 1：创建数据库。

1、创建 jim 和 tom 用户。

```
CREATE USER jim PASSWORD 'Aa@12345';

CREATE USER tom PASSWORD 'Aa@12345';
```

2、创建一个"UTF-8"编码的数据库 music。

```
CREATE DATABASE music ENCODING 'UTF-8' template = template0;
```

3、创建数据库 music2，并指定所有者为 jim。

```
CREATE DATABASE music2 OWNER jim;
```

4、用模板 template0 创建数据库 music3，并指定所有者为 jim。

```
CREATE DATABASE music3 OWNER jim TEMPLATE template0;
```

5、设置 music 数据库的连接数为 10。

```
ALTER DATABASE music CONNECTION LIMIT= 10;
```

6、将 music 名称改为 music4。

```
ALTER DATABASE music RENAME TO music4;
```

7、将数据库 music2 的所属者改为 tom。

```
ALTER DATABASE music2 OWNER TO tom;
```

8、设置 music3 的表空间为 PG_DEFAULT。

```
ALTER DATABASE music3 SET TABLESPACE PG_DEFAULT;
```

9、关闭在数据库 music3 上缺省的索引扫描。

```
ALTER DATABASE music3 SET enable_indexscan TO off;
```

10、重置 enable_indexscan 参数。

```
ALTER DATABASE music3 RESET enable_indexscan;
```

11、删除数据库和用户。

```
DROP DATABASE music2;

DROP DATABASE music3;

DROP DATABASE music4;

DROP USER jim;

DROP USER tom;
```

示例 2：创建中文名称的数据库。

1、创建数据库。

```
create database 数据库;
```

2、创建英文名称的数据库。

```
create database db_1096053;
```

3、将英文数据库名称修改为中文。

```
alter database db_1096053 rename to 新数据库;
```

4、查询数据库。

```
\l
```

返回结果为：

```
                    List of databases
    Name    | Owner   |Encoding| Collate   |  Ctype    |    Access privileges
-----------+----------+----------+------------+------------+-----------------------
----
 postgres  |panweidb|UTF8    |en_US.utf8|en_US.utf8|
 template0|panweidb|UTF8    |en_US.utf8|en_US.utf8|=c/panweidb
+
          |       |       |         |         |panweidb=CTc/panweidb
 template1|panweidb|UTF8    |en_US.utf8|en_US.utf8|=c/panweidb
+
          |       |       |         |         |panweidb=CTc/panweidb


 数据库    |panweidb|UTF8    |en_US.utf8|en_US.utf8|
 新数据库  |panweidb|UTF8    |en_US.utf8|en_US.utf8|
(5 rows)
```

示例 3：使用自定义模板创建数据库。

1、 创建一个数据库 A 并切换至数据库 A。

```
create database A;
\c A
```

2、在数据库 A 中创建表并插入数据。

```
create table aa_table(a int);
insert into aa_table values(1);
```

3、以数据库 A 为模板创建数据库 B 并切换至数据库 B（模板数据库的兼容类型必须和新创建的数据库兼容类型一致）。

```
create database B template = A;

\c B
```

4、查询新创建的数据库 B 中是否有 A 中所创建的表。

```
select * from aa_table;
```

返回结果如下，表示支持使用自定义数据库模板创建数据库。

```
 a
---
 1
(1 row)
```

4.5.65.CREATE DIRECTORY

功能描述

使用 CREATE DIRECTORY 语句创建一个目录对象，该目录对象定义了服务器文件系统上目录的别名，用于存放用户使用的数据文件。

注意事项

- 当 enable_access_server_directory=off 时，只允许初始用户创建 directory 对象；当 enable_access_server_directory=on 时，具有 SYSADMIN 权限的用户和继承了内置角色 gs_role_directory_create 权限的用户可以创建 directory 对象。
- 创建用户默认拥有此路径的 READ 和 WRITE 操作权限。
- 目录的默认 owner 为创建 directory 的用户。
- 以下路径禁止创建：
  - 路径含特殊字符。
  - 路径是相对路径。

- 路径是符号连接。

❖ 创建目录时会进行以下合法性校验：

- 创建时会检查添加路径是否为操作系统实际存在路径，如不存在会提示用户使用风险。
- 创建时会校验数据库初始化（panweidb）用户对于添加路径的权限(即操作系统目录权限，读/写/执行 - R/W/X)，如果权限不全，会提示用户使用风险。

❖ 在 PanWeiDB 环境下用户指定的路径需要用户保证各节点上路径的一致性，否则在不同节点上执行会产生找不到路径的问题。

语法格式

```
CREATE [OR REPLACE] DIRECTORY directory_name
AS 'path_name';
```

参数说明

❖ **directory_name**

目录名称。

取值范围：字符串，要符合标识符的命名规范。

❖ **path_name**

操作系统的路径。

取值范围： 有效的操作系统路径。

示例

```
--创建目录。
panweidb=# CREATE OR REPLACE DIRECTORY  dir  as '/tmp/';
```

相关链接

参考：SQL 语法参考->SQL 语法中的 ALTER DIRECTORY、DROP DIRECTORY 章节。

功能描述

创建 DDL 触发器。

注意事项

只有超级用户才能创建 DDL 触发器。

在单用户模式下禁用 DDL 触发器。如果错误的 DDL 触发器禁用了数据库而无法取消触发器，请以单用户模式重新启动。

语法格式

```
CREATE EVENT TRIGGER name
  ON event [ ON DATABASE | SCHEMA ]
  [ WHEN filter_variable IN (filter_value [, ... ]) [ AND ... ] ]
  EXECUTE PROCEDURE function_name()

CREATE EVENT TRIGGER name
  { AFTER LOGON | BEFORE LOGOFF } ON DATABASE
  EXECUTE PROCEDURE function_name()
```

参数说明

- **name**

  DDL 触发器名称。在该数据库中这个名称必须唯一。

- **ON DATABASE**

  表示 DDL 触发器绑定于 database 对象，LOGON 与 LOGOFF 事件仅支持指定 ON DATABASE。

- **ON SCHEMA**

  表示 DDL 触发器绑定于 schema 对象。

- **event**

  会触发对给定函数调用的事件名称，分别为 ddl_command_start、ddl_command_end、sql_drop、table_rewrite、LOGON 和 LOGOFF。

  - ddl_command_start：在 CREATE、ALTER、DROP、SECURITY LABEL、COMMENT、GRANT 或 REVOKE 命令执行前触发。

- ddl_command_end： 在 CREATE、ALTER、DROP、SECURITY LABEL、COMMENT、GRANT 或 REVOKE 命令执行后触发。
- sql_drop：为任何删除数据库对象的操作在 ddl_command_end DDL 触发器之前触发。
- table_rewrite：在表被命令 ALTER TABLE 和 ALTER TYPE 的某些动作重写之前触发。
- LOGON：用户登录后触发，仅支持 AFTER LOGON。
- LOGOFF：用户登出前触发，仅支持 BEFORE LOGOFF。

❖ **filter_variable**

用来过滤事件的变量名称。这可以用来限制触发器的触发事件。当前仅支持通过 TAG 过滤。

TAG 是 SQL 命令执行后的客户端的返回值，例如执行 SQL 语句 CREATE TABLE fruit (id INTEGER)； ，则返回的 TAG 为 CREATE TABLE。

❖ **filter_value**

与该触发器要为其引发的 filter_variable 相关联的一个值列表。对于 TAG 这表示一个命令标签列表（例如 DROP FUNCTION）。

❖ **function_name**

一个用户提供的函数。触发器函数需要声明为无参数，且返回类型为 Event Trigger 的函数。

示例

示例 1： 创建 ddl_command_start DDL 触发器，指定多条命令触发。

1、创建 DDL 触发器函数。

```
CREATE OR REPLACE FUNCTION eth1()
  RETURNS event_trigger
  LANGUAGE plpgsql
  AS $$ BEGIN
    RAISE NOTICE 'test_event_trigger: % %', tg_event, tg_tag;
  END;
```

```
$$;
```

2、创建 DDL 触发器。

```
 CREATE EVENT TRIGGER estart ON ddl_command_start WHEN TAG IN ('CREATE
TABLE','ALTER TABLE','DROP TABLE') EXECUTE PROCEDURE eth1();
3、执行 DDL 触发器中指定命令 CREATE TABLE。
 CREATE TABLE  t_eventtr(id int);
```

结果返回如下：

```
NOTICE:  test_event_trigger: ddl_command_start CREATE TABLE
CREATE TABLE
```

4、执行 DDL 触发器中指定命令 ALTER TABLE。

```
ALTER TABLE t_eventtr RENAME TO t_eventtr_new;
```

结果返回如下：

```
NOTICE:  test_event_trigger: ddl_command_start ALTER TABLE
ALTER TABLE
```

5、执行 DDL 触发器中指定命令 DROP TABLE。

```
DROP TABLE t_eventtr_new;
```

结果返回如下：

```
NOTICE:  test_event_trigger: ddl_command_start DROP TABLE
DROP TABLE
```

示例 2：创建 ddl_command_end DDL 触发器。

1、创建 DDL 触发器函数。

```
CREATE OR REPLACE FUNCTION eth1()
  RETURNS event_trigger
  LANGUAGE plpgsql
  AS $$ BEGIN
    RAISE NOTICE 'test_event_trigger: % %', tg_event, tg_tag;
  END;
$$;
```

2、创建测试表 t_eventtr。

```
CREATE TABLE t_eventtr(id int);
```

返回结果如下：

```
NOTICE:  test_event_trigger: ddl_command_start CREATE TABLE
CREATE TABLE
```

3、创建 DDL 触发器。

```
CREATE EVENT TRIGGER eend ON ddl_command_end EXECUTE PROCEDURE  eth1();
```

4、删除测试表 t_eventtr。

```
DROP TABLE t_eventtr;
```

结果返回如下：

```
NOTICE:  test_event_trigger: ddl_command_end DROP TABLE
DROP TABLE
```

示例 3： 创建 sql_drop DDL 触发器。

1、创建 DDL 触发器函数。

```
CREATE OR REPLACE FUNCTION test_event_trigger_drop()
  RETURNS event_trigger LANGUAGE plpgsql AS $$
  DECLARE
    obj record;
  BEGIN
    FOR obj IN SELECT * FROM pg_event_trigger_dropped_objects()
    LOOP
      RAISE NOTICE '% dropped object: % %.% %',
      tg_tag,
      obj.object_type,
      obj.schema_name,
      obj.object_name,
      obj.object_identity;
    END LOOP;
```

```
  END;
$$;
```

2、创建测试表 t_eventtr。

```
CREATE TABLE t_eventtr(id int);
```

3、创建 DDL 触发器 edrop。

```
CREATE EVENT TRIGGER edrop ON sql_drop EXECUTE PROCEDURE test_event_
trigger_drop();
```

4、删除表 t_eventtr。

```
DROP TABLE t_eventtr;
```

结果返回如下：

```
NOTICE:  DROP TABLE dropped object: table public.t_eventtr public.t_eventtr
 NOTICE:  DROP TABLE dropped object: type public.t_eventtr public.t_eventtr
 NOTICE:  DROP TABLE dropped object: type public._t_eventtr public.t_eventtr
[]
 DROP TABLE
```

示例 4： 创建 table_rewrite DDL 触发器。

1、创建 DDL 触发器函数。

```
CREATE OR REPLACE FUNCTION eth1()
   RETURNS event_trigger
   LANGUAGE plpgsql
   AS $$ BEGIN
     RAISE NOTICE 'test_event_trigger: % %', tg_event, tg_tag;
   END;
$$;
```

2、创建 DDL 触发器。

```
CREATE EVENT TRIGGER erewrite ON table_rewrite EXECUTE PROCEDURE eth1();
```

示例 5： 修改 DDL 触发器。

1、创建 DDL 触发器函数。

```
CREATE OR REPLACE FUNCTION eth1() RETURNS event_trigger AS $$ BEGIN RAIS
E NOTICE 'test_event_trigger: % %',tg_event,tg_tag; END $$ LANGUAGE plpgsql;
```

2、创建 DDL 触发器。

```
CREATE EVENT TRIGGER logon_event_trigger AFTER logon ON database EXECU
TE PROCEDURE eth1();
```

3、执行以下语句修改 DDL 触发器名称。

```
ALTER EVENT TRIGGER logon_event_trigger RENAME TO test_eventtri_new;
```

4、系统表中查询改名后的 DDL 触发器。

```
SELECT * FROM pg_event_trigger WHERE evtname = 'test_eventtri_new';
```

结果返回如下：

```
    evtname     | evtevent | evtowner | evtfoid | evtenabled | isschema | evttags
 ------------------+----------+----------+---------+------------+----------+-----
----
  test_eventtri_new | logon    |       10 |   18586 | O          | f        |
 (1 row)
```

示例 6：创建 Logon/Logoff 系统 DDL 触发器。

1、创建测试表 test2。

```
create table test2(id int,col timestamp);
```

2、创建触发器函数 eth3()和 eth4()。

```
CREATE OR REPLACE FUNCTION eth3()
  RETURNS event_trigger
  LANGUAGE plpgsql
  AS $$ BEGIN
    insert into test2 values(1,systimestamp);
  END;
$$;
```

```
CREATE OR REPLACE FUNCTION eth4()
 RETURNS event_trigger
 LANGUAGE plpgsql
 AS $$ BEGIN
   insert into test2 values(2,systimestamp);
 END;
$$;
```

3、创建 Logon/Logoff DDL 触发器。

```
CREATE EVENT TRIGGER logon_eventtri after Logon ON database
  EXECUTE PROCEDURE eth3();

CREATE EVENT TRIGGER logoff_eventtri before Logoff ON database
  EXECUTE PROCEDURE eth4();
```

4、退出数据库重新登陆。

```
\q
```

退出后在命令行执行登陆命令。

```
gsql -r
```

5、查询 test2 中记录的 Logon/Logoff 事件。

```
SELECT * FROM test2;
结果返回如下：
id |       col
----+---------------------
 2 | 2023-02-16 11:39:30
 1 | 2023-02-16 11:39:32
(2 rows)
```

相关链接

ALTER EVENT TRIGGER(参见：开发者指南->SQL 语法参考->SQL 语法->ALTER EVENT TRIGGER 章节),DROP EVENT TRIGGER(参见：开发者指南->SQL 语法参考->SQL 语法->DROP EVENT TRIGGER 章节)

4.5.66.CREATE EXTENSION

功能描述

安装一个扩展。

注意事项

- CREATE EXTENSION 命令安装一个新的扩展到一个数据库中，必须保证没有同名的扩展已经被安装。
- 安装一个扩展意味着执行一个扩展的脚本文件，这个脚本会创建一个新的 SQL 实体，例如函数、数据类型、操作符、和索引支持的方法。
- 安装扩展需要有和创建他的组件对象相同的权限。对于大多数扩展这意味着需要超户或者数据库所有者的权限，对于后续的权限检查和该扩展脚本所创建的实体，运行 CREATE EXTENSION 命令的角色将变为扩展的所有者。

语法格式

```
CREATE EXTENSION [ IF NOT EXISTS ] extension_name
[ WITH ] [ SCHEMA schema_name ]
[ VERSION version ]
[ FROM old_version ];
```

参数说明

- **IF NOT EXISTS**

  如果系统已经存在一个同名的扩展，不会报错。这种情况下会给出一个提示。请注意该参数不保证系统存在的扩展和现在脚本创建的扩展相同。

- **extension_name**

  将被安装扩展的名字。

- **schema_name**

  扩展的实例被安装在该模式下，扩展的内容可以被重新安装。指定的模式必须已经存在，如果没有指定，扩展的控制文件也不指定一个模式，这样将使用默认模式。

  注意：

扩展不认为它在任何模式里面：扩展在一个数据库范围内的名字是不受限制的，但是一个扩展的实例是属于一个模式的。

- **version**

  安装扩展的版本，可以写为一个标识符或者字符串.默认的版本在扩展的控制文件中指定。

- **old_version**

  当你想升级安装“old style”模块中没有的内容时,你必须指定 FROM old_version。这个选项使 CREATE EXTENSION 运行一个安装脚本将新的内容安装到扩展中，而不是创建一个新的实体.注意 SCHEMA 指定了包括这些已存在实体的模式。

示例

在当前数据库安装 hstore 扩展：

```
CREATE EXTENSION hstore;
```

4.5.67.CREATE FOREIGN DATA WRAPPER

功能描述

定义一个新的外部数据包装器。

语法格式

```
CREATE FOREIGN DATA WRAPPER name
  [ HANDLER handler_function | NO HANDLER ]
  [ VALIDATOR validator_function | NO VALIDATOR ]
  [ OPTIONS ( option 'value' [,...] ) ]
```

参数说明

- **name**

  要创建的外部数据包装器名称。

- **handler_function**

  handler_function 是先前注册的函数的名称，该函数将被调用以检索外部表的执行函数。

  处理器函数不能带任何参数，其返回类型必须是 fdw_handler。

- **validator_function**

  validator_function 是先前注册的函数的名称，该函数将被调用以检查给定给外部数据包装器的通用选项，以及使用外部数据包装器的外部服务器和用户映射的选项。（外部数据包装器可能会在运行时忽略或拒绝无效的选项规范，具体取决于实现。）验证器函数必须接受两个参数：一个类型为 text[]，它将包含存储在系统目录中的选项数组，另一个类型是 oid，它将是包含选项的系统目录的 oid。忽略返回类型；该函数应该使用 ereport（ERROR）函数报告无效选项。

- **OPTIONS (option 'value' [,...])**

  该子句用于指定新外部数据包装器的选项。允许的选项名称和值特定于每个外部数据包装器，并使用外部数据包装器的验证器函数进行验证。选项名称必须唯一。

示例

- 创建一个无用的外部数据包装器 dummy。

```
CREATE FOREIGN DATA WRAPPER dummy;
```

- 使用处理器函数 file_fdw_handler 创建外部数据包装器 file。

```
CREATE FOREIGN DATA WRAPPER file HANDLER file_fdw_handler;
```

- 创建外部数据包装器 mywrapper

```
CREATE FOREIGN DATA WRAPPER mywrapper OPTIONS (debug 'true');
```

4.5.68.CREATE FOREIGN TABLE

功能描述

创建外表。

注意事项

- 外表中暂不支持使用系统列（如 tableoid、ctid 等），其中 Private 和 Shares 模式的外表，需要初始用户和运维模式下（operation_mode）的运维管理员权限。
- 不支持使用 default values。

语法格式

```
CREATE FOREIGN TABLE [ IF NOT EXISTS  ] table_name ( {
  column_name type_name POSITION ( offset, length ) [column_constraint ]
    | LIKE source_table | table_constraint } [, ...] )
    SEVER gsmpp_server
    OPTIONS (  { option_name ' value '  }  [, ...] )
    [ { WRITE ONLY  |  READ ONLY  }]
    [ WITH error_table_name | LOG INTO error_table_name ]
    [ REMOTE LOG 'name' ]
    [PER NODE REJECT LIMIT 'value']
    [ TO { GROUP groupname | NODE ( nodename [, ... ] ) } ];
CREATE FOREIGN TABLE [ IF NOT EXISTS ] table_name ( {
  column_name type_name
  [ { [CONSTRAINT constraint_name] NULL |
  [CONSTRAINT constraint_name] NOT NULL |
    column_constraint [...]} ] |
    table_constraint} [, ...] )
    SERVER server_name
    OPTIONS ( { option_name ' value ' } [, ...] )
    DISTRIBUTE BY {ROUNDROBIN | REPLICATION}
    [ TO { GROUP groupname | NODE ( nodename [, ... ] ) } ]
    [ PARTITION BY ( column_name ) [AUTOMAPPED]] ;
CREATE FOREIGN TABLE [ IF NOT EXISTS ] table_name ( [ {
  column_name type_name | LIKE source_table } [, ...] ] )
    SERVER server_name
    OPTIONS ( { option_name ' value ' } [, ...] )
    [ READ ONLY ]
    [ DISTRIBUTE BY {ROUNDROBIN} ]
    [ TO { GROUP groupname | NODE ( nodename [, ... ] ) } ];
```

这里 column_constraint 可以是：

```
[ CONSTRAINT constraint_name ]
{ PRIMARY KEY | UNIQUE }
[ NOT ENFORCED [ ENABLE QUERY OPTIMIZATION | DISABLE QUERY OPTIMIZATION ] |
ENFORCED ]
where table_constraint can be:
[ CONSTRAINT constraint_name ]
```

```
{ PRIMARY KEY | UNIQUE } ( column_name )
[ NOT ENFORCED [ ENABLE QUERY OPTIMIZATION | DISABLE QUERY OPTIMIZATION ] |
ENFORCED ]
```

参数说明

- **IF NOT EXISTS**

  如果已经存在相同名称的表，不会抛出一个错误，而会发出一个通知，告知表关系已存在。

- **table_name**

  外表的表名。

  取值范围：字符串，要符合标识符的命名规范。

- **column_name**

  外表中的字段名。

  取值范围：字符串，要符合标识符的命名规范。

- **type_name**

  字段的数据类型。

- **SERVER server_name**

  外表的 server 名称。默认值为 mot_server。

- **OPTIONS (option 'value' [, ...])**

  选项与新外部表或外部表中的字段有关。允许的选项名称和值，是由每一个外部数据封装器指定的。 也是通过外部数据封装器的验证函数来验证。重复的选项名称是不被允许的（尽管表选项和表字段选项可以有相同的名字）。

  - oracle_fdw 支持的 options 包括：
    - **table**

      oracle server 侧的表名。需要同 oracle 系统表中记录的表名完全一致，通常是由大写字符组成。

    - **schema**

      表所对应的 schema（或 owner）。需要同 oracle 系统表中记录的表名完全一致，通常是由大写字符组成。

- mysql_fdw 支持的 options 包括：
  - **dbname**

    MySQL 的 database 名称。
  - **table_name**

    MySQL 侧的表名。
- postgres_fdw 支持的 options 包括：
  - **schema_name**

    远端 server 的 schema 名称。如果不指定的话，将使用外表自身的 schema 名称作为远端的 schema 名称。
  - **table_name**

    远端 server 的表名。如果不指定的话，将使用外表自身的表名作为远端的表名。
  - **column_name**

    远端 server 的表的列名。如果不指定的话，将使用外表自身的列名作为远端的的表的列名。
- file_fdw 支持的 options 包括：
  - filename

    指定要读取的文件，必需的参数，且必须是一个绝对路径名。
  - format

    远端 server 的文件格式，支持 text/csv/binary/fixed 四种格式，和 COPY 语句的 FORMAT 选项相同。
  - header

    指定的文件是否有标题行，与 COPY 语句的 HEADER 选项相同。

  - delimiter

    指定文件的分隔符，与 COPY 的 DELIMITER 选项相同。

  - quote

    指定文件的引用字符，与 COPY 的 QUOTE 选项相同。

  - escape

    指定文件的转义字符，与 COPY 的 ESCAPE 选项相同。

  - null

指定文件的 null 字符串，与 COPY 的 NULL 选项相同。

- encoding

指定文件的编码，与 COPY 的 ENCODING 选项相同。

- force_not_null

这是一个布尔选项。如果为真，则声明字段的值不应该匹配空字符串（也就是文件级别 null 选项）。与 COPY 的 FORCE_NOT_NULL 选项里的字段相同。

- jdbc_fdw 支持的 options 包括：
  - table_name：

    远端的表名。
  - query：

    指定要查询的 select 语句。

相关链接

参考：SQL 语法参考->SQL 语法中的 ALTER FOREIGN TABLE、DROP FOREIGN TABLE 章节。

4.5.69.CREATE GROUP

功能描述

创建一个新用户组。

注意事项

CREATE GROUP 是 CREATE ROLE 的别名，非 SQL 标准语法，不推荐使用，建议用户直接使用 CREATE ROLE 替代。

语法格式

```
CREATE GROUP group_name [ [ WITH ] option [ ... ] ] [ ENCRYPTED | UNENCRYPTED ] { PASSWORD | IDENTIFIED BY } { 'password' [ EXPIRED ] | DISABLE };
```

其中可选项 option 子句语法为：

```
{SYSADMIN | NOSYSADMIN}
```

```
| {MONADMIN | NOMONADMIN}
| {OPRADMIN | NOOPRADMIN}
| {POLADMIN | NOPOLADMIN}
| {AUDITADMIN | NOAUDITADMIN}
| {CREATEDB | NOCREATEDB}
| {USEFT | NOUSEFT}
| {CREATEROLE | NOCREATEROLE}
| {INHERIT | NOINHERIT}
| {LOGIN | NOLOGIN}
| {REPLICATION | NOREPLICATION}
| {INDEPENDENT | NOINDEPENDENT}
| {VCADMIN | NOVCADMIN}
| {PERSISTENCE | NOPERSISTENCE}
| CONNECTION LIMIT connlimit
| VALID BEGIN 'timestamp'
| VALID UNTIL 'timestamp'
| RESOURCE POOL 'respool'
| PERM SPACE 'spacelimit'
| TEMP SPACE 'tmpspacelimit'
| SPILL SPACE 'spillspacelimit'
| IN ROLE role_name [, ...]
| IN GROUP role_name [, ...]
| ROLE role_name [, ...]
| ADMIN rol e_name [, ...]
| USER role_name [, ...]
| SYSID uid
| DEFAULT TABLESPACE tablespace_name
| PROFILE DEFAULT
| PROFILE profile_name
| PGUSER
```

参数说明

请参考：SQL 语法参考->SQL 语法->CREATE ROLE 章节中的参数说明。

相关链接

参考：SQL 语法参考->SQL 语法中的 ALTER GROUP、DROP GROUP、CREATE ROLE 章节。

4.5.70.CREATE FUNCTION

功能描述

创建一个函数。

注意事项

- 如果创建函数时参数或返回值带有精度，不进行精度检测。
- 创建函数时，函数定义中对表对象的操作建议都显式指定模式，否则可能会导致函数执行异常。
- 在创建函数时，函数内部通过 SET 语句设置 current_schema 和 search_path 无效。执行完函数 search_path 和 current_schema 与执行函数前的 search_path 和 current_schema 保持一致。
- 如果函数参数中带有出参，SELECT 调用函数必须缺省出参，CALL 调用函数必须指定出参，对于调用重载的带有 PACKAGE 属性的函数，CALL 调用函数可以缺省出参，具体信息参见：SQL 语法参考->SQL 语法中 CALL 的示例。
- 兼容 Postgresql 风格的函数或者带有 PACKAGE 属性的函数支持重载。在指定 REPLACE 的时候，如果参数个数、类型、返回值有变化，不会替换原有函数，而是会建立新的函数。
- 不能创建仅形参名字不同（函数名和参数列表类型都一样)的重载函数。
- 重载的函数在调用时变量需要明确具体的类型。
- 不能创建与存储过程拥有相同名称和参数列表的函数。
- 不支持形式参数仅在自定义 ref cursor 类型和 sys_refcursor 类型不同的重载。
- 在函数内部使用未声明的变里，函数被调用时会报错。
- SELECT 调用可以指定不同参数来进行同名函数调用。由于语法不支持调用不带有 PACKAGE 属性的同名函数。

- 在创建 function 时，不能在 avg 函数外面嵌套其他 agg 函数或者其他系统函数。
- 新创建的函数默认会给 PUBLIC 授予执行权限（详见 SQL 语法参考->SQL 语法->GRANT 章节）。用户可以选择收回 PUBLIC 默认执行权限，然后根据需要将执行权限授予其他用户，为了避免出现新函数能被所有人访问的时间窗口，应在一个事务中创建函数并且设置函数执行权限。
- 在函数内部调用其他无参数的函数时，可以省略括号，直接使用函数名进行调用。
- 兼容 Oracle 风格的函数支持参数注释的查看与导出、导入。
- 兼容 Oracle 风格的函数支持介于 IS/AS 与 plsql_body 之间的注释的查看与导出、导入。
- 不可与同一模式下已存在的 synonym 产生命名冲突。
- 当 language 为 internal 时，自定义函数的参数类型和返回值类型（void 除外）需要与引用的系统函数保持一致，参数个数超出系统函数的部分不做对比校验。当未指定 strict 时，此选项与系统函数保持一致。
- 通过 CREATE OR REPLACE 语法替换已有的函数时，会一并重建依赖此函数的视图，函数中的参数数据类型变更等情况可能会导致重建视图失败，进而导致替换函数失败。此种情况下，建议先删除依赖的视图，再重建函数，再重新创建视图。

语法格式

- 兼容 PostgreSQL 风格的创建自定义函数语法。

```
CREATE [ OR REPLACE ] FUNCTION function_name
  ( [ { argname [ argmode ] argtype [ { DEFAULT | := | = } expression ]
 } [, ...] ] )
  [ RETURNS rettype [ DETERMINISTIC ]
    | RETURNS TABLE ( { column_name column_type } [, ...] )]
  LANGUAGE lang_name
  [
    {IMMUTABLE | STABLE | VOLATILE}
    | {SHIPPABLE | NOT SHIPPABLE}
```

```
    | [ NOT ] LEAKPROOF
    | WINDOW
    | {CALLED ON NULL INPUT | RETURNS NULL ON NULL INPUT | STRICT}
    | {[ EXTERNAL ] SECURITY INVOKER | [ EXTERNAL ] SECURITY DEFINER | AU
THID DEFINER | AUTHID CURRENT_USER}
    | {FENCED | NOT FENCED}
    | {PACKAGE}
    | COST execution_cost
    | ROWS result_rows
    | SET configuration_parameter { {TO | =} value | FROM CURRENT }
  ] [...]
  {
    AS 'definition'
    | AS 'obj_file', 'link_symbol'
  }
```

❖ 兼容 Oracle 风格的创建自定义函数的语法。

```
CREATE [ OR REPLACE ] FUNCTION function_name
    ( [ { argname [ argmode ] argtype [ { DEFAULT | := | = } expression ] }
  [, ...] ] )
    RETURN rettype [ DETERMINISTIC ]
    [
      {IMMUTABLE | STABLE | VOLATILE }
      | {SHIPPABLE | NOT SHIPPABLE}
      | {PACKAGE}
      | [ NOT ] LEAKPROOF
      | {CALLED ON NULL INPUT | RETURNS NULL ON NULL INPUT | STRICT }
      | {[ EXTERNAL ] SECURITY INVOKER | [ EXTERNAL ] SECURITY DEFINER | |
 AUTHID DEFINER | AUTHID CURRENT_USER}
      | COST execution_cost
      | ROWS result_rows
      | SET configuration_parameter { {TO | =} value | FROM CURRENT }
    ][...]
    {
      IS | AS
```

```
    } plsql_body
/
```

参数说明

- **function_name**

  要创建的函数名称（可以用模式修饰）。

  取值范围：字符串，要符合标识符的命名规范。且最多为 63 个字符。若超过 63 个字符，数据库会截断并保留前 63 个字符当做函数名称。

- **argname**

  函数参数的名称。

  取值范围：字符串，要符合标识符的命名规范。且最多为 63 个字符。若超过 63 个字符，数据库会截断并保留前 63 个字符当做函数参数名称。

- **argmode**

  函数参数的模式。

  取值范围：IN、OUT、INOUT 或 VARIADIC。缺省值是 IN。并且 OUT 和 INOUT 模式的参数不能用在 RETURNS TABLE 的函数定义中。

  说明：

  VARIADIC 用于声明数组类型的参数。

- **argtype**

  函数参数的类型。可以使用%TYPE 或%ROWTYPE 间接引用变量或表的类型，详细可参考：PL/SQL->存储过程->基本语句->定义变量章节。

- **expression**

  参数的默认表达式。

- **rettype**

  函数返回值的数据类型。

  如果存在 OUT 或 INOUT 参数，可以省略 RETURNS 子句。如果存在，该子句必须和输出参数所表示的结果类型一致：如果有多个输出参数，则为 RECORD，否则与单个输出参数的类型相同。

  SETOF 修饰词表示该函数将返回一个集合，而不是单独一项。

  与 argtype 相同，同样可以使用%TYPE 或%ROWTYPE 间接引用类型。

- **column_name**

字段名称。

- **column_type**

  字段类型。

- **definition**

  一个定义函数的字符串常量，含义取决于语言。它可以是一个内部函数名称、一个指向某个目标文件的路径、一个 SQL 查询、一个过程语言文本。

- **DETERMINISTIC**

  SQL 语法兼容接口，未实现功能，不推荐使用。

- **LANGUAGE lang_name**

  用以实现函数的语言的名称。可以是 SQL、internal 或者是用户定义的过程语言名称。为了保证向下兼容，该名称可以用单引号包围。若采用单引号，则引号内必须为小写。

【说明】

internel 函数在定义时，如果 AS 指定为内部系统函数，则新创建函数的参数类型，参数个数，与返回值类型需要与内部系统函数保持一致，且需要有执行此内部系统函数的权限

- **WINDOW**

  表示该函数是窗口函数。替换函数定义时不能改变 WINDOW 属性。

【须知】

自定义窗口函数只支持 LANGUAGE 是 internal，并且引用的内部函数必须是窗口函数。

- **IMMUTABLE**

  表示该函数在给出同样的参数值时总是返回同样的结果。

- **STABLE**

  表示该函数不能修改数据库，对相同参数值，在同一次表扫描里，该函数的返回值不变，但是返回值可能在不同 SQL 语句之间变化。

- **VOLATILE**

  表示该函数值可以在一次表扫描内改变，因此不会做任何优化。

- **SHIPPABLE|NOT SHIPPABLE**

  表示该函数是否可以下推执行。预留接口，不推荐使用。

- **FENCED|NOT FENCED**

  声明用户定义的 C 函数是在保护模式还是非保护模式下执行。预留接口，不推荐使用。

- **PACKAGE**

  表示该函数是否支持重载。PostgreSQL 风格的函数本身就支持重载，此参数主要是针对其他风格的函数。

  - 不允许 package 函数和非 package 函数重载或者替换。
  - package 函数不支持 VARIADIC 类型的参数。
  - 不允许修改函数的 package 属性。

- **LEAKPROOF**

  指出该函数的参数只包括返回值。LEAKPROOF 只能由系统管理员设置。

- **CALLED ON NULL INPUT**

  表明该函数的某些参数是 NULL 的时候可以按照正常的方式调用。该参数可以省略。

- **RETURNS NULL ON NULL INPUT**

  STRICT

  STRICT 用于指定如果函数的某个参数是 NULL，此函数总是返回 NULL。如果声明了这个参数，当有 NULL 值参数时该函数不会被执行；而只是自动返回一个 NULL 结果。

  RETURNS NULL ON NULL INPUT 和 STRICT 的功能相同。

- **EXTERNAL**

  目的是和 SQL 兼容，是可选的，这个特性适合于所有函数，而不仅是外部函数。

- **SECURITY INVOKER**

  AUTHID CURRENT_USER

  表明该函数将带着调用它的用户的权限执行。该参数可以省略。

  SECURITY INVOKER 和 AUTHID CURRENT_USER 的功能相同。

- **SECURITY DEFINER**

  AUTHID DEFINER

  声明该函数将以创建它的用户的权限执行。

  AUTHID DEFINER 和 SECURITY DEFINER 的功能相同。

- **COST execution_cost**

  用来估计函数的执行成本。

  execution_cost 以 cpu_operator_cost 为单位。

  取值范围：正数

- **ROWS result_rows**

  估计函数返回的行数。用于函数返回的是一个集合。

  取值范围：正数，默认值是 1000 行。

- **configuration_parameter**

  - **value**

  ➢把指定的数据库会话参数值设置为给定的值。如果 value 是 DEFAULT 或者 RESET，则在新的会话中使用系统的缺省设置。OFF 关闭设置。

  ➢取值范围：字符串

  - DEFAULT
  - OFF
  - RESET

  ➢指定默认值。

  - **from current**

  ➢取当前会话中的值设置为 configuration_parameter 的值。

- **plsql_body**

  PL/SQL 存储过程体。

  ⚠ **须知：**

  当在函数体中创建用户时，日志中会记录密码的明文。因此不建议用户在函数体中创建用户。

示例

```
--定义函数为 SQL 查询。
panweidb=# CREATE FUNCTION func_add_sql(integer, integer) RETURNS integer
  AS 'select $1 + $2;'
  LANGUAGE SQL
  IMMUTABLE
```

```
  RETURNS NULL ON NULL INPUT;

--利用参数名用 PL/pgSQL 自增一个整数。
panweidb=# CREATE OR REPLACE FUNCTION func_increment_plsql(i integer) RETUR
NS integer AS $$
    BEGIN
        RETURN i + 1;
    END;
$$ LANGUAGE plpgsql;

--返回 RECORD 类型
panweidb=# CREATE OR REPLACE FUNCTION func_increment_sql(i int, out result_1 b
igint, out result_2 bigint)
returns SETOF RECORD
as $$
begin
  result_1 = i + 1;
  result_2 = i * 10;
return next;
end;
$$language plpgsql;

--返回一个包含多个输出参数的记录。
panweidb=# CREATE FUNCTION func_dup_sql(in int, out f1 int, out f2 text)
  AS $$ SELECT $1, CAST($1 AS text) || ' is text' $$
  LANGUAGE SQL;

panweidb=# SELECT * FROM func_dup_sql(42);

--计算两个整数的和，并返回结果。如果输入为 null，则返回 null。
panweidb=# CREATE FUNCTION func_add_sql2(num1 integer, num2 integer) RETUR
N integer
AS
BEGIN
RETURN num1 + num2;
```

```
END;
/
--修改函数 func_add_sql2 的执行规则为 IMMUTABLE，即参数不变时返回相同结果。
panweidb=# ALTER FUNCTION func_add_sql2(INTEGER, INTEGER) IMMUTABLE;

--将函数 func_add_sql2 的名称修改为 add_two_number。
panweidb=# ALTER FUNCTION func_add_sql2(INTEGER, INTEGER) RENAME TO add_t
wo_number;

--将函数 add_two_number 的属者改为 omm。
panweidb=# ALTER FUNCTION add_two_number(INTEGER, INTEGER) OWNER TO om
m;

--删除函数。
panweidb=# DROP FUNCTION add_two_number;
panweidb=# DROP FUNCTION func_increment_sql;
panweidb=# DROP FUNCTION func_dup_sql;
panweidb=# DROP FUNCTION func_increment_plsql;
panweidb=# DROP FUNCTION func_add_sql;
```

相关链接

参考：SQL 语法参考->SQL 语法中的 ALTER FUNCTION、DROP FUNCTION 章节。

4.5.71.CREATE INCREMENTAL MATERIALIZED VIEW

功能描述

CREATE INCREMENTAL MATERIALIZED VIEW 会创建一个增量物化视图，并且后续可以使用 REFRESH MATERIALIZED VIEW（全量刷新）和 REFRESH INCREMENTAL MATERIALIZED VIEW（增量刷新）刷新物化视图的数据。

CREATE INCREMENTAL MATERIALIZED VIEW 类似于 CREATE TABLE AS，不过它会记住被用来初始化该视图的查询， 因此它可以在后续中进行数据刷新。一个物化视图有很多和表相同的属性，但是不支持临时物化视图。

注意事项

- 增量物化视图不可以在临时表或全局临时表上创建。
- 增量物化视图仅支持简单过滤查询和基表 UNION ALL 查询。
- 创建增量物化视图不可指定分布列。
- 创建增量物化视图后，基表中的绝大多数 DDL 操作不再支持。
- 不支持对增量物化视图进行 IUD 操作。
- 增量物化视图创建后，当基表数据发生变化时，需要使用刷新（REFRESH）命令保持物化视图与基表同步。

语法格式

```
CREATE INCREMENTAL MATERIALIZED VIEW mv_name
  [ (column_name [, ...] ) ]
  [ TABLESPACE tablespace_name ]
  AS query;
```

参数说明

- **mv_name**

  要创建的物化视图的名称（可以被模式限定）。

  取值范围：字符串，要符合标识符的命名规范。

- **column_name**

  新物化视图中的一个列名。物化视图支持指定列，指定列需要和后面的查询语句结果的列数量保持一致；如果没有提供列名，会从查询的输出列名中获取列名。

  取值范围：字符串，要符合标识符的命名规范。

- **TABLESPACE tablespace_name**

  指定新建物化视图所属表空间。如果没有声明，将使用默认表空间。

- **AS query**

  一个 SELECT 或者 TABLE 命令。这个查询将在一个安全受限的操作中运行。

示例

```
--创建一个普通表
panweidb=# CREATE TABLE my_table (c1 int, c2 int);
--创建增量物化视图
```

```
panweidb=# CREATE INCREMENTAL MATERIALIZED VIEW my_imv AS SELECT * FROM
my_table;
--基表写入数据
panweidb=# INSERT INTO my_table VALUES(1,1),(2,2);
--对增量物化视图 my_imv 进行增量刷新
panweidb=# REFRESH INCREMENTAL MATERIALIZED VIEW my_imv;
```

相关链接

参考：SQL 语法参考->SQL 语法中的 ALTER MATERIALIZED VIEW、CREATE MATERIALIZED VIEW、CREATE TABLE、DROP MATERIALIZED VIEW、REFRESH INCREMENTAL MATERIALIZED VIEW、REFRESH MATERIALIZED VIEW 章节。

4.5.72.CREATE INDEX

功能描述

在指定的表上创建索引。索引可以用来提高数据库查询性能，但是不恰当的使用将导致数据库性能下降。建议仅在匹配如下某条原则时创建索引：

- 经常执行查询的字段。
- 在连接条件上创建索引，对于存在多字段连接的查询，建议在这些字段上建立组合索引。例如，select * from t1 join t2 on t1.a=t2.a and t1.b=t2.b，可以在 t1 表上的 a、b 字段上建立组合索引。
- where 子句的过滤条件字段上（尤其是范围条件）。
- 经常出现在 order by、group by 和 distinct 后的字段。

在分区表上创建索引与在普通表上创建索引的语法不太一样，使用时请注意，如当索引带 GLOBAL/LOCAL 关键字或者创建索引为 GLOBAL 索引时不支持创建部分索引。

在 Oracle 兼容模式下，在创建全局分区索引时支持函数表达式，并在使用时命中索引。可参见 Oracle 兼容性手册中的 CREATE INDEX。

注意事项

- 索引自身也占用存储空间、消耗计算资源，创建过多的索引将对数据库性能造成负面影响（尤其影响数据导入的性能，建议在数据导入后再建索引）。因此，仅在必要时创建索引。
- 索引定义里的所有函数和操作符都必须是 immutable 类型的，即它们的结果必须只能依赖于它们的输入参数，而不受任何外部的影响（如另外一个表的内容或者当前时间）。这个限制可以确保该索引的行为是定义良好的。要在一个索引上或 WHERE 中使用用户定义函数，请把它标记为 immutable 类型函数。
- 分区表索引分为 LOCAL 索引与 GLOBAL 索引，LOCAL 索引与某个具体分区绑定，而 GLOBAL 索引则对应整个分区表。
- 列存表支持的 PSORT 和 B-tree 索引都不支持创建表达式索引、部分索引，PSORT 不支持创建唯一索引，B-tree 支持创建唯一索引。
- 列存表支持的 GIN 索引支持创建表达式索引，但表达式不能包含空分词、空列和多列，不支持创建部分索引和唯一索引。
- HASH 索引目前仅限于行存表索引、临时表索引和分区表 LOCAL 索引，且不支持创建多字段索引。
- 被授予 CREATE ANY INDEX 权限的用户，可以在 public 模式和用户模式下创建索引。
- 如果表达式索引中调用的是用户自定义函数，按照函数创建者权限执行表达式索引函数。
- 建议仅在匹配如下条件之一时创建索引：
  - 经常执行查询的字段。
  - 在连接条件上创建索引，对于存在多字段连接的查询，建议在这些字段上建立组合索引。例如，select * from t1 join t2 on t1.a=t2.a and t1.b=t2.b，可以在 t1 表上的 a、b 字段上建立组合索引。
  - where 子句的过滤条件字段上（尤其是范围条件）。
  - 在经常出现在 order by、group by 和 distinct 后的字段。
- 约束限制：
  - 分区表上不支持创建部分索引。
  - 分区表创建 GLOBAL 索引时，存在以下约束条件：
    - 不支持表达式索引、部分索引

- ✧ 不支持列存表
- ✧ 仅支持 B-tree 索引

➢ 在相同属性列上，分区 LOCAL 索引与 GLOBAL 索引不能共存。

➢ GLOBAL 索引，最大支持 31 列。

➢ 如果 alter 语句不带有 UPDATE GLOBAL INDEX，那么原有的 GLOBAL 索引将失效，查询时将使用其他索引进行查询；如果 alter 语句带有 UPDATE GLOBAL INDEX，原有的 GLOBAL 索引仍然有效，并且索引功能正确。

➢ 表字段大于等于 2074 字节不支持创建 BTree 索引，建议使用 HASH 或 GIN 索引

语法格式

❖ 在表上创建索引。

```
CREATE [ UNIQUE ] INDEX [ CONCURRENTLY ] [IF NOT EXISTS] [ [schema_name.]index
_name ] ON table_name [ USING method ]
 ({ { column_name | ( expression ) } [ COLLATE collation ] [ opclass ] [ ASC | DESC ] [ N
ULLS { FIRST | LAST } ] }[, ...] )
 [ INCLUDE ( column_name [, ...] )]
 [ WITH ( {storage_parameter = value} [, ... ] ) ]
 [ TABLESPACE tablespace_name ]
 [ WHERE predicate ];
```

❖ 在分区表上创建索引。

```
CREATE [ UNIQUE ] INDEX [ CONCURRENTLY ] [ [ IF NOT EXISTS ] [schema_name.] inde
x_name ] ON table_name [ USING method ]
  ( { { column_name | ( expression ) } [ COLLATE collation ] [ opclass ] [ ASC | DESC ] [
NULLS LAST ] } [, ...] )
  [ LOCAL [ ( { PARTITION index_partition_name | SUBPARTITION index_subpartition_
name [ TABLESPACE index_partition_tablespace ] } [, ...] ) ] | GLOBAL ]
  [ INCLUDE ( { column_name | ( expression ) }[, ...] ) ]
  [ WITH ( { storage_parameter = value } [, ...] ) ]
  [ TABLESPACE tablespace_name ];
```

参数说明

- UNIQUE：创建唯一性索引，每次添加数据时检测表中是否有重复值。如果插入或更新的值会引起重复的记录时，将导致一个错误。目前只有 B-tree 及 UBTree 索引支持唯一索引。
- CONCURRENTLY：以不阻塞 DML 的方式创建索引（加 ShareUpdateExclusiveLock 锁）。创建索引时，一般会阻塞其他语句对该索引所依赖表的访问。指定此关键字，可以实现创建过程中不阻塞 DML。
  - 此选项只能指定一个索引的名称。
  - 普通 CREATE INDEX 命令可以在事务内执行，但是 CREATE INDEX CONCURRENTLY 不可以在事务内执行。
  - 列存表和临时表不支持 CONCURRENTLY 方式创建索引。
  - 支持在行存一级分区表上在线创建全局索引和本地分区索引。

- 创建索引时指定此关键字，需要执行先后两次对该表的全表扫描来完成 build，第一次扫描的时候创建索引，不阻塞读写操作；第二次扫描的时候合并更新第一次扫描到目前为止发生的变更。
- 由于需要执行两次对表的扫描和 build，而且必须等待现有的所有可能对该表执行修改的事务结束。这意味着该索引的创建比正常耗时更长，同时因此带来的 CPU 和 I/O 消耗对其他业务也会造成影响。
- 如果在索引构建时发生失败，那会留下一个“不可用”的索引。这个索引会被查询忽略，但它仍消耗更新开销。这种情况推荐的恢复方法是删除该索引并尝试再次 CONCURRENTLY 建索引。
- 由于在第二次扫描之后，索引构建必须等待任何持有早于第二次扫描拿的快照的事务终止，而且建索引时加的 ShareUpdateExclusiveLock 锁（4 级）会和大于等于 4 级的锁冲突，在创建这类索引时，容易引发卡住（hang）或者死锁问题。例如：
  - 两个会话对同一个表创建 CONCURRENTLY 索引，会引起死锁问题；

- 两个会话，一个对表创建 CONCURRENTLY 索引，一个 drop table，会引起死锁问题；
- 三个会话，会话 1 先对表 a 加锁，不提交，会话 2 接着对表 b 创建 CONCURRENTLY 索引，会话 3 接着对表 a 执行写入操作，在会话 1 事务未提交之前，会话 2 会一直被阻塞；
- 将事务隔离级别设置成可重复读（默认为读已提交），起两个会话，会话 1 起事务对表 a 执行写入操作，不提交，会话 2 对表 b 创建 CONCURRENTLY 索引，在会话 1 事务未提交之前，会话 2 会一直被阻塞。

❖ IF NOT EXISTS：用于判断是否存在同名索引。

❖ schema_name：模式的名称。

取值范围：已存在模式名。

❖ index_name：要创建的索引名，索引的模式与表相同。

取值范围：字符串，要符合标识符的命名规范。

❖ table_name：需要为其创建索引的表的名称，可以用模式修饰。

取值范围：已存在的表名。

❖ USING method：指定创建索引的方法。

取值范围：

- btree：B-tree 索引使用一种类似于 B+树的结构来存储数据的键值，通过这种结构能够快速的查找索引。btree 适合支持比较查询以及查询范围。
- hash：Hash 索引使用 Hash 函数对索引的关键字进行散列。只能处理简单等值比较，比较适合在索引值较长的情况下使用。
- gin：GIN 索引是倒排索引，可以处理包含多个键的值（比如数组）。
- gist：Gist 索引适用于几何和地理等多维数据类型和集合数据类型。目前支持的数据类型有 box、point、poly、circle、tsvector、tsquery、range。
- Psort：Psort 索引是针对列存表进行局部排序的索引。

- ubtree：仅供 ustore 表使用的多版本 B-tree 索引，索引页面上包含事务信息，能并自主回收页面。

列存表对 GIN 索引支持仅限于对于 tsvector 类型的支持，即创建列存 GIN 索引入参需要为 to_tsvector 函数（的返回值)。此方法为 GIN 索引比较普遍的使用方式。

行存表（ASTORE 存储引擎）支持的索引类型：btree（行存表缺省值）、hash、gin、gist。行存表（USTORE 存储引擎）支持的索引类型：ubtree。列存表支持的索引类型：Psort（列存表缺省值）、btree、gin。全局临时表不支持 GIN 索引和 Gist 索引。

- column_name：表中需要创建索引的列的名称（字段名）。如果索引方式支持多字段索引，可以声明多个字段。全局索引最多可以声明 31 个字段，其他索引最多可以声明 32 个字段。
- expression：创建一个基于该表的一个或多个字段的表达式索引，通常必须写在圆括弧中。如果表达式有函数调用的形式，圆括弧可以省略。表达式索引可用于获取对基本数据的某种变形的快速访问。比如，一个在 upper(col)上的函数索引将允许 WHERE upper(col) = 'JIM'子句使用索引。在创建表达式索引时，如果表达式中包含 IS NULL 子句，则这种索引是无效的。此时，建议用户尝试创建一个部分索引。

  在 Oracle 兼容模式下，在创建全局分区索引时支持函数表达式，并在使用时命中索引。可参见 Oracle 兼容性手册中的 CREATE INDEX。
- COLLATE collation：COLLATE 子句指定列的排序规则（该列必须是可排列的数据类型）。如果没有指定，则使用默认的排序规则。排序规则可以使用“select * from pg_collation”命令从 pg_collation 系统表中查询，默认的排序规则为查询结果中以 default 开始的行。
- opclass：操作符类的名称。对于索引的每一列可以指定一个操作符类，操作符类标识了索引那一列的使用的操作符。例如一个 B-tree 索引在一个四字节整数上可以使用 int4_ops；这个操作符类包括四字节整数的比较函数。实际上对于列上的数据类型默认的操作符类是足够用的。操作符类主要用于一些有多种排序的数据。例如，用户想按照绝对值或者

实数部分排序一个复数。能通过定义两个操作符类然后当建立索引时选择合适的类。

- ASC：指定按升序排序（默认）。
- DESC：指定按降序排序。
- NULLS FIRST：指定空值在排序中排在非空值之前，当指定 DESC 排序时，本选项为默认的。
- NULLS LAST：指定空值在排序中排在非空值之后，未指定 DESC 排序时，本选项为默认的。
- LOCAL：指定创建的分区索引为 LOCAL 索引。
- GLOBAL：指定创建的分区索引为 GLOBAL 索引，当不指定 LOCAL、GLOBAL 关键字时，默认创建 GLOBAL 索引。
- INCLUDE (column_name [, ...])：可选的 INCLUDE 子句指定将一些非键列（non-key columns）包含在索引中。
  - 非键列不能用于作为索引扫描的加速搜索条件，同时在检查索引的唯一性约束时会忽略它们。
  - 仅索引扫描 (Index Only Scan) 可以直接返回非键列中的内容，而不必去访问索引所对应的堆表。
  - 将非键列添加为 INCLUDE 列需要保守一些，尤其是对于宽列。如果索引元组超过索引类型允许的最大大小，数据将插入失败。需要注意的是，任何情况下为索引添加非键列都会增加索引的空间占用，从而可能减慢搜索速度。
  - 目前只有 ubtree 索引访问方式支持该特性。非键列会被保存在与堆元组对应的索引叶子元组中，不会包含在索引上层页面的元组中。
- WITH ({storage_parameter = value} [, ...])：指定索引方法的存储参数。取值范围：
  - FILLFACTOR：一个索引的填充因子（fillfactor）是一个介于 10 和 100 之间的百分数。
  - 取值范围：10~100
  - FASTUPDATE：GIN 索引是否使用快速更新。
  - 取值范围：ON，OFF

- 默认值：ON
- GIN_PENDING_LIST_LIMIT：当 GIN 索引启用 fastupdate 时，设置该索引 pending list 容量的最大值。
- 取值范围：64~INT_MAX，单位 KB。
- 默认值：gin_pending_list_limit 的默认取决于 GUC 中 gin_pending_list_limit 的值（默认为 4MB）
- INDEXSPLIT：UBTREE 索引选择采取哪种分裂策略。其中 DEFAULT 策略指的是与 BTREE 相同的分裂策略。INSERTPT 策略能在某些场景下显著降低索引空间占用。
- 取值范围：INSERTPT，DEAFAULT
- 默认值：INSERTPT

只有 GIN 索引支持 FASTUPDATE、GIN_PENDING_LIST_LIMIT 参数。GIN 和 Psort 之外的索引都支持 FILLFACTOR 参数。只有 UBTREE 索引支持 INDEXSPLIT 参数。

❖ TABLESPACE tablespace_name：指定索引的表空间，如果没有声明则使用默认的表空间。

取值范围：已存在的表空间名。

❖ WHERE predicate：

取值范围：predicate 表达式只能引用表的字段，它可以使用所有字段，而不仅是被索引的字段。目前，子查询和聚集表达式不能出现在 WHERE 子句里。不建议使用 int 等数值类型作为 predicate，因为 int 等数值类型可以隐式转换为 bool 值（非 0 值隐式转换为 true，0 转换为 false），可能导致非预期的结果。

对于分区表索引，当创建索引带 GLOBAL/LOCAL 关键字，或者最终创建的索引类型为 GLOBAL 索引时，不支持带 WHERE 子句创建索引。

PARTITION index_partition_name：索引分区的名称。

取值范围：字符串，要符合标识符的命名规范。

❖ SUBPARTITION index_subpartition_name：索引二级分区的名称。

取值范围：字符串，要符合标识符的命名规范

❖ TABLESPACE index_partition_tablespace：索引分区的表空间。

取值范围：如果没有声明，将使用分区表索引的表空间 index_tablespace。

- COMPRESSTYPE：索引参数，设置索引压缩算法。1 代表 pglz 算法，2 代表 zstd 算法，默认不压缩。（仅支持 B-TREE 索引）

取值范围：0~2，默认值为 0

- COMPRESS_LEVEL：索引参数，设置索引压缩算法等级，仅当 COMPRESSTYPE 为 2 时生效。压缩等级越高，索引的压缩效果越好，索引的访问速度越慢。（仅支持 B-TREE 索引）

取值范围：-31~31，默认值为 0

- COMPRESS_CHUNK_SIZE：索引参数，设置索引压缩 chunk 块大小。chunk 数据块越聚集，预期能达到的压缩效果越好；chunk 数据块越离散，索引的访问速度越受影响。（仅支持 B-TREE 索引）

取值范围：与页面大小有关。在页面大小为 8k 场景，取值范围为：512、1024、2048、4096。

默认值：4096

- COMPRESS_PREALLOC_CHUNKS：索引参数，设置索引压缩 chunk 块预分配数量。预分配数量越大，索引的压缩率相对越差，离散度越小，访问性能越好。（仅支持 B-TREE 索引）

取值范围：0~7，默认值为 0

  - 当 COMPRESS_CHUNK_SIZE 为 512 和 1024 时，支持预分配设置最大为 7。
  - 当 COMPRESS_CHUNK_SIZE 为 2048 时，支持预分配设置最大为 3。
  - 当 COMPRESS_CHUNK_SIZE 为 4096 时，支持预分配设置最大为 1。

- COMPRESS_BYTE_CONVERT：

索引参数，设置索引压缩字节转换预处理。在一些场景下可以提升压缩效果，同时会导致一定性能劣化。该参数允许修改，修改后决定变更数据、新增数据是否进行字节转换预处理。当 COMPRESS_DIFF_CONVERT 为真时，该值不允许修改为假。

取值范围：布尔值，默认关闭。

- COMPRESS_DIFF_CONVERT：索引参数，设置索引压缩字节差分预处理。只能与 COMPRESS_BYTE_CONVERT 一起使用。在一些场景下可以提升压缩效果，同时会导致一定性能劣化。该参数允许修改，修改后决定变更数据、新增数据是否进行字节差分预处理。

取值范围：布尔值，默认关闭。

示例

- 创建表 tpcds.ship_mode_t1。

```
CREATE schema tpcds;
CREATE TABLE tpcds.ship_mode_t1
 (
   SM_SHIP_MODE_SK        INTEGER           NOT NULL,
   SM_SHIP_MODE_ID        CHAR(16)          NOT NULL,
   SM_TYPE              CHAR(30)              ,
   SM_CODE               CHAR(10)             ,
   SM_CARRIER             CHAR(20)              ,
   SM_CONTRACT            CHAR(20)
 )
 ;
```

- 在表 tpcds.ship_mode_t1 上的 SM_SHIP_MODE_SK 字段上创建普通的唯一索引。

```
CREATE UNIQUE INDEX ds_ship_mode_t1_index1 ON tpcds.ship_mode_t1(SM_SHIP_MODE_SK);
```

- 在表 tpcds.ship_mode_t1 上的 SM_SHIP_MODE_SK 字段上创建指定 B-tree 索引。

```
CREATE INDEX ds_ship_mode_t1_index4 ON tpcds.ship_mode_t1 USING btree(SM_SHIP_MODE_SK);
```

- 在表 tpcds.ship_mode_t1 上 SM_CODE 字段上创建表达式索引。

```
CREATE INDEX ds_ship_mode_t1_index2 ON tpcds.ship_mode_t1(SUBSTR(SM_CODE, 1 ,4));
```

- 在表 tpcds.ship_mode_t1 上的 SM_SHIP_MODE_SK 字段上创建 SM_SHIP_MODE_SK 大于 10 的部分索引。

```
CREATE UNIQUE INDEX ds_ship_mode_t1_index3 ON tpcds.ship_mode_t1(SM_SHIP_MODE_SK) WHERE SM_SHIP_MODE_SK>10;
```

❖ 重命名一个现有的索引。

```
ALTER INDEX tpcds.ds_ship_mode_t1_index1 RENAME TO ds_ship_mode_t1_index5;
```

❖ 设置索引不可用。

```
ALTER INDEX tpcds.ds_ship_mode_t1_index2 UNUSABLE;
```

❖ 重建索引。

```
ALTER INDEX tpcds.ds_ship_mode_t1_index2 REBUILD;
```

❖ 删除一个现有的索引。

```
DROP INDEX tpcds.ds_ship_mode_t1_index2;
```

❖ 删除表。

```
DROP TABLE tpcds.ship_mode_t1;
```

❖ 创建表空间。

```
CREATE TABLESPACE example1 RELATIVE LOCATION 'tablespace1/tablespace_1';
CREATE TABLESPACE example2 RELATIVE LOCATION 'tablespace2/tablespace_2';
CREATE TABLESPACE example3 RELATIVE LOCATION 'tablespace3/tablespace_3';
CREATE TABLESPACE example4 RELATIVE LOCATION 'tablespace4/tablespace_4';
```

❖ 创建表 tpcds.customer_address_p1。

```
CREATE TABLE tpcds.customer_address_p1
(
    CA_ADDRESS_SK         INTEGER           NOT NULL,
    CA_ADDRESS_ID         CHAR(16)          NOT NULL,
    CA_STREET_NUMBER      CHAR(10)                  ,
    CA_STREET_NAME        VARCHAR(60)               ,
    CA_STREET_TYPE        CHAR(15)                  ,
    CA_SUITE_NUMBER       CHAR(10)                  ,
    CA_CITY               VARCHAR(60)               ,
    CA_COUNTY             VARCHAR(30)               ,
    CA_STATE              CHAR(2)                   ,
    CA_ZIP                CHAR(10)                  ,
    CA_COUNTRY            VARCHAR(20)               ,
    CA_GMT_OFFSET         DECIMAL(5,2)              ,
    CA_LOCATION_TYPE      CHAR(20)
)
```

```
TABLESPACE example1
PARTITION BY RANGE(CA_ADDRESS_SK)
(
  PARTITION p1 VALUES LESS THAN (3000),
  PARTITION p2 VALUES LESS THAN (5000) TABLESPACE example1,
  PARTITION p3 VALUES LESS THAN (MAXVALUE) TABLESPACE example2
)
ENABLE ROW MOVEMENT;
```

❖ 在线在 CA_STREET_NAME 字段创建全局索引。

```
CREATE INDEX CONCURRENTLY idx_global_name ON tpcds.customer_address_p1(CA_STREET_NAME);
```

❖ 在线在 CA_CITY 字段创建一级分区索引。

```
CREATE INDEX CONCURRENTLY idx_local_city ON tpcds.customer_address_p1(CA_CITY);
```

❖ 创建分区表索引 ds_customer_address_p1_index1，不指定索引分区的名称。

```
CREATE INDEX ds_customer_address_p1_index1 ON tpcds.customer_address_p1(CA_ADDRESS_SK) LOCAL;
```

❖ 创建分区表索引 ds_customer_address_p1_index2，并指定索引分区的名称。

```
CREATE INDEX ds_customer_address_p1_index2 ON tpcds.customer_address_p1(CA_ADDRESS_SK) LOCAL
  (
    PARTITION CA_ADDRESS_SK_index1,
    PARTITION CA_ADDRESS_SK_index2 TABLESPACE example3,
    PARTITION CA_ADDRESS_SK_index3 TABLESPACE example4
  )
 TABLESPACE example2;
```

❖ 创建 GLOBAL 分区索引。

```
CREATE INDEX ds_customer_address_p1_index3 ON tpcds.customer_address_p1(CA_ADDRESS_ID) GLOBAL;
```

❖ 不指定关键字，默认创建 GLOBAL 分区索引。

```
CREATE INDEX ds_customer_address_p1_index4 ON tpcds.customer_address_p1(CA_ADDRESS_ID);
```

❖ 修改分区表索引 CA_ADDRESS_SK_index2 的表空间为 example1。

```
ALTER INDEX tpcds.ds_customer_address_p1_index2 MOVE PARTITION CA_ADDRESS
_SK_index2 TABLESPACE example1;
```

❖ 修改分区表索引 CA_ADDRESS_SK_index3 的表空间为 example2。

```
ALTER INDEX tpcds.ds_customer_address_p1_index2 MOVE PARTITION CA_ADDRESS
_SK_index3 TABLESPACE example2;
```

❖ 重命名分区表索引。

```
ALTER INDEX tpcds.ds_customer_address_p1_index2 RENAME PARTITION CA_ADDR
ESS_SK_index1 TO CA_ADDRESS_SK_index4;
```

❖ 删除索引和分区表。

```
DROP INDEX tpcds.ds_customer_address_p1_index1;
DROP INDEX tpcds.ds_customer_address_p1_index2;
DROP TABLE tpcds.customer_address_p1;
```

❖ 删除表空间。

```
DROP TABLESPACE example1;
DROP TABLESPACE example2;
DROP TABLESPACE example3;
DROP TABLESPACE example4;
```

4.5.73.CREATE MASKING POLICY

功能描述

创建脱敏策略。

注意事项

只有 poladmin、sysadmin 或初始用户能执行此操作。

需要开启安全策略开关，即设置 GUC 参数 enable_security_policy=on，脱敏策略才可以生效。

语法格式

```
CREATE MASKING POLICY policy_name masking_clause[, ...]* policy_filter [ENABLE |
DISABLE];
```

❖ masking_clause：

```
masking_function ON LABEL(label_name[, ...]*)
```

❖ masking_function：

maskall 不是预置函数，硬编码在代码中，不支持 df 展示。

预置时脱敏方式如下：

```
maskall | randommasking | creditcardmasking | basicemailmasking | fullemailmaski
ng | shufflemasking | alldigitsmasking | regexpmasking
```

❖ policy_filter:

```
FILTER ON FILTER_TYPE(filter_value [,...]*)[,...]*
```

❖ FILTER_TYPE:

```
IP | APP | ROLES
```

参数说明

❖ **policy_name**

审计策略名称，需要唯一，不可重复。

取值范围：字符串，要符合标识符的命名规范。

❖ **label_name**

资源标签名称。

❖ **masking_clause**

指出使用何种脱敏函数对被 label_name 标签标记的数据库资源进行脱敏，支持用 schema.function 的方式指定脱敏函数。

❖ **policy_filter**

指出该脱敏策略对何种身份的用户生效，若为空表示对所用用户生效。

❖ **FILTER_TYPE**

描述策略过滤的条件类型，包括 IP | APP | ROLES。

❖ **filter_value**

指具体过滤信息内容，例如具体的 IP、具体的 APP 名称、具体的用户名。

❖ **ENABLE|DISABLE**

可以打开或关闭脱敏策略。若不指定 ENABLE|DISABLE，语句默认为 ENABLE。

示例

```
--创建 dev_mask 和 bob_mask 用户。
```

```
panweidb=# CREATE USER dev_mask PASSWORD 'dev@1234';
panweidb=# CREATE USER bob_mask PASSWORD 'bob@1234';

--创建一个表 tb_for_masking
panweidb=# CREATE TABLE tb_for_masking(col1 text, col2 text, col3 text);

--创建资源标签标记敏感列 col1
panweidb=# CREATE RESOURCE LABEL mask_lb1 ADD COLUMN(tb_for_masking.col
1);

--创建资源标签标记敏感列 col2
panweidb=# CREATE RESOURCE LABEL mask_lb2 ADD COLUMN(tb_for_masking.col
2);

--对访问敏感列 col1 的操作创建脱敏策略
panweidb=# CREATE MASKING POLICY maskpol1 maskall ON LABEL(mask_lb1);

--创建仅对用户 dev_mask 和 bob_mask,客户端工具为 psql 和 gsql，IP 地址为'10.2
0.30.40', '127.0.0.0/24'场景下生效的脱敏策略。
panweidb=# CREATE MASKING POLICY maskpol2 randommasking ON LABEL(mask_l
b2) FILTER ON ROLES(dev_mask, bob_mask), APP(psql, gsql), IP('10.20.30.40', '127.
0.0.0/24');
```

相关链接

参考 SQL 语法参考->SQL 语法->ALTER MASKING POLICY、DROP MASKING POLICY 章节。

4.5.74.CREATE MATERIALIZED VIEW

CREATE MATERIALIZED VIEW 会创建一个全量物化视图，并且后续可以使用 REFRESH MATERIALIZED VIEW（全量刷新）刷新物化视图的数据。

CREATE MATERIALIZED VIEW 类似于 CREATE TABLE AS，不过它会记住被用来初始化该视图的查询， 因此它可以在后续中进行数据刷新。一个物化视图有很多和表相同的属性，但是不支持临时物化视图。

注意事项

- 全量物化视图不可以在临时表或全局临时表上创建。
- 全量物化视图不支持 nodegroup。
- 创建全量物化视图后，基表中的绝大多数 DDL 操作不再支持。
- 不支持对全量物化视图进行 IUD 操作。
- 全量物化视图创建后，当基表数据发生变化时，需要使用刷新（REFRESH）命令保持物化视图与基表同步。
- Ustore 引擎不支持物化创建、使用视图。

语法格式

```
CREATE MATERIALIZED VIEW mv_name
  [ (column_name [, ...] ) ]
  [ WITH ( {storage_parameter = value} [, ... ] ) ]
   [ TABLESPACE tablespace_name ]
  AS query
  [ WITH [ NO ] DATA ];
```

参数说明

- **mv_name**

  要创建的物化视图的名称（可以被模式限定）。

  取值范围：字符串，要符合标识符的命名规范。

- **column_name**

  新物化视图中的一个列名。物化视图支持指定列，指定列需要和后面的查询语句结果的列数量保持一致；如果没有提供列名，会从查询的输出列名中获取列名。

  取值范围：字符串，要符合标识符的命名规范。

- **WITH (storage_parameter [= value] [, ...])**

  这个子句为表或索引指定一个可选的存储参数。详见：SQL 语法参考->SQL 语法->CREATE TABLE 章节。

- **TABLESPACE tablespace_name**

  指定新建物化视图所属表空间。如果没有声明，将使用默认表空间。

- **AS query**

一个 SELECT、TABLE 或者 VALUES 命令。这个查询将在一个安全受限的操作中运行。

- **[WITH [NO] DATA]**

  创建表时，是否也插入查询到的数据。默认是要数据，选择“NO”参数时，则不要数据。

示例

```
--创建一个普通表
panweidb=# CREATE TABLE my_table (c1 int, c2 int);
--创建全量物化视图
panweidb=# CREATE MATERIALIZED VIEW my_mv AS SELECT * FROM my_table;
--基表写入数据
panweidb=# INSERT INTO my_table VALUES(1,1),(2,2);
--对全量物化视图 my_mv 进行全量刷新
panweidb=# REFRESH MATERIALIZED VIEW my_mv;
```

相关链接

SQL 语法参考->SQL 语法->ALTER MATERIALIZED VIEW、CREATE INCREMENTAL MATERIALIZED VIEW、CREATE TABLE、DROP MATERIALIZED VIEW、REFRESH INCREMENTAL MATERIALIZED VIEW、REFRESH MATERIALIZED VIEW 章节。

4.5.75.CREATE MODEL

功能描述

训练机器学习模型并保存模型。

注意事项

- 模型名称具有唯一性约束，注意命名格式。
- AI 训练时长波动较大，在部分情况下训练运行时间较长，设置的 GUC 参数 statement_timeout 时长过短会导致训练中断。建议 statement_timeout 设置为 0，不对语句执行时长进行限制。

语法格式

```
CREATE MODEL model_name USING algorithm_name
[FEATURES { {expression [ [ AS ] output_name ]} [, ...] }]
[TARGET { {expression [ [ AS ] output_name ]} [, ...] }]
FROM { table_name | select_query }
WITH hyperparameter_name = { hyperparameter_value | DEFAULT } [, ...] }
```

参数说明

- **model_name**

  对训练模型进行命名，模型名称具有唯一性约束。

  取值范围：字符串，需要符合标识符的命名规范。

- **architecture_name**

  训练模型的算法类型。

  取值范围：字符型，当前支持：logistic_regression、linear_regression、svm_classification、kmeans。

- **attribute_list**

  枚举训练模型的输入列名。

  取值范围：字符型，需要符合数据属性名的命名规范。

- **attribute_name**

  在监督学习任务重训练模型的目标列名(可进行简单的表达式处理)。

  取值范围：字符型，需要符合数据属性名的命名规范。

- **subquery**

  数据源。

  取值范围：字符串，符合数据库 SQL 语法。

- **hyper_parameter_name**

  机器学习模型的超参名称。

  取值范围：字符串，针对不同算法超参类型范围不同，取值范围详情请参考：AI 特性->DB4AI：数据库驱动 AI->原生 DB4AI 引擎章节中的表 2。

- **hp_value**

  超参数值。

取值范围：字符串，针对不同算法范围不同，取值范围详情请参考：：AI 特性->DB4AI：数据库驱动 AI->原生 DB4AI 引擎章节中的表 3。

示例

1、创建测试表 houses，并插入测试数据。

```
CREATE TABLE houses(id int,size int,lot varchar,price int);
INSERT INTO houses values('1','2','3','4');
INSERT INTO houses values('5','6','7','8');
```

2、创建模型。

```
CREATE MODEL price_model USING logistic_regression
 FEATURES size
 TARGET price
 FROM houses
 WITH learning_rate=0.88, max_iterations=default;
```

相关链接

SQL 语法参考->SQL 语法->DROP MODEL、PREDICT BY 章节。

4.5.76.CREATE OPERATOR

功能描述

定义一个新操作符。

注意事项

CREATE OPERATOR 定义一个新的 name 操作符。 定义该操作符的用户将成为其所有者。如果给出了一个模式名， 那么该操作符将在指定的模式中创建。否则它会在当前模式中创建。

操作符 name 是一个由下列字符组成的字符串：

+ - * / < > = ~ ! @ # % ^ & | ` ?

选择名字的时候有几个限制：

- --和/*不能在操作符名的任何地方出现， 因为它们会被认为是一个注释的开始。
- 一个多字符的操作符不能以+或-结尾， 除非该名字还包含至少下面字符之一：

  ~ ! @ # % ^ & | ` ?
- => 作为一个操作符名的使用已经废弃了。

操作符!=在输入时映射成<>， 因此这两个名称总是等价的。

至少需要定义一个 LEFTARG 和 RIGHTARG。对于双目操作符来说， 两者都需要定义。对右目操作符来说，只需要定义 LEFTARG， 而对于左目操作符来说，只需要定义 RIGHTARG。

同样，function_name 过程必须已经用 CREATE FUNCTION 定义过， 而且必须定义为接受正确数量的指定类型参数(一个或是两个)。

其他子句声明可选的操作符优化子句。他们的含义在 PostgreSQL 9.3.1 中文手册第 35.13 节里定义。详见：http://postgres.cn/docs/9.3/xoper-optimization.html。

要想能够创建一个操作符，你必须在参数类型和返回类型上有 USAGE 权限，还要在底层函数上有 EXECUTE 权限。如果指定了交换或者负操作符， 你必须拥有这些操作符。

语法格式

```
CREATE OPERATOR name (
    PROCEDURE = function_name
    [, LEFTARG = left_type ] [, RIGHTARG = right_type ]
    [, COMMUTATOR = com_op ] [, NEGATOR = neg_op ]
    [, RESTRICT = res_proc ] [, JOIN = join_proc ]
    [, HASHES ] [, MERGES ]
)
```

参数说明

- **name**

要定义的操作符。可用的字符见上文。其名字可以用模式修饰， 比如 CREATE OPERATOR myschema.+ (...)。如果没有模式， 则在当前模式中创建操作符。同一个模式中的两个操作符可以有一样的名字， 只要他们操作不同的数据类型。这是一个重载过程。

- **function_name**

  用于实现该操作符的函数。

- **left_type**

  操作符左边的参数数据类型，如果存在的话。如果是左目操作符，这个参数可以省略。

- **right_type**

  操作符右边的参数数据类型，如果存在的话。如果是右目操作符，这个参数可以省略。

- **com_op**

  该操作符对应的交换操作符。

- **neg_op**

  该操作符对应的负操作符。

- **res_proc**

  此操作符约束选择性评估函数。

- **join_proc**

  此操作符连接选择性评估函数。

- **HASHES**

  表明此操作符支持 Hash 连接。

- **MERGES**

  表明此操作符可以支持一个融合连接。

  使用 OPERATOR()语法在 com_op 或者其他可选参数里给出一个模式修饰的操作符名，比如：

```
COMMUTATOR = OPERATOR(myschema.===) ,
```

示例

下面命令定义一个新操作符：面积相等，用于 box 数据类型。

```
CREATE OPERATOR === (
```

```
  LEFTARG = box,
  RIGHTARG = box,
  PROCEDURE = area_equal_procedure,
  COMMUTATOR = ===,
  NEGATOR = !==,
  RESTRICT = area_restriction_procedure,
  JOIN = area_join_procedure,
  HASHES, MERGES
);
```

4.5.77.CREATE PACKAGE

功能描述

创建一个新的 PACKAGE。

注意事项

- package 只支持集中式，无法在分布式中使用。
- 在 package specification 中声明过的函数或者存储过程，必须在 package body 中找到定义。
- 在实例化中，无法调用带有 commit/rollback 的存储过程。
- 不能在 Trigger 中调用 package 函数。
- 不能在外部 SQL 中直接使用 package 当中的变量。
- 不允许在 package 外部调用 package 的私有变量和存储过程。
- 不支持其他存储过程不支持的用法，例如，在 function 中不允许调用 commit/rollback，则 package 的 function 中同样无法调用 commit/rollback。
- 不支持 schema 与 package 同名。
- 只支持 A 风格的存储过程和函数定义。
- 不支持 package 内有同名变量，包括包内同名参数。
- package 的全局变量为 session 级，不同 session 之间 package 的变量不共享。

- package 中调用自治事务的函数，不允许使用 package 中的 cursor 变量，以及递归的使用 package 中 cursor 变量的函数。
- package 中不支持声明 ref cursor 变量。
- package 默认为 SECURITY INVOKER 权限，如果想将默认行为改为 SECURITY DEFINER 权限，需要设置 GUC 参数 behavior_compat_options='plsql_security_definer'。
- 被授予 CREATE ANY PACKAGE 权限的用户，可以在 public 模式和用户模式下创建 PACKAGE。
- 如果需要创建带有特殊字符的 package 名，特殊字符中不能含有空格，并且最好设置 GUC 参数 behavior_compat_options='skip_insert_gs_source',否则可能引起报错。

语法格式

- CREATE PACKAGE SPECIFICATION 语法格式。

```
CREATE [ OR REPLACE ] PACKAGE [ schema. ]package_name
    [ invoker_rights_clause ] { IS | AS } item_list_1 END ;
```

invoker_rights_clause 可以被声明为 AUTHID DEFINER 或者 AUTHID CURRENT_USER，分别为定义者权限和调用者权限。

item_list_1 可以为声明的变量或者存储过程以及函数。

PACKAGE SPECIFICATION（包规格）声明了包内的公有变量、函数、异常等，可以被外部函数或者存储过程调用。在 PACKAGE SPECIFICATION 中只能声明存储过程、函数，不能定义存储过程或者函数。

- CREATE PACKAGE BODY 语法格式。

```
CREATE [ OR REPLACE ] PACKAGE BODY [ schema. ]package_name
    { IS | AS } declare_section [ initialize_section ] END package_name;
```

PACKAGE BODY（包体内）定义了包的私有变量、函数等。如果变量或者函数没有在 PACKAGE SPECIFICATION 中声明过，那么这个变量或者函数则为私有变量或者函数。

PACKAGE BODY 也可以声明实例化部分，用来初始化 package，详见示例。

示例

❖ CREATE PACKAGE SPECIFICATION 示例

```
CREATE OR REPLACE PACKAGE emp_bonus IS
var1 int:=1;--公有变量
var2 int:=2;
PROCEDURE testpro1(var3 int);--公有存储过程，可以被外部调用
END ;
/
```

❖ CREATE PACKAGE BODY 示例

```
drop table if exists test1;
create or replace package body emp_bonus is
var3 int:=3;
var4 int:=4;
procedure testpro1(var3 int)
is
begin
create table if not exists test1(col1 int);
insert into test1 values(var1);
insert into test1 values(var4);
end;
begin  --实例化开始
var4:=9;
testpro1(var4);
end;
end;
/
```

❖ 调用 PACKAGE 示例

```
call emp_bonus.testpro1(1); --使用 call 调用 package 存储过程
select emp_bonus.testpro1(1); --使用 select 调用 package 存储过程
--匿名块里调用 package 存储过程
begin
emp_bonus.testpro1(1);
end;
/
```

4.5.78.CREATE PROCEDURE

功能描述

创建一个新的存储过程。

注意事项

- 如果创建存储过程时参数或返回值带有精度，不进行精度检测。
- 创建存储过程时，存储过程定义中对表对象的操作建议都显示指定模式，否则可能会导致存储过程执行异常。
- 在创建存储过程时，存储过程内部通过 SET 语句设置 current_schema 和 search_path 无效。执行完函数 search_path 和 current_schema 与执行函数前的 search_path 和 current_schema 保持一致。
- SELECT、CALL 调用函数时，必须要在出参位置提供实参进行调用，实参不会发生作用。
- 存储过程指定 package 属性时支持重载。
- 不能创建仅形参名字不同(存储过程名和参数列表类型都一样)的重载存储过程。
- 重载的函数在调用时变量需要明确具体的类型。
- 不能创建与函数拥有相同名称和参数列表的存储过程。
- 在存储过程内部使用未声明的变里，存储过程被调用时会报错。
- 在创建 procedure 时，不能在 avg 函数外面嵌套其他 agg 函数，或者其他系统函数。
- 在存储过程内部调用其他无参数的存储过程时，可以省略括号，直接使用存储过程名进行调用。
- 在存储过程内部调用其他有出参的函数，如果在赋值表达式中调用时，被调函数的出参可以省略，给出了也会被忽略。
- 存储过程支持参数注释的查看与导出、导入。
- 存储过程支持介于 IS/AS 与 plsql_body 之间的注释的查看与导出、导入。

- 存储过程默认为 SECURITY INVOKER 权限，如果想将默认行为改为 SECURITY DEFINER 权限，需要设置 GUC 参数 behavior_compat_options='plsql_security_definer'。
- 被授予 CREATE ANY FUNCTION 权限的用户，可以在用户模式下创建/替换存储过程。
- out/inout 参数必须传入变量，不能够传入常量。
- 集中式环境下，想要调用 in 参数相同，out 参数不同的存储过程，需要设置 GUC 参数 behavior_compat_options='proc_outparam_override'，并且打开参数后，无论使用 select 还是 call 调用存储过程，都必须加上 out 参数。打开参数后，不支持使用 perform 调用存储过程或函数。
- 不支持在函数或存储过程中执行 analyze 操作。
- 不可与同一模式下已存在的 synonym 产生命名冲突。
- 通过 CREATE OR REPLACE 语法替换已有的存储过程时，会一并重建依赖此存储过程的视图，存储过程中的参数数据类型变更等情况可能会导致重建视图失败，进而导致替换存储过程失败。此种情况下，建议先删除依赖的视图，再重建存储过程，再重新创建视图。

语法格式

```
CREATE [ OR REPLACE ] PROCEDURE procedure_name
  [ ( {[ argname ] [ argmode ] argtype [ { DEFAULT | := | = } expression ]}[,...]) ]
  [
    { IMMUTABLE | STABLE | VOLATILE }
    | { SHIPPABLE | NOT SHIPPABLE }
    | {PACKAGE}
    | [ NOT ] LEAKPROOF
    | { CALLED ON NULL INPUT | RETURNS NULL ON NULL INPUT | STRICT }
    | {[ EXTERNAL ] SECURITY INVOKER | [ EXTERNAL ] SECURITY DEFINER | AUTHID DEF
INER | AUTHID CURRENT_USER}
    | COST execution_cost
    | SET configuration_parameter { TO value | = value | FROM CURRENT }
  ][ ... ]
```

```
{ IS | AS }
plsql_body
/
```

参数说明

- **OR REPLACE**

  当存在同名的存储过程时，替换原来的定义。

- **procedure_name**

  创建的存储过程名称，可以带有模式名。

  取值范围：字符串，要符合标识符的命名规范。

- **argmode**

  参数的模式。

  ⚠ **须知：**

  VARIADIC 用于声明数组类型的参数。

  取值范围： IN、OUT、INOUT 或 VARIADIC。缺省值是 IN。只有 OUT 模式的参数能跟在 VARIADIC 参数之后。

- **argname**

  参数的名称。

  取值范围：字符串，要符合标识符的命名规范。

- **argtype**

  参数的数据类型。可以使用%TYPE 或%ROWTYPE 间接引用变量或表的类型，详细可参考：PL/SQL->存储过程->基本语句->定义变量章节。

  取值范围：可用的数据类型。

- **configuration_parameter**

  - **value**

  - 把指定的配置参数设置为给定的值。如果 value 是 DEFAULT，则在新的会话中使用系统的缺省设置。OFF 关闭设置。

  - 取值范围：字符串

    - DEFAULT
    - OFF
    - 指定默认值。

- **from current**

  取当前会话中的值设置为 configuration_parameter 的值。

- **IMMUTABLE、STABLE 等**

  行为约束可选项。各参数的功能与 CREATE FUNCTION 类似，详细说明见：SQL 语法参考->SQL 语法->CREATE FUNCTION 章节。

- **plsql_body**

  PL/SQL 存储过程体。

  须知：

  当在存储过程体中进行创建用户等涉及用户密码相关操作时，系统表及 csv 日志中会记录密码的明文。因此不建议用户在存储过程体中进行涉及用户密码的相关操作。

  说明：

  argname 和 argmode 的顺序没有严格要求，推荐按照 argname、argmode、argtype 的顺序使用。

相关链接

参考：SQL 语法参考->SQL 语法->DROP PROCEDURE 章节。

4.5.79.CREATE PUBLICATION

功能描述

向当前数据库添加一个新的发布，发布的名称必须与当前数据库中任何现有发布的名称不同。发布本质上是通过逻辑复制将一组表的数据变更进行复制。

注意事项

- 如果既没有指定 FOR TABLE，也没有指定 FOR ALL TABLES， 那么这个发布就是以一组空表开始的，可以在后续添加表。
- 创建发布不会开始复制。它只为未来的订阅者定义一个分组和过滤逻辑。要创建一个发布，调用者必须拥有当前数据库的 CREATE 权限。（当然，系统管理员不需要这个检查。）
- 要将表添加到发布中，调用者必须拥有该表的所有权。FOR ALL TABLES

子句要求调用者是具有 SYSADMIN 权限用户。

- 添加到发布 UPDATE 或 DELETE 操作的发布的表必须已经定义了 REPLICA IDENTITY，否则将在这些表上禁止这些操作。
- COPY ... FROM 命令是作为 INSERT 操作发布的。不发布 TRUNCATE 和 DDL 操作。
- 暂不支持跨数据库版本进行逻辑复制。

语法格式

```
CREATE PUBLICATION name
  [ FOR TABLE table_name [, ...]
    | FOR ALL TABLES ]
  [ WITH ( publication_parameter [=value] [, ... ] ) ];
```

参数说明

- **name**

  新发布的名称。

- **FOR TABLE**

  指定要添加到发布的表的列表。 只有持久基表才能成为发布的一部分，临时表、非日志表、外表、MOT 表、物化视图、常规视图不能被发布。

- **FOR ALL TABLES**

  将发布标记为复制数据库中所有表的更改，包括在将来创建的表。

- **WITH (publication_parameter [= value] [, ...])**

  该子句指定发布的可选参数。支持下列参数：

  - **publish (string)**

    这个参数决定了哪些 DML 操作可以发布给订阅者。该值是一个用逗号分隔的操作列表，允许的操作是 insert、update 和 delete，不指定则默认发布所有的动作。该选项的默认值是'insert, update, delete'。

  - **ddl(string)**

    这个参数决定了哪些 DDL 操作可以发布给订阅者。该值是一个用逗号分隔的操作列表，允许的操作是 none、table、all，不指定则默认不发布 DDL 操作。该选项的默认值是'none'。

- **none:** 表示不发布 DDL 操作。
- **table:** 表示只发布数据表的 DDL 操作。
- **all:** 表示发布所有的 DDL 操作，目前支持的对象类型有 TABLE 和 INDEX，设置为该值时，只允许`FOR ALL TABLES` 选项。

示例

1、创建一个发布，发布两个表中所有更改。

```
CREATE PUBLICATION mypublication FOR TABLE users, departments;
```

2、创建一个发布，发布所有表中的所有更改。

```
CREATE PUBLICATION alltables FOR ALL TABLES;
```

3、创建一个发布，只发布一个表中的 INSERT 操作。

```
CREATE PUBLICATION insert_only FOR TABLE mydata WITH (publish = 'insert');
```

4、修改发布的动作。

```
ALTER PUBLICATION insert_only SET (publish='insert,update,delete');
```

5、删除发布。

```
DROP PUBLICATION insert_only;
```

6、创建一个发布，发布所有的 DDL 操作。

```
CREATE PUBLICATION ddl_all FOR ALL TABLES WITH (ddl='all');
```

7、创建一个发布，发布类型为 TABLE 的 DDL 操作。

```
CREATE PUBLICATION ddl_all FOR ALL TABLES WITH (ddl='table');
```

相关链接

参考：SQL 语法参考->SQL 语法->ALTER PUBLICATION、DROP PUBLICATION 章节。

4.5.80.CREATE RESOURCE LABEL

功能描述

创建资源标签。

注意事项

只有 poladmin、sysadmin 或初始用户能正常执行此操作。

语法格式

```
CREATE RESOURCE LABEL [IF NOT EXISTS] label_name ADD label_item_list[, ...]*;
```

- label_item_list：

```
resource_type(resource_path[, ...]*)
```

- resource_type：

```
TABLE | COLUMN | SCHEMA | VIEW | FUNCTION
```

参数说明

- **label_name**

  资源标签名称，创建时要求不能与已有标签重名。

  取值范围：字符串，要符合标识符的命名规范。

- **resource_type**

  指的是要标记的数据库资源类型。

- **resource_path**

  指的是描述具体的数据库资源的路径。

示例

```
--创建一个表 tb_for_label
panweidb=# CREATE TABLE tb_for_label(col1 text, col2 text, col3 text);

--创建一个模式 schema_for_label
panweidb=# CREATE SCHEMA schema_for_label;
```

```
--创建一个视图 view_for_label
panweidb=# CREATE VIEW view_for_label AS SELECT 1;

--创建一个函数 func_for_label
panweidb=# CREATE FUNCTION func_for_label RETURNS TEXT AS $$ SELECT col1 FR
OM tb_for_label; $$ LANGUAGE SQL;

--基于表创建资源标签
panweidb=# CREATE RESOURCE LABEL IF NOT EXISTS table_label add TABLE(public.
tb_for_label);

--基于列创建资源标签
```

```
panweidb=# CREATE RESOURCE LABEL IF NOT EXISTS column_label add COLUMN(public.tb_for_label.col1);

--基于模式创建资源标签
panweidb=# CREATE RESOURCE LABEL IF NOT EXISTS schema_label add SCHEMA(schema_for_label);

--基于视图创建资源标签
panweidb=# CREATE RESOURCE LABEL IF NOT EXISTS view_label add VIEW(view_for_label);

--基于函数创建资源标签
panweidb=# CREATE RESOURCE LABEL IF NOT EXISTS func_label add FUNCTION(func_for_label);
```

相关链接

参考：SQL 语法参考->SQL 语法->ALTER RESOURCE LABEL、DROP RESOURCE LABEL 章节。

4.5.81.CREATE ROLE

功能描述

创建角色。角色是拥有数据库对象和权限的实体。在不同的环境中角色可以认为是一个用户，一个组或者兼顾两者。

语法格式

```
CREATE ROLE role_name [ [ WITH ] option [ ... ] ] [ ENCRYPTED | UNENCRYPTED ] { PASSWORD | IDENTIFIED BY } { 'password' | DISABLE };
```

其中角色信息设置子句 option 语法为：

```
{SYSADMIN | NOSYSADMIN}
  | {MONADMIN | NOMONADMIN}
  | {OPRADMIN | NOOPRADMIN}
  | {POLADMIN | NOPOLADMIN}
  | {AUDITADMIN | NOAUDITADMIN}
```

```
| {SSOADMIN | NOSSOADMIN}
| {CREATEDB | NOCREATEDB}
| {USEFT | NOUSEFT}
| {CREATEROLE | NOCREATEROLE}
| {INHERIT | NOINHERIT}
| {LOGIN | NOLOGIN}
| {REPLICATION | NOREPLICATION}
| {INDEPENDENT | NOINDEPENDENT}
| {VCADMIN | NOVCADMIN}
| {PERSISTENCE | NOPERSISTENCE}
| CONNECTION LIMIT connlimit
| VALID BEGIN 'timestamp'
| VALID UNTIL 'timestamp'
| RESOURCE POOL 'respool'
| PERM SPACE 'spacelimit'
| TEMP SPACE 'tmpspacelimit'
| SPILL SPACE 'spillspacelimit'
| IN ROLE role_name [, ...]
| IN GROUP role_name [, ...]
| ROLE role_name [, ...]
| ADMIN rol e_name [, ...]
| USER role_name [, ...]
| SYSID uid
| DEFAULT TABLESPACE tablespace_name
| PROFILE DEFAULT
| PROFILE profile_name
| PGUSER
```

参数说明

- role_name：角色名称

  取值范围：字符串，要符合标识符的命名规范，且最多为 63 个字符。若超过 63 个字符，数据库会截断并保留前 63 个字符当做角色名称。在创建角色时，超过 63 个字符的时候数据库会给出提示信息。

  标识符需要为字母、下划线、数字（0-9）或美元符号（$），且必须以字

母（a-z）或下划线（_)开头。

- password：登录密码

  密码规则如下：

  - 密码默认不少于 8 个字符。
  - 不能与用户名及用户名倒序相同。
  - 至少包含大写字母（A-Z）、小写字母（a-z）、数字（0-9）、非字母数字字符（限定为~!@#$%^&*()-_=+|[{}];:,<.>/?）四类字符中的三类字符。
  - 密码也可以是符合格式要求的密文字符串，这种情况主要用于用户数据导入场景，不推荐用户直接使用。如果直接使用密文密码，用户需要知道密文密码对应的明文，并且保证明文密码复杂度，数据库不会校验密文密码复杂度，直接使用密文密码的安全性由用户保证。
  - 创建角色时，应当使用双引号或单引号将用户密码括起来。

  取值范围：不为空的字符串。

- EXPIRED：在创建用户时可指定 EXPIRED 参数，即创建密码失效用户，该用户不允许执行简单查询和扩展查询。只有在修改自身密码后才可正常执行语句。
- DISABLE：默认情况下，用户可以更改自己的密码，除非密码被禁用。要禁用用户的密码，请指定 DISABLE。禁用某个用户的密码后，将从系统中删除该密码，此类用户只能通过外部认证来连接数据库，例如：kerberos 认证。只有管理员才能启用或禁用密码。普通用户不能禁用初始用户的密码。要启用密码，请运行 ALTER USER 并指定密码。
- ENCRYPTED | UNENCRYPTED：控制密码存储在系统表里的口令是否加密。按照产品安全要求，密码必须加密存储，所以，UNENCRYPTED 在 PanWeiDB 中禁止使用。因为系统无法对指定的加密口令字符串进行解密，所以如果目前的口令字符串已经是用 SHA256 加密的格式，则会继续照此存放，而不管是否声明了 ENCRYPTED 或 UNENCRYPTED。这样就允许在 dump/restore 的时候重新加载加密的口令。
- SYSADMIN | NOSYSADMIN：决定一个新角色是否为“系统管理员”，具有 SYSADMIN 属性的角色拥有系统最高权限。

缺省为 NOSYSADMIN。

- MONADMIN | NOMONADMIN：定义角色是否是监控管理员。

缺省为 NOMONADMIN。

- OPRADMIN | NOOPRADMIN：定义角色是否是运维管理员。

缺省为 NOOPRADMIN。

- POLADMIN | NOPOLADMIN：定义角色是否是安全策略管理员。

缺省为 NOPOLADMIN。

- SSOADMIN | NOSSOADMIN：定义角色是否有安全管理属性。

缺省为 NOSSOADMIN。

- AUDITADMIN | NOAUDITADMIN：定义角色是否有审计管理属性。

缺省为 NOAUDITADMIN。

- CREATEDB | NOCREATEDB：决定一个新角色是否能创建数据库。新角色没有创建数据库的权限。

缺省为 NOCREATEDB。

- USEFT | NOUSEFT：该参数为保留参数，暂未启用。
- CREATEROLE | NOCREATEROLE：决定一个角色是否可以创建新角色（也就是执行 CREATE ROLE 和 CREATE USER）。一个拥有 CREATEROLE 权限的角色也可以修改和删除其他角色。

缺省为 NOCREATEROLE。

- INHERIT | NOINHERIT：这些子句决定一个角色是否“继承”它所在组的角色的权限。不推荐使用。
- LOGIN | NOLOGIN：具有 LOGIN 属性的角色才可以登录数据库。一个拥有 LOGIN 属性的角色可以认为是一个用户。

缺省为 NOLOGIN。

- REPLICATION | NOREPLICATION：定义角色是否允许流复制或设置系统为备份模式。REPLICATION 属性是特定的角色，仅用于复制。

缺省为 NOREPLICATION。

- INDEPENDENT | NOINDEPENDENT：定义私有、独立的角色。具有 INDEPENDENT 属性的角色，管理员对其进行的控制、访问的权限被分离，具体规则如下：

- 未经 INDEPENDENT 角色授权，系统管理员无权对其表对象进行增、删、查、改、拷贝、授权操作。
- 若将私有用户表的相关权限授予其他非私有用户，系统管理员也会获得同样的权限。
- 未经 INDEPENDENT 角色授权，系统管理员无权修改 INDEPENDENT 角色的继承关系。
- 系统管理员和拥有 CREATEROLE 属性的用户无权修改 INDEPENDENT 角色的表对象的属主。
- 系统管理员和拥有 CREATEROLE 属性的用户无权去除 INDEPENDENT 角色的 INDEPENDENT 属性。
- 系统管理员和拥有 CREATEROLE 属性的用户无权修改 INDEPENDENT 角色的数据库口令，INDEPENDENT 角色需管理好自身口令，口令丢失无法重置。
- 管理员属性用户不允许定义修改为 INDEPENDENT 属性。

❖ VCADMIN | NOVCADMIN：该版本没有实际意义。

❖ PERSISTENCE | NOPERSISTENCE：定义永久用户。仅允许初始用户创建、修改和删除具有 PERSISTENCE 属性的永久用户。

❖ CONNECTION LIMIT：声明该角色可以使用的并发连接数量。

系统管理员不受此参数的限制。

❖ connlimit 数据库主节点单独统计，PanWeiDB 整体的连接数 = connlimit * 当前正常数据库主节点个数。

取值范围：整数，>=-1，缺省值为 100，-1 表示没有限制。

❖ VALID BEGIN：设置角色生效的时间戳。如果省略了该子句，角色无有效开始时间限制。

❖ VALID UNTIL：设置角色失效的时间戳。如果省略了该子句，角色无有效结束时间限制。

❖ RESOURCE POOL：设置角色使用的 resource pool 名称，该名称属于系统表：pg_resource_pool。

❖ PERM SPACE：设置用户使用空间的大小。

❖ TEMP SPACE：设置用户临时表存储空间限额。

- SPILL SPACE：设置用户算子落盘空间限额。
- IN ROLE：新角色立即拥有 IN ROLE 子句中列出的一个或多个现有角色拥有的权限。不推荐使用。
- IN GROUP：IN GROUP 是 IN ROLE 过时的拼法。不推荐使用。
- ROLE：ROLE 子句列出一个或多个现有的角色，它们将自动添加为这个新角色的成员，拥有新角色所有的权限。
- ADMIN：ADMIN 子句类似 ROLE 子句，不同的是 ADMIN 后的角色可以把新角色的权限赋给其他角色。
- USER：USER 子句是 ROLE 子句过时的拼法。
- SYSID：SYSID 子句将被忽略，无实际意义。
- DEFAULT TABLESPACE：DEFAULT TABLESPACE 子句将被忽略，无实际意义。
- PROFILE：PROFILE 子句将被忽略，无实际意义。
- PGUSER：当前版本该属性没有实际意义，仅为了语法的前向兼容而保留。

注意事项

- 在数据库中添加一个新角色，角色无登录权限。
- 创建角色的用户必须具备 CREATE ROLE 的权限或者是系统管理员。
- 只允许创建相同属性的管理员用户。

示例

- 创建一个角色，名为 manager，密码为 Bigdata@123。

```
CREATE ROLE manager IDENTIFIED BY 'Bigdata@123';
```

- 创建一个角色，从 2015 年 1 月 1 日开始生效，到 2026 年 1 月 1 日失效。

```
CREATE ROLE miriam WITH LOGIN PASSWORD 'Bigdata@123' VALID BEGIN '2015-01-01' VALID UNTIL '2026-01-01';
```

- 修改角色 manager 的密码为 abcd@123。

```
ALTER ROLE manager IDENTIFIED BY 'abcd@123' REPLACE 'Bigdata@123';
```

- 修改角色 manager 为系统管理员。

```
ALTER ROLE manager SYSADMIN;
```

- 删除角色 manager。

```
DROP ROLE manager;
```

❖ 删除角色 miriam。

```
DROP ROLE miriam;
```

4.5.82.CREATE ROW LEVEL SECURITY POLICY

功能描述

对表创建行访问控制策略。

当对表创建了行访问控制策略，只有打开该表的行访问控制开关（ALTER TABLE ... ENABLE ROW LEVEL SECURITY），策略才能生效。否则不生效。

当前行访问控制影响数据表的读取操作（SELECT、UPDATE、DELETE），暂不影响数据表的写入操作（INSERT、MERGE INTO）。表所有者或系统管理员可以在 USING 子句中创建表达式，在客户端执行数据表读取操作时，数据库后台在查询重写阶段会将满足条件的表达式拼接并应用到执行计划中。针对数据表的每一条元组，当 USING 表达式返回 TRUE 时，元组对当前用户可见，当 USING 表达式返回 FALSE 或 NULL 时，元组对当前用户不可见。

行访问控制策略名称是针对表的，同一个数据表上不能有同名的行访问控制策略；对不同的数据表，可以有同名的行访问控制策略。

行访问控制策略可以应用到指定的操作（SELECT、UPDATE、DELETE、ALL），ALL 表示会影响 SELECT、UPDATE、DELETE 三种操作；定义行访问控制策略时，若未指定受影响的相关操作，默认为 ALL。

行访问控制策略可以应用到指定的用户（角色），也可应用到全部用户（PUBLIC）；定义行访问控制策略时，若未指定受影响的用户，默认为 PUBLIC。

注意事项

❖ 支持对行存表、行存分区表、列存表、列存分区表、unlogged 表、hash 表定义行访问控制策略。
❖ 不支持外表、本地临时表定义行访问控制策略。
❖ 不支持对视图定义行访问控制策略。

- 同一张表上可以创建多个行访问控制策略，一张表最多创建 100 个行访问控制策略。
- 系统管理员不受行访问控制影响，可以查看表的全量数据。
- 通过 SQL 语句、视图、函数、存储过程查询包含行访问控制策略的表，都会受影响。

语法格式

```
CREATE [ ROW LEVEL SECURITY ] POLICY policy_name ON table_name
    [ AS { PERMISSIVE | RESTRICTIVE } ]
    [ FOR { ALL | SELECT | UPDATE | DELETE } ]
    [ TO { role_name | PUBLIC | CURRENT_USER | SESSION_USER } [, ...] ]
    USING ( using_expression )
```

参数说明

- **policy_name**

  行访问控制策略名称，同一个数据表上行访问控制策略名称不能相同。

- **table_name**

  行访问控制策略的表名。

- **PERMISSIVE | RESTRICTIVE**

  PERMISSIVE 指定行访问控制策略为宽容性策略，宽容性策略的条件用 OR 表达式拼接。

  RESTRICTIVE 指定行访问控制策略为限制性策略，限制性策略的条件用 AND 表达式拼接。拼接方式如下：

  ```
  (using_expression_permissive_1 OR using_expression_permissive_2 ...) AND (using_expression_restrictive_1 AND using_expression_restrictive_2 ...)
  ```

  缺省值为 PERMISSIVE。

- **command**

  当前行访问控制影响的 SQL 操作，可指定操作包括：ALL、SELECT、UPDATE、DELETE。当未指定时，ALL 为默认值，涵盖 SELECT、UPDATE、DELETE 操作。

  当 command 为 SELECT 时，SELECT 类操作受行访问控制的影响，只能查看到满足条件(using_expression 返回值为 TRUE)的元组数据，受影

响的操作包括 SELECT、UPDATE ... RETURNING、DELETE ... RETURNING。不允许修改、删除收到访问限制的数据。

当 command 为 UPDATE 时，UPDATE 类操作受行访问控制的影响，只能更新满足条件(using_expression 返回值为 TRUE)的元组数据，受影响的操作包括 UPDATE、SELECT FOR UPDATE/SHARE、 UPDATE ... RETURNING、 SELECT ... FOR UPDATE/SHARE。

当 command 为 DELETE 时，DELETE 类操作受行访问控制的影响，只能删除满足条件(using_expression 返回值为 TRUE)的元组数据，受影响的操作包括 DELETE、 DELETE ... RETURNING。

行访问控制策略与适配的 SQL 语法关系参加下表：

表 1 ROW LEVEL SECURITY 策略与适配 SQL 语法关系

Command	SELECT/ALL policy	UPDATE/ALL policy	DELETE/ALL policy
SELECT	Existing row	No	No
SELECT FOR UPDATE/SHARE	Existing row	Existing row	No
UPDATE	No	Existing row	No
UPDATE RETURNING	Existing row	Existing row	No
DELETE	No	No	Existing row
DELETE RETURNING	Existing row	No	Existing row

- **role_name**

行访问控制影响的数据库用户。

当未指定时，PUBLIC 为默认值，PUBLIC 表示影响所有数据库用户，可以指定多个受影响的数据库用户。

⚠ **须知：**

系统管理员不受行访问控制特性影响。

- **using_expression**

  行访问控制的表达式（返回 boolean 值）。

  条件表达式中不能包含 AGG 函数和窗口（WINDOW）函数。在查询重写阶段，如果数据表的行访问控制开关打开，满足条件的表达式会添加到计划树中。针对数据表的每条元组，会进行表达式计算，只有表达式返回值为 TRUE 时，行数据对用户才可见（SELECT、UPDATE、DELETE）；当表达式返回 FALSE 时，该元组对当前用户不可见，用户无法通过 SELECT 语句查看此元组，无法通过 UPDATE 语句更新此元组，无法通过 DELETE 语句删除此元组。

示例

```
--创建用户 alice
panweidb=# CREATE USER alice PASSWORD 'xxxxxxxxx';

--创建用户 bob
panweidb=# CREATE USER bob PASSWORD 'xxxxxxxxx';

--创建数据表 all_data
panweidb=# CREATE TABLE all_data(id int, role varchar(100), data varchar(100));

--向数据表插入数据
panweidb=# INSERT INTO all_data VALUES(1, 'alice', 'alice data');
panweidb=# INSERT INTO all_data VALUES(2, 'bob', 'bob data');
panweidb=# INSERT INTO all_data VALUES(3, 'peter', 'peter data');

--将表 all_data 的读取权限赋予 alice 和 bob 用户
panweidb=# GRANT SELECT ON all_data TO alice, bob;

--打开行访问控制策略开关
panweidb=# ALTER TABLE all_data ENABLE ROW LEVEL SECURITY;

--创建行访问控制策略，当前用户只能查看用户自身的数据
```

```
panweidb=# CREATE ROW LEVEL SECURITY POLICY all_data_rls ON all_data USING(r
ole = CURRENT_USER);

--查看表 all_data 相关信息
panweidb=# d+ all_data
                        Table "public.all_data"
 Column |         Type          | Modifiers | Storage  | Stats target | Description
--------+-----------------------+-----------+----------+--------------+-------------
 id     | integer               |           | plain    |              |
 role   | character varying(100) |          | extended |              |
 data   | character varying(100) |          | extended |              |
Row Level Security Policies:
   POLICY "all_data_rls" FOR ALL
       To public
     USING (((role)::name = "current_user"()))
Has OIDs: no
Options: orientation=row, compression=no, enable_rowsecurity=true

--当前用户执行 SELECT 操作
panweidb=# SELECT * FROM all_data;
 id | role |    data
----+-------+------------
  1 | alice | alice data
  2 | bob   | bob data
  3 | peter | peter data
(3 rows)

panweidb=# EXPLAIN(COSTS OFF) SELECT * FROM all_data;
     QUERY PLAN
----------------------
 Seq Scan on all_data
(1 row)

--切换至 alice 用户执行 SELECT 操作
panweidb=# SELECT * FROM all_data;
```

```
id | role |   data
----+-------+------------
 1 | alice | alice data
(1 row)

panweidb=# EXPLAIN(COSTS OFF) SELECT * FROM all_data;
 QUERY PLAN
------------------------------------------------------------
 Seq Scan on all_data
   Filter: ((role)::name = 'alice'::name)
 Notice: This query is influenced by row level security feature
(3 rows)
```

相关链接

参考：SQL 语法参考->SQL 语法->DROP ROW LEVEL SECURITY POLICY、ALTER ROW LEVEL SECURITY POLICY 章节。

4.5.83.CREATE RULE

功能描述

定义一个新的重写规则。

注意事项

- 为了在表上定义或修改规则，你必须是该表的拥有者。
- 如果在同一个表定义了多个相同类型的规则，则按规则的名称字母顺序触发它们。
- 在视图上用于 INSERT、UPDATE、DELETE 的规则中可以添加 RETURNING 子句基于视图的字段返回。如果规则被 INSERT RETURNING、UPDATE RETURNING、DELETE RETURNING 命令触发，这些子句将用来计算输出结果。如果规则被不带 RETURNING 的命令触发，那么规则的 RETURNING 子句将被忽略。目前仅允许无条件的 INSTEAD 规则包含 RETURNING 子句，而且在同一个事件内的所有规则中最多只能

有一个 RETURNING 子句。这样就确保只有一个 RETURNING 子句可以用于计算结果。如果在任何有效规则中都不存在 RETURNING 子句，该视图上的 RETURNING 查询将被拒绝。

- 不建议在 rule 内使用列存表，尤其是一些写操作。因为列存表与行存表的架构实现、事务处理等存在很大差异，因此 rule 的表现也会有很多与行存表不同的地方。

语法格式

```
CREATE [ OR REPLACE ] RULE name AS ON event
  TO table_name [ WHERE condition ]
  DO [ ALSO | INSTEAD ] { NOTHING | command | ( command ; command ... ) }
```

其中 event 包含以下几种：

```
SELECT
INSERT
DELETE
UPDATE
```

参数说明

- **name**

  创建的规则名。它必须在同一个表上的所有规则名字中唯一。

  取值范围：符合标识符命名规范的字符串，且最大长度不超过 63 个字符。

- **table_name**

  规则作用的表或者视图的名字（可以有模式修饰）。

- **condition**

  返回 boolean 的 SQL 条件表达式，决定是否实际执行规则。表达式除了引用 NEW 和 OLD 之外不能引用任何表，并且不能有聚合函数。

- **INSTEAD**

  INSTEAD 指示使用该命令替换初始事件。

- **ALSO**

  ALSO 指示该命令应该在初始事件执行之后执行。如果既没有声明 ALSO 也没有声明 INSTEAD，那么 ALSO 为缺省值。

❖ **command**

组成规则动作的命令。有效的命令是 SELECT、 INSERT、UPDATE、DELETE 语句之一。

示例

```
CREATE RULE "_RETURN" AS
  ON SELECT TO t1
  DO INSTEAD
    SELECT * FROM t2;
```

【须知】

❖ ON SELECT 后指定的规则名必须为"_RETURN"

❖ 目前，ON SELECT 规则必须是 INSTEAD SELECT，而且 TO 所指定的表会被转为视图，这个前提是该表为空且不带有触发器、索引、子表等限制，也即必须为一张初始的空表。 因此，一般不建议采用这种写法，而是直接创建视图。

4.5.84.CREATE SCHEMA

功能描述

创建模式。

访问命名对象时可以使用模式名作为前缀进行访问，如果无模式名前缀，则访问当前模式下的命名对象。创建命名对象时也可用模式名作为前缀修饰。

另外，CREATE SCHEMA 可以包括在新模式中创建对象的子命令，这些子命令和那些在创建完模式后发出的命令没有任何区别。如果使用了 AUTHORIZATION 子句，则所有创建的对象都将被该用户所拥有。

注意事项

❖ 只要用户对当前数据库有 CREATE 权限，就可以创建模式。

❖ 系统管理员在普通用户同名 schema 下创建的对象，所有者为 schema 的同名用户（非系统管理员）。

语法格式

❖ 根据指定的名称创建模式。

```
CREATE SCHEMA [IF NOT EXISTS]  schema_name
  [ AUTHORIZATION user_name ] [WITH BLOCKCHAIN] [ schema_element [ ... ] ];
```

❖ 根据用户名创建模式。

```
CREATE SCHEMA AUTHORIZATION user_name [ schema_element [ ... ] ];
```

参数说明

❖ **schema_name**

模式名称。

【须知】

❖ 模式名不能和当前数据库里其他的模式重名。

❖ 模式的名称不可以“pg_”开头。

取值范围：字符串，要符合标识符的命名规范。

❖ **AUTHORIZATION user_name**

指定模式的所有者。当不指定 schema_name 时，把 user_name 当作模式名，此时 user_name 只能是角色名。

取值范围：已存在的用户名/角色名。

❖ **WITH BLOCKCHAIN**

指定模式的防篡改属性，防篡改模式下的行存普通用户表将自动扩展为防篡改用户表。

❖ **schema_element**

在模式里创建对象的 SQL 语句。目前仅支持 CREATE TABLE、CREATE VIEW、CREATE INDEX、CREATE PARTITION、CREATE SEQUENCE、CREATE TRIGGER、GRANT 子句。

子命令所创建的对象都被 AUTHORIZATION 子句指定的用户所拥有。

【须知】

如果当前搜索路径上的模式中存在同名对象时，需要明确指定引用对象所在的模式。可以通过命令 SHOW SEARCH_PATH 来查看当前搜索路径上的模式。

示例

1、创建一个角色 role1

```
CREATE ROLE role1 IDENTIFIED BY 'xxxxxxxxx';
```

2、为用户 role1 创建一个同名 schema，子命令创建的表 films 和 winners 的拥有者为 role1。

```
CREATE SCHEMA AUTHORIZATION role1
    CREATE TABLE films (title text, release date, awards text[])
    CREATE VIEW winners AS
    SELECT title, release FROM films WHERE awards IS NOT NULL;
```

3、删除 schema。

```
DROP SCHEMA role1 CASCADE;
```

4、删除用户。

```
DROP USER role1 CASCADE;
```

相关链接

参考：SQL 语法参考->SQL 语法->ALTER SCHEMA、DROP SCHEMA 章节。

4.5.85.CREATE SECURITY LABEL

功能描述

定义一个安全标签。

注意事项

无

语法格式

```
CREATE SECURITY LABEL label_name 'label_context';
```

参数说明

- **label_name**

安全标签名称，创建时要求不能与已有标签重名。

取值范围：字符串，要符合标识符的命名规范。

- **abel_context**

标签上下文。

示例

1、开启线程池，修改$PGDATA/postgresql.conf 文件，并重启数据库。

```
enable_thread_pool = on
```

2、新建函数。

```
create or replace function stat_parallel(out rec boolean, inout statements text) retu
rns record
as $$
begin
  begin
    rec := true;
    execute statements;
    exception when others then rec := false;
  end;
end;
$$ LANGUAGE plpgsql;
```

3、连接到 panweidb 的数据库连接池 2，执行以下操作：

```
\parallel on 2
select stat_parallel('create user user1 with password "user1@abcd";');
```

结果显示为：

```
                stat_parallel
-------------------------------------------------------------------
(t,"create user user1 with password ""user1@abcd"";")
(1 row)
```

4、执行以下操作：

```
select stat_parallel('create user user1 with password "user1@abcd";');
\parallel off
```

显示结果为：

```
                    stat_parallel
-----------------------------------------------------------------------
(f,"create user user1 with password ""user1@abcd"";")
(1 row)
```

5、创建安全标签，并为安全标签指定属性。

```
create security label l1 'L3:G1,G2';
security label on role user1 is 'l1';
```

相关链接

无

4.5.86.CREATE SEQUENCE

功能描述

CREATE SEQUENCE 用于向当前数据库里增加一个新的序列。序列的 Owner 为创建此序列的用户。

注意事项

- Sequence 是一个存放等差数列的特殊表。这个表没有实际意义，通常用于为行或者表生成唯一的标识符。
- 如果给出一个模式名，则该序列就在给定的模式中创建，否则会在当前模式中创建。序列名必须和同一个模式中的其他序列、表、索引、视图或外表的名称不同。
- 创建序列后，在表中使用序列的 nextval()函数和 generate_series(1,N)函数对表插入数据，请保证 nextval 的可调用次数大于等于 N+1 次，否则会因为 generate_series()函数会调用 N+1 次而导致报错。
- Sequence 默认最大值为 2^{63}-1，如果使用了 Large 标识则最大值可以支持到 2^{127}-1。

- 被授予 CREATE ANY SEQUENCE 权限的用户，可以在 public 模式和用户模式下创建序列。

语法格式

```
CREATE [ LARGE ] SEQUENCE name [ INCREMENT [ BY ] increment ]
  [ MINVALUE minvalue | NO MINVALUE | NOMINVALUE ] [ MAXVALUE maxvalue | NO MAXVALUE | NOMAXVALUE]
  [ START [ WITH ] start ] [ CACHE cache ] [ [ NO ] CYCLE | NOCYCLE ]
  [ OWNED BY { table_name.column_name | NONE } ];
```

参数说明

- **name**

  将要创建的序列名称。

  取值范围: 仅可以使用小写字母（a~z）、 大写字母（A~Z）、数字和特殊字符“#”，“_”，“$”的组合。

- **increment**

  指定序列的步长。一个正数将生成一个递增的序列，一个负数将生成一个递减的序列。

  缺省值为 1。

- **MINVALUE minvalue | NO MINVALUE| NOMINVALUE**

  执行序列的最小值。如果没有声明 minvalue 或者声明了 NO MINVALUE，则递增序列的缺省值为 1，递减序列的缺省值为-2^{63}+1。NOMINVALUE 等价于 NO MINVALUE。

- **MAXVALUE maxvalue | NO MAXVALUE| NOMAXVALUE**

  执行序列的最大值。如果没有声明 maxvalue 或者声明了 NO MAXVALUE，则递增序列的缺省值为 2^{63}-1，递减序列的缺省值为-1。NOMAXVALUE 等价于 NO MAXVALUE。

- **start**

  指定序列的起始值。缺省值：对于递增序列为 minvalue，递减序列为 maxvalue。

- **cache**

  为了快速访问，而在内存中预先存储序列号的个数。

缺省值为 1，表示一次只能生成一个值，也就是没有缓存。

说明：

不建议同时定义 cache 和 maxvalue 或 minvalue。因为定义 cache 后不能保证序列的连续性，可能会产生空洞，造成序列号段浪费。

- **CYCLE**

用于使序列达到 maxvalue 或者 minvalue 后可循环并继续下去。

如果声明了 NO CYCLE，则在序列达到其最大值后任何对 nextval 的调用都会返回一个错误。

NOCYCLE 的作用等价于 NO CYCLE。

缺省值为 NO CYCLE。

若定义序列为 CYCLE，则不能保证序列的唯一性。

- **OWNED BY**

将序列和一个表的指定字段进行关联。这样，在删除那个字段或其所在表的时候会自动删除已关联的序列。关联的表和序列的所有者必须是同一个用户，并且在同一个模式中。需要注意的是，通过指定 OWNED BY，仅仅是建立了表的对应列和 sequence 之间关联关系，并不会在插入数据时在该列上产生自增序列。

缺省值为 OWNED BY NONE，表示不存在这样的关联。

须知：

通过 OWNED BY 创建的 Sequence 不建议用于其他表，如果希望多个表共享 Sequence，该 Sequence 不应该从属于特定表。

示例

创建一个名为 serial 的递增序列，从 101 开始：

```
panweidb=# CREATE SEQUENCE serial
 START 101
 CACHE 20;
```

从序列中选出下一个数字：

```
panweidb=# SELECT nextval('serial');
```

```
 nextval
---------
     101
```

从序列中选出下一个数字：

```
panweidb=# SELECT nextval('serial');
 nextval
---------
     102
```

创建与表关联的序列：

```
panweidb=# CREATE TABLE customer_address
(
    ca_address_sk             integer               not null,
    ca_address_id             char(16)              not null,
    ca_street_number          char(10)                      ,
    ca_street_name            varchar(60)                   ,
    ca_street_type            char(15)                      ,
    ca_suite_number           char(10)                      ,
    ca_city                   varchar(60)                   ,
    ca_county                 varchar(30)                   ,
    ca_state                  char(2)                       ,
    ca_zip                    char(10)                      ,
    ca_country                varchar(20)                   ,
    ca_gmt_offset             decimal(5,2)                  ,
    ca_location_type          char(20)
);

panweidb=# CREATE SEQUENCE serial1
 START 101
 CACHE 20
OWNED BY customer_address.ca_address_sk;
--删除表和序列
panweidb=# DROP TABLE customer_address;
panweidb=# DROP SEQUENCE serial cascade;
```

```
panweidb=# DROP SEQUENCE serial1 cascade;
```

相关链接

参考：SQL 语法参考->SQL 语法->DROP SEQUENCE、ALTER SEQUENCE 章节。

4.5.87.CREATE SERVER

功能描述

定义一个新的外部服务器。

语法格式

```
CREATE SERVER server_name
  FOREIGN DATA WRAPPER fdw_name
  OPTIONS ( { option_name ' value ' } [, ...] ) ;
```

参数说明

- **server_name**

  server 的名称。

  取值范围：长度必须小于等于 63。

- **fdw_name**

  指定外部数据封装器的名称。

  取值范围：dist_fdw，hdfs_fdw，log_fdw，file_fdw，mot_fdw，oracle_fdw，mysql_fdw，postgres_fdw。

- **OPTIONS ({ option_name ' value ' } [, ...])**

  这个子句为服务器指定选项。这些选项通常定义该服务器的连接细节，但是实际的名称和值取决于该服务器的外部数据包装器。

  - oracle_fdw 支持的 options 包括：
    - **dbserver**

      远端 Oracle 数据库的连接字符串。

    - **isolation_level（默认值为 serializable）**

      oracle 数据库的事务隔离级别。

取值范围：serializable、 read_committed 、 read_only

- mysql_fdw 支持的 options 包括：
  - **host （默认值为 127.0.0.1）**

    MySQL Server/MariaDB 的地址。
  - **port （默认值为 3306）**

    MySQL Server/MariaDB 侦听的端口号。
- postgres_fdw 支持的 options 同 libpq 支持的连接参数一致，如下表所示：

字符串	描述
host	要链接的主机名。如果主机名以斜杠开头，则它声明使用 Unix 域套接字通讯而不是 TCP/IP 通讯；该值就是套接字文件所存储的目录。如果没有声明 host，那么默认是与位于/tmp 目录（或者安装数据库的时候声明的套接字目录）里面的 Unix-域套接字链接。在没有 Unix 域套接字的机器上，默认与 localhost 链接。
hostaddr	与之链接的主机的 IP 地址，是标准的 IPv4 地址格式，比如，172.28.40.9。如果机器支持 IPv6，那么也可以使用 IPv6 的地址。如果声明了一个非空的字符串，那么使用 TCP/IP 通讯机制。使用 hostaddr 取代 host 可以让应用避免一次主机名查找，这一点对于那些有时间约束的应用来说可能是非常重要的。不过，GSSAPI 或 SSPI 认证方法要求主机名（host）。因此，应用下面的规则：如果声明了不带 hostaddr 的 host 那么就强制进行主机名查找。如果声明中没有 host，hostaddr 的值给出服务器网络地址；如果认证方法要求主机名，那么链接尝试将失败。如果同时声明了 host 和 hostaddr，那么 hostaddr 的值作为服务器网络地址。host 的值将被忽略，除非认证方法需要它，在这种情况下它将被用作主机名。须知：要注意如果 host 不是网络地址 hostaddr 处的服务器名，那么认证很有可能失败。如果主机名（host）和主机地址都没有，那么 libpq 将使用一个本地的 Unix 域套接字进行链接；或者是在没有 Unix 域套接字的机器上，它将尝试与 localhost 链接。
port	主机服务器的端口号，或者在 Unix 域套接字链接时的套接字扩展文件名。

字符串	描述
user	要链接的用户名，缺省是与运行该应用的用户操作系统名同名的用户。
dbname	数据库名，缺省和用户名相同。
password	如果服务器要求口令认证，所用的口令。
connect_timeout	链接的最大等待时间，以秒计（用十进制整数字符串书写），0 或者不声明表示无穷。不建议把链接超时的值设置得小于 2 秒。
client_encoding	为这个链接设置 client_encoding 配置参数。除了对应的服务器选项接受的值，你可以使用 auto 从客户端中的当前环境中确定正确的编码（Unix 系统上是 LC_CTYPE 环境变量）。
options	添加命令行选项以在运行时发送到服务器。
application_name	为 application_name 配置参数指定一个值，表明当前用户身份。
keepalives	控制客户端侧的 TCP 保持激活是否使用。缺省值是 1，意思为打开，但是如果不想要保持激活，你可以更改为 0，意思为关闭。通过 Unix 域套接字做的链接忽略这个参数。
keepalives_idle	在 TCP 应该发送一个保持激活的信息给服务器之后，控制不活动的秒数。0 值表示使用系统缺省。通过 Unix 域套接字做的链接或者如果禁用了保持激活则忽略这个参数。
keepalives_interval	在 TCP 保持激活信息没有被应该传播的服务器承认之后，控制秒数。0 值表示使用系统缺省。通过 Unix 域套接字做的链接或者如果禁用了保持激活则忽略这个参数。
keepalives_count	添加命令行选项以在运行时发送到服务器。例如，设置为-c comm_debug_mode=off，则设置 guc 参数 comm_debug_mode 参数的会话的值为 off。

需要注意的是，以下几个 options 不支持设置：

- **user 和 password**

  用户名和密码将在创建 user mapping 时指定。

✧ **client_encoding**

将自动获取本地 server 的编码方式并设置该值。

✧ **application_name**

总是设置成 postgres_fdw。

➢ 用于指定外部服务器的各类参数，详细的参数说明如下所示。

✧ **encrypt**

是否对数据进行加密，该参数仅支持 type 为 OBS 时设置。默认值为 on。

取值范围：

on 表示对数据进行加密，使用 HTTPS 协议通信。

off 表示不对数据进行加密，使用 HTTP 协议通信。

✧ **access_key**

OBS 访问协议对应的 AK 值（OBS 云服务界面由用户获取），创建外表时 AK 值会加密保存到数据库的元数据表中。该参数仅支持 type 为 OBS 时设置。

✧ **secret_access_key**

OBS 访问协议对应的 SK 值（OBS 云服务界面由用户获取），创建外表时 SK 值会加密保存到数据库的元数据表中。该参数仅支持 type 为 OBS 时设置。

除了 libpq 支持的连接参数外，还额外提供 3 个 options：

❖ **use_remote_estimate**

控制 postgres_fdw 是否发出 EXPLAIN 命令以获取运行消耗估算。默认值为 false。

❖ **fdw_startup_cost**

执行一个外表扫描时的启动耗时估算。这个值通常包含建立连接、远端对请求的分析和生成计划的耗时。默认值为 100。

❖ **fdw_typle_cost**

在远端服务器上对每一个元组进行扫描时的额外消耗。这个值通常表示数据在 server 间传输的额外消耗。默认值为 0.01。

示例

创建 server。

```
CREATE FOREIGN DATA WRAPPER extstats_dummy_fdw;
CREATE SERVER extstats_dummy_srv FOREIGN DATA WRAPPER extstats_dummy_fd
w;
```

相关链接

ALTER SERVER、DROP SERVER。参考：SQL 语法参考->SQL 语法->ALTER SERVER、DROP SERVER 章节。

4.5.88.CREATE SUBSCRIPTION

功能描述

为当前数据库添加一个新的订阅。订阅表示到发布者的复制连接。

订阅名称必须与数据库中任何现有的订阅不同。 因此，此命令不仅在本地系统表中添加定义，还会在发布端创建复制槽。 在运行此命令的事务提交时，将启动逻辑复制线程以复制新订阅的数据。

注意事项

- 创建复制槽时（默认行为），CREATE SUBSCRIPTION 不能在事务块内部执行。
- 暂不支持跨数据库版本进行逻辑复制。

语法格式

```
CREATE SUBSCRIPTION subscription_name
  CONNECTION 'conninfo'
  PUBLICATION publication_name [, ...]
  [ WITH ( subscription_parameter [= value] [, ... ] ) ]
```

参数说明

- **subscription_name**

  新订阅的名称。

- **CONNECTION 'conninfo'**

连接发布端的字符串。

如'host=1.1.1.1,2.2.2.2 port=10000,20000 dbname=postgres user=repusr1 password=password_123'。

- host：发布端 IP 地址，可以同时指定发布端主机和备机的 IP 地址，如果同时指定了多个 IP，以英文逗号分隔。
- port：发布端端口，此处的端口不能使用主端口，而应该使用主端口+1 端口，否则会与线程池冲突。可以同时指定发布端主机和备机的端口，如果同时指定了多个端口，以英文逗号分隔。

⚠ 注意

host 和 port 的数量要一致，并且要一一对应。

- dbname：发布所在的数据库。
- user 和 password：用于连接发布端且具有系统管理员权限(SYSADMIN)或者运维管理员权限(OPRADMIN)的用户名和密码。password 需要加密，创建订阅前需要在订阅端执行 pw_guc generate -S xxxxxx -D $GAUSSHOME/bin -o subscription。

❖ **PUBLICATION publication_name**

要订阅的发布端的发布名称，一个订阅可以对应多个发布。

❖ **WITH (subscription_parameter [= value] [, ...])**

该子句指定订阅的可选参数。支持的参数有：

- copy_data (boolean)：指定在复制启动后是否应复制正在订阅的发布中的现有数据。默认值是 true。
- enabled (boolean)：指定订阅是否应该主动复制，或者是否应该只是设置，但尚未启动。默认值是 true。
- slot_name (string)：要使用的复制插槽的名称。默认使用订阅名称作为复制槽的名称。如果创建订阅时设置 enable 为 false，则 slot_name 将被强制设置为 NONE，即空值，即使用户指定了 slot_name 的值，表示复制槽不存在。

- synchronous_commit (enum)：该参数的值会覆盖 synchronous_commit 设置。 默认值是 off。对于逻辑复制使用 off 是安全的，如果订阅端由于缺少同步而丢失事务，数据将从发布者再次发送。进行同步逻辑复制时，一个不同的设置可能是合适的。逻辑复制线程向发布端报告写入和刷新的位置，当使用同步复制时，发布端将等待订阅端的事务日志完成实际刷新。这意味着，当订阅用于同步复制时，将订阅者的 synchronous_commit 设置为 off 可能会增加发布端服务器上 COMMIT 的延迟，原因是 off 可能会增加事务日志刷盘的延迟，从而导致发布端可能需要花费更多的时间等待订阅者完成事务日志刷盘。在这种情况下，将 synchronous_commit 设置为 local 或更高是有利的。
- binary (boolean)：该参数指定是否需要该订阅对应的发布端以二进制格式发送数据，为 true 表示需要以二进制发送，为 false 表示不以二进制格式而知以默认的文本格式发送。默认值 false。
- connect (boolean)：指定`CREATE SUBSCRIPTION`是否应该连接到发布者。设置为 false 会默认将 enabled 和 copy_data 也设置为 false。默认值 true。不允许将 connect 设置为 false 的同时将 enabled 或 copy_data 设置为 true。因为该选项设置为 false 时不会建立连接，因此表没有被订阅，所以当启用订阅后，不会复制任何内容。需要执行`ALTER SUBSCRIPTION ... REFRESH PUBLICATION`才能订阅表。
- matchddlowner (boolean)：指定订阅端在应用 DDL 操作时，是否切换到 DDL 日志信息中指定 owner 用户。

示例

1、对用于订阅的 password 进行加密。

```
pw_guc generate -S "Bigdata@123" -D $GAUSSHOME/bin -o subscription
```

2、创建订阅。

❖ 创建一个到远程服务器的订阅，复制发布 mypublication 和 insert_only 中的表，并在提交时立即开始复制。

```
CREATE SUBSCRIPTION mysub
    CONNECTION 'host=192.168.1.50 port=5432 user=foo dbname=foodb passwor
d=xxxx'
    PUBLICATION mypublication, insert_only;
```

❖ 创建一个到远程服务器的订阅，复制 insert_only 发布中的表，并且不开始复制直到稍后启用复制。

```
CREATE SUBSCRIPTION mysub
    CONNECTION 'host=192.168.1.50 port=5432 user=foo dbname=foodb passwor
d=xxxx '
    PUBLICATION insert_only
        WITH (enabled = false);
```

3、修改订阅的连接信息。

```
ALTER SUBSCRIPTION mysub CONNECTION 'host=192.168.1.51 port=5432 user=foo d
bname=foodb password=xxxx';
```

4、激活订阅。

```
ALTER SUBSCRIPTION mysub SET(enabled=true);
```

5、删除订阅。

```
DROP SUBSCRIPTION mysub;
```

相关链接

参考：SQL 语法参考->SQL 语法->ALTER SUBSCRIPTION、DROP SUBSCRIPTION 章节。

4.5.89.CREATE SYNONYM

功能描述

创建一个同义词对象。同义词是数据库对象的别名，用于记录与其他数据库对象名间的映射关系，用户可以使用同义词访问关联的数据库对象。

注意事项

- 定义同义词的用户成为其所有者。
- 若指定模式名称，则同义词在指定模式中创建。否则，在当前模式创建。
- 支持通过同义词访问的数据库对象包括：表、视图、函数和存储过程以及 package 中的存储过程和函数，不支持 package 中变量的引用。
- 使用同义词时，用户需要具有对关联对象的相应权限。
- 支持使用同义词的 DML 语句包括：SELECT、INSERT、UPDATE、DELETE、EXPLAIN、CALL、REFRESH。
- 不建议对临时表创建同义词。如果需要创建的话，需要指定同义词的目标临时表的模式名，否则无法正常使用该同义词，并且在当前会话结束前执行 DROP SYNONYM 命令。
- 删除原对象后，与之关联同义词不会被级联删除，继续访问该同义词会报错，并提示已失效。
- 被授予了 CREATE ANY SYNONYM 权限的用户能够在用户模式下创建同义词。
- 不可与同一模式下已存在的表、视图、函数以及存储过程产生命名冲突。

语法格式

```
CREATE [ OR REPLACE ] SYNONYM synonym_name
  FOR object_name;
```

参数说明

- **synonym**

  创建的同义词名字，可以带模式名。

  取值范围：字符串，要符合标识符的命名规范。

- **object_name**

  关联的对象名字，可以带模式名。

  取值范围：字符串，要符合标识符的命名规范。

  object_name 可以是不存在的对象名称。

示例

1、创建模式 ot。

```
\create schema ot;
```

2、创建表 ot.t1 及其同义词 t1。

```
CREATE TABLE ot.t1(id int, name varchar2(10));
CREATE OR REPLACE SYNONYM t1 FOR ot.t1;
```

3、使用同义词 t1。

```
SELECT * FROM t1;
 INSERT INTO t1 VALUES (1, 'ada'), (2, 'bob');
UPDATE t1 SET t1.name = 'cici' WHERE t1.id = 2;
```

4、创建同义词 v1 及其关联视图 ot.v_t1。

```
 CREATE SYNONYM v1 FOR ot.v_t1;
 CREATE VIEW ot.v_t1 AS SELECT * FROM ot.t1;
```

5、使用同义词 v1。

```
SELECT * FROM v1;
```

6、创建重载函数 ot.add 及其同义词 add。

```
CREATE OR REPLACE FUNCTION ot.add(a integer, b integer) RETURNS integer AS
$$
SELECT $1 + $2
$$
LANGUAGE sql;

CREATE OR REPLACE FUNCTION ot.add(a decimal(5,2), b decimal(5,2)) RETURNS dec
imal(5,2) AS
$$
SELECT $1 + $2
$$
LANGUAGE sql;

CREATE OR REPLACE SYNONYM add FOR ot.add;
```

7、使用同义词 add。

```
SELECT add(1,2);
 SELECT add(1.2,2.3);
```

8、创建存储过程 ot.register 及其同义词 register。

```
CREATE PROCEDURE ot.register(n_id integer, n_name varchar2(10))
```

```
SECURITY INVOKER
AS
BEGIN
  INSERT INTO ot.t1 VALUES(n_id, n_name);
END;
/

CREATE OR REPLACE SYNONYM register FOR ot.register;
```

9、使用同义词 register，调用存储过程。

```
CALL register(3,'mia');
```

10、创建序列，并为序列创建同义词。

```
create table tab_1100628(id int);
create sequence sequ1_1100628 increment 1 maxvalue 200 start with 1 owned by t
ab_1100628.id;
create synonym syn1_1100628 for sequ1_1100628;
```

11、创建调用同义词的函数。

```
create function fun_1100628(n int)return int
as
a1 int;
begin
for i in 1..n loop
insert into tab_1100628 value(syn1_1100628.nextval);
end loop;
select count(*) into a1 from tab_1100628;
return a1;
end;
/
```

12、调用函数。

```
select fun_1100628(3);
```

返回结果为：

```
 id
----
  2
  3
```

```
 4
(3 rows)
```

13、删除同义词。

```
DROP SYNONYM t1;
DROP SYNONYM IF EXISTS v1;
DROP SYNONYM IF EXISTS add;
DROP SYNONYM register;
DROP SYNONYM syn1_1100628;
DROP SCHEMA ot CASCADE;
```

相关链接

ALTER SYNONYM，DROP SYNONYM。参考：SQL 语法参考->SQL 语法->ALTER SYNONYM、DROP SYNONYM 章节。

4.5.90.CREATE TABLE

功能描述

在当前数据库中创建一个新的空白表，该表由命令执行者所有。

注意事项

- 列存表支持的数据类型请参考《PanWeiDB V2.0-S3.0.1_B01 开发者指南》->列存表支持的数据类型。
- 列存表不支持数组。
- 列存表不支持生成列。
- 列存表不支持创建全局临时表。
- 创建列存表的数量建议不超过 1000 个。
- 表中的主键约束和唯一约束必须包含分布列。
- 如果在建表过程中数据库系统发生故障，系统恢复后可能无法自动清除之前已创建的、大小为 0 的磁盘文件。此种情况出现概率小，不影响数据库系统的正常运行。
- 列存表的表级约束只支持 PARTIAL CLUSTER KEY，不支持主外键等表级约束。

- 列存表的字段约束只支持 NULL、NOT NULL、DEFAULT 常量值、UNIQUE 和 PRIMARY KEY。
- 列存表支持 delta 表，受参数 enable_delta_store 控制是否开启（可参见《PanWeiDB V2.0-S3.0.1_B01 开发者指南》->并行导入）。
- 使用 JDBC 时，支持通过 PrepareStatement 对 DEFAULT 值进行参数化设置。
- 每张表的列数最大为 1600，具体取决于列的类型，所有列的大小加起来不能超过 8192 byte（由于数据存储形式原因，实际上限略小于 8192 byte），text、varchar、char 等长度可变的类型除外。
- 被授予 CREATE ANY TABLE 权限的用户，可以在 public 模式和用户模式下创建表。如果想要创建包含 serial 类型列的表，还需要授予 CREATE ANY SEQUENCE 创建序列的权限。
- 不可与同一模式下已存在的 synonym 产生命名冲突。
- 表名长度最多支持 128 字符。
- 列名支持使用 date_add 作为关键字。

语法格式

```
CREATE [ [ GLOBAL | LOCAL ] { TEMPORARY | TEMP } | UNLOGGED ] TABLE [ IF NOT EXIS
TS ] table_name
 ( { column_name data_type [ CHARACTER SET | CHARSET charset ]
   [ compress_mode ] [ COLLATE collation ] [ column_constraint [ ... ] ]
   | table_constraint
   | LIKE source_table [ like_option [...] ] }
   [, ... ])
   [ AUTO_INCREMENT [ = ] value ]
   [ [ DEFAULT ] CHARACTER SET | CHARSET [ = ] default_charset ] [ [ DEFAULT ] COLLAT
E [ = ] default_collation ]
   [ INHERITS ( parent_table [, ... ] ) ]
   [ WITH ( {storage_parameter = value} [, ... ] ) ]
   [ ON COMMIT { PRESERVE ROWS | DELETE ROWS | DROP } ]
   [ COMPRESS | NOCOMPRESS ]
   [ TABLESPACE tablespace_name ];
```

【须知】

下列语法仅在 MySQL 兼容模式下可用，更多内容可参考 MySQL 兼容性手册下的 CREATE TABLE。

- AUTO_INCREMENT [=] value
- [DEFAULT] CHARACTER SET | CHARSET [=] default_charset] [[DEFAULT] COLLATE [=] default_collation]

- 其中列约束 column_constraint 为：

```
[ CONSTRAINT [ constraint_name ] ]
{ NOT NULL |
NULL |
CHECK ( expression ) [ NO INHERIT ] |
DEFAULT default_expr |
GENERATED ALWAYS AS ( generation_expr ) [STORED] |
GENERATED { ALWAYS | BY DEFAULT } AS IDENTITY [(sequence_options[,...])] |
AUTO_INCREMENT |
IDENTITY [ (seed , increment) ] |
ON UPDATE update_expr |
GENERATED BY DEFAULT [ON NULL] AS IDENTITY [(sequence_options[,...])]|
UNIQUE [KEY] index_parameters |
PRIMARY KEY index_parameters |
ENCRYPTED WITH ( COLUMN_ENCRYPTION_KEY = column_encryption_key, ENCRYPTION_TYPE = encryption_type_value ) |
REFERENCES reftable [ ( refcolumn ) ] [ MATCH FULL | MATCH PARTIAL | MATCH SIMPLE ]
[ ON DELETE action ] [ ON UPDATE action ] }
[ DEFERRABLE | NOT DEFERRABLE | INITIALLY DEFERRED | INITIALLY IMMEDIATE ]
```

【须知】

- Identity 语法仅在 PostgreSQL 兼容模式下可用，更多内容可参考 PostgreSQL 兼容性手册下的 CREATE TABLE。
- 列约束 ON UPDATE 特性仅在 MySQL 兼容模式下可用，更多内容可参考 MySQL 兼容性手册下的 CREATE TABLE。

- 其中列的压缩可选项 compress_mode 为：

```
{ DELTA | PREFIX | DICTIONARY | NUMSTR | NOCOMPRESS }
```

- 其中表约束 table_constraint 为：

```
[ CONSTRAINT constraint_name ]
{ CHECK ( expression ) |
  UNIQUE ( column_name [, ... ] ) index_parameters |
  PRIMARY KEY ( column_name [, ... ] ) index_parameters |
  FOREIGN KEY ( column_name [, ... ] ) REFERENCES reftable [ (refcolumn [, ... ] ) ]
    [ MATCH FULL | MATCH PARTIAL | MATCH SIMPLE ] [ ON DELETE action ] [ ON UPDATE action ] |
  PARTIAL CLUSTER KEY ( column_name [, ... ] ) }
[ DEFERRABLE | NOT DEFERRABLE | INITIALLY DEFERRED | INITIALLY IMMEDIATE ]
```

- 其中 like 选项 like_option 为：

```
{ INCLUDING | EXCLUDING } { DEFAULTS | GENERATED | CONSTRAINTS | INDEXES | STORAGE | COMMENTS | PARTITION | RELOPTIONS | ALL }
```

- 其中索引参数 index_parameters 为：

```
[ WITH ( {storage_parameter = value} [, ... ] ) ]
    [ USING INDEX TABLESPACE tablespace_name ]
```

- 创建虚拟列分区表语法如下：

```
    CREATE TABLE [IF NOT EXISTS ] partition_table_name
    ([{column_name data_type [ COLLATE_COLLATION ] [column_constraint [...]] | table_constraint
    | LIKE source_table [ like_option [...]]}
    [,...]
    [column_name2 data_type GENERATED ALWAYS AS (column_expression) STORED] ])
```

```
    [ WITH ( { storage_parameter = value} [,...] ) ]
    [ COMPRESS | NOCOMPRESS ]
    [ TABLESPACE tablespace_name ]
    [ TO { GROUP groupname | NODE ( nodename [,...] ) } ]
    PARTITION BY {
        {VALUES (column_name [,...] ) }
        | {RANGE ( column_name2 [,...] ) [INTERVAL ('interval_expr') [STORE IN
(tablespace_name [,...])]]}
        | { LIST ( column_name2 [,...])}
        | { HASH (column_name2 [,...])}
    }
    [ SUBPARTITON BY [RANGE | LIST | HASH] (column_name2 [,...])]
    [{ SUBPARTITION TEMPLATE ( subpartition_desc [,...])} |
hash_subpartitions_by_quantity]
    table_partitioning_clauses
    [ { ENABLE | DISABLE} ROW MOVEMENT];
```

【须知】

column_name2 可以使用新增的虚拟列作为分区键。

参数说明

- **UNLOGGED**

  如果指定此关键字，则创建的表为非日志表。在非日志表中写入的数据不会被写入到预写日志中，这样就会比普通表快很多。但是非日志表在冲突、执行操作系统重启、强制重启、切断电源操作或异常关机后会被自动截断，会造成数据丢失的风险。非日志表中的内容也不会被复制到备服务器中。

在非日志表中创建的索引也不会被自动记录。

- 使用场景：非日志表不能保证数据的安全性，用户应该在确保数据已经做好备份的前提下使用，例如系统升级时进行数据的备份。
- 故障处理：当异常关机等操作导致非日志表上的索引发生数据丢失时，用户应该对发生错误的索引进行重建。
  - UNLOGGED 表和表上的索引因为数据写入时不通过 WAL 日志机制，写入速度远高于普通表。因此，可以用于缓冲存储复杂查询的中间结果集，增强复杂查询的性能。
  - UNLOGGED 表无主备机制，在系统故障或异常断点等情况下，会有数据丢失风险，因此，不可用来存储基础数据。

- **GLOBAL | LOCAL**

  创建临时表时可以在 TEMP 或 TEMPORARY 前指定 GLOBAL 或 LOCAL 关键字。如果指定 GLOBAL 关键字，PanWeiDB 会创建全局临时表，否则 PanWeiDB 会创建本地临时表。

- **TEMPORARY | TEMP**

  如果指定 TEMP 或 TEMPORARY 关键字，则创建的表为临时表。临时表分为全局临时表和本地临时表两种类型。创建临时表时如果指定 GLOBAL 关键字则为全局临时表，否则为本地临时表。

  全局临时表的元数据对所有会话可见，会话结束后元数据继续存在。会话与会话之间的用户数据、索引和统计信息相互隔离，每个会话只能看到和更改自己提交的数据。全局临时表有两种模式：一种是基于会话级别的(ON COMMIT PRESERVE ROWS), 当会话结束时自动清空用户数据；一种是基于事务级别的(ON COMMIT DELETE ROWS), 当执行 commit 或 rollback 时自动清空用户数据。建表时如果没有指定 ON COMMIT 选项，则缺省为会话级别。与本地临时表不同，全局临时表建表时可以指定非 pg_temp_开头的 schema。

  本地临时表只在当前会话可见，本会话结束后会自动删除。因此，在除当前会话连接的数据库节点故障时，仍然可以在当前会话上创建和使用临时表。由于临时表只在当前会话创建，对于涉及对临时表操作的 DDL 语句，会产

生 DDL 失败的报错。因此，建议 DDL 语句中不要对临时表进行操作。

TEMP 和 TEMPORARY 等价。

【须知】

- 本地临时表通过每个会话独立的以 pg_temp 开头的 schema 来保证只对当前会话可见，因此，不建议用户在日常操作中手动删除以 pg_temp、pg_toast_temp 开头的 schema。
- 如果建表时不指定 TEMPORARY/TEMP 关键字，而指定表的 schema 为当前会话的 pg_temp_开头的 schema，则此表会被创建为临时表。
- ALTER/DROP 全局临时表和索引，如果其他会话正在使用它，禁止操作（ALTER INDEX index_name REBUILD 除外）。
- 全局临时表的 DDL 只会影响当前会话的用户数据和索引。例如 truncate、reindex、analyze 只对当前会话有效。
- 全局临时表功能可以通过设置 GUC 参数 max_active_global_temporary_table 控制是否启用（参见《PanWeiDB V2.0-S3.0.1_B01 开发者指南》->全局临时表）。如果 max_active_global_temporary_table=0，关闭全局临时表功能。
- 临时表只对当前会话可见，因此不支持与 parallel on 并行执行一起使用。
- 临时表不支持主备切换。
- 全局临时表不响应自动清理，在长链接场景使用时尽量使用 on commit delete rows 的全局临时表，或定期手动执行 vacuum，否则可能导致 clog 日志不回收。

- **IF NOT EXISTS**

  如果已经存在相同名称的表，不会报出错误，而会发出通知，告知通知此表已存在。

- **table_name**

  要创建的表名。

【须知】
物化视图的一些处理逻辑会通过表名的前缀来识别是不是物化视图日志表和物化视图关联表，因此，用户不要创建表名以 mlog 或 matviewmap 为前缀的表，否则会影响此表的一些功能。

❖ **column_name**

新表中要创建的字段名。

❖ **data_type**

字段的数据类型。

❖ **compress_mode**

表字段的压缩选项。该选项指定表字段优先使用的压缩算法。行存表不支持压缩。

取值范围：DELTA、PREFIX、DICTIONARY、NUMSTR、NOCOMPRESS

- DELTA 压缩仅支持长度为 1-8 字节的数据类型（0 < pg_type.typlen <= 8）。
- PREFIX、NUMSTR 压缩仅支持变长数据类型（pg_type.typlen = -1）和 NULL 结尾的 C 字符串（pg_type.typlen = -2）。
- 该压缩选项与列存表自适应压缩算法无关，后者为列存表内部数据存储采用的压缩算法，不支持用户指定。

❖ **COLLATE collation**

COLLATE 子句指定列的排序规则（该列必须是可排列的数据类型）。如果没有指定，则使用默认的排序规则。排序规则可以使用 select * from pg_collation;命令从 pg_collation 系统表中查询，默认的排序规则为查询结果中以 default 开始的行。

❖ **LIKE source_table [like_option ...]**

LIKE 子句声明一个表，新表自动从这个表中继承所有字段名及其数据类型和非空约束。新表与源表之间在创建动作完毕之后是完全无关的。在源表做的任何修改都不会传播到新表中，并且也不可能在扫描源表的时候包含新表的数据。被复制的列和约束并不使用相同的名称进行融合。如果明确的指定了相同的名称或者在另外一个 LIKE 子句中，将会报错。

- 源表上的字段缺省表达式只有在指定 INCLUDING DEFAULTS 时，才会复制到新表中。缺省是不包含缺省表达式的，即新表中的所有字段的缺省值都是 NULL。
- 源表上的 CHECK 约束仅在指定 INCLUDING CONSTRAINTS 时，会复制到新表中，而其他类型的约束永远不会复制到新表中。非空约束总是复制到新表中。此规则同时适用于表约束和列约束。
- 源表上的索引默认在新表上创建，且不影响指定 INCLUDING INDEXES，若不希望复制源表索引，需指定 EXCLUDING INDEXES。
- 如果指定了 INCLUDING STORAGE，则复制列的 STORAGE 设置会复制到新表中，默认情况下不包含 STORAGE 设置。
- 如果指定了 INCLUDING COMMENTS，则源表列、约束和索引的注释会复制到新表中。默认情况下，不复制源表的注释。
- 如果源表为分区表，则源表的分区定义会默认复制到新表中，同时新表将不能再使用 PRTITION BY 子句，不影响指定 INCLUDING PARTITION。若不希望复制分区信息，需指定 EXCLUDING PARTITION。如果源分区表还带有索引，只使用 EXCLUDING PARTITION，目标表定义将是普通表，但默认复制源表分区索引，结果会报错，因为普通表不支持分区索引。
- 如果指定了 INCLUDING RELOPTIONS，则源表的存储参数（即源表的 WITH 子句）会复制到新表中。默认情况下，不复制源表的存储参数。
- INCLUDING ALL 包含了 INCLUDING DEFAULTS、INCLUDING CONSTRAINTS、INCLUDING INDEXES、INCLUDING STORAGE、INCLUDING COMMENTS、INCLUDING PARTITION 和 INCLUDING RELOPTIONS 的内容。
- ATUO_INCREMENT 列需要为主键或唯一约束的第一个字段，若复制包含 AUTO_INCREAMENT 列的表时指定 EXCLUDING INDEX，将会报错。

其中 AUTO_INCREAMENT 只在 B 库中生效。请参考 MySQL 兼容性手册的 CREATE TABLE。

- 新表自动从这个表中继承所有字段名及其数据类型和非空约束，新表与源表之间在创建动作完毕之后是完全无关的。

【须知】

- 如果源表包含 serial、bigserial、smallserial、largeserial 类型，或者源表字段的默认值是 sequence，且 sequence 属于源表（通过 CREATE SEQUENCE ... OWNED BY 创建），这些 Sequence 不会关联到新表中，新表中会重新创建属于自己的 sequence。这和之前版本的处理逻辑不同。如果用户希望源表和新表共享 Sequence，需要首先创建一个共享的 Sequence（避免使用 OWNED BY），并配置为源表字段默认值，这样创建的新表会和源表共享该 Sequence。
- 不建议将其他表私有的 Sequence 配置为源表字段的默认值，尤其是其他表只分布在特定的 NodeGroup 上，这可能导致 CREATE TABLE ... LIKE 执行失败。另外，如果源表配置其他表私有的 Sequence，当该表删除时 Sequence 也会连带删除，这样源表的 Sequence 将不可用。如果用户希望多个表共享 Sequence，建议创建共享的 Sequence。
- 对于分区表 EXCLUDING，需要配合 INCLUDING ALL 使用，如 INCLUDING ALL EXCLUDING DEFAULTS，除源分区表的 DEFAULTS，其他全包含。
- 如果源表是本地临时表，则新表也必须是本地临时表，否则会报错。
- 如果源表是 hash 或 list 分区表，则在 CREATE TABLE ... (LIKE ... INCLUDIING PARTITION)时会报错，不支持复制 hash 或 list 分区表的分区，仅支持 range 分区。对于二级分区表，同样只支持 range-range 二级分区。

- **WITH ({ storage_parameter = value } [, ...])**

  这个子句为表或索引指定一个可选的存储参数。

【须知】

使用任意精度类型 Numeric 定义列时，建议指定精度 p 以及刻度 s。在不指定精度和刻度时，会按输入的显示出来。

参数的详细描述如下所示。

➢ FILLFACTOR：

一个表的填充因子（fillfactor）是一个介于 10 和 100 之间的百分数。100（完全填充）是默认值。如果指定了较小的填充因子，INSERT 操作仅按照填充因子指定的百分率填充表页。每个页上的剩余空间将用于在该页上更新行，这就使得 UPDATE 有机会在同一页上放置同一条记录的新版本，这比把新版本放置在其他页上更有效。对于一个从不更新的表将填充因子设为 100 是最佳选择，但是对于频繁更新的表，选择较小的填充因子则更加合适。该参数对于列存表没有意义。

取值范围：10~100

➢ ORIENTATION：

指定表数据的存储方式，即行存方式、列存方式，该参数设置成功后就不再支持修改。

取值范围：

●ROW，表示表的数据将以行式存储。行存储适合于 OLTP 业务，适用于点查询或者增删操作较多的场景。

●COLUMN，表示表的数据将以列式存储。列存储适合于数据仓库业务，此类型的表上会做大量的汇聚计算，且涉及的列操作较少。

●默认值：若指定表空间为普通表空间，默认值为 ROW。

➢ STORAGE_TYPE：

指定存储引擎类型，该参数设置成功后就不再支持修改。

取值范围：USTORE，表示表支持 Inplace-Update 存储引擎。ASTORE 表示表支持 Append-Only 存储引擎。

默认值：不指定表时，默认是 Append-Only 存储。

- INIT_TD：

创建 Ustore 表时，指定初始化的 TD 个数，该参数只在创建 Ustore 表时才能设置生效。

取值范围：2~128

默认值：4。

- COMPRESSION：

指定表数据的压缩级别，它决定了表数据的压缩比以及压缩时间。一般来讲，压缩级别越高，压缩比也越大，压缩时间也越长；反之亦然。实际压缩比取决于加载的表数据的分布特征。行存表默认增加 COMPRESSION=NO 字段。

取值范围：列存表的有效值为 YES/NO/LOW/MIDDLE/HIGH 。

默认值：LOW。

- COMPRESSLEVEL：

指定表数据同一压缩级别下的不同压缩水平，它决定了同一压缩级别下表数据的压缩比以及压缩时间。对同一压缩级别进行了更加详细的划分，为用户选择压缩比和压缩时间提供了更多的空间。总体来讲，此值越大，表示同一压缩级别下压缩比越大，压缩时间越长；反之亦然。

取值范围：0~3

默认值：0。

- COMPRESSTYPE：

行存表参数，设置行存表压缩算法。1 代表 pglz 算法，2 代表 zstd 算法，默认不压缩。（仅支持 ASTORE 下的普通表）

取值范围：0~2

默认值：0。

- COMPRESS_LEVEL：

行存表参数，设置行存表压缩算法等级，仅当 COMPRESSTYPE 为 2 时生效。压缩等级越高，表的压缩效果越好，表的访问速度越慢。（仅支持 ASTORE 下的普通表）

取值范围：-31~31

默认值：0

➢ COMPRESS_CHUNK_SIZE：

行存表参数，设置行存表压缩 chunk 块大小。chunk 数据块越小，预期能达到的压缩效果越好，同时数据越离散，影响表的访问速度。（仅支持 ASTORE 下的普通表）

取值范围：与页面大小有关。在页面大小为 8k 场景，取值范围为：512、1024、2048、4096。

默认值：4096

➢ COMPRESS_PREALLOC_CHUNKS：

行存表参数，设置行存表压缩 chunk 块预分配数量。预分配数量越大，表的压缩率相对越差，离散度越小，访问性能越好。（仅支持 ASTORE 下的普通表）。

取值范围：0~7

默认值：0

●当 COMPRESS_CHUNK_SIZE 为 512 和 1024 时，支持预分配设置最大为

●当 COMPRESS_CHUNK_SIZE 为 2048 时，支持预分配设置最大为 3。

●当 COMPRESS_CHUNK_SIZE 为 4096 时，支持预分配设置最大为 1。

➢ COMPRESS_BYTE_CONVERT：

行存表参数，设置行存表压缩字节转换预处理。在一些场景下可以提升压缩效果，同时会导致一定性能劣化。

取值范围：布尔值

默认值：关闭

➢ COMPRESS_DIFF_CONVERT：

行存表参数，设置行存表压缩字节差分预处理。只能与 compress_byte_convert 一起使用。在一些场景下可以提升压缩效果，同时会导致一定性能劣化。

取值范围：布尔值

默认值：关闭

➢ AUTOVACUUM_ENABLED：

需数据库开启 autovacuum 功能时，单独设置该表是否进行 autovacuum。

取值范围：布尔值。

默认值：开启

➢ AUTOVACUUM、AUTOANALYZE 相关参数：

参数有：AUTOVACUUM_VACUUM_THREASHOLD、AUTOVACUUM_ANALYZE_THREASHOLD、AUTOVACUUM_VACUUM_COST_DELAY、AUTOVACUUM_VACUUM_COST_LIMIT、AUTOVACUUM_FREEZE_MIN_AGE、AUTOVACUUM_FREEZE_MAX_AGE、AUTOVACUUM_FREEZE_TABLE_AGE、AUTOVACUUM_VACUUM_SCALE_FACTOR、AUTOVACUUM_ANALYZE_SCALE_FACTOR

单独设置此表的 autovacuum、autoanalyze 相关功能参数配置，与同名 GUC 功能相同，优先生效此处的配置。

取值范围：与同名 GUC 相同

➢ MAX_BATCHROW：

指定了在数据加载过程中一个存储单元可以容纳记录的最大数目。该参数只对列存表有效。

取值范围：10000~60000

默认值：60000。

- PARTIAL_CLUSTER_ROWS：

指定了在数据加载过程中进行将局部聚簇存储的记录数目。该参数只对列存表有效。

取值范围：大于等于 MAX_BATCHROW，建议取值为 MAX_BATCHROW 的整数倍。

- DELTAROW_THRESHOLD：

指定列存表导入时小于多少行的数据进入 delta 表，只在 GUC 参数 enable_delta_store 开启时生效。该参数只对列存表有效。

取值范围：0～9999

默认值：100

- segment：

使用段页式的方式存储。本参数仅支持行存表。不支持列存表、临时表、unlog 表。不支持 ustore 存储引擎。

取值范围：on/off

默认值：off

- dek_cipher：

透明数据加密密钥的密文。当开启 enable_tde 选项时会自动申请创建，用户不可单独指定。通过密钥轮转功能可以对密钥进行更新。

取值范围：字符串。

默认值：不开启加密时默认为空。

- cmuids

参数开启：更新表元组时，为元组分配表级唯一标识 id。

取值范围：on/off。

默认值：off

❖ **ON COMMIT { PRESERVE ROWS | DELETE ROWS | DROP }**

ON COMMIT 选项决定在事务中执行创建临时表操作，当事务提交时，此临时表的后续操作。有以下三个选项，当前支持 PRESERVE ROWS 和 DELETE ROWS 选项。

- PRESERVE ROWS（缺省值）：提交时不对临时表做任何操作，临时表及其表数据保持不变。
- DELETE ROWS：提交时删除临时表中数据。
- DROP：提交时删除此临时表。只支持本地临时表，不支持全局临时表。

❖ **COMPRESS | NOCOMPRESS**

创建新表时，需要在 CREATE TABLE 语句中指定关键字 COMPRESS，这样，当对该表进行批量插入时就会触发压缩特性。该特性会在页范围内扫描所有元组数据，生成字典、压缩元组数据并进行存储。指定关键字 NOCOMPRESS 则不对表进行压缩。行存表不支持压缩。该参数已废弃，列存表请使用 COMPRESSION 修改压缩等级。

缺省值：NOCOMPRESS，即不对元组数据进行压缩。

❖ **TABLESPACE tablespace_name**

创建新表时指定此关键字，表示新表将要在指定表空间内创建。如果没有声明，将使用默认表空间。

❖ **CONSTRAINT constraint_name**

列约束或表约束的名称。可选的约束子句用于声明约束，新行或者更新的行必须满足这些约束才能成功插入或更新。定义约束有两种方法：

- 列约束：作为一个列定义的一部分，仅影响该列。
- 表约束：不和某个列绑在一起，可以作用于多个列。

❖ **NOT NULL**

字段值不允许为 NULL。

❖ **NULL**

字段值允许为 NULL，这是缺省值。这个子句只是为和非标准 SQL 数据库兼容。不建议使用。

❖ **CHECK (expression)**

CHECK 约束声明一个布尔表达式，每次要插入的新行或者要更新的行的新值必须使表达式结果为真或未知才能成功，否则会抛出一个异常并且不会修改数据库。声明为字段约束的检查约束应该只引用该字段的数值，而在表约束里出现的表达式可以引用多个字段。

【须知】

expression 表达式中，如果存在“<>NULL”或“！=NULL”，这种写法是无效的，需要写成“is NOT NULL”。

❖ **DEFAULT default_expr**

DEFAULT 子句给字段指定缺省值。该数值可以是任何不含变量的表达式（不允许使用子查询和对本表中的其他字段的交叉引用）。缺省表达式的数据类型必须和字段类型匹配。缺省表达式将被用于任何未声明该字段数值的插入操作。如果没有指定缺省值则缺省值为 NULL 。

❖ **UNIQUE (column_name [, ...]) index_parameters**

UNIQUE 约束表示表里的一个字段或多个字段的组合必须在全表范围内唯一。对于唯一约束，NULL 被认为是互不相等的。

❖ **PRIMARY KEY (column_name [, ...]) index_parameters**

主键约束声明表中的一个或者多个字段只能包含唯一的非 NULL 值。一个表只能声明一个主键。

❖ **REFERENCES reftable [(refcolum)] [MATCH matchtype] [ON DELETE action][ON UPDATE action](column constraint\)**

➢ FOREIGN KEY (column_name [, ...]) REFERENCES reftable [(refcolumn [, ...])] [MATCH matchtype] [ON DELETE action]

[ON UPDATE action \] (table constraint)：外键约束要求新表中一列或多列构成的组应该只包含、匹配被参考表中被参考字段值。若省略 refcolum，则将使用 reftable 的主键。被参考列应该是被参考表中的唯一字段或主键。外键约束不能被定义在临时表和永久表之间。参考字段与被参考字段之间存在三种类型匹配，分别是：

- MATCH FULL：不允许一个多字段外键的字段为 NULL，除非全部外键字段都是 NULL。
- MATCH SIMPLE（缺省）：允许任意外键字段为 NULL。
- MATCH PARTIAL：目前暂不支持。

另外，当被参考表中的数据发生改变时，某些操作也会在新表对应字段的数据上执行。ON DELETE 子句声明当被参考表中的被参考行被删除时要执行的操作。ON UPDATE 子句声明当被参考表中的被参考字段数据更新时要执行的操作。对于 ON DELETE 子句、ON UPDATE 子句的可能动作：

- NO ACTION（缺省）：删除或更新时，创建一个表明违反外键约束的错误。若约束可推迟，且若仍存在任何引用行，那这个错误将会在检查约束的时候产生。
- RESTRICT：删除或更新时，创建一个表明违反外键约束的错误。与 NO ACTION 相同，只是动作不可推迟。
- CASCADE：删除新表中任何引用了被删除行的行，或更新新表中引用行的字段值为被参考字段的新值。
- SET NULL：设置引用字段为 NULL。
- SET DEFAULT：设置引用字段为它们的缺省值。

❖ **DEFERRABLE | NOT DEFERRABLE**

这两个关键字设置该约束是否可推迟。一个不可推迟的约束将在每条命令之后马上检查。可推迟约束可以推迟到事务结尾使用 SET CONSTRAINTS 命令检查。缺省是 NOT DEFERRABLE。目前，UNIQUE 约束、主键约束、外键约束可以接受这个子句。所有其他约束类型都是不可推迟的。

【须知】

Ustore 表不支持 DEFERRABLE 以及 INITIALLY DEFERRED 关键字。

❖ **PARTIAL CLUSTER KEY**

局部聚簇存储，列存表导入数据时按照指定的列(单列或多列)，进行局部排序。

❖ **INITIALLY IMMEDIATE | INITIALLY DEFERRED**

如果约束是可推迟的，则这个子句声明检查约束的缺省时间。

- 如果约束是 INITIALLY IMMEDIATE（缺省），则在每条语句执行之后就立即检查它；
- 如果约束是 INITIALLY DEFERRED，则只有在事务结尾才检查它。

约束检查的时间可以用 SET CONSTRAINTS 命令修改。

❖ **USING INDEX TABLESPACE tablespace_name**

为 UNIQUE 或 PRIMARY KEY 约束相关的索引声明一个表空间。如果没有提供这个子句，这个索引将在 default_tablespace 中创建，如果 default_tablespace 为空，将使用数据库的缺省表空间。

❖ **ENCRYPTION_TYPE = encryption_type_value**

为 ENCRYPTED WITH 约束中的加密类型，encryption_type_value 的值为[DETERMINISTIC | RANDOMIZED]

示例

示例 1 创建简单的表。

```
CREATE TABLE warehouse_t1
(
 W_WAREHOUSE_SK     INTEGER      NOT NULL,
 W_WAREHOUSE_ID     CHAR(16)     NOT NULL,
 W_WAREHOUSE_NAME   VARCHAR(20)       ,
 W_WAREHOUSE_SQ_FT  INTEGER           ,
 W_STREET_NUMBER    CHAR(10)          ,
```

```
 W_STREET_NAME      VARCHAR(60)       ,
 W_STREET_TYPE      CHAR(15)          ,
 W_SUITE_NUMBER     CHAR(10)          ,
 W_CITY         VARCHAR(60)       ,
 W_COUNTY       VARCHAR(30)       ,
 W_STATE        CHAR(2)           ,
 W_ZIP          CHAR(10)          ,
 W_COUNTRY      VARCHAR(20)       ,
 W_GMT_OFFSET       DECIMAL(5,2)
);
```

示例 2：创建表，并指定 W_STATE 字段的缺省值为 GA。

```
CREATE TABLE warehouse_t2
(
 W_WAREHOUSE_SK     INTEGER       NOT NULL,
 W_WAREHOUSE_ID     CHAR(16)      NOT NULL,
 W_WAREHOUSE_NAME   VARCHAR(20)       ,
 W_WAREHOUSE_SQ_FT  INTEGER           ,
 W_STREET_NUMBER    CHAR(10)          ,
 W_STREET_NAME      VARCHAR(60)       ,
 W_STREET_TYPE      CHAR(15)          ,
 W_SUITE_NUMBER     CHAR(10)          ,
 W_CITY         VARCHAR(60)       ,
 W_COUNTY       VARCHAR(30)       ,
 W_STATE        CHAR(2)     DEFAULT 'GA',
 W_ZIP          CHAR(10)          ,
 W_COUNTRY      VARCHAR(20)       ,
 W_GMT_OFFSET       DECIMAL(5,2)
);
```

示例 3：创建表，并在事务结束时检查 W_WAREHOUSE_NAME 字段是否有重复。

```
CREATE TABLE warehouse_t3
(
```

```
 W_WAREHOUSE_SK     INTEGER       NOT NULL,
 W_WAREHOUSE_ID     CHAR(16)      NOT NULL,
 W_WAREHOUSE_NAME     VARCHAR(20)  UNIQUE DEFERRABLE,
 W_WAREHOUSE_SQ_FT    INTEGER           ,
 W_STREET_NUMBER     CHAR(10)         ,
 W_STREET_NAME      VARCHAR(60)        ,
 W_STREET_TYPE      CHAR(15)         ,
 W_SUITE_NUMBER      CHAR(10)         ,
 W_CITY         VARCHAR(60)        ,
 W_COUNTY        VARCHAR(30)        ,
 W_STATE        CHAR(2)          ,
 W_ZIP         CHAR(10)         ,
 W_COUNTRY       VARCHAR(20)        ,
 W_GMT_OFFSET       DECIMAL(5,2)
);
```

示例 4：创建一个带有 70%填充因子的表。

```
CREATE TABLE warehouse_t4
(
 W_WAREHOUSE_SK     INTEGER       NOT NULL,
 W_WAREHOUSE_ID     CHAR(16)      NOT NULL,
 W_WAREHOUSE_NAME     VARCHAR(20)        ,
 W_WAREHOUSE_SQ_FT    INTEGER           ,
 W_STREET_NUMBER     CHAR(10)         ,
 W_STREET_NAME      VARCHAR(60)        ,
 W_STREET_TYPE      CHAR(15)         ,
 W_SUITE_NUMBER      CHAR(10)         ,
 W_CITY         VARCHAR(60)        ,
 W_COUNTY        VARCHAR(30)        ,
 W_STATE        CHAR(2)          ,
 W_ZIP         CHAR(10)         ,
 W_COUNTRY       VARCHAR(20)        ,
 W_GMT_OFFSET       DECIMAL(5,2),
 UNIQUE(W_WAREHOUSE_NAME) WITH(fillfactor=70)
);
```

或者用下面的语法。

```
CREATE TABLE warehouse_t5
(
 W_WAREHOUSE_SK     INTEGER       NOT NULL,
 W_WAREHOUSE_ID     CHAR(16)      NOT NULL,
 W_WAREHOUSE_NAME     VARCHAR(20)      UNIQUE,
 W_WAREHOUSE_SQ_FT    INTEGER          ,
 W_STREET_NUMBER     CHAR(10)         ,
 W_STREET_NAME      VARCHAR(60)         ,
 W_STREET_TYPE      CHAR(15)         ,
 W_SUITE_NUMBER      CHAR(10)         ,
 W_CITY         VARCHAR(60)         ,
 W_COUNTY        VARCHAR(30)         ,
 W_STATE        CHAR(2)          ,
 W_ZIP         CHAR(10)         ,
 W_COUNTRY       VARCHAR(20)         ,
 W_GMT_OFFSET       DECIMAL(5,2)
) WITH(fillfactor=70);
```

示例 5：创建表，并指定该表数据不写入预写日志。

```
CREATE UNLOGGED TABLE warehouse_t6
(
 W_WAREHOUSE_SK     INTEGER       NOT NULL,
 W_WAREHOUSE_ID     CHAR(16)      NOT NULL,
 W_WAREHOUSE_NAME     VARCHAR(20)         ,
 W_WAREHOUSE_SQ_FT    INTEGER          ,
 W_STREET_NUMBER     CHAR(10)         ,
 W_STREET_NAME      VARCHAR(60)         ,
 W_STREET_TYPE      CHAR(15)         ,
 W_SUITE_NUMBER      CHAR(10)         ,
 W_CITY         VARCHAR(60)         ,
 W_COUNTY        VARCHAR(30)         ,
 W_STATE        CHAR(2)          ,
 W_ZIP         CHAR(10)         ,
```

```
 W_COUNTRY       VARCHAR(20)       ,
 W_GMT_OFFSET    DECIMAL(5,2)
);
```

示例 6：创建表临时表。

```
CREATE TEMPORARY TABLE warehouse_t7
(
 W_WAREHOUSE_SK     INTEGER       NOT NULL,
 W_WAREHOUSE_ID     CHAR(16)      NOT NULL,
 W_WAREHOUSE_NAME   VARCHAR(20)       ,
 W_WAREHOUSE_SQ_FT  INTEGER         ,
 W_STREET_NUMBER    CHAR(10)        ,
 W_STREET_NAME      VARCHAR(60)       ,
 W_STREET_TYPE      CHAR(15)        ,
 W_SUITE_NUMBER     CHAR(10)        ,
 W_CITY         VARCHAR(60)       ,
 W_COUNTY       VARCHAR(30)       ,
 W_STATE        CHAR(2)         ,
 W_ZIP          CHAR(10)        ,
 W_COUNTRY      VARCHAR(20)       ,
 W_GMT_OFFSET   DECIMAL(5,2)
);
```

示例 7：事务中创建表临时表，并指定提交事务时删除该临时表数据。

```
CREATE TEMPORARY TABLE warehouse_t8
(
 W_WAREHOUSE_SK     INTEGER       NOT NULL,
 W_WAREHOUSE_ID     CHAR(16)      NOT NULL,
 W_WAREHOUSE_NAME   VARCHAR(20)       ,
 W_WAREHOUSE_SQ_FT  INTEGER         ,
 W_STREET_NUMBER    CHAR(10)        ,
 W_STREET_NAME      VARCHAR(60)       ,
 W_STREET_TYPE      CHAR(15)        ,
 W_SUITE_NUMBER     CHAR(10)        ,
 W_CITY         VARCHAR(60)       ,
 W_COUNTY       VARCHAR(30)       ,
 W_STATE        CHAR(2)         ,
```

```
 W_ZIP         CHAR(10)         ,
 W_COUNTRY     VARCHAR(20)      ,
 W_GMT_OFFSET  DECIMAL(5,2)
) ON COMMIT DELETE ROWS;
```

示例 8：创建表时，不希望因为表已存在而报错。

```
CREATE TABLE IF NOT EXISTS warehouse_t9
(
 W_WAREHOUSE_SK     INTEGER       NOT NULL,
 W_WAREHOUSE_ID     CHAR(16)      NOT NULL,
 W_WAREHOUSE_NAME   VARCHAR(20)       ,
 W_WAREHOUSE_SQ_FT  INTEGER           ,
 W_STREET_NUMBER    CHAR(10)          ,
 W_STREET_NAME      VARCHAR(60)       ,
 W_STREET_TYPE      CHAR(15)          ,
 W_SUITE_NUMBER     CHAR(10)          ,
 W_CITY         VARCHAR(60)       ,
 W_COUNTY       VARCHAR(30)       ,
 W_STATE        CHAR(2)           ,
 W_ZIP          CHAR(10)          ,
 W_COUNTRY      VARCHAR(20)       ,
 W_GMT_OFFSET   DECIMAL(5,2)
);
```

示例 9：创建带有自增列的表。

```
create table test_b
(id serial PRIMARY KEY,
name character varying(64)
);
```

示例 10：创建普通表空间。

```
CREATE TABLESPACE DS_TABLESPACE1 RELATIVE LOCATION 'tablespace/tables
pace_1';
```

示例 11：创建表时，指定表空间。

```
CREATE TABLE warehouse_t10
(
 W_WAREHOUSE_SK     INTEGER       NOT NULL,
 W_WAREHOUSE_ID     CHAR(16)      NOT NULL,
 W_WAREHOUSE_NAME     VARCHAR(20)       ,
 W_WAREHOUSE_SQ_FT    INTEGER           ,
 W_STREET_NUMBER     CHAR(10)          ,
 W_STREET_NAME      VARCHAR(60)       ,
 W_STREET_TYPE      CHAR(15)          ,
 W_SUITE_NUMBER      CHAR(10)          ,
 W_CITY         VARCHAR(60)       ,
 W_COUNTY        VARCHAR(30)       ,
 W_STATE        CHAR(2)           ,
 W_ZIP         CHAR(10)          ,
 W_COUNTRY       VARCHAR(20)       ,
 W_GMT_OFFSET       DECIMAL(5,2)
) TABLESPACE DS_TABLESPACE1;
```

示例 12：创建表时，单独指定 W_WAREHOUSE_NAME 的索引表空间。

```
CREATE TABLE warehouse_t11
(
 W_WAREHOUSE_SK     INTEGER       NOT NULL,
 W_WAREHOUSE_ID     CHAR(16)      NOT NULL,
 W_WAREHOUSE_NAME     VARCHAR(20)     UNIQUE USING INDEX TABLESPACE
DS_TABLESPACE1,
 W_WAREHOUSE_SQ_FT    INTEGER           ,
 W_STREET_NUMBER     CHAR(10)          ,
 W_STREET_NAME      VARCHAR(60)       ,
 W_STREET_TYPE      CHAR(15)          ,
 W_SUITE_NUMBER      CHAR(10)          ,
 W_CITY         VARCHAR(60)       ,
 W_COUNTY        VARCHAR(30)       ,
 W_STATE        CHAR(2)           ,
 W_ZIP         CHAR(10)          ,
```

```
 W_COUNTRY       VARCHAR(20)       ,
 W_GMT_OFFSET      DECIMAL(5,2)
);
```

示例 13：创建一个有主键约束的表。

```
CREATE TABLE warehouse_t12
(
 W_WAREHOUSE_SK     INTEGER      PRIMARY KEY,
 W_WAREHOUSE_ID     CHAR(16)      NOT NULL,
 W_WAREHOUSE_NAME     VARCHAR(20)       ,
 W_WAREHOUSE_SQ_FT    INTEGER          ,
 W_STREET_NUMBER     CHAR(10)          ,
 W_STREET_NAME      VARCHAR(60)        ,
 W_STREET_TYPE      CHAR(15)          ,
 W_SUITE_NUMBER      CHAR(10)          ,
 W_CITY         VARCHAR(60)        ,
 W_COUNTY        VARCHAR(30)        ,
 W_STATE        CHAR(2)          ,
 W_ZIP         CHAR(10)          ,
 W_COUNTRY       VARCHAR(20)       ,
 W_GMT_OFFSET      DECIMAL(5,2)
);
```

或是用下面的语法，效果完全一样。

```
CREATE TABLE  warehouse_t13
(
 W_WAREHOUSE_SK     INTEGER      NOT NULL,
 W_WAREHOUSE_ID     CHAR(16)      NOT NULL,
 W_WAREHOUSE_NAME     VARCHAR(20)       ,
 W_WAREHOUSE_SQ_FT    INTEGER          ,
 W_STREET_NUMBER     CHAR(10)          ,
 W_STREET_NAME      VARCHAR(60)        ,
 W_STREET_TYPE      CHAR(15)          ,
 W_SUITE_NUMBER      CHAR(10)          ,
 W_CITY         VARCHAR(60)        ,
```

```
 W_COUNTY        VARCHAR(30)       ,
 W_STATE         CHAR(2)           ,
 W_ZIP           CHAR(10)          ,
 W_COUNTRY       VARCHAR(20)       ,
 W_GMT_OFFSET    DECIMAL(5,2),
 PRIMARY KEY(W_WAREHOUSE_SK)
);
```

或是用下面的语法，指定约束的名称。

```
CREATE TABLE warehouse_t14
(
 W_WAREHOUSE_SK     INTEGER       NOT NULL,
 W_WAREHOUSE_ID     CHAR(16)      NOT NULL,
 W_WAREHOUSE_NAME   VARCHAR(20)       ,
 W_WAREHOUSE_SQ_FT  INTEGER           ,
 W_STREET_NUMBER    CHAR(10)          ,
 W_STREET_NAME      VARCHAR(60)       ,
 W_STREET_TYPE      CHAR(15)          ,
 W_SUITE_NUMBER     CHAR(10)          ,
 W_CITY          VARCHAR(60)       ,
 W_COUNTY        VARCHAR(30)       ,
 W_STATE         CHAR(2)           ,
 W_ZIP           CHAR(10)          ,
 W_COUNTRY       VARCHAR(20)       ,
 W_GMT_OFFSET    DECIMAL(5,2),
 CONSTRAINT W_CSTR_KEY1 PRIMARY KEY(W_WAREHOUSE_SK)
);
```

示例 14：创建一个有复合主键约束的表。

```
CREATE TABLE warehouse_t15
(
 W_WAREHOUSE_SK     INTEGER       NOT NULL,
 W_WAREHOUSE_ID     CHAR(16)      NOT NULL,
 W_WAREHOUSE_NAME   VARCHAR(20)       ,
 W_WAREHOUSE_SQ_FT  INTEGER           ,
 W_STREET_NUMBER    CHAR(10)          ,
```

```
 W_STREET_NAME      VARCHAR(60)       ,
 W_STREET_TYPE      CHAR(15)          ,
 W_SUITE_NUMBER     CHAR(10)          ,
 W_CITY         VARCHAR(60)       ,
 W_COUNTY       VARCHAR(30)       ,
 W_STATE        CHAR(2)           ,
 W_ZIP          CHAR(10)          ,
 W_COUNTRY      VARCHAR(20)       ,
 W_GMT_OFFSET       DECIMAL(5,2),
 CONSTRAINT W_CSTR_KEY2 PRIMARY KEY(W_WAREHOUSE_SK, W_WAREHOUSE
_ID)
);
```

示例 15：创建列存表。

```
CREATE TABLE warehouse_t16
(
 W_WAREHOUSE_SK     INTEGER       NOT NULL,
 W_WAREHOUSE_ID     CHAR(16)      NOT NULL,
 W_WAREHOUSE_NAME   VARCHAR(20)       ,
 W_WAREHOUSE_SQ_FT  INTEGER           ,
 W_STREET_NUMBER    CHAR(10)          ,
 W_STREET_NAME      VARCHAR(60)       ,
 W_STREET_TYPE      CHAR(15)          ,
 W_SUITE_NUMBER     CHAR(10)          ,
 W_CITY         VARCHAR(60)       ,
 W_COUNTY       VARCHAR(30)       ,
 W_STATE        CHAR(2)           ,
 W_ZIP          CHAR(10)          ,
 W_COUNTRY      VARCHAR(20)       ,
 W_GMT_OFFSET       DECIMAL(5,2)
) WITH (ORIENTATION = COLUMN);
```

示例 16：创建局部聚簇存储的列存表。

```
CREATE TABLE warehouse_t17
(
 W_WAREHOUSE_SK     INTEGER       NOT NULL,
```

```
 W_WAREHOUSE_ID      CHAR(16)       NOT NULL,
 W_WAREHOUSE_NAME    VARCHAR(20)        ,
 W_WAREHOUSE_SQ_FT   INTEGER            ,
 W_STREET_NUMBER     CHAR(10)           ,
 W_STREET_NAME       VARCHAR(60)        ,
 W_STREET_TYPE       CHAR(15)           ,
 W_SUITE_NUMBER      CHAR(10)           ,
 W_CITY          VARCHAR(60)        ,
 W_COUNTY        VARCHAR(30)        ,
 W_STATE         CHAR(2)            ,
 W_ZIP           CHAR(10)           ,
 W_COUNTRY       VARCHAR(20)        ,
 W_GMT_OFFSET        DECIMAL(5,2),
 PARTIAL CLUSTER KEY(W_WAREHOUSE_SK, W_WAREHOUSE_ID)
) WITH (ORIENTATION = COLUMN);
```

示例 17：定义一个带压缩的列存表。

```
CREATE TABLE warehouse_t18
(
 W_WAREHOUSE_SK      INTEGER       NOT NULL,
 W_WAREHOUSE_ID      CHAR(16)      NOT NULL,
 W_WAREHOUSE_NAME    VARCHAR(20)        ,
 W_WAREHOUSE_SQ_FT   INTEGER            ,
 W_STREET_NUMBER     CHAR(10)           ,
 W_STREET_NAME       VARCHAR(60)        ,
 W_STREET_TYPE       CHAR(15)           ,
 W_SUITE_NUMBER      CHAR(10)           ,
 W_CITY          VARCHAR(60)        ,
 W_COUNTY        VARCHAR(30)        ,
 W_STATE         CHAR(2)            ,
 W_ZIP           CHAR(10)           ,
 W_COUNTRY       VARCHAR(20)        ,
 W_GMT_OFFSET        DECIMAL(5,2)
) WITH (ORIENTATION = COLUMN, COMPRESSION=HIGH);
```

示例 18：定义一个检查列约束。

```
CREATE TABLE warehouse_t20
(
 W_WAREHOUSE_SK      INTEGER       PRIMARY KEY CHECK (W_WAREHOUSE_SK
> 0),
 W_WAREHOUSE_ID      CHAR(16)      NOT NULL,
 W_WAREHOUSE_NAME    VARCHAR(20)   CHECK (W_WAREHOUSE_NAME IS N
OT NULL),
 W_WAREHOUSE_SQ_FT   INTEGER         ,
 W_STREET_NUMBER     CHAR(10)        ,
 W_STREET_NAME       VARCHAR(60)       ,
 W_STREET_TYPE       CHAR(15)        ,
 W_SUITE_NUMBER      CHAR(10)        ,
 W_CITY          VARCHAR(60)       ,
 W_COUNTY        VARCHAR(30)       ,
 W_STATE         CHAR(2)         ,
 W_ZIP           CHAR(10)        ,
 W_COUNTRY       VARCHAR(20)       ,
 W_GMT_OFFSET        DECIMAL(5,2)
);

CREATE TABLE warehouse_t21
(
 W_WAREHOUSE_SK      INTEGER       PRIMARY KEY,
 W_WAREHOUSE_ID      CHAR(16)      NOT NULL,
 W_WAREHOUSE_NAME    VARCHAR(20)   CHECK (W_WAREHOUSE_NAME IS N
OT NULL),
 W_WAREHOUSE_SQ_FT   INTEGER         ,
 W_STREET_NUMBER     CHAR(10)        ,
 W_STREET_NAME       VARCHAR(60)       ,
 W_STREET_TYPE       CHAR(15)        ,
 W_SUITE_NUMBER      CHAR(10)        ,
 W_CITY          VARCHAR(60)       ,
 W_COUNTY        VARCHAR(30)       ,
 W_STATE         CHAR(2)         ,
 W_ZIP           CHAR(10)        ,
```

```
 W_COUNTRY       VARCHAR(20)       ,
 W_GMT_OFFSET      DECIMAL(5,2),
 CONSTRAINT W_CONSTR_KEY2 CHECK(W_WAREHOUSE_SK > 0 AND W_WAREH
OUSE_NAME IS NOT NULL)
);
```

示例 19：定义一个表，表中每一个行存在数据库节点中。

```
CREATE TABLE warehouse_t22
(
 W_WAREHOUSE_SK     INTEGER      NOT NULL,
 W_WAREHOUSE_ID     CHAR(16)      NOT NULL,
 W_WAREHOUSE_NAME     VARCHAR(20)       ,
 W_WAREHOUSE_SQ_FT    INTEGER          ,
 W_STREET_NUMBER     CHAR(10)         ,
 W_STREET_NAME      VARCHAR(60)        ,
 W_STREET_TYPE      CHAR(15)         ,
 W_SUITE_NUMBER      CHAR(10)         ,
 W_CITY         VARCHAR(60)        ,
 W_COUNTY        VARCHAR(30)        ,
 W_STATE        CHAR(2)         ,
 W_ZIP         CHAR(10)         ,
 W_COUNTRY       VARCHAR(20)        ,
 W_GMT_OFFSET      DECIMAL(5,2)
);
```

示例 20：使用 ALTER TABLE 修改表属性。

- 向 warehouse_t20 表中增加一个 varchar 列。

```
ALTER TABLE warehouse_t20 ADD W_GOODS_CATEGORY varchar(30);
```

- 给 warehouse_t20 表增加一个检查约束。

```
ALTER TABLE warehouse_t20 ADD CONSTRAINT W_CONSTR_KEY4 CHECK (W_ST
ATE IS NOT NULL);
```

- 在一个操作中改变两个现存字段的类型。

```
ALTER TABLE warehouse_t20
ALTER COLUMN W_GOODS_CATEGORY TYPE varchar(80),
```

```
ALTER COLUMN W_STREET_NAME TYPE varchar(100);
```

- 此语句与上面语句等效。

```
ALTER TABLE warehouse_t20 MODIFY (W_GOODS_CATEGORY varchar(30), W_ST
REET_NAME varchar(60));
```

- 给一个已存在字段添加非空约束。

```
ALTER TABLE warehouse_t20 ALTER COLUMN W_GOODS_CATEGORY SET NOT N
ULL;
```

- 移除已存在字段的非空约束。

```
ALTER TABLE warehouse_t20 ALTER COLUMN W_GOODS_CATEGORY DROP NOT
NULL;
```

- 如果列存表中还未指定局部聚簇，向在一个列存表中添加局部聚簇列。

```
ALTER TABLE warehouse_t18 ADD PARTIAL CLUSTER KEY(W_WAREHOUSE_SK);
```

- 删除一个列存表中的局部聚簇列。

```
ALTER TABLE warehouse_t18 DROP CONSTRAINT warehouse_t18_cluster;
```

- 将表移动到另一个表空间。

```
ALTER TABLE warehouse_t20 SET TABLESPACE PG_DEFAULT;
```

- 创建模式 joe。

```
CREATE SCHEMA joe;
```

- 将表移动到另一个模式中。

```
ALTER TABLE warehouse_t20 SET SCHEMA joe;
```

- 重命名已存在的表。

```
ALTER TABLE joe.warehouse_t20 RENAME TO warehouse_t23;
```

- 从 warehouse_t23 表中删除一个字段。

```
ALTER TABLE joe.warehouse_t23 DROP COLUMN W_STREET_NAME;
```

示例 21：创建虚拟列表，虚拟列作为一级分区键。

```
CREATE TABLE t_virtual
   (
     object_id number,
     object_name varchar2(100),
     created date,
     create_year int GENERATED ALWAYS AS (to_number(to_char(created,'
```

```
MM')))STORED
    )
    partition by list(create_year)
    (
    partition P1 VALUES (1),
    partition P2 VALUES (2),
    partition P3 VALUES (3),
    partition P4 VALUES (4),
    partition P5 VALUES (5)
    );
```

4.5.91.CREATE TABLE AS

功能描述

根据查询结果创建表。

CREATE TABLE AS 创建一个表并且用来自 SELECT 命令的结果填充该表。该表的字段和 SELECT 输出字段的名称及数据类型相关。不过用户可以通过明确地给出一个字段名称列表来覆盖 SELECT 输出字段的名称。

CREATE TABLE AS 对源表进行一次查询，然后将数据写入新表中，而查询视图结果会根据源表的变化而有所改变。相比之下，每次做查询的时候，视图都重新计算定义它的 SELECT 语句。

注意事项

- 分区表不能采用此方式进行创建。
- 如果在建表过程中数据库系统发生故障，系统恢复后可能无法自动清除之前已创建的、大小非 0 的磁盘文件。此种情况出现概率小，不影响数据库系统的正常运行。

语法格式

```
CREATE [ [ GLOBAL | LOCAL ] [ TEMPORARY | TEMP ] | UNLOGGED ] TABLE table_name
  [ (column_name [, ...] ) ]
  [ WITH ( {storage_parameter = value} [, ... ] ) ]
  [ ON COMMIT { PRESERVE ROWS | DELETE ROWS | DROP } ]
```

```
[ COMPRESS | NOCOMPRESS ]
[ TABLESPACE tablespace_name ]
AS query
[ WITH [ NO ] DATA ];
```

参数说明

- **UNLOGGED**

  指定表为非日志表。在非日志表中写入的数据不会被写入到预写日志中，这样就会比普通表快很多。但是，它也是不安全的，非日志表在冲突或异常关机后会被自动删截。非日志表中的内容也不会被复制到备用服务器中。在该类表中创建的索引也不会被自动记录。

  - 使用场景：非日志表不能保证数据的安全性，用户应该在确保数据已经做好备份的前提下使用，例如系统升级时进行数据的备份。
  - 故障处理：当异常关机等操作导致非日志表上的索引发生数据丢失时，用户应该对发生错误的索引进行重建。

- **GLOBAL | LOCAL**

  创建临时表时可以在 TEMP 或 TEMPORARY 前指定 GLOBAL 或 LOCAL 关键字。如果指定 GLOBAL 关键字，PanWeiDB 会创建全局临时表，否则 PanWeiDB 会创建本地临时表。

- **TEMPORARY | TEMP**

  如果指定 TEMP 或 TEMPORARY 关键字，则创建的表为临时表。临时表分为全局临时表和本地临时表两种类型。创建临时表时如果指定 GLOBAL 关键字则为全局临时表，否则为本地临时表。

  全局临时表的元数据对所有会话可见，会话结束后元数据继续存在。会话与会话之间的用户数据、索引和统计信息相互隔离，每个会话只能看到和更改自己提交的数据。全局临时表有两种模式：一种是基于会话级别的（ON COMMIT PRESERVE ROWS），当会话结束时自动清空用户数据；一种是基于事务级别的（ON COMMIT DELETE ROWS），当执行 commit 或 rollback 时自动清空用户数据。建表时如果没有指定 ON COMMIT 选项，则缺省为会话级别。与本地临时表不同，全局临时表建表时可以指定非 pg_temp_开头的 schema。

本地临时表只在当前会话可见，本会话结束后会自动删除。因此，在除当前会话连接的数据库节点故障时，仍然可以在当前会话上创建和使用临时表。由于临时表只在当前会话创建，对于涉及对临时表操作的 DDL 语句，会产生 DDL 失败的报错。因此，建议 DDL 语句中不要对临时表进行操作。TEMP 和 TEMPORARY 等价。

⚠ **须知：**

- 本地临时表通过每个会话独立的以 pg_temp 开头的 schema 来保证只对当前会话可见，因此，不建议用户在日常操作中手动删除以 pg_temp、pg_toast_temp 开头的 schema。
- 如果建表时不指定 TEMPORARY/TEMP 关键字，而指定表的 schema 为当前会话的 pg_temp_开头的 schema，则此表会被创建为临时表。
- ALTER/DROP 全局临时表和索引，如果其他会话正在使用它，禁止操作。
- 全局临时表的 DDL 只会影响当前会话的用户数据和索引。例如 truncate、reindex、analyze 只对当前会话有效。

❖ **table_name**

要创建的表名。

取值范围：字符串，要符合标识符的命名规范。

❖ **column_name**

新表中要创建的字段名。

取值范围：字符串，要符合标识符的命名规范。

❖ **WITH (storage_parameter [= value] [, ...])**

这个子句为表或索引指定一个可选的存储参数。参数的详细说明如下所示。

- FILLFACTOR
- 一个表的填充因子（fillfactor）是一个介于 10 和 100 之间的百分数。100（完全填充）是默认值。如果指定了较小的填充因子，INSERT 操作仅按照填充因子指定的百分率填充表页。每个页上的剩余空间将用于在该页上更新行，这就使得 UPDATE 有机会在同

一页上放置同一条记录的新版本，这比把新版本放置在其他页上更有效。对于一个从不更新的表将填充因子设为 100 是最佳选择，但是对于频繁更新的表，选择较小的填充因子则更加合适。该参数只对行存表有效。

➢取值范围：10~100

➢ ORIENTATION

➢取值范围：

➢COLUMN：表的数据将以列式存储。

➢ROW（缺省值）：表的数据将以行式存储。

➢ COMPRESSION

➢指定表数据的压缩级别，它决定了表数据的压缩比以及压缩时间。一般来讲，压缩级别越高，压缩比也越大，压缩时间也越长；反之亦然。实际压缩比取决于加载的表数据的分布特征。

➢取值范围：

➢列存表的有效值为 YES/NO/LOW/MIDDLE/HIGH，默认值为 LOW。

➢行存表不支持压缩。

➢ MAX_BATCHROW

➢指定了在数据加载过程中一个存储单元可以容纳记录的最大数目。该参数只对列存表有效。

➢取值范围：10000~60000

❖ **ON COMMIT { PRESERVE ROWS | DELETE ROWS | DROP }**

ON COMMIT 选项决定在事务中执行创建临时表操作，当事务提交时，此临时表的后续操作。有以下三个选项，当前仅支持 PRESERVE ROWS 和 DELETE ROWS 选项。

- PRESERVE ROWS（缺省值）：提交时不对临时表执行任何操作，临时表及其表数据保持不变。
- DELETE ROWS：提交时删除临时表中数据。
- DROP：提交时删除此临时表。只支持删除本地临时表，不支持删除全局临时表。

❖ **COMPRESS / NOCOMPRESS**

创建一个新表时，需要在创建表语句中指定关键字 COMPRESS，这样，当对该表进行批量插入时就会触发压缩特性。该特性会在页范围内扫描所有元组数据，生成字典、压缩元组数据并进行存储。指定关键字 NOCOMPRESS 则不对表进行压缩。行存表不支持压缩。该参数已废弃，列存表请使用 COMPRESSION 修改压缩等级。

缺省值：NOCOMPRESS，即不对元组数据进行压缩。

- **TABLESPACE tablespace_name**

  指定新表将要在 tablespace_name 表空间内创建。如果没有声明，将使用默认表空间。

- **AS query**

  一个 SELECT VALUES 命令。

- **[WITH [NO]DATA]**

  创建表时，是否也插入查询到的数据。默认是要数据，选择“NO”参数时，则不要数据。

示例

```
--创建一个表 tpcds.store_returns 表。
panweidb=# CREATE TABLE tpcds.store_returns
(
  W_WAREHOUSE_SK        INTEGER          NOT NULL,
  W_WAREHOUSE_ID        CHAR(16)         NOT NULL,
  sr_item_sk            VARCHAR(20)            ,
  W_WAREHOUSE_SQ_FT     INTEGER
);
--创建一个表 tpcds.store_returns_t1 并插入 tpcds.store_returns 表中 sr_item_sk 字段中大于 16 的数值。
panweidb=# CREATE TABLE tpcds.store_returns_t1 AS SELECT * FROM tpcds.store_returns WHERE sr_item_sk > '4795';

--使用 tpcds.store_returns 拷贝一个新表 tpcds.store_returns_t2。
panweidb=# CREATE TABLE tpcds.store_returns_t2 AS table tpcds.store_returns;

--删除表。
```

```
panweidb=# DROP TABLE tpcds.store_returns_t1 ;
panweidb=# DROP TABLE tpcds.store_returns_t2 ;
panweidb=# DROP TABLE store_returns;
```

相关链接

参考：SQL 语法参考->SQL 语法->CREATE TABLE、SELECT 章节。

4.5.92.CREATE TABLE PARTITION

功能描述

创建分区表。分区表是把逻辑上的一张表根据某种方案分成几张物理块进行存储，这张逻辑上的表称之为分区表，物理块称之为分区。分区表是一张逻辑表，不存储数据，数据实际是存储在分区上的。

常见的分区方案有范围分区（Range Partitioning）、间隔分区（Interval Partitioning）、哈希分区（Hash Partitioning）、列表分区（List Partitioning）、数值分区（Value Partition）等。目前行存表支持范围分区、间隔分区、哈希分区、列表分区，列存表仅支持范围分区。

范围分区是根据表的一列或者多列，将要插入表的记录分为若干个范围，这些范围在不同的分区里没有重叠。为每个范围创建一个分区，用来存储相应的数据。

范围分区的分区策略是指记录插入分区的方式。目前范围分区仅支持范围分区策略。

范围分区策略：根据分区键值将记录映射到已创建的某个分区上，如果可以映射到已创建的某一分区上，则把记录插入到对应的分区上，否则给出报错和提示信息。这是最常用的分区策略。

间隔分区是一种特殊的范围分区，相比范围分区，新增间隔值定义，当插入记录找不到匹配的分区时，可以根据间隔值自动创建分区。

间隔分区只支持基于表的一列分区，并且该列只支持 TIMESTAMP[(p)] [WITHOUT TIME ZONE]、TIMESTAMP[(p)] [WITH TIME ZONE]、DATE 数据类型。

间隔分区策略：根据分区键值将记录映射到已创建的某个分区上，如果可以映射到已创建的某一分区上，则把记录插入到对应的分区上，否则根据分区键值

和表定义信息自动创建一个分区，然后将记录插入新分区中，新创建的分区数据范围等于间隔值。

哈希分区是根据表的一列，为每个分区指定模数和余数，将要插入表的记录划分到对应的分区中，每个分区所持有的行都需要满足条件：分区键的值除以为其指定的模数将产生为其指定的余数。

哈希分区策略：根据分区键值将记录映射到已创建的某个分区上，如果可以映射到已创建的某一分区上，则把记录插入到对应的分区上，否则返回报错和提示信息。

列表分区是根据表的一列，将要插入表的记录通过每一个分区中出现的键值划分到对应的分区中，这些键值在不同的分区里没有重叠。为每组键值创建一个分区，用来存储相应的数据。

列表分区策略：根据分区键值将记录映射到已创建的某个分区上，如果可以映射到已创建的某一分区上，则把记录插入到对应的分区上，否则给出报错和提示信息。

分区可以提供若干好处：

- 某些类型的查询性能可以得到极大提升。特别是表中访问率较高的行位于一个单独分区或少数几个分区上的情况下。分区可以减少数据的搜索空间，提高数据访问效率。
- 当查询或更新一个分区的大部分记录时，连续扫描那个分区而不是访问整个表可以获得巨大的性能提升。
- 如果需要大量加载或者删除的记录位于单独的分区上，则可以通过直接读取或删除那个分区以获得巨大的性能提升，同时还可以避免由于大量 DELETE 导致的 VACUUM 超载（仅范围分区）。

注意事项

- 唯一约束和主键约束的约束键包含所有分区键将为约束创建 LOCAL 索引，否则创建 GLOBAL 索引。
- 目前哈希分区和列表分区仅支持单列构建分区键，暂不支持多列构建分区键。
- 只需要有间隔分区表的 INSERT 权限，往该表 INSERT 数据时就可以自动创建分区。

- 对于分区表 PARTITION FOR (values)语法，values 只能是常量。
- 对于分区表 PARTITION FOR (values)语法，values 在需要数据类型转换时，建议使用强制类型转换，以防隐式类型转换结果与预期不符。
- 分区数最大值为 1048575 个，一般情况下业务不可能创建这么多分区，这样会导致内存不足。应参照参数 local_syscache_threshold 的值合理创建分区，分区表使用内存大致为（分区数 * 3 / 1024）MB。理论上分区占用内存不允许大于 local_syscache_threshold 的值，同时还需要预留部分空间以供其他功能使用。
- 指定分区语句目前不能走全局索引扫描。
- 当分区数太多导致内存不足时，会间接导致性能急剧下降。
- List 分区是按分区数组的第一个元素排序的。

语法格式

```
CREATE TABLE [ IF NOT EXISTS ] partition_table_name
( [
 { column_name data_type [ COLLATE collation ] [ column_constraint [ ... ] ]
   | table_constraint
   | LIKE source_table [ like_option [...] ] }
   [, ... ]
] )
[ WITH ( {storage_parameter = value} [, ... ] ) ]
[ COMPRESS | NOCOMPRESS ]
[ TABLESPACE tablespace_name ]
[ TO { GROUP groupname | NODE ( nodename [, ... ] ) } ]
{ PARTITION BY {
    { VALUES ( column_name [, ... ] ) } |
```

```
        {RANGE [ COLUMNS ] (partition_key) [ INTERVAL ('interval_expr') [ STORE IN
( tablespace_name [, ...] ) ] ] [ PARTITIONS integer ] ( partition_less_than_item [, ... ] )}
|

        {RANGE [ COLUMNS ] (partition_key) [ INTERVAL ('interval_expr') [ STORE IN
( tablespace_name [, ...] ) ] ] [ PARTITIONS integer ] ( partition_start_end_item [, ... ] )}
|

        {{{LIST [ COLUMNS ]} | HASH | KEY} (partition_key) [ PARTITIONS integer ]
(PARTITION partition_name [ VALUES [ IN ] (list_values_clause) ] opt_table_space ) }

        | { COLUMN ( partition_name ( column_name [, ... ] ) [, ... ] ) }

        | { SYSTEM }

      } | { { RANGE | LIST | HASH } ( column_name ) }

    }

    [ SUBPARTITION BY [ RANGE | LIST | HASH ] ( column_name [, ... ] ) ]

    [ { SUBPARTITION TEMPLATE ( subpartition_desc [, ... ] ) } |
hash_subpartitions_by_quantity ]

    table_partitioning_clauses

    [ { ENABLE | DISABLE } ROW MOVEMENT ];
```

❖ 列约束 column_constraint：

```
    [ CONSTRAINT constraint_name ]

    { NOT NULL |

      NULL |

      CHECK ( expression ) |

      DEFAULT default_e xpr |

      GENERATED ALWAYS AS ( generation_expr ) STORED |

      UNIQUE index_parameters |
```

```
    PRIMARY KEY index_parameters |
    REFERENCES reftable [ ( refcolumn ) ] [ MATCH FULL | MATCH PARTIAL | MATCH SIMPLE ]
        [ ON DELETE action ] [ ON UPDATE action ] }
  [ DEFERRABLE | NOT DEFERRABLE | INITIALLY DEFERRED | INITIALLY IMMEDIATE ]
```

❖ 表约束 table_constraint：

```
  [ CONSTRAINT constraint_name ]
  { CHECK ( expression ) |
    UNIQUE ( column_name [, ... ] ) index_parameters |
    PRIMARY KEY ( column_name [, ... ] ) index_parameters |
    FOREIGN KEY ( column_name [, ... ] ) REFERENCES reftable [ ( refcolumn [, ... ] ) ]
      [ MATCH FULL | MATCH PARTIAL | MATCH SIMPLE ] [ ON DELETE action ] [ ON UPDATE action ] }
  [ DEFERRABLE | NOT DEFERRABLE | INITIALLY DEFERRED | INITIALLY IMMEDIATE ]
```

❖ 索引存储参数 index_parameters：

```
  [ WITH ( {storage_parameter = value} [, ... ] ) ]
  [ USING INDEX TABLESPACE tablespace_name ]
```

❖ Like 选项 like_option：

```
  { INCLUDING | EXCLUDING } { DEFAULTS | GENERATED | CONSTRAINTS | INDEXES | STORAGE | COMMENTS | RELOPTIONS| ALL }
```

❖ partition_less_than_item：

```
    { PARTITION | SUBPARTITION } partition_name
  VALUES LESS THAN ( { partition_value | MAXVALUE } [, ... ] )
  [ TABLESPACE tablespace_name ]
```

【须知】

该语法仅在 MySQL 兼容模式下可用，更多内容可参考 MySQL 兼容性手册下的 CREATE TABLE PARTITION。

- partition_start_end_item：

```
{ PARTITION | SUBPARTITION } partition_name
{ { START ( partition_value ) END ( partition_value ) EVERY ( interval_value ) } |
 { START ( partition_value ) END ( { partition_value | MAXVALUE } ) } |
 { START ( partition_value ) } |
 { END ( { partition_value | MAXVALUE } ) } }
[ TABLESPACE tablespace_name ]
```

- 以下兼容 PG 风格的分区声明语法仅在 PostgreSQL 兼容模式下支持，详见 PostgreSQL 兼容性手册中的 CREATE TABLE PRATITION。

```
CREATE TABLE PARTITION parent_table_name ...
PARTITION BY { { RANGE | LIST | HASH } ( column name ) }


In PG-Format Database, CREATE TABLE PRATITION OF can be:
CREATE TABLE [ IF NOT EXISTS ] partition_table_name PARTITION OF
parent_table_name FOR VALUES
{ IN ( partition_value [, ...] ) |
 FROM ( { partition_value | MINVALUE | MAXVALUE } [, ...] )
```

```
TO ( { partition_value | MINVALUE | MAXVALUE } [, ...] ) |

WITH ( MODULUS numeric_literal, REMAINDER numeric_literal ) }

[ PARTITION BY { RANGE | LIST | HASH } ( column_name ) ]
```

参数说明

- **IF NOT EXISTS**

  如果已经存在相同名称的表，不会抛出一个错误，而会发出一个通知，告知表关系已存在。

- **partition_table_name**

  分区表的名称。

  取值范围：字符串，要符合标识符的命名规范。

- **column_name**

  新表中要创建的字段名。

  取值范围：字符串，要符合标识符的命名规范。

- **data_type**

  字段的数据类型。

- **COLLATE collation**

  COLLATE 子句指定列的排序规则（该列必须是可排列的数据类型）。如果没有指定，则使用默认的排序规则。排序规则可以使用 select * from pg_collation;命令从 pg_collation 系统表中查询，默认的排序规则为查询结果中以 default 开始的行。

- **CONSTRAINT constraint_name**

列约束或表约束的名称。可选的约束子句用于声明约束，新行或者更新的行必须满足这些约束才能成功插入或更新。

定义约束有两种方法：

- 列约束：作为一个列定义的一部分，仅影响该列。
- 表约束：不和某个列绑在一起，可以作用于多个列。

❖ **LIKE source_table [like_option ...]**

LIKE 子句声明一个表，新表自动从这个表里面继承所有字段名及其数据类型和非空约束。

和 INHERITS 不同，新表与原来的表之间在创建动作完毕之后是完全无关的。在源表做的任何修改都不会传播到新表中，并且也不可能在扫描源表的时候包含新表的数据。

- 字段缺省表达式只有在声明了 INCLUDING DEFAULTS 之后才会包含进来。缺省是不包含缺省表达式的，即新表中所有字段的缺省值都是 NULL。
- 如果指定了 INCLUDING GENERATED，则源表列的生成表达式会复制到新表中。默认不复制生成表达式。
- 非空约束将总是复制到新表中，CHECK 约束则仅在指定了 INCLUDING CONSTRAINTS 的时候才复制，而其他类型的约束则永远也不会被复制。此规则同时适用于表约束和列约束。和 INHERITS 不同，被复制的列和约束并不使用相同的名称进行融合。如果明确的指定了相同的名称或者在另外一个 LIKE 子句中，将会报错。
- 如果指定了 INCLUDING INDEXES，则源表上的索引也将在新表上创建，默认不建立索引。
- 如果指定了 INCLUDING STORAGE，则源表列的 STORAGE 设置也将被拷贝，默认情况下不包含 STORAGE 设置。
- 如果指定了 INCLUDING COMMENTS，则源表列、约束和索引的注释也会被拷贝过来。默认情况下，不拷贝源表的注释。
- 如果指定了 INCLUDING RELOPTIONS，则源表的存储参数（即源表的 WITH 子句）也将拷贝至新表。默认情况下，不拷贝源表的存储参数。
- INCLUDING ALL 包含了 INCLUDING DEFAULTS、INCLUDING CONSTRAINTS、INCLUDING INDEXES、INCLUDING STORAGE、INCLUDING COMMENTS、INCLUDING PARTITION 和 INCLUDING RELOPTIONS 的内容。

❖ **WITH (storage_parameter [= value] [, ...])**

这个子句为表或索引指定一个可选的存储参数。参数的详细描述如下所示：

➢ FILLFACTOR

一个表的填充因子（fillfactor）是一个介于 10 和 100 之间的百分数。100（完全填充）是默认值。如果指定了较小的填充因子，INSERT 操作仅按照填充因子指定的百分率填充表页。每个页上的剩余空间将用于在该页上更新行，这就使得 UPDATE 有机会在同一页上放置同一条记录的新版本，这比把新版本放置在其他页上更有效。对于一个从不更新的表将填充因子设为 100 是最佳选择，但是对于频繁更新的表，选择较小的填充因子则更加合适。该参数对于列存表没有意义。

取值范围：10~100

➢ ORIENTATION

决定了表的数据的存储方式。

取值范围：

●COLUMN：表的数据将以列式存储。

●ROW（缺省值）：表的数据将以行式存储。

【须知】

orientation 不支持修改。

➢ STORAGE_TYPE

指定存储引擎类型，该参数设置成功后就不再支持修改。

取值范围：

●USTORE，表示表支持 Inplace-Update 存储引擎。特别需要注意，使用 USTORE 表，必须要开启 track_counts 和 track_activities 参数，否则会引起空间膨胀。

●ASTORE，表示表支持 Append-Only 存储引擎。

默认值：不指定表时，默认是 Append-Only 存储。

- COMPRESSION

列存表的有效值为 LOW/MIDDLE/HIGH/YES/NO，压缩级别依次升高，默认值为 LOW。

【须知】
行存表不支持压缩。

- COMPRESSTYPE

行存表参数，设置行存表压缩算法。1 代表 pglz 算法（不推荐使用），2 代表 zstd 算法，默认不压缩。该参数生效后不允许修改。（仅支持 ASTORE 下的普通表和分区表）

取值范围：0~2，默认值为 0。

- COMPRESS_LEVEL

行存表参数，设置行存表压缩算法等级，仅当 COMPRESSTYPE 为 2 时生效。压缩等级越高，表的压缩效果越好，表的访问速度越慢。该参数允许修改，修改后影响变更数据、新增数据的压缩等级。（仅支持 ASTORE 下的普通表和分区表）

取值范围：-31~31，默认值为 0。

- COMPRESS_CHUNK_SIZE

行存表参数，设置行存表压缩 chunk 块大小。chunk 数据块越小，预期能达到的压缩效果越好，同时数据越离散，影响表的访问速度。该参数生效后不允许修改。（仅支持 ASTORE 下的普通表和分区表）

取值范围：与页面大小有关。在页面大小为 8k 场景，取值范围为：512、1024、2048、4096。

默认值：4096

- COMPRESS_PREALLOC_CHUNKS

行存表参数，设置行存表压缩 chunk 块预分配数量。预分配数量越大，表的压缩率相对越差，离散度越小，访问性能越好。该参数允许修改，修改后影响变更数据、新增数据的预分配数量。（仅支持 ASTORE 下的普通表和分区表）

取值范围：0~7，默认值为 0。

- 当 COMPRESS_CHUNK_SIZE 为 512 和 1024 时，支持预分配设置最大为 7。
- 当 COMPRESS_CHUNK_SIZE 为 2048 时，支持预分配设置最大为 3。
- 当 COMPRESS_CHUNK_SIZE 为 4096 时，支持预分配设置最大为 1。

➢ COMPRESS_BYTE_CONVERT

行存表参数，设置行存表压缩字节转换预处理。在一些场景下可以提升压缩效果，同时会导致一定性能劣化。该参数允许修改，修改后决定变更数据、新增数据是否进行字节转换预处理。当 COMPRESS_DIFF_CONVERT 为真时，该值不允许修改为假。

取值范围：布尔值，默认关闭。

➢ COMPRESS_DIFF_CONVERT

行存表参数，设置行存表压缩字节差分预处理。只能与 compress_byte_convert 一起使用。在一些场景下可以提升压缩效果，同时会导致一定性能劣化。该参数允许修改，修改后决定变更数据、新增数据是否进行字节差分预处理。

取值范围：布尔值，默认关闭。

➢ MAX_BATCHROW

指定了在数据加载过程中一个存储单元可以容纳记录的最大数目。该参数只对列存表有效。

取值范围：10000~60000，默认 60000。

➢ PARTIAL_CLUSTER_ROWS

指定了在数据加载过程中进行将局部聚簇存储的记录数目。该参数只对列存表有效。

取值范围：大于等于 MAX_BATCHROW，建议取值为 MAX_BATCHROW 的整数倍数。

- DELTAROW_THRESHOLD

  预留参数。该参数只对列存表有效。

  取值范围：0～9999

- segment

  使用段页式的方式存储。本参数仅支持行存表。不支持列存表、临时表、unlog 表。不支持 ustore 存储引擎。

  取值范围：on/off

  默认值：off

- **COMPRESS / NOCOMPRESS**

  创建一个新表时，需要在创建表语句中指定关键字 COMPRESS，这样，当对该表进行批量插入时就会触发压缩特性。该特性会在页范围内扫描所有元组数据，生成字典、压缩元组数据并进行存储。指定关键字 NOCOMPRESS 则不对表进行压缩。行存表不支持压缩。

  缺省值为 NOCOMPRESS，即不对元组数据进行压缩。

- **TABLESPACE tablespace_name**

  指定新表将要在 tablespace_name 表空间内创建。如果没有声明，将使用默认表空间。

- **PARTITION BY RANGE(partition_key)**

  创建范围分区。partition_key 为分区键的名称。

  1、对于从句是 VALUES LESS THAN 的语法格式：

【须知】

orientation 不支持修改。

对于从句是 VALUE LESS THAN 的语法格式，范围分区策略的分区键最多支持 16 列。

该情形下，分区键支持的数据类型为：SMALLINT、INTEGER、BIGINT、DECIMAL、NUMERIC、REAL、DOUBLE PRECISION、CHARACTER VARYING(n)、VARCHAR(n)、CHARACTER(n)、CHAR(n)、CHARACTER、CHAR、TEXT、NVARCHAR、NVARCHAR2、NAME、TIMESTAMP[(p)] [WITHOUT TIME ZONE]、TIMESTAMP[(p)] [WITH TIME ZONE]、DATE。

2、对于从句是 START END 的语法格式：

【须知】

对于从句是 START END 的语法格式，范围分区策略的分区键仅支持 1 列。

该情形下，分区键支持的数据类型为：SMALLINT、INTEGER、BIGINT、DECIMAL、NUMERIC、REAL、DOUBLE PRECISION、TIMESTAMP[(p)] [WITHOUT TIME ZONE]、TIMESTAMP[(p)] [WITH TIME ZONE]、DATE。

3、对于指定了 INTERVAL 子句的语法格式：

【须知】

对于指定了 INTERVAL 子句的语法格式，范围分区策略的分区键仅支持 1 列。

该情形下，分区键支持的数据类型为：TIMESTAMP[(p)] [WITHOUT TIME ZONE]、TIMESTAMP[(p)] [WITH TIME ZONE]、DATE。

- **PARTITION partition_name VALUES LESS THAN ({ partition_value | MAXVALUE })**

  指定各分区的信息。partition_name 为范围分区的名称。partition_value 为范围分区的上边界，取值依赖于 partition_key 的类型。MAXVALUE 表示分区的上边界，它通常用于设置最后一个范围分区的上边界。

【须知】

每个分区都需要指定一个上边界。

分区上边界的类型应当和分区键的类型一致。

分区列表是按照分区上边界升序排列的，值较小的分区位于值较大的分区之前。

- **PARTITION partition_name {START (partition_value) END (partition_value) EVERY (interval_value)} | {START (partition_value) END (partition_value|MAXVALUE)} | {START(partition_value)} | {END (partition_value | MAXVALUE)**}**

  指定各分区的信息，各参数意义如下：

  - partition_name：范围分区的名称或名称前缀，除以下情形外（假定其中的 partition_name 是 p1），均为分区的名称。
  - 若该定义是 START+END+EVERY 从句，则语义上定义的分区的名称依次为 p1_1, p1_2, ...。例如对于定义“PARTITION p1 START(1) END(4) EVERY(1)”，则生成的分区是：[1, 2), [2, 3) 和 [3, 4)，名称依次为 p1_1, p1_2 和 p1_3，即此处的 p1 是名称前缀。
  - 若该定义是第一个分区定义，且该定义有 START 值，则范围（MINVALUE, START）将自动作为第一个实际分区，其名称为 p1_0，然后该定义语义描述的分区名称依次为 p1_1, p1_2, ...。例如对于完整定义“PARTITION p1 START(1), PARTITION p2 START(2)”，则生成的分区是：(MINVALUE, 1), [1, 2) 和 [2, MAXVALUE)，其名称依次为 p1_0, p1_1 和 p2，即此处 p1 是名称前缀，p2 是分区名称。这里 MINVALUE 表示最小值。
  - partition_value：范围分区的端点值（起始或终点），取值依赖于 partition_key 的类型，不可是 MAXVALUE。
  - interval_value：对[START，END) 表示的范围进行切分，interval_value 是指定切分后每个分区的宽度，不可是 MAXVALUE；如果（END-START）值不能整除以 EVERY 值，则仅最后一个分区的宽度小于 EVERY 值。
  - MAXVALUE：表示最大值，它通常用于设置最后一个范围分区的上边界。

【须知】

1、在创建分区表若第一个分区定义含 START 值，则范围（MINVALUE，START）将自动作为实际的第一个分区。

2、START END 语法需要遵循以下限制：

每个 partition_start_end_item 中的 START 值（如果有的话，下同）必须小于其 END 值。

相邻的两个 partition_start_end_item，第一个的 END 值必须等于第二个的 START 值；

每个 partition_start_end_item 中的 EVERY 值必须是正向递增的，且必须小于（END-START）值；

每个分区包含起始值，不包含终点值，即形如：[起始值，终点值)，起始值是 MINVALUE 时则不包含；

一个 partition_start_end_item 创建的每个分区所属的 TABLESPACE 一样；

partition_name 作为分区名称前缀时，其长度不要超过 57 字节，超过时自动截断；

在创建、修改分区表时请注意分区表的分区总数不可超过最大限制（1048575）；

3、 在创建分区表时 START END 与 LESS THAN 语法不可混合使用。

4、 即使创建分区表时使用 START END 语法，备份（pw_dump）出的 SQL 语句也是 VALUES LESS THAN 语法格式。

❖ **INTERVAL ('interval_expr') [STORE IN (tablespace_name [, ...])]**

间隔分区定义信息。

- interval_expr：自动创建分区的间隔，例如：1 day、1 month。
- STORE IN (tablespace_name [, ...])：指定存放自动创建分区的表空间列表，如果有指定，则自动创建的分区从表空间列表中循环选择使用，否则使用分区表默认的表空间。

【须知】

列存表不支持间隔分区。

❖ **PARTITION BY LIST(partition_key)**

创建列表分区。partition_key 为分区键的名称。

- 对于 partition_key，列表分区策略的分区键仅支持 16 列。
- 对于从句是 VALUES (list_values_clause)的语法格式，list_values_clause 中包含了对应分区存在的键值，每个分区的键值数量不超过 127 个。

分区键支持的数据类型为：INT1、INT2、INT4、INT8、NUMERIC、VARCHAR(n)、CHAR、BPCHAR、NVARCHAR、NVARCHAR2、TIMESTAMP[(p)] [WITHOUT TIME ZONE]、TIMESTAMP[(p)] [WITH TIME ZONE]、DATE。分区个数不能超过 1048575 个。

❖ **PARTITION BY HASH(partition_key)**

创建哈希分区。partition_key 为分区键的名称。

对于 partition_key，哈希分区策略的分区键仅支持 1 列。

分区键支持的数据类型为：INT1、INT2、INT4、INT8、NUMERIC、VARCHAR(n)、CHAR、BPCHAR、TEXT、NVARCHAR、NVARCHAR2、TIMESTAMP[(p)] [WITHOUT TIME ZONE]、TIMESTAMP[(p)] [WITH TIME ZONE]、DATE。分区个数不能超过 1048575 个。

❖ **{ ENABLE | DISABLE } ROW MOVEMENT**

行迁移开关。

如果进行 UPDATE 操作时，更新了元组在分区键上的值，造成了该元组所在分区发生变化，就会根据该开关给出报错信息，或者进行元组在分区间的转移。

取值范围：

- ENABLE（缺省值）：行迁移开关打开。
- DISABLE：行迁移开关关闭。

【须知】

列表/哈希分区表暂不支持 ROW MOVEMENT。

❖ **NOT NULL**

字段值不允许为 NULL。ENABLE 用于语法兼容，可省略。

❖ **NULL**

字段值允许 NULL，这是缺省。

这个子句只是为和非标准 SQL 数据库兼容。不建议使用。

❖ **CHECK (condition) [NO INHERIT]**

CHECK 约束声明一个布尔表达式，每次要插入的新行或者要更新的行的新值必须使表达式结果为真或未知才能成功，否则会抛出一个异常并且不会修改数据库。

声明为字段约束的检查约束应该只引用该字段的数值，而在表约束里出现的表达式可以引用多个字段。

用 NO INHERIT 标记的约束将不会传递到子表中去。

ENABLE 用于语法兼容，可省略。

❖ **DEFAULT default_expr**

DEFAULT 子句给字段指定缺省值。该数值可以是任何不含变量的表达式（不允许使用子查询和对本表中的其他字段的交叉引用）。缺省表达式的数据类型必须和字段类型匹配。

缺省表达式将被用于任何未声明该字段数值的插入操作。如果没有指定缺省值则缺省值为 NULL。

❖ **GENERATED ALWAYS AS (generation_expr) STORED**

该子句将字段创建为生成列，生成列的值在写入（插入或更新）数据时由 generation_expr 计算得到，STORED 表示像普通列一样存储生成列的值。

【须知】

❖ STORED 关键字可省略，与不省略 STORED 语义相同。

❖ 生成表达式不能以任何方式引用当前行以外的其他数据。生成表达式不能引用其他生成列，不能引用系统列。生成表达式不能返回结果集，不

能使用子查询，不能使用聚集函数，不能使用窗口函数。生成表达式调用的函数只能是不可变（IMMUTABLE）函数。

- 不能为生成列指定默认值。
- 生成列不能作为分区键的一部分。
- 生成列不能和 ON UPDATE 约束字句的 CASCADE,SET NULL,SET DEFAULT 动作同时指定。生成列不能和 ON DELETE 约束字句的 SET NULL、SET DEFAULT 动作同时指定。
- 修改和删除生成列的方法和普通列相同。删除生成列依赖的普通列，生成列被自动删除。不能改变生成列所依赖的列的类型。
- 生成列不能被直接写入。在 INSERT 或 UPDATE 命令中, 不能为生成列指定值, 但是可以指定关键字 DEFAULT。
- 生成列的权限控制和普通列一样。
- 列存表、内存表 MOT 不支持生成列。外表中仅 postgres_fdw 支持生成列。

- **UNIQUE index_parameters**

  UNIQUE (column_name [, ...]) index_parameters

  UNIQUE 约束表示表里的一个字段或多个字段的组合必须在全表范围内唯一。

  对于唯一约束，NULL 被认为是互不相等的。

- **PRIMARY KEY index_parameters**

  PRIMARY KEY (column_name [, ...]) index_parameters

  主键约束声明表中的一个或者多个字段只能包含唯一的非 NULL 值。

  一个表只能声明一个主键。

- **DEFERRABLE | NOT DEFERRABLE**

  这两个关键字设置该约束是否可推迟。一个不可推迟的约束将在每条命令之后马上检查。可推迟约束可以推迟到事务结尾使用 SET CONSTRAINTS 命令检查。缺省是 NOT DEFERRABLE。目前，UNIQUE 约束、主键约束、外键约束可以接受这个子句。所有其他约束类型都是不可推迟的。

❖ **INITIALLY IMMEDIATE | INITIALLY DEFERRED**

如果约束是可推迟的，则这个子句声明检查约束的缺省时间。

- 如果约束是 INITIALLY IMMEDIATE（缺省），则在每条语句执行之后就立即检查它；
- 如果约束是 INITIALLY DEFERRED，则只有在事务结尾才检查它。

约束检查的时间可以用 SET CONSTRAINTS 命令修改。

❖ **USING INDEX TABLESPACE tablespace_name**

为 UNIQUE 或 PRIMARY KEY 约束相关的索引声明一个表空间。如果没有提供这个子句，这个索引将在 default_tablespace 中创建，如果 default_tablespace 为空，将使用数据库的缺省表空间。

示例

示例 1： 创建范围分区表 web_returns_p1，含有 8 个分区，分区键为 integer 类型。 分区的范围分别为：wr_returned_date_sk< 2450815、2450815<= wr_returned_date_sk< 2451179、2451179<=wr_returned_date_sk< 2451544、2451544 <= wr_returned_date_sk< 2451910、2451910 <= wr_returned_date_sk< 2452275、2452275 <= wr_returned_date_sk< 2452640、2452640 <= wr_returned_date_sk< 2453005、wr_returned_date_sk>=2453005。

```
  --创建表 web_returns。
panweidb=# CREATE TABLE web_returns
(
  W_WAREHOUSE_SK        INTEGER          NOT NULL,
  W_WAREHOUSE_ID        CHAR(16)         NOT NULL,
  W_WAREHOUSE_NAME      VARCHAR(20)            ,
  W_WAREHOUSE_SQ_FT     INTEGER                ,
```

```
    W_STREET_NUMBER         CHAR(10)                ,
    W_STREET_NAME           VARCHAR(60)             ,
    W_STREET_TYPE           CHAR(15)                ,
    W_SUITE_NUMBER          CHAR(10)                ,
    W_CITY                  VARCHAR(60)             ,
    W_COUNTY                VARCHAR(30)             ,
    W_STATE                 CHAR(2)                 ,
    W_ZIP                   CHAR(10)                ,
    W_COUNTRY               VARCHAR(20)             ,
    W_GMT_OFFSET            DECIMAL(5,2)
);
    --创建分区表  web_returns_p1。
panweidb=# CREATE TABLE web_returns_p1
(
    WR_RETURNED_DATE_SK     INTEGER                 ,
    WR_RETURNED_TIME_SK     INTEGER                 ,
    WR_ITEM_SK              INTEGER                 NOT NULL,
    WR_REFUNDED_CUSTOMER_SK INTEGER                 ,
    WR_REFUNDED_CDEMO_SK    INTEGER                 ,
    WR_REFUNDED_HDEMO_SK    INTEGER                 ,
    WR_REFUNDED_ADDR_SK     INTEGER                 ,
    WR_RETURNING_CUSTOMER_SK INTEGER                ,
    WR_RETURNING_CDEMO_SK   INTEGER                 ,
    WR_RETURNING_HDEMO_SK   INTEGER                 ,
```

```
    WR_RETURNING_ADDR_SK     INTEGER                 ,
    WR_WEB_PAGE_SK           INTEGER                 ,
    WR_REASON_SK             INTEGER                 ,
    WR_ORDER_NUMBER          BIGINT            NOT NULL,
    WR_RETURN_QUANTITY       INTEGER                 ,
    WR_RETURN_AMT            DECIMAL(7,2)            ,
    WR_RETURN_TAX            DECIMAL(7,2)            ,
    WR_RETURN_AMT_INC_TAX    DECIMAL(7,2)            ,
    WR_FEE                   DECIMAL(7,2)            ,
    WR_RETURN_SHIP_COST      DECIMAL(7,2)            ,
    WR_REFUNDED_CASH         DECIMAL(7,2)            ,
    WR_REVERSED_CHARGE       DECIMAL(7,2)            ,
    WR_ACCOUNT_CREDIT        DECIMAL(7,2)            ,
    WR_NET_LOSS              DECIMAL(7,2)
)
WITH (ORIENTATION = COLUMN,COMPRESSION=MIDDLE)
PARTITION BY RANGE(WR_RETURNED_DATE_SK)
    (
        PARTITION P1 VALUES LESS THAN(2450815),
        PARTITION P2 VALUES LESS THAN(2451179),
        PARTITION P3 VALUES LESS THAN(2451544),
        PARTITION P4 VALUES LESS THAN(2451910),
        PARTITION P5 VALUES LESS THAN(2452275),
        PARTITION P6 VALUES LESS THAN(2452640),
```

```
        PARTITION P7 VALUES LESS THAN(2453005),
        PARTITION P8 VALUES LESS THAN(MAXVALUE)
    );

    --从示例数据表导入数据。
panweidb=# INSERT INTO web_returns_p1 SELECT * FROM web_returns;

    --删除分区 P8。
panweidb=# ALTER TABLE web_returns_p1 DROP PARTITION P8;

    --增加分区 WR_RETURNED_DATE_SK 介于 2453005 和 2453105 之间。
panweidb=# ALTER TABLE web_returns_p1 ADD PARTITION P8 VALUES LESS
THAN (2453105);

    --增加分区 WR_RETURNED_DATE_SK 介于 2453105 和 MAXVALUE 之间。
panweidb=# ALTER TABLE web_returns_p1 ADD PARTITION P9 VALUES LESS
THAN (MAXVALUE);

    --删除分区 P8。
panweidb=# ALTER TABLE web_returns_p1 DROP PARTITION FOR (2453005);

    --分区 P7 重命名为 P10。
panweidb=# ALTER TABLE web_returns_p1 RENAME PARTITION P7 TO P10;

    --分区 P6 重命名为 P11。
```

```
    panweidb=# ALTER TABLE web_returns_p1 RENAME PARTITION FOR (2452639)
TO P11;


      --查询分区 P10 的行数。
    panweidb=# SELECT count(*) FROM web_returns_p1 PARTITION (P10);
     count
    --------
     0
    (1 row)


      --查询分区 P1 的行数。
    panweidb=# SELECT COUNT(*) FROM web_returns_p1 PARTITION FOR (2450815);
     count
    --------
     0
    (1 row)
```

示例 2： 创建范围分区表 web_returns_p2，含有 8 个分区，分区键类型为 integer 类型，其中第 8 个分区上边界为 MAXVALUE。

八个分区的范围分别为： wr_returned_date_sk< 2450815、2450815<= wr_returned_date_sk< 2451179、2451179<=wr_returned_date_sk< 2451544、2451544 <= wr_returned_date_sk< 2451910、2451910 <= wr_returned_date_sk< 2452275、2452275 <= wr_returned_date_sk< 2452640、2452640 <= wr_returned_date_sk< 2453005、wr_returned_date_sk>=2453005。

分区表 web_returns_p2 的表空间为 example1；分区 P1 到 P7 没有声明表空间，使用采用分区表 web_returns_p2 的表空间 example1；指定分区 P8 的表空间为 example2。

假定数据库节点的数据目录/pg_location/mount1/path1，数据库节点的数据目录/pg_location/mount2/path2，数据库节点的数据目录/pg_location/mount3/path3，数据库节点的数据目录/pg_location/mount4/path4 是 dwsadmin 用户拥有读写权限的空目录。

```
    panweidb=# CREATE TABLESPACE example1 RELATIVE LOCATION
'tablespace1/tablespace_1';
    panweidb=# CREATE TABLESPACE example2 RELATIVE LOCATION
'tablespace2/tablespace_2';
    panweidb=# CREATE TABLESPACE example3 RELATIVE LOCATION
'tablespace3/tablespace_3';
    panweidb=# CREATE TABLESPACE example4 RELATIVE LOCATION
'tablespace4/tablespace_4';


    panweidb=# CREATE TABLE web_returns_p2
    (
        WR_RETURNED_DATE_SK     INTEGER               ,
        WR_RETURNED_TIME_SK     INTEGER               ,
        WR_ITEM_SK              INTEGER               NOT NULL,
        WR_REFUNDED_CUSTOMER_SK INTEGER               ,
        WR_REFUNDED_CDEMO_SK    INTEGER               ,
        WR_REFUNDED_HDEMO_SK    INTEGER               ,
```

```
    WR_REFUNDED_ADDR_SK     INTEGER                 ,
    WR_RETURNING_CUSTOMER_SK  INTEGER               ,
    WR_RETURNING_CDEMO_SK    INTEGER                ,
    WR_RETURNING_HDEMO_SK    INTEGER                ,
    WR_RETURNING_ADDR_SK     INTEGER                ,
    WR_WEB_PAGE_SK         INTEGER                  ,
    WR_REASON_SK           INTEGER                  ,
    WR_ORDER_NUMBER         BIGINT            NOT NULL,
    WR_RETURN_QUANTITY      INTEGER                 ,
    WR_RETURN_AMT          DECIMAL(7,2)             ,
    WR_RETURN_TAX          DECIMAL(7,2)             ,
    WR_RETURN_AMT_INC_TAX    DECIMAL(7,2)           ,
    WR_FEE                 DECIMAL(7,2)             ,
    WR_RETURN_SHIP_COST      DECIMAL(7,2)           ,
    WR_REFUNDED_CASH        DECIMAL(7,2)            ,
    WR_REVERSED_CHARGE      DECIMAL(7,2)            ,
    WR_ACCOUNT_CREDIT       DECIMAL(7,2)            ,
    WR_NET_LOSS            DECIMAL(7,2)
  )
TABLESPACE example1
PARTITION BY RANGE(WR_RETURNED_DATE_SK)
  (
      PARTITION P1 VALUES LESS THAN(2450815),
      PARTITION P2 VALUES LESS THAN(2451179),
```

```
        PARTITION P3 VALUES LESS THAN(2451544),
        PARTITION P4 VALUES LESS THAN(2451910),
        PARTITION P5 VALUES LESS THAN(2452275),
        PARTITION P6 VALUES LESS THAN(2452640),
        PARTITION P7 VALUES LESS THAN(2453005),
        PARTITION P8 VALUES LESS THAN(MAXVALUE) TABLESPACE example2
    )
ENABLE ROW MOVEMENT;

    --以 like 方式创建一个分区表。
panweidb=# CREATE TABLE web_returns_p3 (LIKE web_returns_p2 INCLUDING
PARTITION);

    --修改分区 P1 的表空间为 example2。
panweidb=# ALTER TABLE web_returns_p2 MOVE PARTITION P1 TABLESPACE
example2;

    --修改分区 P2 的表空间为 example3。
panweidb=# ALTER TABLE web_returns_p2 MOVE PARTITION P2 TABLESPACE
example3;

    --以 2453010 为分割点切分 P8。
panweidb=# ALTER TABLE web_returns_p2 SPLIT PARTITION P8 AT (2453010)
INTO
    (
```

```
        PARTITION P9,
        PARTITION P10
    );

    --将 P6，P7 合并为一个分区。
panweidb=# ALTER TABLE web_returns_p2 MERGE PARTITIONS P6, P7 INTO
PARTITION P8;

    --修改分区表迁移属性。
panweidb=# ALTER TABLE web_returns_p2 DISABLE ROW MOVEMENT;
    --删除表和表空间。
panweidb=# DROP TABLE web_returns_p1;
panweidb=# DROP TABLE web_returns_p2;
panweidb=# DROP TABLE web_returns_p3;
panweidb=# DROP TABLESPACE example1;
panweidb=# DROP TABLESPACE example2;
panweidb=# DROP TABLESPACE example3;
panweidb=# DROP TABLESPACE example4;
```

示例 3：START END 语法创建、修改 Range 分区表。

假定/home/panweidb/startend_tbs1、/home/panweidb/startend_tbs2、/home/panweidb/startend_tbs3、/home/panweidb/startend_tbs4 是 panweidb 用户拥有读写权限的空目录。

```
    -- 创建表空间
```

```
    panweidb=# CREATE TABLESPACE startend_tbs1 LOCATION
'/home/panweidb/startend_tbs1';

    panweidb=# CREATE TABLESPACE startend_tbs2 LOCATION
'/home/panweidb/startend_tbs2';

    panweidb=# CREATE TABLESPACE startend_tbs3 LOCATION
'/home/panweidb/startend_tbs3';

    panweidb=# CREATE TABLESPACE startend_tbs4 LOCATION
'/home/panweidb/startend_tbs4';


      -- 创建临时 schema

    panweidb=# CREATE SCHEMA tpcds;

    panweidb=# SET CURRENT_SCHEMA TO tpcds;


      -- 创建分区表，分区键是 integer 类型

    panweidb=# CREATE TABLE startend_pt (c1 INT, c2 INT)

    TABLESPACE startend_tbs1

    PARTITION BY RANGE (c2) (

        PARTITION p1 START(1) END(1000) EVERY(200) TABLESPACE startend_tbs2,

        PARTITION p2 END(2000),

        PARTITION p3 START(2000) END(2500) TABLESPACE startend_tbs3,

        PARTITION p4 START(2500),

        PARTITION p5 START(3000) END(5000) EVERY(1000) TABLESPACE
startend_tbs4

      )

    ENABLE ROW MOVEMENT;
```

```
-- 查看分区表信息
panweidb=# SELECT relname, boundaries, spcname FROM pg_partition p JOIN pg_tablespace t ON p.reltablespace=t.oid and p.parentid='startend_pt'::regclass ORDER BY 1;
  relname   | boundaries |   spcname
------------+------------+---------------
 p1_0       | {1}        | startend_tbs2
 p1_1       | {201}      | startend_tbs2
 p1_2       | {401}      | startend_tbs2
 p1_3       | {601}      | startend_tbs2
 p1_4       | {801}      | startend_tbs2
 p1_5       | {1000}     | startend_tbs2
 p2         | {2000}     | startend_tbs1
 p3         | {2500}     | startend_tbs3
 p4         | {3000}     | startend_tbs1
 p5_1       | {4000}     | startend_tbs4
 p5_2       | {5000}     | startend_tbs4
 startend_pt |            | startend_tbs1
(12 rows)

-- 导入数据，查看分区数据量
panweidb=# INSERT INTO startend_pt VALUES (GENERATE_SERIES(0, 4999), GENERATE_SERIES(0, 4999));
panweidb=# SELECT COUNT(*) FROM startend_pt PARTITION FOR (0);
 count
```

```
-------

   1

(1 row)


panweidb=# SELECT COUNT(*) FROM startend_pt PARTITION (p3);

 count

-------

  500

(1 row)


   -- 增加分区: [5000, 5300), [5300, 5600), [5600, 5900), [5900, 6000)

panweidb=# ALTER TABLE startend_pt ADD PARTITION p6 START(5000)
END(6000) EVERY(300) TABLESPACE startend_tbs4;


   -- 增加 MAXVALUE 分区: p7

panweidb=# ALTER TABLE startend_pt ADD PARTITION p7 END(MAXVALUE);


   -- 重命名分区 p7 为 p8

panweidb=# ALTER TABLE startend_pt RENAME PARTITION p7 TO p8;


   -- 删除分区 p8

panweidb=# ALTER TABLE startend_pt DROP PARTITION p8;


   -- 重命名 5950 所在的分区为: p71
```

```
panweidb=# ALTER TABLE startend_pt RENAME PARTITION FOR(5950) TO p71;


-- 分裂 4500 所在的分区[4000, 5000)
panweidb=# ALTER TABLE startend_pt SPLIT PARTITION FOR(4500)
INTO(PARTITION q1 START(4000) END(5000) EVERY(250) TABLESPACE startend_tbs3);


-- 修改分区 p2 的表空间为 startend_tbs4
panweidb=# ALTER TABLE startend_pt MOVE PARTITION p2 TABLESPACE
startend_tbs4;


-- 查看分区情形
panweidb=# SELECT relname, boundaries, spcname FROM pg_partition p JOIN
pg_tablespace t ON p.reltablespace=t.oid and p.parentid='startend_pt'::regclass
ORDER BY 1;
  relname  | boundaries |   spcname
-----------+------------+---------------
 p1_0      | {1}        | startend_tbs2
 p1_1      | {201}      | startend_tbs2
 p1_2      | {401}      | startend_tbs2
 p1_3      | {601}      | startend_tbs2
 p1_4      | {801}      | startend_tbs2
 p1_5      | {1000}     | startend_tbs2
 p2        | {2000}     | startend_tbs4
 p3        | {2500}     | startend_tbs3
 p4        | {3000}     | startend_tbs1
```

```
 p5_1      | {4000}    | startend_tbs4
 p6_1      | {5300}    | startend_tbs4
 p6_2      | {5600}    | startend_tbs4
 p6_3      | {5900}    | startend_tbs4
 p71       | {6000}    | startend_tbs4
 q1_1      | {4250}    | startend_tbs3
 q1_2      | {4500}    | startend_tbs3
 q1_3      | {4750}    | startend_tbs3
 q1_4      | {5000}    | startend_tbs3
 startend_pt |         | startend_tbs1
(19 rows)

 -- 删除表和表空间
panweidb=# DROP SCHEMA tpcds CASCADE;
panweidb=# DROP TABLESPACE startend_tbs1;
panweidb=# DROP TABLESPACE startend_tbs2;
panweidb=# DROP TABLESPACE startend_tbs3;
panweidb=# DROP TABLESPACE startend_tbs4;
```

示例 4： 创建间隔分区表 sales，初始包含 2 个分区，分区键为 DATE 类型。分区的范围分别为：time_id < '2019-02-01 00:00:00'、'2019-02-01 00:00:00' <= time_id < '2019-02-02 00:00:00' 。

```
 --创建表 sales
panweidb=# CREATE TABLE sales
```

```
      (prod_id NUMBER(6),
       cust_id NUMBER,
       time_id DATE,
       channel_id CHAR(1),
       promo_id NUMBER(6),
       quantity_sold NUMBER(3),
       amount_sold NUMBER(10,2)
      )
      PARTITION BY RANGE (time_id)
      INTERVAL('1 day')
      ( PARTITION p1 VALUES LESS THAN ('2019-02-01 00:00:00'),
        PARTITION p2 VALUES LESS THAN ('2019-02-02 00:00:00')
      );

      -- 数据插入分区 p1
    panweidb=# INSERT INTO sales VALUES(1, 12, '2019-01-10 00:00:00', 'a', 1, 1,
1);

      -- 数据插入分区 p2
    panweidb=# INSERT INTO sales VALUES(1, 12, '2019-02-01 00:00:00', 'a', 1, 1,
1);

      -- 查看分区信息
```

```
    panweidb=# SELECT t1.relname, partstrategy, boundaries FROM pg_partition
t1, pg_class t2 WHERE t1.parentid = t2.oid AND t2.relname = 'sales' AND t1.parttype
= 'p';
     relname | partstrategy |      boundaries
    ---------+--------------+-------------------------
     p1      | r            | {"2019-02-01 00:00:00"}
     p2      | r            | {"2019-02-02 00:00:00"}
    (2 rows)


      -- 插入数据没有匹配的分区，新创建一个分区，并将数据插入该分区
      -- 新分区的范围为 '2019-02-05 00:00:00' <= time_id < '2019-02-06 00:00:00'
    panweidb=# INSERT INTO sales VALUES(1, 12, '2019-02-05 00:00:00', 'a', 1, 1,
1);


      -- 插入数据没有匹配的分区，新创建一个分区，并将数据插入该分区
      -- 新分区的范围为 '2019-02-03 00:00:00' <= time_id < '2019-02-04 00:00:00'
    panweidb=# INSERT INTO sales VALUES(1, 12, '2019-02-03 00:00:00', 'a', 1, 1,
1);


      -- 查看分区信息
    panweidb=# SELECT t1.relname, partstrategy, boundaries FROM pg_partition
t1, pg_class t2 WHERE t1.parentid = t2.oid AND t2.relname = 'sales' AND t1.parttype
= 'p';
     relname | partstrategy |      boundaries
    ---------+--------------+-------------------------
     sys_p1  | i            | {"2019-02-06 00:00:00"}
```

```
 sys_p2 | i         |{"2019-02-04 00:00:00"}
 p1     | r         |{"2019-02-01 00:00:00"}
 p2     | r         |{"2019-02-02 00:00:00"}
(4 rows)
```

示例 5：创建 LIST 分区表 test_list，初始包含 4 个分区，分区键为 INT 类型。4 个分区的范围分别为：2000、3000、4000、5000。

```
   --创建表 test_list
panweidb=# create table test_list (col1 int, col2 int)
   partition by list(col1)
   (
   partition p1 values (2000),
   partition p2 values (3000),
   partition p3 values (4000),
   partition p4 values (5000)
   );

   -- 数据插入
panweidb=# INSERT INTO test_list VALUES(2000, 2000);
INSERT 0 1
panweidb=# INSERT INTO test_list VALUES(3000, 3000);
INSERT 0 1
```

```
    -- 查看分区信息
panweidb=# SELECT t1.relname, partstrategy, boundaries FROM pg_partition t1, pg_class t2 WHERE t1.parentid = t2.oid AND t2.relname = 'test_list' AND t1.parttype = 'p';
 relname | partstrategy | boundaries
---------+--------------+------------
 p1      | l            | {2000}
 p2      | l            | {3000}
 p3      | l            | {4000}
 p4      | l            | {5000}
(4 rows)

    -- 插入数据没有匹配到分区，报错处理
panweidb=# INSERT INTO test_list VALUES(6000, 6000);
ERROR:  inserted partition key does not map to any table partition

    -- 添加分区
panweidb=# alter table test_list add partition p5 values (6000);
ALTER TABLE
panweidb=# SELECT t1.relname, partstrategy, boundaries FROM pg_partition t1, pg_class t2 WHERE t1.parentid = t2.oid AND t2.relname = 'test_list' AND t1.parttype = 'p';
 relname | partstrategy | boundaries
---------+--------------+------------
 p5      | l            | {6000}
```

```
 p4     | l          | {5000}

 p1     | l          | {2000}

 p2     | l          | {3000}

 p3     | l          | {4000}

(5 rows)

panweidb=# INSERT INTO test_list VALUES(6000, 6000);

INSERT 0 1


   -- 分区表和普通表交换数据

panweidb=# create table t1 (col1 int, col2 int);

CREATE TABLE

panweidb=# select * from test_list partition (p1);

 col1 | col2

------+------

 2000 | 2000

(1 row)

panweidb=# alter table test_list exchange partition (p1) with table t1;

ALTER TABLE

panweidb=# select * from test_list partition (p1);

 col1 | col2

------+------

(0 rows)

panweidb=# select * from t1;

 col1 | col2
```

```
------+------
 2000 | 2000
(1 row)


   -- truncate 分区
panweidb=# select * from test_list partition (p2);
 col1 | col2
------+------
 3000 | 3000
(1 row)
panweidb=# alter table test_list truncate partition p2;
ALTER TABLE
panweidb=# select * from test_list partition (p2);
 col1 | col2
------+------
(0 rows)


   -- 删除分区
panweidb=# alter table test_list drop partition p5;
ALTER TABLE
panweidb=# SELECT t1.relname, partstrategy, boundaries FROM pg_partition
t1, pg_class t2 WHERE t1.parentid = t2.oid AND t2.relname = 'test_list' AND
t1.parttype = 'p';
 relname | partstrategy | boundaries
---------+--------------+------------
```

```
 p4    | l         | {5000}
 p1    | l         | {2000}
 p2    | l         | {3000}
 p3    | l         | {4000}
(4 rows)


panweidb=# INSERT INTO test_list VALUES(6000, 6000);
ERROR:  inserted partition key does not map to any table partition


  -- 删除分区表
panweidb=# drop table test_list;
```

示例 6： 创建 HASH 分区表 test_hash，初始包含 2 个分区，分区键为 INT 类型。

```
   --创建表 test_hash
panweidb=# create table test_hash (col1 int, col2 int)
   partition by hash(col1)
   (
   partition p1,
   partition p2
   );


   -- 数据插入
panweidb=# INSERT INTO test_hash VALUES(1, 1);
```

```
INSERT 0 1

panweidb=# INSERT INTO test_hash VALUES(2, 2);

INSERT 0 1

panweidb=# INSERT INTO test_hash VALUES(3, 3);

INSERT 0 1

panweidb=# INSERT INTO test_hash VALUES(4, 4);

INSERT 0 1


   -- 查看分区信息

panweidb=# SELECT t1.relname, partstrategy, boundaries FROM pg_partition
t1, pg_class t2 WHERE t1.parentid = t2.oid AND t2.relname = 'test_hash' AND
t1.parttype = 'p';

 relname | partstrategy | boundaries

---------+--------------+------------

 p1      | h            | {0}

 p2      | h            | {1}

(2 rows)


   -- 查看数据

panweidb=# select * from test_hash partition (p1);

 col1 | col2

------+------

    3 |    3

    4 |    4

(2 rows)
```

```
panweidb=# select * from test_hash partition (p2);
 col1 | col2
------+------
    1 |    1
    2 |    2
(2 rows)

   -- 分区表和普通表交换数据
panweidb=# create table t1 (col1 int, col2 int);
CREATE TABLE
panweidb=# alter table test_hash exchange partition (p1) with table t1;
ALTER TABLE
panweidb=# select * from test_hash partition (p1);
 col1 | col2
------+------
(0 rows)
panweidb=# select * from t1;
 col1 | col2
------+------
    3 |    3
    4 |    4
(2 rows)
```

```
   -- truncate 分区
panweidb=# alter table test_hash truncate partition p2;
ALTER TABLE
panweidb=#  select * from test_hash partition (p2);
 col1 | col2
------+------
(0 rows)


   -- 删除分区表
panweidb=# drop table test_hash;
```

相关链接

ALTER TABLE PARTITION，DROP TABLE

4.5.93.CREATE TABLE SUBPARTITION

功能描述

创建二级分区表。分区表是把逻辑上的一张表根据某种方案分成几张物理块进行存储，这张逻辑上的表称之为分区表，物理块称之为分区。分区表是一张逻辑表，不存储数据，数据实际是存储在分区上的。对于二级分区表，顶层节点表和一级分区都是逻辑表，不存储数据，只有二级分区（叶子节点）存储数据。二级分区功能可参见管理员指南->数据库使用->创建和管理分区表：支持两级分区。二级分区表的分区方案是由两个一级分区的分区方案组合而来的，一级分区的分区方案详见《PanWeiDB V2.0-S3.0.1_B01 开发者指南 》的 CREATE TABLE PARTITION。

常见的二级分区表组合方案有 Range-Range 分区、Range-List 分区、Range-Hash 分区、List-Range 分区、List-List 分区、List-Hash 分区、

Hash-Range 分区、Hash-List 分区、Hash-Hash 分区等。目前二级分区仅支持行存表。

注意事项

- 二级分区表有两个分区键，每个分区键只能支持 1 列。
- 唯一约束和主键约束的约束键包含所有分区键将为约束创建 LOCAL 索引，否则创建 GLOBAL 索引。
- 二级分区表的二级分区（叶子节点）个数不能超过 1048575 个，一级分区无限制，但一级分区下面至少有一个二级分区。
- 二级分区表只支持行存，不支持列存、段页式、hashbucket。
- 不支持 upsert、merge into。
- 指定分区查询时，如 select * from tablename partition/subpartition (partitionname)，关键字 partition 和 subpartition 注意不要写错。如果写错，查询不会报错，这时查询会变为对表起别名进行查询。
- 不支持对二级分区 subpartition for (values)查询。例如：

```
select * from tablename subpartition for (values);
```

- 不支持密态数据库、账本数据库和行级访问控制。
- 对于二级分区表 PARTITION FOR (values)语法，values 只能是常量。
- 对于分区表 PARTITION/SUBPARTITION FOR (values)语法，values 在需要数据类型转换时，建议使用强制类型转换，以防隐式类型转换结果与预期不符。
- 指定分区语句目前不支持全局索引扫描。
- 当分区数太多导致内存不足时，会间接导致性能急剧下降。
- List 分区是按分区数组的第一个元素排序的。

语法格式

```
CREATE TABLE [ IF NOT EXISTS ] subpartition_table_name
(
{ column_name data_type [ COLLATE collation ] [ column_constraint [ ... ] ]
| table_constraint
| LIKE source_table [ like_option [...] ] }[, ... ]
)
```

```
[ WITH ( {storage_parameter = value} [, ... ] ) ]
[ COMPRESS | NOCOMPRESS ]
[ TABLESPACE tablespace_name ]

PARTITION BY {
      {RANGE (partition_key) [ INTERVAL ('interval_expr') [ STORE IN (tablespace_nam
e [, ... ] ) ] ] ( partition_less_than_item [, ... ] )} |
      {RANGE (partition_key) [ INTERVAL ('interval_expr') [ STORE IN (tablespace_nam
e [, ... ] ) ] ] ( partition_start_end_item [, ... ] )} |
      {LIST | HASH (partition_key) (PARTITION partition_name [VALUES (list_values_cla
use)] opt_table_space )}
   }

(
 PARTITION partition_name1 [ VALUES LESS THAN (val1) | VALUES (val1[, ...]) ] [ TABL
ESPACE tablespace ]
 (
    { SUBPARTITION subpartition_name1 [ VALUES LESS THAN (val1_1) | VALUES (val
1_1[, ...])]  [ TABLESPACE tablespace ] } [, ...]
 )[, ...]
)
[ { ENABLE | DISABLE } ROW MOVEMENT ];
```

列约束 column_constraint：

```
[ CONSTRAINT constraint_name ]
  { NOT NULL |
    NULL |
    CHECK ( expression ) |
    DEFAULT default_e xpr |
    GENERATED ALWAYS AS ( generation_expr ) STORED |
    UNIQUE index_parameters |
    PRIMARY KEY index_parameters |
    REFERENCES reftable [ ( refcolumn ) ] [ MATCH FULL | MATCH PARTIAL | MATCH SI
MPLE ]
        [ ON DELETE action ] [ ON UPDATE action ] }
```

```
[ DEFERRABLE | NOT DEFERRABLE | INITIALLY DEFERRED | INITIALLY IMMEDIATE ]
```

表约束 table_constraint：

```
[ CONSTRAINT constraint_name ]
  { CHECK ( expression ) |
    UNIQUE ( column_name [, ... ] ) index_parameters |
    PRIMARY KEY ( column_name [, ... ] ) index_parameters |
    FOREIGN KEY ( column_name [, ... ] ) REFERENCES reftable [ ( refcolumn [, ... ] ) ]
    [ MATCH FULL | MATCH PARTIAL | MATCH SIMPLE ] [ ON DELETE action ] [ ON UPDA
TE action ] }
[ DEFERRABLE | NOT DEFERRABLE | INITIALLY DEFERRED | INITIALLY IMMEDIATE ]
```

like 选项 like_option：

```
{ INCLUDING | EXCLUDING } { DEFAULTS | GENERATED | CONSTRAINTS | INDEXES | STOR
AGE | COMMENTS | RELOPTIONS| ALL }
```

索引存储参数 index_parameters：

```
[ WITH ( {storage_parameter = value} [, ... ] ) ]
[ USING INDEX TABLESPACE tablespace_name ]
```

参数说明

- **IF NOT EXISTS**

  如果已经存在相同名称的表，不会抛出一个错误，而会发出一个通知，告知表关系已存在。

- **subpartition_table_name**

  二级分区表的名称。

  取值范围：字符串，要符合标识符的命名规范。

- **column_name**

  新表中要创建的字段名。

  取值范围：字符串，要符合标识符的命名规范。

❖ **data_type**

字段的数据类型。

❖ **COLLATE collation**

COLLATE 子句指定列的排序规则（该列必须是可排列的数据类型）。如果没有指定，则使用默认的排序规则。排序规则可以使用 select * from pg_collation;命令从 pg_collation 系统表中查询，默认的排序规则为查询结果中以 default 开始的行。

❖ **CONSTRAINT constraint_name**

列约束或表约束的名称。可选的约束子句用于声明约束，新行或者更新的行必须满足这些约束才能成功插入或更新。 定义约束有两种方法：

- 列约束：作为一个列定义的一部分，仅影响该列。
- 表约束：不和某个列绑在一起，可以作用于多个列。

❖ **LIKE source_table [like_option ...]**

二级分区表暂不支持该功能。

❖ **WITH (storage_parameter [= value] [, ...])**

这个子句为表或索引指定一个可选的存储参数。参数的详细描述如下所示：

- **FILLFACTOR**

  一个表的填充因子（fillfactor）是一个介于 10 和 100 之间的百分数。100（完全填充）是默认值。如果指定了较小的填充因子，INSERT 操作仅按照填充因子指定的百分率填充表页。每个页上的剩余空间将用于在该页上更新行，这就使得 UPDATE 有机会在同一页上放置同一条记录的新版本，这比把新版本放置在其他页上更有效。对于一个从不更新的表将填充因子设为 100 是最佳选择，但是对于频繁更新的表，选择较小的填充因子则更加合适。该参数对于列存表没有意义。

  取值范围：10~100

- **ORIENTATION**

决定了表的数据的存储方式。

【说明】
orientation 不支持修改。

取值范围：

- ✧ COLUMN：表的数据将以列式存储。
- ✧ ROW（缺省值）：表的数据将以行式存储。

➢ **COMPRESSION**

列存表的有效值为 LOW/MIDDLE/HIGH/YES/NO，压缩级别依次升高，默认值为 LOW。

行存表不支持压缩。

➢ **MAX_BATCHROW**

指定了在数据加载过程中一个存储单元可以容纳记录的最大数目。该参数只对列存表有效。

取值范围：10000~60000，默认 60000。

➢ **PARTIAL_CLUSTER_ROWS**

指定了在数据加载过程中进行将局部聚簇存储的记录数目。该参数只对列存表有效。

取值范围：大于等于 MAX_BATCHROW，建议取值为 MAX_BATCHROW 的整数倍数。

➢ **DELTAROW_THRESHOLD**

预留参数。该参数只对列存表有效。

取值范围：0～9999

❖ **COMPRESS / NOCOMPRESS**

创建一个新表时，需要在创建表语句中指定关键字 COMPRESS，这样，当对该表进行批量插入时就会触发压缩特性。该特性会在页范围内扫描所有

元组数据，生成字典、压缩元组数据并进行存储。指定关键字 NOCOMPRESS 则不对表进行压缩。行存表不支持压缩。该参数已废弃，列存表请使用 COMPRESSION 修改压缩等级。

缺省值：NOCOMPRESS，即不对元组数据进行压缩。

- **TABLESPACE tablespace_name**

指定新表将要在 tablespace_name 表空间内创建。如果没有声明，将使用默认表空间。

- **PARTITION BY {RANGE | LIST | HASH} (partition_key)**

对于 partition_key，分区策略的分区键仅支持 1 列。

分区键支持的数据类型和一级分区表约束保持一致。

- **PARTITION BY RANGE(partition_key)**

创建范围分区。partition_key 为分区键的名称。

对于从句是 VALUES LESS THAN 的语法格式：

【须知】
对于从句是 VALUE LESS THAN 的语法格式，范围分区策略的分区键最多支持 4 列。

该情形下，分区键支持的数据类型为：SMALLINT、INTEGER、BIGINT、DECIMAL、NUMERIC、REAL、DOUBLE PRECISION、CHARACTER VARYING(n)、VARCHAR(n)、CHARACTER(n)、CHAR(n)、CHARACTER、CHAR、TEXT、NVARCHAR、NVARCHAR2、NAME、TIMESTAMP[(p)] [WITHOUT TIME ZONE]、TIMESTAMP[(p)] [WITH TIME ZONE]、DATE。

对于从句是 START END 的语法格式：

【须知】
对于从句是 START END 的语法格式，范围分区策略的分区键仅支持 1 列。

该情形下，分区键支持的数据类型为：SMALLINT、INTEGER、BIGINT、DECIMAL、NUMERIC、REAL、DOUBLE PRECISION、TIMESTAMP[(p)] [WITHOUT TIME ZONE]、TIMESTAMP[(p)] [WITH TIME ZONE]、DATE。

对于指定了 INTERVAL 子句的语法格式：

【须知】
对于指定了 INTERVAL 子句的语法格式，范围分区策略的分区键仅支持 1 列。

该情形下，分区键支持的数据类型为：TIMESTAMP[(p)] [WITHOUT TIME ZONE]、TIMESTAMP[(p)] [WITH TIME ZONE]、DATE。

- **PARTITION partition_name VALUES LESS THAN ({ partition_value | MAXVALUE })**

指定各分区的信息。partition_name 为范围分区的名称。partition_value 为范围分区的上边界，取值依赖于 partition_key 的类型。MAXVALUE 表示分区的上边界，它通常用于设置最后一个范围分区的上边界。

【须知】
每个分区都需要指定一个上边界。
分区上边界的类型应当和分区键的类型一致。
分区列表是按照分区上边界升序排列的，值较小的分区位于值较大的分区之前。

- **PARTITION partition_name {START (partition_value) END (partition_value) EVERY (interval_value)} | {START (partition_value) END (partition_value|MAXVALUE)} | {START(partition_value)} | {END (partition_value | MAXVALUE)}**

指定各分区的信息，各参数意义如下：

  - partition_name：范围分区的名称或名称前缀，除以下情形外（假定其中的 partition_name 是 p1），均为分区的名称。
    - 若该定义是 START+END+EVERY 从句，则语义上定义的分区的名称依次为 p1_1, p1_2, ...。例如对于定义 “PARTITION p1

START(1) END(4) EVERY(1)”，则生成的分区是：[1, 2), [2, 3) 和 [3, 4)，名称依次为 p1_1, p1_2 和 p1_3，即此处的 p1 是名称前缀。

- ✧ 若该定义是第一个分区定义，且该定义有 START 值，则范围（MINVALUE, START）将自动作为第一个实际分区，其名称为 p1_0，然后该定义语义描述的分区名称依次为 p1_1, p1_2, ...。例如对于完整定义“PARTITION p1 START(1), PARTITION p2 START(2)”，则生成的分区是：(MINVALUE, 1), [1, 2) 和 [2, MAXVALUE)，其名称依次为 p1_0, p1_1 和 p2，即此处 p1 是名称前缀，p2 是分区名称。这里 MINVALUE 表示最小值。

- ➢ partition_value：范围分区的端点值（起始或终点），取值依赖于 partition_key 的类型，不可是 MAXVALUE。
- ➢ interval_value：对[START，END) 表示的范围进行切分，interval_value 是指定切分后每个分区的宽度，不可是 MAXVALUE；如果（END-START）值不能整除以 EVERY 值，则仅最后一个分区的宽度小于 EVERY 值。
- ➢ MAXVALUE：表示最大值，它通常用于设置最后一个范围分区的上边界。

【须知】

1、在创建分区表若第一个分区定义含 START 值，则范围（MINVALUE，START）将自动作为实际的第一个分区。

2、START END 语法需要遵循以下限制：

- ✧ 每个 partition_start_end_item 中的 START 值（如果有的话，下同）必须小于其 END 值。
- ✧ 相邻的两个 partition_start_end_item，第一个的 END 值必须等于第二个的 START 值；
- ✧ 每个 partition_start_end_item 中的 EVERY 值必须是正向递增的，且必须小于（END-START）值；
- ✧ 每个分区包含起始值，不包含终点值，即形如：[起始值，终点值)，起始值是 MINVALUE 时则不包含；

✧ 一个 partition_start_end_item 创建的每个分区所属的 TABLESPACE 一样；

✧ partition_name 作为分区名称前缀时，其长度不要超过 57 字节，超过时自动截断；

✧ 在创建、修改分区表时请注意分区表的分区总数不可超过最大限制（1048575）；

3、 在创建分区表时 START END 与 LESS THAN 语法不可混合使用。

4、 即使创建分区表时使用 START END 语法，备份（pw_dump）出的 SQL 语句也是 VALUES LESS THAN 语法格式。

❖ **INTERVAL ('interval_expr') [STORE IN (tablespace_name [, ...])]**

间隔分区定义信息。

- interval_expr：自动创建分区的间隔，例如：1 day、1 month。
- STORE IN (tablespace_name [, ...])：指定存放自动创建分区的表空间列表，如果有指定，则自动创建的分区从表空间列表中循环选择使用，否则使用分区表默认的表空间。

【须知】

列存表不支持间隔分区。

❖ **PARTITION BY LIST(partition_key)**

创建列表分区。partition_key 为分区键的名称。

- 对于 partition_key，列表分区策略的分区键最大支持 16 列。
- 对于从句是 VALUES (list_values_clause)的语法格式，list_values_clause 中包含了对应分区存在的键值，每个分区的键值数量不超过 127 个。

分区键支持的数据类型为：INT1、INT2、INT4、INT8、NUMERIC、VARCHAR(n)、CHAR、BPCHAR、NVARCHAR、NVARCHAR2、TIMESTAMP[(p)] [WITHOUT TIME ZONE]、TIMESTAMP[(p)] [WITH TIME ZONE]、DATE。分区个数不能超过 1048575 个。

❖ **PARTITION BY HASH(partition_key)**

创建哈希分区。partition_key 为分区键的名称。

对于 partition_key，哈希分区策略的分区键仅支持 1 列。

分区键支持的数据类型为：INT1、INT2、INT4、INT8、NUMERIC、VARCHAR(n)、CHAR、BPCHAR、TEXT、NVARCHAR、NVARCHAR2、TIMESTAMP[(p)] [WITHOUT TIME ZONE]、TIMESTAMP[(p)] [WITH TIME ZONE]、DATE。分区个数不能超过 1048575 个。

❖ **{ ENABLE | DISABLE } ROW MOVEMENT**

行迁移开关。

如果进行 UPDATE 操作时，更新了元组在分区键上的值，造成了该元组所在分区发生变化，就会根据该开关给出报错信息，或者进行元组在分区间的转移。

取值范围：

- ENABLE（缺省值）：行迁移开关打开。
- DISABLE：行迁移开关关闭。

❖ **NOT NULL**

字段值不允许为 NULL。ENABLE 用于语法兼容，可省略。

❖ **NULL**

字段值允许 NULL，这是缺省。

这个子句只是为和非标准 SQL 数据库兼容。不建议使用。

❖ **CHECK (condition) [NO INHERIT]**

CHECK 约束声明一个布尔表达式，每次要插入的新行或者要更新的行的新值必须使表达式结果为真或未知才能成功，否则会抛出一个异常并且不会修改数据库。

声明为字段约束的检查约束应该只引用该字段的数值，而在表约束里出现的表达式可以引用多个字段。

用 NO INHERIT 标记的约束将不会传递到子表中去。

ENABLE 用于语法兼容，可省略。

❖ **DEFAULT default_expr**

DEFAULT 子句给字段指定缺省值。该数值可以是任何不含变量的表达式(不允许使用子查询和对本表中的其他字段的交叉引用)。缺省表达式的数据类型必须和字段类型匹配。

缺省表达式将被用于任何未声明该字段数值的插入操作。如果没有指定缺省值则缺省值为 NULL 。

❖ **GENERATED ALWAYS AS (generation_expr) STORED**

该子句将字段创建为生成列，生成列的值在写入（插入或更新）数据时由 generation_expr 计算得到，STORED 表示像普通列一样存储生成列的值。

【须知】

- 生成表达式不能以任何方式引用当前行以外的其他数据。生成表达式不能引用其他生成列，不能引用系统列。生成表达式不能返回结果集，不能使用子查询，不能使用聚集函数，不能使用窗口函数。生成表达式调用的函数只能是不可变（IMMUTABLE）函数。
- 不能为生成列指定默认值。
- 生成列不能作为分区键的一部分。
- 生成列不能和 ON UPDATE 约束字句的 CASCADE、SET NULL、SET DEFAULT 动作同时指定。生成列不能和 ON DELETE 约束字句的 SET NULL、SET DEFAULT 动作同时指定。
- 修改和删除生成列的方法和普通列相同。删除生成列依赖的普通列，生成列被自动删除。不能改变生成列所依赖的列的类型。
- 生成列不能被直接写入。在 INSERT 或 UPDATE 命令中, 不能为生成列指定值, 但是可以指定关键字 DEFAULT。
- 生成列的权限控制和普通列一样。
- 不能为生成列指定默认值。
- 生成列不能作为分区键的一部分。

- 生成列不能和 ON UPDATE 约束字句的 CASCADE、SET NULL、SET DEFAULT 动作同时指定。生成列不能和 ON DELETE 约束字句的 SET NULL、SET DEFAULT 动作同时指定。
- 修改和删除生成列的方法和普通列相同。删除生成列依赖的普通列，生成列被自动删除。不能改变生成列所依赖的列的类型。
- 生成列不能被直接写入。在 INSERT 或 UPDATE 命令中, 不能为生成列指定值, 但是可以指定关键字 DEFAULT。
- 生成列的权限控制和普通列一样。
- 列存表、内存表 MOT 不支持生成列。外表中仅 postgres_fdw 支持生成列。

❖ **UNIQUE index_parameters**

UNIQUE (column_name [, ...]) index_parameters

UNIQUE 约束表示表里的一个字段或多个字段的组合必须在全表范围内唯一。

对于唯一约束，NULL 被认为是互不相等的。

❖ **PRIMARY KEY index_parameters**

PRIMARY KEY (column_name [, ...]) index_parameters

主键约束声明表中的一个或者多个字段只能包含唯一的非 NULL 值。

一个表只能声明一个主键。

❖ **DEFERRABLE | NOT DEFERRABLE**

这两个关键字设置该约束是否可推迟。一个不可推迟的约束将在每条命令之后马上检查。可推迟约束可以推迟到事务结尾使用 SET CONSTRAINTS 命令检查。缺省是 NOT DEFERRABLE。目前，UNIQUE 约束、主键约束、外键约束可以接受这个子句。所有其他约束类型都是不可推迟的。

❖ **INITIALLY IMMEDIATE | INITIALLY DEFERRED**

如果约束是可推迟的，则这个子句声明检查约束的缺省时间。

- 如果约束是 INITIALLY IMMEDIATE（缺省），则在每条语句执行之后就立即检查它；

➢ 如果约束是 INITIALLY DEFERRED，则只有在事务结尾才检查它。

约束检查的时间可以用 SET CONSTRAINTS 命令修改。

❖ **USING INDEX TABLESPACE tablespace_name**

为 UNIQUE 或 PRIMARY KEY 约束相关的索引声明一个表空间。如果没有提供这个子句，这个索引将在 default_tablespace 中创建，如果 default_tablespace 为空，将使用数据库的缺省表空间。

注意事项

❖ 二级分区表有两个分区键，每个分区键只能支持 1 列。

❖ 唯一约束和主键约束的约束键包含所有分区键将为约束创建 LOCAL 索引，否则创建 GLOBAL 索引。

❖ 二级分区表的二级分区（叶子节点）个数不能超过 1048575 个，一级分区无限制，但一级分区下面至少有一个二级分区。

❖ 二级分区表只支持行存，不支持列存、段页式、hashbucket。

❖ 不支持 Upsert、Merge into。

❖ 指定分区查询时，如 select * from tablename partition/subpartition (partition_name);，关键字 partition 和 subpartition 注意不要写错。如果写错，查询不会报错，这时查询会变为对表起别名进行查询。

❖ 不支持对二级分区 subpartition for (values)查询。如 select * from tablename subpartition for (values);。

❖ 对于二级分区表 PARTITION FOR (values)语法，values 只能是常量。

❖ 对于分区表 PARTITION/SUBPARTITION FOR (values)语法，values 在需要数据类型转换时，建议使用强制类型转换，以防隐式类型转换结果与预期不符。

❖ 不支持密态数据库、账本数据库和行级访问控制。

❖ 指定分区语句目前不能走全局索引扫描。

示例

示例 1：创建 list_list 组合类型的二级分区表并进行 truncate（清空）和 drop（删除）分区的操作。

1、创建表 list_list（包含一级分区和二级分区）。

```
CREATE TABLE list_list
  (
    month_code VARCHAR2 ( 30 ) NOT NULL ,
    dept_code  VARCHAR2 ( 30 ) NOT NULL ,
    user_no    VARCHAR2 ( 30 ) NOT NULL ,
    sales_amt  int
  )
  PARTITION BY LIST (month_code) SUBPARTITION BY LIST (dept_code)
  (
   PARTITION p_201901 VALUES ( '201902' )
   (
    SUBPARTITION p_201901_a VALUES ( '1' ),
    SUBPARTITION p_201901_b VALUES ( default )
   ),
   PARTITION p_201902 VALUES ( '201903' )
   (
    SUBPARTITION p_201902_a VALUES ( '1' ),
    SUBPARTITION p_201902_b VALUES ( '2' )
   )
  );
```

2、插入测试数据。

```
insert into list_list values('201902', '1', '1', 1);
insert into list_list values('201902', '2', '1', 1);
insert into list_list values('201902', '1', '1', 1);
insert into list_list values('201903', '2', '1', 1);
insert into list_list values('201903', '1', '1', 1);
insert into list_list values('201903', '2', '1', 1);
```

3、查看表。

```
select * from list_list;
```

返回结果为：

```
 month_code | dept_code | user_no | sales_amt
------------+-----------+---------+-----------
 201902     | 1         | 1       |         1
 201902     | 1         | 1       |         1
 201902     | 2         | 1       |         1
 201903     | 1         | 1       |         1
 201903     | 2         | 1       |         1
 201903     | 2         | 1       |         1
(6 rows)
```

4、查看其中的一个分区 p_201901。

```
select * from list_list partition (p_201901);
```

返回结果为：

```
 month_code | dept_code | user_no | sales_amt
------------+-----------+---------+-----------
 201902     | 1         | 1       |         1
 201902     | 1         | 1       |         1
 201902     | 2         | 1       |         1
(3 rows)
```

5、清空其中的一个分区 p_201901。

```
alter table list_list truncate partition p_201901;
```

6、再次查看这个分区。

```
select * from list_list partition (p_201901);
```

返回结果为空，表示该分区已经被清空。

```
 month_code | dept_code | user_no | sales_amt
------------+-----------+---------+-----------
(0 rows)
```

7、删除一级分区 p_201901。

```
alter table list_list drop partition p_201901;
```

8、查看表 list_list。

```
select * from list_list;
```

返回结果如下，分区 p_201901 已被删除。

```
 month_code | dept_code | user_no | sales_amt
------------+-----------+---------+-----------
 201903     | 1         | 1       |         1
 201903     | 2         | 1       |         1
 201903     | 2         | 1       |         1
(3 rows)
```

示例 2：创建一个 list_range 分区组合的分区表，并给分区表新增一级分区和二级分区。

1、创建表 list_range。（一级分区为 list 分区，二级分区为 range 分区。）

```
create table list_range
(    id number not null,
     partition_key   int,
     subpartition_key   int,
     col2   varchar2(10)
)
partition by list(partition_key)
subpartition by range(subpartition_key)
(
   partition t_partition_01 values (100)
   (subpartition sub_1_1 values less than (10),
    subpartition sub_1_2 values less than (20)
    ),
   partition t_partition_02 values (200)
   (subpartition sub_2_1 values less than (10),
    subpartition sub_2_2 values less than (20)
```

```
    )
);
```

2、新增分区。

```
alter table list_range add partition t_partition_03 values (300)
( subpartition sub_3_1 values less than (10),
 subpartition sub_3_2 values less than (20)
);
```

3、在系统表 pg_partition 中查询分区信息。

```
\x  --启用列式方式显示结果
select * from pg_partition where relname='t_partition_03';
```

返回结果如下，查看到其中 parentid 为 19029。

```
-[ RECORD 1 ]------+----------------------------------------------------------------
relname          | list_range
parttype         | r
parentid         | 19029
rangenum         | 0
intervalnum      | 0
partstrategy     | l
relfilenode      | 0
reltablespace    | 0
relpages         | 0
reltuples        | 0
relallvisible    | 0
reltoastrelid    | 0
reltoastidxid    | 0
indextblid       | 0
indisusable      | t
reldeltarelid    | 0
reldeltaidx      | 0
relcudescrelid   | 0
relcudescidx     | 0
relfrozenxid     | 0
```

```
intspnum         |
partkey          | 2
intervaltablespace |
interval         |
boundaries       |
transit          |
reloptions       | {orientation=row,compression=no,fillfactor=80,wait_clean_gpi=n}
relfrozenxid64   | 0
relminmxid       | 0
```

4、根据上一步查询到的 parentid，查询对应分区信息。

```
select * from pg_partition where parentid='19029';
```

返回结果如下，一级分区和二级分区添加成功。

```
-[ RECORD 1 ]------+-----------------------------------------------------------------
relname          | t_partition_03
parttype         | p
parentid         | 19029
rangenum         | 0
intervalnum      | 0
partstrategy     | l
relfilenode      | 0
reltablespace    | 0
relpages         | 0
reltuples        | 0
relallvisible    | 0
reltoastrelid    | 0
reltoastidxid    | 0
indextblid       | 0
indisusable      | t
reldeltarelid    | 0
reldeltaidx      | 0
relcudescrelid   | 0
relcudescidx     | 0
relfrozenxid     | 18570
```

```
intspnum          |
partkey           | 3
intervaltablespace |
interval          |
boundaries        | {300}
transit           |
reloptions        | {orientation=row,compression=no,fillfactor=80}
relfrozenxid64    | 18570
relminmxid        | 2
-[ RECORD 2 ]------+-----------------------------------------------------------------
relname           | list_range
parttype          | r
parentid          | 19029
rangenum          | 0
intervalnum       | 0
partstrategy      | l
relfilenode       | 0
reltablespace     | 0
relpages          | 0
reltuples         | 0
relallvisible     | 0
reltoastrelid     | 0
reltoastidxid     | 0
indextblid        | 0
indisusable       | t
reldeltarelid     | 0
reldeltaidx       | 0
relcudescrelid    | 0
relcudescidx      | 0
relfrozenxid      | 0
intspnum          |
partkey           | 2
intervaltablespace |
interval          |
boundaries        |
```

```
transit          |
reloptions       | {orientation=row,compression=no,fillfactor=80,wait_clean_gpi=n}
relfrozenxid64   | 0
relminmxid       | 0
-[ RECORD 3 ]------+-------------------------------------------------------------
relname          | t_partition_01
parttype         | p
parentid         | 19029
rangenum         | 0
intervalnum      | 0
partstrategy     | l
relfilenode      | 0
reltablespace    | 0
relpages         | 0
reltuples        | 0
relallvisible    | 0
reltoastrelid    | 0
reltoastidxid    | 0
indextblid       | 0
indisusable      | t
reldeltarelid    | 0
reldeltaidx      | 0
relcudescrelid   | 0
relcudescidx     | 0
relfrozenxid     | 18566
intspnum         |
partkey          | 3
intervaltablespace |
interval         |
boundaries       | {100}
transit          |
reloptions       | {orientation=row,compression=no,fillfactor=80}
relfrozenxid64   | 18566
relminmxid       | 2
-[ RECORD 4 ]------+-------------------------------------------------------------
```

```
relname          | t_partition_02
parttype         | p
parentid         | 19029
rangenum         | 0
intervalnum      | 0
partstrategy     | l
relfilenode      | 0
reltablespace    | 0
relpages         | 0
reltuples        | 0
relallvisible    | 0
reltoastrelid    | 0
reltoastidxid    | 0
indextblid       | 0
indisusable      | t
reldeltarelid    | 0
reldeltaidx      | 0
relcudescrelid   | 0
relcudescidx     | 0
relfrozenxid     | 18566
intspnum         |
partkey          | 3
intervaltablespace |
interval         |
boundaries       | {200}
transit          |
reloptions       | {orientation=row,compression=no,fillfactor=80}
relfrozenxid64   | 18566
relminmxid       | 2
```

示例 3：对二级分区表进行 split（分区切割）操作。

1、创建分区表。

```
CREATE TABLE list_list
(
  month_code VARCHAR2 ( 30 ) NOT NULL ,
```

```
  dept_code  VARCHAR2 ( 30 ) NOT NULL ,
  user_no    VARCHAR2 ( 30 ) NOT NULL ,
  sales_amt  int
)
PARTITION BY LIST (month_code) SUBPARTITION BY LIST (dept_code)
(
 PARTITION p_201901 VALUES ( '201902' )
 (
  SUBPARTITION p_201901_a VALUES ( '1' ),
  SUBPARTITION p_201901_b VALUES ( default )
 ),
 PARTITION p_201902 VALUES ( '201903' )
 (
  SUBPARTITION p_201902_a VALUES ( '1' ),
  SUBPARTITION p_201902_b VALUES ( default )
 )
);
```

2、插入测试数据。

```
insert into list_list values('201902', '1', '1', 1);
insert into list_list values('201902', '2', '1', 1);
insert into list_list values('201902', '1', '1', 1);
insert into list_list values('201903', '2', '1', 1);
insert into list_list values('201903', '1', '1', 1);
insert into list_list values('201903', '2', '1', 1);
```

3、查看表 list_list。

```
select * from list_list;
```

返回结果如下：

```
 month_code | dept_code | user_no | sales_amt
------------+-----------+---------+-----------
 201903     | 2         | 1       |         1
 201903     | 2         | 1       |         1
 201903     | 1         | 1       |         1
```

```
201902   | 2      | 1    |      1
201902   | 1      | 1    |      1
201902   | 1      | 1    |      1
(6 rows)
```

4、查看二级分区 p_201901_a。

```
select * from list_list subpartition (p_201901_a);
```

返回结果为：

```
 month_code | dept_code | user_no | sales_amt
------------+-----------+---------+-----------
 201902   | 1      | 1    |      1
 201902   | 1      | 1    |      1
(2 rows)
```

5、查看二级分区 p_201901_b。

```
select * from list_list subpartition (p_201901_b);
```

返回结果为：

```
 month_code | dept_code | user_no | sales_amt
------------+-----------+---------+-----------
 201902   | 2      | 1    |      1
(1 row)
```

6、将分区 p_201901_b 分割为 p_201901_b 和 p_201901_c。

```
alter table list_list split subpartition p_201901_b values (2) into
(
 subpartition p_201901_b,
 subpartition p_201901_c
);
```

7、查看分区 p_201901_a、p_201901_b、p_201901_c。

```
select * from list_list subpartition (p_201901_a);
select * from list_list subpartition (p_201901_b);
select * from list_list subpartition (p_201901_c);
```

返回结果为：

```
 month_code | dept_code | user_no | sales_amt
------------+-----------+---------+-----------
 201902     | 1         | 1       |         1
 201902     | 1         | 1       |         1
(2 rows)

 month_code | dept_code | user_no | sales_amt
------------+-----------+---------+-----------
 201902     | 2         | 1       |         1
(1 row)

 month_code | dept_code | user_no | sales_amt
------------+-----------+---------+-----------
(0 rows)
```

4.5.94.CREATE TABLESPACE

功能描述

在数据库中创建一个新的表空间。

注意事项

- 系统管理员或者继承了内置角色 gs_roles_tablespace 权限的用户可以创建表空间。
- 不允许在一个事务块内部执行 CREATE TABLESPACE。
- 执行 CREATE TABLESPACE 失败，如果内部创建目录（文件）操作成功了就会产生残留的目录（文件），重新创建时需要用户手动清理表空间指定的目录下残留的内容。如果在创建过程中涉及到数据目录下的表空间软连接残留，需要先将软连接的残留文件删除，再重新执行 OM 相关操作。
- CREATE TABLESPACE 不支持两阶段事务，如果部分节点执行失败，不支持回滚。
- 创建表空间前的准备工作参考下述参数说明。

- 在 HCS 等场景下一般不建议用户使用自定义的表空间。原因：用户自定义表空间通常配合主存（即默认表空间所在的存储设备，如磁盘）以外的其他存储介质使用，以隔离不同业务可以使用的 IO 资源，而在 HCS 等场景下，存储设备都是采用标准化的配置，无其他可用的存储介质，自定义表空间使用不当不利于系统长稳运行以及影响整体性能，因此建议使用默认表空间即可。

语法格式

```
CREATE TABLESPACE tablespace_name
    [ OWNER user_name ] [ RELATIVE ] LOCATION 'directory' [ MAXSIZE 'space_size' ]
    [with_option_clause] [ THRESHOLD 'space_percentile' ];
```

其中普通表空间的 with_option_clause 为：

```
WITH ( filesystem= { 'systemtype'| "systemtype" | systemtype }
[ { , address = { 'ip:port [ , ... ]' | "ip:port [ , ... ]"} } ]
, cfgpath = { 'path'| "path" } ,storepath = { 'rootpath'| "rootpath"}
[{, random_page_cost = { 'value'| "value" | value }}]
[{,seq_page_cost = { 'value'| "value" | value }}])
```

参数说明

- **tablespace_name**

  要创建的表空间名称。

  表空间名称不能和 PanWeiDB 中的其他表空间重名，且名称不能以“pg”开头，这样的名称留给系统表空间使用。

  取值范围：字符串，要符合标识符的命名规范。

- **OWNER user_name**

  指定该表空间的所有者。缺省时，新表空间的所有者是当前用户。

  只有系统管理员可以创建表空间，但是可以通过 OWNER 子句把表空间的所有权赋给其他非系统管理员。

  取值范围：字符串，已存在的用户。

- **RELATIVE**

若指定该参数，表示使用相对路径，LOCATION 目录是相对于各个数据库节点数据目录下的。

目录层次：数据库节点的数据目录/pg_location/相对路径

相对路径最多指定两层。

若没有指定该参数，表示使用绝对表空间路径，LOCATION 目录需要使用绝对路径。

- **LOCATION directory**

用于表空间的目录。当创建绝对表空间路径时，对于目录有如下要求：

- PanWeiDB 系统用户必须对该目录拥有读写权限，并且目录为空。如果该目录不存在，将由系统自动创建。
- 目录必须是绝对路径，目录中不得含有特殊字符（如$、>等）。
- 目录不允许指定在数据库数据目录下。
- 目录需为本地路径。

取值范围：字符串，有效的目录。

- **MAXSIZE 'space_size'**

指定表空间在单个数据库节点上的最大值。

取值范围：字符串格式为正整数+单位，单位当前支持 K/M/G/T/P。解析后的数值以 K 为单位，且范围不能够超过 64 比特表示的有符号整数，即 1KB~9007199254740991KB。

- **THRESHOLD 'space_percentile'**

指定表空间阈值。当表空间实际大小达到该表空间阈值（THRESHOLD）时，告知用户该表空间使用信息，包括该表空间名、阈值、当前表空间大小、申请空间大小等，以便用户能够做出相应处理。

取值范围：字符串格式为正整数（1-99 默认为 90），为 maxsize 的百分比信息。

- **random_page_cost**

指定随机读取 page 的开销。

取值范围：0~1.79769e+308。

默认值：使用 GUC 参数 random_page_cost 的值。

- **seq_page_cost**

指定顺序读取 page 的开销。

取值范围：0~1.79769e+308。

默认值：使用 GUC 参数 seq_page_cost 的值。

示例

1、创建表空间。

```
CREATE TABLESPACE ds_location1 RELATIVE LOCATION 'tablespace/tablespace_1';
```

2、创建用户 joe，用户 jay。

```
CREATE ROLE joe IDENTIFIED BY 'Aa123456';
CREATE ROLE jay IDENTIFIED BY 'Aa123456';
```

3、创建表空间，且所有者指定为用户 joe。

```
 CREATE TABLESPACE ds_location2 OWNER joe RELATIVE LOCATION 'tablespace/tabl
espace_1';
```

4、把表空间 ds_location1 重命名为 ds_location3。

```
ALTER TABLESPACE ds_location1 RENAME TO ds_location3;
```

5、改变表空间 ds_location2 的所有者。

```
ALTER TABLESPACE ds_location2 OWNER TO jay;
```

6、删除表空间。

```
DROP TABLESPACE ds_location2;
 DROP TABLESPACE ds_location3;
```

7、删除用户。

```
DROP ROLE joe;
DROP ROLE jay;
```

相关链接

参考：SQL 语法参考->SQL 语法->CREATE DATABASE、CREATE TABLE、CREATE INDEX、DROP TABLESPACE、ALTER TABLESPACE 章节。

4.5.95.CREATE TEXT SEARCH CONFIGURATION

功能描述

创建新的文本搜索配置。一个文本搜索配置声明一个能将一个字符串划分成符号的文本搜索解析器，加上可以用于确定搜索对哪些标记感兴趣的字典。

注意事项

- 若仅声明分析器，那么新的文本搜索配置初始没有从符号类型到词典的映射， 因此会忽略所有的单词。后面必须调用 ALTER TEXT SEARCH CONFIGURATION 命令创建映射使配置生效。如果声明了 COPY 选项，那么会自动拷贝指定的文本搜索配置的解析器、映射、配置选项等信息。
- 若模式名称已给出，那么文本搜索配置会在声明的模式中创建。否则会在当前模式创建。
- 定义文本搜索配置的用户成为其所有者。
- PARSER 和 COPY 选项是互相排斥的，因为当一个现有配置被复制，其分析器配置也被复制了。
- 若仅声明分析器，那么新的文本搜索配置初始没有从符号类型到词典的映射， 因此会忽略所有的单词。

语法格式

```
CREATE TEXT SEARCH CONFIGURATION name
  ( PARSER = parser_name | COPY = source_config )
  [ WITH ( {configuration_option = value} [, ...] )];
```

参数说明

- **name**

  要创建的文本搜索配置的名称。该名称可以有模式修饰。

- **parser_name**

  用于该配置的文本搜索分析器的名称。

- **source_config**

  要复制的现有文本搜索配置的名称。

- **configuration_option**

  文本搜索配置的配置参数，主要是针对 parser_name 执行的解析器或者 source_config 隐含的解析器而言的。

  取值范围：目前共支持 default、ngram 两种类型的解析器，其中 default 类型的解析器没有对应的 configuration_option、ngram 类型解析器对应的 configuration_option 如表 1 所示。

  表 1 ngram 类型解析器对应的配置参数

解析器	配置参数	参数描述	取值范围
ngram	gram_size	分词长度。	正整数，1~4 默认值：2
	punctuation_ignore	是否忽略标点符号。	❖ true（默认值）：忽略标点符号。 ❖ false：不忽略标点符号。
	grapsymbol_ignore	是否忽略图形化字符。	❖ true：忽略图形化字符。 ❖ false（默认值）：不忽略图形化字符。

示例

1、创建文本搜索配置。

```
panweidb=# CREATE TEXT SEARCH CONFIGURATION ngram2 (parser=ngram) WITH (gram_size = 2, grapsymbol_ignore = false);
```

2、创建文本搜索配置。

```
panweidb=# CREATE TEXT SEARCH CONFIGURATION ngram3 (copy=ngram2) WITH (gram_size = 2, grapsymbol_ignore = false);
```

3、添加类型映射。

```
panweidb=# ALTER TEXT SEARCH CONFIGURATION ngram2 ADD MAPPING FOR multisymbol WITH simple;
```

4、创建用户 joe。

```
panweidb=# CREATE USER joe IDENTIFIED BY 'xxxxxxxxx';
```

5、修改文本搜索配置的所有者。

```
panweidb=# ALTER TEXT SEARCH CONFIGURATION ngram2 OWNER TO joe;
```

6、修改文本搜索配置的 schema。

```
panweidb=# ALTER TEXT SEARCH CONFIGURATION ngram2 SET SCHEMA joe;
```

7、重命名文本搜索配置。

```
panweidb=# ALTER TEXT SEARCH CONFIGURATION joe.ngram2 RENAME TO ngram_
2;
```

8、删除类型映射。

```
panweidb=# ALTER TEXT SEARCH CONFIGURATION joe.ngram_2 DROP MAPPING IF E
XISTS FOR multisymbol;
```

9、删除文本搜索配置。

```
panweidb=# DROP TEXT SEARCH CONFIGURATION joe.ngram_2;
panweidb=# DROP TEXT SEARCH CONFIGURATION ngram3;
```

10、-删除 Schema 及用户 joe。

```
panweidb=# DROP SCHEMA IF EXISTS joe CASCADE;
panweidb=# DROP ROLE IF EXISTS joe;
```

相关链接

参考：SQL 语法参考->SQL 语法->ALTER TEXT SEARCH CONFIGURATION、DROP TEXT SEARCH CONFIGURATION 章节。

4.5.96.CREATE TEXT SEARCH DICTIONARY

功能描述

创建一个新的全文检索词典。词典是一种指定在全文检索时识别特定词并处理的方法。

词典的创建依赖于预定义模板（在系统表 PG_TS_TEMPLATE 中定义），支持创建五种类型的词典，分别是 Simple、Ispell、Synonym、Thesaurus 以及 Snowball，每种类型的词典可以完成不同的任务。

注意事项

- 具有 SYSADMIN 权限的用户可以执行创建词典操作，创建该词典的用户自动成为其所有者。

- 临时模式（pg_temp）下不允许创建词典。
- 创建或修改词典之后，任何对于用户自定义的词典定义文件的修改，将不会影响到数据库中的词典。如果需要在数据库中使用这些修改，需使用 ALTER 语句更新对应词典的定义文件。

语法格式

```
CREATE TEXT SEARCH DICTIONARY name (
  TEMPLATE = template
  [, option = value [, ... ]]
);
```

参数说明

- **name**

  要创建的词典的名称（可指定模式名，否则在当前模式下创建）。

  取值范围：符合标识符命名规范的字符串，且最大长度不超过 63 个字符。

- **template**

  模板名。

  取值范围：系统表 PG_TS_TEMPLATE 中定义的模板：

  Simple/Synonym/Thesaurus/Ispell/Snowball。

- **option**

  参数名。与 template 值对应，不同的词典模板具有不同的参数列表，且与指定顺序无关。

  - Simple 词典对应的 option
    - **STOPWORDS**

      停用词表文件名，默认后缀名为 stop。停用词文件格式为一组 word 列表，每行定义一个停用词。词典处理时，文件中的空行和空格会被忽略，并将 stopword 词组转换为小写形式。

    - **ACCEPT**

      是否将非停用词设置为已识别。默认值为 true。

      当 Simple 词典设置参数 ACCEPT=true 时，将不会传递任何 token 给后继词典，此时建议将其放置在词典列表的最后。反

之，当 ACCEPT=false 时，建议将该 Simple 词典放置在列表中的至少一个词典之前。

- **FILEPATH**

  词典文件所在目录。目录可以指定为本地目录和 OBS 目录（只能在安全模式下指定 OBS 目录，通过启动时添加 securitymode 选项进入安全模式）。其中，本地目录格式为“file://absolute_path”，OBS 目录格式为“obs://bucket_name/path accesskey=ak secretkey=sk region=rg”。默认值为预定义词典文件所在目录。FILEPATH 参数必须和 STOPWORDS 参数同时指定，不允许单独指定。

- Synonym 词典对应的 option
  - **SYNONYM**
  - 同义词词典的定义文件名，默认后缀名为 syn。

    文件格式为一组同义词列表，每行格式为“token synonym”，即 token 和其对应的 synonym，中间以空格相连。
  - **CASESENSITIVE**

    设置是否大小写敏感，默认值为 false，此时词典文件中的 token 和 synonym 均会转为小写形式处理。如果设置为 true，则不会进行小写转换。
  - **FILEPATH**

    同义词词典文件所在目录。目录可以指定为本地目录和 OBS 目录两种形式（只能在安全模式下指定 OBS 目录，通过启动时添加 securitymode 选项进入安全模式）。其中，本地目录格式为“file://absolute_path”，OBS 目录格式为“obs://bucket_name/path accesskey=ak secretkey=sk region=rg”。默认值为预定义词典文件所在目录。
- Thesaurus 词典对应的 option
  - **DICTFILE**
  - 词典定义文件名，默认后缀名为 ths。

文件格式为一组同义词列表，每行格式为“sample words：indexed words”，中间冒号（:）作为短语和其替换词间的分隔符。TZ 词典处理时，如果有多个匹配的 sample words，将选择最长匹配输出。

- **DICTIONARY**

  用于词规范化的子词典名，必须且仅能定义一个。该词典必须是已经存在的，在检查短语匹配之前使用，用于识别和规范输入文本。

  如果子词典无法识别输入词，将会报错。此时，需要移除该词或者更新子词典使其识别。此外，可在 indexed words 的开头放上一个星号（*）来跳过在其上应用子词典，但是所有 sample words 必须可以被子词典识别。

  如果词典文件定义的 sample words 中，含有子词典中定义的停用词，需要用问号（？）替代停用词。假设 a 和 the 是子词典中所定义的停用词，如(? one ? two : swsw)。

  上述同义词组定义会匹配“a one the two”以及“the one a two”，这两个短语均会被 swsw 替代输出。

- **FILEPATH**

词典定义文件所在目录。目录可以指定为本地目录和 OBS 目录两种形式（只能在安全模式下指定 OBS 目录，通过启动时添加 securitymode 选项进入安全模式）。其中，本地目录格式为“file://absolute_path”，OBS 目录格式为“obs://bucket_name/path accesskey=ak secretkey=sk region=rg”。默认值为预定义词典文件所在目录。

- Ispell 词典
  - **DICTFILE**

    词典定义文件名，默认后缀名为 dict。

  - **AFFFILE**

    词缀文件名，默认后缀名为 affix。

  - **STOPWORDS**

    停用词文件名，默认后缀名为 stop，文件格式要求与 Simple 类型词典的停用词文件相同。

  - **FILEPATH**

词典文件所在目录。可以指定为本地目录和 OBS 目录两种形式（只能在安全模式下指定 OBS 目录，通过启动时添加 securitymode 选项进入安全模式）。其中，本地目录格式为“file://absolute_path”，OBS 目录格式为“obs://bucket_name/pathaccesskey=ak secretkey=sk region=rg”。默认值为预定义词典文件 所在目录。

- Snowball 词典
  - **LANGUAGE**

    语言名，标识使用哪种语言的词干分析算法。算法按照对应语言中的拼写规则，缩减输入词的常见变体形式为一个基础词或词干。
  - **STOPWORDS**

    停用词表文件名，默认后缀名为 stop，文件格式要求与 Simple 类 型词典的停用词文件相同。
  - **FILEPATH**

    词典定义文件所在目录。可以指定为本地目录或者 OBS 目录（只能在安全模式下指定 OBS 目录，通过启动时添加 securitymode 选项进入安全模式）。其中，本地目录格式为“file://absolute_path”，OBS 目录格式为“obs://bucket_name/path accesskey=ak secretkey=sk region=rg”。默认值为预定义词典文件所在目录。FILEPATH 参数必须和 STOPWORDS 参数同时指定，不允许单独指定。

📖 **说明：**

词典定义文件的文件名仅支持小写字母、数据、下划线混合。

- **value**

参数值。如果不是简单的标识符或数字，则参数值必须加单引号（标示符和数字同样可以加上单引号）。

示例

参考：SQL 语法参考->全文检索->词典->配置示例章节的示例。

相关链接

参考：SQL 语法参考->SQL 语法->ALTER TEXT SEARCH DICTIONARY、CREATE TEXT SEARCH DICTIONARY 章节。

4.5.97.CREATE TRIGGER

功能描述

创建一个触发器。 触发器将与指定的表或视图关联，并在特定条件下执行指定的函数。

注意事项

- 当前仅支持在普通行存表上创建触发器，不支持在列存表、unlogged 表等类型表上创建触发器。

【说明】
仅 Oracle 兼容模式下（即数据库实例初始化时指定 DBCOMPATIBILITY='A'）支持在全局临时表上创建触发器，其余情况下不支持在临时表上创建触发器。

- 如果为同一事件定义了多个相同类型的触发器，则按触发器的名称字母顺序触发它们。
- 触发器常用于多表间数据关联同步场景，对 SQL 执行性能影响较大，不建议在大数据量同步及对性能要求高的场景中使用。
- 执行创建触发器操作的用户需要拥有指定表的 TRIGGER 权限或被授予了 CREATE ANY TRIGGER 权限。
- 当使用 WHEN(condition)时，此处的括号不能省略，condition 需要被括号修饰。

语法格式

```
CREATE [ CONSTRAINT ] TRIGGER trigger_name { BEFORE | AFTER | INSTEAD OF } { eve
nt [ OR ... ] }
   ON { table_name | global_temporary_table }
   [ FROM referenced_table_name ]
   { NOT DEFERRABLE | [ DEFERRABLE ] { INITIALLY IMMEDIATE | INITIALLY DEFERRED } }
   [ FOR [ EACH ] { ROW | STATEMENT } ]
   [ WHEN ( condition ) ]
   EXECUTE PROCEDURE function_name ( arguments );
```

其中 event 包含以下几种：

```
INSERT
UPDATE [ OF column_name [, ... ] ]
DELETE
TRUNCATE
```

参数说明

- **CONSTRAINT**

  可选项，指定此参数将创建约束触发器，即触发器作为约束来使用。除了可以使用 SET CONSTRAINTS 调整触发器触发的时间之外，这与常规触发器相同。 约束触发器必须是 AFTER ROW 触发器。

- **trigger_name**

  触发器名称，该名称不能限定模式，因为触发器自动继承其所在表的模式，且同一个表的触发器不能重名。 对于约束触发器，使用 SET CONSTRAINTS 修改触发器行为时也使用此名称。

  取值范围：符合标识符命名规范的字符串，且最大长度不超过 63 个字符。

- **BEFORE**

  触发器函数是在触发事件发生前执行。

- **AFTER**

  触发器函数是在触发事件发生后执行，约束触发器只能指定为 AFTER。

- **INSTEAD OF**

  触发器函数直接替代触发事件。

- **event**

  启动触发器的事件，取值范围包括：INSERT、UPDATE、DELETE 或 TRUNCATE，也可以通过 OR 同时指定多个触发事件。

  对于 UPDATE 事件类型，可以使用下面语法指定列：

  ```
  UPDATE OF column_name1 [, column_name2 ... ]
  ```

  表示当这些列作为 UPDATE 语句的目标列时，才会启动触发器，但是 INSTEAD OF UPDATE 类型不支持指定列信息。如果 UPDATE OF 指定的列包含生成列，当生成列依赖的列是 UPDATE 语句的目标列时，也会启动触发器。

- **table_name**

  需要创建触发器的表名称。

  取值范围：数据库中已经存在的表名称。

- **global_temporary_table**

  需要创建触发器的全局临时表名。包括会话级临时表（ON COMMIT PRESERVE ROWS）和事务级临时表（ON COMMIT DELETE ROWS）。

  【说明】

  该功能仅在数据库兼容模式为 Oracle 时支持（即数据库初始化时指定 DBCOMPATIBILITY='A'），参考《PanWeiDB V2.0-S3.0.1_B01 Oracle 兼容性手册》中的 SQL 语法 CREATE TRIGGER。

- **referenced_table_name**

  约束引用的另一个表的名称。 只能为约束触发器指定，常见于外键约束。

  取值范围：数据库中已经存在的表名称。

- **DEFERRABLE | NOT DEFERRABLE**

  约束触发器的启动时机，仅作用于约束触发器。这两个关键字设置该约束是否可推迟。

  详细介绍请参见：SQL 语法参考->SQL 语法->CREATE TABLE 章节。

- **INITIALLY IMMEDIATE | INITIALLY DEFERRED**

  如果约束是可推迟的，则这个子句声明检查约束的缺省时间，仅作用于约束触发器。

  详细介绍请参见：SQL 语法参考->SQL 语法->CREATE TABLE 章节。

- **FOR EACH ROW | FOR EACH STATEMENT**

  触发器的触发频率。

  - FOR EACH ROW 是指该触发器是受触发事件影响的每一行触发一次。
  - FOR EACH STATEMENT 是指该触发器是每个 SQL 语句只触发一次。

  未指定时默认值为 FOR EACH STATEMENT。约束触发器只能指定为 FOR EACH ROW。

- **condition**

决定是否实际执行触发器函数的条件表达式。当指定 WHEN 时，只有在条件返回 true 时才会调用该函数。

在 FOR EACH ROW 触发器中，WHEN 条件可以通过分别写入 OLD.column_name 或 NEW.column_name 来引用旧行或新行值的列。 当然，INSERT 触发器不能引用 OLD 和 DELETE 触发器不能引用 NEW。

INSTEAD OF 触发器不支持 WHEN 条件。

WHEN 表达式不能包含子查询。

对于约束触发器，WHEN 条件的评估不会延迟，而是在执行更新操作后立即发生。 如果条件返回值不为 true，则触发器不会排队等待延迟执行。

- **function_name**

用户定义的函数，必须声明为不带参数并返回类型为触发器，在触发器触发时执行。

- **arguments**

执行触发器时要提供给函数的可选的以逗号分隔的参数列表。参数是文字字符串常量，简单的名称和数字常量也可以写在这里，但它们都将被转换为字符串。 请检查触发器函数的实现语言的描述，以了解如何在函数内访问这些参数。

【说明】

关于触发器种类：

- INSTEAD OF 的触发器必须标记为 FOR EACH ROW，并且只能在视图上定义。
- BEFORE 和 AFTER 触发器作用在视图上时，只能标记为 FOR EACH STATEMENT。
- TRUNCATE 类型触发器仅限 FOR EACH STATEMENT。

表 1 表和视图上支持的触发器种类：

触发时机	触发事件	行级	语句级
BEFORE	INSERT/UPDATE/DELETE	表	表和视图
	TRUNCATE	不支持	表

触发时机	触发事件	行级	语句级
AFTER	INSERT/UPDATE/DELETE	表	表和视图
	TRUNCATE	不支持	表
INSTEAD OF	INSERT/UPDATE/DELETE	视图	不支持
	TRUNCATE	不支持	不支持

表 2 PLPGSQL 类型触发器函数特殊变量：

变量名	变量含义
NEW	INSERT 及 UPDATE 操作涉及 tuple 信息中的新值，对 DELETE 为空。
OLD	UPDATE 及 DELETE 操作涉及 tuple 信息中的旧值，对 INSERT 为空。
TG_NAME	触发器名称。
TG_WHEN	触发器触发时机（BEFORE/AFTER/INSTEAD OF）。
TG_LEVEL	触发频率（ROW/STATEMENT）。
TG_OP	触发操作（INSERT/UPDATE/DELETE/TRUNCATE）。
TG_RELID	触发器所在表 OID。
TG_RELNAME	触发器所在表名（已废弃，现用 TG_TABLE_NAME 替代）。
TG_TABLE_NAME	触发器所在表名。
TG_TABLE_SCHEMA	触发器所在表的 SCHEMA 信息。
TG_NARGS	触发器函数参数个数。

变量名	变量含义
TG_ARGV[]	触发器函数参数列表。

示例

```
--创建源表及触发表
panweidb=# CREATE TABLE test_trigger_src_tbl(id1 INT, id2 INT, id3 INT);
panweidb=# CREATE TABLE test_trigger_des_tbl(id1 INT, id2 INT, id3 INT);

--创建触发器函数
panweidb=# CREATE OR REPLACE FUNCTION tri_insert_func() RETURNS TRIGGER AS
        $$
        DECLARE
        BEGIN
            INSERT INTO test_trigger_des_tbl VALUES(NEW.id1, NEW.id2, NEW.id3);
            RETURN NEW;
        END
        $$ LANGUAGE PLPGSQL;

panweidb=# CREATE OR REPLACE FUNCTION tri_update_func() RETURNS TRIGGER AS
        $$
        DECLARE
        BEGIN
            UPDATE test_trigger_des_tbl SET id3 = NEW.id3 WHERE id1=OLD.id1;
            RETURN OLD;
        END
        $$ LANGUAGE PLPGSQL;

panweidb=# CREATE OR REPLACE FUNCTION TRI_DELETE_FUNC() RETURNS TRIGGER
AS
        $$
        DECLARE
        BEGIN
            DELETE FROM test_trigger_des_tbl WHERE id1=OLD.id1;
            RETURN OLD;
```

```
    END
    $$ LANGUAGE PLPGSQL;

--创建 INSERT 触发器
panweidb=# CREATE TRIGGER insert_trigger
    BEFORE INSERT ON test_trigger_src_tbl
    FOR EACH ROW
    EXECUTE PROCEDURE tri_insert_func();

--创建 UPDATE 触发器
panweidb=# CREATE TRIGGER update_trigger
    AFTER UPDATE ON test_trigger_src_tbl
    FOR EACH ROW
    EXECUTE PROCEDURE tri_update_func();

--创建 DELETE 触发器
panweidb=# CREATE TRIGGER delete_trigger
    BEFORE DELETE ON test_trigger_src_tbl
    FOR EACH ROW
    EXECUTE PROCEDURE tri_delete_func();

--执行 INSERT 触发事件并检查触发结果
panweidb=# INSERT INTO test_trigger_src_tbl VALUES(100,200,300);
panweidb=# SELECT * FROM test_trigger_src_tbl;
panweidb=# SELECT * FROM test_trigger_des_tbl; //查看触发操作是否生效。

--执行 UPDATE 触发事件并检查触发结果
panweidb=# UPDATE test_trigger_src_tbl SET id3=400 WHERE id1=100;
panweidb=# SELECT * FROM test_trigger_src_tbl;
panweidb=# SELECT * FROM test_trigger_des_tbl; //查看触发操作是否生效

--执行 DELETE 触发事件并检查触发结果
panweidb=# DELETE FROM test_trigger_src_tbl WHERE id1=100;
panweidb=# SELECT * FROM test_trigger_src_tbl;
panweidb=# SELECT * FROM test_trigger_des_tbl; //查看触发操作是否生效
```

```
--修改触发器
panweidb=# ALTER TRIGGER delete_trigger ON test_trigger_src_tbl RENAME TO dele
te_trigger_renamed;

--禁用 insert_trigger 触发器
panweidb=# ALTER TABLE test_trigger_src_tbl DISABLE TRIGGER insert_trigger;

--禁用当前表上所有触发器
panweidb=# ALTER TABLE test_trigger_src_tbl DISABLE TRIGGER ALL;

--删除触发器
panweidb=# DROP TRIGGER insert_trigger ON test_trigger_src_tbl;
panweidb=# DROP TRIGGER update_trigger ON test_trigger_src_tbl;
panweidb=# DROP TRIGGER delete_trigger_renamed ON test_trigger_src_tbl;
```

相关链接

参考：SQL 语法参考->SQL 语法->ALTER TRIGGER、DROP TRIGGER、ALTER TABLE 章节。

4.5.98.CREATE TYPE

功能描述

在当前数据库中定义一种新的数据类型。定义数据类型的用户将成为该数据类型的拥有者。类型只适用于行存表

有五种形式的 CREATE TYPE，分别为：复合类型、基本类型、shell 类型、枚举类型和集合类型。

- 复合类型

  复合类型由一个属性名和数据类型的列表指定。如果属性的数据类型是可排序的，也可以指定该属性的排序规则。复合类型本质上和表的行类型相同，但是如果只想定义一种类型，使用 CREATE TYPE 避免了创建一个实际的表。单独的复合类型也是很有用的，例如可以作为函数的参数或者返回类型。

为了能够创建复合类型，必须拥有在其所有属性类型上的 USAGE 特权。

- 基本类型

  用户可以自定义一种新的基本类型（标量类型）。通常来说这些函数必须是底层语言所编写。

- shell 类型

  shell 类型是一种用于后面要定义的类型的占位符，通过发出一个不带除类型名之外其他参数的 CREATE TYPE 命令可以创建这种类型。在创建基本类型时，需要 shell 类型作为一种向前引用。

- 枚举类型

  由若干个标签构成的列表，每一个标签值都是一个非空字符串，且字符串长度必须不超过 63 个字节。

- 集合类型

  类似数组，但是没有长度限制，主要在存储过程中使用。

- 被授予 CREATE ANY TYPE 权限的用户，可以在 public 模式和用户模式下创建类型。

注意事项

如果给定一个模式名，那么该类型将被创建在指定的模式中。否则它会被创建在当前模式中。类型名称必须与同一个模式中任何现有的类型或者域相区别（因为表具有相关的数据类型，类型名称也必须与同一个模式中任何现有表的名称不同）。

语法格式

```
CREATE TYPE name AS
  ( [ attribute_name data_type [ COLLATE collation ] [, ... ] ] )

CREATE TYPE name (
  INPUT = input_function,
  OUTPUT = output_function
  [ , RECEIVE = receive_function ]
  [ , SEND = send_function ]
  [ , TYPMOD_IN =
type_modifier_input_function ]
```

```
  [ , TYPMOD_OUT =
type_modifier_output_function ]
  [ , ANALYZE = analyze_function ]
  [ , INTERNALLENGTH = { internallength |
VARIABLE } ]
  [ , PASSEDBYVALUE ]
  [ , ALIGNMENT = alignment ]
 [ , STORAGE = storage ]
  [ , LIKE = like_type ]
  [ , CATEGORY = category ]
  [ , PREFERRED = preferred ]
  [ , DEFAULT = default ]
  [ , ELEMENT = element ]
  [ , DELIMITER = delimiter ]
  [ , COLLATABLE = collatable ]
)

CREATE TYPE name

CREATE TYPE name AS ENUM
  ( [ 'label' [, ... ] ] )

CREATE TYPE name AS TABLE OF data_type
```

参数说明

复合类型

- **name**

  要创建的类型的名称（可以被模式限定）。

- **attribute_name**

  复合类型的一个属性（列）的名称。

- **data_type**

要成为复合类型的一个列的现有数据类型的名称。可以使用%ROWTYPE 间接引用表的类型，或者使用%TYPE 间接引用表或复合类型中某一列的类型。

❖ **collation**

要关联到复合类型的一列的现有排序规则的名称。排序规则可以使用“select * from pg_collation”命令从 pg_collation 系统表中查询，默认的排序规则为查询结果中以 default 开始的行。

基本类型

自定义基本类型时，参数可以以任意顺序出现，input_function 和 output_function 为必选参数，其他为可选参数。

❖ **input_function**

将数据从类型的外部文本形式转换为内部形式的函数名。

输入函数可以被声明为有一个 cstring 类型的参数，或者有三个类型分别为 cstring、 oid、integer 的参数。

- cstring 参数是以 C 字符串存在的输入文本。
- oid 参数是该类型自身的 OID（对于数组类型则是其元素类型的 OID）。
- integer 参数是目标列的 typmod（如果知道，不知道则将传递 -1）。

输入函数必须返回一个该数据类型本身的值。通常，一个输入函数应该被声明为 STRICT。 如果不是这样，在读到一个 NULL 输入值时，调用输入函数时第一个参数会是 NULL。在这种情况下，该函数必须仍然返回 NULL，除非调用函数发生了错误（这种情况主要是想支持域输入函数，域输入函数可能需要拒绝 NULL 输入）。

【说明】

❖ 输入和输出函数能被声明为具有新类型的结果或参数是因为：必须在创建新类型之前创建这两个函数。而新类型应该首先被定义为一种 shell type，它是一种占位符类型，除了名称和拥有者之外它没有其他属性。这可以通过不带额外参数的命令 CREATE TYPE name 做到。然后用 C 写的 I/O 函数可以被定义为引用这种 shell type。最后，用带有完整定义的 CREAT

E TYPE 把该 shell type 替换为一个完全的、合法的类型定义，之后新类型就可以正常使用了。

- 输入和输出函数若为 internel 类型且指定为内部系统函数，则其输入函数和输出函数的参数类型需保持一致，且新类型的 INTERNALLENGTH 和 PASSEDBYVALUE 需要与输入函数和输出函数的参数类型保持一致。

- **output_function**

  将数据从类型的内部形式转换为外部文本形式的函数名。

  输出函数必须被声明为有一个新数据类型的参数。输出函数必须返回类型 cstring。对于 NULL 值不会调用输出函数。

- **receive_function**

  可选参数。将数据从类型的外部二进制形式转换成内部形式的函数名。

  如果没有该函数，该类型不能参与到二进制输入中。二进制表达转换成内部形式代价更低，然而却更容易移植（例如，标准的整数数据类型使用网络字节序作为外部二进制表达，而内部表达是机器本地的字节序）。receive_function 应该执行足够的检查以确保该值是有效的。

  接收函数可以被声明为有一个 internal 类型的参数，或者有三个类型分别为 internal、oid、integer 的参数。

  - internal 参数是一个指向 StringInfo 缓冲区的指针，其中保存着接收到的字节串。
  - oid 和 integer 参数和文本输入函数的相同。

  接收函数必须返回一个该数据类型本身的值。通常，一个接收函数应该被声明为 STRICT。如果不是这样，在读到一个 NULL 输入值时调用接收函数时第一个参数会是 NULL。在这种情况下，该函数必须仍然返回 NULL，除非接收函数发生了错误（这种情况主要是想支持域接收函数，域接收函数可能需要拒绝 NULL 输入）。

- **send_function**

  可选参数。将数据从类型的内部形式转换为外部二进制形式的函数名。

  如果没有该函数，该类型将不能参与到二进制输出中。发送函数必须被声明为有一个新数据类型的参数。发送函数必须返回类型 bytea。对于 NULL 值不会调用发送函数。

- **type_modifier_input_function**

  可选参数。将类型的修饰符数组转换为内部形式的函数名。

- **type_modifier_output_function**

  可选参数。将类型的修饰符的内部形式转换为外部文本形式的函数名。

  说明：

  如果该类型支持修饰符（附加在类型声明上的可选约束，例如，char(5) 或 numeric(30,2)），则需要可选的 type_modifier_input_function 以及 type_modifier_output_function。PanWeiDB 允许用户定义的类型有一个或者多个简单常量或者标识符作为修饰符。不过，为了存储在系统目录中，该信息必须能被打包到一个非负整数值中。所声明的修饰符会被以 cstring 数组的形式传递给 type_modifier_input_function。 type_modifier_input_function 必须检查该值的合法性（如果值错误就抛出一个错误），如果值正确，要返回一个非负 integer 值，该值将被存储在“typmod”列中。如果类型没有 type_modifier_input_function 则类型修饰符将被拒绝。type_modifier_output_function 把内部的整数 typmod 值转换回正确的形式用于用户显示。

  type_modifier_output_function 必须返回一个 cstring 值，该值就是追加到类型名称后的字符串。例如，numeric 的函数可能会返回(30,2)。如果默认的显示格式就是只把存储的 typmod 整数值放在圆括号内，则允许省略 type_modifier_output_function。

- **analyze_function**

  可选参数。为该数据类型执行统计分析的函数名的可选参数。

  默认情况下，如果该类型有一个默认的 B-tree 操作符类，ANALYZE 将尝试用类型的“equals”和“less-than”操作符来收集统计信息。这种行为对于非标量类型并不合适，因此可以通过指定一个自定义分析函数来覆盖这种行为。分析函数必须被声明为有一个类型为 internal 的参数，并且返回一个 boolean 结果。

- **internallength**

可选参数。一个数字常量，用于指定新类型的内部表达的字节长度。默认为变长。

虽然只有 I/O 函数和其他为该类型创建的函数才知道新类型的内部表达的细节，但是内部表达的一些属性必须被向 PanWeiDB 声明。其中最重要的是 internallength。基本数据类型可以是定长的（这种情况下 internallength 是一个正整数）或者是变长的（把 internallength 设置为 VARIABLE，在内部通过把 typlen 设置为-1 表示）。所有变长类型的内部表达都必须以一个 4 字节整数开始，internallength 定义了总长度。

- **PASSEDBYVALUE**

可选参数。表示这种数据类型的值需要被传值而不是传引用。传值的类型必须是定长的，并且它们的内部表达不能超过 Datum 类型（某些机器上是 4 字节，其他机器上是 8 字节）的尺寸。

- **alignment**

可选参数。该参数指定数据类型的存储对齐需求。如果被指定，必须是 char、int2、int4 或者 double。默认是 int4。

允许的值等同于以 1、2、4 或 8 字节边界对齐。要注意变长类型的 alignment 参数必须至少为 4，因为它们需要包含一个 int4 作为它们的第一个组成部分。

- **storage**

可选参数。该数据类型的存储策略。

如果被指定，必须是 plain、external、extended 或者 main。默认是 plain。

  - plain 指定该类型的数据将总是被存储在线内并且不会被压缩。（对定长类型只允许 plain）
  - extended 指定系统将首先尝试压缩一个长的数据值，并且将在数据仍然太长的情况下把值移出主表行。
  - external 允许值被移出主表，但是系统将不会尝试对它进行压缩。
  - main 允许压缩，但是不鼓励把值移出主表（如果没有其他办法让行的大小变得合适，具有这种存储策略的数据项仍将被移出主

表，但比起 extended 以及 external 项来，这种存储策略的数据项会被优先考虑保留在主表中）。

➢除 plain 之外所有的 storage 值都暗示该数据类型的函数能处理被 TOAST 过的值。指定的值仅仅是决定一种可 TOAST 数据类型的列的默认 TOAST 存储策略，用户可以使用 ALTER TABLE SET STORAGE 为列选取其他策略。

❖ **like_type**

可选参数。与新类型具有相同表达的现有数据类型的名称。会从这个类型中复制 internallength、 passedbyvalue、 alignment 以及 storage 的值（ 除非在这个 CREATE TYPE 命令的其他地方用显式说明覆盖）。当新类型的低层实现是以一种现有的类型为参考时，用这种方式指定表达特别有用。

❖ **category**

可选参数。这种类型的分类码（一个 ASCII 字符）。 默认是“用户定义类型”的'U'。为了创建自定义分类， 也可以选择其他 ASCII 字符。

❖ **preferred**

可选参数。如果这种类型是其类型分类中的优先类型则为 TRUE，否则为 FALSE。默认为假。在一个现有类型分类中创建一种新的优先类型要非常谨慎， 因为这可能会导致很大的改变。

📖 **说明：**

category 和 preferred 参数可以被用来帮助控制在混淆的情况下应用哪一种隐式造型。每一种数据类型都属于一个用单个 ASCII 字符命名的分类，并且每一种类型可以是其所属分类中的“首选”。当有助于解决重载函数或操作符时，解析器将优先造型到首选类型（但是只能从同类的其他类型造型）。对于没有隐式转换到或来自任意其他类型的类型，让这些设置保持默认即可。不过，对于有隐式转换的相关类型的组，把它们都标记为属于同一个类别并且选择一种或两种“最常用”的类型作为该类别的首选通常是很有用的。在把一种用户定义的类型增加到一个现有的内建类别（例如，数字或者字符串类型）中时，category 参数特别有用。不过，也可以创建新的全

部是用户定义类型的类别。对这样的类别，可选择除大写字母之外的任何 ASCII 字符。

- **default**

可选参数。数据类型的默认值。如果被省略，默认值是空。

如果用户希望该数据类型的列被默认为某种非空值，可以指定一个默认值。默认值可以用 DEFAULT 关键词指定（这样一个默认值可以被附加到一个特定列的显式 DEFAULT 子句覆盖）。

- **element**

可选参数。被创建的类型是一个数组，element 指定了数组元素的类型。例如，要定义一个 4 字节整数的数组（int4），应指定 ELEMENT = int4。

- **delimiter**

可选参数。指定这种类型组成的数组中分隔值的定界符。

可以把 delimiter 设置为一个特定字符，默认的定界符是逗号（,）。注意定界符是与数组元素类型相关的，而不是数组类型本身相关。

- **collatable**

可选参数。如果这个类型的操作可以使用排序规则信息，则为 TRUE。默认为 FALSE。

如果 collatable 为 TRUE，这种类型的列定义和表达式可能通过使用 COLLATE 子句携带有排序规则信息。在该类型上操作的函数的实现负责真正利用这些信息，仅把类型标记为可排序的并不会让它们自动地去使用这类信息。

- **lable**

可选参数。与枚举类型的一个值相关的文本标签，其值为长度不超过 63 个字符的非空字符串。

说明：

在创建用户定义类型的时候， PanWeiDB 会自动创建一个与之关联的数组类型，其名称由该元素类型的名称前缀一个下划线组成。

示例

```
--创建一种复合类型，建表并插入数据以及查询。
```

```
panweidb=# CREATE TYPE compfoo AS (f1 int, f2 text);
panweidb=# CREATE TABLE t1_compfoo(a int, b compfoo);
panweidb=# CREATE TABLE t2_compfoo(a int, b compfoo);
panweidb=# INSERT INTO t1_compfoo values(1,(1,'demo'));
panweidb=# INSERT INTO t2_compfoo select * from t1_compfoo;
panweidb=# SELECT (b).f1 FROM t1_compfoo;
panweidb=# SELECT * FROM t1_compfoo t1 join t2_compfoo t2 on (t1.b).f1=(t1.b).f1;

--创建一个集合类型
panweidb=# CREATE TYPE compfoo_table AS TABLE OF compfoo;

--重命名数据类型。
panweidb=# ALTER TYPE compfoo RENAME TO compfoo1;

--要改变一个用户定义类型 compfoo1 的所有者为 usr1。
panweidb=# CREATE USER usr1 PASSWORD 'xxxxxxxxx';
panweidb=# ALTER TYPE compfoo1 OWNER TO usr1;

--把用户定义类型 compfoo1 的模式改变为 usr1。
panweidb=# ALTER TYPE compfoo1 SET SCHEMA usr1;

--给一个数据类型增加一个新的属性。
panweidb=# ALTER TYPE usr1.compfoo1 ADD ATTRIBUTE f3 int;

--删除 compfoo1 类型。
panweidb=# DROP TYPE usr1.compfoo1 cascade;

--删除相关表和用户。
panweidb=# DROP TABLE t1_compfoo;
panweidb=# DROP TABLE t2_compfoo;
panweidb=# DROP SCHEMA usr1;
panweidb=# DROP USER usr1;

--创建一个枚举类型。
panweidb=# CREATE TYPE bugstatus AS ENUM ('create', 'modify', 'closed');
```

```
--添加一个标签值。
panweidb=# ALTER TYPE bugstatus ADD VALUE IF NOT EXISTS 'regress' BEFORE 'clos
ed';

--重命名一个标签值。
panweidb=# ALTER TYPE bugstatus RENAME VALUE 'create' TO 'new';
```

相关链接

参考：SQL 语法参考->SQL 语法->ALTER TYPE、DROP TYPE 章节。

4.5.99.CREATE USER

功能描述

创建一个用户。

注意事项

- 通过 CREATE USER 创建的用户，默认具有 LOGIN 权限。
- 通过 CREATE USER 创建用户的同时，系统会在执行该命令的数据库中，为该用户创建一个同名的 SCHEMA。
- 系统管理员在普通用户同名 schema 下创建的对象，所有者为 schema 的同名用户（非系统管理员）。

语法格式

```
    CREATE USER [IF NOT EXISTS] user_name [ [ WITH ] option [ ... ] ] [ ENCRYPTED |
UNENCRYPTED ] { PASSWORD | IDENTIFIED BY } { 'password' [ EXPIRED ] | DISABLE };
```

其中 option 子句用于设置权限及属性等信息。

```
{SYSADMIN | NOSYSADMIN}
  | {MONADMIN | NOMONADMIN}
  | {OPRADMIN | NOOPRADMIN}
  | {POLADMIN | NOPOLADMIN}
  | {AUDITADMIN | NOAUDITADMIN}
  | {CREATEDB | NOCREATEDB}
```

```
| {USEFT | NOUSEFT}
| {CREATEROLE | NOCREATEROLE}
| {INHERIT | NOINHERIT}
| {LOGIN | NOLOGIN}
| {REPLICATION | NOREPLICATION}
| {INDEPENDENT | NOINDEPENDENT}
| {VCADMIN | NOVCADMIN}
| {PERSISTENCE | PERSISTENCE}
| CONNECTION LIMIT connlimit
| VALID BEGIN 'timestamp'
| VALID UNTIL 'timestamp'
| RESOURCE POOL 'respool'
| USER GROUP 'groupuser'
| PERM SPACE 'spacelimit'
| TEMP SPACE 'tmpspacelimit'
| SPILL SPACE 'spillspacelimit'
| NODE GROUP logic_cluster_name
| IN ROLE role_name [, ...]
| IN GROUP role_name [, ...]
| ROLE role_name [, ...]
| ADMIN role_name [, ...]
| USER role_name [, ...]
| SYSID uid
| DEFAULT TABLESPACE tablespace_name
| PROFILE DEFAULT
| PROFILE profile_name
| PGUSER
```

参数说明

- **user_name**

  用户名称。

  取值范围：字符串，要符合标识符的命名规范。且最大长度不超过 63 个字符。

若 GUC 参数 b_compatibility_user_host_auth 打开且在 B 数据库模式下，则支持以下特殊形式的语法：

- 'user_name' @ 'host_name'。例如：'user1'@'127.0.0.%'。
- 'user_name'，例如：'user1'。但是不允许包含特殊字符@，例如：'user1@127.0.0.%'。
- user_name@host_name。例如：user1@127.%.0.1。

❖ **password**

登录密码。

密码规则如下：

- 密码默认取值范围 8-32 个字符。
- 不能与用户名及用户名倒序相同。
- 至少包含大写字母（A-Z）、小写字母（a-z）、数字（0-9）、非字母数字字符（限定为~!@#$%^&*()-_=+|[{}];:,<.>/?）四类字符中的三类字符。
- 密码也可以是符合格式要求的密文字符串，这种情况主要用于用户数据导入场景，不推荐用户直接使用。如果直接使用密文密码，用户需要知道密文密码对应的明文，并且保证明文密码复杂度，数据库不会校验密文密码复杂度，直接使用密文密码的安全性由用户保证。
- 创建用户时，应当使用双引号或单引号将用户密码括起来。

取值范围：字符串。

❖ **groupuser**

用户组名称。

取值范围：字符串。

❖ **logic_cluster_name**

节点组名称。

取值范围：字符串。

CREATE USER 的其他参数值请参考：SQL 语法参考->SQL 语法->CREATE ROLE 章节。

示例

1、创建用户 jim，登录密码为 Aa123456。

方法 1：

```
CREATE USER jim PASSWORD 'Aa123456';
```

方法 2：

```
CREATE USER jim IDENTIFIED BY 'Aa123456';
```

2、如果创建有“创建数据库”权限的用户，则需要加 CREATEDB 关键字。

```
CREATE USER dim CREATEDB PASSWORD 'Aa123456';
```

3、将用户 jim 的登录密码由 Aa123456 修改为 Abcd@123。

```
ALTER USER jim IDENTIFIED BY 'Abcd@123' REPLACE 'Aa123456';
```

4、为用户 jim 追加 CREATEROLE 权限。

```
ALTER USER jim CREATEROLE;
```

5、将 enable_seqscan（可参考《PanWeiDB V2.0-S3.0.1_B01 开发者指南》->优化器方法配置）的值设置为 on， 设置成功后，在下一会话中生效。

```
ALTER USER jim SET enable_seqscan TO on;
```

6、重置 jim 的 enable_seqscan 参数。

```
ALTER USER jim RESET enable_seqscan;
```

7、锁定 jim 帐户

```
ALTER USER jim ACCOUNT LOCK;
```

8、删除用户。

```
DROP USER jim CASCADE;
DROP USER dim CASCADE;
```

相关链接

参考：SQL 语法参考->SQL 语法->ALTER USER、CREATE ROLE、DROP USER 章节。

4.5.100.CREATE USER MAPPING

功能描述

定义一个用户到一个外部服务器的新映射。

注意事项

当在 OPTIONS 中出现 password 选项时，需要保证 PanWeiDB 每个节点的$GAUSSHOME/bin 目录下存在 usermapping.key.cipher 和 usermapping.key.rand 文件，如果不存在这两个文件，请使用 pw_guc 工具生成并发布到 PanWeiDB 每个节点的$GAUSSHOME/bin 目录下。

语法格式

```
CREATE USER MAPPING FOR { user_name | USER | CURRENT_USER | PUBLIC }
  SERVER server_name
  [ OPTIONS ( option 'value' [ , ... ] ) ]
```

参数说明

- **user_name**

  要映射到外部服务器的一个现有用户的名称。

  CURRENT_USER 和 USER 匹配当前用户的名称。 当 PUBLIC 被指定时，一个公共映射会被创建，当没有特定用户的映射可用时将会使用它。

- **server_name**

  将为其创建用户映射的现有服务器的名称。

- **OPTIONS ({ option_name ' value ' } [, ...])**

  这个子句指定用户映射的选项。这些选项通常定义该映射实际的用户名和口令。选项名必须唯一。允许的选项名和值与该服务器的外部数据包装器有关。

  📖 **说明：**

  - 用户的口令会加密后保存到系统表 PG_USER_MAPPING 中，加密时需要使用 usermapping.key.cipher 和 usermapping.key.rand 作为加密密码文件和加密因子。首

次使用前需要通过如下命令创建这两个文件，并将这两个文件放入各节点目录$GAUSSHOME/bin，且确保具有读权限。

```
pw_guc generate -o usermapping -S default -D $GAUSSHOME/bin
```

- 其中-S 参数指定 default 时会随机生成密码，用户也可为-S 参数指定密码，此密码用于保证生成密码文件的安全性和唯一性，用户无需保存或记忆。其他参数请参考“工具参考中 pw_guc 工具说明”。
- oracle_fdw 支持的 options 包括：
  - ✧ user

    oracle server 的用户名。
  - ✧ password

    oracle 用户对应的密码。
- mysql_fdw 支持的 options 包括：
  - ✧ username

    MySQL Server/MariaDB 的用户名。
  - ✧ password

    MySQL Server/MariaDB 用户对应的密码。
- postgres_fdw 支持的 options 包括：
  - ✧ user

    远端 PanWeiDB 的用户名。
  - ✧ password

    远端 PanWeiDB 用户对应的密码。

    📖 **说明：**

    PanWeiDB 在后台会对用户输入的 password 加密以保证安全性。该加密所需密钥文件需要使用 pw_guc 工具生成，并被发布到 PanWeiDB 每个节点的 $GAUSSHOME/bin 目录下。password 不应当包含'encryptOpt'前缀，否则会被认为是加密后的密文。

相关链接

参考：SQL 语法参考->SQL 语法->ALTER USER MAPPING、DROP USER MAPPING 章节。

4.5.101.CREATE VIEW

功能描述

创建一个视图。视图与基本表不同，是一个虚拟的表。数据库中仅存放视图的定义，而不存放视图对应的数据，这些数据仍存放在原来的基本表中。若基本表中的数据发生变化，从视图中查询出的数据也随之改变。从这个意义上讲，视图就像一个窗口，透过它可以看到数据库中用户感兴趣的数据及变化。

注意事项

- 被授予 CREATE ANY TABLE 权限的用户，可以在 public 模式和用户模式下创建视图。
- 不可与同一模式下已存在的 synonym 产生命名冲突。

语法格式

```
    CREATE [ OR REPLACE ] [ TEMP | TEMPORARY ] [ FORCE | NOFORCE ] [DEFINER = user] [SQL SECURITY { DEFINER | INVOKER }]
    VIEW view_name [ ( column_name [, ...] ) ]
    [ WITH ( {view_option_name [= view_option_value]} [, ... ] ) ]
    AS query [ WITH {{ [ CASCADED | LOCAL ] CHECK OPTION } | READ ONLY}];
```

【须知】

- 创建视图时使用 WITH(security_barrier)可以创建一个相对安全的视图，避免攻击者利用低成本函数的 RAISE 语句打印出隐藏的基表数据。
- 当视图创建后，不允许使用 REPLACE 修改本视图当中的列名，也不允许删除列。

参数说明

- **OR REPLACE**

当 CREATE VIEW 语句中存在 OR REPLACE 时，表示若以前存在该视图就进行替换，但新查询不能改变原查询的列定义，包括顺序、列名、数据类型、类型精度等，只可在列表末尾添加其他的列。

- **[DEFINER = user]**

  指定 user 作为视图的属主。该选项仅在 MySQL 兼容模式下可用。

【须知】
支持指定 definer 为 current_user。

- **[SQL SECURITY { DEFINER | INVOKER }]**

  SQL SECURITY 语法可以为视图添加安全属性。

  该选项仅在 MySQL 兼容模式下可用，参考 MySQL 兼容性手册的 CREATE VIEW。

- **TEMP | TEMPORARY**

  创建临时视图。

- **view_name**

  要创建的视图名称。可以用模式修饰。

  取值范围：字符串，符合标识符命名规范。

- **column_name**

  可选的名称列表，用作视图的字段名。如果没有给出，字段名取自查询中的字段名。

  取值范围：字符串，符合标识符命名规范。

- **view_option_name [= view_option_value]**

  该子句为视图指定一个可选的参数。

  目前 view_option_name 支持的参数仅有 security_barrier，当 VIEW 试图提供行级安全时，应使用该参数。

  取值范围：Boolean 类型，TRUE、FALSE

- **query**

  为视图提供行和列的 SELECT 或 VALUES 语句。

- **WITH [CASCADED | LOCAL] CHECK OPTION**

该选项控制自动更新视图的行为，对视图的 insert 和 update，要检查确保新行满足视图定义的条件，即新行可以通过视图看到。如果没有通过检查，则拒绝修改。如果没有添加该选项，则允许通过对视图的 insert 和 update 来创建该视图不可见的行。支持下列检查选项：

- **LOCAL**

只检查视图本身直接定义的条件，除非底层视图也定义了 CHECK OPTION，否则它们定义的条件都不检查。

- **CASCADED**

检查该视图和所有底层视图定义的条件。如果仅声明了 CHECK OPTION，没有声明 LOCAL 和 CASCADED，默认是 CASCADED。

【须知】
只有在可自动更新、没有 INSTEAD OF 触发器或者 INSTEAD 规则的视图上才支持 CHECK OPTION。如果一个自动更新的视图被定义在一个具有 INSTEAD OF 触发器的视图上，那么 CHECK OPTION 可以被用来检查该自动更新视图上的条件，但具有 INSTEAD OF 触发器的视图上的条件不会被检查。如果该视图或者任何底层关系具有导致 INSERT 或 UPDATE 命令被重写的 INSTEAD 规则，那么在被重写的查询中将忽略所有检查选项，包括任何来自定义在有 STEAD 规则关系上的可自动更新视图的检查。
基于 MySQL 外表的视图不支持 CHECK OPTION 选项。

- **with read only**

指定此选项将被标记为只读视图。

此功能仅在 Oracle 兼容模式下可用，参见 Oracle 兼容性手册的 CREATE VIEW 创建只读视图。

示例

1、创建字段 spcname 为 pg_default 组成的视图。

```
CREATE VIEW myView AS SELECT * FROM pg_tablespace WHERE spcname =
'pg_default';
```

2、查看视图。

```
SELECT * FROM myView ;
```

返回结果如下：

```
 spcname  | spcowner | spcacl | spcoptions | spcmaxsize | relative |
spcthreshold
-----------+----------+--------+------------+------------+----------+-------------
-
 pg_default |    10 |      |         |         | f        |
(1 row)
```

3、删除视图 myView。

```
DROP VIEW myView;
```

相关链接

参考：SQL 语法参考->SQL 语法->ALTER VIEW、DROP VIEW 章节。

4.5.102.CREATE WEAK PASSWORD DICTIONARY

功能描述

向 gs_global_config 表中插入一个或者多个弱口令。

注意事项

- 未开启三权分立时，只有初始用户、系统管理员与安全管理员拥有权限执行本语法。启用三权分立时，只有初始用户和系统管理员有权限执行本语法。

- 弱口令字典中的口令存放在 gs_global_config 系统表中。
- 弱口令字典默认为空，用户通过本语法可以新增一条或多条弱口令。
- 当用户尝试通过本语法插入 gs_global_config 表中已存在的弱口令时，会只在表中保留一条该弱口令。

语法格式

```
CREATE WEAK PASSWORD DICTIONARY
    [WITH VALUES] ( {'weak_password'} [, ...] );
```

参数说明

weak_password

弱口令。

范围：字符串。

示例

```
--向 gs_global_config 系统表中插入单个弱口令。
panweidb=# CREATE WEAK PASSWORD DICTIONARY WITH VALUES ('password1');

--向 gs_global_config 系统表中插入多个弱口令。
panweidb=# CREATE WEAK PASSWORD DICTIONARY WITH VALUES ('password2'),('p
assword3');

--清空 gs_global_config 系统表中所有弱口令。
panweidb=# DROP WEAK PASSWORD DICTIONARY;

--查看现有弱口令。
panweidb=# SELECT * FROM gs_global_config WHERE NAME LIKE 'weak_password';
```

相关链接

参考：SQL 语法参考->SQL 语法->DROP WEAK PASSWORD DICTIONARY 章节。

4.5.103.CURSOR

功能描述

CURSOR 命令定义一个游标，用于在一个大的查询里面检索少数几行数据。

为了处理 SQL 语句，存储过程进程分配一段内存区域来保存上下文联系。游标是指向上下文区域的句柄或指针。借助游标，存储过程可以控制上下文区域的变化。

注意事项

- 游标命令只能在事务块里使用。
- 通常游标和 SELECT 一样返回文本格式。因为数据在系统内部是用二进制格式存储的，系统必须对数据做一定转换以生成文本格式。一旦数据是以文本形式返回，客户端应用需要把它们转换成二进制进行操作。使用 FETCH 语句，游标可以返回文本或二进制格式。
- 应该小心使用二进制游标。文本格式一般都比对应的二进制格式占用的存储空间大。二进制游标返回内部二进制形态的数据，可能更易于操作。如果想以文本方式显示数据，则以文本方式检索会为用户节约很多客户端的工作。比如，如果查询从某个整数列返回 1，在缺省的游标里将获得一个字符串 1，但在二进制游标里将得到一个 4 字节的包含该数值内部形式的数值（大端顺序）。

语法格式

```
CURSOR cursor_name
  [ BINARY ] [ INSENSITIVE ] [ [ NO ] SCROLL ]
  FOR query ;
```

参数说明

- **cursor_name**

  将要创建的游标名。

  取值范围：遵循数据库对象命名规范。

- **BINARY**

指明游标以二进制而不是文本格式返回数据。

- **INSENSITIVE**

  指定该选项，则表示定义一个游标，以创建为该游标所使用的数据临时副本。对游标的所有请求都从 tempdb 中的这一临时表中得到应答。

  因此，对基表所做的修改不会反映在对该游标进行的提取所返回的数据中，并且该游标不允许进行修改。

- **[NO] SCROLL**

  声明游标检索数据行的方式。

  - NO SCROLL：声明该游标不能用于以倒序的方式检索数据行。
  - 未声明：根据执行计划的不同，自动判断该游标是否可以用于以倒序的方式检索数据行。

【说明】

在 cmdsql 中，只支持添加 NO SCROLL 关键字或者不添加滚动选项，不支持添加 SCROLL 关键字。

在存储过程中，支持添加 NO SCROLL、SCROLL 关键字或者不添加滚动选项。显示游标中，只有申明为 NO SCROLL 的游标可以并行执行，建议将不需要用到倒序检索数据行方式的游标设置为 NO SCROLL。

- **query**

  使用 SELECT 或 VALUES 子句指定游标返回的行。

  取值范围：SELECT 或 VALUES 子句。

示例

参考：SQL 语法参考->SQL 语法->FETCH 章节的示例。

相关链接

参考：SQL 语法参考->SQL 语法->FETCH 章节。

4.5.104.DEALLOCATE

功能描述

DEALLOCATE 用于删除前面编写的预备语句。如果用户没有明确删除一个预备语句，那么它将在会话结束的时候被删除。

PREPARE 关键字总被忽略。

注意事项

无。

语法格式

```
DEALLOCATE [ PREPARE ] { name | ALL };
```

参数说明

- **name**

  将要删除的预备语句。

- **ALL**

  删除所有预备语句。

示例

无。

4.5.105.DELETE

功能描述

DELETE 从指定的表里删除满足 WHERE 子句的行。如果 WHERE 子句不存在，将删除表中所有行，结果只保留表结构。

注意事项

- 表的所有者、被授予了表 DELETE 权限的用户或被授予 DELETE ANY TABLE 权限的用户有权删除表中数据，系统管理员默认拥有此权限。同时也必须有 USING 子句引用的表以及 condition 上读取的表的 SELECT 权限。
- 对于列存表，暂时不支持 RETURNING 子句。
- 如果要删除表中的所有记录，建议使用 TRUNCATE 语法。

语法格式

```
[ WITH [ RECURSIVE ] with_query [, ...] ]
DELETE [/*+ plan_hint */] [FROM] [ ONLY ] table_name [ @dblink_name ] [ * ] [ [ [partit
ion_clause] [ [ AS ] alias ] ] | [ [ [ AS ] alias ] [ partitions_clause ] ] ]
  [ USING using_list ]
  [ WHERE condition | WHERE CURRENT OF cursor_name ]
  [ ORDER BY {expression [ [ ASC | DESC | USING operator ]
  [ LIMIT { count } ]
  [ RETURNING { * | { output_expr [ [ AS ] output_name ] } [, ...] } ];
```

多表删除：

【说明】

- ❖ 该语法仅在数据库兼容模式为 MySQL 时支持（即数据库实例初始化时指定 DBCOMPATIBILITY='B'）。
- ❖ 有关多表删除的详细内容请参考 MySQL 兼容性 DELETE。

```
[ WITH [ RECURSIVE ] with_query [, ...] ]
DELETE [/*+ plan_hint */] [FROM]
  {[ ONLY ] table_name [ * ] [ [ [partition_clause]  [ [ AS ] alias ] ] | [ [ [ AS ] alias ] [parti
tions_clause] ] ]} [, ...]
  [ USING using_list ]
  [ WHERE condition | WHERE CURRENT OF cursor_name ];
```

或者

```
[ WITH [ RECURSIVE ] with_query [, ...] ]
DELETE [/*+ plan_hint */]
  {[ ONLY ] table_name [ * ] [ [ [partition_clause]  [ [ AS ] alias ] ] | [ [ [ AS ] alias ] [parti
tions_clause] ] ]} [, ...]
  [ FROM using_list ]
  [ WHERE condition | WHERE CURRENT OF cursor_name ];
```

参数说明

- ❖ **WITH [RECURSIVE] with_query [, ...]**

用于声明一个或多个可以在主查询中通过名称引用的子查询，相当于临时表。

如果声明了 RECURSIVE，那么允许 SELECT 子查询通过名称引用它自己。

其中 with_query 的详细格式为：

```
with_query_name [ ( column_name [, ...] ) ] AS [ [ NOT ] MATERIALIZED]
  ( {select | values | insert | update | delete} )
```

with_query_name 指定子查询生成的结果集名称，在查询中可使用该名称访问子查询的结果集。

column_name 指定子查询结果集中显示的列名。

- with_query_name 指定子查询生成的结果集名称，在查询中可使用该名称访问子查询的结果集。
- column_name 指定子查询结果集中显示的列名。
- 每个子查询可以是 SELECT、VALUES、INSERT、UPDATE 或 DELETE 语句。
- 用户可以使用 MATERIALIZED / NOT MATERIALIZED 对 CTE 进行修饰。

  如果声明为 MATERIALIZED，WITH 查询将被物化，生成一个子查询结果集的拷贝，在引用处直接查询该拷贝，因此 WITH 子查询无法和主干 SELECT 语句进行联合优化（如谓词下推、等价类传递等），对于此类场景可以使用 NOT MATERIALIZED 进行修饰，如果 WITH 查询语义上可以作为子查询内联执行，则可以进行上述优化。

  如果用户没有显示声明物化属性则遵守以下规则：如果 CTE 只在所属主干语句中被引用一次，且语义上支持内联执行，则会被改写为子查询内联执行，否则以 CTE Scan 的方式物化执行。

❖ **plan_hint 子句**

以/*+* /的形式在 DELETE 关键字后，用于对 DELETE 对应的语句块生成的计划进行 hint 调优，详细用法请参见章节：《PanWeiDB V2.0-

S3.0.1_B01 管理员指南->性能调优->SQL 调优指南->使用 Plan-Hint 进行调优》章节。每条语句中只有第一个/*+ plan_hint */注释块会作为 hint 生效，里面可以写多条 hint。

- **ONLY**

  如果指定 ONLY 则只有该表被删除；如果没有声明，则该表和它的所有子表将都被删除。

- **table_name**

  目标表的名称（可以有模式修饰）。

  取值范围：已存在的表名。

- **partition_clause**

  指定分区删除操作

```
PARTITION { ( partition_name ) | FOR ( partition_value [, ...] ) } |
SUBPARTITION { ( subpartition_name ) | FOR ( subpartition_value [, ...] ) }
```

  关键字详见：SQL 语法参考->SQL 语法->SELECT 章节。

  示例详见：SQL 语法参考->SQL 语法->CREATE TABLE SUBPARTITION 章节。

- **partitions_clause**

  指定多个分区删除操作。

```
PARTITION { ( { partition_name | subpartition_name } [, ...] ) }
```

  【说明】

  该语法仅在数据库兼容模式为 MySQL 时支持（即数据库实例初始化时指定 DBCOMPATIBILITY='B'）。

  关键字详见：SQL 语法参考->SQL 语法->SELECT 章节。

  示例详见：SQL 语法参考->SQL 语法->CREATE TABLE SUBPARTITION 章节。

- **alias**

  目标表的别名。

取值范围：字符串，符合标识符命名规范。

- **using_list**

  using 子句。

  【说明】

  当参数 sql_compatibility='B'或删除多张目标表时，using_list 指定关联表的集合时可以同时出现目标表，并且可以定义表的别名并在目标表中使用。其他情况下则目标表不可重复出现在 using_list 中。

- **condition**

  一个返回 Boolean 值的表达式，用于判断哪些行需要被删除。不建议使用 int 等数值类型作为 condition，因为 int 等数值类型可以隐式转换为 bool 值（非 0 值隐式转换为 true，0 转换为 false），可能导致非预期的结果。

- **WHERE CURRENT OF cursor_name**

  当前不支持，仅保留语法接口。

- **ORDER BY 子句**

  关键字详见：SQL 语法参考->SQL 语法->SELECT 章节。

- **LIMIT 子句**

  关键字详见：SQL 语法参考->SQL 语法->SELECT 章节。

- **RETURNING**

  返回实际删除的行，RETURNING 列表的语法与 SELECT 的输出列表一致。

  ```
  [ RETURNING {* | {output_expression [ [ AS ] output_name ] }[, ...]} ]
  ```

- **output_expr**

  DELETE 命令删除行之后计算输出结果的表达式。该表达式可以使用表的任意字段。可以使用*返回被删除行的所有字段。

- **output_name**

  一个字段的输出名称。

  取值范围：字符串，符合标识符命名规范。

示例

1、 创建 customer_address 表。

```
CREATE TABLE customer_address
(
  ca_address_sk         integer           not null,
  ca_address_id         char(16)          not null,
  ca_street_number      char(10)                  ,
  ca_street_name        varchar(60)               ,
  ca_street_type        char(15)                  ,
  ca_suite_number       char(10)                  ,
  ca_city               varchar(60)               ,
  ca_county             varchar(30)               ,
  ca_state              char(2)                   ,
  ca_zip                char(10)                  ,
  ca_country            varchar(20)               ,
  ca_gmt_offset         decimal(5,2)              ,
  ca_location_type      char(20)
);
```

2、 创建基于 customer_address 表的 customer_address_bak 表。

```
CREATE TABLE customer_address_bak AS TABLE customer_address;
```

3、 执行删除操作。

删除 customer_address_bak 中 ca_address_sk 小于 14888 的职员。

```
DELETE FROM customer_address_bak WHERE ca_address_sk < 14888;
```

删除 customer_address_bak 中所有数据。

```
DELETE FROM customer_address_bak;
```

删除 customer_address_bak 表。

```
DROP TABLE customer_address_ba
```

4.5.106.DECLARE

功能描述

DECLARE 命令既可以定义一个游标，用于在一个大的查询里面检索少数几行数据，也可以作为一个匿名块的开始。

本节主要描述定义为游标的用法，开启匿名块的用法参考：SQL 语法参考->SQL 语法->BEGIN 章节。

为了处理 SQL 语句，存储过程进程分配一段内存区域来保存上下文联系。游标是指向上下文区域的句柄或指针。借助游标，存储过程可以控制上下文区域的变化。

通常游标和 SELECT 一样返回文本格式。因为数据在系统内部是用二进制格式存储的，系统必须对数据做一定转换以生成文本格式。一旦数据是以文本形式返回，客户端应用需要把它们转换成二进制进行操作。使用 FETCH 语句，游标可以返回文本或二进制格式。

注意事项

- 游标命令只能在事务块里使用。
- 应该小心使用二进制游标。文本格式一般都比对应的二进制格式占用的存储空间大。二进制游标返回内部二进制形态的数据，可能更易于操作。如果想以文本方式显示数据，则以文本方式检索会为用户节约很多客户端的工作。比如，如果查询从某个整数列返回 1，在缺省的游标里将获得一个字符串 1，但在二进制游标里将得到一个 4 字节的包含该数值内部形式的数值（大端顺序）。

语法格式

- 定义游标

```
DECLARE cursor_name [ BINARY ] [ NO SCROLL ]
  CURSOR [ { WITH | WITHOUT } HOLD ] FOR query ;
```

- 开启匿名块

```
[DECLARE [declare_statements]]
  BEGIN
```

```
execution_statements
END;
/
```

参数说明

- **cursor_name**

  将要创建的游标名。

  取值范围：遵循数据库对象命名规范。

- **BINARY**

  指明游标以二进制而不是文本格式返回数据。

- **NO SCROLL**

  声明游标检索数据行的方式。

  - NO SCROLL：声明该游标不能用于以倒序的方式检索数据行。
  - 未声明：根据执行计划的不同，自动判断该游标是否可以用于以倒序的方式检索数据行。

- **WITH HOLD**

  WITHOUT HOLD

  声明当创建游标的事务结束后，游标是否能继续使用。

  - WITH HOLD：声明该游标在创建它的事务结束后仍可继续使用。
  - WITHOUT HOLD：声明该游标在创建它的事务之外不能再继续使用，此游标将在事务结束时被自动关闭。
  - 如果不指定 WITH HOLD 或 WITHOUT HOLD，默认行为是 WITHOUT HOLD。

- **query**

  使用 SELECT 或 VALUES 子句指定游标返回的行。

  取值范围：SELECT 或 VALUES 子句。

- **declare_statements**

  声明变量，包括变量名和变量类型，如“sales_cnt int”。

- **execution_statements**

  匿名块中要执行的语句。

  取值范围：已存在的函数名称。

示例

定义游标示例请参考：SQL 语法参考->SQL 语法->FETCH 章节的示例。

相关链接

参考：SQL 语法参考->SQL 语法->BEGIN、FETCH 章节。

4.5.107.DO

功能描述

执行匿名代码块。

代码块被看做是没有参数的一段函数体，返回值类型是 void。它的解析和执行是同一时刻发生的。

注意事项

- 程序语言在使用之前，必须通过命令 CREATE LANGUAGE 安装到当前的数据库中。 plpgsql 是默认的安装语言，其他语言安装时必须指定。
- 如果语言是不受信任的，用户必须有使用程序语言的 USAGE 权限，或者是系统管理员。

语法格式

```
DO [ LANGUAGE lang_name ] code;
```

参数说明

- **lang_name**

  用来解析代码的程序语言的名称，如果缺省，默认的语言是 plpgsql。

- **code**

  程序语言代码可以被执行的。程序语言必须指定为字符串才行。

示例

```
--创建用户 webuser。
panweidb=# CREATE USER webuser PASSWORD 'xxxxxxxxx';

--授予用户 webuser 对模式 tpcds 下视图的所有操作权限。
```

```
panweidb=# DO $$DECLARE r record;
BEGIN
  FOR r IN SELECT c.relname table_name,n.nspname table_schema FROM pg_class
c,pg_namespace n
        WHERE c.relnamespace = n.oid AND n.nspname = 'tpcds' AND relkind IN ('r','
v')
  LOOP
    EXECUTE 'GRANT ALL ON ' || quote_ident(r.table_schema) || '.' || quote_ident(r.ta
ble_name) || ' TO webuser';
  END LOOP;
END$$;


--删除用户 webuser。
panweidb=# DROP USER webuser CASCADE;
```

4.5.108.DROP AGGREGATE

功能描述

删除一个聚合函数。

注意事项

DROP AGGREGATE 删除一个现存的聚合函数，执行这条命令的用户必须是该聚合函数的所有者。

语法格式

```
DROP AGGREGATE [ IF EXISTS ] name ( argtype [ , ... ] ) [ CASCADE | RESTRICT ]
```

参数说明

- **IF EXISTS**

  如果指定的聚合不存在，那么发出一个 notice 而不是抛出一个错误。

- **name**

  现存的聚合函数名（可以有模式修饰）。

- **argtype**

聚合函数操作的输入数据类型，要引用一个零参数聚合函数，请用*代替输入数据类型列表。

- **CASCADE**

  级联删除依赖于这个聚合函数的对象。

- **RESTRICT**

  如果有任何依赖对象，则拒绝删除这个聚合函数。这是缺省处理。

示例

将 integer 类型的聚合函数 myavg 删除：

```
DROP AGGREGATE myavg(integer);
```

兼容性

SQL 标准里没有 DROP AGGREGATE 语句。

4.5.109.DROP AUDIT POLICY

功能描述

删除一个审计策略。

注意事项

只有 poladmin、sysadmin 或初始用户才能进行此操作。

语法格式

```
DROP AUDIT POLICY [IF EXISTS] policy_name;
```

参数说明

policy_name

审计策略名称，需要唯一，不可重复。

取值范围：字符串，要符合标识符的命名规范。

示例

参考：SQL 语法参考->SQL 语法->CREATE AUDIT POLICY 章节的示例。

相关链接

参考：SQL 语法参考->SQL 语法->ALTER AUDIT POLICY、CREATE AUDIT POLICY 章节。

4.5.110.DROP CAST

功能描述

删除一个类型转换。

注意事项

DROP CAST 删除一个先前定义过的类型转换。

要能删除一个类型转换，你必须拥有源或者目的数据类型。这是和创建一个类型转换相同的权限。

语法格式

```
DROP CAST [ IF EXISTS ] (source_type AS target_type) [ CASCADE | RESTRICT ]
```

参数说明

- **IF EXISTS**

  如果指定的转换不存在，那么发出一个 notice 而不是抛出一个错误。

- **source_type**

  类型转换里的源数据类型。

- **target_type**

  类型转换里的目标数据类型。

- **CASCADE**

  RESTRICT

  这些键字没有任何效果，因为在类型转换上没有依赖关系。

示例

删除从 text 到 int 的转换：

```
DROP CAST (text AS int);
```

兼容性

DROP CAST 遵循 SQL 标准。

4.5.111.DROP CLIENT MASTER KEY

功能描述

删除一个客户端加密主密钥（CMK）。

注意事项

- 只有客户端加密主密钥所有者或者被授予了 DROP 权限的用户有权限执行命令，系统管理员默认拥有此权限。
- 该命令不仅删除数据库中的密钥对象，还会同时删除客户端指定路径下该密钥对象对应的密钥文件。

语法格式

```
DROP CLIENT MASTER KEY [ IF EXISTS ] client_master_key_name [CASCADE];
```

参数说明

- **IF EXISTS**

  如果指定的客户端加密主密钥不存在，则发出一个 notice 而不是抛出一个错误。

- **client_master_key_name**

  要删除的客户端加密主密钥名称。

  取值范围：字符串，已存在的客户端加密主密钥对象的名称。

- **CASCADE**

  - **CASCADE**：表示允许级联删除依赖于客户端加密主密钥的对象。
  - ⚠ **须知：**
  - 在执行本语法的生命周期中，同时需要客户端和服务端更改状态，发生异常时可能存在服务端已删除密钥信息，但客户端未删除密钥文件的情况。

➢此时，客户端并不会在执行下一条语法的生命周期中，检查是否有期望被删除但却因发生异常而未被删除的密钥文件，而是需要用户定期检查密钥文件夹，对未被使用的密钥文件进行确认并处理。

示例

```
--删除客户端加密主密钥对象。
panweidb=> DROP CLIENT MASTER KEY ImgCMK CASCADE;
NOTICE:  drop cascades to column setting: imgcek
DROP CLIENT MASTER KEY
```

4.5.112.DROP COLUMN ENCRYPTION KEY

功能描述

删除一个列加密密钥（CEK）。

注意事项

只有列加密密钥所有者或者被授予了 DROP 权限的用户有权限执行命令，系统管理员默认拥有此权限。

语法格式

```
DROP COLUMN ENCRYPTION KEY [ IF EXISTS ] column_encryption_key_name [CASCADE];
```

参数说明

- **IF EXISTS**

  如果指定的列加密密钥不存在，则发出一个 notice 而不是抛出一个错误。

- **column_encryption_key_name**

  要删除的列加密密钥名称。

  取值范围：字符串，已存在的列加密密钥名称。

示例

```
--删除列加密密钥。
panweidb=# DROP COLUMN ENCRYPTION KEY ImgCEK CASCADE;
```

```
ERROR:  cannot drop column setting: imgcek cascadely because encrypted column depend on it.
HINT:  we have to drop encrypted column: name, ... before drop column setting: imgcek cascadely.
```

4.5.113.DROP DATA SOURCE

功能描述

删除一个 Data Source 对象。

注意事项

只有属主/系统管理员/初始用户才可以删除一个 Data Source 对象。

语法格式

```
DROP DATA SOURCE [IF EXISTS] src_name [CASCADE | RESTRICT];
```

参数说明

- **src_name**

  待删除的 Data Source 对象名称。

  取值范围：字符串，符合标识符命名规范。

- **IF EXISTS**

  如果指定的 Data Source 不存在，则发出一个 notice 而不是报错。

- **CASCADE | RESTRICT**

  - **CASCADE**：表示允许级联删除依赖于 Data Source 的对象。
  - **RESTRICT**（缺省值）：表示有依赖于该 Data Source 的对象存在，则该 Data Source 无法删除。

  目前 Data Source 对象没有被依赖的对象，CASCADE 和 RESTRICT 效果一样，保留此选项是为了向后兼容性。

示例

```
--创建 Data Source 对象。
panweidb=# CREATE DATA SOURCE ds_tst1;
```

```
--删除 Data Source 对象。
panweidb=# DROP DATA SOURCE ds_tst1 CASCADE;
panweidb=# DROP DATA SOURCE IF EXISTS ds_tst1 RESTRICT;
```

相关链接

参考：SQL 语法参考->SQL 语法->CREATE DATA SOURCE、ALTER DATA SOURCE 章节。

4.5.114.DROP DATABASE

功能描述

删除一个数据库。

注意事项

- 只有数据库所有者或者被授予了数据库 DROP 权限的用户有权限执行 DROP DATABASE 命令，系统管理员默认拥有此权限。
- 不能对系统默认安装的三个数据库（POSTGRES、TEMPLATE0 和 TEMPLATE1）执行删除操作，系统做了保护。如果想查看当前服务中有哪几个数据库，可以用 gsql 的\l 命令查看。
- 如果有用户正在与要删除的数据库连接，则删除操作失败。
- 不能在事务块中执行 DROP DATABASE 命令。
- 如果执行 DROP DATABASE 失败，事务回滚，需要再次执行一次 DROP DATABASE IF EXISTS。

须知

DROP DATABASE 一旦执行将无法撤销，请谨慎使用。

语法格式

```
DROP DATABASE [ IF EXISTS ] database_name ;
```

参数说明

- **IF EXISTS**

如果指定的数据库不存在，则发出一个 notice 而不是抛出一个错误。

- **database_name**

  要删除的数据库名称。

  取值范围：字符串，已存在的数据库名称。

示例

参考：SQL 语法参考->SQL 语法->CREATE DATABASE 章节的示例。

相关链接

参考：SQL 语法参考->SQL 语法->CREATE DATABASE 章节。

4.5.115.DROP DATABASE LINK

4.5.116.DROP DIRECTORY

功能描述

删除指定的 directory 对象。

注意事项

- 当 enable_access_server_directory=off 时，只允许初始用户删除 directory 对象。
- 当 enable_access_server_directory=on 时，具有 SYSADMIN 权限的用户、directory 对象的属主、被授予了该 directory 的 DROP 权限的用户或者继承了内置角色 gs_rloe_directory_drop 权限的用户可以删除 directory。

语法格式

```
DROP DIRECTORY [ IF EXISTS ] directory_name;
```

参数说明

- **directory_name**

  目录名称。

  取值范围：已经存在的目录名。

示例

```
--创建目录。
panweidb=# CREATE OR REPLACE DIRECTORY  dir  as '/tmp/';

--删除目录。
panweidb=# DROP DIRECTORY dir;
```

相关链接

参考：SQL 语法参考->SQL 语法->CREATE DIRECTORY、ALTER DIRECTORY 章节。

功能描述

创建 DDL 触发器。

注意事项

只有超级用户才能创建 DDL 触发器。

在单用户模式下禁用 DDL 触发器。如果错误的 DDL 触发器禁用了数据库而无法取消触发器，请以单用户模式重新启动。

语法格式

```
CREATE EVENT TRIGGER name
  ON event [ ON DATABASE | SCHEMA ]
  [ WHEN filter_variable IN (filter_value [, ... ]) [ AND ... ] ]
  EXECUTE PROCEDURE function_name()

CREATE EVENT TRIGGER name
  { AFTER LOGON | BEFORE LOGOFF } ON DATABASE
  EXECUTE PROCEDURE function_name()
```

参数说明

- **name**

  DDL 触发器名称。在该数据库中这个名称必须唯一。

- **ON DATABASE**

表示 DDL 触发器绑定于 database 对象，LOGON 与 LOGOFF 事件仅支持指定 ON DATABASE。

- **ON SCHEMA**

  表示 DDL 触发器绑定于 schema 对象。

- **event**

  会触发对给定函数调用的事件名称，分别为 ddl_command_start、ddl_command_end、sql_drop、table_rewrite、LOGON 和 LOGOFF。

  - ddl_command_start：在 CREATE、ALTER、DROP、SECURITY LABEL、COMMENT、GRANT 或 REVOKE 命令执行前触发。
  - ddl_command_end： 在 CREATE、ALTER、DROP、SECURITY LABEL、COMMENT、GRANT 或 REVOKE 命令执行后触发。
  - sql_drop：为任何删除数据库对象的操作在 ddl_command_end DDL 触发器之前触发。
  - table_rewrite：在表被命令 ALTER TABLE 和 ALTER TYPE 的某些动作重写之前触发。
  - LOGON：用户登录后触发，仅支持 AFTER LOGON。
  - LOGOFF：用户登出前触发，仅支持 BEFORE LOGOFF。

- **filter_variable**

  用来过滤事件的变量名称。这可以用来限制触发器的触发事件。当前仅支持通过 TAG 过滤。

  TAG 是 SQL 命令执行后的客户端的返回值，例如执行 SQL 语句 CREATE TABLE fruit (id INTEGER)； ，则返回的 TAG 为 CREATE TABLE。

- **filter_value**

  与该触发器要为其引发的 filter_variable 相关联的一个值列表。对于 TAG 这表示一个命令标签列表（例如 DROP FUNCTION）。

- **function_name**

一个用户提供的函数。触发器函数需要声明为无参数，且返回类型为 Event Trigger 的函数。

示例

示例 1： 创建 ddl_command_start DDL 触发器，指定多条命令触发。

1、创建 DDL 触发器函数。

```
CREATE OR REPLACE FUNCTION eth1()
  RETURNS event_trigger
  LANGUAGE plpgsql
  AS $$ BEGIN
    RAISE NOTICE 'test_event_trigger: % %', tg_event, tg_tag;
  END;
$$;
```

2、创建 DDL 触发器。

```
 CREATE EVENT TRIGGER estart ON ddl_command_start WHEN TAG IN ('CREATE
TABLE','ALTER TABLE','DROP TABLE') EXECUTE PROCEDURE eth1();
3、执行 DDL 触发器中指定命令 CREATE TABLE。
 CREATE TABLE  t_eventtr(id int);
```

结果返回如下：

```
NOTICE:  test_event_trigger: ddl_command_start CREATE TABLE
CREATE TABLE
```

4、执行 DDL 触发器中指定命令 ALTER TABLE。

```
ALTER TABLE t_eventtr RENAME TO t_eventtr_new;
```

结果返回如下：

```
NOTICE:  test_event_trigger: ddl_command_start ALTER TABLE
ALTER TABLE
```

5、执行 DDL 触发器中指定命令 DROP TABLE。

```
DROP TABLE t_eventtr_new;
```

结果返回如下：

```
NOTICE:  test_event_trigger: ddl_command_start DROP TABLE
DROP TABLE
```

示例 2：创建 ddl_command_end DDL 触发器。

1、创建 DDL 触发器函数。

```
CREATE OR REPLACE FUNCTION eth1()
  RETURNS event_trigger
  LANGUAGE plpgsql
  AS $$ BEGIN
    RAISE NOTICE 'test_event_trigger: % %', tg_event, tg_tag;
  END;
$$;
```

2、创建测试表 t_eventtr。

```
CREATE TABLE t_eventtr(id int);
```

返回结果如下：

```
NOTICE:  test_event_trigger: ddl_command_start CREATE TABLE
CREATE TABLE
```

3、创建 DDL 触发器。

```
 CREATE EVENT TRIGGER eend ON ddl_command_end EXECUTE PROCEDURE  eth1();
```

4、删除测试表 t_eventtr。

```
DROP TABLE t_eventtr;
```

结果返回如下：

```
NOTICE:  test_event_trigger: ddl_command_end DROP TABLE
DROP TABLE
```

示例 3：创建 sql_drop DDL 触发器。

1、创建 DDL 触发器函数。

```
CREATE OR REPLACE FUNCTION test_event_trigger_drop()
  RETURNS event_trigger LANGUAGE plpgsql AS $$
  DECLARE
    obj record;
  BEGIN
    FOR obj IN SELECT * FROM pg_event_trigger_dropped_objects()
    LOOP
      RAISE NOTICE '% dropped object: % %.% %',
      tg_tag,
      obj.object_type,
      obj.schema_name,
      obj.object_name,
      obj.object_identity;
    END LOOP;
  END;
$$;
```

2、创建测试表 t_eventtr。

```
CREATE TABLE t_eventtr(id int);
```

3、创建 DDL 触发器 edrop。

```
CREATE EVENT TRIGGER edrop ON sql_drop EXECUTE PROCEDURE test_event_
trigger_drop();
```

4、删除表 t_eventtr。

```
DROP TABLE t_eventtr;
```

结果返回如下：

```
NOTICE:  DROP TABLE dropped object: table public.t_eventtr public.t_eventtr
 NOTICE:  DROP TABLE dropped object: type public.t_eventtr public.t_eventtr
 NOTICE:  DROP TABLE dropped object: type public._t_eventtr public.t_eventtr
[]
 DROP TABLE
```

示例 4： 创建 table_rewrite DDL 触发器。

1、创建 DDL 触发器函数。

```
CREATE OR REPLACE FUNCTION eth1()
   RETURNS event_trigger
   LANGUAGE plpgsql
   AS $$ BEGIN
     RAISE NOTICE 'test_event_trigger: % %', tg_event, tg_tag;
   END;
$$;
```

2、创建 DDL 触发器。

```
CREATE EVENT TRIGGER erewrite ON table_rewrite EXECUTE PROCEDURE eth1();
```

示例 5： 修改 DDL 触发器。

1、创建 DDL 触发器函数。

```
CREATE OR REPLACE FUNCTION eth1() RETURNS event_trigger AS $$ BEGIN RAIS
E NOTICE 'test_event_trigger: % %',tg_event,tg_tag; END $$ LANGUAGE plpgsql;
```

2、创建 DDL 触发器。

```
CREATE EVENT TRIGGER logon_event_trigger AFTER logon ON database EXECU
TE PROCEDURE eth1();
```

3、执行以下语句修改 DDL 触发器名称。

```
ALTER EVENT TRIGGER logon_event_trigger RENAME TO test_eventtri_new;
```

4、系统表中查询改名后的 DDL 触发器。

```
SELECT * FROM pg_event_trigger WHERE evtname = 'test_eventtri_new';
```

结果返回如下：

```
   evtname     | evtevent | evtowner | evtfoid | evtenabled | isschema | evttags
 ------------------+----------+----------+---------+------------+----------+-----
----
  test_eventtri_new | logon    |       10 |   18586 | O          | f        |
 (1 row)
```

示例 6：创建 Logon/Logoff 系统 DDL 触发器。

1、创建测试表 test2。

```
create table test2(id int,col timestamp);
```

2、创建触发器函数 eth3()和 eth4()。

```
CREATE OR REPLACE FUNCTION eth3()
  RETURNS event_trigger
  LANGUAGE plpgsql
  AS $$ BEGIN
    insert into test2 values(1,systimestamp);
  END;
$$;

CREATE OR REPLACE FUNCTION eth4()
  RETURNS event_trigger
  LANGUAGE plpgsql
  AS $$ BEGIN
    insert into test2 values(2,systimestamp);
  END;
$$;
```

3、创建 Logon/Logoff DDL 触发器。

```
  CREATE EVENT TRIGGER logon_eventtri after Logon ON database
    EXECUTE PROCEDURE eth3();

  CREATE EVENT TRIGGER logoff_eventtri before Logoff ON database
    EXECUTE PROCEDURE eth4();
```

4、退出数据库重新登陆。

```
\q
```

退出后在命令行执行登陆命令。

```
gsql -r
```

5、查询 test2 中记录的 Logon/Logoff 事件。

```
SELECT * FROM test2;
结果返回如下：
id |        col
----+---------------------
 2 | 2023-02-16 11:39:30
 1 | 2023-02-16 11:39:32
(2 rows)
```

相关链接

ALTER EVENT TRIGGER(参见：开发者指南->SQL 语法参考->SQL 语法->ALTER EVENT TRIGGER 章节),DROP EVENT TRIGGER(参见：开发者指南->SQL 语法参考->SQL 语法->DROP EVENT TRIGGER 章节)

4.5.117.DROP EVENT TRIGGER

功能描述

DROP EVENT TRIGGER 删除现有 DDL 触发器。

注意事项

要执行此命令，当前用户必须是 DDL 触发器的所有者。

语法格式

```
DROP EVENT TRIGGER [IF EXISTS ]name [CASCADE | RESTRICT]
```

参数说明

- **IF EXISTS**

  如果指定的 DDL 触发器不存在，则不会抛出一个错误而是返回一个提示。

- **name**

  要删除的 DDL 触发器的名称。

- **CASCADE**

  表示级联删除所有依赖于触发器的对象。

❖ **RESTRICT**

如果有任何对象依赖于该触发器，则拒绝删除。缺省为 RESTRICT。

相关链接

CREATE EVENT TRIGGER(参见：开发者指南->SQL 语法参考->SQL 语法->CREATE EVENT TRIGGER 章节),ALTER EVENT TRIGGER(参见：开发者指南->SQL 语法参考->SQL 语法->ALTER EVENT TRIGGER 章节)

4.5.118.DROP EXTENSION

功能描述

删除一个扩展。

注意事项

❖ DROP EXTENSION 命令从数据库中删除一个扩展。在删除扩展的过程中，构成扩展的组件也会一起删除。

❖ 必须是扩展的拥有者才能够使用 DROP EXTENSION 命令。

语法格式

```
DROP EXTENSION [ IF EXISTS ] name [, ...] [ CASCADE | RESTRICT ]
```

参数说明

❖ **IF EXISTS**

当使用 IF EXISTS 参数，如果扩展不存在时，不会抛出错误，而是产生一个通知。

❖ **name**

已经安装的扩展模块的名称。

❖ **CASCADE**

自动删除依赖于该扩展的对象。

❖ **RESTRICT**

如果有依赖于扩展的对象，则不允许删除次扩展（除非它所有的成员对象和其他扩展对象在一条 DROP 命令一起删除）。这是缺省行为。

示例

从当前数据库中删除扩展 hstore

```
DROP EXTENSION hstore;
```

说明：

在当前数据库中，如果有使用 hstore 的对象的，这条命令就会失败，比如任一表中的字段使用 hstore 类型。这时增加 CASCADE 选项会强制删除扩展和依赖于扩展的对象。

4.5.119.DROP FOREIGN DATA WRAPPER

功能描述

删除外部数据包装器。

注意事项

使用该功能需要开启 GUC 参数 support_extended_features，使用如下步骤开启该参数。

1、设置 support_extended_features 为 on。

```
ALTER SYSTEM SET support_extended_features TO on;
```

2、重启数据库。

```
pw_ctl restart
```

【说明】

support_extended_features

参数说明： 控制是否支持数据库的扩展特性。

该参数属于 POSTMASTER 类型参数。

取值范围： 布尔型

- on：表示支持数据库的扩展特性。
- off：表示不支持数据库的扩展特性。

默认值：off

语法格式

```
DROP FOREIGN DATA WRAPPER [ IF EXISTS ] name [ CASCADE | RESTRICT ]
```

参数说明

- **name**

  要删除的外部数据包装器名。

- **IF EXISTS**

  如果指定的外部数据包装器不存在，则发出一个 notice 而不是 error。

- **handler_function**

  handler_function 是先前注册的函数的名称，该函数将被调用以检索外部表的执行函数。

  处理器函数不能带任何参数，其返回类型必须是 fdw_handler。

- **CASCADE | RESTRICT**

  - CASCADE：自动删除依赖于外部数据包装器的对象（如服务器）。
  - RESTRICT：如果有任何依赖于外部数据包装器，则拒绝删除外部数据包装器。此选项为默认选项。

示例

1、设置 support_extended_features 为 on。

```
ALTER SYSTEM SET support_extended_features TO on;
```

2、重启数据库使参数生效。

3、创建外部数据包装器 dbi。

```
CREATE FOREIGN DATA WRAPPER dbi OPTIONS (debug 'true');
```

4、删除外部数据包装器 dbi。

```
DROP FOREIGN DATA WRAPPER dbi;
```

4.5.120.DROP FOREIGN TABLE

功能描述

删除指定的外表。

注意事项

DROP FOREIGN TABLE 会强制删除指定的表，删除表后，依赖该表的索引会被删除，因此引用该表的函数和存储过程将无法执行。

语法格式

```
DROP FOREIGN TABLE [ IF EXISTS ]
  table_name [, ...] [ CASCADE | RESTRICT ];
```

参数说明

- **IF EXISTS**

  如果指定的表不存在，则发出一个 notice 而不是抛出一个错误。

- **table_name**

  表名称。

  取值范围：已存在的表名。

- **CASCADE | RESTRICT**
  - CASCADE：级联删除依赖于表的对象（比如视图）。
  - RESTRICT：如果存在依赖对象，则拒绝删除该表。这个是缺省。

相关链接

参考：SQL 语法参考->SQL 语法->ALTER FOREIGN TABLE、CREATE FOREIGN TABLE 章节。

4.5.121.DROP FUNCTION

功能描述

删除一个已存在的函数。

注意事项

- 如果函数中涉及对临时表相关操作，则无法使用 DROP FUNCTION 删除函数。
- 只有函数的所有者或者被授予了函数 DROP 权限的用户才能执行 DROP FUNCTION 命令，系统管理员默认拥有该权限。
- 通过 DROP PROCEDURE 语法也能删除函数。

语法格式

```
DROP FUNCTION [ IF EXISTS ] function_name
[ ( [ {[ argname ] [ argmode ] argtype} [, ...] ] ) [ CASCADE | RESTRICT ] ];
```

参数说明

- **IF EXIST**

  IF EXISTS 表示，如果函数存在则执行删除操作，函数不存在也不会报错，只是发出一个 notice。

- **function_name**

  要删除的函数名称。

  取值范围：已存在的函数名。

- **argmode**

  函数参数的模式。

- **argname**

  函数参数的名称。

- **argtype**

  函数参数的类型。

示例

参考：SQL 语法参考->SQL 语法->CREATE-FUNCTION 章节的示例。

相关链接

参考：SQL 语法参考->SQL 语法->ALTER FUNCTION、CREATE FUNCTION 章节。

4.5.122.DROP GLOBAL CONFIGURATION

功能描述

删除系统表 gs_global_config 中的参数值。

注意事项

- 仅支持数据库初始用户运行此命令。
- 不支持删除关键字为 weak_password。

语法格式

```
DROP GLOBAL CONFIGURATION 参数名称，参数名称...;
```

参数说明

参数名称是 gs_global_config 中已经存在的参数，删除不存在的参数将报错。

4.5.123.DROP GROUP

功能描述

删除用户组。

DROP GROUP 是 DROP ROLE 的别名。

注意事项

DROP GROUP 是 PanWeiDB 管理工具封装的接口，用来实现 PanWeiDB 管理。该接口不建议用户直接使用，以免对 PanWeiDB 状态造成影响。

语法格式

```
DROP GROUP [ IF EXISTS ] group_name [, ...];
```

参数说明

参考：SQL 语法参考->SQL 语法->DROP ROLE 章节的参数说明。

相关链接

参考：SQL 语法参考->SQL 语法->CREATE GROUP、ALTER GROUP、DROP ROLE 章节。

4.5.124.DROP INDEX

功能描述

删除索引。

注意事项

只有索引的所有者或者拥有索引所在表的 INDEX 权限的用户有权限执行 DROP INDEX 命令，系统管理员默认拥有此权限。

对于全局临时表，当某个会话已经初始化了全局临时表对象（包括创建全局临时表和第一次向全局临时表内插入数据）时，其他会话不能够执行该表上索引的删除操作。

语法格式

```
DROP INDEX [ CONCURRENTLY ] [ IF EXISTS ]
  index_name [, ...] [ CASCADE | RESTRICT ];
```

参数说明

- **CONCURRENTLY**

  以不加锁的方式删除索引。删除索引时，一般会阻塞其他语句对该索引所依赖表的访问。加此关键字，可实现删除过程中不做阻塞。

  此选项只能指定一个索引的名称，并且 CASCADE 选项不支持。

  普通 DROP INDEX 命令可以在事务内执行，但是 DROP INDEX CONCURRENTLY 不可以在事务内执行。

- **IF EXISTS**

  如果指定的索引不存在，则发出一个 notice 而不是抛出一个错误。

- **index_name**

  要删除的索引名。

  取值范围：已存在的索引。

- **CASCADE | RESTRICT**
  - CASCADE：表示允许级联删除依赖于该索引的对象。
  - RESTRICT（缺省值）：表示有依赖与此索引的对象存在，则该索引无法被删除。

示例

参考：SQL 语法参考->SQL 语法->CREATE INDEX 章节的示例。

相关链接

参考：SQL 语法参考->SQL 语法->ALTER INDEX、CREATE INDEX 章节。

4.5.125.DROP MASKING POLICY

功能描述

删除脱敏策略。

注意事项

只有 poladmin、sysadmin 或初始用户才能执行此操作。

语法格式

```
DROP MASKING POLICY [IF EXISTS] policy_name;
```

参数说明

policy_name

审计策略名称，需要唯一，不可重复。

取值范围：字符串，要符合标识符的命名规范。

示例

```
--删除一个脱敏策略。
panweidb=# DROP MASKING POLICY IF EXISTS maskpol1;

--删除一组脱敏策略。
panweidb=# DROP MASKING POLICY IF EXISTS maskpol1, maskpol2, maskpol3;
```

相关链接

参考：SQL 语法参考->SQL 语法->ALTER MASKING POLICY、CREATE MASKING POLICY 章节。

4.5.126.DROP MATERIALIZED VIEW

功能描述

强制删除数据库中已有的物化视图。

注意事项

物化视图的所有者、物化视图所在模式的所有者、被授予了物化视图 DROP 权限的用户或拥有 DROP ANY TABLE 权限的用户才有权限执行 DROP MATERIALIZED VIEW 命令，系统管理员默认拥有此权限。

语法格式

```
DROP MATERIALIZED VIEW [ IF EXISTS ] mv_name [, ...] [ CASCADE | RESTRICT ];
```

参数说明

- **IF EXISTS**

  如果指定的物化视图不存在，则发出一个 notice 而不是抛出一个错误。

- **mv_name**

  要删除的物化视图名称。

- **CASCADE | RESTRICT**
  - CASCADE：级联删除依赖此物化视图的对象。
  - RESTRICT：如果有依赖对象存在，则拒绝删除此物化视图。此选项为缺省值。

示例

```
--删除名为 my_mv 的物化视图。
panweidb=# DROP MATERIALIZED VIEW my_mv;
```

相关链接

参考：SQL 语法参考->SQL 语法->ALTER MATERIALIZED VIEW、CREATE INCREMENTAL MATERIALIZED VIEW、CREATE MATERIALIZED VIEW、REFRESH INCREMENTAL MATERIALIZED VIEW、REFRESH MATERIALIZED VIEW 章节。

4.5.127.DROP MODEL

功能描述

删除一个已训练完成保存的模型对象。

注意事项

所删除模型可在系统表 gs_model_warehouse 中查看到。

语法格式

```
DROP MODEL model_name;
```

参数说明

model_name

模型名称。

取值范围：字符串，需要符合标识符的命名规范。

相关链接

参考：SQL 语法参考->SQL 语法->CREATE MODEL、PREDICT BY 章节。

4.5.128.DROP OPERATOR

PanWeiDB 不支持 drop operator 功能。

4.5.129.DROP OWNED

功能描述

删除一个数据库角色所拥有的数据库对象。

注意事项

- 所有该角色在当前数据库里和共享对象（数据库、表空间）上的所有对象上的权限都将被撤销。
- DROP OWNED 常常被用来为移除一个或者多个角色做准备。因为 DROP OWNED 只影响当前数据库中的对象，通常需要在包含将被移除角色所拥有的对象的每一个数据库中都执行这个命令。
- 使用 CASCADE 选项可能导致这个命令递归去删除由其他用户所拥有的对象。
- 角色所拥有的数据库、表空间将不会被移除。

语法格式

```
DROP OWNED BY name [, ...] [ CASCADE | RESTRICT ];
```

参数说明

- **name**

  角色名。

- **CASCADE | RESTRICT**
  - CASCADE：级联删除所有依赖于被删除对象的对象。
  - RESTRICT（缺省值）：拒绝删除那些有任何依赖对象存在的对象。

相关链接

参考：SQL 语法参考->SQL 语法->REASSIGN OWNED、DROP ROLE 章节。

4.5.130.DROP PACKAGE

功能描述

删除已存在的 PACKAGE 或者 PACKAGE BODY。

注意事项

删除 PACKAGE BODY 后，PACKAGE 内的存储过程及函数会同时失效。

语法格式

```
DROP PACKAGE [ IF EXISTS ] package_name;
DROP PACKAGE BODY [ IF EXISTS ] package_name;
```

4.5.131.DROP PACKAGE BODY

功能描述

删除已存在的 PACKAGE BODY。

注意事项

删除 PACKAGE BODY 后，PACKAGE 内的存储过程及函数会同时失效。

语法格式

```
DROP PACKAGE BODY [ IF EXISTS ] package_name;
```

参数说明

package_name

已存在的包名。

示例

1、创建一个 CREATE PACKAGE SPECIFICATION。

```
CREATE OR REPLACE PACKAGE emp_bonus IS
var1 int:=1;                                --公有变量
var2 int:=2;
PROCEDURE testpro1(var3 int);--公有存储过程，可以被外部调用
END ;
/
```

2、创建一个 package body。

```
create or replace package body emp_bonus is
var3 int:=3;
var4 int:=4;
procedure testpro1(var3 int)
is
begin
create table if not exists test1(col1 int);
insert into test1 values(var1);
insert into test1 values(var4);
end;
begin  #实例化开始
var4:=9;
testpro1(var4);
end;
end;
/
```

3、删除 package body。

```
drop package body emp_bonus;
```

相关链接

DROP PACKAGE（参考：开发者指南->SQL 语法参考->SQL 语法->DROP PACKAGE 章节）

4.5.132.DROP PROCEDURE

功能描述

删除已存在的存储过程。

注意事项

通过 DROP FUNCTION 也能删除存储过程。

语法格式

```
DROP PROCEDURE [ IF EXISTS ] procedure_name ;
```

参数说明

- **IF EXISTS**

  如果指定的存储过程不存在，发出一个 notice 而不是抛出一个错误。

- **procedure_name**

  要删除的存储过程名称。

  取值范围：已存在的存储过程名。

相关链接

参考：SQL 语法参考->SQL 语法->CREATE PROCEDURE 章节。

4.5.133.DROP PUBLICATION

功能描述

从数据库中删除一个现有的发布。

注意事项

发布只能被其属主或系统管理员删除。

语法格式

```
DROP PUBLICATION [ IF EXISTS ] name [ CASCADE | RESTRICT ]
```

参数说明

- **IF EXISTS**

  如果发布不存在，不要抛出一个错误，而是发出一个提示。

- **name**

  现有发布的名称。

- **CASCADE|RESTRICT**

  当前这些关键词没有任何作用，因为发布没有依赖关系。

示例

参考：SQL 语法参考->SQL 语法->CREATE-PUBLICATION 章节的示例。

相关链接

参考：SQL 语法参考->SQL 语法->ALTER PUBLICATION、CREATE PUBLICATION 章节。

4.5.134.DROP RESOURCE LABEL

功能描述

删除资源标签。

注意事项

只有 poladmin、sysadmin 或初始用户才能执行此操作。

语法格式

```
DROP RESOURCE LABEL [IF EXISTS] policy_name[, ...]*;
```

参数说明

label_name

资源标签名称；

取值范围：字符串，要符合标识符的命名规范。

示例

```
--删除一个资源标签。
panweidb=# DROP RESOURCE LABEL IF EXISTS res_label1;

--删除一组资源标签。
panweidb=# DROP RESOURCE LABEL IF EXISTS res_label1, res_label2, res_label3;
```

相关链接

参考：SQL 语法参考->SQL 语法->ALTER RESOURCE LABEL、CREATE RESOURCE LABEL 章节。

4.5.135.DROP ROLE

功能描述

删除指定的角色。

语法格式

```
DROP ROLE [ IF EXISTS ] role_name [, ...];
```

参数说明

- **IF EXISTS**

  如果指定的角色不存在，则发出一个 notice 而不是抛出一个错误。

- **role_name**

  要删除的角色名称。

  取值范围：已存在的角色。

注意事项

只允许删除同类型管理员属性的用户。

示例

1、创建测试用户 utest。

```
CREATE ROLE utest IDENTIFIED BY 'Bigdata@123';
```

2、删除测试用户。

```
DROP ROLE utest;
```

4.5.136.DROP ROW LEVEL SECURITY POLICY

功能描述

删除表上某个行访问控制策略。

注意事项

仅表的所有者或者管理员用户才能删除表的行访问控制策略。

语法格式

```
DROP [ ROW LEVEL SECURITY ] POLICY [ IF EXISTS ] policy_name ON table_name [ CA
SCADE | RESTRICT ]
```

参数说明

- **IF EXISTS**

  如果指定的行访问控制策略不存在，发出一个 notice 而不是抛出一个错误。

- **policy_name**

  要删除的行访问控制策略的名称。

  - table_name

  ➢行访问控制策略所在的数据表名。

  - CASCADE/RESTRICT

  ➢仅适配此语法，无对象依赖于该行访问控制策略，CASCADE 和 RESTRICT 效果相同。

示例

```
--创建数据表 all_data
panweidb=# CREATE TABLE all_data(id int, role varchar(100), data varchar(100));

--创建行访问控制策略
panweidb=# CREATE ROW LEVEL SECURITY POLICY all_data_rls ON all_data USING(r
ole = CURRENT_USER);
```

```
--删除行访问控制策略
panweidb=# DROP ROW LEVEL SECURITY POLICY all_data_rls ON all_data;
```

相关链接

参考：SQL 语法参考->SQL 语法->ALTER ROW LEVEL SECURITY POLICY、CREATE ROW LEVEL SECURITY POLICY 章节。

4.5.137.DROP RULE

功能描述

删除一个重写规则。

语法格式

```
DROP RULE [ IF EXISTS ] name ON table_name [ CASCADE | RESTRICT ]
```

参数说明

- **IF EXISTS**

  如果该规则不存在，会抛出一个 NOTICE。

- **name**

  要删除的现存规则名称。

- **table_name**

  该规则应用的表名。

- **CASCADE**

  自动级联删除依赖于此规则的对象。

- **RESTRICT**

  缺省情况下，如果有任何依赖对象，则拒绝删除此规则。

示例

```
--删除重写规则 newrule
DROP RULE newrule ON mytable;
```

4.5.138.DROP SCHEMA

功能描述

从数据库中删除模式。

注意事项

只有模式的所有者或者被授予了模式 DROP 权限的用户有权限执行 DROP SCHEMA 命令，系统管理员默认拥有此权限。

语法格式

```
DROP SCHEMA [ IF EXISTS ] schema_name [, ...] [ CASCADE | RESTRICT ];
```

参数说明

- **IF EXISTS**

  如果指定的模式不存在，发出一个 notice 而不是抛出一个错误。

- **schema_name**

  模式的名称。

  取值范围：已存在模式名。

- **CASCADE | RESTRICT**
  - CASCADE：自动删除包含在模式中的对象。
  - RESTRICT：如果模式包含任何对象，则删除失败（缺省行为）。

【须知】

不要随意删除 pg_temp 或 pg_toast_temp 开头的模式，这些模式是系统内部使用的，如果删除，可能导致无法预知的结果。

【说明】

无法删除当前模式。如果要删除当前模式，须切换到其他模式下。

示例

参考：SQL 语法参考->SQL 语法->CREATE SCHEMA 章节的示例。

相关链接

参考：SQL 语法参考->SQL 语法->ALTER SCHEMA、CREATE SCHEMA 章节。

4.5.139.DROP SECURITY LABEL

功能描述

删除一个安全标签。

注意事项

无

语法格式

```
DROP SECURITY LABEL label_name;
```

参数说明

label_name

安全标签名称，删除时要求标签存在。

示例

参考：SQL 语法参考->SQL 语法->CREATE SECURITYLABEL 章节的示例。

相关链接

CREATE SECURITYLABEL（参考：SQL 语法参考->SQL 语法->CREATE SECURITYLABEL 章节）

4.5.140.DROP SEQUENCE

功能描述

从当前数据库里删除序列。

注意事项

- 序列的所有者、序列所在模式的所有者或者被授予了序列 DROP 权限的用户或者被授予了 DROP ANY SEQUENCE 权限的用户才能删除，系统管理员默认拥有该权限。
- 如果 SEQUENCE 被创建时使用了 LARGE 标识，DROP 时也需要使用 LARGE 标识。

语法格式

```
DROP [ LARGE ] SEQUENCE [ IF EXISTS ] {[schema.]sequence_name} [ , ... ] [ CASCADE | RESTRICT ];
```

参数说明

- **IF EXISTS**

  如果指定的序列不存在，则发出一个 notice 而不是抛出一个错误。

- **name**

  序列名称。

- **CASCADE**

  级联删除依赖序列的对象。

- **RESTRICT**

  如果存在任何依赖的对象，则拒绝删除序列。此项是缺省值。

示例

```
--创建一个名为 serial 的递增序列，从 101 开始。
panweidb=# CREATE SEQUENCE serial START 101;

--删除序列。
panweidb=# DROP SEQUENCE serial;
```

相关链接

参考：SQL 语法参考->SQL 语法->ALTER SEQUENCE、CREATE SEQUENCE 章节。

4.5.141.DROP SERVER

功能描述

删除现有的一个数据服务器。

注意事项

只有 server 的所有者或者被授予了 server 的 DROP 权限的用户才可以删除，系统管理员默认拥有该权限。

语法格式

```
DROP SERVER [ IF EXISTS ] server_name [ {CASCADE | RESTRICT} ] ;
```

参数描述

- **IF EXISTS**

  如果指定的数据服务器不存在，则发出一个 notice 而不是抛出一个错误。

- **server_name**

  服务器名称。

- **CASCADE | RESTRICT**
  - CASCADE：级联删除依赖于 server 的对象 。
  - RESTRICT（缺省值）：如果存在依赖对象，则拒绝删除该 server。

相关链接

参考：SQL 语法参考->SQL 语法->ALTER SERVER、CREATE SERVER 章节。

4.5.142.DROP SUBSCRIPTION

功能描述

删除数据库实例中的一个订阅。

注意事项

- 只有系统管理员才可以删除订阅。

- 如果该待删除订阅与复制槽相关联，就不能在事务块内部执行 DROP SUBSCRIPTION。

语法格式

```
DROP SUBSCRIPTION [ IF EXISTS ] name [ CASCADE | RESTRICT ]
```

参数说明

- **name**

  要删除的订阅的名称。

- **CASCADE|RESTRICT**

  当前这些关键词没有任何作用，因为订阅没有依赖关系。

示例

参考：SQL 语法参考->SQL 语法->CREATE SUBSCRIPTION 章节的示例。

相关链接

参考：SQL 语法参考->SQL 语法->ALTER SUBSCRIPTION、CREATE SUBSCRIPTION 章节。

4.5.143.DROP SYNONYM

功能描述

删除指定的 SYNONYM 对象。

注意事项

SYNONYM 的所有者或者被授予了 DROP ANY SEQUENCE 权限的用户有权限执行 DROP SYNONYM 命令，系统管理员默认拥有此权限。

语法格式

```
DROP SYNONYM [ IF EXISTS ] synonym_name [ CASCADE | RESTRICT ];
```

参数描述

- **IF EXISTS**

  如果指定的同义词不存在，则发出一个 notice 而不是抛出一个错误。

- **synonym_name**

  同义词名字，可以带模式名。

- **CASCADE | RESTRICT**
  - CASCADE：级联删除依赖同义词的对象（比如视图）。
  - RESTRICT：如果有依赖对象存在，则拒绝删除同义词。此选项为缺省值。

示例

参考：SQL 语法参考->SQL 语法->CREATE SYNONYM 章节的示例。

相关链接

参考：SQL 语法参考->SQL 语法->ALTER SYNONYM、CREATE SYNONYM 章节。

4.5.144.DROP TABLE

功能描述

删除指定的表。

注意事项

- DROP TABLE 会强制删除指定的表，删除表后，依赖该表的索引会被删除，而使用到该表的函数和存储过程将无法执行。删除分区表，会同时删除分区表中的所有分区。
- 表的所有者、被授予了表的 DROP 权限的用户或被授予 DROP ANY TABLE 权限的用户，有权删除指定表 ，系统管理员默认拥有该权限。

语法格式

```
DROP TABLE [ IF EXISTS ]
  { [schema.]table_name } [, ...] [ CASCADE | RESTRICT ] [ PURGE ];
```

参数说明

- **IF EXISTS**

  如果指定的表不存在，则发出一个 notice 而不是抛出一个错误。

- **schema**

  模式名称。

- **table_name**

  表名称。

- **CASCADE | RESTRICT**

  - CASCADE：级联删除依赖于表的对象（比如视图）。
  - RESTRICT（缺省项）：如果存在依赖对象，则拒绝删除该表。这个是缺省。

- **PURGE**

  该参数表示即使开启回收站功能，DROP 表时，也会直接物理删除表，而不是将其放入回收站中。

示例

参考：SQL 语法参考->SQL 语法->CREATE TABLE 章节中的示例。

相关链接

参考：SQL 语法参考->SQL 语法->ALTER TABLE、CREATE TABLE 章节。

4.5.145.DROP TABLESPACE

功能描述

删除一个表空间。

注意事项

- 只有表空间所有者或者被授予了表空间 DROP 权限的用户有权限执行 DROP TABLESPACE 命令，系统管理员默认拥有此权限。
- 在删除一个表空间之前，表空间里面不能有任何数据库对象，否则会报错。
- DROP TABLESPACE 不支持回滚，因此，不能出现在事务块内部。
- 执行 DROP TABLESPACE 操作时，如果有另外的会话执行 db 查询操作，可能会由于 tablespace 事务的原因导致查询失败，请重新执行 db 查询操作。

- 如果执行 DROP TABLESPACE 失败，需要再次执行一次 DROP TABLESPACE IF EXISTS。

语法格式

```
DROP TABLESPACE [ IF EXISTS ] tablespace_name;
```

参数说明

- **IF EXISTS**

  如果指定的表空间不存在，则发出一个 notice 而不是抛出一个错误。

- **tablespace_name**

  表空间的名称。

  取值范围：已存在的表空间的名称。

示例

参考：SQL 语法参考->SQL 语法->CREATE TABLESPACE 章节中的示例。

相关链接

参考：SQL 语法参考->SQL 语法->ALTER TABLESPACE、CREATE TABLESPACE 章节。

4.5.146.DROP TEXT SEARCH CONFIGURATION

功能描述

删除已有文本搜索配置。

注意事项

要执行这个命令，用户必须是该配置的所有者。

语法格式

```
DROP TEXT SEARCH CONFIGURATION [ IF EXISTS ] name [ CASCADE | RESTRICT ];
```

参数说明

- **IF EXISTS**

如果指定的文本搜索配置不存在，那么发出一个 notice 而不是抛出一个错误。

- **name**

  要删除的文本搜索配置名称（可有模式修饰）。

- **CASCADE**

  级联删除依赖文本搜索配置的对象。

- **RESTRICT**

  若有任何对象依赖文本搜索配置则拒绝删除它。这是默认情况。

示例

参考：SQL 语法参考->SQL 语法->CREATE TEXT SEARCH CONFIGURATION 章节中的示例。

相关链接

参考：SQL 语法参考->SQL 语法->ALTER TEXT SEARCH CONFIGURATION、CREATE TEXT SEARCH CONFIGURATION 章节。

4.5.147.DROP TEXT SEARCH DICTIONARY

功能描述

删除全文检索词典。

注意事项

- 预定义词典不支持 DROP 操作。
- 只有词典的所有者可以执行 DROP 操作，系统管理员默认拥有此权限。
- 谨慎执行 DROP...CASCADE 操作，该操作将级联删除使用该词典的文本搜索配置（TEXT SEARCH CONFIGURATION）。

语法格式

```
DROP TEXT SEARCH DICTIONARY [ IF EXISTS ] name [ CASCADE | RESTRICT ]
```

参数说明

- **IF EXISTS**

如果指定的全文检索词典不存在，那么发出一个 Notice 而不是报错。

- **name**

  要删除的词典名称（可指定模式名，否则默认在当前模式下）。

  取值范围：已存在的词典名。

- **CASCADE**

  自动删除依赖于该词典的对象，并依次删除依赖于这些对象的所有对象。

  如果存在任何一个使用该词典的文本搜索配置，此 DROP 命令将不会成功。

  可添加 CASCADE 以删除引用该词典的所有文本搜索配置以及词典。

- **RESTRICT**

  如果任何对象依赖词典，则拒绝删除该词典。这是缺省值。

示例

```
--删除词典 english
panweidb=# DROP TEXT SEARCH DICTIONARY english;
```

相关链接

参考：SQL 语法参考->SQL 语法->ALTER TEXT SEARCH DICTIONARY、CREATE TEXT SEARCH DICTIONARY 章节。

4.5.148.DROP TRIGGER

功能描述

删除触发器。

注意事项

触发器的所有者或者被授予了 DROP ANY TRIGGER 权限的用户可以执行 DROP TRIGGER 操作，系统管理员默认拥有此权限。

语法格式

```
DROP TRIGGER [ IF EXISTS ] trigger_name ON table_name [ CASCADE | RESTRICT ];
```

参数说明

- **IF EXISTS**

如果指定的触发器不存在，则发出一个 notice 而不是抛出一个错误。

- **trigger_name**

  要删除的触发器名称。

  取值范围：已存在的触发器。

- **table_name**

  要删除的触发器所在的表名称。

  取值范围：已存在的含触发器的表。

- **CASCADE | RESTRICT**
  - CASCADE：级联删除依赖此触发器的对象。
  - RESTRICT：如果有依赖对象存在，则拒绝删除此触发器。此选项为缺省值。

示例

参考：SQL 语法参考->SQL 语法->CREATE TRIGGER 章节中的示例。

相关链接

参考：SQL 语法参考->SQL 语法->CREATE TRIGGER、ALTER TRIGGER、ALTER TABLE 章节。

4.5.149.DROP TYPE

功能描述

删除一个用户定义的数据类型。

注意事项

类型的所有者或者被授予了类型 DROP 权限的用户或者被授予了 DROP ANY TYPE 权限的用户有权限执行 DROP TYPE 命令，系统管理员默认拥有此权限。

语法格式

```
DROP TYPE [ IF EXISTS ] name [, ...] [ CASCADE | RESTRICT ]
```

参数说明

- **IF EXISTS**

  如果指定的类型不存在，那么发出一个 notice 而不是抛出一个错误。

- **name**

  要删除的类型名(可以有模式修饰)。

- **CASCADE**

  级联删除依赖该类型的对象(比如字段、函数、操作符等)。

  RESTRICT

  如果有依赖对象，则拒绝删除该类型（缺省行为）。

示例

参考：SQL 语法参考->SQL 语法->CREATE TYPE 章节中的示例。

相关链接

参考：SQL 语法参考->SQL 语法->CREATE TYPE、ALTER TYPE 章节。

4.5.150.DROP USER

功能描述

删除用户，同时会删除同名的 schema。

注意事项

- 须使用 CASCADE 级联删除依赖用户的对象（除数据库外）。当删除用户的级联对象时，如果级联对象处于锁定状态，则此级联对象无法被删除，直到对象被解锁或锁定级联对象的进程被杀死。
- 在 PanWeiDB 中，存在一个配置参数 enable_kill_query，此参数在配置文件 postgresql.conf 中。此参数影响级联删除用户对象的行为：
  - 当参数 enable_kill_query 为 on，且使用 CASCADE 模式删除用户时，会自动 kill 锁定用户级联对象的进程，并删除用户。
  - 当参数 enable_kill_query 为 off，且使用 CASCADE 模式删除用户时，会等待锁定级联对象的进程结束之后再删除用户。
- 在数据库中删除用户时，如果依赖用户的对象在其他数据库中或者依赖用户的对象是其他数据库，请用户先手动删除其他数据库中的依赖对象或直

接删除依赖数据库，再删除用户。即 drop user 不支持跨数据库进行级联删除。

- 如果该用户被 DATA SOURCE 对象依赖时，无法直接级联删除该用户，需要手动删除对应的 DATA SOURCE 对象之后再删除该用户。

语法格式

```
DROP USER [ IF EXISTS ] user_name [, ...] [ CASCADE | RESTRICT ];
```

参数说明

- **IF EXISTS**

  如果指定的用户不存在，发出一个 notice 而不是抛出一个错误。

- **user_name**

  待删除的用户名。

  取值范围：已存在的用户名。

- **CASCADE | RESTRICT**
  - CASCADE：级联删除依赖用户的对象。
  - RESTRICT：如果用户还有任何依赖的对象，则拒绝删除该用户（缺省行为）。

【说明】

在 PanWeiDB 中，存在一个配置参数 enable_kill_query，此参数在配置文件 postgresql.conf 中。此参数影响级联删除用户对象的行为：

- 当参数 enable_kill_query 为 on，且使用 CASCADE 模式删除用户时，会自动 kill 锁定用户级联对象的进程，并删除用户。
- 当参数 enable_kill_query 为 off，且使用 CASCADE 模式删除用户时，会等待锁定级联对象的进程结束之后再删除用户。

示例

参考：SQL 语法参考->SQL 语法->CREATE USER 章节中的示例。

相关链接

参考：SQL 语法参考->SQL 语法->ALTER USER、CREATE USER 章节。

4.5.151.DROP USER MAPPING

功能描述

移除一个用于外部服务器的用户映射。

语法格式

```
DROP USER MAPPING [ IF EXISTS ] FOR { user_name | USER | CURRENT_USER | PUBLIC }
SERVER server_name;
```

参数描述

- **IF EXISTS**

  如果该用户映射不存在则不要抛出一个错误，而是发出一个提示。

- **user_name**

  该映射的用户名。

  CURRENT_USER 和 USER 匹配当前用户的名称。PUBLIC 被用来匹配系统中所有现存和未来的用户名。

- **server_name**

  用户映射的服务器名。

相关链接

参考：SQL 语法参考->SQL 语法->ALTER USER MAPPING、CREATE USER MAPPING 章节。

4.5.152.DROP VIEW

功能描述

数据库中强制删除已有的视图。

注意事项

视图的所有者或者被授予了视图 DROP 权限的用户或拥有 DROP ANY TABLE 权限的用户，有权限执行 DROP VIEW 的命令，系统管理员默认拥有此权限。

语法格式

```
DROP VIEW [ IF EXISTS ] view_name [, ...] [ CASCADE | RESTRICT ];
```

参数说明

- **IF EXISTS**

  如果指定的视图不存在，则发出一个 notice 而不是抛出一个错误。

- **view_name**

  要删除的视图名称。

  取值范围：已存在的视图。

- **CASCADE | RESTRICT**

  - CASCADE：级联删除依赖此视图的对象（比如其他视图）。
  - RESTRICT：如果有依赖对象存在，则拒绝删除此视图。此选项为缺省值。

示例

参考：SQL 语法参考->SQL 语法->CREATE VIEW 章节中的示例。

相关链接

参考：SQL 语法参考->SQL 语法->ALTER VIEW、CREATE VIEW 章节。

4.5.153.DROP WEAK PASSWORD DICTIONARY

功能描述

清空 gs_global_config 中的所有弱口令。

注意事项

只有初始用户、系统管理员和安全管理员拥有权限执行本语法。

语法格式

```
DROP WEAK PASSWORD DICTIONARY;
```

参数说明

无。

示例

参见 CREATE WEAK PASSWORD DICTIONARY 的示例。

相关链接

参考：SQL 语法参考->SQL 语法->CREATE WEAK PASSWORD DICTIONARY 章节。

4.5.154.EXECUTE

功能描述

执行一个前面准备好的预备语句。因为一个预备语句只在会话的生命期里存在，那么预备语句必须是在当前会话的前些时候用 PREPARE 语句创建的。

注意事项

如果创建预备语句的 PREPARE 语句声明了一些参数，那么传递给 EXECUTE 语句的必须是一个兼容的参数集，否则就会生成一个错误。

语法格式

```
EXECUTE name [ ( parameter [, ...] ) ];
```

参数说明

- **name**

  要执行的预备语句的名称。

- **parameter**

  给预备语句的一个参数的具体数值。它必须是一个和生成与创建这个预备语句时指定参数的数据类型相兼容的值的表达式。

示例

```
--创建表 reason。
panweidb=# CREATE TABLE reason (
  CD_DEMO_SK       INTEGER       NOT NULL,
  CD_GENDER        character(16)        ,
  CD_MARITAL_STATUS   character(100)
)
;
```

```
--插入数据。
panweidb=# INSERT INTO reason VALUES(51, 'AAAAAAAADDAAAAAA', 'reason 51');

--创建表 reason_t1。
panweidb=# CREATE TABLE reason_t1 AS TABLE reason;

--为一个 INSERT 语句创建一个预备语句然后执行它。
panweidb=# PREPARE insert_reason(integer,character(16),character(100)) AS INSE
RT INTO reason_t1 VALUES($1,$2,$3);

panweidb=# EXECUTE insert_reason(52, 'AAAAAAAADDAAAAAA', 'reason 52');

--删除表 reason 和 reason_t1。
panweidb=# DROP TABLE reason;
panweidb=# DROP TABLE reason_t1;
```

4.5.155.EXPLAIN

功能描述

显示 SQL 语句的执行计划。

执行计划将显示 SQL 语句所引用的表会采用什么样的扫描方式，如：简单的顺序扫描、索引扫描等。如果引用了多个表，执行计划还会显示用到的 JOIN 算法。

执行计划的最关键的部分是语句的预计执行开销，这是计划生成器估算执行该语句将花费多长的时间。

若指定了 ANALYZE 选项，则该语句会被执行，然后根据实际的运行结果显示统计数据，包括每个计划节点内时间总开销（毫秒为单位）和实际返回的总行数。这对于判断计划生成器的估计是否接近现实非常有用。

注意事项

❖ 在指定 ANALYZE 选项时，语句会被执行。如果用户想使用 EXPLAIN 分析 INSERT、UPDATE、DELETE、CREATE TABLE AS 或 EXECUTE 语句，而不想改动数据（执行这些语句会影响数据），请使用如下方法。

```
START TRANSACTION;
EXPLAIN ANALYZE ...;
ROLLBACK;
```

❖ 由于参数 DETAIL、NODES、NUM_NODES 是分布式模式下的功能，在单机模式中是被禁止使用的。假如使用，会产生如下错误。

```
panweidb=# create table student(id int, name char(20));
CREATE TABLE
panweidb=# explain (nodes true) insert into student values(5,'a'),(6,'b');
ERROR:  unrecognized EXPLAIN option "nodes"
panweidb=# explain (num_nodes true) insert into student values(5,'a'),(6,'b');
ERROR:  unrecognized EXPLAIN option "num_nodes"
```

语法格式

❖ 显示 SQL 语句的执行计划，支持多种选项，对选项顺序无要求。

```
EXPLAIN [ ( option [, ...] ) ] statement;
```

其中选项 option 子句的语法为。

```
ANALYZE [ boolean ] |
  ANALYSE [ boolean ] |
  VERBOSE [ boolean ] |
  COSTS [ boolean ] |
  CPU [ boolean ] |
  DETAIL [ boolean ] |(不可用)
  NODES [ boolean ] |(不可用)
  NUM_NODES [ boolean ] |(不可用)
  BUFFERS [ boolean ] |
  TIMING [ boolean ] |
  PLAN [ boolean ] |
  FORMAT { TEXT | XML | JSON | YAML }
```

❖ 显示 SQL 语句的执行计划，且要按顺序给出选项。

```
EXPLAIN { [ { ANALYZE | ANALYSE } ] [ VERBOSE ] | PERFORMANCE } statement;
```

参数说明

- **statement**

  指定要分析的 SQL 语句。

- **ANALYZE boolean | ANALYSE boolean**

  显示实际运行时间和其他统计数据。

  取值范围：

  - TRUE（缺省值）：显示实际运行时间和其他统计数据。
  - FALSE：不显示。

- **VERBOSE boolean**

  显示有关计划的额外信息。

  取值范围：

  - TRUE（缺省值）：显示额外信息。
  - FALSE：不显示。

- **COSTS boolean**

  包括每个规划节点的估计总成本，以及估计的行数和每行的宽度。

  取值范围：

  - TRUE（缺省值）：显示估计总成本和宽度。
  - FALSE：不显示。

- **CPU boolean**

  打印 CPU 的使用情况的信息。

  取值范围：

  - TRUE（缺省值）：显示 CPU 的使用情况。
  - FALSE：不显示。

- **DETAIL boolean（不可用）**

  打印数据库节点上的信息。

  取值范围：

  - TRUE（缺省值）：打印数据库节点的信息。
  - FALSE：不打印。

- **NODES boolean（不可用）**

  打印 query 执行的节点信息。

取值范围：

- TRUE（缺省值）：打印执行的节点的信息。
- FALSE：不打印。

❖ **NUM_NODES boolean（不可用）**

打印执行中的节点的个数信息。

取值范围：

- TRUE（缺省值）：打印数据库节点个数的信息。
- FALSE：不打印。

❖ **BUFFERS boolean**

包括缓冲区的使用情况的信息。

取值范围：

- TRUE：显示缓冲区的使用情况。
- FALSE（缺省值）：不显示。

❖ **TIMING boolean**

包括实际的启动时间和花费在输出节点上的时间信息。

取值范围：

- TRUE（缺省值）：显示启动时间和花费在输出节点上的时间信息。
- FALSE：不显示。

❖ **PLAN**

是否将执行计划存储在 plan_table 中。当该选项开启时，会将执行计划存储在 PLAN_TABLE 中，不打印到当前屏幕，因此该选项为 on 时，不能与其他选项同时使用。

取值范围：

- ON（缺省值）：将执行计划存储在 plan_table 中，不打印到当前屏幕。执行成功返回 EXPLAIN SUCCESS。
- OFF：不存储执行计划，将执行计划打印到当前屏幕。

❖ **FORMAT**

指定输出格式。

取值范围：TEXT、XML、JSON 和 YAML。

默认值：TEXT。

❖ **PERFORMANCE**

使用此选项时，即打印执行中的所有相关信息。

示例

1、创建测试表。

```
create table test_t(c1 int, c2 varchar(30));
```

2、显示 SQL 执行计划。

```
 explain select * from test_t;
```

回显如下：

```
                    QUERY PLAN
-----------------------------------------------------------
 Seq Scan on test_t  (cost=0.00..17.29 rows=729 width=82)
(1 row)
```

3、在查看计划时可以指定输出格式。

注意：只有当 explain_perf_mode 为 normal 时，才支持 json 格式

```
 SET explain_perf_mode=normal;
explain (format json) select * from test_t;
```

回显如下：

```
        QUERY PLAN
--------------------------------
[                    +
 {                   +
   "Plan": {             +
     "Node Type": "Seq Scan",  +
     "Relation Name": "test_t",+
     "Alias": "test_t",      +
     "Startup Cost": 0.00,    +
     "Total Cost": 17.29,     +
     "Plan Rows": 729,        +
     "Plan Width": 82         +
   }                   +
```

```
 }                  +
]
(1 row)
```

4、如果一个查询中的 where 子句的列有索引，在条件或数据等不一样时，可能会显示不同的执行计划。

```
create index idx_test_t_c1 on test_t(c1);
insert into test_t values(generate_series(1, 200), 'hello panweidb');
explain select c1, c2 from test_t where c1=100;
```

回显如下：

```
      QUERY PLAN
------------------------------------------------------------------------------
 Bitmap Heap Scan on test_t  (cost=4.28..12.74 rows=4 width=82)
   Recheck Cond: (c1 = 100)
   ->  Bitmap Index Scan on idx_test_t_c1  (cost=0.00..4.28 rows=4 width=0)
         Index Cond: (c1 = 100)
(4 rows)
```

5、可以通过 costs 选项，指定是否显示开销。

```
explain (costs false) select * from test_t where c1=100;
```

回显如下：

```
    QUERY PLAN
----------------------
 Seq Scan on test_t
   Filter: (c1 = 100)
(2 rows)
```

相关链接

参考：SQL 语法参考->SQL 语法->ANALYZE 和 ANALYSE 章节。

4.5.156.EXPLAIN PLAN

功能描述

通过 EXPLAIN PLAN 命令可以将查询执行的计划信息存储于 PLAN_TABLE 表中。与 EXPLAIN 命令不同的是，EXPLAIN PLAN 仅将计划信息进行存储，而不会打印到屏幕。

语法格式

```
EXPLAIN PLAN
[ SET STATEMENT_ID = string ]
FOR statement ;
```

参数说明

- EXPLAIN 中的 PLAN 选项表示需要将计划信息存储于 PLAN_TABLE 中，存储成功将返回“EXPLAIN SUCCESS”。
- STATEMENT_ID 用户可以对查询设置标签，输入的标签信息也将存储于 PLAN_TABLE 中。

【说明】

用户在执行 EXPLAIN PLAN 时，如果没有进行 SET STATEMENT_ID，则默认为空值。同时，用户可输入的 STATEMENT_ID 最大长度为 30 个字节，超过长度将会产生报错。

注意事项

- EXPLAIN PLAN 不支持在数据库节点上执行。
- 对于执行错误的 SQL 无法进行计划信息的收集。
- PLAN_TABLE 中的数据是 session 级生命周期并且 session 隔离和用户隔离，用户只能看到当前 session、当前用户的数据。

示例

使用 EXPLAIN PLAN 收集 SQL 语句的执行计划，通常包括以下步骤：

1、创建模式 pw。

说明：

执行 EXPLAIN PLAN 后会将计划信息自动存储于 PLAN_TABLE 中，不支持对 PLAN_TABLE 进行 INSERT、UPDATE、ANALYZE 等操作。

PLAN_TABLE 详细介绍参考：系统表和系统视图->系统视图->PLAN_TABLE 章节。

```
DROP SCHEMA IF EXISTS pw CASCADE;
CREATE SCHEMA pw;
```

2、创建测试表。

```
CREATE TABLE pw.t (
 id integer primary key NOT NULL,
 a integer DEFAULT NULL,
 b integer DEFAULT NULL
) ;
```

3、创建函数。

```
create or replace function pw.creatData() returns
boolean AS
$BODY$
declare ii integer;
 begin
 II:=1;
 FOR ii IN 1..500000 LOOP
 INSERT INTO pw.t (id,a,b) VALUES (ii,ii,ii);
 end loop;
 return true;
 end;
$BODY$
LANGUAGE plpgsql;
```

4、像表中插入数据。

```
select * from pw.creatData() as testexplain;
create index ind_a on pw.T(a);
```

5、执行 EXPLAIN PLAN 后会将计划信息自动存储于 PLAN_TABLE 中。

```
explain plan SET STATEMENT_ID = 'test_ex_plan01' FOR SELECT MAX(id),AVG(b) fro
m pw.T where a between 3764 and 47491;
```

6、查询 PLAN_TABLE。

```
SELECT * FROM PLAN_TABLE;
```

返回结果为：

```
 statement_id |    plan_id    | id |  operation  |    options    | object_name | object_t
ype | object_owner |   projection   |
cost     | cardinality

---------------+-----------------+----+-------------+-----------------+-----------
--+------------+-------------+----------------+-------
----------+-------------
 test_ex_plan01 | 6473924464813365 |  1 | AGGREGATE   | PLAIN          |          |
|          | max(id), avg(b) | 3380.2
2577317446 |         1
 test_ex_plan01 | 6473924464813365 |  2 | TABLE ACCESS | BITMAP HEAP SCAN  | t
  | TABLE      | pw         | id, a, b      | 3367.7
1327317446 |      2500
 test_ex_plan01 | 6473924464813365 |  3 | INDEX      | BITMAP INDEX SCAN | ind_a
| INDEX      | pw         |            |
   53.25 |      2500
(3 rows)
```

7、清理 PLAN_TABLE 表中的数据。

```
DELETE FROM PLAN_TABLE WHERE STATEMENT_ID = 'test_ex_plan01';
```

4.5.157.FETCH

功能描述

FETCH 通过已创建的游标来检索数据。

每个游标都有一个供 FETCH 使用的关联位置。游标的关联位置可以在查询结果的第一行之前，或者在结果中的任意行，或者在结果的最后一行之后：

- 游标刚创建完之后，关联位置在第一行之前的。
- 在抓取了一些移动行之后，关联位置在检索到的最后一行上。

- 如果 FETCH 抓取完了所有可用行，它就停在最后一行后面，或者在反向抓取的情况下是停在第一行前面。
- FETCH ALL 或 FETCH BACKWARD ALL 将总是把游标的关联位置放在最后一行或者在第一行前面。

注意事项

- 如果游标定义了 NO SCROLL，则不允许使用例如 FETCH BACKWARD 之类的反向抓取。
- NEXT、PRIOR、FIRST、LAST、ABSOLUTE、RELATIVE 形式在恰当地移动游标之后抓取一条记录。如果后面没有数据行，就返回一个空的结果，此时游标就会停在查询结果的最后一行之后（向后查询时）或者第一行之前（向前查询时）。
- FORWARD 和 BACKWARD 形式在向前或者向后移动的过程中抓取指定的行数，然后把游标定位在最后返回的行上；或者是，如果 count 大于可用的行数，则在所有行之后（向后查询时）或者之前（向前查询时）。
- RELATIVE 0、FORWARD 0、BACKWARD 0 都要求在不移动游标的前提下抓取当前行，也就是重新抓取最近刚抓取过的行。除非游标定位在第一行之前或者最后一行之后，这个动作都应该成功，而在那两种情况下，不返回任何行。
- 当 FETCH 的游标上涉及列存表时，不支持 BACKWARD、PRIOR 等涉及反向获取操作。

语法格式

```
FETCH [ direction { FROM | IN } ] cursor_name;
```

其中 direction 子句为可选参数。

```
NEXT
  | PRIOR
  | FIRST
  | LAST
  | ABSOLUTE count
  | RELATIVE count
```

```
| count
| ALL
| FORWARD
| FORWARD count
| FORWARD ALL
| BACKWARD
| BACKWARD count
| BACKWARD ALL
```

参数说明

- **direction_clause**

  定义抓取数据的方向。

  取值范围：

  - NEXT（缺省值）

    从当前关联位置开始，抓取下一行。

  - PRIOR

    从当前关联位置开始，抓取上一行。

  - FIRST

    抓取查询的第一行（和 ABSOLUTE 1 相同）。

  - LAST

    抓取查询的最后一行（和 ABSOLUTE -1 相同）。

  - ABSOLUTE count

    抓取查询中第 count 行。

    ABSOLUTE 抓取不会比用相对位移移动到需要的数据行更快，因为下层的实现必须遍历所有中间的行。

    count 取值范围：有符号的整数

    - count 为正数，就从查询结果的第一行开始，抓取第 count 行。
    - count 为负数，就从查询结果末尾抓取第 abs(*count*) 行。
    - count 为 0 时，定位在第一行之前。

  - RELATIVE count

➢从当前关联位置开始，抓取随后或前面的第 count 行。

➢取值范围：有符号的整数

- ✧ count 为正数就抓取当前关联位置之后的第 count 行。
- ✧ count 为负数就抓取当前关联位置之前的第 abs(count) 行。
- ✧ 如果当前行没有数据的话，RELATIVE 0 返回空。

➢ count

➢抓取随后的 count 行（和 FORWARD count 一样）。

➢ ALL

➢从当前关联位置开始，抓取所有剩余的行（和 FORWARD ALL 一样）。

➢ FORWARD

➢抓取下一行（和 NEXT 一样）。

➢ FORWARD count

➢从当前关联位置开始，抓取随后或前面的 count 行。

➢ FORWARD ALL

➢从当前关联位置开始，抓取所有剩余行。

➢ BACKWARD

➢从当前关联位置开始，抓取前面一行(和 PRIOR 一样)。

➢ BACKWARD count

➢从当前关联位置开始，抓取前面的 count 行（向后扫描）。

➢取值范围：有符号的整数

- ✧ count 为正数就抓取当前关联位置之前的 count 行。
- ✧ count 为负数就抓取当前关联位置之后的 abs（count）行。
- ✧ 如果有数据的话，BACKWARD 0 重新抓取当前行。

➢ BACKWARD ALL

➢从当前关联位置开始，抓取所有前面的行（向后扫描）。

❖ **{ FROM | IN } cursor_name**

使用关键字 FROM 或 IN 指定游标名称。

取值范围：已创建的游标的名称。

示例

```
--SELECT 语句，用一个游标读取一个表。开始一个事务。
panweidb=# START TRANSACTION;

--建立一个名为 cursor1 的游标。
panweidb=# CURSOR cursor1 FOR SELECT * FROM tpcds.customer_address ORDER B
Y 1;

--抓取头 3 行到游标 cursor1 里。
panweidb=# FETCH FORWARD 3 FROM cursor1;
 ca_address_sk | ca_address_id  | ca_street_number |  ca_street_name  | ca_street_
type | ca_suite_number |    ca_city    |   ca_county    | ca_state |  ca_zip  | ca_countr
y  | ca_gmt_offset |  ca_location_type
---------------+------------------+------------------+--------------------+-------------
----+-----------------+-----------------+-----------------+----------+------------+-----
----------+---------------+----------------------
          1 | AAAAAAAABAAAAAAA | 18               | Jackson          | Parkway       | Suite 28
0      | Fairfield      | Maricopa County | AZ       | 86192    | United States |       -7.00 | co
ndo
          2 | AAAAAAAACAAAAAAA | 362              | Washington 6th   | RD            | Suite 8
0      | Fairview       | Taos County     | NM       | 85709    | United States |       -7.00 | co
ndo
          3 | AAAAAAAADAAAAAAA | 585              | Dogwood Washington | Circle        | S
uite Q       | Pleasant Valley | York County     | PA       | 12477    | United States |       -5.
00 | single family
(3 rows)

--关闭游标并提交事务。
panweidb=# CLOSE cursor1;

--结束一个事务。
panweidb=# END;

--VALUES 子句，用一个游标读取 VALUES 子句中的内容。开始一个事务。
```

```
panweidb=# START TRANSACTION;

--建立一个名为 cursor2 的游标。
panweidb=# CURSOR cursor2 FOR VALUES(1,2),(0,3) ORDER BY 1;

--抓取头 2 行到游标 cursor2 里。
panweidb=# FETCH FORWARD 2 FROM cursor2;
column1 | column2
---------+---------
0 |       3
1 |       2
(2 rows)

--关闭游标并提交事务。
panweidb=# CLOSE cursor2;

--结束一个事务。
panweidb=# END;

--WITH HOLD 游标的使用，开启事务。
panweidb=# START TRANSACTION;

--创建一个 with hold 游标。
panweidb=# DECLARE cursor1 CURSOR WITH HOLD FOR SELECT * FROM tpcds.custo
mer_address ORDER BY 1;

--抓取头 2 行到游标 cursor1 里。
panweidb=# FETCH FORWARD 2 FROM cursor1;
 ca_address_sk | ca_address_id  | ca_street_number |  ca_street_name  | ca_street_
type | ca_suite_number |    ca_city    |   ca_county    | ca_state |  ca_zip  | ca_countr
y  | ca_gmt_offset |  ca_location_type
---------------+------------------+------------------+--------------------+------------
----+-----------------+----------------+----------------+----------+------------+-----
----------+---------------+----------------------
```

```
         1 | AAAAAAAABAAAAAAA | 18           | Jackson         | Parkway       | Suite 28
0     | Fairfield      | Maricopa County | AZ      | 86192     | United States |       -7.00 | co
ndo
         2 | AAAAAAAACAAAAAAA | 362          | Washington 6th    | RD            | Suite 8
0     | Fairview       | Taos County     | NM      | 85709     | United States |       -7.00 | co
ndo
(2 rows)

--结束事务。
panweidb=# END;

--抓取下一行到游标 cursor1 里。
panweidb=# FETCH FORWARD 1 FROM cursor1;
 ca_address_sk | ca_address_id  | ca_street_number |  ca_street_name  | ca_street_
type  | ca_suite_number |    ca_city    |   ca_county    | ca_state |  ca_zip  | ca_countr
y   | ca_gmt_offset |  ca_location_type
---------------+-----------------+-----------------+------------------+------------
----+-----------------+-----------------+-----------------+----------+------------+-----
----------+---------------+----------------------
         3 | AAAAAAAADAAAAAAA | 585          | Dogwood Washington | Circle        | S
uite Q        | Pleasant Valley | York County     | PA      | 12477     | United States |       -5.
00 | single family
(1 row)

--关闭游标。
panweidb=# CLOSE cursor1;
```

相关链接

参考：SQL 语法参考->SQL 语法->CLOSE、MOVE 章节。

4.5.158.MERGE

功能描述

通过 MERGE INTO 语句，将目标表和源表中数据针对关联条件进行匹配，若关联条件匹配时对目标表进行 UPDATE，无法匹配时对目标表执行 INSERT。此语法可以很方便地用来合并执行 UPDATE 和 INSERT，避免多次执行。

注意事项

进行 MERGE INTO 操作的用户需要同时拥有目标表的 UPDATE 和 INSERT 权限，以及源表的 SELECT 权限。

语法格式

```
MERGE [/*+ plan_hint */] INTO table_name [ partition_clause ] [ [ AS ] alias ]
USING { { table_name | view_name } | subquery } [ [ AS ] alias ]
ON ( condition )
[
 WHEN MATCHED THEN
 UPDATE SET { column_name = { expression | subquery | DEFAULT } |
       ( column_name [, ...] ) = ( { expression | subquery | DEFAULT } [, ...] ) } [, ...]
 [ WHERE condition ]
]
[
 WHEN NOT MATCHED THEN
 INSERT { DEFAULT VALUES |
 [ ( column_name [, ...] ) ] VALUES ( { expression | subquery | DEFAULT } [, ...] ) [, ...] [
WHERE condition ] }
];
```

其中 partition_clause 可以是：

```
PARTITION { ( partition_name ) | FOR ( partition_value [, ...] ) } |
SUBPARTITION { ( subpartition_name ) | FOR ( subpartition_value [, ...] ) }
```

参数说明

- **plan_hint 子句**

  以/*+*/的形式在 MERGE 关键字后，用于对 MERGE 对应的语句块生成的计划进行 hint 调优，详细用法请参见：《PanWeiDB V2.0-S3.0.1_B01

管理员指南->性能调优->SQL 调优指南->使用 Plan Hint 进行调优》章节。

每条语句中只有第一个/+ plan_hint */注释块会作为 hint 生效，里面可以写多条 hint。

- **INTO 子句**

  指定正在更新或插入的目标表。

  - **talbe_name**

    目标表的表名。

  - **Alias**

    目标表的别名。

    取值范围：字符串，符合标识符命名规范。

- **partition_clause**

  指定分区 MERGE 操作：

```
PARTITION { ( partition_name ) | FOR ( partition_value [, ...] ) } |
SUBPARTITION { ( subpartition_name ) | FOR ( subpartition_value [, ...] ) }
```

关键字详见：SQL 语法参考->SQL 语法->SELECT 章节。

如果 value 子句的值和指定分区不一致，会抛出异常。

示例详见：SQL 语法参考->SQL 语法->CREATE TABLESUBPARTITION 章节。

- **alias**

  目标表的别名。

  取值范围：字符串，符合标识符命名规范。

- **USING 子句**

  指定源表，源表可以为表、视图或子查询。

- **ON 子句**

  关联条件，用于指定目标表和源表的关联条件。不支持更新关联条件中的字段。

- **WHEN MATCHED 子句**

  当源表和目标表中数据针对关联条件可以匹配上时，选择 WHEN MATCHED 子句进行 UPDATE 操作。

不支持更新系统表、系统列。

- **WHEN NOT MATCHED 子句**

  当源表和目标表中数据针对关联条件无法匹配时，选择 WHEN NOT MATCHED 子句进行 INSERT 操作。

  不支持 INSERT 子句中包含多个 VALUES。

  WHEN MATCHED 和 WHEN NOT MATCHED 子句顺序可以交换，可以缺省其中一个，但不能同时缺省，不支持同时指定两个 WHEN MATCHED 或 WHEN NOT MATCHED 子句。

- **DEFAULT**

  用对应字段的缺省值填充该字段。

  如果没有缺省值，则为 NULL。

- **WHERE condition**

  UPDATE 子句和 INSERT 子句的条件，只有在条件满足时才进行更新操作，可缺省。不支持 WHERE 条件中引用系统列。不建议使用 int 等数值类型作为 condition，因为 int 等数值类型可以隐式转换为 bool 值（非 0 值隐式转换为 true，0 转换为 false），可能导致非预期的结果。

示例

1、创建目标表 products 和源表 newproducts，并插入数据。

```
CREATE TABLE products
(
product_id INTEGER,
product_name VARCHAR2(60),
category VARCHAR2(60)
);

INSERT INTO products VALUES (1501, 'vivitar 35mm', 'electrncs');
INSERT INTO products VALUES (1502, 'olympus is50', 'electrncs');
INSERT INTO products VALUES (1600, 'play gym', 'toys');
INSERT INTO products VALUES (1601, 'lamaze', 'toys');
INSERT INTO products VALUES (1666, 'harry potter', 'dvd');
```

```
CREATE TABLE newproducts
(
product_id INTEGER,
product_name VARCHAR2(60),
category VARCHAR2(60)
);

INSERT INTO newproducts VALUES (1502, 'olympus camera', 'electrncs');
INSERT INTO newproducts VALUES (1601, 'lamaze', 'toys');
INSERT INTO newproducts VALUES (1666, 'harry potter', 'toys');
INSERT INTO newproducts VALUES (1700, 'wait interface', 'books');
```

2、进行 MERGE INTO 操作。

```
MERGE INTO products p
USING newproducts np
ON (p.product_id = np.product_id)
WHEN MATCHED THEN
 UPDATE SET p.product_name = np.product_name, p.category = np.category WHER
E p.product_name != 'play gym'
WHEN NOT MATCHED THEN
 INSERT VALUES (np.product_id, np.product_name, np.category) WHERE np.catego
ry = 'books';
```

提交回显结果为：

```
MERGE 4
```

3、查询更新后的结果。

```
SELECT * FROM products ORDER BY product_id;
```

显示结果为：

```
product_id |  product_name  | category
------------+----------------+-----------
      1501 | vivitar 35mm   | electrncs
      1502 | olympus camera | electrncs
      1600 | play gym       | toys
```

```
 1601 | lamaze       | toys
 1666 | harry potter | toys
 1700 | wait interface | books
(6 rows)
```

功能描述

向表中添加一行或多行数据。

注意事项

- 只有拥有表 INSERT 权限的用户，才可以向表中插入数据。用户被授予 insert any table 权限，相当于用户对除系统模式之外的任何模式具有 USAGE 权限，并且拥有这些模式下表的 INSERT 权限
- 如果使用 RETURNING 子句，用户必须要有该表的 SELECT 权限。
- 如果使用 ON DUPLICATE KEY UPDATE，用户必须要有该表的 SELECT、UPDATE 权限，唯一约束（主键或唯一索引）的 SELECT 权限。
- 如果使用 query 子句插入来自查询里的数据行，用户还需要拥有在查询里使用的表的 SELECT 权限。
- 生成列不能被直接写入。在 INSERT 命令中不能为生成列指定值，但是可以指定关键字 DEFAULT。
- 当连接到 TD 兼容的数据库时，td_compatible_truncation 参数设置为 on 时，将启用超长字符串自动截断功能，在后续的 insert 语句中（不包含外表的场景下），对目标表中 char 和 varchar 类型的列上插入超长字符串时，系统会自动按照目标表中相应列定义的最大长度对超长字符串进行截断。

【说明】

如果向字符集为字节类型编码（SQL_ASCII，LATIN1 等）的数据库中插入多字节字符数据（如汉字等），且字符数据跨越截断位置，这种情况下，按照字节长度自动截断，自动截断后会在尾部产生非预期结果。如果用户有对于截断结果正确性的要求，建议用户采用 UTF8 等能够按照字符截断的输入字符集作为数据库的编码集。

❖ 优化建议：通过 insert 语句批量插入数据时，建议将多条记录合并入一条语句中执行插入，以提高数据加载性能。

例如，INSERT INTO sections VALUES (30, 'Administration', 31, 1900),(40, 'Development', 35, 2000), (50, 'Development' , 60 , 2001);。

语法格式

```
  [ WITH [ RECURSIVE ] with_query [, ...] ]
  INSERT [/*+ plan_hint */] [IGNORE] INTO table_name [ @dblink_name ] [partition_c
lause] [ AS alias ] [ ( column_name [, ...] ) ]
    { DEFAULT VALUES | VALUES | VALUE {( { expression | DEFAULT } [, ...] ) }[, ...] | quer
y }
    [ upsert_clause ]
    [ RETURNING {* | {output_expression [ [ AS ] output_name ] }[, ...]} ];
  INSERT [ FIRST | ALL ]
  [ WHEN { condition } THEN INTO table [ [ AS ] alias ] [ ( column_name [, ...] ) ] VALUE
S (v1, ...), ...]
  { subquery }
```

❖ 其中 upsert_clause 可以是下列之一：

```
      ON DUPLICATE KEY UPDATE { NOTHING | { column_name = { expression |
DEFAULT } } [, ...] }

      ON CONFLICT [ conflict_target ] conflict_action
```

❖ 其中 conflict_target 可以是下列之一：

```
      ( { index_column_name | ( index_expression ) } [ COLLATE collation ] [ opclass ]
[, ...] ) [ WHERE index_predicate ]

      ON CONSTRAINT constraint_name
```

❖ 其中 conflict_action 可以是下列之一：

```
    DO NOTHING
    DO UPDATE SET { column_name = { expression | DEFAULT } |
                    ( column_name [, ...] ) = [ ROW ] ( { expression | DEFAULT } [, ...] ) |
                    ( column_name [, ...] ) = ( sub-SELECT )
                  } [, ...]
              [ WHERE condition ]
```

❖ 其中 with_query 可以是：

```
with_query_name [ ( column_name [, ...] ) ] AS [ [ NOT ] MATERIALIZED ]
( {select | values | insert | update | delete} )
```

❖ 其中 partition_clause 可以是：

```
PARTITION { ( partition_name ) | FOR ( partition_value [, ...] ) } |
SUBPARTITION { ( subpartition_name ) | FOR ( subpartition_value [, ...] ) }
```

【须知】
其中 partition_clause 仅在集中式模式支持。

参数说明

❖ **WITH [RECURSIVE] with_query [, ...]**

用于声明一个或多个可以在主查询中通过名称引用的子查询，相当于临时表。

如果声明了 RECURSIVE，那么允许 SELECT 子查询通过名称引用它自己。

1、其中 with_query 的详细格式为：

```
with_query_name [ ( column_name [, ...] ) ] AS [ [ NOT ] MATERIALIZED ]
  ( {select | values | insert | update | delete} )
```

2、with_query_name 指定子查询生成的结果集名称，在查询中可使用该名称访问子查询的结果集。

3、column_name 指定子查询结果集中显示的列名。

4、每个子查询可以是 SELECT，VALUES，INSERT，UPDATE 或 DELETE 语句。

5、用户可以使用 MATERIALIZED / NOT MATERIALIZED 对 CTE 进行修饰。

- 如果声明为 MATERIALIZED，WITH 查询将被物化，生成一个子查询结果集的拷贝，在引用处直接查询该拷贝，因此 WITH 子查询无法和主干 SELECT 语句进行联合优化（如谓词下推、等价类传递等），对于此类场景可以使用 NOT MATERIALIZED 进行修饰，如果 WITH 查询语义上可以作为子查询内联执行，则可以进行上述优化。
- 如果用户没有显示声明物化属性则遵守以下规则：如果 CTE 只在所属主干语句中被引用一次，且语义上支持内联执行，则会被改写为子查询内联执行，否则以 CTE Scan 的方式物化执行。

【说明】

INSERT ON DUPLICATE KEY UPDATE 不支持 WITH 及 WITH RECURSIVE 子句。

❖ **plan_hint 子句**

以/*+ */的形式在 INSERT 关键字后，用于对 INSERT 对应的语句块生成的计划进行 hint 调优，详细用法请参见：《PanWeiDB V2.0-S3.0.1_B01 管理员指南->性能调优->SQL 调优指南->使用 Plan Hint 进行调优》章节。每条语句中只有第一个/*+ plan_hint */注释块会作为 hint 生效，里面可以写多条 hint。

❖ **table_name**

要插入数据的目标表名。

取值范围：已存在的表名。

❖ **partition_clause**

指定分区插入操作

```
PARTITION { ( partition_name ) | FOR ( partition_value [, ...] ) } |
 SUBPARTITION { ( subpartition_name ) | FOR ( subpartition_value [, ...] ) }
```

关键字详见 SQL 语法参考->SQL 语法->SELECT 章节。

如果 value 子句的值和指定分区不一致，会抛出异常。

示例详见：SQL 语法参考->SQL 语法->CREATE TABLE SUBPARTITION 章节。

❖ **column_name**

目标表中的字段名：

- 字段名可以有子字段名或者数组下标修饰。
- 没有在字段列表中出现的每个字段，将由系统默认值，或者声明时的默认值填充，若都没有则用 NULL 填充。例如，向一个复合类型中的某些字段插入数据的话，其他字段将是 NULL。
- 目标字段（column_name）可以按顺序排列。如果没有列出任何字段，则默认全部字段，且顺序为表声明时的顺序。
- 如果 value 子句和 query 中只提供了 N 个字段，则目标字段为前 N 个字段。
- value 子句和 query 提供的值在表中从左到右关联到对应列。

取值范围：已存在的字段名。

❖ **expression**

赋予对应 column 的一个有效表达式或值：

- 如果是 INSERT ON DUPLICATE KEY UPDATE 语句下，expression 可以为 VALUES(column_name)或 EXCLUDED.column_name 用来表示引用冲突行对应的 column_name 字段的值。需注意，其中 VALUES(column_name)不支持嵌套在表达式中（例如 VALUES(column_name)+1），但 EXCLUDED 不受此限制。
- 向表中字段插入单引号“'”时需要使用单引号自身进行转义。
- 如果插入行的表达式不是正确的数据类型，系统试图进行类型转换，若转换不成功，则插入数据失败，系统返回错误信息。

❖ **DEFAULT**

对应字段名的缺省值。如果没有缺省值，则为 NULL。

- **query**

  一个查询语句（SELECT 语句），将查询结果作为插入的数据。

- **RETURNING**

  返回实际插入的行，RETURNING 列表的语法与 SELECT 的输出列表一致。

  注意：INSERT ON DUPLICATE KEY UPDATE 不支持 RETURNING 子句。

- **output_expression**

  INSERT 命令在每一行都被插入之后用于计算输出结果的表达式。

  取值范围：该表达式可以使用 table 的任意字段。可以使用*返回被插入行的所有字段。

- **output_name**

  字段的输出名称。

  取值范围：字符串，符合标识符命名规范。

- **ON DUPLICATE KEY UPDATE**

  对于带有唯一约束（UNIQUE INDEX 或 PRIMARY KEY）的表，如果插入数据违反唯一约束，则对冲突行执行 UPDATE 子句完成更新，对于不带唯一约束的表，则仅执行插入。UPDATE 时，若指定 NOTHING 则忽略此条插入，可通过“EXCLUDE.”或者 "VALUES()" 来选择源数据相应的列。

  - 支持触发器，触发器执行顺序由实际执行流程决定：
    - 执行 insert：触发 before insert、after insert 触发器。
    - 执行 update：触发 before insert、before update、after update 触发器。
    - 执行 update nothing：触发 before insert 触发器。
  - 不支持延迟生效（DEFERRABLE）的唯一约束或主键。
  - 如果表中存在多个唯一约束，如果所插入数据违反多个唯一约束，对于检测到冲突的第一行进行更新，其他冲突行不更新（检查顺序与索引维护具有强相关性，一般先创建的索引先进行冲突检查）。
  - 如果插入多行，这些行均与表中同一行数据存在唯一约束冲突，则按照顺序，第一条执行插入或更新，之后依次执行更新。
  - 主键、唯一索引列不允许 UPDATE。

- 不支持列存，不支持外表、内存表。
- expression 支持使用子查询表达式，其语法与功能同 UPDATE。子查询表达式中支持使用“EXCLUDED.”来选择源数据相应的列。

ON CONFLICT [conflict_target] conflict_action

PG 风格的 ON CONFLICT 子句指定插入数据违反唯一约束时的替换动作，无冲突时直接插入，存在冲突时执行 UPDATE，因此也被称为 UPSERT 功能--“UPDATE 或 INSERT”，用法介绍详见《PanWeiDB V2.0-S3.0.1_B01 管理员指南》->UPSERT 功能。

示例

1、创建表 reason_t2。

```
CREATE TABLE reason_t2
(
 r_reason_sk   integer,
 r_reason_id   character(16),
 r_reason_desc character(100)
);
```

2、向表中插入一条记录并查询。

```
INSERT INTO reason_t2(r_reason_sk, r_reason_id, r_reason_desc) VALUES (1,
'AAAAAAAABAAAAAAA', 'reason1');
SELECT * FROM reason_t2;
```

查询结果为：

```
 r_reason_sk | r_reason_id   |                    r_reason_
desc
```

```
    ------------+-----------------+-------------------------------------------------
--

    ------------------------------------------------

          1 | AAAAAAAABAAAAAAA | reason1


    (1 row)
```

2、向表中插入一条记录并查询，和上一条语法等效。

```
    INSERT INTO reason_t2 VALUES (2, 'AAAAAAAABAAAAAAA', 'reason2');

    SELECT * FROM reason_t2;
```

查询结果为：

```
     r_reason_sk |  r_reason_id   |                          r_reason_
    desc

    ------------+-----------------+-------------------------------------------------
--

    ------------------------------------------------

          1 | AAAAAAAABAAAAAAA | reason1


          2 | AAAAAAAABAAAAAAA | reason2


    (2 rows)
```

3、向表中插入多条记录并查询。

```
    INSERT INTO reason_t2 VALUES (3, 'AAAAAAAACAAAAAAA','reason3'),(4,
'AAAAAAAADAAAAAAA', 'reason4'),(5, 'AAAAAAAAEAAAAAAA','reason5');

    SELECT * FROM reason_t2;
```

查询结果为：

```
 r_reason_sk| r_reason_id  |                    r_reason_
desc
------------+----------------+---------------------------------------------
--
-----------------------------------------------
        1|AAAAAAAABAAAAAAA|reason1

        2|AAAAAAAABAAAAAAA|reason2

        3|AAAAAAAACAAAAAAA|reason3

        4|AAAAAAAADAAAAAAA|reason4

        5|AAAAAAAAEAAAAAAA|reason5

(5 rows)
```

4、向表中插入 reason 中 r_reason_sk 小于 5 的记录并查询。

```
INSERT INTO reason_t2 SELECT * FROM reason_t2 WHERE r_reason_sk <5;
SELECT * FROM reason_t2;
```

查询结果为：

```
 r_reason_sk| r_reason_id  |                    r_reason_
desc
------------+----------------+---------------------------------------------
--
-----------------------------------------------
```

```
        1 | AAAAAAAABAAAAAAA | reason1

        2 | AAAAAAAABAAAAAAA | reason2

        3 | AAAAAAAACAAAAAAA | reason3

        4 | AAAAAAAADAAAAAAA | reason4

        5 | AAAAAAAAEAAAAAAA | reason5

        1 | AAAAAAAABAAAAAAA | reason1

        2 | AAAAAAAABAAAAAAA | reason2

        3 | AAAAAAAACAAAAAAA | reason3

        4 | AAAAAAAADAAAAAAA | reason4

(9 rows)
```

5、对表创建索引。

```
CREATE INDEX reason_t2_u_index ON reason_t2(r_reason_sk);
```

6、向表中插入多条记录，如果冲突则更新冲突数据行中 r_reason_id 字段为'BBBBBBBBCAAAAAAA'，并查询。

```
    INSERT INTO reason_t2 VALUES (5, 'BBBBBBBBCAAAAAAA','reason5'),(6,
'AAAAAAAADAAAAAAA', 'reason6') ON DUPLICATE KEY UPDATE r_reason_id =
'BBBBBBBBCAAAAAAA';

    SELECT * FROM reason_t2;
```

查询结果为：

```
    r_reason_sk|  r_reason_id   |                          r_reason_

    desc

    ------------+-----------------+-----------------------------------------------
--

    ----------------------------------------------

           1|AAAAAAAABAAAAAAA|reason1


           2|AAAAAAAABAAAAAAA|reason2


           3|AAAAAAAACAAAAAAA|reason3


           4|AAAAAAAADAAAAAAA|reason4


           5|AAAAAAAAEAAAAAAA|reason5


           1|AAAAAAAABAAAAAAA|reason1


           2|AAAAAAAABAAAAAAA|reason2


           3|AAAAAAAACAAAAAAA|reason3
```

```
        4 | AAAAAAAADAAAAAAA | reason4

        5 | BBBBBBBBCAAAAAAA | reason5

        6 | AAAAAAAADAAAAAAA | reason6

(11 rows)
```

7、删除表 reason_t2。

```
DROP TABLE reason_t2;
```

4.5.159.GRANT

功能描述

对角色和用户进行授权操作。

使用 GRANT 命令进行用户授权包括以下场景：

- **将系统权限授权给角色或用户**

  系统权限又称为用户属性，包括 SYSADMIN、CREATEDB、CREATEROLE、AUDITADMIN、MONADMIN、OPRADMIN、POLADMIN、INHERIT、REPLICATION、VCADMIN 和 LOGIN 等。

  系统权限一般通过 CREATE/ALTER ROLE 语法来指定。其中，SYSADMIN 权限可以通过 GRANT/REVOKE ALL PRIVILEGE 授予或撤销。但系统权限无法通过 ROLE 和 USER 的权限被继承，也无法授予 PUBLIC。

  有关系统权限的详细信息，请参见[CREATE-ROLE]

- **将数据库对象授权给角色或用户**

  将数据库对象（表和视图、指定字段、数据库、函数、模式、表空间等）的相关权限授予特定角色或用户；

GRANT 命令将数据库对象的特定权限授予一个或多个角色。这些权限会追加到已有的权限上。

关键字 PUBLIC 表示该权限要赋予所有角色，包括以后创建的用户。PUBLIC 可以看做是一个隐含定义好的组，它总是包括所有角色。任何角色或用户都将拥有通过 GRANT 直接赋予的权限和所属的权限，再加上 PUBLIC 的权限。

如果声明了 WITH GRANT OPTION，则被授权的用户也可以将此权限赋予他人，否则就不能授权给他人。这个选项不能赋予 PUBLIC，这是 PanWeiDB 特有的属性。

PanWeiDB 会将某些类型的对象上的权限授予 PUBLIC。默认情况下，对表、表字段、序列、外部数据源、外部服务器、模式或表空间对象的权限不会授予 PUBLIC，而以下这些对象的权限会授予 PUBLIC：数据库的 CONNECT 权限和 CREATE TEMP TABLE 权限、函数的 EXECUTE 特权、语言和数据类型（包括域）的 USAGE 特权。当然，对象拥有者可以撤销默认授予 PUBLIC 的权限并专门授予权限给其他用户。为了更安全，建议在同一个事务中创建对象并设置权限，这样其他用户就没有时间窗口使用该对象。另外可参考安全加固指南的权限控制章节，对 PUBLIC 用户组的权限进行限制。这些初始的默认权限可以使用 ALTER DEFAULT PRIVILEGES 命令修改。

对象的所有者缺省具有该对象上的所有权限，出于安全考虑所有者可以舍弃部分权限，但 ALTER、DROP、COMMENT、INDEX、VACUUM 以及对象的可再授予权限属于所有者固有的权限，隐式拥有。

- **将角色或用户的权限授权给其他角色或用户**

将一个角色或用户的权限授予一个或多个其他角色或用户。在这种情况下，每个角色或用户都可视为拥有一个或多个数据库权限的集合。

当声明了 WITH ADMIN OPTION，被授权的用户可以将该权限再次授予其他角色或用户，以及撤销所有由该角色或用户继承到的权限。当授权的角色或用户发生变更或被撤销时，所有继承该角色或用户权限的用户拥有的权限都会随之发生变更。

数据库系统管理员可以给任何角色或用户授予/撤销任何权限。拥有 CREATE ROLE 权限的角色可以赋予或者撤销任何非系统管理员角色的权限。

- **将 ANY 权限授予给角色或用户**

  将 ANY 权限授予特定的角色和用户，ANY 权限的取值范围参见语法格式。当声明了 WITH ADMIN OPTION，被授权的用户可以将该 ANY 权限再次授予其他角色/用户，或从其他角色/用户处回收该 ANY 权限。ANY 权限可以通过角色被继承，但不能赋予 PUBLIC。初始用户和三权分立关闭时的系统管理员用户可以给任何角色/用户授予或撤销 ANY 权限。

  目前支持以下 ANY 权限：CREATE ANY TABLE、ALTER ANY TABLE、DROP ANY TABLE、SELECT ANY TABLE、INSERT ANY TABLE、UPDATE ANY TABLE、DELETE ANY TABLE、CREATE ANY SEQUENCE、CREATE ANY INDEX、CREATE ANY FUNCTION、EXECUTE ANY FUNCTION、CREATE ANY PACKAGE、EXECUTE ANY PACKAGE、CREATE ANY TYPE。详细的 ANY 权限范围描述参考表 1：ANY 权限列表。

注意事项

- 不允许将 ANY 权限授予 PUBLIC，也不允许从 PUBLIC 回收 ANY 权限。
- ANY 权限属于数据库内的权限，只对授予该权限的数据库内的对象有效，例如 SELECT ANY TABLE 只允许用户查看当前数据库内的所有用户表数据，对其他数据库内的用户表无查看权限。
- 即使用户被授予 ANY 权限，也不能对私有用户下的对象进行访问操作（INSERT、DELETE、UPDATE、SELECT）。
- ANY 权限与原有的权限相互无影响。
- 如果用户被授予 CREATE ANY TABLE 权限，在同名 schema 下创建表的属主是该 schema 的创建者，用户对表进行其他操作时，需要授予相应的操作权限。
- 需要谨慎授予用户 CREATE ANY FUNCTION 或 CREATE ANY PACKAGE 的权限，以免其他用户利用 DEFINER 类型的函数或 PACKAGE 进行权限提升。

语法格式

❖ 将表或视图的访问权限赋予指定的用户或角色。

```
GRANT { { SELECT | INSERT | UPDATE | DELETE | TRUNCATE | REFERENCES | ALTER | DROP
| COMMENT | INDEX | VACUUM } [, ...]  | ALL [ PRIVILEGES ] }
    ON { [ TABLE ] table_name [, ...]
      | ALL TABLES IN SCHEMA schema_name [, ...] }
    TO { [ GROUP ] role_name | PUBLIC } [, ...]
    [ WITH GRANT OPTION ];
```

❖ 将表中字段的访问权限赋予指定的用户或角色。

```
GRANT { {{ SELECT | INSERT | UPDATE | REFERENCES | COMMENT } ( column_name [, ...]
)} [, ...]
     | ALL [ PRIVILEGES ] ( column_name [, ...] ) }
    ON [ TABLE ] table_name [, ...]
    TO { [ GROUP ] role_name | PUBLIC } [, ...]
    [ WITH GRANT OPTION ];
```

❖ 将序列的访问权限赋予指定的用户或角色，LARGE 字段属性可选，赋权语句不区分序列是否为 LARGE。

```
GRANT { { SELECT | UPDATE | USAGE | ALTER | DROP | COMMENT } [, ...]
     | ALL [ PRIVILEGES ] }
    ON { [ [ LARGE ] SEQUENCE ] sequence_name [, ...]
      | ALL SEQUENCES IN SCHEMA schema_name [, ...] }
    TO { [ GROUP ] role_name | PUBLIC } [, ...]
    [ WITH GRANT OPTION ];
```

❖ 将数据库的访问权限赋予指定的用户或角色。

```
GRANT { { CREATE | CONNECT | TEMPORARY | TEMP | ALTER | DROP | COMMENT } [, ...]
     | ALL [ PRIVILEGES ] }
    ON DATABASE database_name [, ...]
    TO { [ GROUP ] role_name | PUBLIC } [, ...]
    [ WITH GRANT OPTION ];
```

❖ 将域的访问权限赋予指定的用户或角色。

```
GRANT { USAGE | ALL [ PRIVILEGES ] }
    ON DOMAIN domain_name [, ...]
    TO { [ GROUP ] role_name | PUBLIC } [, ...]
    [ WITH GRANT OPTION ];
```

【说明】

本版本暂时不支持赋予域的访问权限。

❖ 将客户端加密主密钥 CMK 的访问权限赋予指定的用户或角色。

```
GRANT { { USAGE | DROP } [, ...] | ALL [ PRIVILEGES ] }
    ON CLIENT_MASTER_KEY client_master_key [, ...]
    TO { [ GROUP ] role_name | PUBLIC } [, ...]
    [ WITH GRANT OPTION ];
```

❖ 将列加密密钥 CEK 的访问权限赋予指定的用户或角色。

```
GRANT { { USAGE | DROP } [, ...] | ALL [ PRIVILEGES ] }
    ON COLUMN_ENCRYPTION_KEY column_encryption_key [, ...]
    TO { [ GROUP ] role_name | PUBLIC } [, ...]
    [ WITH GRANT OPTION ];
```

❖ 将外部数据源的访问权限赋予给指定的用户或角色。

```
GRANT { USAGE | ALL [ PRIVILEGES ] }
    ON FOREIGN DATA WRAPPER fdw_name [, ...]
    TO { [ GROUP ] role_name | PUBLIC } [, ...]
    [ WITH GRANT OPTION ];
```

❖ 将外部服务器的访问权限赋予给指定的用户或角色。

```
GRANT { { USAGE | ALTER | DROP | COMMENT } [, ...] | ALL [ PRIVILEGES ] }
    ON FOREIGN SERVER server_name [, ...]
    TO { [ GROUP ] role_name | PUBLIC } [, ...]
    [ WITH GRANT OPTION ];
```

❖ 将函数的访问权限赋予给指定的用户或角色。

```
GRANT { { EXECUTE | ALTER | DROP | COMMENT } [, ...] | ALL [ PRIVILEGES ] }
    ON { FUNCTION {function_name ( [ {[ argmode ] [ arg_name ] arg_type} [, ...] ] )} [,
...]
      | ALL FUNCTIONS IN SCHEMA schema_name [, ...] }
    TO { [ GROUP ] role_name | PUBLIC } [, ...]
    [ WITH GRANT OPTION ];
```

❖ 将存储过程的访问权限赋予给指定的用户或角色。

```
 GRANT { { EXECUTE | ALTER | DROP | COMMENT } [, ...] | ALL [ PRIVILEGES ] }
   ON { PROCEDURE {proc_name ( [ {[ argmode ] [ arg_name ] arg_type} [, ...] ] )} [,
...]}
   TO { [ GROUP ] role_name | PUBLIC } [, ...]
```

```
    [ WITH GRANT OPTION ];
```

❖ 将过程语言的访问权限赋予给指定的用户或角色。

```
GRANT { USAGE | ALL [ PRIVILEGES ] }
    ON LANGUAGE lang_name [, ...]
    TO { [ GROUP ] role_name | PUBLIC } [, ...]
    [ WITH GRANT OPTION ];
```

❖ 将大对象的访问权限赋予指定的用户或角色。

```
GRANT { { SELECT | UPDATE } [, ...] | ALL [ PRIVILEGES ] }
    ON LARGE OBJECT loid [, ...]
    TO { [ GROUP ] role_name | PUBLIC } [, ...]
    [ WITH GRANT OPTION ];
```

【说明】

本版本暂时不支持大对象。

❖ 将模式的访问权限赋予指定的用户或角色。

```
GRANT { { CREATE | USAGE | ALTER | DROP | COMMENT } [, ...] | ALL [ PRIVILEGES ] }
    ON SCHEMA schema_name [, ...]
    TO { [ GROUP ] role_name | PUBLIC } [, ...]
    [ WITH GRANT OPTION ];
```

【说明】

将模式中的表或者视图对象授权给其他用户时，需要将表或视图所属的模式的 USAGE 权限同时授予该用户，若没有该权限，则只能看到这些对象的名称，并不能实际进行对象访问。 同名模式下创建表的权限无法通过此语法赋予，可以通过将角色的权限赋予其他用户或角色的语法，赋予同名模式下创建表的权限。

❖ 将表空间的访问权限赋予指定的用户或角色。

```
GRANT { { CREATE | ALTER | DROP | COMMENT } [, ...] | ALL [ PRIVILEGES ] }
    ON TABLESPACE tablespace_name [, ...]
    TO { [ GROUP ] role_name | PUBLIC } [, ...]
    [ WITH GRANT OPTION ];
```

❖ 将类型的访问权限赋予指定的用户或角色。

```
GRANT { { USAGE | ALTER | DROP | COMMENT } [, ...] | ALL [ PRIVILEGES ] }
    ON TYPE type_name [, ...]
    TO { [ GROUP ] role_name | PUBLIC } [, ...]
```

```
    [ WITH GRANT OPTION ];
```

【说明】

本版本暂时不支持赋予类型的访问权限。

❖ 将 Data Source 对象的权限赋予指定的角色。

```
GRANT { USAGE | ALL [PRIVILEGES]}
    ON DATA SOURCE src_name [, ...]
    TO { [GROUP] role_name | PUBLIC } [, ...]
    [WITH GRANT OPTION];
```

❖ 将 directory 对象的权限赋予指定的角色。

```
GRANT { { READ | WRITE | ALTER | DROP } [, ...] | ALL [PRIVILEGES] }
    ON DIRECTORY directory_name [, ...]
    TO { [GROUP] role_name | PUBLIC } [, ...]
    [WITH GRANT OPTION];
```

❖ 将 package 对象的权限赋予指定的角色。

```
GRANT { { EXECUTE | ALTER | DROP | COMMENT } [, ...] | ALL [PRIVILEGES] }
    ON PACKAGE package_name [, ...]
    TO { [GROUP] role_name | PUBLIC } [, ...]
    [WITH GRANT OPTION];
```

❖ 将角色的权限赋予其他用户或角色的语法。

```
GRANT role_name [, ...]
    TO role_name [, ...]
    [ WITH ADMIN OPTION ];
```

❖ 将 sysadmin 权限赋予指定的角色。

```
GRANT ALL { PRIVILEGES | PRIVILEGE }
    TO role_name;
```

❖ 将 ANY 权限赋予其他用户或角色的语法。

```
 GRANT { CREATE ANY TABLE | ALTER ANY TABLE | DROP ANY TABLE | SELECT ANY TAB
LE | INSERT ANY TABLE | UPDATE ANY TABLE |
  DELETE ANY TABLE | CREATE ANY SEQUENCE | CREATE ANY INDEX | CREATE ANY FU
NCTION | EXECUTE ANY FUNCTION |
  CREATE ANY PACKAGE | EXECUTE ANY PACKAGE | CREATE ANY TYPE } [, ...]
  TO [ GROUP ] role_name [, ...]
  [ WITH ADMIN OPTION ];
```

参数说明

GRANT 的权限分类如下所示。

- **SELECT**

  允许对指定的表、视图、序列执行 SELECT 命令，update 或 delete 时也需要对应字段上的 select 权限。

- **INSERT**

  允许对指定的表执行 INSERT 命令。

- **UPDATE**

  允许对声明的表中任意字段执行 UPDATE 命令。通常，update 命令也需要 select 权限来查询出哪些行需要更新。SELECT... FOR UPDATE、SELECT... FOR NO KEY UPDATE、SELECT... FOR SHARE 和 SELECT... FOR KEY SHARE 除了需要 SELECT 权限外，还需要 UPDATE 权限。

- **DELETE**

  允许执行 DELETE 命令删除指定表中的数据。通常，delete 命令也需要 select 权限来查询出哪些行需要删除。

- **TRUNCATE**

  允许执行 TRUNCATE 语句删除指定表中的所有记录。

- **REFERENCES**

  创建一个外键约束，必须拥有参考表和被参考表的 REFERENCES 权限。

- **CREATE**

  - 对于数据库，允许在数据库里创建新的模式。
  - 对于模式，允许在模式中创建新的对象。如果要重命名一个对象，用户除了必须是该对象的所有者外，还必须拥有该对象所在模式的 CREATE 权限。
  - 对于表空间，允许在表空间中创建表，允许在创建数据库和模式的时候把该表空间指定为缺省表空间。

- **CONNECT**

  允许用户连接到指定的数据库。

- **TEMPORARY**

  允许在指定的数据库中创建临时表。

- **TEMP**

  TEMPORARY 的缩略拼写。

- **EXECUTE**

  允许使用指定的函数，以及利用这些函数实现的操作符。

- **USAGE**

  - 对于过程语言，允许用户在创建函数的时候指定过程语言。
  - 对于模式，USAGE 允许访问包含在指定模式中的对象，若没有该权限，则只能看到这些对象的名称。
  - 对于序列，USAGE 允许使用 nextval 函数。
  - 对于 Data Source 对象，USAGE 是指访问权限，也是可赋予的所有权限，即 USAGE 与 ALL PRIVILEGES 等价。

- **ALTER**

  允许用户修改指定对象的属性，但不包括修改对象的所有者和修改对象所在的模式。

- **DROP**

  允许用户删除指定的对象。

- **COMMENT**

  允许用户定义或修改指定对象的注释。

- **INDEX**

  允许用户在指定表上创建索引，并管理指定表上的索引，还允许用户对指定表执行 REINDEX 和 CLUSTER 操作。

- **VACUUM**

  允许用户对指定的表执行 ANALYZE 和 VACUUM 操作。

- **ALL PRIVILEGES**

  一次性给指定用户/角色赋予所有可赋予的权限。只有系统管理员有权执行 GRANT ALL PRIVILEGES。

GRANT 的参数说明如下所示。

- **role_name**

  已存在用户名称。

- **table_name**

已存在表名称。

- **column_name**

  已存在字段名称。

- **schema_name**

  已存在模式名称。

- **database_name**

  已存在数据库名称。

- **function_name**

  已存在函数名称。

- **procedure_name**

  已存在存储过程名称。

- **sequence_name**

  已存在序列名称。

- **domain_name**

  已存在域类型名称。

- **fdw_name**

  已存在外部数据包名称。

- **lang_name**

  已存在语言名称。

- **type_name**

  已存在类型名称。

- **src_name**

  已存在的 Data Source 对象名称。

- **argmode**

  参数模式。

  取值范围：字符串，要符合标识符命名规范。

- **arg_name**

  参数名称。

  取值范围：字符串，要符合标识符命名规范。

- **arg_type**

  参数类型。

取值范围：字符串，要符合标识符命名规范。

- **loid**

  包含本页的大对象的标识符。

  取值范围：字符串，要符合标识符命名规范。

- **tablespace_name**

  表空间名称。

- **client_master_key**

  客户端加密主密钥的名称。

  取值范围：字符串，要符合标识符命名规范。

- **column_encryption_key**

  列加密密钥的名称。

  取值范围：字符串，要符合标识符命名规范。

- **directory_name**

  目录名称。

  取值范围：字符串，要符合标识符命名规范。

- **WITH GRANT OPTION**

  如果声明了 WITH GRANT OPTION，则被授权的用户也可以将此权限赋予他人，否则就不能授权给他人。这个选项不能赋予 PUBLIC。

非对象所有者给其他用户授予对象权限时，命令按照以下规则执行：

- 如果用户没有该对象上指定的权限，命令立即失败。
- 如果用户有该对象上的部分权限，则 GRANT 命令只授予他有授权选项的权限。
- 如果用户没有可用的授权选项，GRANT ALL PRIVILEGES 形式将发出一个警告信息，其他命令形式将发出在命令中提到的且没有授权选项的相关警告信息。

说明：

数据库系统管理员可以访问所有对象，而不会受对象的权限设置影响。这个特点类似 Unix 系统的 root 的权限。和 root 一样，除了必要的情况外，建议不要总是以系统管理员身份进行操作。

不允许对表分区进行 GRANT 操作，对分区表进行 GRANT 操作会引起告警。

- WITH ADMIN OPTION

对于角色，当声明了 WITH ADMIN OPTION，被授权的用户可以将该角色再授予其他角色/用户，或从其他角色/用户回收该角色。

对于 ANY 权限，当声明了 WITH ADMIN OPTION，被授权的用户可以将该 ANY 权限再授予其他角色/用户，或从其他角色/用户回收该 ANY 权限。

表 1 ANY 权限列表

系统权限名称	描述
CREATE ANY TABLE	用户能够在 public 模式和用户模式下创建表或视图。如果想要创建 serial 类型列的表，还需要授予创建序列的权限。
ALTER ANY TABLE	用户拥有对 public 模式和用户模式下表或视图的 ALTER 权限。如果想要修改表的唯一索引为表增加主键约束或唯一约束，还需要授予该表的索引权限。
DROP ANY TABLE	用户拥有对 public 模式和用户模式下表或视图的 DROP 权限。
SELECT ANY TABLE	用户拥有对 public 模式和用户模式下表或视图的 SELETCT 权限，仍然受行级访问控制限制。
UPDATE ANY TABLE	用户拥有对 public 模式和用户模式下表或视图的 UPDATE 权限，仍然受行级访问控制限制。

系统权限名称	描述
INSERT ANY TABLE	用户拥有对 public 模式和用户模式下表或视图的 INSERT 权限。
DELETE ANY TABLE	用户拥有对 public 模式和用户模式下表或视图的 DELETE 权限，仍然受行级访问控制限制。
CREATE ANY FUNCTION	用户能够在用户模式下创建函数或存储过程。
EXECUTE ANY FUNCTION	用户拥有在 public 模式和用户模式下函数或存储过程的 EXECUTE 权限。
CREATE ANY PACKAGE	用户能够在 public 模式和用户模式下创建 PACKAGE。
EXECUTE ANY PACKAGE	用户拥有在 public 模式和用户模式下 PACKAGE 的 EXECUTE 权限。
CREATE ANY TYPE	用户能够在 public 模式和用户模式下创建类型。
CREATE ANY SEQUENCE	用户能够在 public 模式和用户模式下创建序列。
CREATE ANY INDEX	用户能够在 public 模式和用户模式下创建索引。如果在某表空间创建分区表索引，需要授予用户该表空间的创建权

系统权限名称	描述
	限。

说明：

用户被授予任何一种 ANY 权限后，用户对 public 模式和用户模式具有 USAGE 权限，对除 public 之外的系统模式没有 USAGE 权限。

示例

示例 1：将系统权限授权给用户或者角色。

创建名为 joe 的用户，并将 sysadmin 权限授权给他。

```
panweidb=# CREATE USER joe PASSWORD 'xxxxxxxxx';
panweidb=# GRANT ALL PRIVILEGES TO joe;
```

授权成功后，用户 joe 会拥有 sysadmin 的所有权限。

示例 2：将对象权限授权给用户或者角色。

1、销 joe 用户的 sysadmin 权限，然后将模式 tpcds 的使用权限和表 tpcds.reason 的所有权限授权给用户 joe。

```
panweidb=# REVOKE ALL PRIVILEGES FROM joe;
panweidb=# GRANT USAGE ON SCHEMA tpcds TO joe;
panweidb=# GRANT ALL PRIVILEGES ON tpcds.reason TO joe;
```

授权成功后，joe 用户就拥有了 tpcds.reason 表的所有权限，包括增删改查等权限。

2、将 tpcds.reason 表中 r_reason_sk、r_reason_id、r_reason_desc 列的查询权限，r_reason_desc 的更新权限授权给 joe。

```
panweidb=# GRANT select (r_reason_sk,r_reason_id,r_reason_desc),update (r_reason_desc) ON tpcds.reason TO joe;
```

授权成功后，用户 joe 对 tpcds.reason 表中 r_reason_sk，r_reason_id 的查询权限会立即生效。如果 joe 用户需要拥有将这些权限授权给其他用户的权限，可以通过以下语法对 joe 用户进行授权。

```
panweidb=# GRANT select (r_reason_sk, r_reason_id) ON tpcds.reason TO joe WITH GRANT OPTION;
```

将数据库 PanWeiDB 的连接权限授权给用户 joe，并给予其在 PanWeiDB 中创建 schema 的权限，而且允许 joe 将此权限授权给其他用户。

```
panweidb=# GRANT create,connect on database PanWeiDB TO joe WITH GRANT OPTION;
```

创建角色 tpcds_manager，将模式 tpcds 的访问权限授权给角色 tpcds_manager，并授予该角色在 tpcds 下创建对象的权限，不允许该角色中的用户将权限授权给其他人。

```
panweidb=# CREATE ROLE tpcds_manager PASSWORD 'xxxxxxxxx';
panweidb=# GRANT USAGE,CREATE ON SCHEMA tpcds TO tpcds_manager;
```

将表空间 tpcds_tbspc 的所有权限授权给用户 joe，但用户 joe 无法将权限继续授予其他用户。

```
panweidb=# CREATE TABLESPACE tpcds_tbspc RELATIVE LOCATION 'tablespace/tablespace_1';
panweidb=# GRANT ALL ON TABLESPACE tpcds_tbspc TO joe;
```

示例 3：将用户或者角色的权限授权给其他用户或角色。

1、创建角色 manager，将 joe 的权限授权给 manager，并允许该角色将权限授权给其他人。

```
panweidb=# CREATE ROLE manager PASSWORD 'xxxxxxxxx';
panweidb=# GRANT joe TO manager WITH ADMIN OPTION;
```

2、创建用户 senior_manager，将用户 manager 的权限授权给该用户。

```
panweidb=# CREATE ROLE senior_manager PASSWORD 'xxxxxxxxx';
panweidb=# GRANT manager TO senior_manager;
```

3、撤销权限，并清理用户。

```
panweidb=# REVOKE manager FROM joe;
panweidb=# REVOKE senior_manager FROM manager;
panweidb=# DROP USER manager;
```

示例 4：将 CMK 或者 CEK 的权限授权给其他用户或角色。

1、连接密态数据库。

```
gsql -r
panweidb=#  CREATE CLIENT MASTER KEY MyCMK1 WITH ( KEY_STORE = localkms , K
EY_PATH = "key_path_value" , ALGORITHM = RSA_2048);
CREATE CLIENT MASTER KEY
panweidb=# CREATE COLUMN ENCRYPTION KEY MyCEK1 WITH VALUES (CLIENT_MAS
TER_KEY = MyCMK1, ALGORITHM = AEAD_AES_256_CBC_HMAC_SHA256);
CREATE COLUMN ENCRYPTION KEY
```

2、创建角色 newuser，将密钥的权限授权给 newuser。

```
panweidb=# CREATE USER newuser PASSWORD 'xxxxxxxxx';
CREATE ROLE
panweidb=# GRANT ALL ON SCHEMA public TO newuser;
GRANT
panweidb=# GRANT USAGE ON COLUMN_ENCRYPTION_KEY MyCEK1 to newuser;
GRANT
panweidb=# GRANT USAGE ON CLIENT_MASTER_KEY MyCMK1 to newuser;
GRANT
```

3、设置该用户连接数据库，使用该 CEK 创建加密表。

```
panweidb=# SET SESSION AUTHORIZATION newuser PASSWORD 'xxxxxxxxx';
panweidb=>  CREATE TABLE acltest1 (x int, x2 varchar(50) ENCRYPTED WITH (COLUM
N_ENCRYPTION_KEY = MyCEK1, ENCRYPTION_TYPE = DETERMINISTIC));
CREATE TABLE
panweidb=> SELECT cm_cek_privilege('newuser', 'MyCEK1', 'USAGE');
 cm_cek_privilege
------------------
 t
(1 row)
```

4、撤销权限，并清理用户。

```
panweidb=# REVOKE USAGE ON COLUMN_ENCRYPTION_KEY MyCEK1 FROM newuser;
panweidb=# REVOKE USAGE ON CLIENT_MASTER_KEY MyCMK1 FROM newuser;
panweidb=# DROP TABLE newuser.acltest1;
panweidb=# DROP COLUMN ENCRYPTION KEY MyCEK1;
panweidb=# DROP CLIENT MASTER KEY MyCMK1;
panweidb=# DROP SCHEMA IF EXISTS newuser CASCADE;
panweidb=# REVOKE ALL ON SCHEMA public FROM newuser;
panweidb=# DROP ROLE IF EXISTS newuser;
```

示例：撤销上述授予的权限，并清理角色和用户。

```
panweidb=# REVOKE ALL PRIVILEGES ON tpcds.reason FROM joe;
panweidb=# REVOKE ALL PRIVILEGES ON SCHEMA tpcds FROM joe;
panweidb=# REVOKE ALL ON TABLESPACE tpcds_tbspc FROM joe;
panweidb=# DROP TABLESPACE tpcds_tbspc;
panweidb=# REVOKE USAGE,CREATE ON SCHEMA tpcds FROM tpcds_manager;
panweidb=# DROP ROLE tpcds_manager;
panweidb=# DROP ROLE senior_manager;
panweidb=# DROP USER joe CASCADE;
```

相关链接

参考：SQL 语法参考->SQL 语法->REVOKE、ALTER DEFAULT PRIVILEGES 章节。

4.5.160.INSERT

功能描述

向表中添加一行或多行数据。

注意事项

- 只有拥有表 INSERT 权限的用户，才可以向表中插入数据。用户被授予 insert any table 权限，相当于用户对除系统模式之外的任何模式具有 USAGE 权限，并且拥有这些模式下表的 INSERT 权限
- 如果使用 RETURNING 子句，用户必须要有该表的 SELECT 权限。

- 如果使用 ON DUPLICATE KEY UPDATE，用户必须要有该表的 SELECT、UPDATE 权限，唯一约束（主键或唯一索引）的 SELECT 权限。
- 如果使用 query 子句插入来自查询里的数据行，用户还需要拥有在查询里使用的表的 SELECT 权限。
- 生成列不能被直接写入。在 INSERT 命令中不能为生成列指定值，但是可以指定关键字 DEFAULT。
- 通过 insert 语句批量插入数据时，建议将多条记录合并入一条语句中执行插入，以提高数据加载性能。例如：

```
INSERT INTO sections VALUES (30, 'DatabaseAdministrationGuide', 31, 1900)、
(40, 'Development', 35, 2000)、 (50, 'Development' , 60 , 2001);
```

- 当连接到 TD 兼容模式的数据库，td_compatible_truncation 参数设置为 on 时，将启用超长字符串自动截断功能，在后续的 insert 语句中（不包含外表的场景下），对目标表中 char 和 varchar 类型的列上插入超长字符串时，系统会自动按照目标表中相应列定义的最大长度对超长字符串进行截断。

  【说明】

  如果向字符集为字节类型编码（SQL_ASCII，LATIN1 等）的数据库中插入多字节字符数据（如汉字等），且字符数据跨越截断位置，这种情况下，按照字节长度自动截断，自动截断后会在尾部产生非预期结果。如果用户有对于截断结果正确性的要求，建议用户采用 UTF8 等能够按照字符截断的输入字符集作为数据库的编码集。

语法格式

```
[ WITH [ RECURSIVE ] with_query [, ...] ]
INSERT [/*+ plan_hint */] [IGNORE] INTO table_name [ @dblink_name ] [partition_cla
use] [ AS alias ] [ ( column_name [, ...] ) ]
  { DEFAULT VALUES | VALUES | VALUE {( { expression | DEFAULT } [, ...] ) }[, ...] | query }
  [ upsert_clause ]
  [ RETURNING {* | {output_expression [ [ AS ] output_name ] }[, ...]} ];
INSERT [ FIRST | ALL ]
```

```
[ WHEN { condition } THEN INTO table [ [ AS ] alias ] [ ( column_name [, ...] ) ] VALUES
(v1, ...), ...]
{ subquery }
```

【说明】

IGNORE 选项仅在 MySQL 兼容模式下可用，更多内容可参考《PanWeiDB V2.0-S3.0.1_B01_MySQL 兼容性手册》-> INSERT 章节。

- 其中 upsert_clause 可以是下列之一：

```
ON DUPLICATE KEY UPDATE { NOTHING | { column_name = { expression | DEFAU
LT } } [, ...] }
ON CONFLICT [ conflict_target ] conflict_action
```

  - 其中 conflict_target 可以是下列之一：

```
( { index_column_name | ( index_expression ) } [ COLLATE collation ] [ opcla
ss ] [, ...] ) [ WHERE index_predicate ]
ON CONSTRAINT constraint_name
```

  - 其中 conflict_action 可以是下列之一：

```
DO NOTHING
DO UPDATE SET { column_name = { expression | DEFAULT } |
        ( column_name [, ...] ) = [ ROW ] ( { expression | DEFAULT } [, ...] ) |
        ( column_name [, ...] ) = ( sub-SELECT )
      } [, ...]
    [ WHERE condition ]
```

- 其中 with_query 可以是：

```
with_query_name [ ( column_name [, ...] ) ] AS [ [ NOT ] MATERIALIZED ]
( {select | values | insert | update | delete} )
```

- 其中 partition_clause 可以是：

```
PARTITION { ( partition_name ) | FOR ( partition_value [, ...] ) } |
SUBPARTITION { ( subpartition_name ) | FOR ( subpartition_value [, ...] ) }
```

【说明】

其中 partition_clause 仅在集中式模式支持。

参数说明

❖ **WITH [RECURSIVE] with_query [, ...]**

用于声明一个或多个可以在主查询中通过名称引用的子查询，相当于临时表。

如果声明了 RECURSIVE，那么允许 SELECT 子查询通过名称引用它自己。

- with_query_name 指定子查询生成的结果集名称，在查询中可使用该名称访问子查询的结果集。
- column_name 指定子查询结果集中显示的列名。
- 每个子查询可以是 SELECT，VALUES，INSERT，UPDATE 或 DELETE 语句。
- 用户可以使用 MATERIALIZED / NOT MATERIALIZED 对 CTE（Common Table Expression，即在 SQL 中构建一个临时数据集，给 SQL 的其他部分语义使用）进行修饰。
  - 如果声明为 MATERIALIZED，WITH 查询将被物化，生成一个子查询结果集的拷贝，在引用处直接查询该拷贝，因此 WITH 子查询无法和主干 SELECT 语句进行联合优化（如谓词下推、等价类传递等），对于此类场景可以使用 NOT MATERIALIZED 进行修饰，如果 WITH 查询语义上可以作为子查询内联执行，则可以进行上述优化。
  - 如果用户没有显示声明物化属性则遵守以下规则：如果 CTE 只在所属主干语句中被引用一次，且语义上支持内联执行，则会被改写为子查询内联执行，否则以 CTE Scan 的方式物化执行。

【说明】
INSERT ON DUPLICATE KEY UPDATE 不支持 WITH 及 WITH RECURSIVE 子句。

❖ **plan_hint 子句**

以/*+ */的形式在 INSERT 关键字后，用于对 INSERT 对应的语句块生成的计划进行 hint 调优，详细用法请参见章节使用 Plan Hint 进行调优。

每条语句中只有第一个/*+ plan_hint */注释块会作为 hint 生效，里面可以写多条 hint。

- **table_name**

  要插入数据的目标表名。

  取值范围：已存在的表名。

- **partition_clause**

  指定分区插入操作

  ```
  PARTITION { ( partition_name ) | FOR ( partition_value [, ...] ) } |SUBPARTITION { ( subpartition_name ) | FOR ( subpartition_value [, ...] ) }
  ```

  关键字详见 SELECT 一节介绍。 如果 value 子句的值和指定分区不一致，会抛出异常。 示例参见 CREATE TABLE SUBPARTITION 的示例。

- **column_name**

  目标表中的字段名：

  - 字段名可以有子字段名或者数组下标修饰。
  - 没有在字段列表中出现的每个字段，将由系统默认值，或者声明时的默认值填充，若都没有则用 NULL 填充。例如，向一个复合类型中的某些字段插入数据的话，其他字段将是 NULL。
  - 目标字段（column_name）可以按顺序排列。如果没有列出任何字段，则默认全部字段，且顺序为表声明时的顺序。
  - 如果 value 子句和 query 中只提供了 N 个字段，则目标字段为前 N 个字段。
  - value 子句和 query 提供的值在表中从左到右关联到对应列。

  取值范围：已存在的字段名。

- **expression**

  赋予对应 column 的一个有效表达式或值：

  - 如果是 INSERT ON DUPLICATE KEY UPDATE 语句下，expression 可以为 VALUES(column_name)或 EXCLUDED.column_name 用来表示引用冲突行对应的 column_name 字段的值。需注意，其中 VALUES(column_name) 不支持嵌套在表达式中（例如 VALUES(column_name)+1），但 EXCLUDED 不受此限制。

- 向表中字段插入单引号 '时需要使用单引号自身进行转义。
- 如果插入行的表达式不是正确的数据类型，系统试图进行类型转换，若转换不成功，则插入数据失败，系统返回错误信息。

❖ **DEFAULT**

对应字段名的缺省值。如果没有缺省值，则为 NULL。

❖ **query**

一个查询语句（SELECT 语句），将查询结果作为插入的数据。

❖ **RETURNING**

返回实际插入的行，RETURNING 列表的语法与 SELECT 的输出列表一致。

```
[ RETURNING {* | {output_expression [ [ AS ] output_name ] }[, ...]} ]
```

【说明】

❖ INSERT ON DUPLICATE KEY UPDATE 不支持 RETURNING 子句。

❖ 在 SQL Server 兼容模式下支持 RETURNING 子句中的列名大小写不敏感的特性，详见 SQL Server 兼容性手册中的 INSERT。

❖ **output_expression**

INSERT 命令在每一行都被插入之后用于计算输出结果的表达式。

取值范围：该表达式可以使用 table 的任意字段。可以使用*返回被插入行的所有字段。

❖ **output_name**

字段的输出名称。

取值范围：字符串，符合标识符命名规范。

❖ **ON DUPLICATE KEY UPDATE**

- 对于带有唯一约束（UNIQUE INDEX 或 PRIMARY KEY）的表，如果插入数据违反唯一约束，则对冲突行执行 UPDATE 子句完成更新，对于不带唯一约束的表，则仅执行插入。UPDATE 时，若指定 NOTHING 则忽略此条插入，可通过 EXCLUDE.或者 VALUES()来选择源数据相应的列。
- 支持触发器，触发器执行顺序由实际执行流程决定：
  - 执行 insert：触发 before insert、after insert 触发器。

- 执行 update：触发 before insert、before update、after update 触发器。
- 执行 update nothing：触发 before insert 触发器。

➢ 不支持延迟生效（DEFERRABLE）的唯一约束或主键。

➢ 如果表中存在多个唯一约束，如果所插入数据违反多个唯一约束，对于检测到冲突的第一行进行更新，其他冲突行不更新（检查顺序与索引维护具有强相关性，一般先创建的索引先进行冲突检查）。

➢ 如果插入多行，这些行均与表中同一行数据存在唯一约束冲突，则按照顺序，第一条执行插入或更新，之后依次执行更新。

➢ 主键、唯一索引列不允许 UPDATE。

➢ 不支持列存，不支持外表、内存表。

➢ expression 支持使用子查询表达式，其语法与功能同 UPDATE。子查询表达式中支持使用 EXCLUDED.来选择源数据相应的列。

❖ **ON CONFLICT [conflict_target] conflict_action**

PG 风格的 ON CONFLICT 子句指定插入数据违反唯一约束时的替换动作，无冲突时直接插入，存在冲突时执行 UPDATE，因此也被称为 UPSERT 功能--“UPDATE 或 INSERT”，用法介绍详见 UPSERT 功能。

示例

1、创建表 reason_t2。

```
CREATE TABLE reason_t2
(
 r_reason_sk   integer,
 r_reason_id   character(16),
 r_reason_desc  character(100)
);
```

2、向表中插入一条记录并查询。

```
INSERT INTO reason_t2(r_reason_sk, r_reason_id, r_reason_desc) VALUES (1, 'A
AAAAAAABAAAAAAA', 'reason1');
SELECT * FROM reason_t2;
```

查询结果为：

```
r_reason_sk| r_reason_id   |                    r_reason_
```

```
desc
------------+-----------------+-------------------------------------------------
--
-----------------------------------------------
       1 | AAAAAAAABAAAAAAA | reason1

(1 row)
```

3、向表中插入一条记录并查询，和上一条语法等效。

```
INSERT INTO reason_t2 VALUES (2, 'AAAAAAAABAAAAAAA', 'reason2');
SELECT * FROM reason_t2;
```

查询结果为：

```
r_reason_sk|  r_reason_id   |                          r_reason_
desc
------------+-----------------+-------------------------------------------------
--
-----------------------------------------------
       1 | AAAAAAAABAAAAAAA | reason1

       2 | AAAAAAAABAAAAAAA | reason2

(2 rows)
```

4、向表中插入多条记录并查询。

```
INSERT INTO reason_t2 VALUES (3, 'AAAAAAAACAAAAAAA','reason3'),(4, 'AAAA
AAAADAAAAAAA', 'reason4'),(5, 'AAAAAAAAEAAAAAAA','reason5');
SELECT * FROM reason_t2;
```

查询结果为：

```
r_reason_sk|  r_reason_id   |                          r_reason_
desc
------------+-----------------+-------------------------------------------------
--
-----------------------------------------------
       1 | AAAAAAAABAAAAAAA | reason1
       2 | AAAAAAAABAAAAAAA | reason2
       3 | AAAAAAAACAAAAAAA | reason3
```

```
       4 | AAAAAAAADAAAAAAA | reason4
       5 | AAAAAAAAEAAAAAAA | reason5
(5 rows)
```

5、向表中插入 reason 中 r_reason_sk 小于 5 的记录并查询。

```
INSERT INTO reason_t2 SELECT * FROM reason_t2 WHERE r_reason_sk <5;
SELECT * FROM reason_t2;
```

查询结果为：

```
r_reason_sk | r_reason_id   |                          r_reason_
desc
------------+----------------+----------------------------------------------------
--
----------------------------------------------
       1 | AAAAAAAABAAAAAAA | reason1
       2 | AAAAAAAABAAAAAAA | reason2
       3 | AAAAAAAACAAAAAAA | reason3
       4 | AAAAAAAADAAAAAAA | reason4
       5 | AAAAAAAAEAAAAAAA | reason5
       1 | AAAAAAAABAAAAAAA | reason1
       2 | AAAAAAAABAAAAAAA | reason2
       3 | AAAAAAAACAAAAAAA | reason3
       4 | AAAAAAAADAAAAAAA | reason4
(9 rows)
```

6、对表创建索引。

```
CREATE INDEX reason_t2_u_index ON reason_t2(r_reason_sk);
```

7、向表中插入多条记录，如果冲突则更新冲突数据行中 r_reason_id 字段为 'BBBBBBBBCAAAAAAA'，并查询。

```
INSERT INTO reason_t2 VALUES (5, 'BBBBBBBBCAAAAAAA','reason5'),(6, 'AAAA
AAAADAAAAAAA', 'reason6') ON DUPLICATE KEY UPDATE r_reason_id = 'BBBBB
BBBCAAAAAAA';
SELECT * FROM reason_t2;
```

查询结果为：

```
r_reason_sk | r_reason_id   |                          r_reason_
desc
------------+----------------+----------------------------------------------------
```

```
--
-----------------------------------------------
      1 | AAAAAAAABAAAAAAA | reason1
      2 | AAAAAAAABAAAAAAA | reason2
      3 | AAAAAAAACAAAAAAA | reason3
      4 | AAAAAAAADAAAAAAA | reason4
      5 | AAAAAAAAEAAAAAAA | reason5
      1 | AAAAAAAABAAAAAAA | reason1
      2 | AAAAAAAABAAAAAAA | reason2
      3 | AAAAAAAACAAAAAAA | reason3
      4 | AAAAAAAADAAAAAAA | reason4
      5 | BBBBBBBBCAAAAAAA | reason5
      6 | AAAAAAAADAAAAAAA | reason6
(11 rows)
```

7、删除表 reason_t2。

```
DROP TABLE reason_t2;
```

4.5.161.LOCK

功能描述

LOCK TABLE 获取表级锁。

PanWeiDB 在为一个引用了表的命令自动请求锁时，尽可能选择最小限制的锁模式。如果用户需要一种更为严格的锁模式，可以使用 LOCK 命令。例如，一个应用是在 Read Committed 隔离级别上运行事务，并且它需要保证表中的数据在事务的运行过程中不被修改。为实现这个目的，则可以在查询之前对表使用 SHARE 锁模式进行锁定。这样将防止数据不被并发修改，从而保证后续的查询可以读到已提交的持久化的数据。因为 SHARE 锁模式与任何写操作需要的 ROW EXCLUSIVE 模式冲突，并且 LOCK TABLE name IN SHARE MODE 语句将等到所有当前持有 ROW EXCLUSIVE 模式锁的事务提交或回滚后才能执行。因此，一旦获得该锁，就不会存在未提交的写操作，并且其他操作也只能等到该锁释放之后才能开始。

注意事项

- LOCK TABLE 只能在一个事务块的内部生效标识当前统计信息模式，因为锁在事务结束时就会被释放。出现在任意事务块外面的 LOCK TABLE 都会报错。
- 如果没有声明锁模式，缺省为最严格的模式 ACCESS EXCLUSIVE。
- LOCK TABLE ... IN ACCESS SHARE MODE 需要在目标表上有 SELECT 权限。所有其他形式的 LOCK 需要 UPDATE 和/或 DELETE 权限。
- 没有 UNLOCK TABLE 命令，锁总是在事务结束时释放。
- LOCK TABLE 只处理表级的锁，因此那些带“ROW”字样的锁模式都是有歧义的。这些模式名称通常可理解为用户试图在一个被锁定的表中获取行级的锁。同样，ROW EXCLUSIVE 模式也是一个可共享的表级锁。注意，只要是涉及到 LOCK TABLE ，所有锁模式都有相同的语意，区别仅在于规则中锁与锁之间是否冲突，规则请参见表 1：冲突的锁模式。
- 如果没有打开 xc_maintenance_mode 参数，那么对系统表申请 ACCESS EXCLUSIVE 级别锁将报错。

语法格式

```
LOCK [ TABLE ] {[ ONLY ] name [, ...]| {name [ * ]} [, ...]}
  [ IN {ACCESS SHARE | ROW SHARE | ROW EXCLUSIVE | SHARE UPDATE EXCLUSIVE | SHARE | SHARE ROW EXCLUSIVE | EXCLUSIVE | ACCESS EXCLUSIVE} MODE ]
  [ NOWAIT ];
```

参数说明

表 1 冲突的锁模式

请求的锁模式/当前锁模式	ACCESS SHARE	ROW SHARE	ROW EXCLUSIVE	SHARE UPDATE EXCLUSIVE	SHARE	SHARE ROW EXCLUSIVE	EXCLUSIVE	ACCESS EXCLUSIVE
ACCESS SHARE	-	-	-	-	-	-	-	X
ROW SHARE	-	-	-	-	-	-	X	X

请求的锁模式/当前锁模式	ACCESS SHARE	ROW SHARE	ROW EXCLUSIVE	SHARE UPDATE EXCLUSIVE	SHARE	SHARE ROW EXCLUSIVE	EXCLUSIVE	ACCESS EXCLUSIVE
ROW EXCLUSIVE	-	-	-	-	X	X	X	X
SHARE UPDATE EXCLUSIVE	-	-	-	X	X	X	X	X
SHARE	-	-	X	X	-	X	X	X
SHARE ROW EXCLUSIVE	-	-	X	X	X	X	X	X
EXCLUSIVE	-	X	X	X	X	X	X	X
ACCESS EXCLUSIVE	X	X	X	X	X	X	X	X

LOCK 的参数说明如下所示：

- **name**

  要锁定的表的名称，可以有模式修饰。

  LOCK TABLE 命令中声明的表的顺序就是上锁的顺序。

  取值范围：已存在的表名。

- **ONLY**

如果指定 ONLY，只有该表被锁定。如果没有声明，该表和他的所有子表将都被锁定。

- **ACCESS SHARE**

  ACCESS 锁只允许对表进行读取，而禁止对表进行修改。所有对表进行读取而不修改的 SQL 语句都会自动请求这种锁。例如，SELECT 命令会自动在被引用的表上请求一个这种锁。

- **ROW SHARE**

  与 EXCLUSIVE 和 ACCESS EXCLUSIVE 锁模式冲突。

  表递归连接条件

- **ROW EXCLUSIVE**

  与 ROW SHARE 锁相同，ROW EXCLUSIVE 允许并发读取表，但是禁止修改表中数据。UPDATE，DELETE，INSERT 命令会自动在目标表上请求这个锁（且所有被引用的其他表上还会自动加上的 ACCESS SHARE 锁）。通常情况下，所有会修改表数据的命令都会请求表的 ROW EXCLUSIVE 锁。

- **SHARE UPDATE EXCLUSIVE**

  这个模式保护一个表的模式不被并发修改，以及禁止在目标表上执行垃圾回收命令（VACUUM ）。

  VACUUM（不带 FULL 选项），ANALYZE，CREATE INDEX CONCURRENTLY 命令会自动请求这样的锁。

- **SHARE**

  SHARE 锁允许并发的查询，但是禁止对表进行修改。

  CREATE INDEX（不带 CONCURRENTLY 选项）语句会自动请求这种锁。

- **SHARE ROW EXCLUSIVE**

  SHARE ROW EXCLUSIVE 锁禁止对表进行任何的并发修改，而且是独占锁，因此一个会话中只能获取一次。

  任何 SQL 语句都不会自动请求这个锁模式。

- **EXCLUSIVE**

  EXCLUSIVE 锁允许对目标表进行并发查询，但是禁止任何其他操作。

  这个模式只允许并发加 ACCESS SHARE 锁，也就是说，只有对表的读动作可以和持有这个锁模式的事务并发执行。

任何 SQL 语句都不会在用户表上自动请求这个锁模式。然而在某些操作的时候，会在某些系统表上请求它。

- **ACCESS EXCLUSIVE**

  这个模式保证其所有者（事务）是可以访问该表的唯一事务。

  ALTER TABLE，DROP TABLE，TRUNCATE，REINDEX 命令会自动请求这种锁。

  在 LOCK TABLE 命令没有明确声明需要的锁模式时，它是缺省锁模式。

- **NOWAIT**

  声明 LOCK TABLE 不去等待任何冲突的锁释放，如果无法立即获取该锁，该命令退出并且发出一个错误信息。

  在不指定 NOWAIT 的情况下获取表级锁时，如果有其他互斥锁存在的话，则等待其他锁的释放。

示例

```
--在执行删除操作时对一个有主键的表进行 SHARE ROW EXCLUSIVE 锁。
panweidb=# CREATE TABLE reason_t1 AS TABLE reason;

panweidb=# START TRANSACTION;

panweidb=# LOCK TABLE reason_t1 IN SHARE ROW EXCLUSIVE MODE;

panweidb=# DELETE FROM reason_t1 WHERE r_reason_desc IN(SELECT r_reason_de
sc FROM reason_t1 WHERE r_reason_sk < 6 );

panweidb=# DELETE FROM reason_t1 WHERE r_reason_sk = 7;

panweidb=# COMMIT;

--删除表 reason_t1。
panweidb=# DROP TABLE reason_t1;
```

4.5.162.MOVE

功能描述

MOVE 在不检索数据的情况下重新定位一个游标。MOVE 的作用类似于 FETCH 命令，但只是重定位游标而不返回行。

注意事项

无。

语法格式

```
MOVE [ direction [ FROM | IN ] ] cursor_name;
```

其中 direction 子句为可选参数。

```
NEXT
 | PRIOR
 | FIRST
 | LAST
 | ABSOLUTE count
 | RELATIVE count
 | count
 | ALL
 | FORWARD
 | FORWARD count
 | FORWARD ALL
 | BACKWARD
 | BACKWARD count
 | BACKWARD ALL
```

参数说明

MOVE 命令的参数与 FETCH 的相同，详细请参见：SQL 语法参考->SQL 语法->FETCH 章节中的参数说明。

说明：

成功完成时，MOVE 命令将返回一个 “MOVE count” 的标签，count 是一个使用相同参数的 FETCH 命令会返回的行数（可能为零）。

示例

```
--开始一个事务。
```

```
panweidb=# START TRANSACTION;

--定义一个名为 cursor1 的游标。
panweidb=# CURSOR cursor1 FOR SELECT * FROM tpcds.reason;

--忽略游标 cursor1 的前 3 行。
panweidb=# MOVE FORWARD 3 FROM cursor1;

--抓取游标 cursor1 的前 4 行。
panweidb=# FETCH 4 FROM cursor1;
 r_reason_sk |  r_reason_id   |                      r_reason_desc

-------------+------------------+-----------------------------------------------------
-------------------------------------------
        4 | AAAAAAAAEAAAAAAA | Not the product that was ordred

        5 | AAAAAAAAFAAAAAAA | Parts missing

        6 | AAAAAAAAGAAAAAAA | Does not work with a product that I have

        7 | AAAAAAAAHAAAAAAA | Gift exchange

(4 rows)

--关闭游标。
panweidb=# CLOSE cursor1;

--结束一个事务。
panweidb=# END;
```

相关链接

参考：SQL 语法参考->SQL 语法->CLOSE、FETCH 章节。

4.5.163.PREDICT BY

功能描述

利用完成训练的模型进行推测任务。

注意事项

调用的模型名称在系统表 gs_model_warehouse 中可查看到。

语法格式

```
PREDICT BY model_name [ (FEATURES attribute [, attribute] +]) ]
```

参数说明

- **model_name**

  用于推测任务的模型名称。

  取值范围：字符串，需要符合标识符的命名规则。

- **attribute**

  推测任务的输入特征列名。

  取值范围：字符串，需要符合标识符的命名规则。

示例

```
SELECT id, PREDICT BY price_model (FEATURES size,lot), price
FROM houses;
```

相关链接

参考：SQL 语法参考->SQL 语法->CREATE MODEL、DROP MODEL 章节。

4.5.164.PREPARE

功能描述

创建一个预备语句。

预备语句是服务端的对象，可以用于优化性能。在执行 PREPARE 语句的时候，指定的查询被解析、分析、重写。当随后发出 EXECUTE 语句的时候，预备语句被规划和执行。这种设计避免了重复解析、分析工作。PREPARE 语句创建

后在整个数据库会话期间一直存在，一旦创建成功，即便是在事务块中创建，事务回滚，PREPARE 也不会删除。只能通过显式调用 DEALLOCATE 进行删除，会话结束时，PREPARE 也会自动删除。

注意事项

无。

语法格式

```
PREPARE name [ ( data_type [, ...] ) ] AS statement;
```

参数说明

- **name**

  指定预备语句的名称。它必须在该会话中是唯一的。

- **data_type**

  参数的数据类型。

- **statement**

  是 SELECT INSERT、UPDATE、DELETE、MERGE INTO 或 VALUES 语句之一。

示例

请参见：SQL 语法参考->SQL 语法->EXECUTE 章节中的示例。

相关链接

参考：SQL 语法参考->SQL 语法->DEALLOCATE 章节。

4.5.165.PREPARE TRANSACTION

功能描述

为当前事务做两阶段提交的准备。

在命令之后，事务就不再和当前会话关联了；它的状态完全保存在磁盘上，它被提交成功的可能性非常高，即使是在请求提交之前数据库发生了崩溃也如此。

一旦准备好了，一个事务就可以在稍后用 COMMIT PREPARED 或 ROLLBACK PREPARED 命令分别进行提交或者回滚。这些命令可以从任何会话中发出，而不光是最初执行事务的那个会话。

从发出命令的会话的角度来看，PREPARE TRANSACTION 不同于 ROLLBACK：在执行它之后，就不再有活跃的当前事务了，并且预备事务的效果无法见到（在事务提交的时候其效果会再次可见）。

如果 PREPARE TRANSACTION 因为某些原因失败，那么它就会变成一个 ROLLBACK，当前事务被取消。

注意事项

- 事务功能由数据库自动维护，不应显式使用事务功能。
- 在运行 PREPARE TRANSACTION 命令时，必须在 postgresql.conf 配置文件中增大 max_prepared_transactions 的数值。建议至少将其设置为等于 max_connections，这样每个会话都可以有一个等待中的预备事务。

语法格式

```
PREPARE TRANSACTION transaction_id;
```

参数说明

transaction_id

待提交事务的标识符，用于后面在 COMMIT PREPARED 或 ROLLBACK PREPARED 的时候标识这个事务。它不能和任何当前预备事务已经使用了的标识符同名。

取值范围：标识符必须以字符串文本的方式书写，并且必须小于 200 字节长。

相关链接

参考：SQL 语法参考->SQL 语法->COMMIT PREPARED、ROLLBACK PREPARED 章节。

4.5.166.PURGE

功能描述

使用 PURGE 语句可以实现如下功能：

- 从回收站中清理表或索引，并释放对象相关的全部空间。
- 清理回收站。
- 清理回收站中指定表空间的对象。

注意事项

- 清除（PURGE）操作支持：表（PURGE TABLE）、索引（PURGE INDEX）、回收站（PURGE RECYCLEBIN）。
- 执行 PURGE 操作的权限要求如下：
  - PURGE TABLE：用户必须是表的所有者，且用户必须拥有表所在模式的 USAGE 权限，系统管理员默认拥有此权限。
  - PURGE INDEX：用户必须是索引的所有者，用户必须拥有索引所在模式的 USAGE 权限，系统管理员默认拥有此权限。
  - PURGE RECYCLEBIN：普通用户只能清理回收站中当前用户拥有的对象，且用户必须拥有对象所在模式的 USAGE 权限，系统管理员默认可以清理回收站所有对象。

语法格式

```
PURGE { TABLE [schema_name.]table_name
    | INDEX index_name
    | RECYCLEBIN
   }
```

参数说明

- [schema_name.]

  模式名。
- TABLE [schema_name.] table_name

  清空回收站中指定的表。
- INDEX index_name

  清空回收站中指定的索引。
- RECYCLEBIN

  清空回收站中的对象。

示例

```
-- 创建表空间 reason_table_space
panweidb=# CREATE TABLESPACE REASON_TABLE_SPACE1 owner tpcds RELATIVE lo
cation 'tablespace/tsp_reason1';
-- 在表空间创建表 tpcds.reason_t1
panweidb=# CREATE TABLE tpcds.reason_t1
(
 r_reason_sk    integer,
 r_reason_id    character(16),
 r_reason_desc  character(100)
 ) tablespace reason_table_space1;
-- 在表空间创建表 tpcds.reason_t2
panweidb=# CREATE TABLE tpcds.reason_t2
(
 r_reason_sk    integer,
 r_reason_id    character(16),
 r_reason_desc  character(100)
 ) tablespace reason_table_space1;
-- 在表空间创建表 tpcds.reason_t3
panweidb=# CREATE TABLE tpcds.reason_t3
(
 r_reason_sk    integer,
 r_reason_id    character(16),
 r_reason_desc  character(100)
 ) tablespace reason_table_space1;
-- 对表 tpcds.reason_t1 创建索引
panweidb=# CREATE INDEX index_t1 on tpcds.reason_t1(r_reason_id);
panweidb=# DROP TABLE tpcds.reason_t1;
panweidb=# DROP TABLE tpcds.reason_t2;
panweidb=# DROP TABLE tpcds.reason_t3;
--查看回收站
panweidb=# SELECT rcyname,rcyoriginname,rcytablespace FROM GS_RECYCLEBIN;
     rcyname       | rcyoriginname | rcytablespace
-------------------+---------------+---------------
```

```
 BIN$16409$2CEE988==$0 | reason_t1    |       16408
 BIN$16412$2CF2188==$0 | reason_t2    |       16408
 BIN$16415$2CF2EC8==$0 | reason_t3    |       16408
 BIN$16418$2CF3EC8==$0 | index_t1     |         0
(4 rows)
--PURGE 清除表
panweidb=# PURGE TABLE tpcds.reason_t3;
panweidb=# SELECT rcyname,rcyoriginname,rcytablespace FROM GS_RECYCLEBIN;
      rcyname       | rcyoriginname | rcytablespace
----------------------+---------------+---------------
 BIN$16409$2CEE988==$0 | reason_t1    |       16408
 BIN$16412$2CF2188==$0 | reason_t2    |       16408
 BIN$16418$2CF3EC8==$0 | index_t1     |         0
(3 rows)
--PURGE 清除索引
panweidb=# PURGE INDEX tcpds.index_t1;
panweidb=# SELECT rcyname,rcyoriginname,rcytablespace FROM GS_RECYCLEBIN;
      rcyname       | rcyoriginname | rcytablespace
----------------------+---------------+---------------
 BIN$16409$2CEE988==$0 | reason_t1    |       16408
 BIN$16412$2CF2188==$0 | reason_t2    |       16408
(2 rows)
--PURGE 清除回收站所有对象
panweidb=# PURGE recyclebin;
panweidb=# SELECT rcyname,rcyoriginname,rcytablespace FROM GS_RECYCLEBIN;
      rcyname       | rcyoriginname | rcytablespace
----------------------+---------------+---------------
(0 rows)
```

4.5.167.REASSIGN OWNED

功能描述

修改数据库对象的属主。

REASSIGN OWNED 要求系统将当前数据库中 old_role 拥有的所有数据库对象的属主更改为 new_role。

注意事项

- REASSIGN OWNED 常用于在删除角色之前的准备工作。
- 执行 REASSIGN OWNED 需要有原角色和目标角色上的权限。
- REASSIGN OWNED 不会更改数据本身的所有权。
- REASSIGN OWNED 不会更改 TRIGGER 对象的所有权。

语法格式

```
REASSIGN OWNED BY old_role [, ...] TO new_role;
```

参数说明

- **old_role**

  旧属主的角色名。

- **new_role**

  将要成为这些对象属主的新角色的名称。

示例

无。

4.5.168.REFRESH INCREMENTAL MATERIALIZED VIEW

功能描述

REFRESH INCREMENTAL MATERIALIZED VIEW 会以增量刷新的方式对物化视图进行刷新。

注意事项

- 增量刷新仅支持增量物化视图。
- 刷新物化视图需要当前用户拥有基表的 SELECT 权限。

语法格式

```
REFRESH INCREMENTAL MATERIALIZED VIEW mv_name;
```

参数说明

- **mv_name**

  要刷新的物化视图的名称。

示例

```
--创建一个普通表
panweidb=# CREATE TABLE my_table (c1 int, c2 int);
--创建增量物化视图
panweidb=# CREATE INCREMENTAL MATERIALIZED VIEW my_imv AS SELECT * FROM
my_table;
--基表写入数据
panweidb=# INSERT INTO my_table VALUES(1,1),(2,2);
--对增量物化视图 my_imv 进行增量刷新
panweidb=# REFRESH INCREMENTAL MATERIALIZED VIEW my_imv;
```

相关链接

参考：SQL 语法参考->SQL 语法->ALTER MATERIALIZED VIEW、CREATE INCREMENTAL MATERIALIZED VIEW、CREATE MATERIALIZED VIEW、CREATE TABLE、DROP MATERIALIZED VIEW、REFRESH MATERIALIZED VIEW 章节。

4.5.169.MATERIALIZED VIEW

功能描述

REFRESH MATERIALIZED VIEW 会以全量刷新的方式对物化视图进行刷新。

注意事项

- 全量刷新既可以对全量物化视图执行，也可以对增量物化视图执行。
- 刷新物化视图需要当前用户拥有基表的 SELECT 权限。

语法格式

```
REFRESH MATERIALIZED VIEW mv_name;
```

参数说明

- **mv_name**

  要刷新的物化视图的名称。

示例

```
--创建一个普通表
panweidb=# CREATE TABLE my_table (c1 int, c2 int);
--创建全量物化视图
panweidb=# CREATE MATERIALIZED VIEW my_mv AS SELECT * FROM my_table;
--创建增量物化视图
panweidb=# CREATE INCREMENTAL MATERIALIZED VIEW my_imv AS SELECT * FROM
my_table;
--基表写入数据
panweidb=# INSERT INTO my_table VALUES(1,1),(2,2);
--对全量物化视图 my_mv 进行全量刷新
panweidb=# REFRESH MATERIALIZED VIEW my_mv;
--对增量物化视图 my_imv 进行全量刷新
panweidb=# REFRESH MATERIALIZED VIEW my_imv;
```

相关链接

参考：SQL 语法参考->SQL 语法->ALTER MATERIALIZED VIEW、CREATE INCREMENTAL MATERIALIZED VIEW、REATE MATERIALIZED VIEW、CREATE TABLE、DROP MATERIALIZED VIEW、REFRESH INCREMENTAL MATERIALIZED VIEW 章节。

4.5.170.REINDEX

功能描述

为表中的数据重建索引。

在以下几种情况下需要使用 REINDEX 重建索引：

- 索引崩溃，并且不再包含有效的数据。
- 索引变得“臃肿”，包含大量的空页或接近空页。
- 为索引更改了存储参数（例如填充因子），并且希望这个更改完全生效。
- 使用 CONCURRENTLY 选项创建索引失败，留下了一个“非法”索引。

注意事项

- REINDEX DATABASE 和 SYSTEM 这种形式的重建索引不能在事务块中执行。
- REINDEX CONCURRENTLY 这种形式的重建索引不能在事务块中执行。
- 不建议在事务中 REINDEX DATABASE。
- 不建议在事务中 REINDEX 系统表。

语法格式

- 重建普通索引。

```
REINDEX { INDEX | [INTERNAL] TABLE | DATABASE | SYSTEM } [CONCURRENTLY] name [ FORCE ];
```

- 重建索引分区。

```
REINDEX { INDEX| [INTERNAL] TABLE} [CONCURRENTLY] name
  PARTITION partition_name [ FORCE ];
```

参数说明

- **INDEX**

  重新建立指定的索引。

- **INTERNAL TABLE**

  重建列存表的 Desc 表的索引，如果表有从属的 “TOAST” 表，则这个表也会重建索引。

  不支持 CONCURRENTLY 方式重建索引，REINDEX INTERNAL TABLE CONCURRENTLY 相当于执行 REINDEX INTERNAL TABLE。

- **TABLE**

  重新建立指定表的所有索引，如果表有从属的"TOAST"表，则这个表也会重建索引。如果表上有索引已经被 alter unusable 失效，则这个索引无法被重新创建。当指定 CONCURRENTLY 选项时，暂不支持重建从属"TOAST"表上的索引。

- **DATABASE**

  重建当前数据库里的所有索引。当指定 CONCURRENTLY 选项时，暂不支持重建数据库中表的从属"TOAST"表上的索引。

❖ **SYSTEM**

在系统库上重建所有系统表上的索引。不会处理在用户表上的索引。

❖ **CONCURRENTLY**

以不阻塞 DML 的方式重建索引（加 ShareUpdateExclusiveLock 锁）。重建索引时，一般会阻塞其他语句对该索引所依赖表的访问。指定此关键字，可以实现重建过程中不阻塞 DML。不支持在线重建系统表上的索引。不支持 REINDEX INTERNAL TABLE CONCURRENTLY 和 REINDEX SYSTEM CONCURRENTLY。当执行 REINDEX DATABASE CONCURRENTLY 时，在线重建当前数据库中用户表上的所有索引（不会处理系统表上的索引）。REINDEX CONCURRENTLY 不可以在事务内执行。在线重建索引只支持 B-tree 索引和 UB-tree 索引，只支持普通索引、GLOBAL 索引、LOCAL 索引。在线并行重建索引只支持 Astore 的普通索引、GLOBAL 索引、LOCAL 索引。如果在线重建索引失败，可能会留下非法的新索引，在系统无法自动清理失败新索引的情况下（比如数据库宕机），需要尽快手动清除（使用 DROP INDEX 语句）非法新索引，以防占用更多资源。一般来说，非法的新索引的后缀名为_ccnew。REINDEX INDEX CONCURRENTLY 对表加 4 级会话锁，且其前几个阶段与 CREATE INDEX CONCURRENTLY 相似，因此也可能产生卡住或死锁的问题，具体场景与 CREATE INDEX CONCURRENTLY 相似（比如两个会话同时对同一个索引或表进行 REINDEX CONCURRENTLY 操作，会引发死锁问题），详见 CREATE INDEX 章节。

- 普通 REINDEX 命令可以在事务内执行，但是 REINDEX CONCURRENTLY 不可以在事务内执行。
- 列存表、ustore 表、临时表、全局分区索引不支持 CONCURRENTLY 方式重建索引。
- REINDEX SYSTEM CONCURRENTLY 不会执行任何操作，因为系统表不支持在线重建索引。
- REINDEX DATABASE CONCURRENTLY 执行时会跳过系统表、列存表、ustore 表、临时表。
- 在线重建普通表会跳过“非法”（invalid）的普通索引和“不可用”（unusable）的普通索引。

- 在线重建普通索引会将“非法”索引变为“合法”（valid），将“不可用”索引变为“可用”（usable），除了 toast 表的“非法”索引。
- 在线重建分区表会跳过“不可用”分区表索引和含有“不可用”分区索引的分区表索引。
- 在线重建分区表特定分区会跳过“不可用”分区索引和在“不可用”分区表索引内的分区索引。
- 在线重建分区表索引会将“不可用”分区表索引变为“可用”，将分区表索引内的“不可用”分区索引变为“可用”。
- 在线重建分区表索引特定分区索引会将“不可用”分区索引变为“可用”。

【说明】

- 重建索引时指定此关键字，需要执行先后两次对该表的全表扫描来完成 build，第一次扫描的时候创建新索引，不阻塞读写操作；第二次扫描的时候合并更新第一次扫描到目前为止发生的变更。
- 因为需要执行两次对表的扫描和 build，且必须等待现有的所有可能对该表执行修改的事务结束，所以该索引的重建比正常耗时更长，同时带来的 CPU 和 I/O 消耗对其他业务也会造成影响。
- 如果在索引构建时发生失败，那会留下一个“不可用”的索引。这个索引会被查询忽略，但它仍消耗更新开销。这种情况推荐的恢复方法是删除该索引并尝试再次 CONCURRENTLY 重建索引。
- 由于在第二次扫描之后，索引构建必须等待任何持有早于第二次扫描拿的快照的事务终止，而且建索引时加的 ShareUpdateExclusiveLock 锁（4 级）会和大于等于 4 级的锁冲突，因此在创建这类索引时，容易引发卡住（hang）或者死锁问题。例如：
  - 两个会话对同一个表重建 CONCURRENTLY 索引，会引起死锁问题；
  - 两个会话，一个对表重建 CONCURRENTLY 索引，一个 drop table，会引起死锁问题；

- 三个会话，会话 1 先对表 a 加锁，不提交，会话 2 接着对表 b 重建 CONCURRENTLY 索引，会话 3 接着对表 a 执行写入操作，在会话 1 事务未提交之前，会话 2 会一直被阻塞；
- 将事务隔离级别设置成可重复读（默认为读已提交），起两个会话，会话 1 起事务对表 a 执行写入操作，不提交，会话 2 对表 b 重建 CONCURRENTLY 索引，在会话 1 事务未提交之前，会话 2 会一直被阻塞。

❖ **name**

需要重建索引的索引、表、数据库的名称。表和索引可以有模式修饰。

REINDEX DATABASE 和 SYSTEM 只能重建当前数据库的索引，所以 name 必须和当前数据库名称相同。

❖ **FORCE**

无效选项，会被忽略。

❖ **partition_name**

需要重建索引的分区的名称或者索引分区的名称。

取值范围：

- 如果前面是 REINDEX INDEX，则这里应该指定索引分区的名称；
- 如果前面是 REINDEX TABLE，则这里应该指定分区的名称；
- 如果前面是 REINDEX INTERNAL TABLE，则这里应该指定列存分区表的分区的名称。

REINDEX DATABASE 和 SYSTEM 这种形式的重建索引不能在事务块中执行。

示例

```
--创建一个行存表 customer_t1，并在 customer_t1 表上的 c_customer_sk 字段创建索引。
panweidb=# CREATE TABLE customer_t1
(
  c_customer_sk         integer           not null,
  c_customer_id         char(16)          not null,
  c_current_cdemo_sk    integer                   ,
```

```
  c_current_hdemo_sk      integer               ,
  c_current_addr_sk       integer               ,
  c_first_shipto_date_sk  integer               ,
  c_first_sales_date_sk   integer               ,
  c_salutation            char(10)              ,
  c_first_name            char(20)              ,
  c_last_name             char(30)              ,
  c_preferred_cust_flag   char(1)               ,
  c_birth_day             integer               ,
  c_birth_month           integer               ,
  c_birth_year            integer               ,
  c_birth_country         varchar(20)           ,
  c_login                 char(13)              ,
  c_email_address         char(50)              ,
  c_last_review_date      char(10)
)
WITH (orientation = row);

panweidb=# CREATE INDEX customer_index1 ON customer_t1 (c_customer_sk);

panweidb=# INSERT INTO customer_t1 SELECT * FROM customer WHERE c_customer
_sk < 10;

--重建一个单独索引。
panweidb=# REINDEX INDEX customer_index1;

--实时重建一个单独索引。
panweidb=# REINDEX INDEX CONCURRENTLY customer_index1;

--重建表 customer_t1 上的所有索引。
panweidb=# REINDEX TABLE customer_t1;

--实时重建表 customer_t1 上的所有索引。
panweidb=# REINDEX TABLE CONCURRENTLY customer_t1;
```

```
--删除 tpcds.customer_t1 表。
panweidb=# DROP TABLE customer_t1;
```

优化建议

- INTERNAL TABLE

  此种情况大多用于故障恢复，不建议进行并发操作。

- DATABASE

  不能在事务中 reindex database。

- SYSTEM

  不能在事务中 reindex 系统表。

4.5.171.RELEASE SAVEPOINT

功能描述

RELEASE SAVEPOINT 删除一个当前事务先前定义的保存点。

把一个保存点删除就令其无法作为回滚点使用，除此之外它没有其他用户可见的行为。它并不能撤销在保存点建立起来之后执行的命令的影响。要撤销那些命令可以使用 ROLLBACK TO SAVEPOINT 。在不再需要的时候删除一个保存点可以令系统在事务结束之前提前回收一些资源。

RELEASE SAVEPOINT 也删除所有在指定的保存点建立之后的所有保存点。

注意事项

- 不能 RELEASE 一个没有定义的保存点，语法上会报错。
- 如果事务在回滚状态，则不能释放保存点。
- 如果多个保存点拥有同样的名称，只有最近定义的那个才被释放。

语法格式

```
RELEASE [ SAVEPOINT ] savepoint_name;
```

参数说明

savepoint_name

要删除的保存点的名称。

示例

```
--创建一个新表。
panweidb=# CREATE TABLE tpcds.table1(a int);

--开启事务。
panweidb=# START TRANSACTION;

--插入数据。
panweidb=# INSERT INTO tpcds.table1 VALUES (3);

--建立保存点。
panweidb=# SAVEPOINT my_savepoint;

--插入数据。
panweidb=# INSERT INTO tpcds.table1 VALUES (4);

--删除保存点。
panweidb=# RELEASE SAVEPOINT my_savepoint;

--提交事务。
panweidb=# COMMIT;

--查询表的内容，会同时看到 3 和 4。
panweidb=# SELECT * FROM tpcds.table1;

--删除表。
panweidb=# DROP TABLE tpcds.table1;
```

相关链接

参考：SQL 语法参考->SQL 语法->SAVEPOINT、ROLLBACK TO SAVEPOINT 章节。

4.5.172.RESET

功能描述

RESET 将指定的运行时参数恢复为缺省值。这些参数的缺省值是指 postgresql.conf 配置文件中所描述的参数缺省值。

RESET 命令与如下命令的作用相同：

SET configuration_parameter TO DEFAULT

注意事项

RESET 的事务性行为和 SET 相同：它的影响将会被事务回滚撤销。

语法格式

```
RESET {configuration_parameter | CURRENT_SCHEMA | TIME ZONE | TRANSACTION ISOLATION LEVEL | SESSION AUTHORIZATION | ALL };
```

参数说明

- **configuration_parameter**

  运行时参数的名称。

  取值范围：可以使用 SHOW ALL 命令查看运行时参数。

- **CURRENT_SCHEMA**

  当前模式。

- **TIME ZONE**

  时区。

- **TRANSACTION ISOLATION LEVEL**

  事务的隔离级别。

- **SESSION AUTHORIZATION**

  当前会话的用户标识符。

- **ALL**

  所有运行时参数。

示例

```
--把 timezone 设为缺省值。
panweidb=# RESET timezone;

--把所有参数设置为缺省值。
```

```
panweidb=# RESET ALL;
```

相关链接

参考：SQL 语法参考->SQL 语法->SET、SHOW 章节。

4.5.173.REVOKE

功能描述

REVOKE 用于撤销一个或多个角色的权限。

注意事项

非对象所有者试图在对象上 REVOKE 权限，命令按照以下规则执行：

- 如果授权用户没有该对象上的权限，则命令立即失败。
- 如果授权用户有部分权限，则只撤销那些有授权选项的权限。
- 如果授权用户没有授权选项，REVOKE ALL PRIVILEGES 形式将发出一个错误信息，而对于其他形式的命令而言，如果是命令中指定名称的权限没有相应的授权选项，该命令将发出一个警告。
- 不允许对表分区进行 REVOKE 操作，对分区表进行 REVOKE 操作会引起告警。

语法格式

- 回收指定表或视图上权限。

```
REVOKE [ GRANT OPTION FOR ]
  { { SELECT | INSERT | UPDATE | DELETE | TRUNCATE | REFERENCES | ALTER | DROP |
    COMMENT | INDEX | VACUUM }[, ...]
  | ALL [ PRIVILEGES ] }
  ON { [ TABLE ] table_name [, ...]
    | ALL TABLES IN SCHEMA schema_name [, ...] }
  FROM { [ GROUP ] role_name | PUBLIC } [, ...]
  [ CASCADE | RESTRICT ];
```

- 回收表上指定字段权限。

```
REVOKE [ GRANT OPTION FOR ]
```

```
{ {{ SELECT | INSERT | UPDATE | REFERENCES | COMMENT } ( column_name [, ...] )}[,
    ...]
| ALL [ PRIVILEGES ] ( column_name [, ...] ) }
ON [ TABLE ] table_name [, ...]
FROM { [ GROUP ] role_name | PUBLIC } [, ...]
[ CASCADE | RESTRICT ];
```

- 回收指定序列上权限，LARGE 字段属性可选，回收语句不区分序列是否为 LARGE。

```
REVOKE [ GRANT OPTION FOR ]
  { { SELECT | UPDATE | ALTER | DROP | COMMENT }[, ...]
  | ALL [ PRIVILEGES ] }
   ON { [ [ LARGE ] SEQUENCE ] sequence_name [, ...]
     | ALL SEQUENCES IN SCHEMA schema_name [, ...] }
  FROM { [ GROUP ] role_name | PUBLIC } [, ...]
  [ CASCADE | RESTRICT ];
```

- 回收指定数据库上权限。

```
REVOKE [ GRANT OPTION FOR ]
  { { CREATE | CONNECT | TEMPORARY | TEMP | ALTER | DROP | COMMENT } [, ...]
  | ALL [ PRIVILEGES ] }
  ON DATABASE database_name [, ...]
  FROM { [ GROUP ] role_name | PUBLIC } [, ...]
  [ CASCADE | RESTRICT ];
```

- 回收指定域上权限。

```
REVOKE [ GRANT OPTION FOR ]
  { USAGE | ALL [ PRIVILEGES ] }
  ON DOMAIN domain_name [, ...]
  FROM { [ GROUP ] role_name | PUBLIC } [, ...]
  [ CASCADE | RESTRICT ];
```

- 回收指定客户端加密主密钥上的权限。

```
REVOKE [ GRANT OPTION FOR ]
  { { USAGE | DROP } [, ...] | ALL [PRIVILEGES] }
  ON CLIENT_MASTER_KEYS client_master_keys_name [, ...]
  FROM { [ GROUP ] role_name | PUBLIC } [, ...]
  [ CASCADE | RESTRICT ];
```

❖ 回收指定列加密密钥上的权限。

```
REVOKE [ GRANT OPTION FOR ]
  { { USAGE | DROP } [, ...] | ALL [PRIVILEGES]}
  ON COLUMN_ENCRYPTION_KEYS column_encryption_keys_name [, ...]
  FROM { [ GROUP ] role_name | PUBLIC } [, ...]
  [ CASCADE | RESTRICT ];
```

❖ 回收指定目录上权限。

```
REVOKE [ GRANT OPTION FOR ]
  { { READ | WRITE | ALTER |DROP } [, ...] | ALL [ PRIVILEGES ] }
  ON DIRECTORY directory_name [, ...]
  FROM { [ GROUP ] role_name | PUBLIC } [, ...]
  [ CASCADE | RESTRICT ];
```

❖ 回收指定外部数据源上权限。

```
REVOKE [ GRANT OPTION FOR ]
  { USAGE | ALL [ PRIVILEGES ] }
  ON FOREIGN DATA WRAPPER fdw_name [, ...]
  FROM { [ GROUP ] role_name | PUBLIC } [, ...]
  [ CASCADE | RESTRICT ];
```

❖ 回收指定外部服务器上权限。

```
REVOKE [ GRANT OPTION FOR ]
  { { USAGE | ALTER | DROP | COMMENT } [, ...] | ALL [ PRIVILEGES ] }
  ON FOREIGN SERVER server_name [, ...]
  FROM { [ GROUP ] role_name | PUBLIC } [, ...]
  [ CASCADE | RESTRICT ];
```

❖ 回收指定函数上权限。

```
REVOKE [ GRANT OPTION FOR ]
  { { EXECUTE | ALTER | DROP | COMMENT } [, ...] | ALL [ PRIVILEGES ] }
  ON { FUNCTION {function_name ( [ {[ argmode ] [ arg_name ] arg_type} [, ...] ] )}
     [, ...]
    | ALL FUNCTIONS IN SCHEMA schema_name [, ...] }
  FROM { [ GROUP ] role_name | PUBLIC } [, ...]
  [ CASCADE | RESTRICT ];
```

❖ 回收指定存储过程上权限。

```
REVOKE [ GRANT OPTION FOR ]
```

```
  { { EXECUTE | ALTER | DROP | COMMENT } [, ...] | ALL [ PRIVILEGES ] }
  ON { PROCEDURE {proc_name ( [ {[ argmode ] [ arg_name ] arg_type} [, ...] ] )} [,
      ...]
    | ALL PROCEDURE IN SCHEMA schema_name [, ...] }
  FROM { [ GROUP ] role_name | PUBLIC } [, ...]
  [ CASCADE | RESTRICT ];
```

❖ 回收指定过程语言上权限。

```
REVOKE [ GRANT OPTION FOR ]
  { USAGE | ALL [ PRIVILEGES ] }
   ON LANGUAGE lang_name [, ...]
   FROM { [ GROUP ] role_name | PUBLIC } [, ...]
   [ CASCADE | RESTRICT ];
```

❖ 回收指定大对象上权限。

```
REVOKE [ GRANT OPTION FOR ]
   { { SELECT | UPDATE } [, ...] | ALL [ PRIVILEGES ] }
   ON LARGE OBJECT loid [, ...]
   FROM { [ GROUP ] role_name | PUBLIC } [, ...]
   [ CASCADE | RESTRICT ];
```

❖ 回收指定模式上权限。

```
REVOKE [ GRANT OPTION FOR ]
   { { CREATE | USAGE | ALTER | DROP | COMMENT } [, ...] | ALL [ PRIVILEGES ] }
   ON SCHEMA schema_name [, ...]
   FROM { [ GROUP ] role_name | PUBLIC } [, ...]
   [ CASCADE | RESTRICT ];
```

❖ 回收指定表空间上权限。

```
REVOKE [ GRANT OPTION FOR ]
   { { CREATE | ALTER | DROP | COMMENT } [, ...] | ALL [ PRIVILEGES ] }
   ON TABLESPACE tablespace_name [, ...]
   FROM { [ GROUP ] role_name | PUBLIC } [, ...]
   [ CASCADE | RESTRICT ];
```

❖ 回收指定类型上权限。

```
REVOKE [ GRANT OPTION FOR ]
  { { USAGE | ALTER | DROP | COMMENT } [, ...] | ALL [ PRIVILEGES ] }
   ON TYPE type_name [, ...]
```

```
  FROM { [ GROUP ] role_name | PUBLIC } [, ...]
  [ CASCADE | RESTRICT ];
```

❖ 回收 Data Source 对象上的权限。

```
REVOKE [ GRANT OPTION FOR ]
  { USAGE | ALL [PRIVILEGES] }
   ON DATA SOURCE src_name [, ...]
   FROM {[GROUP] role_name | PUBLIC} [, ...]
  [ CASCADE | RESTRICT ];
```

❖ 回收 package 对象的权限。

```
REVOKE [ GRANT OPTION FOR ]
  { { EXECUTE | ALTER | DROP | COMMENT } [, ...] | ALL [PRIVILEGES] }
  ON PACKAGE package_name [, ...]
  FROM {[GROUP] role_name | PUBLIC} [, ...]
  [ CASCADE | RESTRICT ];
```

❖ 按角色回收角色上的权限。

```
REVOKE [ ADMIN OPTION FOR ]
   role_name [, ...] FROM role_name [, ...]
   [ CASCADE | RESTRICT ];
```

❖ 回收角色上的 sysadmin 权限。

```
REVOKE ALL { PRIVILEGES | PRIVILEGE } FROM role_name;
```

❖ 回收 ANY 权限。

```
REVOKE [ ADMIN OPTION FOR ]
 { CREATE ANY TABLE | ALTER ANY TABLE | DROP ANY TABLE | SELECT ANY TABLE | I
     NSERT ANY TABLE | UPDATE ANY TABLE |
 DELETE ANY TABLE | CREATE ANY SEQUENCE | CREATE ANY INDEX | CREATE ANY F
     UNCTION | EXECUTE ANY FUNCTION |
 CREATE ANY PACKAGE | EXECUTE ANY PACKAGE | CREATE ANY TYPE | ALTER ANY
     TYPE | DROP ANY TYPE | ALTER ANY SEQUENCE | DROP ANY SEQUENCE |
 SELECT ANY SEQUENCE | ALTER ANY INDEX | DROP ANY INDEX | CREATE ANY SYN
     ONYM | DROP ANY SYNONYM | CREATE ANY TRIGGER | ALTER ANY TRIGGER | D
     ROP ANY TRIGGER
} [, ...]
 FROM [ GROUP ] role_name [, ...];
```

参数说明

- 关键字 PUBLIC 表示一个隐式定义的拥有所有角色的组。
- 权限类别和参数说明，请参见：SQL 语法参考->SQL 语法->GRANT 章节中的参数说明。
- 任何特定角色拥有的特权包括直接授予该角色的特权、从该角色作为其成员的角色中得到的权限以及授予给 PUBLIC 的权限。因此，从 PUBLIC 收回 SELECT 特权并不一定会意味着所有角色都会失去在该对象上的 SELECT 特权，那些直接被授予的或者通过另一个角色被授予的角色仍然会拥有它。类似地，从一个用户收回 SELECT 后，如果 PUBLIC 仍有 SELECT 权限，该用户还是可以使用 SELECT。
- 指定 GRANT OPTION FOR 时，只撤销对该权限授权的权力，而不撤销该权限本身。
- 如用户 A 拥有某个表的 UPDATE 权限，及 WITH GRANT OPTION 选项，同时 A 把这个权限赋予了用户 B，则用户 B 持有的权限称为依赖性权限。当用户 A 持有的权限或者授权选项被撤销时，必须声明 CASCADE，将所有依赖性权限都撤销。
- 一个用户只能撤销由它自己直接赋予的权限。例如，如果用户 A 被指定授权（WITH ADMIN OPTION）选项，且把一个权限赋予了用户 B，然后用户 B 又赋予了用户 C，则用户 A 不能直接将 C 的权限撤销。但是，用户 A 可以撤销用户 B 的授权选项，并且使用 CASCADE。这样，用户 C 的权限就会自动被撤销。另外一个例子：如果 A 和 B 都赋予了 C 同样的权限，则 A 可以撤销他自己的授权选项，但是不能撤销 B 的，因此 C 仍然拥有该权限。
- 如果执行 REVOKE 的角色持有的权限是通过多层成员关系获得的，则具体是哪个包含的角色执行的该命令是不确定的。在这种场合下，最好的方法是使用 SET ROLE 成为特定角色，然后执行 REVOKE，否则可能导致删除了不想删除的权限，或者是任何权限都没有删除。

示例

参考：SQL 语法参考->SQL 语法->GRANT 章节中的示例。

相关链接

参考：SQL 语法参考->SQL 语法->GRANT 章节。

4.5.174.ROLLBACK

功能描述

回滚当前事务并取消当前事务中的所有更新。

在事务运行的过程中发生了某种故障，事务不能继续执行，系统将事务中对数据库的所有已完成的操作全部撤销，数据库状态回到事务开始前。

语法格式

```
ROLLBACK [ WORK | TRANSACTION ];
```

参数说明

WORK | TRANSACTION：可选关键字。除了增加可读性，没有任何其他作用。

注意事项

无。

示例

1、 创建测试表 test。

```
CREATE TABLE test(id int);
```

2、 开启一个事务 。

```
START TRANSACTION;
```

3、 插入数据。

```
INSERT INTO test(id) VALUES(1);
```

4、 查询当前数据。

```
select * from test;
```

当结果显示如下信息，则表示插入成功。

```
 id
----
  1
(1 row)
```

5、 取消所有更改 。

```
ROLLBACK;
```

6、 查询数据进行验证。

```
select * from test;
```

当结果显示如下信息，则表示回滚成功。

```
id
----
(0 rows)
```

4.5.175.ROLLBACK PREPARED

功能描述

取消一个先前为两阶段提交准备好的事务。

注意事项

- 要想回滚一个预备事务，必须是最初发起事务的用户，或者是系统管理员。
- 事务功能由数据库自动维护，不应显式使用事务功能。

语法格式

```
ROLLBACK PREPARED transaction_id ;
```

参数说明

transaction_id

待提交事务的标识符。它不能和任何当前预备事务已经使用了的标识符同名。

相关链接

参考：SQL 语法参考->SQL 语法->COMMIT PREPARED、PREPARE TRANSACTION 章节。

4.5.176.ROLLBACK TO SAVEPOINT

功能描述

ROLLBACK TO SAVEPOINT 用于回滚到一个保存点，隐含地删除所有在该保存点之后建立的保存点。

回滚所有指定保存点建立之后执行的命令。保存点仍然有效，并且需要时可以再次回滚到该点。

注意事项

- 不能回滚到一个未定义的保存点，语法上会报错。
- 在保存点方面，游标有一些非事务性的行为。任何在保存点里打开的游标都会在回滚掉这个保存点之后关闭。如果一个前面打开了的游标在保存点里面，并且游标被一个 FETCH 命令影响，而这个保存点稍后回滚了，那么这个游标的位置仍然在 FETCH 让它指向的位置（也就是 FETCH 不会被回滚）。关闭一个游标的行为也不会被回滚给撤销掉。如果一个游标的操作导致事务回滚，那么这个游标就会置于不可执行状态，所以，尽管一个事务可以用 ROLLBACK TO SAVEPOINT 重新恢复，但是游标不能再使用了。
- 使用 ROLLBACK TO SAVEPOINT 回滚到一个保存点。使用 RELEASE SAVEPOINT 删除一个保存点，但是保留该保存点建立后执行的命令的效果。

语法格式

```
ROLLBACK [ WORK | TRANSACTION ] TO [ SAVEPOINT ] savepoint_name;
```

参数说明

savepoint_name：回滚截至的保存点。

示例

```
--撤销 my_savepoint 建立之后执行的命令的影响。
panweidb=# START TRANSACTION;
```

```
panweidb=# SAVEPOINT my_savepoint;
panweidb=# ROLLBACK TO SAVEPOINT my_savepoint;
--游标位置不受保存点回滚的影响。
panweidb=# DECLARE foo CURSOR FOR SELECT 1 UNION SELECT 2;
panweidb=# SAVEPOINT foo;
panweidb=# FETCH 1 FROM foo;
 ?column?
----------
        1
panweidb=# ROLLBACK TO SAVEPOINT foo;
panweidb=# FETCH 1 FROM foo;
 ?column?
----------
        2
panweidb=# RELEASE SAVEPOINT my_savepoint;
panweidb=# COMMIT;
```

相关链接

参考：SQL 语法参考->SQL 语法->SAVEPOINT、RELEASE SAVEPOINT 章节。

4.5.177.SAVEPOINT

功能描述

SAVEPOINT 用于在当前事务里建立一个新的保存点。

保存点是事务中的一个特殊记号，它允许将那些在它建立后执行的命令全部回滚，把事务的状态恢复到保存点所在的时刻。

注意事项

- 使用 ROLLBACK TO SAVEPOINT 回滚到一个保存点。使用 RELEASE SAVEPOINT 删除一个保存点，但是保留该保存点建立后执行的命令的效果。
- 保存点只能在一个事务块里面建立。在一个事务里面可以定义多个保存点。

- 由于节点故障或者通信故障引起的节点线程或进程退出导致的报错，以及由于 COPY FROM 操作中源数据与目标表的表结构不一致导致的报错，均不能正常回滚到保存点之前，而是整个事务回滚。
- SQL 标准要求，使用 savepoint 建立一个同名保存点时，需要自动删除前面那个同名保存点。在 PanWeiDB 数据库里，我们将保留旧的保存点，但是在回滚或者释放的时候，只使用最近的那个。释放了新的保存点将导致旧的再次成为 ROLLBACK TO SAVEPOINT 和 RELEASE SAVEPOINT 可以访问的保存点。除此之外，SAVEPOINT 是完全符合 SQL 标准的。

语法格式

```
SAVEPOINT savepoint_name;
```

参数说明

savepoint_name：新建保存点的名称。

示例

```
--创建一个新表。
panweidb=# CREATE TABLE table1(a int);

--开启事务。
panweidb=# START TRANSACTION;

--插入数据。
panweidb=# INSERT INTO table1 VALUES (1);

--建立保存点。
panweidb=# SAVEPOINT my_savepoint;

--插入数据。
panweidb=# INSERT INTO table1 VALUES (2);

--回滚保存点。
panweidb=# ROLLBACK TO SAVEPOINT my_savepoint;
```

```
--插入数据。
panweidb=# INSERT INTO table1 VALUES (3);

--提交事务。
panweidb=# COMMIT;

--查询表的内容，会同时看到 1 和 3,不能看到 2，因为 2 被回滚。
panweidb=# SELECT * FROM table1;

--删除表。
panweidb=# DROP TABLE table1;

--创建一个新表。
panweidb=# CREATE TABLE table2(a int);

--开启事务。
panweidb=# START TRANSACTION;

--插入数据。
panweidb=# INSERT INTO table2 VALUES (3);

--建立保存点。
panweidb=# SAVEPOINT my_savepoint;

--插入数据。
panweidb=# INSERT INTO table2 VALUES (4);

--回滚保存点。
panweidb=# RELEASE SAVEPOINT my_savepoint;

--提交事务。
panweidb=# COMMIT;

--查询表的内容，会同时看到 3 和 4。
panweidb=# SELECT * FROM table2;
```

```
--删除表。
panweidb=# DROP TABLE table2;
```

相关链接

参考：SQL 语法参考->SQL 语法->RELEASE SAVEPOINT、ROLLBACK TO SAVEPOINT 章节。

4.5.178.SECURITY LABEL

功能描述

定义或更改应用于对象的安全标签。

注意事项

无

语法格式

```
{
 TABLE object_name |
 FUNCTION function_name [ ( [ [ argmode ] [ argname ] argtype [, ...] ] ) ] |
 MATERIALIZED VIEW object_name |
 PROCEDURE procedure_name [ ( [ [ argmode ] [ argname ] argtype [, ...] ] ) ] |
 ROLE object_name |
 SEQUENCE object_name |
 TRIGGER object_name OF table_name |
 VIEW object_name |
 COLUMN table_name.column.name
} IS 'label' | NULL;
```

参数说明

- object_name

  对象名称。

- function_name

函数名称。

- argmode

  函数参数的模式。

  取值范围：IN，OUT，INOUT 或 VARIADIC（用于声明数组类型的参数），缺省值是 IN。只有 OUT 模式的参数后面能跟 VARIADIC。并且 OUT 和 INOUT 模式的参数不能用在 RETURNS TABLE 的函数定义 中。

- argname

  函数参数的名称。

  取值范围：字符串，要符合标识符命令规范。

- argtype

  函数参数的类型。

- procedure_name

  存储过程名称。

示例

更改一个表的安全标签。

```
SECURITY LABEL FOR panweidb ON TABLE mytable IS 'system_u:object_r:sepgsql_table_t:s0';
```

相关链接

无

4.5.179.SELECT

功能描述

SELECT 用于从表或视图中查询数据。

SELECT 语句就像叠加在数据库表上的过滤器，利用 SQL 关键字从数据表中过滤出用户需要的数据。

注意事项

- 需要对每个在 SELECT 命令中使用的字段具有 SELECT 权限。
- 使用 FOR UPDATE，FOR NO KEY UPDATE，FOR SHARE 或 FOR KEY SHARE 还要求 UPDATE 权限。

语法格式

- 查询数据

```
[ WITH [ RECURSIVE ] with_query [, ...] ]
SELECT [/*+ plan_hint */] [ ALL | DISTINCT | UNIQUE [ ON ( expression [, ...] ) ] ]
  { * | {expression [ [ AS ] output_name ]} [, ...] }
  [ FROM from_item [, ...] ]
  [START WITH condition CONNECT BY PRIOR condition]
  [ WHERE condition ]
  [ [ START WITH condition ] CONNECT BY condition [ ORDER SIBLINGS BY expression
] ]
  [ GROUP BY grouping_element [, ...] ]
  [ HAVING condition [, ...] ]
  [ WINDOW {window_name AS ( window_definition )} [, ...] ]
  [ { UNION | INTERSECT | EXCEPT | MINUS } [ ALL | DISTINCT ] select ]
  [ ORDER BY {expression [ [ ASC | DESC | USING operator ] | nlssort_expression_claus
e ] [ NULLS {
FIRST | LAST } ]} [, ...] ]
  [ LIMIT { [offset,] count | ALL } ]
  [ OFFSET start [ ROW | ROWS ] ]
  [ FETCH { FIRST | NEXT } [ count ] { ROW | ROWS } ONLY ]
  [ {FOR { UPDATE | NO KEY UPDATE | SHARE | KEY SHARE } [ OF table_name [, ...] ] [ N
OWAIT |  WAIT N | SKIP LOCKED]} [...] ]
TABLE { ONLY { (table_name) | table_name } | table_name [ * ]};
```

- 其中子查询 with_query 为：

```
with_query_name [ ( column_name [, ...] ) ] AS [ [ NOT ] MATERIALIZED ] ( {select | values | insert | update | delete} )
```

❖ 其中指定查询源 from_item 为：

```
{[ ONLY ] table_name [ * ] [ partition_clause ] [ [ AS ]alias [ ( column_alias [, ...] ) ] ]
[ TABLESAMPLE sampling_method ( argument [, ...] ) [ REPEATABLE (seed ) ] ]|( select ) [ AS ] alias [ ( column_alias [, ...] ) ]
|with_query_name [ [ AS ] alias [ ( column_alias [, ...] ) ]]
|function_name ( [ argument [, ...] ] ) [ AS ] alias [ (column_alias [, ...]
| column_definition [, ...] ) ]
|function_name ( [ argument [, ...] ] ) AS ( column_definition[, ...] )
|from_item [ NATURAL ] join_type from_item [ ON join_condition |USING ( join_column [, ...] ) ]}
```

❖ 其中 group 子句为：

```
expression
| ( expression [, ...] )
| ROLLUP ( { expression | ( expression [, ...] ) } [, ...] )
| CUBE ( { expression | ( expression [, ...] ) } [, ...] )
| GROUPING SETS ( grouping_element [, ...] )
```

❖ 其中指定分区 partition_clause 为：

```
    PARTITION { ( partition_name ) |

    FOR ( partition_value [, ...] ) }
```

【说明】
指定分区只适合分区表。

❖ 其中设置排序方式 nlssort_expression_clause 为：

```
    NLSSORT ( column_name, ' NLS_SORT = { SCHINESE_PINYIN_M |generic_m_ci } ' )
```

❖ 简化版查询语法，功能相当于 select * from table_name。

```
TABLE { ONLY {(table_name)| table_name} | table_name [ * ]};
```

❖ 在 MySQL 兼容模式下，支持如下语法：

【须知】

- 该语法仅在数据库兼容模式为 MySQL 时能够使用（即数据库初始化时指定 DBCOMPATIBILITY='B'），在其他数据库兼容模式下不能使用该特性。
- 有关该语法的详细内容请参考 MySQL 兼容性的 IGNORE|FORCE INDEX 语法。

```
        table_name [ { FORCE | IGNORE } { INDEX | KEY } (index_hint_list) | USE { INDEX | KEY } (opt_index_hint_list) ]

        index_hint_list:

        index[,index...]

        opt_index_hint_list:

        empty or index_hint_list
```

参数说明

- **WITH [RECURSIVE] with_query [, ...]**
  - 用于声明一个或多个可以在主查询中通过名称引用的子查询，相当于只在主查询中存在的临时表，其可以将复杂的查询简化。

  其中 with_query 的详细格式为：

```
with_query_name [ ( column_name [, ...]) ] AS ( {select | values | insert | update | delete} )
```

  - with_query_name 指定子查询生成的结果集名称，在查询中可使用该名称访问子查询的结果集。
  - column_name 指定子查询结果集中显示的列名。
  - 每个子查询可以是 SELECT，VALUES，INSERT，UPDATE 或 DELETE 语句。
  - 如果声明了 RECURSIVE，那么允许 AS 后的 SELECT 子查询通过名称引用自己，详细格式为：

```
non_recursive_term UNION [ ALL | DISTINCT ] recursive_term
```

- 即由非递归项、UNION、递归项组成
- 只有递归项中可以引用自己
- 每个查询中只允许一个递归自引用

❖ **plan_hint 子句**

以/*+*/的形式在 SELECT 关键字后，用于对 SELECT 对应的语句块生成的计划进行 hint 调优。

❖ **ALL**

声明返回所有符合条件的行，是默认行为，可以省略该关键字。

❖ **DISTINCT [ON (expression [, ...])]**

从 SELECT 的结果集中删除所有重复的行，使结果集中的每行都是唯一的。ON (expression [, ...])只保留那些在给出的表达式上运算出相同结果的行集合中的第一行。

【须知】

DISTINCT ON 表达式是使用与 ORDER BY 相同的规则进行解释的。除非使用了 ORDER BY 来保证需要的行首先出现，否则，"第一行" 是不可预测的。

❖ **SELECT 列表**

指定查询表中列名，可以是部分列或者是全部（使用通配符*表示）。 通过使用子句 AS output_name 可以为输出字段取个别名，这个别名通常用于输出字段的显示。列名可以用下面几种形式表达：

- 手动输入列名，多个列之间用英文逗号（,）分隔。
- 可以是 FROM 子句里面计算出来的字段。

❖ **FROM 子句**

为 SELECT 声明一个或者多个源表。 FROM 子句涉及的元素如下所示。

- table_name：表名或视图名，名称前可加上模式名，如：schema_name.table_name。
- alias：给表或复杂的表引用起一个临时的表别名，以便被其余的查询引用。别名用于缩写或者在自连接中消除歧义。如果提供了别名，它就会完全隐藏表的实际名。

- TABLESAMPLE sampling_method (argument [, ...]) [REPEATABLE (seed)]：table_name 之后的 TABLESAMPLE 子句表示应该用指定的 sampling_method 来检索表中行的子集。

  sampling_method 可以被指定为以下方法：

  - bernoulli，按行采样。
  - system，按块(页)采样。
  - hybrid，混合采样，argument 可以分别指定按行采样和按块(页)采样的百分比。

  argument 部分用来指定采样百分比，是一个非空数值表达式（如 10 或'10'），取值范围是[0.000001,100)。

  可选的 REPEATABLE 部分指定一个用于产生采样方法中随机数的种子值 seed，是一个非空数值表达式（如 10 或'10'），种子值的取值范围是[0,4294967295]。如果查询时表没有被更改，指定相同 seed 和 argument 值的两个查询将会选择该表相同的采样。但是不同的种子值通常将会产生不同的采样。如果没有给出 REPEATABLE，则会基于一个系统产生的种子为每一个查询选择一个新的随机采样。

  table_name 可以是普通表，分区表，普通视图，物化视图和键保留（key-preserved）的连接视图；普通视图指在单表上查询创建的视图；键保留的连接视图指对键保留表进行连接操作后创建的视图，但不支持对包含超过一个非键保留表的连接视图进行数据采样。
- column_alias：列别名。
- PARTITION：查询分区表的某个分区的数据。
- partition_name：分区名。
- partition_value：指定的分区键值。在创建分区表时，如果指定了多个分区键，可以通过 PARTITION FOR 子句指定的这一组分区键的值，唯一确定一个分区。
- subquery：FROM 子句中可以出现子查询，创建一个临时表保存子查询的输出。
- with_query_name：WITH 子句同样可以作为 FROM 子句的源，可以通过 WITH 查询的名称对其进行引用。
- function_name：函数名称。函数调用也可以出现在 FROM 子句中。

- USING(join_column[，...])：ON left_table.a = right_table.a AND left_table.b = right_table.b ...的简写。要求对应的列必须同名。USING 意味着，每个等号表达式里面的列，只有一列会输出，而不是全部。
- NATURAL：NATURAL 是具有相同名称的两个表的所有列的 USING 列表的简写，如果没有一样名称的列，则等效于 join on true。
- from_item：用于连接的查询源对象的名称。
- join_type：有 5 种类型，如下所示。
  - [INNER] JOIN：一个 JOIN 子句组合两个 FROM 项。可使用圆括弧以决定嵌套的顺序。如果没有圆括弧，JOIN 从左向右嵌套。在任何情况下，JOIN 都比逗号分隔的 FROM 项绑定得更紧。
  - LEFT [OUTER] JOIN：返回笛卡尔积中所有符合连接条件的行，再加上左表中通过连接条件没有匹配到右表行的那些行。这样，左边的行将扩展为生成表的全长，方法是在那些右表对应的字段位置填上 NULL。请注意，只在计算匹配的时候，才使用 JOIN 子句的条件，外层的条件是在计算完毕之后 施加的。
  - RIGHT [OUTER] JOIN：返回所有内连接的结果行，加上每个不匹配的右边行（左边用 NULL 扩展）。这只是一个符号上的方便，因为总是可以把它转换成一个 LEFT OUTER JOIN，只要把左边和右边的输入互换位置即可。
  - FULL [OUTER] JOIN：返回所有内连接的结果行，加上每个不匹配的左边行（右边用 NULL 扩展），再加上每个不匹配的右边行（左边用 NULL 扩展）。
  - CROSS JOIN：CROSS JOIN 等效于 INNER JOIN ON（TRUE），即没有被条件删除的行。这种连接类型只是符号上的方便，因为它们与简单的 FROM 和 WHERE 的效果相同。

【须知】

必须为 INNER 和 OUTER 连接类型声明一个连接条件，即 NATURAL ON，join_condition，USING (join_column [， ...]) 之一。但是它们不能

出现在 CROSS JOIN 中。其中 CROSS JOIN 和 INNER JOIN 生成一个简单的笛卡尔积，和在 FROM 的顶层列出两个项的结果相同。

- ON join_condition：连接条件，用于限定连接中的哪些行是匹配的。如：

```
ON left_table.a = right_table.a
```

❖ **WHERE 子句**

WHERE 子句构成一个行选择表达式，用来缩小 SELECT 查询的范围。condition 是返回值为布尔型的任意表达式，任何不满足该条件的行都不会被检索。

WHERE 子句中可以通过指定“（+）”操作符的方法将表的连接关系转换为外连接。但是不建议用户使用这种用法，因为这并不是 SQL 的标准语法，在做平台迁移的时候可能面临语法兼容性的问题。同时，使用“（+）”有很多限制：

- “（+）”只能出现在 where 子句中。
- 如果 from 子句中已经有指定表连接关系，那么不能再在 where 子句中使用“（+）”。
- “（+）”只能作用在表或者视图的列上，不能作用在表达式上。
- 如果表 A 和表 B 有多个连接条件，那么必须在所有的连接条件中指定“（+）”，否则“（+）”将不会生效，表连接会转化成内连接，并且不给出任何提示信息。
- 在任何外连接的 WHERE 子句中，无论指定哪种形式，都不能将带有"（+）"的列与子查询进行比较。
- 如果“（+）”作用的表，不在当前查询或者子查询的 from 子句中，则会报错。如果“（+）”作用的对端的表不存在，则不报错，同时连接关系会转化为内连接。
- ''(+)"作用的表达式不能直接通过"OR"连接。
- 如果“（+）”作用的列是和一个常量的比较关系，那么这个表达式会成为 join 条件的一部分。
- 同一个表不能对应多个外表。
- “（+）”只能出现"比较表达式"，"NOT 表达式"，"ANY 表达式"，"ALL 表达式"，"IN 表达式"，"NULLIF 表达式"，"IS DISTINCT FROM 表达

式"，"IS OF"表达式。“（+）”不能出现在其他类型表达式中，并且这些表达式中不允许出现通过"AND"和"OR"连接的表达式。

- “（+）”只能转化为左外连接或者右外连接，不能转化为全连接，即不能在一个表达式的两个表上同时指定“（+）”。

【须知】

对于 WHERE 子句的 LIKE 操作符，当 LIKE 中要查询特殊字符“%”、“_”、“\”的时候需要使用反斜杠“\”来进行转义。

❖ **START WITH 子句**

START WITH 子句通常与 CONNECT BY 子句同时出现，数据进行层次递归遍历查询，START WITH 代表递归的初始条件。若省略该子句，单独使用 CONNECT BY 子句，则表示以表中的所有行作为初始集合。

❖ **CONNECT BY 子句**

CONNECT BY 代表递归连接条件，和 START WITH 一起子句一起使用，实现数据遍历递归的功能。

```
create table test(name varchar, id int, fatherid int);
 insert into test values('A', 1, 0), ('B', 2, 1), ('C', 3, 1), ('D', 4, 1), ('E', 5, 2);
  select * from test start with id = 1 connect by prior id = fatherid order sibli
ngs by id desc;
   name | id | fatherid
   ------+----+----------
   A    | 1 |      0
   D    | 4 |      1
   C    | 3 |      1
   B    | 2 |      1
   E    | 5 |      2
   (5 rows)
```

CONNECT BY 代表递归连接条件，CONNECT BY 条件中可以对列指定 PRIOR 关键字代表以这列为递归键进行递归。当前约束只能对表中的列指定 PRIOR，不支持对表达式、类型转换指定 PRIOR 关键字。

若在递归连接条件前加 NOCYCLE，则表示遇到循环记录时停止递归。

（注：含 START WITH .. CONNECT BY 子句的 SELECT 语句不支持使用 FOR KEY SHARE/SHARE/NO KEY UPDATE/UPDATE 锁）。

Start with 语句的执行流程是：

（1）由 start with 区域的条件选择初始的数据集。上述例子里，先把('A', 1, 0) 选择出来。然后把初始的数据集设置为工作集。

（2）只要工作集不为空，会用工作集的数据作为输入，查询下一轮的数据，过滤条件有 conntect by 区域指定。其中，PRIOR 关键字表示当前记录，比如上位例子中 prior id = fatherid 表示当前记录的 id 是下一条记录的 fatherid。

（3）把 2 中筛选出来的数据集，设为工作集，返回第二步重复操作。

同时，数据库为每一条选出来的数据添加下述的伪列，方便用户了解数据在递归或者树状结构中的位置。

- LEVEL：节点的层级
- CONNECT_BY_ISLEAF：是否为叶子结点

除了伪列外，还提供下述的查询函数

- sys_connect)by_path(col, separtor)：返回从根节点到当前行的连接路径。参数 col 为路径中显示的列的名称，separator 为连接符。
- connect_by_root(col)：显示该节点最顶级的节点，col 为输出列的名称。

当前 Start with 默认行为是宽度优先搜索，但可以通过和伪列配合，实现深度优先搜索。比如：

```
select sys_connect_by_path(name,'-') as path,*,LEVEL from test start with i
d = 1 connect by fatherid=prior id order by path;
   path  | name | id | fatherid | level
--------+------+----+----------+-------
-A    |A   | 1|     0|   1
-A-B  |B   | 2|     1|   2
-A-B-E|E   | 5|     2|   3
-A-C  |C   | 3|     1|   2
-A-D  |D   | 4|     1|   2
(5 rows)
```

❖ **ORDER SIBLINGS BY 子句**

ORDER SIBLINGS BY 通常和 START WITH、CONNECT BY 子句同时使用，用法和 ORDER BY 子句一样，用于在递归过程中的层级排序。当前不支持接

函数调用，如果有函数调用，且函数参数中存在列，则取第一个列作为排序键排序。

- **GROUP BY 子句**

  将查询结果按某一列或多列的值分组，值相等的为一组。

  - GROUPING SETS (grouping_element [, ...])：GROUPING SETS 子句是 GROUP BY 子句的进一步扩展，它可以使用户指定多个 GROUP BY 选项。这样做可以通过裁剪用户不需要的数据组来提高效率。当用户指定了所需的数据组时，数据库不需要执行完整 CUBE 或 ROLLUP 生成的聚合集合。
  - CUBE ({ expression I(expression [, ...])} [, ...])：CUBE 是自动对 group by 子句中列出的字段进行分组汇总，结果集将包含维度列中各值的所有可能组合，以及与这些维度值组合相匹配的基础行中的聚合值。它会为每个分组返回一行汇总信息，用户可以使用 CUBE 来产生交叉表值。比如，在 CUBE 子句中给出二个表达式(n= 3)，运算结果为 2" = 23 = 8 组。以 n 个表达式的值分组的行称为常规行，其余的行称为超级聚集行。例如: CUBE(C1,c2,c3)等效于

```
GROUPING SETS(
(c1, c2，c3),
(c1，c2),
(c2，c3),
(c1，c3),
(c1),
(c2),
(c3),
()
)
```

  - ROLLUP ({ expression 1 (expression [, ...])3[. ...])：ROLLUP 是生成多个分组集合的快捷功能。与 CUBE 子句的差异是，ROLLUP 不生成基于特定列所有可能的分组集合，生成分组集合为其子集.ROLLUP 假设输入列之间存在层次结构，从而生成有意义的所有分组集合。例如: ROLLUP(C1.c2.c3)仅生成 4 种分组集合，等效于:

```
GROUPING SETS (
```

```
(c1,c2,  c3),
(c1, c2),
(c1),
)
```

- SELECT 子句中指定字段必须要包含在 group by 语句后，作为分组的依据，如果没有包含则会报错。除非 SELECT 子句中的字段包含在聚集函数中，因为对于未分组的字段，可能会返回多个数值。
- 如果任何 GROUPING SETS、ROLLUP、CUBE 作为分组字段存在，则 GROUP BY 子句整体上定义了数个独立的分组集。其效果等效于在子查询间构建一个 UNIONALL，子查询带有分组集作为它们的 GROUP BY 子句。
- 当前，FOR UPDATE、FOR SHARE、FOR NO KEY UPDATE、FOR KEY SHARE 不能和 GROUP BY 子句一起指定。
- **HAVING 子句**

  与 GROUP BY 子句配合用来选择特殊的组。HAVING 子句将组的一些属性与一个常数值比较，只有满足 HAVING 子句中的逻辑表达式的组才会被提取出来。

【须知】

当前，FOR UPDATE、FOR SHARE、FOR NO KEY UPDATE、FOR KEY SHARE 不能和 HAVING 子句一起指定。

- **WINDOW 子句**

  一般形式为：

```
WINDOW window_name AS ( window_definition ) [,  ...]
```

  window_name 是可以被随后的窗口定义所引用的名称，

  window_definition 可以是以下的形式：

```
[ existing_window_name ]
[ PARTITION BY expression [, ...] ]
[ ORDER BY expression [ ASC | DESC | USING operator ] [ NULLS {FIRST | LAST } ] [,
 ...] ]
[ frame_clause]
```

frame_clause 为窗函数定义一个窗口帧 window frame，窗函数（并非所有）依赖于帧，window frame 是当前查询行的一组相关行。

frame_clause 可以是以下的形式：

```
[ RANGE | ROWS ] frame_start
[ RANGE | ROWS ] BETWEEN frame_start AND frame_end
```

frame_start 和 frame_end 可以是：

```
UNBOUNDED PRECEDING      //分区第一行开始的帧
value PRECEDING           //当前行前 value 行开始或结束的帧
CURRENT ROW               //当前行开始或结束的帧
value FOLLOWING           //当前行后 value 行开始或结束的帧
UNBOUNDED FOLLOWING   //分区最后一行结束的帧
```

- frame_start 缺省为 UNBOUNDED PRECEDING，frame_end 缺省为 CURRENT ROW
- frame_end 的取值在上面列出的顺序中必须晚于 frame_start 的取值
- expr 可以是表达式或数字。若为数字则不可为负数或空，可以为 0；若为表达式，则结果可以为其中表达式的结果可以是 0、正数和时间区间。

【须知】

- 其中 expr FOLLOWING 和 expr PRECEDING 选项在 Oracle 兼容模式下支持，详情可见 Oracle 兼容性手册的开窗函数支持表达式。
- 对列存表的查询目前只支持 row_number 窗口函数，不支持 frame_clause。

WINDOW 子句可指定窗口函数的行为，在处理窗口定义相同的窗口函数时，使用 WINDOW 子句，在 OVER 中引用，能使 SQL 语句更简单，例如：

```
SELECT sum(count) OVER w, avg(count) OVER w
  FROM table_count
  WINDOW w AS (PARTITION BY name ORDER BY count DESC);
```

- **UNION 子句**

UNION 计算多个 SELECT 语句返回行集合的并集。

UNION 子句有如下约束条件：

- 除非声明了 ALL 子句，否则缺省的 UNION 结果不包含重复的行。

- 同一个 SELECT 语句中的多个 UNION 操作符是从左向右计算的，除非用圆括弧进行了标识
- FOR UPDATE 不能在 UNION 的结果或输入中声明。

一般表达式：

```
select_statement UNION [ALL] select_statement
```

- select_statement 可以是任何没有 ORDER BY、LIMIT、FOR UPDATE 子句的 SELECT 语句。
- 如果用圆括弧包围，ORDER BY 和 LIMIT 可以附着在子表达式里。

❖ **INTERSECT 子句**

INTERSECT 计算多个 SELECT 语句返回行集合的交集，不含重复的记录。

INTERSECT 子句有如下约束条件：

- 同一个 SELECT 语句中的多个 INTERSECT 操作符是从左向右计算的，除非用圆括弧进行了标识。
- 当对多个 SELECT 语句的执行结果进行 UNION 和 INTERSECT 操作的时候，会优先处理 INTERSECT。

一般形式：

```
select_statement INTERSECT select_statement
```

select_statement 可以是任何没有 FOR UPDATE 子句的 SELECT 语句。

❖ **EXCEPT 子句**

EXCEPT 子句有如下的通用形式：

```
select_statement EXCEPT [ ALL ] select_statement
```

select_statement 是任何没有 FOR UPDATE 子句的 SELECT 表达式。

- EXCEPT 操作符计算存在于左边 SELECT 语句的输出而不存在于右边 SELECT 语句输出的行。
- EXCEPT 的结果不包含任何重复的行，除非声明了 ALL 选项。使用 ALL 时，一个在左边表中有 m 个重复而在右边表中有 n 个重复的行将在结果中出现 max(m-n,0)次。
- 除非用圆括弧指明顺序，否则同一个 SELECT 语句中的多个 EXCEPT 操作符是从左向右计算的。EXCEPT 和 UNION 的绑定级别相同。

- 不能给 EXCEPT 的结果或者任何 EXCEPT 的输入声明 FOR UPDATE 子句。

❖ **MINUS 子句**

与 EXCEPT 子句具有相同的功能和用法。

❖ **ORDER BY 子句**

对 SELECT 语句检索得到的数据进行升序或降序排序。对于 ORDER BY 表达式中包含多列的情况：

- 首先根据最左边的列进行排序，如果这一列的值相同，则根据下一个表达式进行比较，依此类推。
- 如果对于所有声明的表达式都相同，则按随机顺序返回。
- 在与 DISTINCT 关键字一起使用的情况下，ORDER BY 中排序的列必须包括在 SELECT 语句所检索的结果集的列中。例外：在 B 兼容模式下，DISTINCT 列时可以使用非 ORDER-BY 列。（参见《PanWeiDB V2.0-S3.0.1_B01 兼容性手册》->MySQL 兼容性->SQL 语法->DISTINCT 列时可以使用非 ORDER-BY 列。）

【须知】

❖ 如果要支持中文拼音排序和不区分大小写排序，需要在初始化数据库时指定编码格式为 UTF-8 或 GBK。 命令如下：

❖ initdb --E UTF8 --D ../data --locale=zh_CN.UTF-8 或 initdb --E GBK --D ../data -- locale=zh_CN.GBK

❖ 注意：若要使用 pg_zhtrgm 插件，initdb 时指定的 -E 和--locale 参数对应的编码规则必须一致。

❖ **LIMIT 子句**

```
LIMIT { count | ALL }
```

当 limit 后面跟一个参数的时候，该参数表示要取的行数。 当 limit 后面跟两个参数的时候，第一个数表示要跳过的行数，后一位表示要取的行数。

❖ **OFFSET 子句**

```
OFFSET start { ROW | ROWS }
```

start 声明开始返回行之前忽略的行数。当 limit 和 offset 组合使用的时候，limit 后面只能有一个参数，表示要取的行数,offset 表示要跳过的行数。

- FETCH { FIRST | NEXT } [count] { ROW | ROWS } ONLY

如果不指定 count，默认值为 1，FETCH 子句限定返回查询结果从第一行开始的总行数。

❖ **锁定子句**

- FOR UPDATE 子句将对 SELECT 检索出来的行进行加锁。这样避免它们在当前事务结束前被其他事务修改或者删除，即其他企图 UPDATE、DELETE、 SELECT FOR UPDATE 这些行的事务将被阻塞，直到当前事务结束。
- 支持在锁定子句中使用 SKIP LOCKED 选项，表示跳过其他未提交的事务锁定记录，可以避免由于多个使用者同时访问表引起的锁争用问题。PostgreSQL 兼容模式下的 SKIP LOCKED 功能，请参考《PanWeiDB V2.0-S3.0.1_B01 PostgreSQL 兼容性》：SELECT FOR 支持 SKIP LOCKED。其他兼容模式下使用此功能时，锁的行为不完全相同，请注意区分。
- 为了避免操作等待其他事务提交，可使用 NOWAIT 选项，如果被选择的行不能立即被锁住，将会立即汇报一个错误，而不是等待。
- WAIT N 选项：如果被选择的行不能立即被锁住，等待 N 秒（其中，N 为 int 类型，取值范围：0<=N<=2147483），N 秒内获取锁则正常执行，否则报错。
- FOR NO KEY UPDATE 行为与 FOR UPDATE 类似，不过获得的锁较弱：这种锁将不会阻塞尝试在相同行上获得锁的 SELECT FOR KEY SHARE 命令。任何不获取 FOR UPDATE 锁的 UPDATE 也会获得这种锁模式。FOR SHARE 的行为类似，只是它在每个检索出来的行上要求一个共享锁，而不是一个排他锁。一个共享锁阻塞其他事务执行 UPDATE、DELETE、SELECT，不阻塞 SELECT FOR SHARE。

- 如果在 FOR UPDATE 或 FOR SHARE 中明确指定了表名称，则只有这些指定的表被锁定，其他在 SELECT 中使用的表将不会被锁定。否则，将锁定该命令中所有使用的表。
- 如果 FOR UPDATE 或 FOR SHARE 应用于一个视图或者子查询，它同样将锁定所有该视图或子查询中使用到的表。
- 多个 FOR UPDATE 和 FOR SHARE 子句可以用于为不同的表指定不同的锁定模式。
- 如果一个表中同时出现（或隐含同时出现）在 FOR UPDATE 和 FOR SHARE 子句中，则按照 FOR UPDATE 处理。类似的，如果影响一个表的任意子句中出现了 NOWAIT，该表将按照 NOWAIT 处理。

【须知】

- 对列存表的查询不支持 for update/share。
- PostgreSQL 兼容模式下的 SKIP LOCKED 功能请参考《PostgreSQL 兼容性手册》中的 SQL 语法：SELECT FOR 支持 SKIP LOCKED。在其他兼容模式下使用此功能时，锁的行为不完全相同，请注意区分。

- **NLS_SORT**

  指定某字段按照特殊方式排序。目前仅支持中文拼音格式排序和不区分大小写排序。

  取值范围：

  - SCHINESE_PINYIN_M，按照中文拼音排序。如果要支持此排序方式，在创建数据库时需要指定编码格式为"GBK"，否则排序无效。
  - generic_m_ci，不区分大小写排序。

- **PARTITION 子句**

  查询某个分区表中相应分区的数据。

示例

通过子查询得到一张临时表 temp_t，然后查询表 temp_t 中的所有数据。

```
WITH temp_t(name,isdba) AS (SELECT usename,usesuper FROM pg_user) SELECT * F
ROM temp_t;
```

创建多级菜单表

```
CREATE TABLE exp_menu (
 id int,
 name text,
 parent_id int
);

--插入数据
INSERT INTO exp_menu VALUES (1, 'grandpa', 0),(2, 'father', 1),(3, 'son', 2);

--使用 WITH RECURSIVE 递归查询菜单关系
WITH RECURSIVE res AS (
  SELECT id, name, parent_id
  FROM exp_menu
  WHERE id = 3
  UNION
  SELECT m.id,
       m.name || ' > ' || r.name,
       m.parent_id
  FROM res r INNER JOIN exp_menu m ON m.id = r.parent_id
)
select * from res;
 id |         name          | parent_id
----+-----------------------+-----------
  3 | son                   |         2
  2 | father > son          |         1
  1 | grandpa > father > son |         0
(3 rows)
```

创建表 test 并插入测试数据。

```
CREATE TABLE test(id int,name varchar);
INSERT INTO test(id,name) VALUES('1','zhangsan');
INSERT INTO test(id,name) VALUES('2','lisi');
INSERT INTO test(id,name) VALUES('3','zhangsan');
INSERT INTO test(id,name) VALUES('4','wangwu');
```

```
INSERT INTO test(id,name) VALUES('5','xiaoming');
INSERT INTO test(id,name) VALUES('6','xiaohong');
INSERT INTO test(id,name) VALUES('7','LISI');
```

查询 test 表的所有 name 记录，且去除重复。

```
SELECT DISTINCT(name) FROM test;
```

LIMIT 子句示例：获取表中一条记录。

```
SELECT * FROM test LIMIT 1;
```

查询所有记录，且按字母升序排列。

```
SELECT name FROM test ORDER BY name;
```

GROUP BY 子句示例：根据查询条件过滤，并对结果进行分组。

```
SELECT name, AVG(id) FROM test GROUP BY name HAVING AVG(id) > 2;
```

GROUP BY CUBE 子句示例：根据查询条件过滤，并对结果进行分组汇总。

```
SELECT name,AVG(id) FROM test GROUP BY CUBE(id,name);
```

GROUP BY GROUPING SETS 子句示例:根据查询条件过滤，并对结果进行分组汇总。

```
SELECT name,AVG(id) FROM test GROUP BY GROUPING SETS((id,name),id);
```

UNION 子句示例：将表 test 里 name 字段中的内容以'l'开头和以'z'开头的进行合并。

```
SELECT id, name  FROM test WHERE test.name LIKE 'l%' UNION SELECT id, name FRO
M test WHERE test.name LIKE 'z%';
```

NLS_SORT 子句示例：中文拼音排序。

```
SELECT * FROM test ORDER BY NLSSORT(name, 'NLS_SORT = SCHINESE_PINYIN_M');
```

不区分大小写排序:

```
SELECT * FROM test ORDER BY NLSSORT(name, 'NLS_SORT = generic_m_ci');
```

通过表别名，从 pg_user 和 pg_user_status 这两张表中获取数据。

```
SELECT a.usename,b.locktime FROM pg_user a,pg_user_status b WHERE a.usesysid
=b.roloid;
```

FULL JOIN 子句示例：将 pg_user 和 pg_user_status 这两张表的数据进行全连接显示，即数据的合集。

```
SELECT a.usename,b.locktime,a.usesuper FROM pg_user a FULL JOIN pg_user_statu
s b on a.usesysid=b.roloid;
```

创建分区表 test_p

```
CREATE TABLE test_p
(
r_reason_sk integer,
r_reason_id character(16),
r_reason_desc character(100)
)
PARTITION BY RANGE (r_reason_sk)
(
partition P_05_BEFORE values less than (05),
partition P_15 values less than (15),
partition P_25 values less than (25),
partition P_35 values less than (35),
partition P_45_AFTER values less than (MAXVALUE)
)
;
```

插入数据。

```
INSERT INTO test_p
values(3,'AAAAAAAABAAAAAAA','reason
1'),(10,'AAAAAAAABAAAAAAA','reason
2'),(4,'AAAAAAAABAAAAAAA','reason
3'),(10,'AAAAAAAABAAAAAAA','reason
4'),(10,'AAAAAAAABAAAAAAA','reason
5'),(20,'AAAAAAAACAAAAAAA','reason
6'),(30,'AAAAAAAACAAAAAAA','reason 7');
```

PARTITION 子句示例：从 test_p 的表分区 P_05_BEFORE 中获取数据。

```
SELECT * FROM test_p PARTITION (P_05_BEFORE);
```

GROUP BY 子句示例：按 r_reason_id 分组统计 test_p 表中的记录数。

```
SELECT COUNT(*),r_reason_id FROM test_p GROUP BY r_reason_id;
```

HAVING 子句示例：按 r_reason_id 分组统计 tpcds.reason_p 表中的记录，并只显示 r_reason_id 个数大于 2 的信息。

```
SELECT COUNT(*) c,r_reason_id FROM test_p GROUP BY r_reason_id HAVING c>2;
```

IN 子句示例：按 r_reason_id 分组统计 tpcds.reason_p 表中的 r_reason_id 个数，并只显示 r_reason_id 值为 AAAAAAAABAAAAAAA 或 AAAAAAAADAAAAAAA 的个数。

```
SELECT COUNT(*),r_reason_id FROM test_p GROUP BY
r_reason_id HAVING r_reason_id IN('AAAAAAAABAAAAAAA','AAAAAAAADAAAAAAA
');
```

INTERSECT 子句示例：查询 r_reason_id 等于 AAAAAAAABAAAAAAA，并且 r_reason_sk 小于 5 的信息。

```
SELECT * FROM test_p WHERE r_reason_id='AAAAAAAABAAAAAAA' INTERSECT SELEC
T * FROM test_p WHERE r_reason_sk<5;
```

EXCEPT 子句示例：查询 r_reason_id 等于 AAAAAAAABAAAAAAA，并且去除 r_reason_sk 小于 4 的信息。

```
SELECT * FROM test_p WHERE r_reason_id='AAAAAAAABAAAAAAA' EXCEPT SELECT *
FROM test_p WHERE r_reason_sk<4;
```

通过在 where 子句中指定“（+）”来实现左连接。

```
SELECT test_p.r_reason_sk ,test.id FROM test_p,test WHERE test_p.r_reason_sk=test.i
d(+);
```

通过在 where 子句中指定“（+）”来实现右连接。

```
SELECT test_p.r_reason_sk ,test.id FROM test_p,test WHERE test_p.r_reason_sk(+)=te
st.id;
```

通过在 where 子句中指定“（+）”来实现左连接，并且增加连接条件。

```
SELECT test_p.r_reason_sk ,test.id FROM test_p,test WHERE test_p.r_reason_sk=test.id(+) AND test.id(+) < 1 ORDER BY 1 LIMIT 1 ;
```

【说明】

- 不支持在 where 子句中指定“（+）”的同时使用内层嵌套 AND/OR 的表达式。
- where 子句在不支持表达式宏指定“（+）”会报错。
- where 子句在表达式的两边都指定“（+）”会报错。

删除表。

```
DROP TABLE test_p,test;
```

4.5.180.SELECT INTO

功能描述

SELECT INTO 用于根据查询结果创建一个新表，并且将查询到的数据插入到新表中。

数据并不返回给客户端，这一点和普通的 SELECT 不同。新表的字段具有和 SELECT 的输出字段相同的名称和数据类型。

注意事项

- CREATE TABLE AS 的作用和 SELECT INTO 类似，且提供了 SELECT INTO 所提供功能的超集。建议使用 CREATE TABLE AS 语法替代 SELECT INTO，因为 SELECT INTO 不能在存储过程中使用。

语法格式

```
[ WITH [ RECURSIVE ] with_query [, ...] ]
SELECT [ ALL | DISTINCT [ ON ( expression [, ...] ) ] ]
  { * | {expression [ [ AS ] output_name ]} [, ...] }
  INTO [ [ GLOBAL | LOCAL ] [ TEMPORARY | TEMP ] | UNLOGGED ] [ TABLE ] new_table
  [ FROM from_item [, ...] ]
```

```
  [ WHERE condition ]
  [ GROUP BY expression [, ...] ]
  [ HAVING condition [, ...] ]
  [ WINDOW {window_name AS ( window_definition )} [, ...] ]
  [ { UNION | INTERSECT | EXCEPT | MINUS } [ ALL | DISTINCT ] select ]
  [ ORDER BY {expression [ [ ASC | DESC | USING operator ] | nlssort_expression_claus
e ] [ NULLS { FIRST | LAST } ]} [, ...] ]
  [ LIMIT { count | ALL } ]
  [ OFFSET start [ ROW | ROWS ] ]
  [ FETCH { FIRST | NEXT } [ count ] { ROW | ROWS } ONLY ]
  [ {FOR { UPDATE | SHARE } [ OF table_name [, ...] ] [ NOWAIT | WAIT N]} [...] ];
```

参数说明

- **new_table**

  new_table 指定新建表的名称。

- **UNLOGGED**

  指定表为非日志表。在非日志表中写入的数据不会被写入到预写日志中，这样就会比普通表快很多。但是，它也是不安全的，非日志表在冲突或异常关机后会被自动删截。非日志表中的内容也不会被复制到备用服务器中。在该类表中创建的索引也不会被自动记录。

  - 使用场景：非日志表不能保证数据的安全性，用户应该在确保数据已经做好备份的前提下使用，例如系统升级时进行数据的备份。
  - 故障处理：当异常关机等操作导致非日志表上的索引发生数据丢失时，用户应该对发生错误的索引进行重建。

- **GLOBAL | LOCAL**

  创建临时表时可以在 TEMP 或 TEMPORARY 前指定 GLOBAL 或 LOCAL 关键字。如果指定 GLOBAL 关键字，PanWeiDB 会创建全局临时表，否则 PanWeiDB 会创建本地临时表。

- **TEMPORARY | TEMP**

  如果指定 TEMP 或 TEMPORARY 关键字，则创建的表为临时表。临时表分为全局临时表和本地临时表两种类型。创建临时表时如果指定 GLOBAL 关键字则为全局临时表，否则为本地临时表。

全局临时表的元数据对所有会话可见，会话结束后元数据继续存在。会话与会话之间的用户数据、索引和统计信息相互隔离，每个会话只能看到和更改自己提交的数据。全局临时表有两种模式：一种是基于会话级别的（ON COMMIT PRESERVE ROWS），当会话结束时自动清空用户数据；一种是基于事务级别的（ON COMMIT PRESERVE ROWS），当执行 commit 或 rollback 时自动清空用户数据。建表时如果没有指定 ON COMMIT 选项，则缺省为会话级别。与本地临时表不同，全局临时表建表时可以指定非 pg_temp_开头的 schema。

由于临时表只在当前会话创建，对于涉及对临时表操作的 DDL 语句，会产生 DDL 失败的报错。因此，建议 DDL 语句中不要对临时表进行操作。

TEMP 和 TEMPORARY 等价。

【须知】

本地临时表通过每个会话独立的以 pg_temp 开头的 schema 来保证只对当前会话可见，因此，不建议用户在日常操作中手动删除以 pg_tem、pg_toast_temp 开头的 schema。

如果建表时不指定 TEMPORARY/TEMP 关键字，而指定表的 schema 为当前会话的 pg_temp_开头的 schema，则此表会被创建为临时表。

ALTER/DROP 全局临时表和索引，如果其他会话正在使用它，禁止操作。

全局临时表的 DDL 只会影响当前会话的用户数据和索引。例如 truncate、reindex、analyze 只对当前会话有效。

【说明】

SELECT INTO 的其他参数可以参考：SQL 语法参考->SQL 语法->SELECT 章节中的参数说明。

示例

前置操作：

1、创建表 reason_t2。

```
CREATE TABLE reason_t2
(
 r_reason_sk    integer,
 r_reason_id    character(16),
 r_reason_desc  character(100)
);
```

2、向表中插入一条记录并查询。

```
INSERT INTO reason_t2(r_reason_sk, r_reason_id, r_reason_desc) VALUES (1, 'AAAAAAAABAAAAAAA', 'reason1');
INSERT INTO reason_t2 VALUES (2, 'AAAAAAAABAAAAAAA', 'reason2');
INSERT INTO reason_t2 VALUES (3, 'AAAAAAAACAAAAAAA','reason3'),(4, 'AAAAAAAADAAAAAAA', 'reason4'),(5, 'AAAAAAAAEAAAAAAA','reason5');
```

示例：

1、将 reason 表中 r_reason_sk 小于 5 的值加入到新建表中。

```
SELECT * INTO reason_t1 FROM reason_t2 WHERE r_reason_sk < 5;
```

2、查询 reason_t1 表的内容。

```
select * from reason_t1;
```

查询结果显示为：

```
 r_reason_sk |  r_reason_id    |                    r_reason_desc
-------------+------------------+-------------------------------------------------
--------------------------------------------------
           1 | AAAAAAAABAAAAAAA | reason1
           2 | AAAAAAAABAAAAAAA | reason2
           3 | AAAAAAAACAAAAAAA | reason3
           4 | AAAAAAAADAAAAAAA | reason4
(4 rows)
```

3、删除 reason_t1 表。

```
DROP TABLE reason_t1;
```

相关链接

参考：SQL 语法参考->SQL 语法->SELECT 章节。

4.5.181.SET

功能描述

用于修改运行时配置参数。

注意事项

大多数运行时参数都可以用 SET 在运行时设置，但有些则在服务运行过程中或会话开始之后不能修改。

语法格式

- 设置所处的时区。

```
SET [ SESSION | LOCAL ] TIME ZONE { timezone | LOCAL | DEFAULT };
```

- 设置所属的模式。

```
SET [ SESSION | LOCAL ]
    {CURRENT_SCHEMA { TO | = } { schema | DEFAULT }
    | SCHEMA 'schema'};
```

- 设置客户端编码集。

```
SET [ SESSION | LOCAL ] NAMES encoding_name;
```

- 设置 XML 的解析方式。

```
SET [ SESSION | LOCAL ] XML OPTION { DOCUMENT | CONTENT };
```

- 设置其他运行时参数。

```
SET [ LOCAL | SESSION ]
    {{config_parameter { { TO | = } { value | DEFAULT }
                | FROM CURRENT }}};
```

- 设置运行时的参数

```
SET { GLOBAL | @@GLOBAL.}
{ {config_parameter = { expr | DEFAULT }}};
SET [ SESSION | @@SESSION. | @@]
{ {config_parameter = { expr | DEFAULT }}};
```

参数说明

- **SESSION**

  声明的参数只对当前会话起作用。如果 SESSION 和 LOCAL 都没出现，则 SESSION 为缺省值。

  如果在事务中执行了此命令，命令的产生影响将在事务回滚之后消失。如果该事务已提交，影响将持续到会话的结束，除非被另外一个 SET 命令重置参数。

- **LOCAL**

  声明的参数只在当前事务中有效。在 COMMIT 或 ROLLBACK 之后，会话级别的设置将再次生效。

  不论事务是否提交，此命令的影响只持续到当前事务结束。一个特例是：在一个事务里面，即有 SET 命令，又有 SET LOCAL 命令，且 SET LOCAL 在 SET 后面，则在事务结束之前，SET LOCAL 命令会起作用，但事务提交之后，则是 SET 命令会生效。

- **TIME ZONE timezone**

  用于指定当前会话的本地时区。

  取值范围：有效的本地时区。该选项对应的运行时参数名称为 TimeZone，DEFAULT 缺省值为 PRC。

- **CURRENT_SCHEMA**

  schema

  CURRENT_SCHEMA 用于指定当前的模式。

  取值范围：已存在模式名称。如果模式名不存在，会导致 CURRENT_SCHEMA 值为空。

- **SCHEMA schema**

  同 CURRENT_SCHEMA。此处的 schema 是个字符串。

  例如：set schema 'public'。

- **NAMES encoding_name**

  用于设置客户端的字符编码。等价于 set client_encoding to encoding_name。

取值范围：有效的字符编码。该选项对应的运行时参数名称为 client_encoding，默认编码为 UTF8。

- **XML OPTION option**

  用于设置 XML 的解析方式。

  取值范围：CONTENT（缺省）、DOCUMENT。

- **config_parameter**

  可设置的运行时参数的名称。可用的运行时参数可以使用 SHOW ALL 命令查看。

说明

部分通过 SHOW ALL 查看的参数不能通过 SET 设置。如 max_datanodes。

- **value**

  config_parameter 的新值。可以声明为字符串常量、标识符、数字，或者逗号分隔的列表。DEFAULT 用于把这些参数设置为它们的缺省值。

- **GLOBAL | @@GLOBAL.**

  声明的参数生效范围为 postmaster、sighup、backend，可通过 pg_settings 系统视图的 context 字段确定。设置参数范围和生效方式与 ALTER SYSTEM SET 语法相同。支持 config_parameter 赋值为表达式。

- **SESSION | @@SESSION. | @@**

  声明的参数生效方式为 superuser、user，可通过 pg_settings 系统视图的 context 字段确定，如果没有出现 GLOBAL /SESSION，则 SESSION 为缺省值。支持 config_parameter 赋值为表达式。

  【说明】

  - SET SESSION/GLOBAL 语法只有在 B 模式下（sql_compatibility = B）支持，并且 GUC 参数 enable_set_variable_b_format 打开的场景下才支持（enable_set_variable_b_format = on)。
  - 使用@@config_parameter 进行操作符运算时，尽量使用空格隔开。比如 set @config_parameter1=@config_parameter1*2; 命令中，

会将=@当做操作符，可将其修改为 set @config_parameter1= @config_parameter1 * 2 。

示例

1、设置模式搜索路径。

```
SET search_path TO tpcds, public;
```

2、把日期时间风格设置为传统的 POSTGRES 风格(日在月前)。

```
SET datestyle TO postgres,dmy;
```

3、Global 变量设置：

```
set global most_available_sync = t;
set @@global.most_available_sync = t;
```

4、Session 变量设置：

```
set @@codegen_cost_threshold = 10000;
set @@session.codegen_cost_threshold = @@codegen_cost_threshold * 2;
```

相关链接

参考：SQL 语法参考->SQL 语法->RESET、SHOW 章节。

4.5.182.SET CONSTRAINTS

功能描述

SET CONSTRAINTS 设置当前事务检查行为的约束条件。

IMMEDIATE 约束是在每条语句后面进行检查。DEFERRED 约束一直到事务提交时才检查。每个约束都有自己的模式。

从创建约束条件开始，一个约束总是设定为 DEFERRABLE INITIALLY DEFERRED、DEFERRABLE INITIALLY IMMEDIATE、NOT DEFERRABLE 三个特性之一。第三种总是 IMMEDIATE，并且不会受 SET CONSTRAINTS 影响。前两种以指定的方式启动每个事务，但是其行为可以在事务里用 SET CONSTRAINTS 改变。

带着一个约束名列表的 SET CONSTRAINTS 改变这些约束的模式（都必须是可推迟的）。如果有多个约束匹配某个名称，则所有都会被影响。SET CONSTRAINTS ALL 改变所有可推迟约束的模式。

当 SET CONSTRAINTS 把一个约束从 DEFERRED 改成 IMMEDIATE 的时候，新模式反作用式地起作用：任何将在事务结束准备进行的数据修改都将在 SET CONSTRAINTS 的时候执行检查。如果违反了任何约束，SET CONSTRAINTS 都会失败（并且不会修改约束模式）。因此，SET CONSTRAINTS 可以用于强制在事务中某一点进行约束检查。

检查约束总是不可推迟的。

注意事项

SET CONSTRAINTS 只在当前事务里设置约束的行为。因此，如果用户在事务块（START TRANSACTION/COMMIT 对）之外执行这个命令，它将没有任何作用。

语法格式

```
SET CONSTRAINTS { ALL | { name } [, ...] } { DEFERRED | IMMEDIATE };
```

参数说明

- **name**

  约束名。

  取值范围：已存在的约束名。可以在系统表 pg_constraint 中查到。

- **ALL**

  所有约束。

- **DEFERRED**

  约束一直到事务提交时才检查。

- **IMMEDIATE**

  约束在每条语句后进行检查。

示例

```
--设置所有约束在事务提交时检查。
panweidb=# SET CONSTRAINTS ALL DEFERRED;
```

4.5.183.SET ROLE

功能描述

设置当前会话的当前用户标识符。

注意事项

- 当前会话的用户必须是指定的 rolename 角色的成员，但系统管理员可以选择任何角色。
- 使用这条命令，它可能会增加一个用户的权限，也可能会限制一个用户的权限。如果会话用户的角色有 INHERITS 属性，则它自动拥有它能 SET ROLE 变成的角色的所有权限；在这种情况下，SET ROLE 实际上是删除了所有直接赋予会话用户的权限，以及它的所属角色的权限，只剩下指定角色的权限。另一方面，如果会话用户的角色有 NOINHERITS 属性，SET ROLE 删除直接赋予会话用户的权限，而获取指定角色的权限。

语法格式

- 设置当前会话的当前用户标识符。

```
SET [ SESSION | LOCAL ] ROLE role_name PASSWORD 'password';
```

- 重置当前用户标识为当前会话用户标识符。

```
RESET ROLE;
```

参数说明

- **SESSION**

  声明这个命令只对当前会话起作用，此参数为缺省值。

- **LOCAL**

  声明该命令只在当前事务中有效。

- **role_name**

  角色名。

  取值范围：字符串，要符合标识符的命名规范。

- **password**

  角色的密码。要求符合密码的命名规则。

- **RESET ROLE**

  用于重置当前用户标识。

示例

```
--创建角色 paul。
panweidb=# CREATE ROLE paul IDENTIFIED BY 'xxxxxxxxx';

--设置当前用户为 paul。
panweidb=# SET ROLE paul PASSWORD 'xxxxxxxxx';

--查看当前会话用户，当前用户。
panweidb=# SELECT SESSION_USER, CURRENT_USER;

--重置当前用户。
panweidb=# RESET role;

--删除用户。
panweidb=# DROP USER paul;
```

4.5.184.SET SESSION AUTHORIZATION

功能描述

把当前会话里的会话用户标识和当前用户标识都设置为指定的用户。

注意事项

只有在初始会话用户有系统管理员权限的时候，会话用户标识符才能改变。否则，只有在指定了被认证的用户名的情况下，系统才接受该命令。

语法格式

❖ 为当前会话设置会话用户标识符和当前用户标识符。

```
SET [ SESSION | LOCAL ] SESSION AUTHORIZATION role_name PASSWORD 'password';
```

❖ 重置会话和当前用户标识符为初始认证的用户名。

```
{SET [ SESSION | LOCAL ] SESSION AUTHORIZATION DEFAULT
| RESET SESSION AUTHORIZATION};
```

参数说明

❖ **SESSION**

声明这个命令只对当前会话起作用。

- **LOCAL**

  声明该命令只在当前事务中有效。

- **role_name**

  用户名。

  取值范围：字符串，要符合标识符的命名规范。

- **password**

  角色的密码。要求符合密码的命名规则。

- **DEFAULT**

  重置会话和当前用户标识符为初始认证的用户名。

示例

```
--创建角色 paul。
panweidb=# CREATE ROLE paul IDENTIFIED BY 'xxxxxxxxx';

--设置当前用户为 paul。
panweidb=# SET SESSION AUTHORIZATION paul password 'xxxxxxxxx';

--查看当前会话用户，当前用户。
panweidb=# SELECT SESSION_USER, CURRENT_USER;

--重置当前用户。
panweidb=# RESET SESSION AUTHORIZATION;

--删除用户。
panweidb=# DROP USER paul;
```

相关参考

参考：SQL 语法参考->SQL 语法->SET ROLE 章节。

4.5.185.SET TRANSACTION

功能描述

为事务设置特性。事务特性包括事务隔离级别、事务访问模式（读/写或者只读）。可以设置当前事务的特性（LOCAL)，也可以设置会话的默认事务特性（SESSION）。

语法格式

设置事务的隔离级别、读写模式。

```
{ SET [ LOCAL ] TRANSACTION|SET SESSION CHARACTERISTICS AS TRANSACTION }
 { ISOLATION LEVEL { READ COMMITTED | READ UNCOMMITTED | SERIALIZABLE | REPE
ATABLE READ  }
 { READ WRITE | READ ONLY } } [, ...]
```

参数说明

- LOCAL：声明该命令只在当前事务中有效。
- SESSION：声明这个命令只对当前会话起作用。

  取值范围：字符串，要符合标识符的命名规范。
- ISOLATION_LEVEL：指定事务隔离级别，该参数决定当一个事务中存在其他并发运行事务时能够看到什么数据。

  在事务中第一个数据修改语句（SELECT、INSERT、DELETE、UPDATE、FETCH、COPY)执行之后，当前事务的隔离级别就不能再次设置。

  取值范围：
  - READ COMMITTED：读已提交隔离级别，只能读到已经提交的数据，而不会读到未提交的数据。这是缺省值。
  - READ UNCOMMITTED：读未提交隔离级别，也就是说事务所作的修改在未提交前，其他并发事务是可以读到的。
  - REPEATABLE READ：可重复读隔离级别，仅仅能看到事务开始之前提交的数据，不能看到未提交的数据，以及在事务执行期间由其他并发事务提交的修改。
  - SERIALIZABLE：PanWeiDB 目前功能上不支持此隔离级别，等价于 REPEATABLE READ。
- READ WRITE | READ ONLY：指定事务访问模式（读/写或者只读）。

注意事项

设置当前事务特性需要在事务中执行（即执行 SET TRANSACTION 之前需要执行 START TRANSACTION 或者 BEGIN），否则设置不生效。

示例

开启一个事务，设置事务的隔离级别为 READ COMMITTED，访问模式为 READ ONLY。

```
START TRANSACTION;
SET LOCAL TRANSACTION ISOLATION LEVEL READ COMMITTED READ ONLY;
COMMIT;
```

4.5.186.SHOW

功能描述

SHOW 将显示当前运行时参数的数值。

注意事项

无。

语法格式

```
SHOW
{
 configuration_parameter |
 VARIABLES LIKE var_name|
 CURRENT_SCHEMA |
 TIME ZONE |
 TRANSACTION ISOLATION LEVEL |
 SESSION AUTHORIZATION |
 WARNINGS [LIMIT [offset,] row_count] | ERRORS [LIMIT [offset,] row_count] |
 ALL
```

```
};
```

【须知】

WARNINGS [LIMIT [offset,] row_count] | ERRORS [LIMIT [offset,] row_count] 语法仅在 MySQL 兼容模式下可用，更多内容可参考 MySQL 兼容性手册下的 SHOW WARNINGS/ERRORS。

参数说明

显示变量的参数请参见《PanWeiDB V2.0-S3.0.1_B01 开发者指南》->RESET 的参数说明。

示例

1、显示 timezone 参数值。

```
SHOW timezone;
```

2、显示所有参数。

```
SHOW ALL;
```

3、显示参数名中包含"var"的所有参数。

```
SHOW VARIABLES LIKE var;
```

相关链接

SET，RESET

4.5.187.SHUTDOWN

功能描述

SHUTDOWN 将关闭当前连接的数据库节点。

注意事项

仅拥有管理员权限的用户可以运行此命令。

语法格式

```
SHUTDOWN [ SMART | FAST | IMMEDIATE ]
```

参数说明

- **smart**

  不接受新的连接，且等待已有连接全部断开后，关闭数据库节点。

- **fast**

  不等待客户端中断连接，将所有活跃事务回滚并且强制断开客户端，然后关闭数据库节点。

- **immediate**

  强行关闭，在下次重新启动的时候将导致故障恢复。

示例

```
--关闭当前数据库节点。
panweidb=# SHUTDOWN;

--使用 fast 模式关闭当前数据库节点。
panweidb=# SHUTDOWN FAST;
```

4.5.188.SNAPSHOT

功能描述

针对多用户情况下，对数据进行统一的版本控制。

注意事项

- 当快照选用增量存储方式时，各个快照中具有依赖关系。删除快照需要按照依赖顺序进行删除。
- snapshot 特性用于团队不同成员间维护数据，涉及管理员和普通用户之间的数据转写。所以在私有用户、三权分立(enableSeparationOfDuty=ON)等状态下，数据库不支持 snapshot 功能特性。
- 当需要稳定可用的快照用于 AI 训练等任务时，用户需要将快照发布。
- 本特性相关 GUC 参数有（以下参数详见：开发者指南->GUC 参数说明->AI 特性章节）：

- db4ai_snapshot_mode：快照存储模型分为 MSS 和 CSS 两种
- db4ai_snapshot_version_delimiter：用于设定版本分隔符，仅接受设定单字节参数值，默认为“@”
- db4ai_snapshot_version_separator：用于设定子版本分隔符，仅接受设定单字节参数值，默认为“.”。

语法格式

1、创建快照。

可以采用“CREATE SNAPSHOT … AS”以及“CREATE SNAPSHOT … FROM”语句创建数据表快照。

❖ CREATE SNAPSHOT AS

```
CREATE SNAPSHOT <qualified_name> [@ <version | ident | sconst>]
  [COMMENT IS <sconst>}
  AS query;
```

❖ CREATE SNAPSHOT FROM

```
CREATE SNAPSHOT <qualified_name> [@ <version | ident | sconst>]
  FROM @ <version | ident | sconst>
  [COMMENT IS <sconst>}
    USING (
    { INSERT [INTO SNAPSHOT] …
     | UPDATE [SNAPSHOT] [AS <alias>] SET … [FROM …] [WHERE …]
     | DELETE [FROM SNAPSHOT] [AS <alias>] [USING …] [WHERE …]
     | ALTER [SNAPSHOT] { ADD … | DROP … } [, …]
    } [; …]
  );
```

2、删除快照。

PURGE SNAPSHOT

```
PURGE SNAPSHOT <qualified_name> @ <version | ident | sconst>;
```

3、快照采样。

SAMPLE SNAPSHOT

```
SAMPLE SNAPSHOT <qualified_name> @ <version | ident | sconst>
  [STRATIFY BY attr_list]
  { AS <label> AT RATIO <num> [COMMENT IS <comment>] } [, ...]
```

4、快照发布。

PUBLISH SNAPSHOT

```
PUBLISH SNAPSHOT <qualified_name> @ <version | ident | sconst>;
```

5、快照存档。

ARCHIVE SNAPSHOT

```
ARCHIVE SNAPSHOT <qualified_name> @ <version | ident | sconst>;
```

参数说明

- qualified_name

  创建 snapshot 的名称。

  取值范围：字符串，需要符合标识符命名规则。

- version

  (可省略)snapshot 的版本号，当省略设置。系统会自动顺延编号。

  取值范围：字符串，数字编号配合分隔符。

示例

1、创建测试表。

```
create table t1(id int,name text);
```

2、创建新快照 s1。

```
create snapshot s1@1.0 comment is 'first version' as select * from t1;
```

3、从现有快照创建快照修订。

```
create snapshot s1@3.0 from @1.0 comment is 'inherits from @1.0' using (INSERT V
ALUES(6,6), (7,7); DELETE WHERE id = 1);
```

返回结果为：

```
schema |  name
```

```
--------+--------
 public | s1@3.0
(1 row)
```

4、删除一个快照。

```
purge snapshot s1@1.0;
```

5、快照采样。

```
sample snapshot s1@3.0 stratify by id as nick at ratio .5;
```

返回结果为：

```
 schema |    name
--------+------------
 public | s1nick@3.0
(1 row)
```

6、发布一个快照。

```
publish snapshot s1@3.0;
```

返回结果为：

```
 schema |  name
--------+--------
 public | s1@3.0
(1 row)
```

7、存档快照。

```
archive snapshot s1@3.0;
```

相关链接

无。

4.5.189.START TRANSACTION

功能描述

通过 START TRANSACTION 启动事务。如果声明了隔离级别、读写模式，那么新事务就使用这些特性，类似执行了 SET TRANSACTION。

语法格式

格式一：START TRANSACTION 格式

```
START TRANSACTION
  [ { ISOLATION LEVEL { READ COMMITTED | READ UNCOMMITTED | SERIALIZABLE | REP
EATABLE READ  }
  | { READ WRITE | READ ONLY }
  } [, ...] ];
```

格式二：BEGIN 格式

```
BEGIN [ WORK | TRANSACTION ]
 [ { ISOLATION LEVEL { READ COMMITTED | READ UNCOMMITTED | SERIALIZABLE | REPE
ATABLE READ }
 | { READ WRITE | READ ONLY }
 } [, ...] ];
```

参数说明

- WORK | TRANSACTIONBEGIN：格式中的可选关键字，没有实际作用。
- ISOLATION LEVEL：指定事务隔离级别，它决定当一个事务中存在其他并发运行事务时它能够看到什么数据。

  在事务中第一个数据修改语句（SELECT、 INSERT、DELETE、UPDATE、FETCH、COPY）执行之后，事务隔离级别就不能再次设置。

  取值范围：

  - READ COMMITTED：读已提交隔离级别，只能读到已经提交的数据，而不会读到未提交的数据。这是缺省值。
  - READ UNCOMMITTED：读未提交隔离级别，也就是说事务所作的修改在未提交前，其他并发事务是可以读到的。

- REPEATABLE READ：可重复读隔离级别，仅仅看到事务开始之前提交的数据，它不能看到未提交的数据，以及在事务执行期间由其他并发事务提交的修改。
- SERIALIZABLE：PanWeiDB 目前功能上不支持此隔离级别，等价于 REPEATABLE READ。

❖ READ WRITE | READ ONLY：指定事务访问模式（读/写或者只读）。

注意事项

无。

示例

1、创建测试表 test。

```
CREATE TABLE test(id int);
```

2、以不同方式启动事务

❖ 以默认方式（START TRANSACTION）启动事务，并在操作后提交。

```
START TRANSACTION;
SELECT * FROM test;
END;
```

❖ 以默认方式（BEGIN）启动事务，并在操作后提交。

```
BEGIN;
SELECT * FROM test;
END;
```

❖ 以隔离级别为 READ COMMITTED，读/写方式启动事务，并在操作后提交。

```
START TRANSACTION ISOLATION LEVEL READ COMMITTED READ WRITE;
SELECT * FROM test;
COMMIT;
```

3、删除测试表

```
DROP TABLE test;
```

4.5.190.TIMECAPSULE TABLE

功能描述

在人为操作或应用程序错误时，使用 TIMECAPSULE TABLE 语句恢复可将表恢复到一个早期状态。

表可以闪回到过去的时间点，这依赖于系统中保存的旧版本数据。此外 PanWeiDB 数据库不能恢复到通过 DDL 操作改变了表结构的早期状态。

注意事项

- TIMECAPSULE TABLE 语句的用法：从回收站中闪回。
  - 回收站记录了 DROP 和 TRUNCATE 的对象数据。TO BEFORE DROP 和 TO BEFORE TRUNCATE 就是从回收站中闪回。
- 不支持闪回表的对象类型：系统表、列存表、内存表、DFS 表、全局临时表、本地临时表、UNLOGGED 表、序列表、hashbucket 表。
- 闪回点和当前点之间，执行过修改表结构或影响物理存储的语句（DDL、DCL、VACUUM FULL），闪回失败。
- 执行闪回删除需要用户具有如下权限：用户必须具有垃圾对象所在 schema 的 create 和 usage 权限，并且用户必须是 schema 的所有者或者是垃圾对象的所有者。
- 执行闪回 TRUNCATE 需要用户具有如下权限：用户必须具有垃圾对象所在 schema 的 create 和 usage 权限，并且用户必须是 schema 的所有者或者是垃圾对象的所有者，另外用户必须具有垃圾对象的 TRUNCATE 权限。
- 不适用闪回 drop/truncate 功能的场景或表：
  - 回收站关闭场景：enable_recyclebin = off；
  - 系统处于维护态（xc_maintenance_mode = on）或升级场景；
  - 多对象删除场景：DROP/TRUNCATE TABLE 命令同时指定多个对象；
  - 系统表、列存表、内存表、DFS 表、全局临时表、本地临时表、UNLOGGED 表、序列表、hashbucket 表。

语法格式

```
TIMECAPSULE TABLE [ schema.]table_name TO {BEFORE { DROP [RENAME TO table_name] | TRUNCATE } }
```

参数说明

- **schema_name**

  指定模式包含的表。如果缺省，则为当前模式。

- **table_name**

  指定表名。

- **TO BEFORE DROP**

  使用这个子句检索回收站中已删除的表及其子对象。

  你可以指定原始用户指定的表的名称，或对象删除时数据库分配的系统生成名称。

  - 回收站中系统生成的对象名称是唯一的。因此，如果指定系统生成名称，那么数据库检索指定的对象。使用“select * from gs_recyclebin;”语句查看回收站中的内容。
  - 如果指定了用户指定的名称，且如果回收站中包含多个该名称的对象，然后数据库检索回收站中最近移动的对象。如果想要检索更早版本的表，你可以这样做：
    - 指定你想要检索的表的系统生成名称。
    - 执行 TIMECAPSULE TABLE ... TO BEFORE DROP 语句，直到你要检索的表。
  - 恢复 DROP 表时，只恢复基表名，其他子对象名均保持回收站对象名。用户可根据需要，执行 DDL 命令手工调整子对象名。
  - 回收站对象不支持 DML、DCL、DDL 等写操作，不支持 DQL 查询操作（后续支持）。
  - recyclebin_retention_time 配置参数用于设置回收站对象保留时间，超过该时间的回收站对象将被自动清理。

- **RENAME TO**

  为从回收站中检索的表指定一个新名称。

- **TRUNCATE**

闪回到 TRUNCATE 之前。

示例

```
--当前数据库已存在 tpcds 模式
-- 删除表 tpcds.reason_t2
DROP TABLE IF EXISTS tpcds.reason_t2;
-- 创建表 tpcds.reason_t2
panweidb=# CREATE TABLE tpcds.reason_t2
(
 r_reason_sk   integer,
 r_reason_id   character(16),
 r_reason_desc character(100)
 );

--向表 tpcds.reason_t2 中插入记录
panweidb=# INSERT INTO tpcds.reason_t2 VALUES (1, 'AA', 'reason1'),(2, 'AB', 'reas
on2'),(3, 'AC', 'reason3');
INSERT 0 3

--清空 tpcds.reason_t2 表中的数据
panweidb=# TRUNCATE TABLE tpcds.reason_t2;

--查询 tpcds.reason_t2 表中的数据
panweidb=# select * from tpcds.reason_t2;
 r_reason_sk | r_reason_id | r_reason_desc
-------------+-------------+---------------
(0 rows)

--执行闪回 TRUNCATE
panweidb=# TIMECAPSULE TABLE tpcds.reason_t2 to BEFORE TRUNCATE;

panweidb=# select * from tpcds.reason_t2;
 r_reason_sk |  r_reason_id   |                          r_reason_desc
```

```
------------+-----------------+-------------------------------------------------------
-------------------------------------------
      1 | AA           | reason1
      2 | AB           | reason2
      3 | AC           | reason3
(3 rows)
--删除表 tpcds.reason_t2
panweidb=# DROP TABLE tpcds.reason_t2;

--执行闪回 DROP
panweidb=# TIMECAPSULE TABLE tpcds.reason_t2 to BEFORE DROP;
TimeCapsule Table
```

4.5.191.TRUNCATE

功能描述

清理表数据，TRUNCATE 快速地从表中删除所有行。

它和在目标表上进行无条件的 DELETE 有同样的效果，但由于 TRUNCATE 不做表扫描，因而快得多。在大表上操作效果更明显。

注意事项

- TRUNCATE TABLE 在功能上与不带 WHERE 子句 DELETE 语句相同：二者均删除表中的全部行。
- TRUNCATE TABLE 比 DELETE 速度快且使用系统和事务日志资源少：
  - DELETE 语句每次删除一行，并在事务日志中为所删除每行记录一项。
  - TRUNCATE TABLE 通过释放存储表数据所用数据页来删除数据，并且只在事务日志中记录页的释放。
- TRUNCATE、DELETE、DROP 三者的差异如下：
  - TRUNCATE TABLE，删除内容，释放空间，但不删除定义。
  - DELETE TABLE，删除内容，不删除定义，不释放空间。
  - DROP TABLE，删除内容和定义，释放空间。

语法格式

❖ 清理表数据。

```
TRUNCATE [ TABLE ] [ ONLY ] {table_name [ * ]} [, ... ]
  [ CONTINUE IDENTITY ] [ CASCADE | RESTRICT][PURGE]};
```

❖ 清理表分区的数据。

```
ALTER TABLE [ IF EXISTS  ] { [ ONLY  ] table_name
                  | table_name *
                  | ONLY ( table_name ) }
  TRUNCATE PARTITION { partition_name
              | FOR (  partition_value  [, ...] ) } [ UPDATE GLOBAL INDEX ];
```

参数说明

❖ **ONLY**

如果声明 ONLY，只有指定的表会被清空。如果没有声明 ONLY，这个表以及其所有子表（若有）会被清空。

❖ **table_name**

目标表的名称（可以有模式修饰）。

取值范围：已存在的表名。

❖ **CONTINUE IDENTITY**

不改变序列的值。这是缺省值。

❖ **CASCADE | RESTRICT**

- CASCADE：级联清空所有由于 CASCADE 而被添加到组中的表。
- RESTRICT（缺省值）：完全清空。

❖ **PURGE**：默认将表数据放入回收站中，PURGE 直接清理。

❖ **partition_name**

目标分区表的分区名。

取值范围：已存在的分区名。

❖ **partition_value**

指定的分区键值。

通过 PARTITION FOR 子句指定的这一组值，可以唯一确定一个分区。

取值范围：需要进行删除数据分区的分区键的取值范围。

⚠ **须知：**

使用 PARTITION FOR 子句时，partition_value 所在的整个分区会被清空。

❖ **UPDATE GLOBAL INDEX**

如果使用该参数，则会更新分区表上的所有全局索引，以确保使用全局索引可以查询出正确的数据；如果不使用该参数，则分区表上的所有全局索引将会失效。

示例

```
--创建表。
panweidb=# CREATE TABLE tpcds.reason_t1 AS TABLE tpcds.reason;

--清空表 tpcds.reason_t1。
panweidb=# TRUNCATE TABLE tpcds.reason_t1;

--删除表。
panweidb=# DROP TABLE tpcds.reason_t1;
--创建分区表。
panweidb=# CREATE TABLE tpcds.reason_p
(
 r_reason_sk integer,
 r_reason_id character(16),
 r_reason_desc character(100)
)PARTITION BY RANGE (r_reason_sk)
(
 partition p_05_before values less than (05),
 partition p_15 values less than (15),
 partition p_25 values less than (25),
 partition p_35 values less than (35),
 partition p_45_after values less than (MAXVALUE)
);
```

```
--插入数据。
panweidb=# INSERT INTO tpcds.reason_p SELECT * FROM tpcds.reason;

--清空分区 p_05_before。
panweidb=# ALTER TABLE tpcds.reason_p TRUNCATE PARTITION p_05_before;

--清空分区 p_15。
panweidb=# ALTER TABLE tpcds.reason_p TRUNCATE PARTITION for (13);

--清空分区表。
panweidb=# TRUNCATE TABLE tpcds.reason_p;

--删除表。
panweidb=# DROP TABLE tpcds.reason_p;
```

4.5.192.UPDATE

功能描述

更新表中的数据。UPDATE 修改满足条件的所有行中指定的字段值，WHERE 子句声明条件，SET 子句指定的字段会被修改，没有出现的字段则保持它们的原值。

注意事项

- 表的所有者、拥有表 UPDATE 权限的用户或拥有 UPDATE ANY TABLE 权限的用户，有权更新表中的数据，系统管理员默认拥有此权限。
- 对 expression 或 condition 条件里涉及到的任何表要有 SELECT 权限。
- 对于列存表，暂时不支持 RETURNING 子句。
- 列存表不支持结果不确定的更新(non-deterministic update)。试图对列存表用多行数据更新一行时会报错。
- 列存表的更新操作，旧记录空间不会回收，需要执行 VACUUM FULL table_name 进行清理。
- 对于列存复制表，暂不支持 UPDATE 操作。

- 生成列不能被直接写入。在 UPDATE 命令中不能为生成列指定值，但是可以指定关键字 DEFAULT。
- 对于多表更新，仅在参数 sql_compatibility=B 时生效，暂时不支持对列存表、视图和含有 RULE 的表进行多表更新。

语法格式

```
[ WITH [ RECURSIVE ] with_query [, ...] ]
UPDATE [ ONLY ] table_name [ @dblink_name ] [ partition_clause ] [ * ] [ [ AS ] alias ]
SET {column_name = { expression | DEFAULT }
  |( column_name [, ...] ) = {( { expression | DEFAULT } [, ...] ) |sub_query }}[, ...]
  [ FROM from_list] [ WHERE condition | WHERE CURRENT OF cursor_name ]
  [ ORDER BY {expression [ [ ASC | DESC | USING operator ]
  [ LIMIT { count } ]
  [ RETURNING {* | {output_expression [ [ AS ] output_name ]} [, ...] }];
```

多表更新：

```
[ WITH [ RECURSIVE ] with_query [, ...] ]
UPDATE [/*+ plan_hint */] table_list
SET {column_name = { expression | DEFAULT }
  |( column_name [, ...] ) = {( { expression | DEFAULT } [, ...] ) |sub_query }}[, ...]
  [ FROM from_list] [ WHERE condition | WHERE CURRENT OF cursor_name ];
```

其中 sub_query 为：

```
SELECT [ ALL | DISTINCT [ ON ( expression [, ...] ) ] ]
{ * | {expression [ [ AS ] output_name ]} [, ...] }
[ FROM from_item [, ...] ]
[ WHERE condition ]
[ GROUP BY grouping_element [, ...] ]
[ HAVING condition [, ...] ]
```

其中 partition_clause 为：

【说明】

'partition_clause'选项仅在集中式模式下可用。

```
PARTITION { ( partition_name ) | FOR ( partition_value [, ...] ) } |
```

```
SUBPARTITION { ( subpartition_name ) | FOR ( subpartition_value [, ...] ) }
```

参数说明

- **WITH [RECURSIVE] with_query [, ...]**

  用于声明一个或多个可以在主查询中通过名称引用的子查询，相当于临时表。

  如果声明了 RECURSIVE，那么允许 SELECT 子查询通过名称引用它自己。

  其中 with_query 的详细格式为：with_query_name [(column_name [, ...])] AS [[NOT] MATERIALIZED] ({select | values | insert | update | delete})

  - with_query_name 指定子查询生成的结果集名称，在查询中可使用该名称访问子查询的结果集。
  - column_name 指定子查询结果集中显示的列名。
  - 每个子查询可以是 SELECT，VALUES，INSERT，UPDATE 或 DELETE 语句。
  - 用户可以使用 MATERIALIZED / NOT MATERIALIZED 对 CTE 进行修饰。
    - 如果声明为 MATERIALIZED，WITH 查询将被物化，生成一个子查询结果集的拷贝，在引用处直接查询该拷贝，因此 WITH 子查询无法和主干 SELECT 语句进行联合优化（如谓词下推、等价类传递等），对于此类场景可以使用 NOT MATERIALIZED 进行修饰，如果 WITH 查询语义上可以作为子查询内联执行，则可以进行上述优化。
    - 如果用户没有显示声明物化属性则遵守以下规则：如果 CTE 只在所属 SELECT 主干中被引用一次，且语义上支持内联执行，则会被改写为子查询内联执行，否则以 CTE Scan 的方式物化执行。

- **plan_hint 子句**

  以/*+ /的形式在 UPDATE 关键字后，用于对 UPDATE 对应的语句块生成的计划进行 hint 调优，详细用法请参见：《PanWeiDB V2.0-S3.0.1_B01 管理员指南->性能调优->SQL 调优指南->使用 Plan Hint 进行调优》

章节。每条语句中只有第一个/+ plan_hint */注释块会作为 hint 生效，里面可以写多条 hint。

- **table_name**

  要更新的表名，可以使用模式修饰。

  取值范围：已存在的表名称。

- **partition_clause**

  指定分区更新操作

  ```
  PARTITION { ( partition_name ) | FOR ( partition_value [, ...] ) } |
  SUBPARTITION { ( subpartition_name ) | FOR ( subpartition_value [, ...] ) }
  ```

  关键字详见：SQL 语法参考->SQL 语法->SELECT 章节。

  示例详见：SQL 语法参考->SQL 语法->CREATE TABLE SUBPARTITION 章节。

- **alias**

  目标表的别名。

  取值范围：字符串，符合标识符命名规范。

- **table_list**

  一个表的表达式列表，与 from_list 类似，但可以同时声明目标表和关联表，仅在多表更新语法中使用。

- **column_name**

  要修改的字段名。

  支持使用目标表的别名加字段名来引用这个字段。例如：

  ```
  UPDATE foo AS f SET f.col_name = 'namecol';
  ```

  取值范围：已存在的字段名。

- **expression**

  赋给字段的值或表达式。

- **DEFAULT**

  用对应字段的缺省值填充该字段。

  如果没有缺省值，则为 NULL。

- **sub_query**

子查询。

使用同一数据库里其他表的信息来更新一个表可以使用子查询的方法。其中 SELECT 子句具体介绍请参考：SQL 语法参考->SQL 语法->SELECT 章节。

在 update 单列时，支持使用 order by 子句与 limit 子句；而在 update 多列时，则不支持使用 order by 子句与 limit 子句。

- **from_list**

  一个表的表达式列表，允许在 WHERE 条件里使用其他表的字段。与在一个 SELECT 语句的 FROM 子句里声明表列表类似。

【说明】

目标表绝对不能出现在 from_list 里，除非在使用一个自连接（此时它必须以 from_list 的别名出现）。

- **condition**

  一个返回 Boolean 类型结果的表达式。只有这个表达式返回 true 的行才会被更新。不建议使用 int 等数值类型作为 condition，因为 int 等数值类型可以隐式转换为 bool 值（非 0 值隐式转换为 true，0 转换为 false），可能导致非预期的结果。

- **ORDER BY 子句**

  关键字详见 SELECT 一节介绍。

- **LIMIT 子句**

  关键字详见 SELECT 一节介绍。

- **RETURNING**

  返回实际删除的行，RETURNING 列表的语法与 SELECT 的输出列表一致。

```
[ RETURNING {* | {output_expression [ [ AS ] output_name ] }[, ...]} ]
```

- **output_expression**

  在所有需要更新的行都被更新之后，UPDATE 命令用于计算返回值的表达式。

  取值范围：使用任何 table 以及 FROM 中列出的表的字段。*表示返回所有字段。

❖ **output_name**

字段的返回名称。

示例

1、创建表 student1。

```
CREATE TABLE student1
(
  stuno    int,
  classno  int
);
```

2、插入数据。

```
INSERT INTO student1 VALUES(1,1);
INSERT INTO student1 VALUES(2,2);
INSERT INTO student1 VALUES(3,3);
```

3、查看数据。

```
SELECT * FROM student1;
```

查询结果显示为：

```
 stuno | classno
-------+---------
     1 |       1
     2 |       2
     3 |       3
(3 rows)
```

4、直接更新所有记录的值。

```
UPDATE student1 SET classno = classno*2;
```

5、查看数据。

```
SELECT * FROM student1;
```

查询结果显示为：

```
 stuno | classno
-------+---------
     1 |       2
     2 |       4
     3 |       6
```

```
(3 rows)
```

6、删除表。

```
DROP TABLE student1;
```

4.5.193.VACUUM

功能描述

VACUUM 回收表或 B-Tree 索引中已经删除的行所占据的存储空间。在一般的数据库操作里，那些已经 DELETE 的行并没有从它们所属的表中物理删除；在完成 VACUUM 之前它们仍然存在。因此有必要周期地运行 VACUUM，特别是在经常更新的表上。

注意事项

- 如果没有参数，VACUUM 处理当前数据库里用户拥有相应权限的每个表。如果参数指定了一个表，VACUUM 只处理指定的那个表。
- 要对一个表进行 VACUUM 操作，通常用户必须是表的所有者或者被授予了指定表 VACUUM 权限的用户，默认系统管理员有该权限。数据库的所有者允许对数据库中除了共享目录以外的所有表进行 VACUUM 操作（该限制意味着只有系统管理员才能真正对一个数据库进行 VACUUM 操作）。VACUUM 命令会跳过那些用户没有权限的表进行垃圾回收操作。
- VACUUM 不能在事务块内执行。
- 建议生产数据库经常清理（至少每晚一次），以保证不断地删除失效的行。尤其是在增删了大量记录之后，对受影响的表执行 VACUUM ANALYZE 命令是一个很好的习惯。这样将更新系统目录为最近的更改，并且允许查询优化器在规划用户查询时有更好地选择。
- 不建议日常使用 FULL 选项，但是可以在特殊情况下使用。例如在用户删除了一个表的大部分行之后，希望从物理上缩小该表以减少磁盘空间占用。VACUUM FULL 通常要比单纯的 VACUUM 收缩更多的表尺寸。FULL 选项并不清理索引，所以推荐周期性的运行 REINDEX 命令。实际上，首先删除所有索引，再运行 VACUUM FULL 命令，最后重建索引通常是更快的选择。如果执行此命令后所占用物理空间无变化（未减少），请确认是否有其他活跃事务（删除数据事务开始之前开始的事务，并在 VACUUM

FULL 执行前未结束）存在，如果有等其他活跃事务退出进行重试。

- VACUUM 会导致 I/O 流量的大幅增加，这可能会影响其他活动会话的性能。因此，有时候会建议使用基于开销的 VACUUM 延迟特性。
- 如果指定了 VERBOSE 选项，VACUUM 将打印处理过程中的信息，以表明当前正在处理的表。各种有关当前表的统计信息也会打印出来。但是对于列存表执行 VACUUM 操作，指定了 VERBOSE 选项，无信息输出。
- 当含有带括号的选项列表时，选项可以以任何顺序写入。如果没有括号，则选项必须按语法显示的顺序给出。
- VACUUM 和 VACUUM FULL 时，会根据参数 vacuum_defer_cleanup_age 延迟清理行存表记录，即不会立即清理刚刚删除的元组。
- VACUUM ANALYZE 先执行一个 VACUUM 操作，然后给每个选定的表执行一个 ANALYZE。对于日常维护脚本而言，这是一个很方便的组合。
- 简单的 VACUUM（不带 FULL 选项）只是简单地回收空间并且令其可以再次使用。这种形式的命令可以和对表的普通读写并发操作，因为没有请求排他锁。VACUUM FULL 执行更广泛的处理，包括跨块移动行，以便把表压缩到最少的磁盘块数目里。这种形式要慢许多并且在处理的时候需要在表上施加一个排他锁。
- VACUUM 列存表内部执行的操作包括三个：迁移 delta 表中的数据到主表、VACUUM 主表的 delta 表、VACUUM 主表的 desc 表。该操作不会回收 delta 表的存储空间，如果要回收 delta 表的冗余存储空间，需要对该列存表执行 VACUUM DELTAMERGE。
- 同时执行多个 VACUUM FULL 可能出现死锁。
- 如果没有打开 xc_maintenance_mode 参数，那么 VACUUM FULL 操作将跳过所有系统表。
- 执行 DELETE 后立即执行 VACUUM FULL 命令，可能不会回收空间，这取决于是否有 DELETE 事务之前开启的事务仍处于活跃状态。此时需要等待所有事务结束，或者重启数据库，再重新执行 VACUUM FULL 命令进行清理。
- 执行 VACUUM FULL 时，建议首先删除相关表上的所有索引，再运行 VACUUM FULL 命令，最后重建索引。

语法格式

❖ 回收空间并更新统计信息，对关键字顺序无要求。

```
VACUUM [ ( { FULL | FREEZE | VERBOSE | {ANALYZE | ANALYSE }} [,...] ) ]
    [ table_name [ (column_name [, ...] ) ] ] [ PARTITION ( partition_name ) | SUBPAR
TITION ( subpartition_name ) ];
```

❖ 仅回收空间，不更新统计信息。

```
VACUUM [ FULL [COMPACT] ] [ FREEZE ] [ VERBOSE ] [ table_name ]
[ PARTITION ( partition_name ) | SUBPARTITION ( subpartition_name ) ];
```

❖ 回收空间并更新统计信息，且对关键字顺序有要求。

```
VACUUM [ FULL ] [ FREEZE ] [ VERBOSE ] { ANALYZE | ANALYSE } [ VERBOSE ]
    [ table_name [ (column_name [, ...] ) ] ] [ PARTITION ( partition_name )| SUBPAR
TITION ( subpartition_name ) ];
```

参数说明

❖ **FULL**

选择“FULL”清理，这样可以恢复更多的空间，但是需要耗时更多，并且在表上施加了排他锁。

📖 **说明：**

使用 FULL 参数会导致统计信息丢失，如果需要收集统计信息，请在 VACUUM FULL 语句中加上 analyze 关键字。

❖ **FREEZE**

指定 FREEZE 相当于执行 VACUUM 时将 vacuum_freeze_min_age 参数设为 0。

❖ **VERBOSE**

为每个表打印一份详细的清理工作报告。

❖ **ANALYZE | ANALYSE**

更新用于优化器的统计信息，以决定执行查询的最有效方法。

❖ **table_name**

要清理的表的名称（可以有模式修饰）。

取值范围：要清理的表的名称。缺省时为当前数据库中的所有表。

❖ **column_name**

要分析的具体的字段名称，需要配合 analyze 选项使用。

取值范围：要分析的具体的字段名称。缺省时为所有字段。

- **PARTITION**

  COMPACT 和 PARTITION 参数不能同时使用。

- **partition_name**

  要清理的表的一级分区名称。缺省时为所有一级分区。

- **subpartition_name**

  要清理的表的二级分区名称。缺省时为所有二级分区

- **DELTAMERGE**

  只针对列存表，将列存表的 delta table 中的数据转移到主表存储上。对列存表而言，此操作受 enable_delta_store 和“SQL 语法参考->SQL 语法->CREATE-TABLE”章节中的参数说明中的 deltarow_threshold 控制。

示例

```
--在表 reason 上创建索引。
panweidb=# CREATE UNIQUE INDEX ds_reason_index1 ON reason(r_reason_sk);

--对带索引的表 reason 执行 VACUUM 操作。
panweidb=# VACUUM (VERBOSE, ANALYZE) reason;

--删除索引。
panweidb=# DROP INDEX ds_reason_index1 CASCADE;
panweidb=# DROP TABLE reason;
```

4.5.194.VALUES

功能描述

根据给定的值表达式计算一个或一组行的值。它通常用于在一个较大的命令内生成一个“常数表”。

注意事项

- 应当避免使用 VALUES 返回数量非常大的结果行，否则可能会遭遇内存耗尽或者性能低下。出现在 INSERT 中的 VALUES 是一个特殊情况，因为目标字段类型可以从 INSERT 的目标表获知，并不需要通过扫描 VALUES 列表来推测，所以在此情况下可以处理非常大的结果行。
- 如果指定了多行，那么每一行都必须拥有相同的元素个数。

语法格式

```
VALUES {( expression [, ...] )} [, ...]
  [ ORDER BY { sort_expression [ ASC | DESC | USING operator ] } [, ...] ]
  [ LIMIT { count | ALL } ]
  [ OFFSET start [ ROW | ROWS ] ]
  [ FETCH { FIRST | NEXT } [ count ] { ROW | ROWS } ONLY ];
```

参数说明

- **expression**

  用于计算或插入结果表指定地点的常量或者表达式。

  在一个出现在 INSERT 顶层的 VALUES 列表中，expression 可以被 DEFAULT 替换以表示插入目的字段的缺省值。除此以外，当 VALUES 出现在其他场合的时候是不能使用 DEFAULT 的。

- **sort_expression**

  一个表示如何排序结果行的表达式或者整数常量。

- **ASC**

  指定按照升序排列。

- **DESC**

  指定按照降序排列。

- **operator**

  一个排序操作符。

- **count**

  返回的最大行数。

- **OFFSET start { ROW | ROWS }**

  声明返回的最大行数，而 start 声明开始返回行之前忽略的行数。

- **FETCH { FIRST | NEXT } [count] { ROW | ROWS } ONLY**

FETCH 子句限定返回查询结果从第一行开始的总行数，count 的缺省值为 1。

示例

参考：SQL 语法参考->SQL 语法->INSERT 章节。

4.5.195.WITH FUNCTION

功能描述

with function 功能实现了在 with 语句中定义临时函数，在后续的子语句中可以重复使用该函数。

注意事项

无

语法格式

```
WITH { FUNCTION func_name ( [ [ argmode ] [ argname ] argtype [ { DEFAULT | = } default_expr ] [, ...] ] )
  [ RETURNS rettype
    | RETURNS TABLE ( column_name column_type [, ...] ) ] AS 'definition' } [, ...]
{ SelectStmt }
```

参数说明

- **func_name**

自定义的函数名称。

取值范围：字符串，要符合标识符命令规范。

- **argmode**

函数参数的模式。

取值范围：IN，OUT，INOUT 或 VARIADIC（用于声明数组类型的参数），缺省值是 IN。只有 OUT 模式的参数后面能跟 VARIADIC。并且 OUT 和 INOUT 模式的参数不能用在 RETURNS TABLE 的函数定义中。

- **argname**

函数参数的名称。

取值范围：字符串，要符合标识符命令规范。

- **argtype**

函数参数的类型。

- **rettype**

函数返回值的数据类型。如果存在 OUT 或 IN OUT 参数，可以省略 RETURNS 子句。如果出现了，那么它必须和输出参数隐含的结果类型兼容：如果有多个输出参数，必须是 RECORD， 如果只有一个输出参数，则与其相同。

- **column_name**

字段名称。

- **column_type**

字段类型。

- **definition**

一个定义函数的字符串长，含义取决于语言。它可以是一个内部函数名称、一个指向某个目标文件的路径、一个 SQL 查询、一个过程语言文本。

示例

使用 WITH FUNCTION 字句调用函数。

```
WITH FUNCTION withfunc (x integer) RETURNS INTEGER AS $$
BEGIN
RETURN x+1;
END
$$,
FUNCTION withfunc2(x INTEGER, y INTEGER) RETURNS INTEGER AS $$
```

```
BEGIN
RETURN x+y;
END;
$$,
FUNCTION withfunc3(x TEXT) RETURNS TEXT AS $$
BEGIN
RETURN x || '-test';
END;
$$
SELECT withfunc(1),withfunc2(2,3),withfunc3('4');
```

当结果显示如下，则表示函数调用成功：

```
 withfunc | withfunc2 | withfunc3
----------+-----------+-----------
        2 |         5 | 4-test
(1 row)
```

相关链接

无

4.6.类型转换概述

在 SQL 语言中，每个数据都与一个决定其行为和用法的数据类型相关。PanWeiDB 提供一个可扩展的数据类型系统，该系统比其他 SQL 实现更具通用性和灵活性。因而，PanWeiDB 中大多数类型转换是由通用规则来管理的，这种做法允许使用混合类型的表达式。

PanWeiDB 扫描/分析器只将词法元素分解成五个基本种类：整数、浮点数、字符串、标识符和关键字。大多数非数字类型首先表现为字符串。SQL 语言的定义允许将常量字符串声明为具体的类型。例如下面查询：

```
SELECT text 'Origin' AS "label", point '(0,0)' AS "value";
 label  | value
--------+-------
 Origin | (0,0)
(1 row)
```

示例中有两个文本常量，类型分别为 text 和 point。如果没有为字符串文本声明类型，则该文本首先被定义成一个 unknown 类型。

在 PanWeiDB 分析器里，有四种基本的 SQL 结构需要独立的类型转换规则：

❖ **函数调用**

多数 SQL 类型系统是建筑在一套丰富的函数上的。函数调用可以有一个或多个参数。因为 SQL 允许函数重载，所以不能通过函数名直接找到要调用的函数，分析器必须根据函数提供的参数类型选择正确的函数。

❖ **操作符**

SQL 允许在表达式上使用前缀或后缀（单目）操作符，也允许表达式内部使用双目操作符（两个参数）。像函数一样，操作符也可以被重载，因此操作符的选择也和函数一样取决于参数类型。

❖ **值存储**

INSERT 和 UPDATE 语句将表达式结果存入表中。语句中的表达式类型必须和目标字段的类型一致或者可以转换为一致。

❖ **UNION，CASE 和相关构造**

因为联合 SELECT 语句中的所有查询结果必须在一列里显示出来，所以每个 SELECT 子句中的元素类型必须相互匹配并转换成一个统一类型。类似地，一个 CASE 构造的结果表达式必须转换成统一的类型，这样整个 CASE 表达式会有一个统一的输出类型。同样的要求也存在于 ARRAY 构造以及 GREATEST 和 LEAST 函数中。

系统表 pg_cast 存储了有关数据类型之间的转换关系以及如何执行这些转换的信息。详细信息请参见“PG_CAST”章节。

语义分析阶段会决定表达式的返回值类型并选择适当的转换行为。数据类型的基本类型分类，包括：Boolean，numeric，string，bitstring，datetime，timespan，geometric 和 network。每种类型都有一种或多种首选类型用于解决类型选择的问题。根据首选类型和可用的隐含转换，就可能保证有歧义的表达式（那些有多个候选解析方案的）得到有效的方式解决。

所有类型转换规则都是建立在下面几个基本原则上的：

- 隐含转换决不能有奇怪的或不可预见的输出。
- 如果一个查询不需要隐含的类型转换，分析器和执行器不应该进行更多的额外操作。这就是说，任何一个类型匹配、格式清晰的查询不应该在分析器里耗费更多的时间，也不应该向查询中引入任何不必要的隐含类型转换调用。
- 另外，如果一个查询在调用某个函数时需要进行隐式转换，当用户定义了一个有正确参数的函数后，解释器应该选择使用新函数。

4.6.1.操作符

操作符类型解析

1、从系统表 pg_operator 中选出要考虑的操作符。如果可以找到一个参数类型以及参数个数都一致的操作符，那么这个操作符就是最终使用的操作符。如果找到了多个备选的操作符，我们将从中选择一个最合适的。

2、寻找最优匹配。

- 抛弃那些输入类型不匹配并且也不能隐式转换成匹配的候选操作符。unknown 文本在这种情况下可以转换成任何东西。如果只剩下一个候选项，则用之，否则继续下一步。
- 遍历所有候选操作符，保留那些输入类型匹配最准确的。此时，域被看作和他们的基本类型相同。如果没有一个操作符能被保留，则保留所有候选。如果只剩下一个候选项，则用之，否则继续下一步。
- 遍历所有候选操作符，保留那些需要类型转换时接受（属于输入数据类型的类型范畴的）首选类型位置最多的操作符。如果没有接受首选类型的操作符，则保留所有候选。如果只剩下一个候选项，则用之，否则继续下一步。
- 如果有任何输入参数是 unknown 类型，检查剩余的候选操作符对应参数位置的类型范畴。在每一个能够接受字符串类型范畴的位置使用 string 类型（这种对字符串的偏爱是合适的，因为 unknown 文本确实像字符串）。另外，如果所有剩下的候选操作符都接受相同的类型范畴，则选择该类型范畴，否则抛出一个错误（因为在没有更多线索的条件下无法作出正确的

选择）。现在抛弃不接受选定的类型范畴的候选操作符，然后，如果任意候选操作符在某个给定的参数位置接受一个首选类型，则抛弃那些在该参数位置接受非首选类型的候选操作符。如果没有一个操作符能被保留，则保留所有候选。如果只剩下一个候选项，则用之，否则继续下一步。

- 如果同时有 unknown 和已知类型的参数，并且所有已知类型的参数都是相同的类型，那么假设 unknown 参数也是那种类型，并检查哪个候选操作符在 unknown 参数位置接受那个类型。如果只有一个操作符符合，那么使用它。否则，产生一个错误。

示例

- 阶乘操作符类型解析。在系统表中里只有一个阶乘操作符（后缀!），它以 bigint 作为参数。扫描器给下面查询表达式的参数赋予 bigint 的初始类型：

```
SELECT 40 ! AS "40 factorial";
```

结果显示如下：

```
      40 factorial
-----------------------------------------------
 815915283247897734345611269596115894272000000000
(1 row)
```

分析器对参数做类型转换，查询等效于：

```
SELECT CAST(40 AS bigint) ! AS "40 factorial";
```

- 字符串连接操作符类型分析。一种字符串风格的语法既可以用于字符串也可以用于复杂的扩展类型。未声明类型的字符串将被所有可能的候选操作符匹配。有一个未声明的参数的例子：

```
SELECT text 'abc' || 'def' AS "text and unknown";
```

结果显示如下：

```
 text and unknown
------------------
 abcdef
(1 row)
```

本例中分析器寻找两个参数都是 text 的操作符。确实有这样的操作符，两个参数都是 text 类型。

下面是连接两个未声明类型的值：

```
SELECT 'abc' || 'def' AS "unspecified";
```

结果显示如下：

```
unspecified
-------------
 abcdef
(1 row)
```

说明

因为查询中没有声明任何类型，所以本例中对类型没有任何初始提示。因此，分析器查找所有候选操作符，发现既存在接受字符串类型范畴的操作符也存在接受位串类型范畴的操作符。因为字符串类型范畴是首选，所以选择字符串类型范畴的首选类型 text 作为解析未知类型文本的声明类型。

- 绝对值和取反操作符类型分析。PanWeiDB 操作符表里面有几条记录对应于前缀操作符@，它们都用于为各种数值类型实现绝对值操作。其中之一用于 float8 类型，它是数值类型范畴中的首选类型。因此，在面对 unknown 输入的时候， PanWeiDB 会使用该类型：

```
SELECT @ '-4.5' AS "abs";
```

结果显示如下：

```
abs
-----
 4.5
(1 row)
```

此处，系统在应用选定的操作符之前隐式的转换 unknown 类型的文字为 float8 类型。

❖ 数组包含操作符类型分析。这里是解决一个操作符带有一个已知和一个未知类型输入的例子：

```
SELECT array[1,2] <@ '{1,2,3}' as "is subset";
```

结果显示如下：

```
is subset
-----------
t
(1 row)
```

说明

PanWeiDB 操作符表有几条记录对应于中缀操作符<@，但是只有两个可以在左侧接受一个整数数组的操作符是数组包含(anyarray <@ anyarray) 和范围包含(anyelement <@ anyrange)的。因为没有多态的伪类型(参阅伪类型)是首选的，所以解析器不能解决这个基础上的歧义。然而，最后一个解析规则告诉用户，假设未知类型的文字是和另外一个输入相同的类型，也就是，整数数组。现在只有两个操作符中的一个可以匹配，所以选择数组包含。（如果用户选择了范围包含，用户将得到一个错误，因为字符串没有正确的格式成为范围的文字。）

4.6.2.UNION,CASE 和相关构造

SQL UNION 构造必须把那些可能不太相似的类型匹配起来成为一个结果集。解析算法分别应用于联合查询的每个输出字段。INTERSECT 和 EXCEPT 构造对不相同的类型使用和 UNION 相同的算法进行解析。CASE、ARRAY、VALUES、GREATEST 和 LEAST 构造也使用同样的算法匹配它的部件表达式并且选择一个结果数据类型。

UNION，CASE 和相关构造解析

❖ 如果所有输入都是相同的类型，并且不是 unknown 类型，那么解析成这种类型。

- 如果所有输入都是 unknown 类型则解析成 text 类型（字符串类型范畴的首选类型）。否则，忽略 unknown 输入。
- 如果输入不属于同一个类型范畴（unknown 类型除外），失败。
- 如果输入类型是同一个类型范畴，则选择该类型范畴的首选类型（例外：union 操作会选择第一个分支的类型作为所选类型）。

说明

系统表 pg_type 中 typcategory 表示数据类型范畴 typispreferred 表示是否是 typcategory 分类中的首选类型。

- 把所有输入转换为所选的类型（对于字符串保持原有长度）。如果从给定的输入到所选的类型没有隐式转换则失败。
- 若输入中含 json、txid_snapshot、sys_refcursor 或几何类型，则不能进行 union。

对于 case 和 coalesce，在 TD 兼容模式下的处理

- 如果所有输入都是相同的类型，并且不是 unknown 类型，那么解析成这种类型。
- 如果所有输入都是 unknown 类型则解析成 text 类型。
- 如果输入字符串（包括 unknown，unknown 当 text 来处理）和数字类型，那么解析成字符串类型，如果是其他不同的类型范畴，则报错。
- 如果输入类型是同一个类型范畴，则选择该类型的优先级较高的类型。
- 把所有输入转换为所选的类型。如果从给定的输入到所选的类型没有隐式转换则失败。

对于 case，在 ORA 兼容模式下的处理

decode(expr, search1, result1, search2, result2, ..., defresult)，也即 case expr when search1 then result1 when search2 then result2 else defresult end; 在 ORA 兼容模式下的处理，将整个表达式最终的返回值类型定为 result1 的数据类型，或者与 result1 同类型范畴的更高精度的数据类型。

（例如，numeric 与 int 同属数值类型范畴，但 numeric 比 int 精度要高，具有更高优先级）

- 将 result1 的数据类型置为最终的返回值类型 preferType，其所属类型范畴为 preferCategory。
- 依次考虑 result2、result3 直至 defresult 的数据类型。如果其类型范畴也是 preferCategory，即与 result1 具有相同的类型范畴，则判断其精度（优先级）是否高于 preferType，如果高于，则将 preferType 更新为更高精度的数据类型；如果其类型范畴不是 preferCategory，则判断其数据类型是否可以隐式转换为 preferType，不可以则报错。
- 将最终 preferType 记录的数据类型作为整个表达式最终的返回值类型；表达式的结果向此类型进行隐式转换。

⚠ 注意

- 为了兼容一种特殊情况，即表示了超大数字的字符类型向数值类型转换的情况，例如 select decode(1, 2, 2, “53465465676465454657567678676”)，大数超过了 bigint、double 等的表示范围。所以，当 result1 的类型范畴为数值类型时，将返回值的类型直接置为 numeric，以兼容此种特殊情况。
- 数值类型的优先级排序：numeric>float8>float4>int8>int4>int2>int1
- 字符类型的优先级排序：text>varchar=nvarchar2>bpchar>char
- 日期类型的优先级排序 timestamptz>timestamp>smalldatetime>date>abstime>timetz>time
- 日期跨度类型的优先级排序：interval>tinterval>reltime

示例

- Union 中的待定类型解析。这里，unknown 类型文本'b'将被解析成 text 类型。

```
SELECT text 'a' AS "text" UNION SELECT 'b';
```

结果显示如下：

```
text
------
 a
 b
(2 rows)
```

❖ 简单 Union 中的类型解析。文本 1.2 的类型为 numeric，而且 integer 类型的 1 可以隐含地转换为 numeric，因此使用这个类型。

```
SELECT 1.2 AS "numeric" UNION SELECT 1;
```

结果显示如下：

```
 numeric
---------
   1
  1.2
(2 rows)
```

❖ 转置 Union 中的类型解析。这里，因为类型 real 不能被隐含转换成 integer，但是 integer 可以隐含转换成 real，那么联合的结果类型将是 real。

```
SELECT 1 AS "real" UNION SELECT CAST('2.2' AS REAL);
```

结果显示如下：

```
real
------
 1
 2.2
(2 rows)
```

❖ TD 模式下，coalesce 参数输入 int 和 varchar 类型，那么解析成 varchar 类型。

1、在 TD 模式下，创建 TD 兼容模式的数据库 td_1 并切换数据库为 td_1。

```
CREATE DATABASE td_1;
\c td_1
```

2、创建表 t1。

```
CREATE TABLE t1(a int, b varchar(10));
```

3、查看 coalesce 参数输入 int 和 varchar 类型的查询语句的执行计划。

```
EXPLAIN VERBOSE select coalesce(a, b) from t1;
```

当结果显示如下信息，则表示验证完成。

```
                QUERY PLAN
------------------------------------------------------------------------------------
 Seq Scan on public.t1 (cost=0.00..24.18 rows=1134 width=42)
  Output: (COALESCE((a)::varchar, b)
(2 rows)
```

4、删除测试库

```
\c panweidb
DROP DATABASE td_1;
```

4.6.3.函数

函数类型解析

1、从系统表 pg_proc 中选择所有可能被选到的函数。如果使用了一个不带模式修饰的函数名称，那么认为该函数是那些在当前搜索路径中的函数。如果给出一个带修饰的函数名，那么只考虑指定模式中的函数。如果搜索路径中找到了多个不同参数类型的函数。将从中选择一个合适的函数。

2、查找和输入参数类型完全匹配的函数。如果找到一个，则用之。如果输入的实参类型都是 unknown 类型，则不会找到匹配的函数。

3、如果未找到完全匹配，请查看该函数是否为一个特殊的类型转换函数。

4、寻找最优匹配。

- 抛弃那些输入类型不匹配并且也不能隐式转换成匹配的候选函数。
  unknown 文本在这种情况下可以转换成任何东西。如果只剩下一个候选项，则用之，否则继续下一步。

- 遍历所有候选函数，保留那些输入类型匹配最准确的。此时，域被看作和它们的基本类型相同。如果没有一个函数能准确匹配，则保留所有候选。如果只剩下一个候选项，则用之，否则继续下一步。
- 遍历所有候选函数，保留那些需要类型转换时接受首选类型位置最多的函数。如果没有接受首选类型的函数，则保留所有候选。如果只剩下一个候选项，则用之，否则继续下一步。
- 如果有任何输入参数是 unknown 类型，检查剩余的候选函数对应参数位置的类型范畴。在每一个能够接受字符串类型范畴的位置使用 string 类型（这种对字符串的偏爱是合适的，因为 unknown 文本确实像字符串）。另外，如果所有剩下的候选函数都接受相同的类型范畴，则选择该类型范畴，否则抛出一个错误（因为在没有更多线索的条件下无法作出正确的选择）。现在抛弃不接受选定的类型范畴的候选函数，然后，如果任意候选函数在那个范畴接受一个首选类型，则抛弃那些在该参数位置接受非首选类型的候选函数。如果没有一个候选符合这些测试则保留所有候选。如果只有一个候选函数符合，则使用它；否则，继续下一步。
- 如果同时有 unknown 和已知类型的参数，并且所有已知类型的参数有相同的类型，假设 unknown 参数也是这种类型，检查哪个候选函数可以在 unknown 参数位置接受这种类型。如果正好一个候选符合，那么使用它。否则，产生一个错误。

示例

- 圆整函数参数类型解析。只有一个 round 函数有两个参数（第一个是 numeric，第二个是 integer）。所以下面的查询自动把第一个类型为 integer 的参数转换成 numeric 类型。

```
SELECT round(4, 4);
```

结果显示如下：

```
 round
--------
 4.0000
(1 row)
```

实际上它被分析器转换成：

```
SELECT round(CAST (4 AS numeric), 4);
```

因为带小数点的数值常量初始时被赋予 numeric 类型，因此下面的查询将不需要类型转换，并且可能会略微高效一些：

```
SELECT round(4.0, 4);
```

- 子字符串函数类型解析。有好几个 substr 函数，其中一个接受 text 和 integer 类型。如果用一个未声明类型的字符串常量调用它，系统将选择接受 string 类型范畴的首选类型（也就是 text 类型）的候选函数。

```
SELECT substr('1234', 3);
```

结果显示如下：

```
 substr
--------
  34
(1 row)
```

如果该字符串声明为 varchar 类型，就像从表中取出来的数据一样，分析器将试着将其转换成 text 类型：

```
SELECT substr(varchar '1234', 3);
```

结果显示如下：

```
 substr
--------
  34
(1 row)
```

被分析器转换后实际上变成：

```
SELECT substr(CAST (varchar '1234' AS text), 3);
```

说明

分析器从 pg_cast 表中了解到 text 和 varchar 是二进制兼容的，意思是说一个可以传递给接受另一个的函数而不需要做任何物理转换。因此，在这种情况下，实际上没有做任何类型转换。

如果以 integer 为参数调用函数，分析器将试图将其转换成 text 类型：

```
SELECT substr(1234, 3);
```

结果显示如下：

```
substr
--------
 34
(1 row)
```

被分析器转换后实际上变成：

```
SELECT substr(CAST (1234 AS text), 3);
```

结果显示如下：

```
 substr
--------
  34
(1 row)
```

4.6.4.值存储

值存储数据类型解析

1、查找与目标字段准确的匹配。

2、试着将表达式直接转换成目标类型。如果已知这两种类型之间存在一个已注册的转换函数，那么直接调用该转换函数即可。如果表达式是一个未知类型文本，该文本字符串的内容将交给目标类型的输入转换过程。

3、检查一下看目标类型是否有长度转换。长度转换是一个从某类型到自身的转换。如果在 pg_cast 表里面找到一个，那么在存储到目标字段之前先在表达式上应用。这样的转换函数总是接受一个额外的类型为 integer 的参数，它接

收目标字段的 atttypmod 值（实际上是其声明长度，atttypmod 的解释随不同的数据类型而不同），并且它可能接受一个 Boolean 类型的第三个参数，表示转换是显式的还是隐式的。转换函数负责施加那些长度相关的语义，比如长度检查或者截断。

示例

character 存储类型转换。对一个目标列定义为 character(20)的语句，下面的示例显示存储值的长度正确：

1、创建测试表。

```
CREATE TABLE value_storage_t1 (
 VS_COL1 CHARACTER(20)
);
```

2、插入测试数据。

```
INSERT INTO value_storage_t1 VALUES('abcdef');
```

3、查询结果。

```
SELECT VS_COL1, octet_length(VS_COL1) FROM value_storage_t1;
```

当结果显示如下信息，则表示存储值长度正确。

```
  vs_col1    | octet_length
----------------------+--------------
 abcdef      |     20
(1 row)
```

4、删除测试表。

```
DROP TABLE value_storage_t1;
```

这里真正发生的事情是两个 unknown 文本缺省解析成 text，这样就允许||操作符解析成 text 连接。然后操作符的 text 结果转换成 bpchar("空白填充的字符型", character 类型内部名称)以匹配目标字段类型。不过，从 text 到 bpchar 的转换是二进制兼容的，这样的转换是隐含的并且实际上不做任何函数调用。最后，在系统表里找到长度转换函数

bpchar(bpchar, integer, Boolean) 并且应用于该操作符的结果和存储的字段长。这个类型相关的函数执行所需的长度检查和额外的空白填充。

4.7.全文检索

4.7.1.介绍

4.7.2.全文检索概述

功能描述

文本搜索操作符在数据库中已存在多年。PanWeiDB 为文本数据类型提供~、~*、LIKE 和 ILIKE 操作符用于匹配字符进行检索，同时可以通过使用词典和索引满足现代信息系统所要求的许多必要属性。

注意事项

- 每个分词长度必须小于 2K 字节。
- tsvector 结构（分词+位置）的长度必须小于 1M 字节。
- tsvector 的位置值必须大于 0，且小于等于 16,383 字节。
- 每个分词在文档中位置数必须小于 256，若超过将舍弃后面的位置信息。
- tsquery 中的关键字及对应运算符最大支持到 32768 字节。
- 没有语义支持，即使是英语。由于要识别派生词并不是那么容易，因此正则表达式也不能满足要求。如，satisfies 和 satisfy，当使用正则表达式寻找 satisfy 时，并不会查询到包含 satisfies 的文档。用户可以使用 OR 搜索多种派生形式，但过程非常繁琐。并且有些词会有上千的派生词，因此容易出错。
- 没有对搜索结果的分类（排序）。当搜索出成千的文档时，查找效率很低。
- 由于没有索引的支持，每一次的搜索需要遍历所有的文档，整体搜索比较缓慢。使用全文索引可以对文档进行预处理，并且可以使后续的搜索更快速。预处理过程包括：
- 将文档解析成 token。为每个文档标记不同类别的 token 是非常有必要的，例如：数字、文字、复合词、电子邮件地址，这样就可以做不同的处理。

原则上 token 的类别依赖于具体的应用，但对于大多数的应用来说，可以使用一组预定义的 token 类。

- 将 token 转换为词素。词素像 token 一样是一个字符串，但它已经标准化处理，这样同一个词的不同形式是一样的。例如，标准化通常包括：将大写字母转换成小写字母、删除后缀（如英语中的 s 或者 es）。这将允许通过搜索找到同一个词的不同形式，不需要繁琐地输入所有可能的变形样式。同时，这一步通常会删除停用词。这些停用词通常因为太常见而对搜索无用。（总之，token 是文档文本的原片段，而词素被认为是有用的索引和搜索词。）PanWeiDB 使用词典执行这一步，且提供了各种标准的词典。
- 保存搜索优化后的预处理文档。比如，每个文档可以呈现为标准化词素的有序组合。伴随词素，通常还需要存储词素位置信息以用于邻近排序。因此文档包含的查询词越密集其排序越高。
- 词典能够对 token 如何标准化做到细粒度控制。使用合适的词典，可以定义不被索引的停用词。
- 数据类型 tsvector 用于存储预处理文档，tsquery 用于存储查询条件，详细请参见“SQL 语法参考->数据类型->文本搜索类型”章节。为这些数据类型提供的函数和操作符请参见“SQL 语法参考->数据类型->文本检索函数和操作符”章节。其中最重要的是匹配运算符@@，将在“基本文本匹配”章节中介绍。
- 分词器

功能描述

全文检索功能还可以做更多事情：忽略索引某个词（停用词），处理同义词和使用复杂解析，例如：不仅基于空格的解析。这些功能通过文本搜索分词器控制。PanWeiDB 支持多语言的预定义的分词器，并且可以创建分词器（\dF 命令显示了所有可用分词器）。

为了更方便的建立自定义文本搜索分词器，可以通过简单的数据库对象建立分词器。 PanWeiDB 文本搜索功能提供了四种类型与分词器相关的数据库对象：

- 文本搜索解析器将文档分解为 token，并且分类每个 token（例如：词和数字）。

- 文本搜索词典将 token 转换成规范格式并且丢弃停用词。
- 文本搜索模板提供潜在的词典功能：一个词典指定一个模板，并且为模板设置参数。
- 文本搜索分词器选择一个解析器，并且使用一系列词典规范化语法分析器产生的 token。

使用方法

- 在安装期间选择一个合适的分词器，并且在 postgresql.conf 中相应的设置 default_text_search_config。
- 如果为了 PanWeiDB 使用同一个文本搜索分词器可以使用 postgresql.conf 中的值。
- 如果需要在 PanWeiDB 中使用不同分词器，可以使用 ALTER DATABASE ... SET 在任一数据库进行配置。
- 用户也可以在每个会话中设置 default_text_search_config。

注意事项

每个依赖于分词器的文本搜索函数有一个可选的配置参数，用以明确声明所使用的分词器。仅当忽略这个参数的时候，才使用 default_text_search_config。

4.7.2.1.基本文本匹配

功能描述

PanWeiDB 的全文检索基于匹配算子@@，当一个 tsvector(document)匹配到一个 tsquery(query)时，则返回 true。其中，tsvector(document)和 tsquery(query)两种数据类型可以任意排序。

示例

- 查询验证结果。

```
SELECT 'a fat cat sat on a mat and ate a fat rat'::tsvector @@ 'cat & rat'::tsquery AS RESULT;
```

结果返回如下：

```
 result
----------
 t
(1 row)
SELECT 'fat & cow'::tsquery @@ 'a fat cat sat on a mat and ate a fat rat'::tsvector AS
 RESULT;
```

结果返回如下：

```
 result
----------
 f
(1 row)
```

- tsquery 不仅是文本，且比 tsvector 包含的要多。tsquery 包含已经标注化为词条的搜索词，同时可能是使用 AND、OR、或 NOT 操作符连接的多个术语。详细请参见“SQL 语法参考->数据类型->文本搜索类型”章节。函数 to_tsquery 和 plainto_tsquery 对于将用户书写文本转换成适合的 tsquery 是非常有用的，比如将文本中的词标准化。类似地，to_tsvector 用于解析和标准化文档字符串。因此，实际中文本搜索匹配看起来更像这样：

```
SELECT to_tsvector('fat cats ate fat rats') @@ to_tsquery('fat & rat') AS RESULT;
```

结果返回如下：

```
result
----------
 t
(1 row)
```

需要注意的是，下面这种方式是不可行的：

```
SELECT 'fat cats ate fat rats'::tsvector @@ to_tsquery('fat & rat')AS RESULT;
```

结果返回如下：

```
result
----------
 f
```

```
(1 row)
```

❖ 由于 tsvector 没有对 rats 进行标准化，所以 rats 不匹配 rat。

@@操作符也支持 text 输入，允许一个文本字符串的显示转换为 tsvector 或者在简单情况下忽略 tsquery。可用形式是：

```
tsvector @@ tsquery
tsquery  @@ tsvector
text @@ tsquery
text @@ text
```

我们已经看到了前面两种，形式 text @@ tsquery 等价于 to_tsvector(text) @@ tsquery，而 text @@ text 等价于 to_tsvector(text) @@ plainto_tsquery(text)。

4.7.2.2.文档

功能描述

文档是全文搜索系统的搜索单元，例如：杂志上的一篇文章或电子邮件消息。文本搜索引擎必须能够解析文档，而且可以存储父文档的关联词素（关键词）。后续，这些关联词素用来搜索包含查询词的文档。

在 PanWeiDB 中，文档通常是一个数据库表中一行的文本字段，或者这些字段的可能组合（级联）。文档可能存储在多个表中或者需动态获取。换句话说，一个文档由被索引化的不同部分构成，因此无法存储为一个整体，如下面示例所示。

另外一种可能是：文档在文件系统中作为简单的文本文件存储。在这种情况下，数据库可以用于存储全文索引并且执行搜索，同时可以使用一些唯一标识从文件系统中检索文档。然而，从数据库外部检索文件需要拥有系统管理员权限或者特殊函数支持。因此，还是将所有数据保存在数据库中比较方便。同时，将所有数据保存在数据库中可以方便地访问文档元数据以便于索引和显示。

注意事项

为了实现文本搜索目的，必须将每个文档减少至预处理后的 tsvector 格式。搜索和相关性排序都是在 tsvector 形式的文档上执行的。原始文档只有在被选中要呈现给用户时才会被检索。因此，我们常将 tsvector 说成文档，但是很显然其实它只是完整文档的一种紧凑表示。

示例

1、创建测试表并插入数据。

```
create table date_dim(
d_dow int,
d_dom int,
d_fy_week_seq int
);

insert into date_dim values(5,6,1);
insert into date_dim values(0,8,1);
insert into date_dim values(2,3,1);
insert into date_dim values(3,4,1);
insert into date_dim values(4,5,1);
insert into date_dim values(1,2,1);
insert into date_dim values(6,7,1);
insert into date_dim values(4,5,2);
insert into date_dim values(1,2,2);
insert into date_dim values(6,7,2);
```

2、查询数据。

```
SELECT d_dow || '-' || d_dom || '-' || d_fy_week_seq  AS identify_serials FROM date_di
m WHERE d_fy_week_seq = 1;
```

结果显示如下

```
identify_serials
-------------------
 5-6-1
 0-8-1
 2-3-1
 3-4-1
 4-5-1
 1-2-1
 6-7-1
(7 rows)
```

实际上，在这些示例查询中，应该使用 coalesce 防止一个独立的 NULL 属性导致整个文档的 NULL 结果。

4.7.3.表和索引

4.7.3.1.创建 GIN 索引

功能描述

为了加速文本搜索，可以创建 GIN 索引。

注意事项

- to_tsvector()函数有两个版本。只输一个参数的版本和输两个参数的版本。只输一个参数时，系统默认采用 default_text_search_config 所指定的分词器。
- 创建索引时必须使用 to_tsvector 的两参数版本。只有指定了分词器名称的全文检索函数才可以在索引表达式中使用。这是因为索引的内容必须不受 default_text_search_config 的影响，否则索引内容可能不一致。由于 default_text_search_config 的值可以随时调整，从而导致不同条目生成的 tsvector 采用了不同的分词器，并且没有办法区分究竟使用了哪个分词器。正确地转储和恢复这样的索引也是不可能的。
- 因为在上述创建索引中 to_tsvector 使用了两个参数，只有当查询时也使用了两个参数，且参数值与索引中相同时，才会使用该索引。也就是说，WHERE to_tsvector('english', body) @@ 'a & b' 可以使用索引，但 WHERE to_tsvector(body) @@ 'a & b'不能使用索引。这确保只使用这样的索引——索引各条目是使用相同的分词器创建的。

示例

- 方法一：

1、创建 GIN 索引，索引中的分词器名称由另一列指定时可以建立更复杂的表达式索引。

```
CREATE INDEX pgweb_idx_1 ON tsearch.pgweb USING gin(to_tsvector('ngram', body));
```

2、索引可以连接列。

```
CREATE INDEX pgweb_idx_2 ON tsearch.pgweb USING gin(to_tsvector('english', titl
e || ' ' || body));
```

❖ 方法二：

1、创建一个单独的 tsvector 列控制 to_tsvector 的输出。下面的例子是 title 和 body 的连接，当其他是 NULL 的时候，使用 coalesce 确保一个字段仍然会被索引：

```
ALTER TABLE tsearch.pgweb ADD COLUMN textsearchable_index_col tsvector;
UPDATE tsearch.pgweb SET textsearchable_index_col = to_tsvector('english', coale
sce(title,'') || ' ' || coalesce(body,''));
```

2、为加速搜索创建一个 GIN 索引。

```
CREATE INDEX textsearch_idx_3 ON tsearch.pgweb USING gin(textsearchable_index
_col);
```

3、执行一个快速全文搜索。

```
SELECT title
FROM tsearch.pgweb
WHERE textsearchable_index_col @@ to_tsquery('north & america')
ORDER BY last_mod_date DESC
LIMIT 10;
```

结果显示如下：

```
 title
--------
 Canada
 Mexico
(2 rows)
```

相比于一个表达式索引，单独列方法的一个优势是：它没有必要在查询时明确指定分词器以便能使用索引。正如上面例子所示，查询可以依赖于 default_text_search_config。另一个优势是搜索比较快速，因为它没有必要重新利用 to_tsvector 调用来验证索引匹配。表达式索引方法更容易建立，且它需要较少的磁盘空间，因为 tsvector 形式没有明确存储。

4.7.3.2.搜索表

功能描述

在不使用索引的情况下也可以进行全文检索。

示例

创建测试表并插入数据。

```
DROP SCHEMA IF EXISTS tsearch CASCADE;
CREATE SCHEMA tsearch;
CREATE TABLE tsearch.pgweb(id int, body text, title text, last_mod_date date);

INSERT INTO tsearch.pgweb VALUES(1, 'China, officially the People''s Republic of Ch
ina (PRC), located in Asia, is the world''s most populous state.', 'China', '2010-1-1');

INSERT INTO tsearch.pgweb VALUES(2, 'America is a rock band, formed in England i
n 1970 by multi-instrumentalists Dewey Bunnell, Dan Peek, and Gerry Beckley.', 'A
merica', '2010-1-1');
INSERT INTO tsearch.pgweb VALUES(3, 'England is a country that is part of the Unit
ed Kingdom. It shares land borders with Scotland to the north and Wales to the we
st.', 'England', '2010-1-1');
INSERT INTO tsearch.pgweb VALUES(4, 'Australia, officially the Commonwealth of A
ustralia, is a country comprising the mainland of the Australian continent, the islan
d of Tasmania, and numerous smaller islands.', 'Australia', '2010-1-1');
INSERT INTO tsearch.pgweb VALUES(6, 'Japan is an island country in East Asia.', 'Ja
pan', '2010-1-1');
INSERT INTO tsearch.pgweb VALUES(7, 'Germany, officially the Federal Republic of
Germany, is a sovereign state and federal parliamentary republic in central-wester
n Europe.', 'Germany', '2010-1-1');
INSERT INTO tsearch.pgweb VALUES(8, 'France, is a sovereign state comprising terri
tory in western Europe and several overseas regions and territories.', 'France', '201
0-1-1');
INSERT INTO tsearch.pgweb VALUES(9, 'Italy officially the Italian Republic, is a unita
ry parliamentary republic in Europe.', 'Italy', '2010-1-1');
```

```
INSERT INTO tsearch.pgweb VALUES(10, 'India, officially the Republic of India, is a c
ountry in South Asia.', 'India', '2010-1-1');
INSERT INTO tsearch.pgweb VALUES(11, 'Brazil, officially the Federative Republic of
 Brazil, is the largest country in both South America and Latin America.', 'Brazil', '20
10-1-1');
INSERT INTO tsearch.pgweb VALUES(12, 'Canada is a country in the northern half of
 North America.', 'Canada', '2010-1-1');
INSERT INTO tsearch.pgweb VALUES(13, 'Mexico, officially the United Mexican State
s, is a federal republic in the southern part of North America.', 'Mexico', '2010-1-1');
```

❖ 示例 1：一个简单查询，将 body 字段中包含 america 的每一行打印出来。像 America 这样的相关词也会被找到，因为这些词都被处理成了相同标准的词条。

```
SELECT id, body, title FROM tsearch.pgweb WHERE to_tsvector('english', body) @@ t
o_tsquery('english', 'america');
```

结果显示如下：

```
 id |                                      body                                      | title
----+--------------------------------------------------------------------------------
-----------------------------------+---------
  2 | America is a rock band, formed in England in 1970 by multi-instrumentalists De
wey Bunnell, Dan Peek, and Gerry Beckley. | America
 11 | Brazil, officially the Federative Republic of Brazil, is the largest country in both
South America and Latin America.   | Brazil
 12 | Canada is a country in the northern half of North America.
                  | Canada
 13 | Mexico, officially the United Mexican States, is a federal republic in the souther
n part of North America.           | Mexico
(4 rows)
```

上面的查询指定 english 配置来解析和规范化字符串。当然也可以省略此配置，通过 default_text_search_config 进行配置设置。

查询 default_text_search_config 参数。

```
SHOW default_text_search_config;
```

```
 default_text_search_config
----------------------------
 pg_catalog.english
(1 row)
```

执行查询

```
SELECT id, body, title FROM tsearch.pgweb WHERE to_tsvector(body) @@ to_tsquer
y('america');
```

结果显示如下：

```
 id |                                        body                                        | title
----+------------------------------------------------------------------------------------
-----------------------------------+---------
  2 | America is a rock band, formed in England in 1970 by multi-instrumentalists De
wey Bunnell, Dan Peek, and Gerry Beckley. | America
 11 | Brazil, officially the Federative Republic of Brazil, is the largest country in both
South America and Latin America.   | Brazil
 12 | Canada is a country in the northern half of North America.
                  | Canada
 13 | Mexico, officially the United Mexican States, is a federal republic in the souther
n part of North America.           | Mexico
(4 rows)
```

❖ 示例 2：一个复杂查询：检索出在 title 或者 body 字段中包含 north 和 america 的最近 10 篇文档：

```
SELECT title FROM tsearch.pgweb WHERE to_tsvector(title || ' ' || body) @@ to_tsquery
('north & america') ORDER BY last_mod_date DESC LIMIT 10;
```

结果显示如下：

```
 title
--------
 Mexico
 Canada
(2 rows)
```

- 为了清晰，举例中没有调用 coalesce 函数在两个字段中查找包含 NULL 的行。
- 以上例子均在没有索引的情况下进行查询。对于大多数应用程序来说，这个方法很慢。因此除了偶尔的特定搜索，文本搜索在实际使用中通常需要创建索引。

4.7.3.3.索引使用约束

下面是一个使用索引的例子：

```
create table table1 (c_int int,c_bigint bigint,c_varchar varchar,c_text text) with(orie
ntation=row);
create text search configuration ts_conf_1(parser=POUND);
create text search configuration ts_conf_2(parser=POUND) with(split_flag='%');
set default_text_search_config='ts_conf_1';
create index idx1 on table1 using gin(to_tsvector(c_text));
set default_text_search_config='ts_conf_2';
create index idx2 on table1 using gin(to_tsvector(c_text));
```

查询测试

```
select c_varchar,to_tsvector(c_varchar) from table1 where to_tsvector(c_text) @@ p
lainto_tsquery('￥#@……&**') and to_tsvector(c_text) @@ plainto_tsquery('某公司 ')
 and c_varchar is not null order by 1 desc limit 3;
```

该例子的关键点是表 table1 的同一个列 c_text 上建立了两个 gin 索引：idx1 和 idx2，但这两个索引是在不同 default_text_search_config 的设置下建立的。该例子和同一张表的同一个列上建立普通索引的不同之处在于：

- gin 索引使用了不同的 parser（即分隔符不同），那么 idx1 和 idx2 的索引数据是不同的；
- 在同一张表的同一个列上建立的多个普通索引的索引数据是相同的。

因此当执行同一个查询时，使用 idx1 和 idx2 查询出的结果是不同的。

使用约束

通过上面的例子，索引使用满足如下条件时：

- 在同一个表的同一个列上建立了多个 gin 索引；
- 这些 gin 索引使用了不同的 parser（即分隔符不同）；
- 在查询中使用了该列，且执行计划中使用索引进行扫描；

为了避免使用不同 gin 索引导致查询结果不同的问题，需要保证在物理表的一列上只有一个 gin 索引可用。

4.7.4.测试和调试文本搜索

4.7.4.1.分词器测试

功能描述

函数 ts_debug 允许简单测试文本搜索分词器。

语法格式

```
ts_debug([ config regconfig, ] document text,
    OUT alias text,
    OUT description text,
    OUT token text,
    OUT dictionaries regdictionary[],
    OUT dictionary regdictionary,
    OUT lexemes text[])
    returns setof record
```

参数说明

ts_debug 为文本解析器标识的每个 token 返回一行记录。记录中的列分别是：

- alias：text 类型，token 的别名。
- description：text 类型，token 的描述。
- token：text 类型，token 的文本内容。
- dictionaries：regdictionary 数组类型，是分词器为 token 选定的词典。
- dictionary：regdictionary 类型，用来识别 token 的词典。如果为空，则不做识别。

- lexemes：text 数组类型，词典识别 token 时生成的词素。如果为空，则不生成词素。空数组（{}）意味着 token 将被识别成停用词。

注意事项

ts_debug 显示 document 的每个 token 信息，token 是由解析器生成，由指定的词典进行处理。如果忽略对应参数，则使用 config 指定的分词器或者 default_text_search_config 指定的分词器。

示例

```
SELECT * FROM ts_debug('english','a fat  cat sat on a mat - it ate a fat rats');
```

结果返回如下：

```
 alias |  description  | token |  dictionaries  |  dictionary  | lexemes
-----------+-----------------+-------+----------------+--------------+---------
asciiword | Word, all ASCII | a     | {english_stem} | english_stem | {}
blank     | Space symbols   |       | {}             |              |
asciiword | Word, all ASCII | fat   | {english_stem} | english_stem | {fat}
blank     | Space symbols   |       | {}             |              |
asciiword | Word, all ASCII | cat   | {english_stem} | english_stem | {cat}
blank     | Space symbols   |       | {}             |              |
asciiword | Word, all ASCII | sat   | {english_stem} | english_stem | {sat}
blank     | Space symbols   |       | {}             |              |
asciiword | Word, all ASCII | on    | {english_stem} | english_stem | {}
blank     | Space symbols   |       | {}             |              |
asciiword | Word, all ASCII | a     | {english_stem} | english_stem | {}
blank     | Space symbols   |       | {}             |              |
asciiword | Word, all ASCII | mat   | {english_stem} | english_stem | {mat}
blank     | Space symbols   |       | {}             |              |
blank     | Space symbols   | -     | {}             |              |
asciiword | Word, all ASCII | it    | {english_stem} | english_stem | {}
blank     | Space symbols   |       | {}             |              |
asciiword | Word, all ASCII | ate   | {english_stem} | english_stem | {ate}
blank     | Space symbols   |       | {}             |              |
asciiword | Word, all ASCII | a     | {english_stem} | english_stem | {}
blank     | Space symbols   |       | {}             |              |
```

```
asciiword | Word, all ASCII | fat  | {english_stem} | english_stem | {fat}
blank   | Space symbols |    | {}      |       |
asciiword | Word, all ASCII | rats  | {english_stem} | english_stem | {rat}
(24 rows)
```

4.7.4.2.解析器测试

❖ 函数 ts_parse 可以直接测试文本搜索解析器。

```
ts_parse(parser_name text, document text,
        OUT tokid integer, OUT token text) returns setof record
```

ts_parse 解析指定的 document 并返回一系列的记录，一条记录代表一个解析生成的 token。每条记录包括标识 token 类型的 tokid 及 token 文本。例如：

```
SELECT * FROM ts_parse('default', '123 - a number');
```

结果返回如下：

```
tokid | token
-------+--------
   22 | 123
   12 |
   12 | -
    1 | a
   12 |
    1 | number
(6 rows)
```

❖ 函数 ts_token_type 返回指定解析器的 token 类型及其描述信息。

```
ts_token_type(parser_name text, OUT tokid integer,
            OUT alias text, OUT description text) returns setof record
```

ts_token_type 返回一个表，这个表描述了指定解析器可以识别的每种 token 类型。

对于每个 token 类型，表中给出了三个字段：

❖ tokid：用于解析器标记对应的 token 类型。

❖ alias：命名分词器命令中的 token 类型。

❖ description：简单描述。

```
SELECT * FROM ts_token_type('default');
```

结果返回如下：

```
 tokid |    alias    |      description
-------+-----------------+-----------------------------------------
  1 | asciiword    | Word, all ASCII
  2 | word      | Word, all letters
  3 | numword     | Word, letters and digits
  4 | email     | Email address
  5 | url      | URL
  6 | host      | Host
  7 | sfloat     | Scientific notation
  8 | version     | Version number
  9 | hword_numpart  | Hyphenated word part, letters and digits
 10 | hword_part    | Hyphenated word part, all letters
 11 | hword_asciipart | Hyphenated word part, all ASCII
 12 | blank     | Space symbols
 13 | tag      | XML tag
 14 | protocol     | Protocol head
 15 | numhword     | Hyphenated word, letters and digits
 16 | asciihword    | Hyphenated word, all ASCII
 17 | hword     | Hyphenated word, all letters
 18 | url_path     | URL path
 19 | file      | File or path name
 20 | float     | Decimal notation
 21 | int      | Signed integer
 22 | uint      | Unsigned integer
 23 | entity     | XML entity
(23 rows)
```

4.7.5.词典

4.7.5.1.词典概述

词典用于定义停用词（stop words），即全文检索时不搜索哪些词。

词典还可以用于对同一个词的不同形式进行规范化，这样同一个词的不同派生形式都可以进行匹配。规范化后的词称为词位（lexeme）。

除了提高检索质量外，词的规范化和删除停用词可以减少文档 tsvector 格式的大小， 从而提高性能。词的规范化和删除停用词并不总是具有语言学意义，用户可以根据应用环境在词典定义文件中自定义规范化和删除规则。

一个词典是一个程序，接收标记（token）作为输入，并返回：

- 如果 token 在词典中已知，返回对应 lexeme 数组（注意，一个标记可能对应多个 lexeme）。
- 一个 lexeme。一个新 token 会代替输入 token 被传递给后继词典（当前词典可被称为过滤词典）。
- 如果 token 在词典中已知，但它是一个停用词，返回空数组。
- 如果词典不能识别输入的 token，返回 NULL。

PanWeiDB 提供了多种语言的预定义字典，同时提供了五种预定义的词典模板，分别是 Simple，Synonym，Thesaurus，Ispell 和 Snowball，可用于创建自定义参数的新词典。

在使用全文检索时，建议用户：

- 可以在文本搜索配置中定义一个解析器，以及一组用于处理该解析器的输出标记词典。对于解析器返回的每个标记类型，可以在配置中指定不同的词典列表进行处理。当解析器输出一种类型的标记后，在对应列表的每个字典中会查阅该标记，直到某个词典识别它。如果它被识别为一个停用词，或者没有任何词典识别，该 token 将被丢弃，即不被索引或检索到。通常情况下，第一个返回非空结果的词典决定了最终结果，后继词典将不会继续处理。但是一个过滤类型的词典可以依据规则替换输入 token，然后将替换后的 token 传递给后继词典进行处理。

- 配置字典列表的一般规则是，第一个位置放置一个应用范围最小的、最具体化定义的词典，其次是更一般化定义的词典， 最后是一个普适定义的词典，比如 Snowball 词干词典或 Simple 词典。在下面例子中，对于一个针对天文学的文本搜索配置 astro_en，可以定义标记类型 asciiword（ASCII 词）对应的词典列表为：天文术语的 Synonym 同义词词典，Ispell 英语词典和 Snowball 英语词干词典。

```
ALTER TEXT SEARCH CONFIGURATION astro_en
 ADD MAPPING FOR asciiword WITH astro_syn, english_ispell, english_stem;
```

过滤类型的词典可以放置在词典列表中除去末尾的任何地方，放置在末尾时是无效的。使用这些词典对标记进行部分规范化，可以有效简化后继词典的处理。

4.7.5.2.Simple 词典

功能描述

Simple 词典首先将输入标记转换为小写字母，然后检查停用词表。如果识别为停用词则返回空数组，即表示该标记会被丢弃。否则，输入标记的小写形式作为规范化后的 lexeme 返回。此外，Simple 词典可通过设置参数 Accept 为 false（默认值 true），将非停用词报告为未识别，传递给后继词典继续处理。

注意事项

- 大多数词典的功能依赖于词典定义文件，词典定义文件名仅支持小写字母、数字、下划线组合。
- 临时模式 pg_temp 下不允许创建词典。
- 词典定义文件的字符集编码必须为 UTF-8 格式。实际应用时，如果与数据库的字符编码格式不一致，在读入词典定义文件时会进行编码转换。
- 通常情况下，每个 session 仅读取词典定义文件一次，当且仅当在第一次使用该词典时。需要修改词典文件时，可通过 ALTER TEXT SEARCH DICTIONARY 命令进行词典定义文件的更新和重新加载。

示例

1、创建 Simple 词典。

```
CREATE TEXT SEARCH DICTIONARY public.simple_dict (
  TEMPLATE = pg_catalog.simple,
  STOPWORDS = english
);
```

其中，停用词表文件全名为 english.stop。关于创建 simple 词典的语法和更多参数，请参见章节“CREATE TEXT SEARCH DICTIONARY”。

2、使用 Simple 词典。

```
SELECT ts_lexize('public.simple_dict','YeS');
```

结果显示如下

```
 ts_lexize
-----------
 {yes}
(1 row)
 SELECT ts_lexize('public.simple_dict','The');
```

结果显示如下：

```
 ts_lexize
-----------
 {}
(1 row)
```

3、设置参数 ACCEPT=false，使 Simple 词典返回 NULL，而不是返回非停用词的小写形式。

```
ALTER TEXT SEARCH DICTIONARY public.simple_dict ( Accept = false );
```

查询验证。

```
SELECT ts_lexize('public.simple_dict','YeS');
```

结果显示如下：

```
 ts_lexize
-----------
(1 row)
```

查询验证。

```
SELECT ts_lexize('public.simple_dict','The');
```

结果显示如下：

```
 ts_lexize
-----------
 {}
(1 row)
```

4.7.5.3.Snowball 词典

Snowball 词典模板支持词干分析词典，基于 Martin Porter 的 Snowball 项目，内置有许多语言的词干分析算法。PanWeiDB 中预定义有多种语言的 Snowball 词典，可通过系统表章节“PG_TS_DICT”查看预定义的词干分析词典以及支持的语言词干分析算法。

无论是否可以简化，Snowball 词典将标示所有输入为已识别，因此它应当被放置在词典列表的最后。把 Snowball 词典放在任何其他词典前面会导致后继词典失效，因为输入 token 不会通过 Snowball 词典进入到下一个词典。

4.7.5.4.Synonym 词典

功能描述

Synonym 词典用于定义、识别 token 的同义词并转化，不支持词组（词组形式的同义词可用 Thesaurus 词典定义，详细请参见章节“Thesaurus 词典”）。

示例

1、Synonym 词典可用于解决语言学相关问题，例如，为避免使单词"Paris"变成"pari"，可在 Synonym 词典文件中定义一行"Paris paris"，并将该词典放置在预定义的 english_stem 词典之前。

```
SELECT * FROM ts_debug('english', 'Paris');
```

结果返回如下：

```
alias | description | token | dictionaries | dictionary | lexemes
```

```
-----------+-----------------+-------+---------------+--------------+---------
 asciiword | Word, all ASCII | Paris | {english_stem} | english_stem | {pari}
(1 row)
```

2、创建词典，FILEPATH 以 dicts 文件实际目录为准。

```
CREATE TEXT SEARCH DICTIONARY my_synonym (
 TEMPLATE = synonym,
 SYNONYMS = my_synonyms,
 FILEPATH = 'file:///home/dicts/'
);
```

3、修改词典。

```
ALTER TEXT SEARCH CONFIGURATION english
 ALTER MAPPING FOR asciiword
 WITH my_synonym, english_stem;
```

4、 查询验证。

```
SELECT * FROM ts_debug('english', 'Paris');
```

结果显示如下：

```
 alias  |  description  | token |    dictionaries    | dictionary | lexemes
-----------+-----------------+-------+--------------------------+------------+---------
 asciiword | Word, all ASCII | Paris | {my_synonym,english_stem} | my_synonym | {pari
s}
(1 row)
```

5、 查询验证。

```
SELECT * FROM ts_debug('english', 'paris');
```

结果显示如下：

```
 alias  |  description  | token |    dictionaries    | dictionary | lexemes
-----------+-----------------+-------+--------------------------+------------+---------
 asciiword | Word, all ASCII | Paris | {my_synonym,english_stem} | my_synonym | {pari
s}
(1 row)
```

6、修改词典。

```
ALTER TEXT SEARCH DICTIONARY my_synonym ( CASESENSITIVE=true);
```

7、查询验证。

```
SELECT * FROM ts_debug('english', 'Paris');
```

结果显示如下：

```
 alias |  description  | token |    dictionaries    | dictionary | lexemes
-----------+-----------------+-------+--------------------------+------------+---------
 asciiword | Word, all ASCII | Paris | {my_synonym,english_stem} | my_synonym | {pari
s}
(1 row)
```

8、查询验证 。

```
SELECT * FROM ts_debug('english', 'paris');
```

结果显示如下：

```
 alias |  description  | token |    dictionaries    | dictionary | lexemes
----------+-----------------+-------+--------------------------+------------+---------
 asciiword | Word, all ASCII | Paris | {my_synonym,english_stem} | my_synonym | {pari}

(1 row)
```

其中，同义词词典文件全名为 my_synonyms.syn，所在目录为当前连接数据库主节点的/home/dicts/（以实际路径为准）下。关于创建词典的语法和更多参数，请参见章节“ALTER TEXT SEARCH DICTIONARY”。

- 星号（*）可用于词典文件中的同义词结尾，表示该同义词是一个前缀。在 to_tsvector()中该星号将被忽略，但在 to_tsquery()中会匹配该前缀并对应输出结果（参照章节“附加功能->处理查询”）。

假设词典文件 synonym_sample.syn 内容如下：

```
postgres    pgsql
postgresql    pgsql
postgre pgsql
```

```
gogle  googl
indices index*
```

1、创建并使用词典。

```
CREATE TEXT SEARCH DICTIONARY syn (
 TEMPLATE = synonym,
 SYNONYMS = synonym_sample
);
```

2、验证结果。

```
SELECT ts_lexize('syn','indices');
```

结果返回如下：

```
 ts_lexize
-----------
 {index}
(1 row)
```

3、创建词典并修改。

```
CREATE TEXT SEARCH CONFIGURATION tst (copy=simple);
ALTER TEXT SEARCH CONFIGURATION tst ALTER MAPPING FOR asciiword WITH syn;
```

4、验证结果。

```
SELECT to_tsvector('tst','indices');
```

结果返回如下：

```
to_tsvector
-------------
 'index':1
(1 row)


 SELECT to_tsquery('tst','indices');
```

结果返回如下：

```
to_tsquery
------------
 'index':*
(1 row)
SELECT 'indexes are very useful'::tsvector;
```

结果返回如下：

```
    tsvector
--------------------------------
 'are' 'indexes' 'useful' 'very'
(1 row)
SELECT 'indexes are very useful'::tsvector @@ to_tsquery('tst','indices');
```

结果返回如下：

```
 ?column?
----------
 t
(1 row)
```

4.7.5.5.Thesaurus 词典

功能描述

Thesaurus 词典，也叫做分类词典（缩写为 TZ），是一组定义了词以及词组间关系的集合，包括广义词（BT）、狭义词（NT）、首选词、非首选词、相关词等。根据词典文件中的定义，TZ 词典用一个指定的短语替换对应匹配的所有短语，并且可选择保留原始短语进行索引。TZ 词典实际上是 Synonym 词典的一个扩展，增加了短语支持。

注意事项

- 由于 TZ 词典需要识别短语，所以在处理过程中必须保存当前状态并与解析器进行交互，以决定是否处理下一个 token 或是结束当前识别。此外，TZ 词典配置时需谨慎，如果设置 TZ 词典仅处理 asciiword 类型的 token，则类似 one 7 的分类词典定义将不会生效，因为 uint 类型的 token 不会传给 TZ 词典处理。

- 在索引期间要用到分类词典，因此分类词典参数中的任何变化都要求重新索引。对于其他大多数类型的词典来说，类似添加或删除停用词这种修改并不需要强制重新索引。

示例

1、创建一个名为 thesaurus_astro 的 TZ 词典。

以一个简单的天文学词典 thesaurus_astro 为例，其中定义了两组天文短语及其同义词如下：

```
supernovae stars : sn
crab nebulae : crab
```

执行如下语句创建 TZ 词典：

```
CREATE TEXT SEARCH DICTIONARY thesaurus_astro (
 TEMPLATE = thesaurus,
 DictFile = thesaurus_astro,
 Dictionary = pg_catalog.english_stem,
 FILEPATH = 'file:///home/dicts/'
);
```

其中，词典定义文件全名为 thesaurus_astro.ths，所在目录为当前连接数据库主节点的/home/dicts/下 。子词典 pg_catalog.english_stem 是预定义的 Snowball 类型的英语词干词典，用于规范化输入词，子词典自身相关配置（例如停用词等）不在此处显示。关于创建词典的语法和更多参数，请参考章节“CREATE TEXT SEARCH DICTIONARY”。

2、创建词典后，将其绑定到对应文本搜索配置中需要处理的 token 类型上：

```
ALTER TEXT SEARCH CONFIGURATION russian
 ALTER MAPPING FOR asciiword, asciihword, hword_asciipart
 WITH thesaurus_astro, english_stem;
```

3、使用 TZ 词典。

- 测试 TZ 词典。

ts_lexize 函数对于测试 TZ 词典作用不大，因为该函数是按照单个 token 处理输入。可以使用 plainto_tsquery、to_tsvector、to_tsquery 函数测试 TZ 词典，这些函数能够将输入分解成多个 token（to_tsquery 函数需要将输入加上引号）。

```
SELECT plainto_tsquery('russian','supernova star');
```

结果显示如下：

```
plainto_tsquery
-----------------
 'sn'
(1 row)
SELECT to_tsvector('russian','supernova star');
```

结果显示如下：

```
to_tsvector
-------------
 'sn':1
(1 row)
   SELECT to_tsquery('russian','''supernova star''');
```

结果显示如下：

```
to_tsquery
------------
 'sn'
(1 row)
```

其中，supernova star 匹配了词典 thesaurus_astro 定义中的 supernovae stars，这是因为在 thesaurus_astro 词典定义中指定了 Snowball 类型的子词典 english_stem，该词典移除了 e 和 s。

- 如果同时需要索引原始短语，只要将其同时放置在词典定义文件中对应定义的右侧即可（在 thesaurus_astro 文件中修改），如下：

```
supernovae stars : sn supernovae stars
```

数据库中执行以下语句修改词典并查询验证。

```
ALTER TEXT SEARCH DICTIONARY thesaurus_astro (
 DictFile = thesaurus_astro,
 FILEPATH = 'file:///home/dicts/');
 SELECT plainto_tsquery('russian','supernova star');
```

结果返回如下：

```
  plainto_tsquery
-----------------------------
'sn' & 'supernova' & 'star'
(1 row)
```

4.7.5.6.词典测试

功能描述

函数 ts_lexize 用于进行词典测试。

注意事项

ts_lexize(dict regdictionary, token text) returns text[]如果输入的 token 可以被词典识别，那么 ts_lexize 返回词素的数组；如果 token 可以被词典识别到它是一个停用词，则返回空数组；如果是一个不可识别的词则返回 NULL。

示例

```
SELECT ts_lexize('english_stem', 'stars');
```

结果返回如下：

```
ts_lexize
-----------
{star}
```

查询不可识别值。

```
SELECT ts_lexize('english_stem', 'a');
```

结果返回如下：

```
ts_lexize
```

```
-----------
{}
```

须知

ts_lexize 函数支持单一 token，不支持文本。

4.7.5.7.Ispell 词典

功能描述

Ispell 词典模板支持词法词典，它可以把一个词的各种语言学形式规范化成相同的词位。比如，一个 Ispell 英语词典可以匹配搜索词 bank 的词尾变化和词形变化，如 banking、banked、banks、banks'和 bank's 等。

PanWeiDB 不提供任何预定义的 Ispell 类型词典或词典文件。dict 文件和 affix 文件支持多种开源词典格式，包括 Ispell、MySpell 和 Hunspell 等。

示例

1、获取词典定义文件和词缀文件。

用户可以使用开源词典，直接获取的开源词典后缀名可能为.aff 和.dic，此时需要将扩展名改为.affix 和.dict。此外，对于某些词典文件，还需要使用下面的命令把字符转换成 UTF-8 编码，比如挪威语词典：

```
iconv -f ISO_8859-1 -t UTF-8 -o nn_no.affix nn_NO.aff
iconv -f ISO_8859-1 -t UTF-8 -o nn_no.dict nn_NO.dic
```

2、创建 Ispell 词典。

```
CREATE TEXT SEARCH DICTIONARY norwegian_ispell (
 TEMPLATE = ispell,
 DictFile = nn_no,
 AffFile = nn_no,
 FilePath = 'file:///home/dicts'
);
```

其中，词典文件全名为 nn_no.dict 和 nn_no.affix，所在目录为当前连接数据库主节点的/home/dicts/下 。关于创建词典的语法和更多参数，请参见章节“CREATE TEXT SEARCH DICTIONARY”。

3、使用 Ispell 词典进行复合词拆分。

```
SELECT ts_lexize('norwegian_ispell', 'sjokoladefabrikk');
```

结果显示如下：

```
  ts_lexize
---------------------
{sjokolade,fabrikk}
(1 row)
```

MySpell 不支持复合词，Hunspell 对复合词有较好的支持。PanWeiDB 仅持 Hunspell 中基本的复合词操作。通常情况下，Ispell 词典能够识别的词是一个有限集合，其后应该配置一个更广义的词典，例如一个可以识别所有词的 Snowball 词典。

4.7.5.8.配置示例

文本搜索配置（Text Search Configuration），指定了将文档转换成 tsvector 过程中所必需的组件：

- 解析器，用于把文本分解成标记 token；
- 词典列表，用于将每个 token 转换成词位 lexeme。

每次 to_tsvector 或 to_tsquery 函数调用时，都需要指定一个文本搜索配置来指定具体的处理过程。GUC 参数章节“default_text_search_config”指定了默认的文本搜索配置，当文本搜索函数中没有显式指定文本搜索配置参数时，将会使用该默认值进行处理。

PanWeiDB 中预定义有一些可用的文本搜索配置，用户也可创建自定义的文本搜索配置。此外，为了便于管理文本搜索对象，还提供有多个 gsql 元命令，可以显示有关文本搜索对象的信息（详细请参见《工具参考》中“gsql”章节）。

操作步骤

1、创建一个文本搜索配置 ts_conf，复制预定义的文本搜索配置 english。

```
CREATE TEXT SEARCH CONFIGURATION ts_conf ( COPY = pg_catalog.english );
```

2、创建 Synonym 词典。

假设同义词词典定义文件 pg_dict.syn 内容如下：

```
postgres   pg
pgsql    pg
postgresql  pg
```

执行如下语句创建 Synonym 词典：

```
CREATE TEXT SEARCH DICTIONARY pg_dict (
  TEMPLATE = synonym,
  SYNONYMS = pg_dict,
  FILEPATH = 'file:///home/dicts'
);
```

3、创建一个 Ispell 词典 english_ispell（词典定义文件来自开源词典）。

```
CREATE TEXT SEARCH DICTIONARY english_ispell (
 TEMPLATE = ispell,
 DictFile = english,
 AffFile = english,
 StopWords = english,
 FILEPATH = 'file:///home/dicts'
);
```

4、设置文本搜索配置 ts_conf，修改某些类型的 token 对应的词典列表。关于 token 类型的详细信息，请参见章节“解析器”。

```
ALTER TEXT SEARCH CONFIGURATION ts_conf
```

```
ALTER MAPPING FOR asciiword, asciihword, hword_asciipart,
        word, hword, hword_part
WITH pg_dict, english_ispell, english_stem;
```

5、在文本搜索配置中，选择设置不索引或搜索某些 token 类型。

```
ALTER TEXT SEARCH CONFIGURATION ts_conf
DROP MAPPING FOR email, url, url_path, sfloat, float;
```

6、使用文本检索调测函数 ts_debug()对所创建的词典配置 ts_conf 进行测试。

```
SELECT * FROM ts_debug('ts_conf', '
PostgreSQL, the highly scalable, SQL compliant, open source object-relational
database management system, is now undergoing beta testing of the next
version of our software.
');
```

7、可以设置当前 session 使用 ts_conf 作为默认的文本搜索配置。此设置仅在当前 session 有效。

```
\dF+ ts_conf
```

结果返回如下：

```
 Text search configuration "public.ts_conf"
Parser: "pg_catalog.default"
  Token    |      Dictionaries
-----------------+-----------------------------------
 asciihword     | pg_dict,english_ispell,english_stem
 asciiword      | pg_dict,english_ispell,english_stem
 file       | simple
 host       | simple
 hword      | pg_dict,english_ispell,english_stem
 hword_asciipart | pg_dict,english_ispell,english_stem
 hword_numpart  | simple
 hword_part     | pg_dict,english_ispell,english_stem
 int        | simple
 numhword     | simple
```

```
numword    | simple
uint       | simple
version    | simple
word       | pg_dict,english_ispell,english_stem
SET default_text_search_config = 'public.ts_conf';
SHOW default_text_search_config;
 default_text_search_config
----------------------------
 public.ts_conf
(1 row)
```

4.7.5.9.停用词

功能描述

停用词是很常见的词，几乎出现在每一个文档中，并且没有区分值。因此，在全文搜索的语境下可忽视它们。停用词处理逻辑和词典类型相关。例如，Ispell 词典会先对标记进行规范化，然后再查看停用词表，而 Snowball 词典会最先检查输入标记是否为停用词。

示例

- 每个英文文本包含像 a 和 the 的单词，因此没必要将它们存储在索引中。然而，停用词影响 tsvector 中的位置，同时位置也会影响相关度。

```
SELECT to_tsvector('english','in the list of stop words');
```

结果显示如下：

```
   to_tsvector
----------------------------
 'list':3 'stop':5 'word':6
```

- 位置 1、2、4 是停用词，所以不显示。为包含和不包含停用词的文档计算出的排序是完全不同的：

```
panweidb=# SELECT ts_rank_cd (to_tsvector('english','in the list of stop words'), to_
tsquery('list & stop'));

 ts_rank_cd
------------
```

```
  .05
panweidb=# SELECT ts_rank_cd (to_tsvector('english','list stop words'), to_tsquery('l
ist & stop'));

 ts_rank_cd
------------
   .1
```

4.7.6.附加功能

4.7.6.1.查询重写

功能描述

ts_rewrite 函数族可以从 tsquery 中搜索一个特定的目标子查询，并在该子查询每次出现的地方都替换为另一个子查询。 实际上这只是通过字串替换而得到的一个特定 tsquery 版本。目标子查询和替换查询组合起来可以被认为是一个重写规则。一组类似的重写规则可以为搜索提供强大的帮助。例如，可以使用同义词扩大搜索范围（例如，new york, big apple, nyc, gotham）或限制搜索范围在用户直接感兴趣的热点话题上。

语法格式

- ts_rewrite 的这种形式只适用于一个单一的重写规则：任何出现目标子查询的地方都被无条件替换。

```
ts_rewrite (query tsquery, target tsquery, substitute tsquery) returns tsquery
```

- ts_rewrite 的这种形式接受一个起始查询和 SQL 查询命令。 这里的查询命令是文本字串形式，必须产生两个 tsquery 列。查询结果的每一行，第一个字段的值（目标子查询） 都会被第二个字段（替代子查询）替换。

```
ts_rewrite (query tsquery, select text) returns tsquery
```

说明

当多个规则需要重写时，重写顺序非常重要； 因此在实践中需要使用 ORDER BY 将源查询按照某些字段进行排序。

示例

举一个现实生活中天文学上的例子。我们将使用表驱动的重写规则扩大 supernovae 的查询范围：

1、创建测试表并插入数据。

```
CREATE TABLE tsearch.aliases (id int, t tsquery, s tsquery);
INSERT INTO tsearch.aliases VALUES(1, to_tsquery('supernovae'), to_tsquery('super
novae|sn'));
```

2、查询数据。

```
SELECT ts_rewrite(to_tsquery('supernovae & crab'), 'SELECT t, s FROM tsearch.aliase
s');
```

结果返回如下：

```
    ts_rewrite
--------------------------------
 'crab' & ( 'supernova' | 'sn' )
```

3、可以通过更新表修改重写规则。

```
UPDATE tsearch.aliases
SET s = to_tsquery('supernovae|sn & !nebulae')
WHERE t = to_tsquery('supernovae');
```

4、查询数据。

```
SELECT ts_rewrite(to_tsquery('supernovae & crab'), 'SELECT t, s FROM tsearch.aliase
s');
```

结果显示如下：

```
     ts_rewrite
--------------------------------------------
 'crab' & ( 'supernova' | 'sn' & !'nebula' )
```

5、需要重写的规则越多，重写操作就越慢。因为它要检查每一个可能匹配的规则。为了过滤明显的非候选规则，可以使用 tsquery 类型的操作符来实现。在下面的例子中， 我们只选择那些可能与原始查询匹配的规则。

```
SELECT ts_rewrite('a & b'::tsquery, 'SELECT t,s FROM tsearch.aliases WHERE ''a & b''::
tsquery @> t');
```

结果显示如下：

```
 ts_rewrite
------------
 'b' & 'a'
(1 row)
```

4.7.6.2.处理 tsvector

PanWeiDB 提供了用来操作 tsvector 类型的函数和操作符。

- tsvector || tsvector

  tsvector 连接操作符返回一个新的 tsvector 类型，它综合了两个 tsvector 中词素和位置信息，并保留词素的位置信息和权重标签。右侧的 tsvector 的起始位置位于左侧 tsvector 的最后位置，因此，新生成的 tsvector 几乎等同于将两个原始文档字串连接后进行 to_tsvector 操作。（这个等价是不准确的，因为任何从左边 tsvector 中删除的停用词都不会影响结果，但是，在使用文本连接时，则会影响词素在右侧 tsvector 中的位置。）

  相较于对文本进行连接后再执行 to_tsvector 操作，使用 tsvector 类型进行连接操作的优势在于，可以对文档的不同部分使用不同配置进行解析。因为 setweight 函数会对给定的 tsvector 中的语素进行统一设置，如果想要对文档的不同部分设置不同的权重，需要在连接之前对文本进行解析和权重设置。

- setweight(vector tsvector, weight "char") returns tsvector

  setweight 返回一个输入 tsvector 的副本，其中每一个位置都使用给定的权重做了标记。权值可以为 A、B、C 或 D（D 是 tsvector 副本的默认权重，并且不在输出中呈现）。当对 tsvector 进行连接操作时，这些权重标签将会被保留，文档不同部分以不同的权重进行排序。

- length(vector tsvector) returns integer

  返回 vector 中的词素的数量。

❖ strip(vector tsvector) returns tsvector

返回一个 tsvector 类型，其中包含输入的 tsvector 的同义词，但不包含任何位置和权重信息。虽然在相关性排序中，这里返回的 tsvector 要比未拆分的 tsvector 的作用小很多，但它通常都比未拆分的 tsvector 小的多。

注意事项

权重标签作用于位置，而不是词素。如果传入的 tsvector 已经被剥离了位置信息，那么 setweight 函数将什么都不做。

4.7.6.3.处理查询

PanWeiDB 提供了函数和操作符用来操作 tsquery 类型的查询。

❖ tsquery && tsquery

返回两个给定查询 tsquery 的与结果。

❖ tsquery || tsquery

返回两个给定查询 tsquery 的或结果。

❖ !! tsquery

返回给定查询 tsquery 的非结果。

❖ numnode(query tsquery) returns integer

返回 tsquery 中的节点数目（词素加操作符），这个函数在检查查询是否有效（返回值大于 0），或者只包含停用词（返回值等于 0）时，是有用的。例如：

```
SELECT numnode(plainto_tsquery('the any'));
```

结果显示如下：

```
NOTICE:  text-search query contains only stop words or doesn't contain lexemes, ig
nored
CONTEXT:  referenced column: numnode
 numnode
---------
```

```
 0
SELECT numnode('foo & bar'::tsquery);
```

结果显示如下：

```
 numnode
---------
  3
```

❖ querytree(query tsquery) returns text

返回可用于索引搜索的 tsquery 部分，该函数对于检测非索引查询是有用的（例如只包含停用词或否定项）。例如：

```
SELECT querytree(to_tsquery('!defined'));
```

结果显示如下

```
querytree
-----------
 T
(1 row)
```

4.7.6.4.收集文献统计

功能描述

函数 ts_stat 可用于检查配置和查找候选停用词。

语法格式

```
ts_stat(sqlquery text, [ weights text, ]
    OUT word text, OUT ndoc integer,
    OUT nentry integer) returns setof record
```

参数说明

❖ sqlquery：是一个包含 SQL 查询语句的文本，该 SQL 查询将返回一个 tsvector。

- ts_stat 执行 SQL 查询语句并返回一个包含 tsvector 中每一个不同的语素（词）的统计信息。返回信息包括：
  - word text：词素。
  - ndoc integer：词素在文档（tsvector）中的编号。
  - nentry integer：词素出现的频率。

示例

1、创建测试表并插入数据。

```
DROP SCHEMA IF EXISTS tsearch CASCADE;
CREATE SCHEMA tsearch;
CREATE TABLE tsearch.pgweb(id int, body text, title text, last_mod_date date);

INSERT INTO tsearch.pgweb VALUES(1, 'China, officially the People''s Republic of Ch
ina (PRC), located in Asia, is the world''s most populous state.', 'China', '2010-1-1');

INSERT INTO tsearch.pgweb VALUES(2, 'America is a rock band, formed in England i
n 1970 by multi-instrumentalists Dewey Bunnell, Dan Peek, and Gerry Beckley.', 'A
merica', '2010-1-1');
INSERT INTO tsearch.pgweb VALUES(3, 'England is a country that is part of the Unit
ed Kingdom. It shares land borders with Scotland to the north and Wales to the we
st.', 'England', '2010-1-1');
INSERT INTO tsearch.pgweb VALUES(4, 'Australia, officially the Commonwealth of A
ustralia, is a country comprising the mainland of the Australian continent, the islan
d of Tasmania, and numerous smaller islands.', 'Australia', '2010-1-1');
INSERT INTO tsearch.pgweb VALUES(6, 'Japan is an island country in East Asia.', 'Ja
pan', '2010-1-1');
INSERT INTO tsearch.pgweb VALUES(7, 'Germany, officially the Federal Republic of
Germany, is a sovereign state and federal parliamentary republic in central-wester
n Europe.', 'Germany', '2010-1-1');
INSERT INTO tsearch.pgweb VALUES(8, 'France, is a sovereign state comprising terri
tory in western Europe and several overseas regions and territories.', 'France', '201
0-1-1');
INSERT INTO tsearch.pgweb VALUES(9, 'Italy officially the Italian Republic, is a unita
ry parliamentary republic in Europe.', 'Italy', '2010-1-1');
```

```
INSERT INTO tsearch.pgweb VALUES(10, 'India, officially the Republic of India, is a c
ountry in South Asia.', 'India', '2010-1-1');
INSERT INTO tsearch.pgweb VALUES(11, 'Brazil, officially the Federative Republic of
 Brazil, is the largest country in both South America and Latin America.', 'Brazil', '20
10-1-1');
INSERT INTO tsearch.pgweb VALUES(12, 'Canada is a country in the northern half of
 North America.', 'Canada', '2010-1-1');
INSERT INTO tsearch.pgweb VALUES(13, 'Mexico, officially the United Mexican State
s, is a federal republic in the southern part of North America.', 'Mexico', '2010-1-1');
```

2、如果设置了权重条件，只有标记了对应权重的词素才会统计频率。例如，在一个文档集中检索使用频率最高的十个单词。

```
SELECT * FROM ts_stat('SELECT to_tsvector(''english'',title) FROM tsearch.pgweb WH
ERE id < 10') ORDER BY nentry DESC, ndoc DESC, word LIMIT 10;
```

结果显示如下：

```
  word   | ndoc | nentry
-----------+------+--------
 america  |   1 |     1
 australia |   1 |     1
 china    |   1 |     1
 england  |   1 |     1
 franc    |   1 |     1
 germani  |   1 |     1
 itali    |   1 |     1
 japan    |   1 |     1
(8 rows)
```

3、同样的情况，但是只计算权重为 A 或者 B 的单词使用频率。

```
SELECT * FROM ts_stat('SELECT to_tsvector(''english'', title) FROM tsearch.pgweb WH
ERE id < 10', 'a') ORDER BY nentry DESC, ndoc DESC, word LIMIT 10;
```

结果显示如下：

```
 word | ndoc | nentry
------+------+--------
```

```
(0 rows)
```

4.7.7.控制文本搜索

4.7.7.1.高亮搜索结果

功能描述

搜索结果的理想显示是：列出每篇文档中与搜索相关的部分，并标识为什么与查询相关。搜索引擎能够显示标识了搜索词的文档片段。PanWeiDB 提供了函数 ts_headline 支持这部分功能。

语法格式

```
ts_headline([ config regconfig, ] document text, query tsquery [, options text ]) retur
ns text
```

参数说明

- config：ts_headline 的输入是带有查询条件的文档，其返回文档中的摘录，在摘录中查询词是高亮显示的。用来解析文档的分词器由 config 参数指定。如果省略 config，则使用 default_text_search_config 的值所指定的分词器。
- options：指定 options 字符串时，需由一个或多个 option=value 对组成，且必须用逗号分隔。options 可以是下面的选项：
  - StartSel，StopSel：分隔文档中出现的查询词，以区别于其他摘录词。当包含有空格或逗号时，必须用双引号将字符串引起来。
  - MaxWords，MinWords：定义摘录的最长和最短值。
  - ShortWord：在摘录的开始和结束会丢弃此长度或更短的词。默认值 3 会消除常见的英语冠词。
  - HighlightAll：布尔标志。如果为真，整个文档将作为摘录。忽略前面三个参数的值。
  - MaxFragments：要显示的文本摘录或片段的最大数量。默认值 0 表示选择非片段的摘录生成方法。 大于 0 的值表示选择基于片段的摘录生成。此方法查找带有尽可能多查询词的文本片段，并显示查询词周围的上下文片段。因此，查询词临近每个片段的

中间，且查询词两边都有词。每个片段至多有 MaxWords，并且长度为 ShortWord 或更短的词在每一个片段开始和结束被丢弃。如果在文档中没有找到所有的查询词，则文档中开头将显示 MinWords 单片段。

❖ 不声明选项时，采用下面的缺省值：

```
StartSel=<b>, StopSel=</b>,
MaxWords=35, MinWords=15, ShortWord=3, HighlightAll=FALSE,
MaxFragments=0, FragmentDelimiter=" ... "
```

示例

```
SELECT ts_headline('english',
'The most common type of search
is to find all documents containing given query terms
and return them in order of their similarity to the
query.',
to_tsquery('english', 'query & similarity'));
```

结果返回如下

```
        ts_headline
------------------------------------------------------------

 containing given <b>query</b> terms
 and return them in order of their <b>similarity</b> to the
 <b>query</b>.
(1 row)
```

ts_headline 使用原始文档，而不是 tsvector 摘录，因此使用起来会慢，应慎重使用。

4.7.7.2.解析查询

功能描述

PanWeiDB 提供了函数 to_tsquery 和 plainto_tsquery 将查询转换为 tsquery 数据类型，to_tsquery 提供比 plainto_tsquery 更多的功能，但对其输入要求更严格。

语法格式

```
to_tsquery([ config regconfig, ] querytext text) returns tsquery
```

注意事项

to_tsquery 从 querytext 中创建一个 tsquery，querytext 必须由布尔运算符& (AND)，|(OR)和! (NOT)分割的单个 token 组成。这些运算符可以用圆括弧分组。换句话说，to_tsquery 输入必须遵循 tsquery 输入的通用规则，具体请参见“SQL 语法参考->数据类型->文本搜索类型”章节。不同的是基本 tsquery 以 token 表面值作为输入，而 to_tsquery 使用指定或默认分词器将每个 token 标准化成词素，并依据分词器丢弃属于停用词的 token。

示例

基本 tsquery 以 token 表面值作为输入，而 to_tsquery 使用指定或默认分词器将每个 token 标准化成词素，并依据分词器丢弃属于停用词的 token。比如：

```
SELECT to_tsquery('english', 'The & Fat & Rats');

to_tsquery
---------------
 'fat' & 'rat'
(1 row)
```

像在基本 tsquery 中的输入一样，weight(s)可以附加到每个词素来限制它只匹配那些有相同 weight(s)的 tsvector 词素。比如：

```
SELECT to_tsquery('english', 'Fat | Rats:AB');

 to_tsquery
------------------
 'fat' | 'rat':AB
(1 row)
```

同时，也可以附加到词素来指定前缀匹配：

```
SELECT to_tsquery('supern:*A & star:A*B');

to_tsquery
--------------------------
 'supern':*A & 'star':*AB
(1 row)
```

这样的词素将匹配 tsquery 中指定字符串和权重的项。

```
plainto_tsquery([ config regconfig, ] querytext text) returns tsquery
```

plainto_tsquery 将未格式化的文本 querytext 变换为 tsquery。类似于 to_tsvector，文本被解析并且标准化，然后在存在的词之间插入&(AND)布尔算子。比如：

```
SELECT plainto_tsquery('english', 'The Fat Rats');

 plainto_tsquery
-----------------
'fat' & 'rat'
(1 row)
```

请注意，plainto_tsquery 无法识别布尔运算符、权重标签，或在其输入中的前缀匹配标签：

```
SELECT plainto_tsquery('english', 'The Fat & Rats:C');

  plainto_tsquery
---------------------
 'fat' & 'rat' & 'c'
(1 row)
```

在这里，所有输入的标点符号作为空格符号丢弃。

4.7.7.3.解析文档

功能描述

PanWeiDB 中提供了 to_tsvector 函数把文档处理成 tsvector 数据类型。

语法格式

```
to_tsvector([ config regconfig, ] document text) returns tsvector
```

to_tsvector 函数

to_tsvector 将文本文档解析为 token，再将 token 简化到词素，并返回一个 tsvector。其中 tsvector 中列出了词素及它们在文档中的位置。文档是根据指定的或默认的文本搜索分词器进行处理的。

to_tsvector 函数内部调用一个解析器，将文档的文本分解成 token 并给每个 token 指定一个类型。对于每个 token，有一系列词典可供查询。词典系列因 token 类型的不同而不同。识别 token 的第一本词典将发出一个或多个标准词素来表示 token。例如：

- rats 变成 rat 因为词典认为词 rats 是 rat 的复数形式。
- 有些词被作为停用词（请参考“全文检索->词典->停用词”章节），这样它们就会被忽略，因为它们出现得太过频繁以致于搜索中没有用处。比如例子中的 a、on 和 it。
- 如果没有词典识别 token，那么它也被忽略。在这个例子中，符号“-”被忽略，因为词典没有给它分配 token 类型（空间符号），即空间记号永远不会被索引。

语法解析器、词典和要索引的 token 类型由选定的文本搜索分词器决定。可以在同一个数据库中有多种不同的分词器，以及提供各种语言的预定义分词器。在以上例子中，使用缺省分词器 english。

函数 setweight 可以给 tsvector 的记录加权重，权重是字母 A、B、C、D 之一。这通常用于标记来自文档不同部分的记录，比如标题、正文。之后，这些信息可以用于排序搜索结果。

因为 to_tsvector(NULL)会返回空，当字段可能是空的时候，建议使用 coalesce。

示例

- 示例 1：结果 tsvector 不包含词 a、on 或者 it，rats 变成 rat，并且忽略标点符号-。

```
SELECT to_tsvector('english', 'a fat  cat sat on a mat - it ate a fat rats');
```

结果显示如下：

```
                to_tsvector
-----------------------------------------------------
 'ate':9 'cat':3 'fat':2,11 'mat':7 'rat':12 'sat':4
(1 row)
```

- 示例 2：结构化文档创建 tsvector 的方法：

```
CREATE TABLE tsearch.tt (id int, title text, keyword text, abstract text, body text, ti ts
vector);

INSERT INTO tsearch.tt(id, title, keyword, abstract, body) VALUES (1, 'China', 'Beijing
', 'China','China, officially the People''s Republic of China (PRC), located in Asia, is t
he world''s most populous state.');

UPDATE tsearch.tt SET ti =
 setweight(to_tsvector(coalesce(title,'')), 'A')  ||
 setweight(to_tsvector(coalesce(keyword,'')), 'B') ||
 setweight(to_tsvector(coalesce(abstract,'')), 'C') ||
 setweight(to_tsvector(coalesce(body,'')), 'D');
```

结果显示如下：

```
id | title | keyword | abstract |                    body                    |


                ti
```

```
----+-------+---------+----------+-------------------------------------------------------
-------------------------------------------------------+------------------------------
------
-----------------------------------------------------------------------------------------
 1 | China | Beijing | China    | China, officially the People's Republic of China (PRC), l
ocated in Asia, is the world's most populous state. | 'asia':15 'beij':2B 'china':1A,3C,
4,1
1 'locat':13 'offici':5 'peopl':7 'popul':21 'prc':12 'republ':9 'state':22 'world':18
```

4.7.7.4.排序查询结果

功能描述

排序试图针对特定查询衡量文档的相关度，从而将众多的匹配文档中相关度最高的文档排在最前。PanWeiDB 提供了两个预置的排序函数。函数考虑了词法、距离和结构信息；也就是，他们考虑查询词在文档中出现的频率、紧密程度以及他们出现的地方在文档中的重要性。然而，相关性的概念是模糊的，并且是跟应用强相关的。不同的应用程序可能需要额外的信息来排序，比如，文档的修改时间、内置的排序函数等。也可以开发自己的排序函数或者采用附加因素组合这些排序函数的结果来满足特定需求。

预置排序函数

```
ts_rank([ weights float4[], ] vector tsvector, query tsquery [, normalization integer ])
returns float4
```

基于词素匹配率对 vector 进行排序：

```
ts_rank_cd([ weights float4[], ] vector tsvector, query tsquery [, normalization intege
r ]) returns float4
```

该函数需要位置信息的输入。因此它不能在“剥离”tsvector 值的情况下运行–它将总是返回零。

权重

- 对于这两个函数，可选的 weights 参数提供给词加权重的能力，词的权重大小取决于所加的权值。权重阵列指定在排序时为每类词汇加多大的权重。如果没有提供 weights，则使用缺省值：{0.1, 0.2, 0.4, 1.0}。

```
{D-weight, C-weight, B-weight, A-weight}
```

- 通常的权重是用来标记文档特殊领域的词，如标题或最初的摘要，所以相对于文章主体中的词它们有着更高或更低的重要性。
- 由于较长的文档有更多的机会包含查询词，因此有必要考虑文档的大小。例如，包含有 5 个搜索词的一百字文档比包含有 5 个搜索词的一千字文档相关性更高。两个预置的排序函数都采用了一个整型的标准化选项来定义文档长度是否影响排序及如何影响。这个整型选项控制多个行为，所以它是一个屏蔽字：可以使用|指定一个或多个行为（例如，2|4）。
  - 0（缺省）表示：跟长度大小没有关系
  - 1 表示：排名（rank）除以（文档长度的对数+1）
  - 2 表示：排名除以文档的长度
  - 4 表示：排名除以两个扩展词间的调和平均距离。只能使用 ts_rank_cd 实现
  - 8 表示：排名除以文档中单独词的数量
  - 16 表示：排名除以单独词数量的对数+1
  - 32 表示：排名除以排名本身+1
- 当指定多个标志位时，会按照所列的顺序依次进行转换。

注意事项

需要特别注意的是，排序函数不使用任何全局信息，所以不可能产生一个某些情况下需要的 1%或 100%的理想标准值。标准化选项 32 (rank/(rank+1))可用于所有规模的从零到一之间的排序，当然，这只是一个表面变化；它不会影响搜索结果的排序。

示例

创建测试表并插入数据。

```
DROP SCHEMA IF EXISTS tsearch CASCADE;
CREATE SCHEMA tsearch;
CREATE TABLE tsearch.pgweb(id int, body text, title text, last_mod_date date);
```

```
INSERT INTO tsearch.pgweb VALUES(1, 'China, officially the People''s Republic of Ch
ina (PRC), located in Asia, is the world''s most populous state.', 'China', '2010-1-1');

INSERT INTO tsearch.pgweb VALUES(2, 'America is a rock band, formed in England i
n 1970 by multi-instrumentalists Dewey Bunnell, Dan Peek, and Gerry Beckley.', 'A
merica', '2010-1-1');
INSERT INTO tsearch.pgweb VALUES(3, 'England is a country that is part of the Unit
ed Kingdom. It shares land borders with Scotland to the north and Wales to the we
st.', 'England', '2010-1-1');
INSERT INTO tsearch.pgweb VALUES(4, 'Australia, officially the Commonwealth of A
ustralia, is a country comprising the mainland of the Australian continent, the islan
d of Tasmania, and numerous smaller islands.', 'Australia', '2010-1-1');
INSERT INTO tsearch.pgweb VALUES(6, 'Japan is an island country in East Asia.', 'Ja
pan', '2010-1-1');
INSERT INTO tsearch.pgweb VALUES(7, 'Germany, officially the Federal Republic of
Germany, is a sovereign state and federal parliamentary republic in central-wester
n Europe.', 'Germany', '2010-1-1');
INSERT INTO tsearch.pgweb VALUES(8, 'France, is a sovereign state comprising terri
tory in western Europe and several overseas regions and territories.', 'France', '201
0-1-1');
INSERT INTO tsearch.pgweb VALUES(9, 'Italy officially the Italian Republic, is a unita
ry parliamentary republic in Europe.', 'Italy', '2010-1-1');
INSERT INTO tsearch.pgweb VALUES(10, 'India, officially the Republic of India, is a c
ountry in South Asia.', 'India', '2010-1-1');
INSERT INTO tsearch.pgweb VALUES(11, 'Brazil, officially the Federative Republic of
 Brazil, is the largest country in both South America and Latin America.', 'Brazil', '20
10-1-1');
INSERT INTO tsearch.pgweb VALUES(12, 'Canada is a country in the northern half of
 North America.', 'Canada', '2010-1-1');
INSERT INTO tsearch.pgweb VALUES(13, 'Mexico, officially the United Mexican State
s, is a federal republic in the southern part of North America.', 'Mexico', '2010-1-1');
```

❖ 仅选择排名前十的匹配：

```
SELECT id, title, ts_rank_cd(to_tsvector(body), query) AS rank
FROM tsearch.pgweb, to_tsquery('america') query
```

```
WHERE query @@ to_tsvector(body)
ORDER BY rank DESC
LIMIT 10;
```

结果显示如下：

```
 id |  title  | rank
----+---------+------
 11 | Brazil  | .2
  2 | America |  .1
 12 | Canada  |  .1
 13 | Mexico  |  .1
(4 rows)
```

❖ 使用标准化排序的相同例子：

```
SELECT id, title, ts_rank_cd(to_tsvector(body), query, 32 /* rank/(rank+1) */ ) AS ran
k
FROM tsearch.pgweb, to_tsquery('america') query
WHERE  query @@ to_tsvector(body)
ORDER BY rank DESC
LIMIT 10;
```

结果显示如下

```
 id |  title  |  rank
----+---------+----------
 11 | Brazil  | .166667
  2 | America | .0909091
 12 | Canada  | .0909091
 13 | Mexico  | .0909091
(4 rows)
```

❖ 使用中文分词法排序查询的例子：

1、创建测试表并插入数据。

```
CREATE TABLE ts_ngram(id int, body text);
INSERT INTO ts_ngram VALUES(1, '中文');
INSERT INTO ts_ngram VALUES(2, '中文检索');
INSERT INTO ts_ngram VALUES(3, '检索中文');
```

2、精确匹配 。

```
SELECT id, body, ts_rank_cd(to_tsvector('ngram',body), query) AS rank FROM ts_ngr
am, to_tsquery('中文') query WHERE query @@ to_tsvector(body);
```

结果显示如下：

```
id | body | rank
----+------+------
 1 | 中文 | .1
(1 row)
```

3、模糊匹配

```
SELECT id, body, ts_rank_cd(to_tsvector('ngram',body), query) AS rank FROM ts_ngr
am, to_tsquery('中文') query WHERE query @@ to_tsvector('ngram',body);
```

结果显示如下：

```
 id |  body  | rank
----+----------+------
 3 | 检索中文 | .1
 1 | 中文   | .1
 2 | 中文检索 | .1
(3 rows)
```

排序要遍历每个匹配的 tsvector，因此资源消耗多，可能会因为 I/O 限制导致排序慢。可是这是很难避免的，因为实际查询中通常会有大量的匹配。

4.7.8.解析器

功能描述

文本搜索解析器负责将原文档文本分解为多个 token，并标识每个 token 的类型。这里的类型集由解析器本身定义。注意，解析器并不修改文本，它只是确定合理的单词边界。由于这一限制，人们更需要定制词典，而不是为每个应用程序定制解析器。

目前 PanWeiDB 提供了三个内置的解析器，分别为 pg_catalog.default、pg_catalog.ngram、pg_catalog.pound，其中 pg_catalog.default 适用

于英文分词场景，pg_catalog.ngram、pg_catalog.pound 是为了支持中文全文检索功能新增的两种解析器，适用于中文及中英混合分词场景。

内置解析器 pg_catalog.default，它能识别 23 种 token 类型，如下表所示：

别名	描述	示例
asciiword	Word, all ASCII letters	elephant
word	Word, all letters	mañana
numword	Word, letters and digits	beta1
asciihword	Hyphenated word, all ASCII	up-to-date
hword	Hyphenated word, all letters	lógico-matemática
numhword	Hyphenated word, letters and digits	openGauss-beta1
hword_asciipart	Hyphenated word part, all ASCII	openGauss in the context openGauss-beta1
hword_part	Hyphenated word part, all letters	lógico or matemática in the context lógico-matemática

别名	描述	示例
hword_numpart	Hyphenated word part, letters and digits	beta1 in the context openGauss-beta1
email	Email address	foo@example.com
protocol	Protocol head	http://
url	URL	example.com/stuff/index.html
host	Host	example.com
url_path	URL path	/stuff/index.html, in the context of a URL
file	File or path name	/usr/local/foo.txt, if not within a URL
sfloat	Scientific notation	-1.23E+56
float	Decimal notation	-1.234
int	Signed integer	-1234
uint	Unsigned integer	1234

别名	描述	示例
version	Version number	8.3.0
tag	XML tag	<a href= “dictionaries.html” >
entity	XML entity	&
blank	Space symbols	(any whitespace or punctuation not otherwise recognized)

注意事项

- 对于解析器来说，一个“字母”的概念是由数据库的语言区域设置，即 lc_ctype 设置决定的。只包含基本 ASCII 字母的词被报告为一个单独的 token 类型，因为这类词有时需要被区分出来。大多数欧洲语言中，对 token 类型 word 和 asciiword 的处理方法是类似的。
- email 不支持某些由 RFC 5322 定义的有效电子邮件字符。具体来说，可用于 email 用户名的非字母数字字符仅包含句号、破折号和下划线。

解析器可能对同一内容进行重叠 token。例如，包含连字符的单词将作为一个整体进行报告，其组件也会分别被报告。

```
SELECT alias, description, token FROM ts_debug('english','foo-bar-beta1');
```

结果显示如下：

```
  alias   |      description       |  token
----------------+------------------------------------------+---------------
numhword    | Hyphenated word, letters and digits    | foo-bar-beta1
hword_asciipart | Hyphenated word part, all ASCII     | foo
blank     | Space symbols            | -
```

```
hword_asciipart | Hyphenated word part, all ASCII     | bar
blank     | Space symbols             | -
hword_numpart  | Hyphenated word part, letters and digits | beta1
```

这种行为是有必要的，因为它支持搜索整个复合词和各组件。这里是另一个例子：

```
SELECT alias, description, token FROM ts_debug('english','http://example.com/stuff/index.html');
```

结果显示如下：

```
 alias  |  description |      token
----------+--------------+----------------------------
protocol | Protocol head | http://
url    | URL      | example.com/stuff/index.html
host   | Host     | example.com
url_path | URL path    | /stuff/index.html
```

N-gram 是一种机械分词方法，适用于无语义中文分词场景。N-gram 分词法可以保证分词的完备性，但是为了照顾所有可能，把很多不必要的词也加入到索引中，导致索引项增加。N-gram 支持中文编码包括 GBK、UTF-8。内置 6 种 token 类型，如表 2 所示。

表 2 token 类型

别名	描述
zh_words	chinese words
en_word	english word
numeric	numeric data
alnum	alnum string

别名	描述
grapsymbol	graphic symbol
multisymbol	multiple symbol

Pound 是一种固定格式分词方法，适用于无语意但待解析文本以固定分隔符分割开来的中英文分词场景。支持中文编码包括 GBK、UTF8，支持英文编码包括 ASCII。内置 6 种 token 类型，如表 3 所示；支持 5 种分隔符，如表 4 所示，在用户不进行自定义设置的情况下分隔符默认为“#”。Pound 限制单个 token 长度不能超过 256 个字符。

表 3 token 类型

别名	描述
zh_words	chinese words
en_word	english word
numeric	numeric data
alnum	alnum string
grapsymbol	graphic symbol
multisymbol	multiple symbol

表 4 分隔符类型

分隔符	描述
@	Special character
#	Special character
$	Special character
%	Special character
/	Special character

4.8.聚合函数嵌套

功能描述

PanWeiDB 支持聚合函数嵌套使用。其中内层聚合函数用于计算通过 group by 子句定义的分组的聚合结果，再将内层聚合函数返回的结果集使用外层聚合函数再进行一次聚合计算，从而返回整体的结果。

语法格式

```
aggregate_name1(aggregate_name2(expression[,...][order_by_clause]))
aggregate_name1(aggregate_name2(ALL expression [,...][order_by_clause]))
aggregate_name1(aggregate_name2(DISTINCT expression[,...][order_by_clause]))
aggregate_name1(aggregate_name2(*))
aggregate_name1(aggregate_name2([expression[,...]]))
```

参数说明

- aggregate_name1：预定义的聚合函数名称 1，内层函数。
- aggregate_name2：预定义的聚合函数名称 2，外层函数。
- expression：自身不包含聚合函数表达式的任意值表达式。

注意事项

聚合函数嵌套的用法约束有如下限制：

❖ 必须含有 group by 子句，否则报错。例如：

```
select avg(max(salary)) from empsalary;
```

❖ 查询列中同时包括嵌套聚合函数和非嵌套聚合函数时报错。例如：

```
select avg(max(salary)),max(salary) from empsalary group by department_id;
```

❖ 查询列中同时包括嵌套聚合函数和一般列时报错。例如：

```
select avg(max(salary)),salary from empsalary group by department_id;
```

示例

1、创建测试表 empsalary。

```
create table empsalary (
depname  varchar,
empno  int,
salary int
);
```

2、插入数据。

```
insert into empsalary values('develop',10,5200);
insert into empsalary values('develop',10,5200);
insert into empsalary values('saleseeee',10,5000);
insert into empsalary values('personnel11',5,500);
insert into empsalary values('personnel11',5,500);
insert into empsalary values('personnel11',5,500);
insert into empsalary values('personnel1',5,500);
insert into empsalary values('sales',1,5200);
insert into empsalary values('sales',1,5200);
insert into empsalary values('selaseee',1,5000);
```

3、调用聚合嵌套函数 1。

```
select max(max(Empno)),max(sum(salary)) from empsalary group by depname;
```

返回结果如下，则表示调用成功：

```
max | max
-----------
10  | 10400
(1 row)
```

4、调用聚合嵌套函数 2。

```
select max(sum(Empno)),max(sum(salary)) from empsalary group by depname;
```

返回结果如下，则表示调用成功：

```
 max |  max
-----+-------
  20 | 10400
(1 row)
```

4.9.普通表和分区表在线重建索引

功能描述

支持在线 rebuild 普通行存表索引和行存一级分区表索引，其中行存一级分区表索引包括本地分区索引和全局索引。

语法格式

```
ALTER INDEX index_name
REBUILD [ PARTITION partition_name ] CONCURRENTLY;

REINDEX INDEX CONCURRENTLY
index_name [ PARTITION partiotion_name ];
```

参数说明

- index_name：自定义的索引名称。
- CONCURRENTLY：并行的含义是不会阻塞 DML 和 DQL 语句。

注意事项

- 目前仅支持并行重建普通行存表和行存一级分区表。
- 支持行级一级分区表上 local 和 global 类型的主表索引重建。

- 支持行级一级分区表上 local 类型索引在某一特定分区上的分区索引的重建。
- 不支持行级一级分区表上 global 类型索引在某一特定分区上的重建。

示例

1、创建测试表和索引，并插入数据。

```
CREATE TABLE t_rebuild_row(id int,name varchar(20));
INSERT INTO t_rebuild_row values(1,'liuyi');
INSERT INTO t_rebuild_row values(2,'chener');
INSERT INTO t_rebuild_row values(3,'zhangsan');
INSERT INTO t_rebuild_row values(4,'lisi');
INSERT INTO t_rebuild_row values(5,'wangwu');
INSERT INTO t_rebuild_row values(6,'zhaoliu');
INSERT INTO t_rebuild_row values(7,'sunqi');
INSERT INTO t_rebuild_row values(8,'zhouba');
INSERT INTO t_rebuild_row values(9,'wujiu');
CREATE INDEX idx_row ON t_rebuild_row (id);
```

2、查看索引是否创建成功。

```
\d t_rebuild_row
```

返回结果如下，则表示索引创建成功：

```
  Table "public.t_rebuild_row"
 Column |  Type   | Modifiers
--------+---------+-----------
 id     | integer |
 name   | varchar(20) |
Indexes:
    "idx_row" btree (id) TABLESPACE pg_default
```

3、设置扫描开关。

```
set enable_seqscan =off;
```

4、检查执行计划是否为索引扫描。

```
explain select /*+tablescan(t_rebuild_row) */ count(*) from t_rebuild_row where id
<6;
explain select /*+indexonlyscan(t_rebuild_row idx_row )*/ count(*) from t_rebuild_
row where id <6;
```

两个执行计划分别为顺序扫描及索引扫描,返回结果为：

```
                          QUERY PLAN
-------------------------------------------------------------------------------------
 Aggregate  (cost=10000000000111.26..10000000000111.27 rows=1 width=8)
   ->  Seq Scan on t_rebuild_row  (cost=10000000000.00..10000000000111.25 rows=3
width=0)
         Filter: (id < 6)
(3 rows)

                          QUERY PLAN
-------------------------------------------------------------------------------------
 Aggregate  (cost=8.31..8.32 rows=1 width=8)
   ->  Index Only Scan using idx_row on t_rebuild_row  (cost=0.00..8.30 rows=3 widt
h=0)
         Index Cond: (id < 6)
(3 rows)
```

5、打开两个会话，会话 1 不断执行 DML 和 DQL 操作。

```
do language plpgsql $$
declare
begin
 for i in 5..100000 loop
   INSERT INTO t_rebuild_row values(10,'zhengshi');
    DELETE FROM t_rebuild_row WHERE id=10;
    perform name FROM t_rebuild_row WHERE id<6 order by name;
   commit;
 end loop;
end;
$$;
```

6、切换至会话 2，重建索引。

```
ALTER  INDEX idx_row REBUILD concurrently;
```

或

```
REINDEX INDEX CONCURRENTLY idx_row;
```

7、会话 2 中执行如下命令，检查索引是否创建成功(会话 1 执行过程中，会话 2 重建索引成功，且会话 1 不报错)。

```
explain select /*+tablescan(t_rebuild_row) */ count(*) from t_rebuild_row where id <6;
explain select /*+indexonlyscan(t_rebuild_row idx_row )*/ count(*) from t_rebuild_row where id <6;
```

两个执行计划分别为顺序扫描及索引扫描,返回结果如下则表示索引创建成功：

```
                        QUERY PLAN
-------------------------------------------------------------------------
 Aggregate  (cost=1390.03..1390.04 rows=1 width=8)
   ->  Seq Scan on t_rebuild_row  (cost=0.00..1332.40 rows=23051 width=0)
         Filter: (id < 6)
(3 rows)


                                QUERY PLAN
-------------------------------------------------------------------------------------------
------
 Aggregate  (cost=2637.27..2637.28 rows=1 width=8)
   ->  Index Only Scan using idx_row on t_rebuild_row  (cost=0.00..2579.64 rows=23051 width=0)
         Index Cond: (id < 6)
(3 rows)
```

8、在会话 2 中执行如下命令，检查两者的结果是否相同。

```
select /*+tablescan(t_rebuild_row) */ count(*) from t_rebuild_row where id <6;
select /*+indexonlyscan(t_rebuild_row idx_row )*/ count(*) from t_rebuild_row where id <6;
```

返回结果相同，均为：

```
count
-------
   5
(1 row)
```

9、还原扫描开关。

```
reset enable_seqscan ;
```

4.10.附录

4.10.1.GIN 索引

4.10.1.1.GIN 索引介绍

GIN（Generalized Inverted Index）通用倒排索引。设计为处理索引项为组合值的情况，查询时需要通过索引搜索出出现在组合值中的特定元素值。例如，文档是由多个单词组成，需要查询出文档中包含的特定单词。

使用 item 表示索引的组合值，key 表示一个元素值。GIN 用来存储和搜索 key，而不是 item。

GIN 索引存储一系列（key，posting list）键值对，这里的 posting list 是一组出现 key 的行 ID。由于每个 item 都可能包含多个 key，同一个行 ID 可能会出现在多个 posting list 中，而每个 key 值只被存储一次，所以在相同的 key 在 item 中出现多次的情况下，GIN 索引是非常简洁的。

因为 GIN 索引的访问方式不需要了解他的运行方式，所以 GIN 索引是通用的。GIN 索引使用为特殊数据类型定义的策略。策略定义了如何从索引选项和查询条件中抽出 key，以及如何确定在查询中包含某些 key 值的行是否实际满足查询条件。

4.10.1.2.GIN 索引扩展性

GIN 索引的接口实现了一个高层次的抽象，要求访问用户仅需要实现被访问数据类型的语义。GIN 层自身可以处理并发操作、记录日志、搜索树结构的任务。

定义 GIN 索引的访问方式所要做的事情就是实现多个用户定义的方法，这些方法定义了键在树中的行为、键与键之间的关系、需要索引的 item、能够使用索引的查询。简而言之，GIN 索引将扩展性与普遍性、代码重用、清晰的接口结合在了一起。

实现 GIN 索引的操作符类有如下四个方法：

- int compare(Datum a, Datum b)

  比较两个 key（不是索引的 item）然后返回一个小于零、零或大于零的值，分别表示第一个 key 小于、等于或大于第二个 key。NULL 不会被传入这个函数。

- Datum *extractValue(Datum itemValue, int32 *nkeys, bool **nullFlags)

  给定一个要被索引的 item，返回一个对应 key 的数组。返回 key 的数目必须存储在 *nkeys 中。如果任何 key 都可能为 NULL，还要分配包含 nkeys 个布尔元素的数组，将地址存储到 *nullFlags，并且根据需要设置 NULL 值。如果所有 key 都是非 NULL，可以让 nullFlags 保持为 NULL（他的初始值）。如果 item 不包含任何 key，返回值可以为 NULL。

- Datum *extractQuery(Datum query, int32 *nkeys, StrategyNumber n, bool **pmatch, Pointer** extra_data, bool **nullFlags, int32 *searchMode)

  给定一个被查询的值，返回一个对应的 key 的数组。也就是说，query 是可索引操作符右侧的值，而该操作符左侧是被索引的字段。 n 是操作符类中操作符的策略号。通常，extractQuery 需要参考 n 来决定 query 的数据类型以及抽取键值的方法。返回 key 的个数必须存放在 *nkeys 中。如果任何 key 都可能为 NULL，还要分配包含 nkeys 个布尔元素的数组，将地址存储到 *nullFlags，并且根据需要设置 NULL 值。如果所有 key 都是非 NULL 的，可以让 nullFlags 保持为 NULL（他的初始值）。如果 query 不包含任何 key，返回值可以为 NULL。

  searchMode 是一个输出参数，他允许 extractQuery 指定一些关于如何执行搜索的细节。如果设置 *searchMode 为

GIN_SEARCH_MODE_DEFAULT（这也是调用函数前此参数的初始化值），只有那些至少返回一个 key 的 item 才能被考虑作为候选匹配项。如果设置 searchMode 为 GIN_SEARCH_MODE_INCLUDE_EMPTY，除了包含至少一个匹配 key 的 item 之外，根本不包含任何 key 的 item 也被考虑作为候选匹配项。（这个模式对于实现像“是否是子集”这样的操作是有用的）如果设置*searchMode 为 GIN_SEARCH_MODE_ALL，索引中所有非 NULL 的 item 都被考虑作为候选匹配项，不管他们是否匹配返回 key 中的任何一个。

pmatch 是一个允许支持部分匹配的输出参数。如果使用此参数，extractQuery *必须分配有 nkeys 个布尔元素的数组，并把数组地址保存到* pmatch。如果需要部分匹配相应的 key，则数组的每个元素应该设置为 TRUE；如果不需要匹配，则设置为 FALSE。如果设置*pmatch 为 NULL，则假设 GIN 不需要部分匹配。在函数调用前这个值被初始化为 NULL，因此，对于不支持部分匹配的操作符类，可以忽略这个参数。

extra_data 是一个允许 extractQuery 以 consistent 和 comparePartial 的方式传递额外数据的输出参数。如果使用他，extractQuery *必须分配一个包含 nkeys 个 Pointer 元素的数组，并把数组地址保存到* extra_data，然后把他想附加的东西存储到各个独立的指针中。在函数调用前这个值初始化为 NULL，因此，对于不需要附加数据的操作符类，可以忽略这个参数。如果设置了*extra_data，那么以 consistent 方式传递整个数组，使用 comparePartial 方式传递适当的元素。

- bool consistent(bool check[], StrategyNumber n, Datum query, int32 nkeys, Pointer extra_data[], bool *recheck, Datum queryKeys[], bool nullFlags[])

如果被索引项满足 StrategyNumber 为 n 的查询操作符则返回 TRUE。这个函数并不直接访问被索引项的值，因为 GIN 并没有精确的把项目保存下来，但是需要知道从查询中提取的哪些键值出现在给定的被索引项中。 check 数组的长度是 nkeys，这个与 query 调用 extractQuery 函数返回的键值的数目相同。如果索引项包含了相应的查询键，check 数组中对应的元素值就是 TRUE。比如，如果(check[i] == TRUE)，那么

意味着 extractQuery 的结果数组的第 i 个键出现在索引项中。考虑可能会用到 consistent 方式，原始的 query 也被作为参数传入进来。与此相同的还有 extractQuery 函数返回的 queryKeys[]和 nullFlags[]。extra_data 是 extractQuery 函数返回的额外数据数组，如果没有的话就是 NULL。

当 extractQuery 在 queryKeys[]中返回一个 NULL 的键值，如果被索引项包含 NULL 键值，相应的 check[]中的元素是 TRUE。也就是说，check[]的语义很像 IS NOT DISTINCT FROM。如果需要知道是通常值匹配还是 NULL 匹配，consistent 函数可以检查相应的 nullFlags[]元素。

成功执行后，如果堆元组需要针对查询运算符进行重新检查，*recheck 需要设置为 TRUE，如果索引测试已经是精确的了，则设为 FALSE。也就是说，FALSE 的返回值确保堆元组不匹配这个查询；设置* recheck 为 FALSE 的 TRUE 的返回值确保堆元组匹配这个查询；设置*recheck 为 TRUE 的 TRUE 的返回值意味着堆元组可能匹配这个查询，因此需要通过直接对照原始索引项对查询运算符进行获取和重新检查。

GIN 操作符类可以可选地提供第五个函数。

- int comparePartial(Datum partial_key, Datum key, StrategyNumber n, Pointer extra_data)

比较一个部分匹配查询键和一个索引键。返回一个整型值，它的符号代表了不同的含义：小于 0 意味着索引键不匹配查询，但是索引扫描应该继续；0 意味着索引键匹配查询；大于 0 指示应该终止索引扫描，因为不可能再有更多的匹配。在需要确定何时结束扫描的语义的情况下，这里提供了生成部分一致查询的操作符的策略号 n。同样的，extra_data 是 extractQuery 生成的额外数据数组中的相应元素，如果没有对应的元素，则为 NULL。 NULL 的键永远不会被传入这个函数。

为了支持"部分匹配"查询，一个操作符类必须提供 comparePartial 方法，并且当遇到部分匹配查询时，他的 extractQuery 方法必须设置 pmatch 参数。

上面的各种 Datum 值的实际数据类型根据操作符类的不同而不同。传入到 extractValue 中的项目值总是操作符类的输入类型，所有的键值类型必须是这个类的 STORAGE 类型。传入到 extractQuery 和 consistent 的 query 参数的类型是由策略号识别的类成员操作符的右操作数的输入类型。他不需要和项目类型相同，只要可以从中抽取出正确类型的键值。

4.10.1.3.GIN 索引实现

在内部，GIN 索引包含一个在键上构造的 B-tree 索引，每个键是一个或多个被索引项的一个元素（比如，一个数组的一个成员）。并且页面上每个元组包含了堆指针的 B-tree 的一个指针（一个 posting tree），当列表小到足以和键值一起存储到一个索引元组中时，则是堆指针的一个简单列表（一个 posting list）。

多列 GIN 索引通过在组合值（列号、键值）上建立一个单个的 B-tree 实现。不同列的键值可以有不同的类型。

GIN 快速更新技术

由于倒排索引的本身特性影响，更新一个 GIN 索引可能会比较慢。插入或更新一个堆行可能导致许多往索引的插入。当对表执行 VACUUM 后，或者如果待处理实体的列表太大了（大于 work_mem），这些实体被使用和初始索引创建时用到的相同的 bulk 插入方法，移动到主要的 GIN 数据结构。即使把额外的 VACUUM 开销算进去，这也大大提升了 GIN 索引更新的速度。而且，这种额外开销的工作可以通过后台进程而不是前端查询来处理。

这种方法的主要缺点在于搜索时除了常规的索引还必须要扫描待处理实体的列表。因此，大的待处理实体的列表会显著的拖慢搜索。另一个缺点是，虽然大多数更新很快，但是一个导致待处理列表（pending list）变得“太大”的更新将引发一个立即清理，并因此比起其他更新会非常慢。恰当的使用 autovacuum 可以弱化这两个问题。

如果一致的响应时间（清理实体速度和更新速度的响应时间）比更新速度更重要，可以通过把 GIN 索引的存储参数 FASTUPDATE 设置为 off 而不使用待处理实体。详细请参考：SQL 语法参考->SQL 语法->CREATE INDEX 章节。

部分匹配算法

GIN 可以支持"部分匹配"查询。即：查询并不决定单个或多个键的一个精确的匹配，而是，可能的匹配落在一个合理的狭窄键值范围内（根据 compare 支持函数决定的键值排序顺序）。此时，extractQuery 方法并不返回一个用于精确匹配的键值，取而代之的是，返回一个要被搜索的键值范围的下边界，并且设置 pmatch 为 true。然后，使用 comparePartial 方式扫描这个键值范围。comparePartial 必须为一个相匹配的索引键返回 0，如果不匹配但依然在被搜索范围内时返回小于 0 的值，对超过可以匹配的范围的索引键则返回大于 0 的值。

4.10.1.4.GIN 提示与技巧

创建 vs 插入

由于可能要为每个项目插入很多键，所以 GIN 索引的插入可能比较慢。对于向表中大量插入的操作，我们建议先删除 GIN 索引，在完成插入之后再重建索引。与 GIN 索引创建、查询性能相关的 GUC 参数如下：

- maintenance_work_mem

  GIN 索引的构建时间对 maintenance_work_mem 的设置非常敏感。

- work_mem

  在向启用了 FASTUPDATE 的 GIN 索引执行插入操作的期间，只要待处理实体列表的大小超过了 work_mem，系统就会清理这个列表。为了避免可观察到的响应时间的大起大落，让待处理实体列表在后台被清理是比较合适的（比如通过 autovacuum）。前端清理操作可以通过增加 work_mem 或者执行 autovacuum 来避免。然而，扩大 work_mem 意味着如果发生了前端清理，那么他的执行时间将更长。

- gin_fuzzy_search_limit

  开发 GIN 索引的主要目的是为了让 PanWeiDB 支持高度可伸缩的全文索引，并且常常会遇见全文索引返回海量结果的情形。而且，这经常发生在查询高频词的时候，因而这样的结果集没什么用处。因为从磁盘读取大量记录并对其进行排序会消耗大量资源，这在产品环境下是不能接受的。为了控制这种情况，GIN 索引有一个可配置的返回结果行数的软上限的配置参数 gin_fuzzy_search_limit。缺省值 0 表示没有限制。如果设置了非零值，那么返回结果就是从完整结果集中随机选择的一部分。

"软上限"的意思是返回结果的实际数量可能与指定的限制有偏差，这取决于查询和系统随机数生成器的质量。

4.10.2.扩展函数

下表列举了 PanWeiDB 中支持的扩展函数，不作为商用特性交付，仅供参考。

分类	函数名称	描述
访问权限查询函数	cm_sequence_privilege(user, sequence, privilege)	指定用户是否有访问序列的权限
	cm_sequence_privilege(sequence, privilege)	当前用户是否有访问序列的权限
触发器函数	pg_get_triggerdef(oid)	为触发器获取 CREATE [CONSTRAINT] TRIGGER 命令
	pg_get_triggerdef(oid, boolean)	为触发器获取 CREATE [CONSTRAINT] TRIGGER 命令

4.10.3.扩展语法

PanWeiDB 提供的扩展语法如下。

表 1 扩展 SQL 语法

类别	语法关键字	描述
创建表 CREATE TABLE	column_constraint: REFERENCES reftable [(refcolumn)] [MATCH FULL | MATCH PARTIAL | MATCH SIMPLE][ON DELETE action] [ON UPDATE action]	支持用 REFERENCES reftable[(refcolumn)] [MATCH FULL |MATCH PARTIAL | MATCH SIMPLE] [ON DELETE action] [ON UPDATE action] 为表创建外键约束。
加载	CREATE EXTENSION	把一个新的模块（例如 DBLINK）加

类别	语法关键字	描述
模块		载进当前数据库中。
	DROP EXTENSION	删除已加载的模块。
聚集函数	CREATE AGGREGATE	定义一个新的聚集函数。
	ALTER AGGREGATE	修改一个聚集函数的定义。
	DROP AGGREGATE	删除一个现存的聚集函数。

5. 系统表和系统视图

本章主要介绍 PanWeiDB 的系统表和系统视图。

5.1.系统表和系统视图概述

系统表是 PanWeiDB 存放结构元数据的地方，它是 PanWeiDB 数据库系统运行控制信息的来源，是数据库系统的核心组成部分。

系统视图提供了查询系统表和访问数据库内部状态的方法。

系统表和系统视图要么只对管理员可见，要么对所有用户可见。后续的系统表和系统视图章节有些标识了需要管理员权限，这些系统表和系统视图只有管理员可以查询。

用户可以删除后重新创建这些表、增加列、插入和更新数值，但是用户修改系统表会导致系统信息的不一致，从而导致系统控制紊乱。正常情况下不应该由用户手工修改系统表或系统视图，或者手工重命名系统表或系统视图所在的模式，而是由 SQL 语句关联的系统表操作自动维护系统表信息。

【须知】
不建议用户修改系统表和系统视图的权限。
用户应该禁止对系统表进行增删改等操作，人为对系统表的修改或破坏可能会导致系统各种异常情况甚至 openGauss 不可用。

5.2.使用系统表和系统视图

除了创建的表以外，数据库还包含很多系统表。这些系统表包含 PanWeiDB 安装信息以及 PanWeiDB 上运行的各种查询和进程的信息。可以通过查询系统表来收集有关数据库的信息。

系统表和系统视图中指出了表是对所有用户可见还是只对初始化用户可见。必须以初始化用户身份登录才能查询只对初始化用户可见的表。

PanWeiDB 提供了以下类型的系统表和视图：

- 继承自 PG 的系统表和视图：这类系统表和视图具有 PG 前缀。

❖ 继承自 openGauss 的系统表和视图：这类系统表和视图具有 GS 前缀。

5.2.1.查看数据库中包含的表

在 PG_TABLES 系统表中查看 public schema 中包含的所有表。

```
SELECT distinct(tablename) FROM pg_tables WHERE SCHEMANAME = 'public';
```

结果如下所示（示例仅供参考，具体返回结果请以实际情况为准）：

```
   tablename
------------------
 err_hr_staffs
 test
 err_hr_staffs_ft3
 web_returns_p1
 mig_seq_table
 films4
(6 rows)
```

5.2.2.查看数据库用户

通过 PG_USER 可以查看数据库中所有用户的列表，还可以查看用户 ID（USESYSID）和用户权限。

```
SELECT * FROM pg_user;
```

结果如下所示（示例仅供参考，具体返回结果请以实际情况为准）：

```
 usename  | usesysid | usecreatedb | usesuper | usecatupd | userepl |  passwd  | valbe
gin | valuntil |   respool    | parent | spacelimit | useconfig | nodegroup | tempspaceli
mit | spillspacelimit | u
semonitoradmin | useoperatoradmin | usepolicyadmin
----------+----------+-------------+----------+-----------+---------+----------+-------
--+----------+-------------+--------+------------+-----------+-----------+------------
---+-----------------+--
--------------+-----------------+---------------
 user2    |   53382 | f       | f     | f     | f     | ******** |       |       | default_pool |    0 |
       |       |       |         |          | f
```

```
       |f         |f
user1  |  53378|f      |f    |f     |f    |********|      |      |default_pool|    0|
       |      |      |         |         |f
       |f         |f
user9  |  20593|f      |f    |f     |f    |********|      |      |default_pool|    0|
       |      |      |         |         |f
       |f         |f
user8  |  20589|f      |f    |f     |f    |********|      |      |default_pool|    0|
       |      |      |         |         |f
       |f         |f
user7  |  20585|f      |f    |f     |f    |********|      |      |default_pool|    0|
       |      |      |         |         |f
       |f         |f
user6  |  20581|f      |f    |f     |f    |********|      |      |default_pool|    0|
       |      |      |         |         |f
       |f         |f
user5  |  20577|f      |f    |f     |f    |********|      |      |default_pool|    0|
       |      |      |         |         |f
       |f         |f
user4  |  20573|f      |f    |f     |f    |********|      |      |default_pool|    0|
       |      |      |         |         |f
       |f         |f
user3  |  20569|f      |f    |f     |f    |********|      |      |default_pool|    0|
       |      |      |         |         |f
       |f         |f
pwaudit|    34|f      |f    |f     |f    |********|      |      |default_pool|    0|
       |      |      |         |         |f
       |f         |f
pwadmin|    33|t      |t    |t     |t    |********|      |      |default_pool|    0
|      |      |      |         |         |f
       |f         |f
pwsso  |    35|f      |f    |f     |f    |********|      |      |default_pool|    0|
     |      |      |         |         |f
       |f         |f
```

```
panweidb |    10 | t       | t     | t     | t    | ******** |       |       | default_pool |    0
|       |       |       |       |          |          | t
        | t          | t
(13 rows)
```

5.2.3.查看和停止正在运行的查询语句

通过视图 PG_STAT_ACTIVITY 可以查看正在运行的查询语句。方法如下：

1、设置参数 track_activities 为 on（当此参数为 on 时，数据库系统才会收集当前活动查询的运行信息。）。

```
SET track_activities = on;
```

2、查看正在运行的查询语句。以查看正在运行的查询语句所连接的数据库名、执行查询的用户、查询状态及查询对应的 PID 为例：

```
SELECT datname, usename, state,pid FROM pg_stat_activity;
```

结果如下所示（示例仅供参考，具体返回结果请以实际情况为准）：

```
 datname  | usename  | state  |     pid
----------+----------+--------+-----------------
 panweidb | panweidb | active | 139855352559360
 postgres | panweidb | idle   | 139855526815488
 postgres | panweidb | idle   | 139855510034176
 postgres | panweidb | active | 139855543596800
 postgres | panweidb | idle   | 139855925278464
 postgres | panweidb | idle   | 139855891715840
 postgres | panweidb | idle   | 139855942059776
 postgres | panweidb | active | 139856025966336
 postgres | panweidb | active | 139855908497152
 postgres | panweidb | idle   | 139855975622400
(10 rows)
```

如果 state 字段显示为 idle，则表明此连接处于空闲，等待用户输入命令。如果仅需要查看非空闲的查询语句，则使用如下命令查看：

```
SELECT datname, usename, state FROM pg_stat_activity WHERE state != 'idle';
```

结果类似如下这样：

```
datname  | usename  | state
----------+----------+--------
panweidb | panweidb | active
postgres | panweidb | active
postgres | panweidb | active
postgres | panweidb | active
(4 rows)
```

3、若需要取消运行时间过长的查询，通过 PG_TERMINATE_BACKEND 函数，根据线程 ID 结束会话。

```
SELECT PG_TERMINATE_BACKEND(139855975622400);
```

显示类似如下信息，表示结束会话成功。

```
pg_terminate_backend
----------------------
t
(1 row)
```

显示类似如下信息，表示用户执行了结束当前会话的操作。

```
FATAL:  terminating connection due to administrator command
FATAL:  terminating connection due to administrator command
```

【说明】

gsql 客户端使用 PG_TERMINATE_BACKEND 函数结束当前会话后台线程时，客户端不会退出而是自动重连。即还会返回 “The connection to the server was lost. Attempting reset: Succeeded.”

```
FATAL:  terminating connection due to administrator command
FATAL:  terminating connection due to administrator command
The connection to the server was lost. Attempting reset: Succeeded.
```

5.3.系统表

5.3.1.GS_ASP

GS_ASP 显示被持久化的 ACTIVE SESSION PROFILE 样本，该表只在系统库下查询，在用户库下查询无数据。

表 1 GS_ASP 字段

名称	类型	描述
sampleid	bigint	采样 ID。
sample_time	timestamp with time zone	采样的时间。
need_flush_sample	boolean	该样本是否需要刷新到磁盘。 ❖ t（true）：表示需要。 ❖ f（false）：表示不需要。
databaseid	oid	数据库 ID。
thread_id	bigint	线程的 ID。
sessionid	bigint	会话的 ID。
start_time	timestamp with time zone	会话的启动时间。
event	text	具体的事件名称。
lwtid	integer	当前线程的轻量级线程号。
psessionid	bigint	streaming 线程的父线程。
tlevel	integer	streaming 线程的层级。与执行计划的层级（id）相对应。
smpid	integer	smp 执行模式下并行线程的并行编号。
userid	oid	session 用户的 id。
application_name	text	应用的名字。
client_addr	inet	client 端的地址。
client_hostname	text	client 端的名字。
client_port	integer	客户端用于与后端通讯的 TCP 端口号。
query_id	bigint	debug query id。
unique_query_id	bigint	unique query id。
user_id	oid	unique query 的 key 中的 user_id。
cn_id	integer	表示下发该 unique sql 的节点 id。 unique query 的 key 中的 cn_id。
unique_query	text	规范化后的 Unique SQL 文本串。
locktag	text	会话等待锁信息，可通过 locktag_decode 解析。
lockmode	text	会话等待锁模式： ❖ LW_EXCLUSIVE：排他锁

名称	类型	描述
		❖ LW_SHARED：共享锁 ❖ LW_WAIT_UNTIL_FREE：等待 LW_EXCLUSIVE 可用
block_sessionid	bigint	如果会话正在等待锁，阻塞该会话获取锁的会话标识。
wait_status	text	描述 event 列的更多详细信息。
global_sessionid	text	全局会话 ID。
xact_start_time	timestamp with time zone	事务开始时间。
query_start_time	timestamp with time zone	语句开始执行时间。
state	text	当前事务状态。 可能取值为：active, idle in transaction, fastpath function call, idle in transaction (aborted), disabled, retrying。

5.3.2.GS_AUDITING_POLICY

GS_AUDITING_POLICY 系统表记录统一审计的主体信息，每条记录对应一个设计策略。需要有系统管理员或安全策略管理员权限才可以访问此系统表。

表 1 GS_AUDITING_POLICY 字段

名称	类型	描述
oid	oid	行标识符（隐藏属性，必须明确选择）。
polname	name	策略名称，需要唯一，不可重复。
polcomments	name	策略描述字段，记录策略相关的描述信息，通过 COMMENTS 关键字体现。
modifydate	timestamp without time zone	策略创建或修改的最新时间戳。
polenabled	boolean	用来表示策略启动开关。
pollevel	name	表示策略等级。

5.3.3.GS_AUDITING_POLICY_ACCESS

GS_AUDITING_POLICY_ACCESS 系统表记录与 DML 数据库相关操作的统一审计信息。需要有系统管理员或安全策略管理员权限才可以访问此系统表。

表 1GS_AUDITING_POLICY_ACCESS 字段

名称	类型	描述
oid	oid	行标识符（隐藏属性，必须明确选择）。
accesstype	name	DML 数据库操作相关类型。例如 SELECT、INSERT、DELETE 等。
labelname	name	资源标签名称。对应系统表 gs_auditing_policy 中的 polname 字段。
policyoid	oid	所属审计策略的 Oid，对应系统表 GS_AUDITING_POLICY 中的 oid。
modifydate	timestamp without time zone	创建或修改的最新时间戳。

5.3.4.GS_AUDITING_POLICY_FILTERS

GS_AUDITING_POLICY_FILTERS 系统表记录统一审计相关的过滤策略相关信息，每条记录对应一个设计策略。需要有系统管理员或安全策略管理员权限才可以访问此系统表。

表 1 GS_AUDITING_POLICY_FILTERS 字段

名称	类型	描述
oid	oid	行标识符（隐藏属性，必须明确选择）。
filtertype	name	过滤类型。目前值仅为 logical_expr。
labelname	name	名称。目前值仅为 logical_expr。
policyoid	oid	所属审计策略的 Oid，对应系统表 GS_AUDITING_POLICY 中的 oid。
modifydate	timestamp	创建或修改的最新时间戳。
logicaloperator	text	过滤条件的逻辑字符串。

5.3.5.GS_AUDITING_POLICY_PRIVILEGES

GS_AUDITING_POLICY_PRIVILEGES 系统表记录统一审计 DDL 数据库相关操作信息，每条记录对应一个设计策略。需要有系统管理员或安全策略管理员权限才可以访问此系统表。

表 1 GS_AUDITING_POLICY_PRIVI 字段

名称	类型	描述
oid	oid	行标识符（隐藏列，必须显式指定列名选择）。
privilegetype	name	DDL 数据库操作相关类型。例如 CREATE、ALTER、DROP 等。
labelname	name	资源标签名称。对应系统表 gs_auditing_policy 中的 polname 字段。
policyoid	oid	对应审计策略系统表 GS_AUDITING_POLICY 中的 oid。
modifydate	timestamp	创建或修改的最新时间戳。

5.3.6.GS_CLIENT_GLOBAL_KEYS

GS_CLIENT_GLOBAL_KEYS 系统表记录密态等值特性中客户端加密主密钥相关信息，每条记录对应一个客户端加密主密钥。

表 1 GS_CLIENT_GLOBAL_KEYS 字段

名称	类型	描述
oid	oid	行标识符（隐含字段）。
global_key_name	name	客户端加密主密钥（cmk）名称。
key_namespace	oid	包含此客户端加密主密钥（cmk）的命名空间 OID。
key_owner	oid	客户端加密主密钥（cmk）的所有者。
key_acl	aclitem[]	创建该密钥时所拥有的访问权限。
create_date	timestamp	创建密钥的时间。

5.3.7.GS_CLIENT_GLOBAL_KEYS_ARGS

GS_CLIENT_GLOBAL_KEYS_ARGS 系统表记录密态等值特性中客户端加密主密钥相关元数据信息，每条记录对应客户端加密主密钥的一个键值对信息。

表 1 GS_CLIENT_GLOBAL_KEYS_ARGS 字段

名称	类型	描述
oid	oid	行标识符（隐含字段）。
global_key_id	oid	客户端加密主密钥（cmk）oid。
function_name	name	值为 encryption。
key	name	客户端加密主密钥（cmk）的元数据信息对应的名称。
value	bytea	客户端加密主密钥（cmk）的元数据信息名称的值。

5.3.8.GS_COLUMN_KEYS

GS_COLUMN_KEYS 系统表记录密态等值特性中列加密密钥相关信息，每条记录对应一个列加密密钥。

表 1 GS_COLUMN_KEYS 字段

名称	类型	描述
oid	oid	行标识符（隐含字段）。
column_key_name	name	列加密密钥（cek）名称。
column_key_distributed_id	oid	根据加密密钥（cek）全称域名 hash 值得到的 id。
global_key_id	oid	外键。客户端加密主密钥（cmk）的 oid。
key_namespace	oid	包含此列加密密钥（cek）的命名空间 oid。
key_owner	oid	列加密密钥（cek）的所有者。
create_date	timestamp	创建列加密密钥的时间。
key_acl	aclitem[]	创建该列加密密钥时所拥有的访问权限。

5.3.9.GS_COLUMN_KEYS_ARGS

GS_COLUMN_KEYS_ARGS 系统表记录密态等值特性中客户端加密主密钥相关元数据信息，每条记录对应客户端加密主密钥的一个键值对信息。

表 1 GS_COLUMN_KEYS_ARGS 字段

名称	类型	描述
oid	oid	行标识符（隐含字段）。
column_key_id	oid	列加密密钥（cek）oid。
function_name	name	值为 encryption。
key	name	列加密密钥（cek）的元数据信息对应的名称。
value	bytea	列加密密钥（cek）的元数据信息名称的值。

5.3.10.GS_DB_PRIVILEGE

GS_DB_PRIVILEGE 系统表记录 ANY 权限的授予情况，每条记录对应一条授权信息。

表 1 GS_DB_PRIVILEGE 字段

名称	类型	描述
oid	oid	行标识符（隐含字段，必须明确选择）。
roleid	oid	用户标识。
privilege_type	text	用户拥有的 ANY 权限，取值参考 ANY 权限列表。
admin_option	boolean	是否具有 privilege_type 列记录的 ANY 权限的再授权权限。-t：表示具有。-f：表示不具有。

5.3.11.GS_ENCRYPTED_COLUMNS

GS_ENCRYPTED_COLUMNS 系统表记录密态等值特性中表的加密列相关信息，每条记录对应一条加密列信息。

表 1 GS_ENCRYPTED_COLUMNS 字段

名称	类型	描述
oid	oid	行标识符（隐含字段）。
rel_id	oid	表的 OID。

名称	类型	描述
column_name	name	加密列的名称。
column_key_id	oid	外键，列加密密钥的 OID。
encryption_type	int1	加密类型，取值为 2（DETERMINISTIC）或者 1（RANDOMIZED）。
data_type_original_oid	oid	加密列的原始数据类型 id，参考系统表 PG_TYPE 中的 oid。
data_type_original_mod	int4	加密列的原始数据类型修饰符，参考系统表 PG_ATTRIBUTE 中的 atttypmod。其值对那些不需要的类型 data_type_original_mod 通常为-1。
create_date	timestamp	创建加密列的时间。

5.3.12.GS_ENCRYPTED_PROC

GS_ENCRYPTED_PROC 系统表提供了密态函数/存储过程函数参数、返回值的原始数据类型，加密列等信息。

表 1 GS_ENCRYPTED_PROC 字段

名称	类型	描述
oid	oid	行标识符（隐含字段）。
func_id	oid	function 的 oid，对应 pg_proc 系统表中的 oid 行标识符。
prorettype_orig	int4	返回值的原始数据类型。
proargcachedcol	oidvector	函数 INPUT 参数对应的加密列的 oid，对应 gs_encrypted_columns 系统表中的 oid 行标识符。
proallargtypes_orig	oid[]	所有函数参数的原始数据类型。

5.3.13.GS_GLOBAL_CHAIN

GS_GLOBAL_CHAIN 系统表记录用户对防篡改用户表的修改操作信息，每条记录对应一次表级修改操作。具有审计管理员权限的用户可以查询此系统表，所有用户均不允许修改此系统表。

表 1 GS_GLOBAL_CHAIN 字段

名称	类型	描述
blocknum	bigint	区块号，当前用户操作在账本中记录的序号。
dbname	name	数据库名称。被修改的防篡改用户表所属的 database。
username	name	用户名，执行用户表修改操作的用户名。
starttime	timestamp with time zone	用户操作执行的最新时间戳。
relid	oid	用户表 Oid，被修改的防篡改用户表 Oid。
relnsp	name	模式 Oid，被修改的防篡改用户表所属的 namesapce oid。
relname	name	用户表名，被修改的防篡改用户表名。
relhash	hash16	当前操作产生的表级别 hash 变化量。
globalhash	hash32	全局摘要，由当前行信息与前一行 globalhash 计算而来，将整个表串联起来，用于验证 GS_GLOBAL_CHAIN 数据完整性。
txcommand	text	被记录操作的 SQL 语句。

5.3.14.GS_GLOBAL_CONFIG

GS_GLOBAL_CONFIG 记录了数据库实例初始化时，用户指定的参数值。除此之外，还存放了用户设置的弱口令，支持数据库初始用 ei 户通过 ALTER 和 DROP 语法对系统表中的参数进行写入、修改和删除。

表 1 GS_GLOBAL_CONFIG 字段

名称	类型	描述
name	name	数据库实例初始化时系统内置的指定参数名称。当前版本第一行默认为 buckets_len，第二行起存放弱口令名称。
value	text	数据库实例初始化时系统内置的指定参数值。当前版本第一行默认为 bucketmap 长度；第二行起存放弱口令。

5.3.15.GS_MASKING_POLICY

GS_MASKING_POLICY 系统表记录动态数据脱敏策略的主体信息，每条记录对应一个脱敏策略。需要有系统管理员或安全策略管理员权限才可以访问此系统表。

表 1 GS_MASKING_POLICY 表字段

名称	类型	描述
oid	oid	行标识符（隐藏列，必须显式指定列名选择）。
polname	name	策略名称，唯一不可重复。
polcomments	name	策略描述字段，记录策略相关的描述信息，通过 COMMENTS 关键字体现。
modifydate	timestamp	策略创建或修改的最新时间戳。
polenabled	Boolean	策略启动开关。

5.3.16.GS_MASKING_POLICY_ACTIONS

GS_MASKING_POLICY_ACTIONS 系统表记录动态数据脱敏策略中相应的脱敏策略包含的脱敏行为，一个脱敏策略对应着该表的一行或多行记录。需要有系统管理员或安全策略管理员权限才可以访问此系统表。

表 1 GS_MASKING_POLICY_ACTIONS 表字段

名称	类型	描述
oid	oid	行标识符（隐藏属性，必须明确选择）。
actiontype	name	脱敏函数，标识脱敏策略使用的脱敏函数。
actparams	name	向脱敏函数中传递的参数信息。
actlabelname	name	被脱敏的 label 名称。
policyoid	oid	该条记录所属的脱敏策略 oid，对应 GS_MASKING_POLICY 中的 oid。
modifydate	timestamp	该条记录创建或修改的最新时间戳。

5.3.17.GS_MASKING_POLICY_FILTERS

GS_MASKING_POLICY_FILTERS 系统表记录动态数据脱敏策略对应的用户过滤条件，当用户条件满足 FILTER 条件时，对应的脱敏策略才会生效。需要有系统管理员或安全策略管理员权限才可以访问此系统表。

表 1 GS_MASKING_POLICY_FILTERS 表字段

名称	类型	描述
oid	oid	行标识符（隐藏列，必须显式指定列名选择）。

名称	类型	描述
filtertype	name	过滤类型。目前值仅为 logical_expr。
filterlabelname	name	过滤范围。目前值仅为 logical_expr。
policyoid	oid	该条用户过滤条件所属的脱敏策略 oid，对应 GS_MASKING_POLICY 中的 oid。
modifydate	timestamp	该条用户过滤条件创建或修改的最新时间戳。
logicaloperator	text	过滤条件的波兰表达式。

5.3.18.GS_MATVIEW

GS_MATVIEW 系统表提供了关于数据库中每一个物化视图的信息。

表 1 GS_MATVIEW 字段

名称	类型	描述
oid	oid	行标识符（隐藏列，必须显式指定列名选择）。
matviewid	oid	物化视图的 oid。
mapid	oid	物化视图 map 表的 oid，map 表为物化视图关联表，与物化视图一一对应。全量物化视图不存在对应的 map 表，该字段为 0。
matnamespace	oid	物化视图所在的 namespace 的 OID。
jobid	integer	物化视图自动刷新的定时任务编号 job_id。
ivm	boolean	物化视图的类型，t 为增量物化视图，f 为全量物化视图。
needrefresh	boolean	保留字段。
refreshtime	timestamp without time zone	物化视图上一次刷新时间，若未刷新则为 null。仅对增量物化视图维护该字段，全量物化视图为 null。

5.3.19.GS_MATVIEW_DEPENDENCY

GS_MATVIEW_DEPENDENCY 系统表提供了关于数据库中每一个增量物化视图、基表和 mlog 表的关联信息。全量物化视图不存在与基表对应的 mlog 表，不会写入记录。

表 1 GS_MATVIEW_DEPENDENCY 字段

名称	类型	描述
oid	oid	行标识符（隐藏属性，必须明确选择）。
matviewid	oid	物化视图的 oid。
relid	oid	物化视图基表的 oid。
mlogid	oid	物化视图 mlog 表的 oid，mlog 表为物化视图日志表，与基表一一对应。
mxmin	int4	保留字段。

5.3.20.GS_MODEL_WAREHOUSE

GS_MODEL_WAREHOUSE 系统表用于存储 AI 引擎训练模型，其中包含模型，训练过程的详细描述。

表 1 GS_MODEL_WAREHOUSE 字段

名称	数据类型	描述
oid	oid	隐含列。
modelname	name	唯一约束。
modelowner	oid	模型拥有者的 OID。
createtime	timestamp	模型创建的时间。
processedtuples	int	训练涉及的元组数。
discardedtuples	int	未参加训练的不合格元组数。
preprocesstime	real	数据预处理时长。
exectime	real	训练时长。
iterations	int	迭代轮次。
outputtype	oid	模型输出的数据类型 OID。
modeltype	text	AI 算子的类型名称。
query	text	创建模型所执行的 query 语句。
modeldata	bytea	保存的二进制模型信息。

名称	数据类型	描述
weight	real[]	目前只适用于 GD 算子模型。
hyperparametersnames	text[]	涉及的超参名称。
hyperparametersvalues	text[]	超参所对应的取值。
hyperparametersoids	oid[]	超参对应的数据类型 OID。
coefnames	text[]	模型参数名称。
coefvalues	text[]	模型参数对应的取值。
coefoids	oid[]	模型参数对应的数据类型 OID。
trainingscoresname	text[]	度量模型性能方法的名称。
trainingscoresvalue	real[]	度量模型性能方法的数值。
modeldescribe	text[]	模型的描述信息。

5.3.21.GS_OPT_MODEL

GS_OPT_MODEL 是启用 AiEngine 执行计划时间预测功能时的数据表，记录机器学习模型的配置、训练结果、功能、对应系统函数、训练历史等相关信息。

表 1 GS_OPT_MODEL 字段

名称	类型	描述
oid	oid	数据库对象 id。
template_name	name	机器学习模型的模板名，决定训练和预测调用的函数接口，目前只实现了 rlstm，方便后续扩展。
model_name	name	模型的实例名，每个模型对应 aiEngine 在线学习进程中的一套参数、训练日志、模型系数。此列需为 unique。
datname	name	该模型所服务的 database 名，每个模型只针对单个 database。此参数决定训练时所使用的数据。
ip	name	AiEngine 端所部署的 host ip 地址。
port	integer	AiEngine 端所侦听的端口号。
max_epoch	integer	模型每次训练的迭代次数上限。
learning_rate	real	模型训练的学习速率，推荐缺省值 1。
dim_red	real	模型特征维度降维系数。
hidden_units	integer	模型隐藏层神经元个数。如果训练发现模型长期无法收敛，可以适量提升本参数。
batch_size	integer	模型每次迭代时一个 batch 的大小，尽量设为大于等

名称	类型	描述
		于训练数据总量的值，加快模型的收敛速度。
feature_size	integer	[不需设置] 模型特征的长度，用于触发重新训练，模型训练后该参数自动更新。
available	boolean	[不需设置]标识模型是否收敛。
Is_training	boolean	[不需设置]标识模型是否正在训练。
label	“char” []	模型的目标任务：S：startup timeT：total timeR：rowsM：peak memory 目前受模型性能限制，推荐{S, T}或{R}。
max	bigint[]	[不需设置]标识模型各任务标签的最大值，用于触发重新训练。
acc	real[]	[不需设置]标识模型各任务的准确率。
description	text	模型注释。

5.3.22.GS_PACKAGE

GS_PACKAGE 系统表记录 PACKAGE 内的信息。

说明

部分内置包是通过 namespace 模式加内置函数或视图的方式实现的，不是 package 对象，因此不在 GS_PACKAGE 系统表内。

表 1 GS_PACKAGE 字段

名称	类型	描述
oid	oid	行标识符（隐藏列，必须显式指定列名选择）。
pkgnamespace	oid	package 所属 schema。
pkgowner	oid	package 的所属者。
pkgname	name	package 的名字。
pkgspecsrc	text	package specification 的内容。
pkgbodydeclsrc	text	package body 的内容。
pkgbodyinitsrc	text	package init 的内容。
pkgacl	aclitem	访问权限。
pkgsecdef	boolean	package 是否是定义者权限。

5.3.23.GS_POLICY_LABEL

GS_POLICY_LABEL 系统表记录资源标签配置信息，一个资源标签对应着一条或多条记录，每条记录标记了数据库资源所属的资源标签。需要有系统管理员或安全策略管理员权限才可以访问此系统表。
FQDN（Fully Qualified Domain Name）标识了数据库资源所属的绝对路径。

表 1 GS_POLICY_LABEL 表字段

名称	类型	描述
oid	oid	行标识符（隐藏列，必须显式指定列名选择）。
labelname	name	资源标签名称。
labeltype	name	资源标签类型，目前仅为 RESOURCE。
fqdnnamespace	oid	被标识的数据库资源所属的 namespace oid。
fqdnid	oid	被标识的数据库资源的 oid，若数据库资源为列，则该列为所属表的 oid。
relcolumn	name	列名，若被标识的数据库资源为列，该列指出列名，否则该列为空。
fqdntype	name	被标识的数据库资源的类型名称。 例如 schema、table、column、view 等。
is_sensitive	boolean	fqdnid 对应的表中的数据是否为敏感数据

5.3.24.GS_RECYCLEBIN

GS_RECYCLEBIN 描述了回收站对象的详细信息。

表 1 gs_recyclebin 字段

名称	类型	描述

名称	类型	描述
oid	oid	系统列。（隐藏列，必须显式指定列名选择。）
rcybaseid	oid	基表对象 id，引用 gs_recyclebin.oid。
rcydbid	oid	当前对象所属数据库 oid。
rcyrelid	oid	当前对象 oid。
rcyname	name	回收站对象名称，格式“BIN$unique_id$oid$0”，其中 unique_id 为最多 16 字符唯一标识，oid 为对象标识符。
rcyoriginname	name	原始对象名称。
rcyoperation	“char”	操作类型。d 表示 dropt 表示 truncate
rcytype	int	对象类型。0 表示 table。1 表示 index。2 表示 toast table。3 表示 toast index。4 表示 sequence，指 serial、bigserial 类型自动关联的序列对象。
rcyrecyclecsn	bigint	对象 drop、truncate 时 csn。
rcyrecycletime	timestamptz	对象 drop、truncate 时间。
rcycreatecsn	bigint	对象创建时 csn。
rcychangecsn	bigint	对象定义改变的 csn。
rcynamespace	oid	包含这个关系的名字空间的 OID。
rcyowner	oid	关系所有者。
rcytablespace	oid	这个关系存储所在的表空间。如果为 0，则意味着使用该数据库的缺省表空间。如果关系在磁盘上没有文件，则这个字段没有什么意义。
rcyrelfilenode	oid	回收站对象在磁盘上的文件的名称，如果没有则为 0，用于 TRUNCATE 对象恢复时纹理文件还原。
rcycanrestore	bool	是否可以被单独闪回。
rcycanpurge	bool	是否可以被单独 purge。
rcyfrozenxid	xid32	该表中所有在这个之前的事务 ID 已经被一个固定的（”frozen”）事务 ID 替换。
rcyfrozenxid64	xid	该表中所有在这个之前的事务 ID 已经被一个固定的（”frozen”）事务 ID 替换。

5.3.25.GS_SQL_PATCH

GS_SQL_PATCH 系统表存储所有 SQL_PATCH 的状态信息。

名称	类型	描述
patch_name	name	PATCH 名称。
unique_sql_id	bigint	查询全局唯一 ID。
owner	oid	PATCH 的创建用户 ID。
enable	bool	PATCH 是否生效。
status	"char"	PATCH 的状态（预留字段）。
abort	bool	是否是 AbortHint。
hint_string	text	Hint 文本。
hint_node	pg_node_tree	Hint 解析&序列化的结果。
original_query	text	原始语句（预留字段）。
patched_query	text	PATCH 之后的语句（预留字段）。
original_query_tree	pg_node_tree	原始语句的解析结果（预留字段）。
patched_query_tree	pg_node_tree	PATCH 之后语句的解析结果（预留字段）。
Description	text	PATCH 的备注。

5.3.26.GS_TXN_SNAPSHOT

GS_TXN_SNAPSHOT 是“时间戳-CSN”映射表，周期性采样，并维护适当的时间范围，用于估算范围内的时间戳对应的 CSN 值。

表 1 GS_TXN_SNAPSHOT 字段

名称	类型	描述
snptime	timestamptz	快照捕获时间
snpxmin	bigint	快照 xmin
snpcsn	bigint	快照 csn
snpsnapshot	text	快照序列化文本

5.3.27.GS_UID

GS_UID 系统表存储了数据库中使用 cmuids 属性表的唯一标识元信息。

表 1 GS_UID 字段

名称	类型	描述
relid	oid	表的 oid 信息。
uid_backup	bigint	当前可以为表分配唯一标识的最大值。

5.3.28.GS_WLM_EC_OPERATOR_INFO

GS_WLM_EC_OPERATOR_INFO 系统表存储执行 EC（Extension Connector）作业结束后的算子相关的记录。当设置 GUC 参数 enable_resource_record（参见：GUC 参数说明->负载管理章节中的 enable_resource_record)为 on 时，系统会每 3 分钟将 GS_WLM_EC_OPERATOR_HISTORY 中的记录导入此系统表，开启此功能会占用系统存储空间并对性能有一定影响。查询该系统表需要 sysadmin 权限。

表 1 GS_WLM_EC_OPERATOR_INFO 的字段

名称	类型	描述
queryid	bigint	EC 语句执行使用的内部 query_id。
plan_node_id	integer	EC 算子对应的执行计划的 plan node id。
start_time	timestamp with time zone	EC 算子处理第一条数据的开始时间。
duration	bigint	EC 算子到结束时候总的执行时间(ms)。
tuple_processed	bigint	EC 算子返回的元素个数。
min_peak_memory	integer	EC 算子在所有 DN 上的最小内存峰值(MB)。
max_peak_memory	integer	EC 算子在所有 DN 上的最大内存峰值(MB)。
average_peak_memory	integer	EC 算子在所有 DN 上的平均内存峰值(MB)。

ec_status	text	EC 作业的执行状态。
ec_execute_datanode	text	执行 EC 作业的 DN 名称。
ec_dsn	text	EC 作业所使用的 DSN。
ec_username	text	EC 作业访问远端集群的 USERNAME（远端集群为 SPARK 类型时该值为空）。
ec_query	text	EC 作业发送给远端集群执行的语句。
ec_libodbc_type	text	EC 作业使用的 unixODBC 驱动类型。

5.3.29.GS_WLM_INSTANCE_HISTORY

GS_WLM_INSTANCE_HISTORY 系统表存储与实例（数据库主节点或数据库节点）相关的资源使用相关信息。该系统表里每条记录都是对应时间点某实例资源使用情况，包括：内存、CPU 核数、磁盘 IO、进程物理 IO 和进程逻辑 IO 信息。查询该系统表需要 sysadmin 权限，且仅在数据库 postgres 下面查询时有数据。

表 1 GS_WLM_INSTANCE_HISTORY 字段

名称	类型	描述
instancename	text	实例名称。
timestamp	timestamp with time zone	时间戳。
used_cpu	int	实例使用 CPU 所占用的百分比。
free_mem	int	实例未使用的内存大小，单位 MB。
used_mem	int	实例已使用的内存大小，单位 MB。
io_await	real	实例所使用磁盘的 io_wait 值（10 秒均值）。
io_util	real	实例所使用磁盘的 io_util 值（10 秒均值）。
disk_read	real	实例所使用磁盘的读速率（10 秒均值），单位 KB/s。
disk_write	real	实例所使用磁盘的写速率（10 秒均值），单位 KB/s。
process_read	bigint	实例对应进程从磁盘读数据的读速率(不包括从磁盘 pagecache 中读取的字节数，10 秒均值)，

名称	类型	描述
		单位 KB/s。
process_write	bigint	实例对应进程向磁盘写数据的写速率(不包括向磁盘 pagecache 中写入的字节数，10 秒均值)，单位 KB/s。
logical_read	bigint	数据库主节点实例：不统计。 数据库节点实例：该实例在本次统计间隙（10 秒）内逻辑读字节速率，单位 KB/s。 仅分布式可用。
logical_write	bigint	数据库主节点实例：不统计。 数据库节点实例：该实例在本次统计间隙（10 秒）内逻辑写字节速率，单位 KB/s。 仅分布式可用。
read_counts	bigint	数据库主节点实例：不统计。数据库节点实例：该实例在本次统计间隙（10 秒）内逻辑读操作次数之和，单位次。 仅分布式可用。
write_counts	bigint	数据库主节点实例：不统计。数据库节点实例：该实例在本次统计间隙（10 秒）内逻辑写操作次数之和，单位次。 仅分布式可用。

5.3.30.GS_WLM_OPERATOR_INFO

GS_WLM_OPERATOR_INFO 系统表显示执行作业结束后的算子相关的记录。此数据是从内核中转储到系统表中的数据。查询该系统表需要 sysadmin 权限，且仅在数据库 postgres 下面查询时有数据。

表 1 GS_WLM_OPERATOR_INFO 的字段

名称	类型	描述
queryid	bigint	语句执行使用的内部 query_id。
pid	bigint	后端线程 id。
plan_node_id	integer	查询对应的执行计划的 plan node id。
plan_node_name	text	对应于 plan_node_id 的算子的名称。
start_time	timestamp	该算子处理第一条数据的开始时间。

名称	类型	描述
	with time zone	
duration	bigint	该算子到结束时候总的执行时间(ms)。
query_dop	integer	当前算子执行时的并行度。
estimated_rows	bigint	优化器估算的行数信息。
tuple_processed	bigint	当前算子返回的元素个数。
min_peak_memory	integer	当前算子在数据库节点上的最小内存峰值(MB)。
max_peak_memory	integer	当前算子在数据库节点上的最大内存峰值(MB)。
average_peak_memory	integer	当前算子在数据库节点平均内存峰值(MB)。
memory_skew_percent	integer	当前算子在数据库节点间的内存使用倾斜率。
min_spill_size	integer	若发生下盘，数据库节点盘的最小数据量(MB)，默认为 0。
max_spill_size	integer	若发生下盘，数据库节点上下盘的最大数据量(MB)，默认为 0。
average_spill_size	integer	若发生下盘，数据库节点盘的平均数据量(MB)，默认为 0。
spill_skew_percent	integer	若发生下盘，数据库节点间下盘倾斜率。
min_cpu_time	bigint	该算子在数据库节点最小执行时间(ms)。
max_cpu_time	bigint	该算子在数据库节点上的最大执行时间(ms)。
total_cpu_time	bigint	该算子在数据库节点上的总执行时间(ms)。
cpu_skew_percent	integer	数据库节点间执行时间的倾斜率。
warning	text	主要显示如下几类告警信息： ❖ Sort/SetOp/HashAgg/HashJoin spill ❖ Spill file size large than 256MB ❖ Broadcast size large than 100MB ❖ Early spill ❖ Spill times is greater than 3

名称	类型	描述
		❖ Spill on memory adaptive ❖ Hash table conflict

5.3.31.GS_WLM_PLAN_ENCODING_TABLE

GS_WLM_PLAN_ENCODING_TABLE 系统表显示计划算子级的编码信息，为机器学习模型的提供包括 startup time、total time、peak memory、rows 等标签值的训练、预测集。

表 1 GS_WLM_PLAN_ENCODING_TABLE 的字段

名称	类型	描述
queryid	bigint	语句执行使用的内部 query_id。
plan_node_id	integer	查询对应的执行计划的 plan node id。
parent_node_id	integer	当前算子的父节点 node id。
startup_time	bignit	该算子处理第一条数据的开始时间。
total_time	bigint	该算子到结束时候总的执行时间（ms）。
rows	bigint	当前算子执行的行数信息。
peak_memory	integer	当前算子在数据库节点上的最大内存峰值（MB）。
encode	text	当前计划算子的编码信息。

5.3.32.GS_WLM_PLAN_OPERATOR_INFO

GS_WLM_PLAN_OPERATOR_INFO 系统表显示执行作业结束后计划算子级的相关的记录。此数据是从内核中转储到系统表中的数据。

表 1 GS_WLM_PLAN_OPERATOR_INFO 的字段

名称	类型	描述
datname	name	收集计划信息所在的 database 名。
queryid	bigint	语句执行使用的内部 query_id。
plan_node_id	integer	查询对应的执行计划的 plan node id。
startup_time	bigint	该算子处理第一条数据的开始时间。
total_time	bigint	该算子到结束时候总的执行时间(ms)。
actual_rows	bigint	实际执行的行数信息。
max_peak_memory	integer	当前算子在数据库节点上的最大内存峰值(MB)。

名称	类型	描述
query_dop	integer	当前算子执行时的并行度。
parent_node_id	integer	当前算子的父节点 node id。
left_child_id	integer	当前算子的左孩子节点 node id。
right_child_id	integer	当前算子的右孩子节点 node id。
operation	text	当前算子进行的操作名称。
orientation	text	当前算子的对齐方式。
strategy	text	当前算子操作的实现方法。
options	text	当前算子操作的选择方式。
condition	text	当前算子操作的过滤条件。
projection	text	当前算子的映射关系。

5.3.33.GS_WLM_SESSION_QUERY_INFO_ALL

GS_WLM_SESSION_QUERY_INFO_ALL 系统表显示当前数据库实例执行作业结束后的负载管理记录。此数据是从内核中转储到系统表中的数据。当设置 GUC 参数 enable_resource_record（参见：GUC 参数说明->负载管理章节中的 enable_resource_record)为 on 时，系统会定时（周期为 3 分钟）将内核中 query 信息导入 GS_WLM_SESSION_QUERY_INFO_ALL 系统表。查询该系统表需要 sysadmin 权限，且仅在数据库 postgres 下面查询时有数据。

表 1 GS_WLM_SESSION_QUERY_INFO_ALL 字段

名称	类型	描述
datid	oid	连接后端的数据库 OID。
dbname	text	连接后端的数据库名称。
schemaname	text	模式的名称。
nodename	text	语句执行的数据库实例名称。
username	text	连接到后端的用户名。
application_name	text	连接到后端的应用名。
client_addr	inet	连接到后端的客户端的 IP 地址。 如果此字段是 null，它表明通过服务器机器上 UNIX 套接字连接客户端或者这是内部进程，如 autovacuum。

名称	类型	描述
client_hostname	text	客户端的主机名，这个字段是通过 client_addr 的反向 DNS 查找得到。这个字段只有在启动 log_hostname 且使用 IP 连接时才非空。
client_port	integer	客户端用于与后端通讯的 TCP 端口号，如果使用 Unix 套接字，则为-1。
query_band	text	用于标示作业类型，可通过 GUC 参数 query_band 进行设置，默认为空字符串。
block_time	bigint	语句执行前的阻塞时间，包含语句解析和优化时间，单位 ms。
start_time	timestamp with time zone	语句执行的开始时间。
finish_time	timestamp with time zone	语句执行的结束时间。
duration	bigint	语句实际执行的时间，单位 ms。
estimate_total_time	bigint	语句预估执行时间，单位 ms。
status	text	语句执行结束状态：正常为 finished，异常为 aborted。
abort_info	text	语句执行结束状态为 aborted 时显示异常信息。
resource_pool	text	用户使用的资源池。
control_group	text	语句所使用的 Cgroup。
estimate_memory	integer	语句估算内存大小。
min_peak_memory	integer	语句在数据库实例上的最小内存峰值，单位 MB。
max_peak_memory	integer	语句在数据库实例上的最大内存峰值，单位 MB。
average_peak_memory	integer	语句执行过程中的内存使用平均值，单位 MB。
memory_skew_percent	integer	语句数据库实例间的内存使用倾斜率。

名称	类型	描述
spill_info	text	语句在数据库实例上的下盘信息： ❖ None：数据库实例均未下盘。 ❖ All：数据库实例均下盘。 ❖ [a:b]：数量为 b 个数据库实例中有 a 个数据库实例下盘。
min_spill_size	integer	若发生下盘，数据库实例上下盘的最小数据量，单位 MB，默认为 0。
max_spill_size	integer	若发生下盘，数据库实例上下盘的最大数据量，单位 MB，默认为 0。
average_spill_size	integer	若发生下盘，数据库实例上下盘的平均数据量，单位 MB，默认为 0。
spill_skew_percent	integer	若发生下盘，数据库实例间下盘倾斜率。
min_dn_time	bigint	语句在数据库实例上的最小执行时间，单位 ms。
max_dn_time	bigint	语句在数据库实例上的最大执行时间，单位 ms。
average_dn_time	bigint	语句在数据库实例上的平均执行时间，单位 ms。
dntime_skew_percent	integer	语句在数据库实例间的执行时间倾斜率。
min_cpu_time	bigint	语句在数据库实例上的最小 CPU 时间，单位 ms。
max_cpu_time	bigint	语句在数据库实例上的最大 CPU 时间，单位 ms。
total_cpu_time	bigint	语句在数据库实例上的 CPU 总时间，单位 ms。
cpu_skew_percent	integer	语句在数据库实例间的 CPU 时间倾斜率。
min_peak_iops	integer	语句在数据库实例上的每秒最小 IO 峰值（列存单位是次/s，行存单位是万次/s）。
max_peak_iops	integer	语句在数据库实例上的每秒最大 IO 峰值（列存单位是次/s，行存单位是万次/s）。
average_peak_iops	integer	语句在数据库实例上的每秒平均 IO 峰值（列存单位是次/s，行存单位是万次/s）。
iops_skew_percent	integer	语句在数据库实例间的 IO 倾斜率。

名称	类型	描述
warning	text	主要显示如下几类告警信息： ❖ Sort/SetOp/HashAgg/HashJoin spill ❖ Spill file size large than 256MB ❖ Broadcast size large than 100MB ❖ Early spill ❖ Spill times is greater than 3 ❖ Spill on memory adaptive ❖ Hash table conflict
queryid	bigint	语句执行使用的内部 query id。
query	text	执行的语句。
query_plan	text	语句的执行计划。
node_group	text	语句所属用户对应的逻辑数据库实例。
cpu_top1_node_name	text	cpu 使用率第 1 的节点名称。
cpu_top2_node_name	text	cpu 使用率第 2 的节点名称。
cpu_top3_node_name	text	cpu 使用率第 3 的节点名称。
cpu_top4_node_name	text	cpu 使用率第 4 的节点名称。
cpu_top5_node_name	text	cpu 使用率第 5 的节点名称。
mem_top1_node_name	text	内存使用量第 1 的节点名称。
mem_top2_node_name	text	内存使用量第 2 的节点名称。
mem_top3_node_name	text	内存使用量第 3 的节点名称。
mem_top4_node_name	text	内存使用量第 4 的节点名称。
mem_top5_node_name	text	内存使用量第 5 的节点名称。
cpu_top1_value	bigint	cpu 使用率。
cpu_top2_value	bigint	cpu 使用率。
cpu_top3_value	bigint	cpu 使用率。
cpu_top4_value	bigint	cpu 使用率。
cpu_top5_value	bigint	cpu 使用率。
mem_top1_value	bigint	内存使用量。
mem_top2_value	bigint	内存使用量。
mem_top3_value	bigint	内存使用量。
mem_top4_value	bigint	内存使用量。
mem_top5_value	bigint	内存使用量。
top_mem_dn	text	内存使用量 topN 信息。

名称	类型	描述
top_cpu_dn	text	cpu 使用量 topN 信息。
n_returned_rows	bigint	Select 返回的结果集行数。
n_tuples_fetched	bigint	随机扫描行数。
n_tuples_returned	bigint	顺序扫描行数。
n_tuples_inserted	bigint	插入行数。
n_tuples_updated	bigint	更新行数。
n_tuples_deleted	bigint	删除行数。
n_blocks_fetched	bigint	Cache 加载次数。
n_blocks_hit	bigint	Cache 命中数。
db_time	bigint	有效的 DB 时间花费，多线程将累加（单位：微秒）。
cpu_time	bigint	CPU 时间（单位：微秒）。
execution_time	bigint	执行器内执行时间（单位：微秒）。
parse_time	bigint	SQL 解析时间（单位：微秒）。
plan_time	bigint	SQL 生成计划时间（单位：微秒）。
rewrite_time	bigint	SQL 重写时间（单位：微秒）。
pl_execution_time	bigint	plpgsql 上的执行时间（单位：微秒）。
pl_compilation_time	bigint	plpgsql 上的编译时间（单位：微秒）。
net_send_time	bigint	网络上的时间花费（单位：微秒）。
data_io_time	bigint	IO 上的时间花费（单位：微秒）。
is_slow_query	bigint	标记是否为慢查询。取值为 1 时表示其为慢查询。

5.3.34.GS_WLM_USER_RESOURCE_HISTORY

GS_WLM_USER_RESOURCE_HISTORY 系统表存储与用户使用资源相关的信息。该系统表的每条记录都是对应时间点某用户的资源使用情况，包括：内存、CPU 核数、存储空间、临时空间、算子落盘空间、逻辑 IO 流量、逻辑 IO 次数和逻辑 IO 速率信息。其中，内存、CPU、IO 相关监控项仅记录用户复杂作业的资源使用情况。对于 IO 相关监控项，当参数 enable_logical_io_statistics 为 on 时才有效；当参数 enable_user_metric_persistent 为 on 时，才会开启用户监控数据保存功能。GS_WLM_USER_RESOURCE_HISTORY 系统表的数据来源于

PG_TOTAL_USER_RESOURCE_INFO 视图。查询该系统表需要 sysadmin 权限，且仅在数据库 postgres 下面查询时有数据。

表 1 GS_WLM_USER_RESOURCE_HISTORY

名称	类型	描述
username	text	用户名
timestamp	timestamp with time zone	时间戳
used_memory	integer	正在使用的内存大小，单位 MB。
total_memory	integer	可以使用的内存大小，单位为 MB。值为 0 表示未限制最大可用内存，其限制取决于数据库最大可用内存。
used_cpu	real	正在使用的 CPU 核数。
total_cpu	integer	该机器节点上，用户关联控制组的 CPU 核数总和。
used_space	bigint	已使用的存储空间大小，单位 KB。
total_space	bigint	可使用的存储空间大小，单位 KB，值为-1 表示未限制最大存储空间。
used_temp_space	bigint	已使用的临时存储空间大小，单位 KB。
total_temp_space	bigint	可使用的临存储空间大小，单位 KB，值为-1 表示未限制最大临时存储空间。
used_spill_space	bigint	已使用的算子落盘存储空间大小，单位 KB。
total_spill_space	bigint	可使用的算子落盘存储空间大小，单位 KB，值为-1 表示未限制最大算子落盘存储空间。
read_kbytes	bigint	监控周期内，读操作的字节流量，单位 KB。仅分布式可用。
write_kbytes	bigint	监控周期内，写操作的字节流量，单位 KB。
read_counts	bigint	监控周期内，读操作的次数，单位次。仅分布式可用。
write_counts	bigint	监控周期内，写操作的次数，单位次。仅分布式可用。
read_speed	real	监控周期内，读操作的字节速率，单位 KB/s。仅分布式可用。

名称	类型	描述
write_speed	real	监控周期内，写操作的字节速率，单位KB/s。 仅分布式可用。

5.3.35.PG_AGGREGATE

PG_AGGREGATE 系统表存储与聚集函数有关的信息。PG_AGGREGATE 里的每条记录都是一条 pg_proc 里面的记录的扩展。PG_PROC 记录承载该聚集的名称、输入和输出数据类型，以及其他一些和普通函数类似的信息。

表 1 PG_AGGREGATE 字段

名称	类型	引用	描述
aggfnoid	regproc	PG_PROC.proname	此聚集函数的 PG_PROC proname。
aggtransfn	regproc	PG_PROC.proname	转换函数。
aggcollectfn	regproc	PG_PROC.proname	收集函数。
aggfinalfn	regproc	PG_PROC.proname	最终处理函数（如果没有则为零）。
aggsortop	oid	PG_OPERATOR.oid	关联排序操作符（如果没有则为零）。
aggtranstype	oid	PG_TYPE.oid	此聚集函数的内部转换（状态）数据的数据类型。 可能取值及其描述见于系统表 pg_type 或查看源码中的 pg_type.h 文件中 type 定义，主要分为多态（isPolymorphicType）和非多态两类。
agginitval	text	-	转换状态的初始值。这是一个文本数据域，它包含初始值的外部字符串表现形式。如果数据域是 null，则转换状态值从 null 开始。
agginitcollect	text	-	收集状态的初始值。这是一个

名称	类型	引用	描述
			文本数据域，它包含初始值的外部字符串表现形式。如果数据域是 null，则收集状态值从 null 开始。
aggkind	“char”	-	此聚集函数类型： -'n'：表示 Normal Agg。 -'o'：表示 Ordered Set Agg。
aggnumdirectargs	smallint	-	Ordered Set Agg 类型聚集函数的直接参数（非聚集相关参数）数量。对 Normal Agg 类型聚集函数，该值为 0。

5.3.36.PG_AM

PG_AM 系统表存储有关索引访问方法的信息。系统支持的每种索引访问方法都有一行。

表 1 PG_AM 字段

名称	类型	引用	描述
oid	oid	-	行标识符（隐藏列，必须显式指定列名选择）。
amname	name	-	访问方法的名称。
amstrategies	smallint	-	访问方法的操作符策略个数，或者如果访问方法没有一个固定的操作符策略集则为 0。
amsupport	smallint	-	访问方法的支持过程个数。
amcanorder	boolean	-	这种访问方式是否支持通过索引字段值的命令扫描排序。
amcanorderbyop	boolean	-	这种访问方式是否支持通过索引字段上操作符的结果的命令扫描排序。
amcanbackward	boolean	-	访问方式是否支持向后扫描。
amcanunique	boolean	-	访问方式是否支持唯一索引。

名称	类型	引用	描述
amcanmulticol	boolean	-	访问方式是否支持多字段索引。
amoptionalkey	boolean	-	访问方式是否支持第一个索引字段上没有任何约束的扫描。
amsearcharray	boolean	-	访问方式是否支持 ScalarArrayOpExpr 搜索。
amsearchnulls	boolean	-	访问方式是否支持 IS NULL/NOT NULL 搜索。
amstorage	boolean	-	允许索引存储的数据类型与列的数据类型是否不同。
amclusterable	boolean	-	是否允许在一个这种类型的索引上聚簇。
ampredlocks	boolean	-	是否允许这种类型的一个索引管理细粒度的谓词锁定。
amkeytype	oid	PG_TYPE.oid	存储在索引里数据的类型，如果不是一个固定的类型则为 0。
aminsert	regproc	PG_PROC.proname	“插入这个行”函数。
ambeginscan	regproc	PG_PROC.proname	“准备索引扫描”函数。
amgettuple	regproc	PG_PROC.proname	“下一个有效行”函数，如果没有则为 0。
amgetbitmap	regproc	PG_PROC.proname	“抓取所有的有效行”函数，如果没有则为 0。
amrescan	regproc	PG_PROC.proname	“（重新）开始索引扫描”函数。
amendscan	regproc	PG_PROC.proname	“索引扫描后清理”函数。
ammarkpos	regproc	PG_PROC.proname	“标记当前扫描位置”函数。
amrestrpos	regproc	PG_PROC.proname	“恢复已标记的扫描位置”函数。
ammerge	regproc	PG_PROC.proname	“归并多个索引对象”函数。
ambuild	regproc	PG_PROC.proname	“建立新索引”函数。
ambuildempty	regproc	PG_PROC.proname	“建立空索引”函数。
ambulkdelete	regproc	PG_PROC.proname	批量删除函数。

名称	类型	引用	描述
amvacuumcleanup	regproc	PG_PROC.proname	VACUUM 后的清理函数。
amcanreturn	regproc	PG_PROC.proname	检查是否索引支持唯一索引扫描的函数，如果没有则为 0。
amcostestimate	regproc	PG_PROC.proname	估计一个索引扫描开销的函数。
amoptions	regproc	PG_PROC.proname	为一个索引分析和确认 reloptions 的函数。

5.3.37.PG_AMOP

PG_AMOP 系统表存储有关和访问方法操作符族关联的信息。如果一个操作符是一个操作符族中的成员，则在这个表中会占据一行。一个族成员是一个 search 操作符或一个 ordering 操作符。一个操作符可以在多个族中出现，但是不能在一个族中的多个搜索位置或多个排序位置中出现。

表 1 PG_AMOP 字段

名称	类型	引用	描述
oid	oid	-	行标识符（隐藏属性，必须明确选择）。
amopfamily	oid	PG_OPFAMILY.oid	这个项的操作符族。
amoplefttype	oid	PG_TYPE.oid	操作符的左输入类型。可能取值及其描述见于系统表 pg_type 或查看源码中的 pg_type.h 文件。
amoprighttype	oid	PG_TYPE.oid	操作符的右输入类型。可能取值及其描述见于系统表 pg_type 或查看源码中的 pg_type.h 文件。
amopstrategy	smallint	-	操作符策略数。
amoppurpose	“char”	-	操作符目的，s 为搜索或 o 为排序。
amopopr	oid	PG_OPERATOR.oid	该操作符的 OID。
amopmethod	oid	PG_AM.oid	索引访问方式操作符族。
amopsortfamily	oid	PG_OPFAMILY.oid	如果是一个排序操作符，则为这个项排序所依据的 btree 操作符族；如果是一个搜索操作符，则为

名称	类型	引用	描述
			0。

search 操作符表明这个操作符族的一个索引可以被搜索，找到所有满足 WHERE indexed_column operator constant 的行。显然，这样的操作符必须返回布尔值，并且它的左输入类型必须匹配索引的字段数据类型。

ordering 操作符表明这个操作符族的一个索引可以被扫描，返回以 ORDER BY indexed_column operator constant 顺序表示的行。这样的操作符可以返回任意可排序的数据类型，它的左输入类型也必须匹配索引的字段数据类型。ORDER BY 的确切的语义是由 amopsortfamily 字段指定的，该字段必须为操作符的返回类型引用一个 btree 操作符族。

5.3.38.PG_AMPROC

PG_AMPROC 系统表存储有关与访问方法操作符族相关联的支持过程的信息。每个属于某个操作符族的支持过程都占有一行。

表 1 PG_AMPROC 字段

名称	类型	引用	描述
oid	oid	-	行标识符（隐藏列，必须显式指定列名选择）。
amprocfamily	oid	PG_OPFAMILY.oid	该项的操作符族。
amproclefttype	oid	PG_TYPE.oid	相关操作符的左输入数据类型。可能取值及其描述见于系统表 pg_type 或查看源码中的 pg_type.h 文件。
amprocrighttype	oid	PG_TYPE.oid	相关操作符的右输入数据类型。可能取值及其描述见于系统表 pg_type 或查看源码中的 pg_type.h 文件。
amprocnum	smallint	-	支持过程编号。
amproc	regproc	PG_PROC.proname	过程的 OID。

amproclefttype 和 amprocrighttype 字段的习惯解释，标识一个特定支持过程支持的操作符的左和右输入类型。对于某些访问方式，匹配支持过程本身

的输入数据类型，对其他的则不这样。有一个对索引的“缺省”支持过程的概念，amproclefttype 和 amprocrighttype 都等于索引操作符类的 opcintype。

5.3.39.PG_APP_WORKLOADGROUP_MAPPING

PG_APP_WORKLOADGROUP_MAPPING 系统表提供了数据库负载映射组的信息。

表 1 PG_APP_WORKLOADGROUP_MAPPING 字段

名称	类型	描述
oid	oid	行标识符（隐藏列，必须显式指定列名选择）。
appname	name	应用名称。
workload_gpname	name	映射到的负载组名称。

5.3.40.PG_ATTRDEF

PG_ATTRDEF 系统表存储列的默认值。

表 1 PG_ATTRDEF 字段

名称	类型	描述
oid	oid	行标识符（隐藏列，必须显式指定列名选择）。
adrelid	oid	该列的所属表。
adnum	smallint	该列的数目。
adbin	pg_node_tree	字段缺省值或生成表达式的内部表现形式。
adsrc	text	可读缺省值或生成表达式的内部表现形式。
adgencol	“char”	标识该列是否为生成列。取值为's'表示该列为生成列，取值为'\0'表示该列为普通列，默认值为'\0'。

5.3.41.PG_ATTRIBUTE

PG_ATTRIBUTE 系统表存储关于表字段的信息。

表 1 PG_ATTRIBUTE 字段

名称	类型	描述

名称	类型	描述
attrelid	oid	此字段所属表。
attname	name	字段名。
atttypid	oid	字段类型。
attstattarget	integer	控制 ANALYZE 为这个字段积累的统计细节的级别。 ❖ 零值表示不收集统计信息。 ❖ 负数表示使用系统缺省的统计对象。 ❖ 正数值的确切信息是和数据类型相关的。 ❖ 对于标量数据类型，ATTSTATTARGET 既是要收集的“最常用数值”的目标数目，也是要创建的柱状图的目标数量。
attlen	smallint	是本字段类型的 pg_type.typlen 的拷贝。
attnum	smallint	字段编号。
attndims	integer	如果该字段是数组，则是维数，否则是 0 。
attcacheoff	integer	在磁盘上的时候总是-1，但是如果加载入内存中的行描述器中，它可能会被更新以缓冲在行中字段的偏移量。
atttypmod	integer	记录创建新表时支持的类型特定的数据（比如一个 varchar 字段的最大长度）。它传递给类型相关的输入和长度转换函数当做第三个参数。其值对那些不需要 ATTTYPMOD 的类型通常为-1。
attbyval	boolean	这个字段类型的 pg_type.typbyval 的拷贝。
attstorage	"char"	这个字段类型的 pg_type.typstorage 的拷贝。
attalign	"char"	这个字段类型的 pg_type.typalign 的拷贝。
attnotnull	boolean	这代表一个非空约束。可以改变这个字段以打开或者关闭这个约束。
attcmdef	boolean	这个字段有一个缺省值，此时它对应 pg_attrdef

名称	类型	描述
		表里实际定义此值的记录。
attisdropped	boolean	这个字段已经被删除了，不再有效。一个已经删除的字段物理上仍然存在表中，但会被分析器忽略，因此不能再通过 SQL 访问。
attislocal	boolean	这个字段是局部定义在关系中的。请注意一个字段可以同时是局部定义和继承的。
attcmprmode	tinyint	对某一列指定压缩方式。压缩方式包括： ❖ ATT_CMPR_NOCOMPRESS ❖ ATT_CMPR_DELTA ❖ ATT_CMPR_DICTIONARY ❖ ATT_CMPR_PREFIX ❖ ATT_CMPR_NUMSTR
attinhcount	integer	这个字段所拥有的直接父表的个数。如果一个字段的父表个数非零，则它就不能被删除或重命名。
attcollation	oid	对此列定义的校对列。
attalias	name	别名。
attisquoted	boolean	列名是否被双引号包裹。
attacl	aclitem[]	列级访问权限控制。
attoptions	text[]	字段属性。目前支持以下两种属性： ❖ n_distinct：该字段的 distinct 值数量（不包含字表）。 ❖ n_distinct_inherited:该字段的 distinct 值数量（包含字表）。
attfdwoptions	text[]	外表字段属性。当前支持的 dist_fdw、file_fdw、log_fdw 未使用外表字段属性。
attinitdefval	bytea	存储了此列默认的值表达式。行存表的 ADD COLUMN 需要使用此字段。

名称	类型	描述
attkvtype	tinyint	对某一列指定 key value 类型。类型包括： ❖ 0. ATT_KV_UNDEFINED：默认。 ❖ 1. ATT_KV_TAG：维度。 ❖ 2. ATT_KV_FIELD：指标。 ❖ 3. ATT_KV_TIMETAG：时间列。
attidentity	"char"	如果是一个零字节''，则不是一个标识列。否则： ❖ a = 总是生成 ❖ d = 默认生成

5.3.42.PG_AUTH_HISTORY

PG_AUTH_HISTORY 系统表记录了角色的认证历史。需要有系统管理员权限才可以访问此系统表。

表 1 PG_AUTH_HISTORY 字段

名称	类型	描述
oid	oid	行标识符（隐藏列，必须显式指定列名选择）。
roloid	oid	角色标识。
passwordtime	timestamp with time zone	创建和修改密码的时间。
rolpassword	text	角色密码密文，加密方式由 GUC 参数 password_encryption_type 确定。

5.3.43.PG_AUTH_MEMBERS

PG_AUTH_MEMBERS 系统表存储显示角色之间的成员关系。

表 1 PG_AUTH_MEMBERS 字段

名称	类型	描述
roleid	oid	拥有成员的角色 ID。
member	oid	属于 ROLEID 角色的一个成员的角色 ID。
grantor	oid	赋予此成员关系的角色 ID。
admin_option	Boolean	如果 MEMBER 可以把 ROLEID 角色的成员关系赋予其他

名称	类型	描述
		角色，则为真。

5.3.44.PG_AUTHID

PG_AUTHID 系统表存储有关数据库认证标识符（角色）的信息。角色把“用户”的概念包含在内。一个用户实际上就是一个 rolcanlogin 标志被设置的角色。任何角色（不管 rolcanlogin 设置与否）都能够把其他角色作为成员。PanWeiDB 中只有一份 pg_authid，不是每个数据库有一份。需要有系统管理员权限才可以访问此系统表。

表 1 PG_AUTHID 字段

名称	类型	描述
oid	oid	行标识符（隐藏列，必须显式指定列名选择）。
rolname	name	角色名称。
rolsuper	boolean	角色是否是拥有最高权限的初始系统管理员。 ❖ t（true）：表示是。 ❖ f（false）：表示不是。
rolinherit	boolean	角色是否自动继承其所属角色的权限。 ❖ t（true）：表示自动继承。 ❖ f（false）：表示不自动继承。
rolcreaterole	boolean	角色是否可以创建更多角色。 ❖ t（true）：表示可以。 ❖ f（false）：表示不可以。
rolcreatedb	boolean	角色是否可以创建数据库。 ❖ t（true）：表示可以。 ❖ f（false）：表示不可以。
rolcatupdate	boolean	角色是否可以直接更新系统表。只有 usesysid=10 的初始系统管理员拥有此权

名称	类型	描述
		限。其他用户无法获得此权限。 ❖ t（true）：表示可以。 ❖ f（false）：表示不可以。
rolcanlogin	boolean	角色是否可以登录，也就是说，这个角色可以给予会话认证标识符。 ❖ t（true）：表示可以。 ❖ f（false）：表示不可以。
rolreplication	boolean	角色是否具有复制权限： ❖ t（true）：表示有。 ❖ f（false）：表示没有。
rolssoadmin	boolean	角色是否有单点登录的权限： ❖ t（true）：表示有。 ❖ f（false）：表示没有。
rolauditadmin	boolean	角色是否具有审计管理员权限： ❖ t（true）：表示有。 ❖ f（false）：表示没有。
rolsystemadmin	boolean	角色是否具有系统管理员权限： ❖ t（true）：表示有。 ❖ f（false）：表示没有。
rolconnlimit	integer	对于可以登录的角色，限制其最大并发连接数量。 -1 表示没有限制。
rolpassword	text	口令（可能是加密的），如果没有口令，则为 NULL。
rolvalidbegin	timestamp with time zone	帐户的有效开始时间，如果没有开始时间，则为 NULL。
rolvaliduntil	timestamp with time zone	帐户的有效结束时间，如果没有结束时间，则为 NULL。

名称	类型	描述
rolrespool	name	用户所能够使用的 resource pool。
roluseft	boolean	角色是否可以操作外表。 ❖ t（true）：表示可以。 ❖ f（false）：表示不可以。
rolparentid	oid	用户所在组用户的 OID。
roltabspace	text	用户数据表的最大空间限额。
rolkind	"char"	特殊用户种类，包括私有用户、普通用户。
rolnodegroup	oid	该字段不支持。
roltempspace	text	用户临时表的最大空间限额，单位为 KB。
rolspillspace	text	用户执行作业时下盘数据的最大空间限额，单位为 KB。
rolexcpdata	text	用户可以设置的查询规则（当前未使用）。
rolmonitoradmin	boolean	角色是否具有监控管理员权限： ❖ t（true）：表示有。 ❖ f（false）：表示没有。
roloperatoradmin	boolean	角色是否具有运维管理员权限： ❖ t（true）：表示有。 ❖ f（false）：表示没有。
rolpolicyadmin	boolean	角色是否具有安全策略管理员权限： ❖ t（true）：表示有。 ❖ f（false）：表示没有。

5.3.45.PG_CAST

PG_CAST 系统表存储数据类型之间的转化关系。

表 1 PG_CAST 字段

名称	类型	描述
oid	oid	行标识符（隐藏列，必须显式指定列名选择）。
castsource	oid	源数据类型的 OID。
casttarget	oid	目标数据类型的 OID。
castfunc	oid	转化函数的 OID。如果为零表明不需要转化函数。
castcontext	“char”	源数据类型和目标数据类型间的转化方式： -'e'：表示只能进行显式转化（使用 CAST 或::语法）。 -'i'：表示能进行隐式转化。 -'a'：表示类型间同时支持隐式和显式转化。
castmethod	“char”	转化方法： -'f'：使用 castfunc 字段中指定的函数进行转化。 -'b'：类型间是二进制强制转化，不使用 castfunc。

5.3.46.PG_CLASS

PG_CLASS 系统表存储数据库对象信息及其之间的关系。

表 1 PG_CLASS 字段

名称	类型	描述
oid	oid	行标识符（隐藏列，必须显式指定列名选择）。
relname	name	表、索引、视图等对象的名称。
relnamespace	oid	包含这个关系的名称空间的 OID。
reltype	oid	对应这个表的行类型的数据类型（索引为零，因为索引没有 pg_type 记录）。
reloftype	oid	复合类型的 OID，0 表示其他类型。
relowner	oid	关系所有者。
relam	oid	如果行是索引，则就是所用的访问模式（B-tree、hash 等）。
relfilenode	oid	关系在磁盘上的文件的名称，如果没有则为

名称	类型	描述
		0。
reltablespace	oid	关系存储所在的表空间。如果为零，则意味着使用该数据库的缺省表空间。如果关系在磁盘上没有文件，则这个字段没有什么意义。
relpages	double precision	以页（大小为 BLCKSZ）为单位的表在磁盘上的大小，它只是优化器用的一个近似值。
reltuples	double precision	表中行的数目，只是优化器使用的一个估计值。
relallvisible	integer	被标识为全可见的表中的页的数量。此字段是优化器用来做 SQL 执行优化使用的。 VACUUM、ANALYZE 和一些 DDL 语句（例如，CREATE INDEX）会引起此字段更新。
reltoastrelid	oid	与此表关联的 TOAST 表的 OID，如果没有则为 0。 TOAST 表在一个从属表里“离线”存储大字段。
reltoastidxid	oid	对于 TOAST 表是它的索引的 OID，如果不是 TOAST 表则为 0。
reldeltarelid	oid	Delta 表的 OID。Delta 表附属于列存表。用于存储数据导入过程中的甩尾数据。
reldeltaidx	oid	Delta 表的索引表 OID。
relcudescrelid	oid	CU 描述表的 OID。CU 描述表（Desc 表）附属于列存表。用于控制表目录中存储数据的可见性。
relcudescidx	oid	CU 描述表的索引表 OID。
relcmindex	boolean	如果它是一个表而且至少有（或者最近有过）一个索引，则为真。它是由 CREATE INDEX 设置的，但 DROP INDEX 不会立即将它清

名称	类型	描述
		除。如果 VACUUM 进程检测一个表没有索引，将会把它将清理 relcmindex 字段，将值设置为假。
relisshared	boolean	如果该表在 PanWeiDB 中由所有数据库共享则为真。只有某些系统表（比如 pg_database）是共享的。
relpersistence	"char"	❖ p：表示永久表。 ❖ u：表示非日志表。 ❖ g：表示临时表。
relkind	"char"	❖ r：表示普通表。 ❖ i：表示索引。 ❖ I：表示分区表 GLOBAL 索引。 ❖ S：表示序列。 ❖ L：表示 Large 序列。 ❖ v：表示视图。 ❖ c：表示复合类型。 ❖ t：表示 TOAST 表。 ❖ f：表示外表。 ❖ m：表示物化视图。
relnatts	smallint	关系中用户字段数目（除了系统字段以外）。在 pg_attribute 里肯定有相同数目对应行。
relchecks	smallint	表里的检查约束的数目，参阅 pg_constraint 表。
relcmoids	boolean	如果为关系中每行都生成一个 OID 则为真。
relcmsecids	boolean	表示表上是否应用了安全标记（强制访问控制功能）
relcmpkey	boolean	如果这个表有一个（或者曾经有一个）主键，则为真。
relcmrules	boolean	如表有规则就为真。是否有规则可参考系统表

名称	类型	描述
		PG_REWRITE。
relcmtriggers	boolean	True 表示表中有触发器，或者曾经有过触发器。系统表 pg_trigger 中记录了表和视图的触发器。
relcmsubclass	boolean	如果有（或者曾经有）任何继承的子表，为真。
relcmprs	tinyint	表示是否启用表的启用压缩特性。需要特别注意，当且仅当批量插入才会触发压缩，普通的 CRUD 并不能够触发压缩。 ❖ 0 表示其他不支持压缩的表（主要是指系统表，不支持压缩属性的修改操作）。 ❖ 1 表示表数据的压缩特性为 NOCOMPRESS 或者无指定关键字。 ❖ 2 表示表数据的压缩特性为 COMPRESS。
relcmclusterkey	boolean	是否有局部聚簇存储。
relrowmovement	boolean	针对分区表进行 update 操作时，是否允许行迁移。 ❖ true：表示允许行迁移。 ❖ false：表示不允许行迁移。
parttype	"char"	表或者索引是否具有分区表的性质。 ❖ a：表示 PG 风格的分区表。 ❖ b：表示 PG 风格的子分区表。 ❖ p：表示带有分区表性质。 ❖ n：表示没有分区表特性。 ❖ v：表示该表为 HDFS 的 Value 分区表。 ❖ s：表示该表为二级分区表。
relfrozenxid	xid32	该表中所有在这个之前的事务 ID 已经被一个固定的（"frozen"）事务 ID 替换。该字段用

名称	类型	描述
		于跟踪此表是否需要为了防止事务 ID 重叠（或者允许收缩 pg_clog）而进行清理。如果该关系不是表则为零（InvalidTransactionId）。 为保持前向兼容，保留此字段，新增 relfrozenxid64 用于记录此信息。
relacl	aclitem[]	访问权限。查询的回显结果为以下形式： rolename=xxxx/yyyy --赋予一个角色的权限 =xxxx/yyyy --赋予 public 的权限 xxxx 表示赋予的权限，yyyy 表示授予这个权限的角色。权限的参数说明请参见表 2。
reloptions	text[]	表或索引的访问方法，使用 keyword=value 格式的字符串。
relreplident	"char"	逻辑解码中解码列的标识： ❖ d = 默认 （主键，如果存在）。 ❖ n = 无。 ❖ f = 所有列。 ❖ i = 索引的 indisreplident 被设置或者为默认。
relfrozenxid64	xid	该表中所有在这个之前的事务 ID 已经被一个固定的（"frozen"）事务 ID 替换。该字段用于跟踪此表是否需要为了防止事务 ID 重叠（或者允许收缩 pg_clog）而进行清理。如果该关系不是表则为零（InvalidTransactionId）。
relbucket	oid	PG_HASHBUCKET 中的桶信息。
relbucketkey	int2vector	哈希分区列号。
relminmxid	xid	该表中所有在这个之前的多事务 ID 已经被一个事务 ID 替换。这用于跟踪该表是否需要为了防止多事务 ID 重叠或者允许收缩 pg_clog

名称	类型	描述
		而进行清理。如果该关系不是表则为零（InvalidTransactionId）。
relrewrite	oid	当 ALTER TABLE 语句需要重写表时，该字段为重写目标表的 OID，其余情况该字段为 0（InvalidOid）。

表 2 权限的参数说明

参数	参数说明
r	SELECT（读）
w	UPDATE（写）
a	INSERT（插入）
d	DELETE
D	TRUNCATE
x	REFERENCES
t	TRIGGER
X	EXECUTE
U	USAGE
C	CREATE
c	CONNECT
T	TEMPORARY
A	ALTER
P	DROP
m	COMMENT
i	INDEX

v	VACUUM
*	给前面权限的授权选项

5.3.47.PG_COLLATION

PG_COLLATION 系统表描述可用的排序规则，本质上从一个 SQL 名称映射到操作系统本地类别。

表 1 PG_COLLATION 字段

名称	类型	引用	描述
oid	oid	-	行标识符（隐藏列，必须显式指定列名选择）。
collname	name	-	排序规则名（每个名称空间和编码唯一）。
collnamespace	oid	PG_NAMESPACE.oid	包含这个排序规则的名称空间的 OID。
collowner	oid	PG_AUTHID.oid	排序规则的所有者。
collencoding	integer	-	排序规则可用的编码，兼容 PostgreSQL 所有的字符编码类型，如果适用于任意编码为-1。
collcollate	name	-	这个排序规则对象的 LC_COLLATE。
collctype	name	-	这个排序规则对象的 LC_CTYPE。
collpadattribute	name	-	列校对规则。 NO PAD：把字符串尾端的空格当作一个字符处理，即字符串等值比较不忽略尾端空格。 PAD SPACE：字符串等值比较忽略尾端空格。

名称	类型	引用	描述
collisdef	bool	-	这个排序规则是否是所属字符集的默认排序规则。

5.3.48.PG_CONSTRAINT

PG_CONSTRAINT 系统表存储表上的检查约束、主键和唯一约束。

表 1 PG_CONSTRAINT 字段

名称	类型	描述
oid	oid	行标识符（隐藏列，必须显式指定列名选择）。
conname	name	约束名称（不一定是唯一的）。
connamespace	oid	包含这个约束的名称空间的 OID。
contype	“char”	c = 检查约束。p = 主键约束。u = 唯一约束。t = 触发器约束。
condeferrable	boolean	这个约束是否可以推迟。
condeferred	boolean	缺省时这个约束是否可以推迟。
convalidated	boolean	约束是否有效。目前，只有外键和 CHECK 约束可将其设置为 FALSE。
conrelid	oid	这个约束所在的表；如果不是表约束则为 0。
contypid	oid	这个约束所在的域；如果不是一个域约束则为 0。
conindid	oid	与约束关联的索引 ID。
confrelid	oid	如果是外键，则为参考的表；否则为 0。
confupdtype	“char”	外键更新动作代码。a = 没动作。r = 限制。c = 级联。n = 设置为 null。d = 设置为缺省。
confdeltype	“char”	外键删除动作代码。a = 没动作。r = 限制。c = 级联。n = 设置为 null。d = 设置为缺省。
confmatchtype	“char”	外键匹配类型。f = 全部。p = 部分。

名称	类型	描述
		u = 未指定（在 f 的基础上允许匹配 NULL 值）。
conislocal	boolean	是否是为关系创建的本地约束。
coninhcount	integer	约束直接继承父表的数目。继承父表数非零时，不能删除或重命名该约束。
connoinherit	boolean	是否可以被继承。
consoft	boolean	是否为信息约束（Informational Constraint）。
conopt	boolean	是否使用信息约束优化执行计划。
conenable	boolean	是否启用约束。
conkey	smallint[]	如果是表约束，则是约束控制的字段列表。
confkey	smallint[]	如果是一个外键，是参考的字段的列表。
conpfeqop	oid[]	如果是一个外键，是做 PK=FK 比较的相等操作符 ID 的列表。
conppeqop	oid[]	如果是一个外键，是做 PK=PK 比较的相等操作符 ID 的列表。
conffeqop	oid[]	如果是一个外键，是做 FK=FK 比较的相等操作符 ID 的列表。由于当前不支持外键，所以值为空。
conexclop	oid[]	如果是一个排他约束，是列的排他操作符 ID 列表。
conbin	pg_node_tree	如果是检查约束，那就是其表达式的内部形式。
consrc	text	如果是检查约束，则是表达式的人类可读形式。
conincluding	smallint[]	不用做约束，但是会包含在 INDEX 中的属性列。

须知

- consrc 在被引用的对象改变之后不会被更新，它不会跟踪字段的名称修改。与其依赖这个字段，最好还是使用 pg_get_constraintdef()来抽取一个检查约束的定义。
- pg_class.relchecks 需要和在此表上为给定关系找到的检查约束的数目一致。

5.3.49.PG_CONVERSION

PG_CONVERSION 系统表描述编码转换信息。

表 1 PG_CONVERSION 字段

名称	类型	引用	描述
oid	oid	-	行标识符（隐藏属性，必须明确选择）。
conname	name	-	转换名称（在一个名称空间里是唯一的）。
connamespace	oid	PG_NAMESPACE.oid	包含这个转换的名称空间的 OID。
conowner	oid	PG_AUTHID.oid	编码转换的属主。
conforencoding	integer	-	源编码 ID。
contoencoding	integer	-	目的编码 ID。
conproc	regproc	PG_PROC.proname	转换过程。
condefault	boolean	-	如果这是缺省转换则为真。

5.3.50.PG_DATABASE

PG_DATABASE 系统表存储关于可用数据库的信息。

表 1 PG_DATABASE 字段

名称	类型	描述
datname	name	数据库名称。

名称	类型	描述
datdba	oid	数据库所有人，通常为其创建者。
encoding	integer	数据库的字符编码方式。
datcollate	name	数据库使用的排序顺序。
datctype	name	数据库使用的字符分类。
datistemplate	Boolean	是否允许作为模板数据库。
datallowconn	Boolean	如果为假，则没有用户可以连接到这个数据库。这个字段用于保护 template0 数据库不被更改。
datconnlimit	integer	该数据库上允许的最大并发连接数，-1 表示无限制。
datlastsysoid	oid	数据库里最后一个系统 OID 。
datfrozenxid	xid32	用于跟踪该数据库是否需要为了防止事务 ID 重叠而进行清理。当前版本该字段已经废弃使用，为保持前向兼容，保留此字段，新增 datfrozenxid64 用于记录此信息。
dattablespace	oid	数据库的缺省表空间。
datcompatibility	name	数据库兼容模式，当前支持四种兼容模式：A、B、C、PG，分别表示兼容 Oracle、MySQL 、TD 和 PostgreSQL。
datlowercasetablenames	integer	参数 lower_case_table_names 的当前值。
datacl	aclitem[]	访问权限。
datfrozenxid64	xid	用于跟踪该数据库是否需要为了防止事务 ID 重叠而进行清理。

名称	类型	描述
datminmxid	xid	该数据库中中所有在这个之前的多事务 ID 已经被一个事务 ID 替换。这用于跟踪该数据库是否需要为了防止事务 ID 重叠或者允许收缩 pg_clog 而进行清理。它是此数据库中所有表的 pg_class.relminmxid 中的最小值。
datpadattribute	name	列校对规则。 ❖ NO PAD：把字符串尾端的空格当作一个字符处理，即字符串等值比较不忽略尾端空格。 ❖ PAD SPACE：字符串等值比较忽略尾端空格。
dbmaxsize	text	数据库空间最大值。
spcthreshold	smallint	数据库空间告警阈值。已使用空

5.3.51.PG_DB_ROLE_SETTING

PG_DB_ROLE_SETTING 系统表存储数据库运行时每个角色与数据绑定的配置项的默认值。

表 1 PG_DB_ROLE_SETTING 字段

名称	类型	描述
setdatabase	oid	配置项所对应的数据库，如果未指定数据库，则为 0。
setrole	oid	配置项所对应的角色，如果未指定角色，则为 0。
setconfig	text[]	运行时配置项的默认值，配置方法参考：GUC 参数说明->配置运行参数->重设参数章节中表 2。

5.3.52.PG_DEFAULT_ACL

PG_DEFAULT_ACL 系统表存储为新建对象设置的初始权限。

表 1 PG_DEFAULT_ACL 字段

名称	类型	描述
oid	oid	行标识符（隐藏列，必须显式指定列名选择）。
defaclrole	oid	与此权限相关的角色 ID。
defaclnamespace	oid	与此权限相关的名称空间，如果没有，则为 0。
defaclobjtype	“char”	此权限的对象类型。r 表示表或视图。S 表示序列。f 表示函数。T 表示类型。K 表示客户端主密钥。k 表示列加密密钥。
defaclacl	aclitem[]	创建该类型时所拥有的访问权限。

5.3.53.PG_DEPEND

PG_DEPEND 系统表记录数据库对象之间的依赖关系。这个信息允许 DROP 命令找出哪些其他对象必须由 DROP CASCADE 删除，或者是在 DROP RESTRICT 的情况下避免删除。

这个表的功能类似 PG_SHDEPEND（参考：系统表和系统视图->系统表->PG_SHDEPEND 章节），用于记录那些在 PanWeiDB 之间共享的对象之间的依赖性关系。

表 1 PG_DEPEND 字段

名称	类型	引用	描述
classid	oid	PG_CLASS.oid	有依赖对象所在系统表的 OID。
objid	oid	任意 OID 属性	指定的依赖对象的 OID。
objsubid	integer	-	对于表字段，这个是该属性的字段数（objid 和 classid 引用表本身）。对于所有其他对象类型，目前这个字段是 0。
refclassid	oid	PG_CLASS.oid	被引用对象所在的系统表的 OID。
refobjid	oid	任意 OID 属性	指定的被引用对象的 OID。
refobjsubid	integer	-	对于表字段，这个是该字段的字段号（refobjid 和 refclassid 引用表本身）。对于所有其他对象类型，

名称	类型	引用	描述
			目前这个字段是 0。
deptype	“char”	-	一个定义这个依赖关系特定语义的代码。

在所有情况下，一个 PG_DEPEND 记录表示被引用的对象不能在有依赖的对象被删除前删除。不过，这里还有几种由 deptype 定义的情况：

- DEPENDENCY_NORMAL (n)：独立创建的对象之间的一般关系。有依赖的对象可以在不影响被引用对象的情况下删除。被引用对象只有在声明了 CASCADE 的情况下删除，这时有依赖的对象也被删除。例子：一个表字段对其数据类型有一般依赖关系。
- DEPENDENCY_AUTO (a)：有依赖对象可以和被引用对象分别删除，并且如果删除了被引用对象则应该被自动删除（不管是 RESTRICT 或 CASCADE 模式）。例子：一个表上面的命名约束是在该表上的自动依赖关系，因此如果删除了表，它也会被删除。
- DEPENDENCY_INTERNAL (i)：有依赖的对象是作为被引用对象的一部分创建的，实际上只是它的内部实现的一部分。DROP 有依赖对象是不能直接允许的（将告诉用户发出一条删除被引用对象的 DROP）。一个对被引用对象的 DROP 将传播到有依赖对象，不管是否声明了 CASCADE。
- DEPENDENCY_EXTENSION (e)：依赖对象是被依赖对象 extension 的一个成员（参考：系统表和系统视图->系统表->PG_EXTENSION 章节）。依赖对象只可以通过在被依赖对象上 DROP EXTENSION 删除。函数上这个依赖类型和内部依赖一样动作，但是它为了清晰和简化 gs_dump 保持分开。
- DEPENDENCY_PIN (p)：没有依赖对象；这种类型的记录标志着系统本身依赖于被引用对象，因此这个对象决不能被删除。这种类型的记录只有在 initdb 的时候创建。有依赖对象的字段里是零。

5.3.54.PG_DESCRIPTION

PG_DESCRIPTION 系统表可以给每个数据库对象存储一个可选的描述（注释）。许多内置的系统对象的描述提供了 PG_DESCRIPTION 的初始内容。

这个表的功能类似 PG_SHDESCRIPTION（参考：系统表和系统视图->系统表->PG_SHDESCRIPTION 章节），用于记录 PanWeiDB 范围内共享对象的注释。

表 1 PG_DESCRIPTION 字段

名称	类型	引用	描述
objoid	oid	任意 OID 属性	这条描述所描述的对象的 OID。
classoid	oid	PG_CLASS.oid	这个对象出现的系统表的 OID。
objsubid	integer	-	对于一个表字段的注释，它是字段号（objoid 和 classoid 指向表自身）。对于其他对象类型，它是零。
description	text	-	对该对象描述的任意文本。

5.3.55.PG_DIRECTORY

PG_DIRECTORY 系统表用于保存用户添加的 directory 对象可以通过 CREATE DIRECTORY（参考：SQL 语法参考->SQL 语法->CREATE DIRECTORY 章节）语句向该表中添加记录，目前只有系统管理员用户可以向该表中添加记录。

表 1 PG_DIRECTORY 字段

名称	类型	描述
oid	oid	行标识符（隐藏列，必须显式指定列名选择）。
dirname	name	目录对象的名称。
owner	oid	目录对象的所有者。
dirpath	text	目录路径。
diracl	aclitem[]	访问权限。

5.3.56.PG_ENUM

PG_ENUM 系统表包含显示每个枚举类型值和标签的记录。给定枚举类型的内部表示实际上是 PG_ENUM 里面相关行的 OID。

表 1 PG_ENUM 字段

名称	类型	引用	描述
oid	oid	-	行标识符（隐藏列，必须显式指定列名选择）。
enumtypid	oid	PG_TYPE.oid	拥有这个枚举值的 pg_type 记录的 OID。
enumsortorder	real	-	这个枚举值在它的枚举类型中的排序位置。
enumlabel	name	-	这个枚举值的文本标签。

PG_ENUM 行的 OID 跟着一个特殊规则：偶数的 OID 保证用和它们的枚举类型一样的排序顺序排序。也就是，如果两个偶数 OID 属于相同的枚举类型，那么较小的 OID 必须有较小 enumsortorder 值。奇数 OID 需要毫无关系的排序顺序。这个规则允许枚举比较例程在许多常见情况下避开目录查找。创建和修改枚举类型的例程只要可能就尝试分配偶数 OID 给枚举值。

当创建了一个枚举类型时，它的成员赋予了排序顺序位置 1 到 n。但是随后添加的成员可能会分配 enumsortorder 的负值或分数值。对这些值的唯一要求是它们要正确的排序和在每个枚举类型中唯一。

5.3.57.PG_EXTENSION

PG_EXTENSION 系统表存储关于所安装扩展的信息。PanWeiDB 默认扩展是 PLPGSQL 和 MOT_FDW。

表 1 PG_EXTENSION

名称	类型	描述
oid	oid	数据库对象 id。
extname	name	扩展名。
extowner	oid	扩展的所有者。
extnamespace	oid	扩展导出对象的名称空间。
extrelocatable	Boolean	标识此扩展是否可迁移到其他名称空间，true 表示允许。
extversion	text	扩展的版本号。
extconfig	oid[]	扩展的配置信息。
extcondition	text[]	扩展配置信息的过滤条件。

5.3.58.PG_EXTENSION_DATA_SOURCE

PG_EXTENSION_DATA_SOURCE 系统表存储外部数据源对象的信息。一个外部数据源对象（Data Source）包含了外部数据库的一些口令编码等信息，主要配合 Extension Connector 使用。

表 1 PG_EXTENSION_DATA_SOURCE 字段

名称	类型	引用	描述
oid	oid	-	行标识符（隐藏列，必须显式指定列名选择）。
srcname	name	-	外部数据源对象的名称。
srcowner	oid	PG_AUTHID.oid	外部数据源对象的所有者。
srctype	text	-	外部数据源对象的类型，缺省为空。
srcversion	text	-	外部数据源对象的版本，缺省为空。
srcacl	aclitem[]	-	访问权限。
srcoptions	text[]	-	外部数据源对象的指定选项，使用“keyword=value”格式的字符串。

5.3.59.PG_EVENT_TRIGGER

PG_EVENT_TRIGGER 系统表，用于维护事件触发器的元信息。

名称	类型	引用	描述
Event	Name	-	触发器名称（必须唯一）。
Evtevent	Name	-	触发器触发事件的标识符。
Evtowner	Oid	Pg_authid_oid	事件触发器的拥有者。
Evtfoid	Oid	Pg_proc_oid	将被调用的函数。

Evtenabled	Char	-	控制事件触发器触发的会话复制角色模式。 ❖ O：触发器在“origin”和“local”模式触发。 ❖ D：触发器被禁用。 ❖ R：触发器在“replica”模式触发。 ❖ A：触发器总是触发。
Evttags	Text[]	-	触发器将触发的命令标签。如果为空，此触发器的触发不会受命令标签的限制。

5.3.60.PG_FOREIGN_DATA_WRAPPER

PG_FOREIGN_DATA_WRAPPER 系统表存储外部数据封装器定义。一个外部数据封装器是在外部服务器上驻留外部数据的机制，是可以访问的。

表 1 PG_FOREIGN_DATA_WRAPPER 字段

名称	类型	引用	描述
oid	oid	-	行标识符（隐藏列，必须显式指定列名选择）。
fdwname	name	-	外部数据封装器名。
fdwowner	oid	PG_AUTHID.oid	外部数据封装器的所有者。
fdwhandler	oid	PG_PROC.oid	引用一个负责为外部数据封装器提供扩展例程的处理函数。如果没有提供处理函数则为零。
fdwvalidator	oid	PG_PROC.oid	引用一个验证器函数，这个验证器函数负责验证给予外部数据封装器的选项、外部服务器选项和使用外部数据封装器的用户映射的有效性。如果没有提供验证器函数则为

名称	类型	引用	描述
			零。
fdwacl	aclitem[]	-	访问权限。
fdwoptions	text[]	-	外部数据封装器指定选项，使用“keyword=value”格式的字符串。

5.3.61.PG_FOREIGN_SERVER

PG_FOREIGN_SERVER 系统表存储外部服务器定义。一个外部服务器描述了一个外部数据源，例如一个远程服务器。外部服务器通过外部数据封装器访问。

表 1 PG_FOREIGN_SERVER 字段

名称	类型	引用	描述
oid	oid	-	行标识符（隐藏列，必须显式指定列名选择）。
srvname	name	-	外部服务器名。
srvowner	oid	PG_AUTHID.oid	外部服务器的所有者。
srvfdw	oid	PG_FOREIGN_DATA_WRAPPER.oid	这个外部服务器的外部数据封装器的 OID。
srvtype	text	-	服务器的类型（可选）。
srvversion	text	-	服务器的版本（可选）。
srvacl	aclitem[]	-	访问权限。
srvoptions	text[]	-	外部服务器指定选项，使用“keyword=value”格式的字符串。

5.3.62.PG_FOREIGN_TABLE

PG_FOREIGN_TABLE 系统表存储外部表的辅助信息。

表 1 PG_FOREIGN_TABLE 字段

名称	类型	描述
ftrelid	oid	外部表的 ID。
ftserver	oid	外部表的所在服务器。

名称	类型	描述
ftwriteonly	Boolean	外部表是否可写。取值如下： t（true）：表示可写。 f（false）：表示不可写。
ftoptions	text[]	外部表的可选项，具体参考 CREATE FOREIGN TABLE 语法说明。

5.3.63.PG_HASHBUCKET

PG_HASHBUCKET 系统表存储 hash bucket 信息。

表 1 PG_HASHBUCKET 字段

名称	类型	描述
oid	oid	行标识符（隐含字段，必须明确选择）。
bucketid	oid	对 bucketvector 计算的 hash 值，通过 hash 值可以加速对 bucketvector 的查找。
bucketcnt	integer	包含分片的个数。
bucketmapsize	integer	所有 DN 上包含的分片总数。
bucketref	integer	预留字段，默认值为 1。
bucketvector	oidvector_extend	记录此行 bucket 信息包含的所有 bucket 的 id，在此列上建立唯一索引，具有相同 bucketid 信息的表共享同一行 pg_hashbucket 数据。

5.3.64.PG_INDEX

PG_INDEX 系统表存储索引的一部分信息，其他的信息大多数在 PG_CLASS(参考：系统表和系统视图->系统表->PG_CLASS 章节）中。

表 1 PG_INDEX 字段

名称	类型	描述
indexrelid	oid	这个索引在 pg_class 里的记录的

名称	类型	描述
		OID。
indrelid	oid	使用这个索引的表在 pg_class 里的记录的 OID。
indnatts	smallint	索引中的字段数目。
indisunique	Boolean	如果为真，这是个唯一索引。如果为假，这不是唯一索引。
indisprimary	Boolean	如果为真，该索引代表该表的主键。这个字段为真的时候 indisunique 总是为真。如果为假，该索引不是该表的主键。
indisexclusion	Boolean	如果为真，该索引支持排他约束。如果为假，该索引不支持排他约束。
indimmediate	Boolean	如果为真，在插入数据时会立即进行唯一性检查。如果为假，在插入数据时不会进行唯一性检查。
indisclustered	Boolean	如果为真，则该表最后在这个索引上建了簇。如果为假，则该表没有再这个索引上建簇。
indisusable	Boolean	如果为真，该索引对 insert/select 可用。如果为假，该索引对 insert/select 不可用。
indisvalid	Boolean	如果为真，则该索引可以用于查询。如果为假，则该索引可能不完整，仍然必须在 INSERT/UPDATE 操作时进行更新，不过不能安全的用于查询。如果是唯一索引，则唯一属性也将不为真。
indcheckxmin	Boolean	如果为 true，查询不能使用索引，直到 pg_index 此行的 xmin 低于其快照的 TransactionXmin，因为该表可能包含它们能看到的不兼容行断开的 HOT 链。如果为 false，查询可以用于索引。
indisready	Boolean	如果为真，表示此索引对插入数据是可用的，否则，在插入或修改数据时忽略

名称	类型	描述
		此索引。
indkey	int2vector	这是一个包含 indnatts 值的数组，这些数组值表示这个索引所建立的表字段。比如一个值为 1 3 的意思是第一个字段和第三个字段组成这个索引键字。这个数组里的零表明对应的索引属性是在这个表字段上的一个表达式，而不是一个简单的字段引用。
indcollation	oidvector	索引用到的各列的 ID。
indclass	oidvector	对于索引键字里面的每个字段，这个字段都包含一个指向所使用的操作符类的 OID，参阅 pg_opclass 获取细节。
indoption	int2vector	存储列前标识位，该标识位是由索引的访问方法定义。
indexprs	pg_node_tree	表达式树（以 nodeToString()形式表现）用于那些非简单字段引用的索引属性。它是一个列表，个数与 INDKEY 中的零值个数相同。如果所有索引属性都是简单的引用，则为空。
indpred	pg_node_tree	部分索引断言的表达式树（以 nodeToString()的形式表现）。如果不是部分索引，则是空字符串。
indisreplident	Boolean	如果为真，则此索引的列成为逻辑解码的解码列。如果为假，则此索引的列不是逻辑解码的解码列。
indnkeyatts	smallint	索引中的总字段数，超出 indnatts 的部分不参与索引查询。

5.3.65.PG_INHERITS

PG_INHERITS 系统表记录关于表继承层次的信息。数据库里每个直接的子系表都有一条记录。间接的继承可以通过追溯记录链来判断。

表 1 PG_INHERITS 字段

名称	类型	引用	描述
inhrelid	oid	PG_CLASS.oid	子表的 OID。
inhparent	oid	PG_CLASS.oid	父表的 OID。
inhseqno	integer	-	如果一个子表存在多个直系父表（多重继承），这个数字表明此继承字段的排列顺序。计数从 1 开始。

5.3.66.PG_JOB

PG_JOB 系统表存储用户创建的定时任务的任务详细信息，定时任务线程定时轮询 pg_job 系统表中的时间，当任务到期会触发任务的执行，并更新 pg_job 表中的任务状态。该系统表属于 Shared Relation，所有创建的 job 记录对所有数据库可见。

表 1 PG_JOB 字段

名称	类型	描述
oid	oid	行标识符（隐藏列，必须显式指定列名选择）。
job_id	bigint	作业 ID，主键，是唯一的（有唯一索引）
current_postgres_pid	bigint	如果当前任务已被执行，那么此处记录运行此任务的 gaussdb 线程 ID。默认为 -1，表示此任务未被执行过。
log_user	name	创建者的 UserName
priv_user	name	作业执行者的 UserName
dbname	name	标识作业要在哪个数据库执行的数据库名称
node_name	name	标识当前作业是在哪个数据库主节点上创建和执行
job_status	“char”	当前任务的执行状态，取值范围：('r', 's', 'f', 'd')，默认为's'，取值含义：Status of job step: r=running, s=successfully

名称	类型	描述
		finished, f=job failed, d=disable 当 job 连续执行失败 16 次，会将 job_status 自动设置为失效状态 'd'，后续不再执行该 job。注：当用户将定时任务关闭（即：guc 参数 job_queue_processes 为 0 时），由于监控 job 执行的线程不会启动，所以该状态不会根据 job 的实时状态进行设置，用户不需要关注此状态。只有当开启定时任务功能（即：guc 参数 job_queue_processes 为非 0 时），系统才会根据当前 job 的实时状态刷新该字段值。
start_date	timestamp without time zone	作业第一次开始执行时间，时间精确到毫秒。
next_run_date	timestamp without time zone	下次定时执行任务的时间，时间精确到毫秒。
failure_count	smallint	失败计数，作业连续执行失败 16 次，不再继续执行。
interval	text	作业执行的重复时间间隔。
last_start_date	timestamp without time zone	上次运行开始时间，时间精确到毫秒。
last_end_date	timestamp without time zone	上次运行的结束时间，时间精确到毫秒。
last_suc_date	timestamp without time zone	上次成功运行的开始时间，时间精确到毫秒。
this_run_date	timestamp without time	正在运行任务的开始时间，时间精确到毫秒。

名称	类型	描述
	zone	
nspname	name	标识作业执行时的 schema 的名称。
job_name	text	DBE_SCHEDULER 定时任务专用，定时任务名称。
end_date	timestamp without time zone	DBE_SCHEDULER 定时任务专用，定时任务失效时间，时间精确到毫秒。
enable	boolean	DBE_SCHEDULER 定时任务专用，定时任务启用状态：true：启用 false：未启用
failure_msg	text	最新一次执行任务报错信息。

5.3.67.PG_JOB_LOG

PG_JOB_LOG 系统表用于记录 job_log，其列与 ALL_SCHEDULER_JOB_LOGALL_SCHEDULER_JOB_LOG 中的列相同。

5.3.68.PG_JOB_PROC

PG_JOB_PROC 系统表对应 PG_JOB（参考：系统表和系统视图->系统表->PG_JOB 章节）表中每个任务的作业内容（包括：PL/SQL 代码块、匿名块）。将存储过程信息独立出来，如果放到 PG_JOB 中，被加载到共享内存的时候，会占用不必要的空间，所以在使用的时候再进行查询获取。

表 1 PG_JOB_PROC 字段

名称	类型	描述
oid	oid	行标识符（隐藏列，必须显式指定列名选择）。
job_id	integer	外键，关联 pg_job 表中的 job_id。
what	text	作业内容，DBE_SCHEDULER 定时任务中的程序内容。
job_name	text	DBE_SCHEDULER 定时任务专用，定时任务或程序名称。

5.3.69.PG_LANGUAGE

PG_LANGUAGE 系统表登记编程语言，用户可以用这些语言或接口写函数或者存储过程。

表 1 PG_LANGUAGE 字段

名称	类型	引用	描述
oid	oid	-	行标识符（隐藏列，必须显式指定列名选择）。
lanname	name	-	语言的名称。
lanowner	oid	PG_AUTHID.oid	语言的所有者。
lanispl	Boolean	-	对于内部语言而言是假（比如 SQL），对于用户定义的语言则是真。目前，gs_dump 仍然使用这个东西判断哪种语言需要转储，但是这些可能在将来被其他机制取代。
lanpltrusted	Boolean	-	如果这是可信语言则为真，意味着系统相信它不会被授予任何正常 SQL 执行环境之外的权限。只有初始用户可以用不可信的语言创建函数。
lanplcallfoid	oid	PG_PROC.oid	对于非内部语言，这是指向该语言处理器的引用，语言处理器是一个特殊函数，负责执行以某种语言写的所有函数。
laninline	oid	PG_PROC.oid	这个字段引用一个负责执行“inline”匿名代码块的函数（DO 块）。如果不支持内联块则为零。
lanvalidator	oid	PG_PROC.oid	这个字段引用一个语言校验器函数，它负责检查新创建的函数的语法和有效性。如果没有提供校验器，则为零。
lanacl	aclitem[]	-	访问权限。

5.3.70.PG_LARGEOBJECT

PG_LARGEOBJECT 系统表保存那些标记着“大对象”的数据。一个大对象是使用其创建时分配的 OID 标识的。每个大对象都分解成足够小的小段或者“页面”以便以行的形式存储在 PG_LARGEOBJECT 里。每页的数据定义为 LOBLKSIZE。

需要有系统管理员权限才可以访问此系统表。

表 1 PG_LARGEOBJECT 字段

名称	类型	引用	描述
loid	oid	PG_LARGEOBJECT_METADATA.oid	包含本页的大对象的标识符。
pageno	integer	-	本页在其大对象数据中的页码从零开始计算。
data	bytea	-	存储在大对象中的实际数据。这些数据绝不会超过 LOBLKSIZE 字节，而且可能更少。

PG_LARGEOBJECT 的每一行保存一个大对象的一个页面，从该对象内部的字节偏移（pageno * LOBLKSIZE）开始。这种实现允许松散的存储：页面可以丢失，而且可以比 LOBLKSIZE 字节少（即使它们不是对象的最后一页）。大对象内丢失的部分读做零。

5.3.71.PG_LARGEOBJECT_METADATA

PG_LARGEOBJECT_METADATA 系统表存储与大数据相关的元数据。实际的大对象数据存储在[PG_LARGEOBJECT（参考：系统表和系统视图->系统表->PG_LARGEOBJECT 章节）里。

表 1 PG_LARGEOBJECT_METADATA 字段

名称	类型	引用	描述
oid	oid	-	行标识符（隐藏列，必须显式指定列名选择）。

名称	类型	引用	描述
lomowner	oid	PG_AUTHID.oid	大对象的所有者。
lomacl	aclitem[]	-	访问权限。

5.3.72.PG_NAMESPACE

PG_NAMESPACE 系统表存储名称空间，即存储 schema 相关的信息。

表 1 PG_NAMESPACE 字段

名称	类型	描述
oid	oid	行标识符（隐藏列，必须显式指定列名选择）。
nspname	name	名称空间的名称。
nspowner	oid	名称空间的所有者。
nsptimeline	bigint	在数据库节点上创建此命名空间时的时间线。此字段为内部使用，仅在数据库节点上有效。
nspacl	aclitem[]	访问权限。
in_redistribution	“char”	是否处于重发布状态。
nspblockchain	Boolean	如果为真，则该模式为防篡改模式。如果为假，则此模式为非防篡改模式。

5.3.73.PG_OBJECT

PG_OBJECT 系统表存储限定类型对象（普通表、索引、序列、视图、存储过程和函数）的创建用户、创建时间和最后修改时间。

表 1 PG_OBJECT 字段

名称	类型	描述
object_oid	oid	对象标识符。
object_type	“char”	对象类型：r 表示普通表。i 表示索引。s 表示序列。l 表示 Large 序列。v 表示视图。p 表示存储过程和函数。
creator	oid	创建用户的标识符。
ctime	timestamp with time zone	对象的创建时间。

名称	类型	描述
mtime	timestamp with time zone	对象的最后修改时间，修改行为包括 ALTER 操作和 GRANT、REVOKE 操作。
createcsn	int8	对象创建时的 CSN。
changecsn	int8	对表或索引执行 DDL 操作时的 CSN。
valid	boolean	表示当前对象是否有效，引起对象失效的原因可能有存储过程或视图依赖的引用对象被修改或删除。

须知

- 无法记录初始化数据库（initdb）过程中所创建或修改的对象，即 PG_OBJECT 无法查询到该对象记录。
- 对于升级前创建的对象，再次修改时会记录其修改时间（mtime），对表或索引执行 DDL 操作时会记录其所属事务的事务提交序列号（changecsn）。由于无法得知该对象创建时间，因此 ctime 和 createcsn 为空。
- ctime 和 mtime 所记录的时间为用户当次操作所属事务的起始时间。
- 由扩容引起的对象修改时间也会被记录。
- createcsn 和 changecsn 记录的是用户当次操作所属事务的事务提交序列号。

5.3.74.PG_OPCLASS

PG_OPCLASS 系统表定义索引访问方法操作符类。

每个操作符类为一种特定数据类型和一种特定索引访问方法定义索引字段的语义。一个操作符类本质上指定一个特定的操作符族适用于一个特定的可索引的

字段数据类型。索引的字段实际可用的族中的操作符集是接受字段的数据类型作为它们的左边的输入的那个。

表 1 PG_OPCLASS 字段

名称	类型	引用	描述
oid	oid	-	行标识符（隐藏列，必须显式指定列名选择）。
opcmethod	oid	PG_AM.oid	操作符类所服务的索引访问方法。
opcname	name	-	这个操作符类的名称。
opcnamespace	oid	PG_NAMESPACE.oid	这个操作符类的名称空间。
opcowner	oid	PG_AUTHID.oid	操作符类属主。
opcfamily	oid	PG_OPFAMILY.oid	包含该操作符类的操作符族。
opcintype	oid	PG_TYPE.oid	操作符类索引的数据类型。
opcdefault	boolean	-	如果操作符类是 opcintype 的缺省，则为真。
opckeytype	oid	PG_TYPE.oid	索引数据的类型，如果和 opcintype 相同则为零。

一个操作符类的 opcmethod 必须匹配包含它的操作符族的 opfmethod。

5.3.75.PG_OPERATOR

PG_OPERATOR 系统表存储有关操作符的信息。

表 1 PG_OPERATOR 字段

名称	类型	引用	描述
oid	oid	-	行标识符（隐藏列，必须显式指定列名选择）。
oprname	name	-	操作符的名称。
oprnamespace	oid	PG_NAMESPACE.oid	包含此操作符的名称空间的 OID。
oprowner	oid	PG_AUTHID.oid	操作符所有者。

名称	类型	引用	描述
oprkind	“char”	-	b=infix =中缀（“两边”）。l=前缀（“左边”）。r=后缀（“右边”）。
oprcanmerge	Boolean	-	这个操作符是否支持合并连接。t（true）：表示支持合并连接。f（false）：表示不支持合并连接。
oprcanhash	Boolean	-	这个操作符是否支持 Hash 连接。t（true）：表示支持 Hash 连接。f（false）：表示不支持 Hash 连接。
oprleft	oid	PG_TYPE.oid	左操作数的类型。
oprright	oid	PG_TYPE.oid	右操作数的类型。
oprresult	oid	PG_TYPE.oid	结果类型。
oprcom	oid	PG_OPERATOR.oid	此操作符的交换符，如果存在的话。
oprnegate	oid	PG_OPERATOR.oid	此操作符的反转器，如果存在的话。
oprcode	regproc	PG_PROC.proname	实现这个操作符的函数。
oprrest	regproc	PG_PROC.proname	此操作符的约束选择性计算函数。
oprjoin	regproc	PG_PROC.proname	此操作符的连接选择性计算函数。

5.3.76.PG_OPFAMILY

PG_OPFAMILY 系统表定义操作符族。

每个操作符族是一个操作符和相关支持例程的集合，其中的例程实现为一个特定的索引访问方式指定的语义。另外，族中的操作符都是“兼容的”，通过由访问方式指定的方法。操作符族的概念允许交叉数据类型操作符和索引一起使用，并且合理的使用访问方式的语义的知识。

表 1 PG_OPFAMILY 字段

名称	类型	引用	描述
oid	oid	-	行标识符（隐藏列，必须显式指定列名选择）。
opfmethod	oid	PG_AM.oid	操作符族使用的索引方法。
opfname	name	-	这个操作符族的名称。
opfnamespace	oid	PG_NAMESPACE.oid	这个操作符的名称空间。
opfowner	oid	PG_AUTHID.oid	操作符族的所有者。

定义一个操作符族的大多数信息不在它的 PG_OPFAMILY 行里面，而是在相关的行参考：系统表和系统视图->系统表->PG_AMOP、PG_AMPROC、PG_OPCLASS 章节。

5.3.77.PG_PARTITION

PG_PARTITION 系统表存储数据库内所有分区表（Partitioned Table）、分区（Table Partition）、子分区（Table SubPartition）、索引分区（Index Partition）和分区上 Toast 表（Toast Table）五类对象的信息。分区表索引（Partitioned Index）的信息不在 PG_PARTITION 系统表中保存。

表 1 PG_PARTITION 字段

名称	类型	描述
oid	oid	行标识符（隐含属性，必须明确选择）。
relname	name	分区表、分区、分区上 Toast 表和分区索引的名称。
parttype	"char"	对象类型： ❖ 'r'：分区表（Partitioned Table）。 ❖ 'p'：分区（Table Partition）。 ❖ 's'：子分区（Table SubPartition）。 ❖ 'x'：索引分区（Index Partition）。 ❖ 't'：分区 Toast 表（Toast Table）。
parentid	oid	当对象为分区表或分区时，此字段表示分区表在 PG_CLASS 中的 OID。当对象为 Index Partition 时，此字段表示所属分区表索引（Partitioned Index）的 OID。

partitionid	oid	分区表的分区号。
rangenum	integer	保留字段。
intervalnum	integer	保留字段。
partstrategy	"char"	分区表分区策略，现在仅支持： ❖ 'r'：范围（Range）分区。 ❖ 'v'：数值（Value）分区。 ❖ 'i'：间隔（Interval）分区。 ❖ 'l'：列表（List）分区。 ❖ 'h'：哈希（Hash）分区，目前非 Oracle 兼容模式的 hash 分区表使用哈希分区。 ❖ 'd'：线性哈希（Dynamic Hash）分区，目前 Oracle 兼容模式的分区表使用线性哈希分区。 ❖ 'n'：无效分区。
subpartstrategy	"char"	子分区策略。
relfilenode	oid	Table Partition、Index Partition、分区上 Toast 表的物理存储位置。
reltablespace	oid	Table Partition、Index Partition、分区上 Toast 表所属表空间的 OID。
relpages	double precision	统计信息：Table Partition、Index Partition 的数据页数量。
reltuples	double precision	统计信息：Table Partition、Index Partition 的元组数。
relallvisible	integer	统计信息：Table Partition、Index Partition 的可见数据页数。
reltoastrelid	oid	Table Partition 所对应 Toast 表的 OID。
reltoastidxid	oid	Table Partition 所对应 Toast 表的索引的 OID。
indextblid	oid	Index Partition 对应 Table Partition 的 OID。
indisusable	boolean	分区索引是否可用。
reldeltarelid	oid	Delta 表的 OID。
reldeltaidx	oid	Delta 表的索引表的 OID。
relcudescrelid	oid	CU 描述表的 OID。
relcudescidx	oid	CU 描述表的索引表的 OID。
relfrozenxid	xid32	冻结事务 ID 号。为保持前向兼容，保留此字段，新

		增 relfrozenxid64 用于记录此信息。
intspnum	integer	间隔分区所属表空间的个数。
partkey	int2vector	分区键的列号。
intervaltablespace	oidvector	间隔分区所属的表空间，间隔分区以 Round-Robin 方式落在这些表空间内。
interval	text[]	间隔分区的间隔值。
boundaries	text[]	范围分区和间隔分区的上边界。
transit	text[]	间隔分区的跳转点。
reloptions	text[]	设置 Partition 的存储属性，与 pg_class.reloptions 的形态一样，用“keyword=value”格式的字符串来表示，目前用于在线扩容的信息搜集。
relfrozenxid64	xid	冻结事务 ID 号。
relminmxid	xid	冻结多事务 ID 号。
subpartkey	int2vector	子分区键的列号。
subparttemplate	pg_node_tree	临时子分区，可以避免增加分区时建在默认表空间。
relfrozenxid64	xid	（”frozen”）事务 ID 替换。该字段用于跟踪此表是否需要为了防止事务 ID 重叠（或者允许收缩 pg_clog）而进行清理。如果该关系不是表则为零（InvalidTransactionId）。
relminmxid	xid	该表中所有在这个之前的多事务 ID 已经被一个事务 ID 替换。这用于跟踪该表是否需要为了防止多事务 ID 重叠或者允许收缩 pg_clog 而进行清理。如果该关系不是表则为零(InvalidTransactionId)。

5.3.78.PG_PLTEMPLATE

PG_PLTEMPLATE 系统表存储过程语言的“模板”信息。

表 1 PG_PLTEMPLATE 字段

名称	类型	描述
tmplname	name	这个模板所应用的语言的名称。
tmpltrusted	Boolean	如果语言被认为是可信的，则为真。

名称	类型	描述
tmpldbacreate	Boolean	如果语言是由数据库所有者创建的，则为真。
tmplhandler	text	调用处理器函数的名称。
tmplinline	text	匿名块处理器的名称，若没有则为NULL。
tmplvalidator	text	校验函数的名称，如果没有则为NULL。
tmpllibrary	text	实现语言的共享库的路径。
tmplacl	aclitem[]	模板的访问权限（未使用）。

5.3.79.PG_PROC

PG_PROC 系统表存储函数或过程的信息。

表 1 PG_PROC 字段

名称	类型	描述
oid	oid	行标识符（隐藏列，必须显式指定列名选择）。
proname	name	函数名称。
pronamespace	oid	包含该函数名称空间的 OID。
proowner	oid	函数的所有者。
prolang	oid	这个函数的实现语言或调用接口。
procost	real	估算的执行成本。
prorows	real	估算的影响行的数目。
provariadic	oid	参数元素的数据类型。
protransform	regproc	此函数的简化调用方式。
proisagg	boolean	函数是聚集函数。 ❖ t（true）：表示是。 ❖ f（false）：表示不是。

名称	类型	描述
proiswindow	boolean	函数是窗口函数。 ❖ t（true）：表示是。 ❖ f（false）：表示不是。
prosecdef	boolean	函数是一个安全定义器（也就是一个“setuid”函数）。 ❖ t（true）：表示是。 ❖ f（false）：表示不是。
proleakproof	boolean	函数没副作用。如果函数没有对参数进行防泄露处理，则会抛出错误。 ❖ t（true）：表示没副作用。 ❖ f（false）：表示有副作用。
proisstrict	boolean	如果任何调用参数是空，则函数返回空。这时函数实际上连调用都不调用。不是“strict”的函数必须准备处理空输入。
proretset	boolean	函数返回一个集合（也就是说，指定数据类型的多个数值）。
provolatile	"char"	告诉该函数的结果是否只依赖于它的输入参数，或者还会被外界因素影响。 ❖ i：“不可变的”（immutable）函数，这样的函数对于相同的输入总是产生相同的结果。 ❖ s：“稳定的”（stable）函数它是 s，（对于固定输入）其结果在一次扫描里不变。 ❖ v：“易变”（volatile）函数它是 v，其结果可能在任何时候变化 v 也用于那些有副作

名称	类型	描述
		用的函数，因此调用它们无法得到优化。
proparallel	"char"	标记函数是否并行安全。安全级别说明： ❖ s：并行安全，指与并行查询的使用不冲突的操作。 ❖ r：并行受限，指不能在并行 worker 中执行，但可以在使用并行查询时在 leader 中执行的操作。并行受限操作永远不能发生在 Gather 节点下面，但可以发生在 包含此类节点的计划的其他地方。 ❖ u：并行不安全，指在使用并行查询时不能执行的操作，其至在 Leader 中也不能执行。当查询包含任何并行不安全的内容时，将对该查询完全禁用并行查询。
pronargs	smallint	参数数目。
pronargdefaults	smallint	有默认值的参数数目。
prorettype	oid	返回值的数据类型。
proargtypes	oidvector	一个存放函数参数的数据类型的数组。数组里只包括输入参数（包括 INOUT 参数）此代表该函数的调用签名（接口）。
proallargtypes	oid[]	一个包含函数参数的数据类型的数组。数组里包括所有参数的类型（包括 OUT 和 INOUT 参数），如果所有参数都是 IN 参数，则这

名称	类型	描述
		个字段就会是空。请注意数组下标是以 1 为起点的，而因为历史原因，proargtypes 的下标起点为 0。
proargmodes	"char"[]	一个保存函数参数模式的数组，编码如下： ❖ i 表示 IN 参数。 ❖ o 表示 OUT 参数。 ❖ b 表示 INOUT 参数。 ❖ v 表示 VARIADIC 参数。 ❖ 如果所有参数都是 IN 参数，则这个字段为空。请注意，下标对应的是 proallargtypes 的位置，而不是 proargtypes。
proargnames	text[]	一个保存函数参数的名称的数组。没有名称的参数在数组里设置为空字符串。如果没有一个参数有名称，这个字段将是空。请注意，此数组的下标对应 proallargtypes 而不是 proargtypes。
proargdefaults	pg_node_tree	默认值的表达式树。是 PRONARGDEFAULTS 元素的列表。
proargtypenames	test[]	输入参数类型的名称。
prorettypenames	text	存储过程 OUT 参数的类型名称；或函数返回值的类型名称。 如果存储过程没有 OUT 参数则为空。

名称	类型	描述
prosrc	text	描述函数或存储过程的定义。例如，对于解释型语言来说就是函数的源程序，或者一个链接符号，一个文件名，或者函数和存储过程创建时指定的其他任何函数体内容，具体取决于语言/调用习惯的实现。
probin	text	关于如何调用该函数的附加信息。同样，其含义也是和语言相关的。
proconfig	text[]	函数针对运行时配置变量的本地设置。
proacl	aclitem[]	访问权限。具体请参见 GRANT 和 REVOKE。
prodefaultargpos	int2vector	函数具有默认值的入参的位置。
fencedmode	boolean	函数的执行模式，表示函数是在 fence 还是 not fence 模式下执行。如果是 fence 执行模式，函数的执行会在重新 fork 的进程中执行。 用户创建的 C 函数，fencedmode 字段默认值均为 true，即 fence 模式；系统内建函数，fencedmode 字段均为 false，即 not fence 模式。
proshippable	boolean	表示该函数是否可以下推到数据库节点上执行，默认值是 false。 对于 IMMUTABLE 类型的函数，函数始终可以下推到数据库节点上执行。 对于 STABLE/VOLATILE 类型的函

名称	类型	描述
		数，仅当函数的属性是 SHIPPABLE 的时候，函数可以下推到数据库节点执行。
propackage	boolean	表示该函数是否支持重载，默认值是 false。 ❖ t（true）：表示支持。 ❖ f（false）：表示不支持。
prokind	"char"	表示该对象为函数还是存储过程： ❖ 值为'f'表示该对象为函数。 ❖ 值为'p'表示该对象为存储过程。
proargsrc	text	描述兼容 oracle 语法定义的函数或存储过程的参数输入字符串，包括参数注释。默认值为 NULL。
proisprivate	boolean	描述函数是否是 PACKAGE 内的私有函数，默认为 false。
propackageid	oid	函数所属的 package oid，如果不在 package 内，则为 0。
proargtypesext	oidvector_extend	当函数参数较多时，用来存放函数参数的数据类型的数组。数组里只包括输入参数（包括 INOUT 参数）此代表该函数的调用签名（接口）。
prodefaultargposext	int2vector_extend	当函数参数较多时，函数具有默认值的入参的位置。
allargtypes	oidvector	不区分参数类型，包含存储过程所有参数（包含入参、出参、INOUT 参数）。
allargtypesext	oidvector_extend	当函数参数较多时，用来存放函数

名称	类型	描述
		参数的数据类型的数组。数组里包含所有参数（包含入参、出参、INOUT 参数）。
prosrcoffset	integer	描述函数或存储过程的偏移量。
protypeoid	oid	对象类型的 OID。
protypekind	"char"	对象方法类别。
isfinal	boolean	对象方法是否继承。

5.3.80.PG_PUBLICATION

PG_PUBLICATION 系统表存储当前数据库中创建的所有 publication。

表 1 PG_PUBLICATION 字段

名称	类型	描述
pubname	name	publication 的名称。
pubowner	oid	publication 的拥有者。
puballtables	boolean	如果为真，这个 publication 自动包括数据库中的所有表，包括未来将会创建的任何表。
pubinsert	boolean	如果为真，为 publication 中的表复制 INSERT 操作。
pubupdate	boolean	如果为真，为 publication 中的表复制 UPDATE 操作。
pubdelete	boolean	如果为真，为 publication 中的表复制 DELETE 操作。
pubtruncate	boolean	如果为真，则发布时指定 publish 选项包含 truncate 操作。

名称	类型	描述
pubddl	bigint	❖ 发布时没有指定 ddl，其值为 0。 ❖ 指定 ddl='table'时，其值为 1。 ❖ 指定 ddl='all'时，其值为-1。

5.3.81.PG_PUBLICATION_REL

系统表 PG_PUBLICATION_REL 包含当前数据库中的表和 publication 之间的映射，这是一种多对多映射。

表 1 PG_PUBLICATION_REL 字段

名称	类型	引用	描述
oid	oid		行标识符（隐藏列，必须显式指定列名选择）。
prpubid	oid	-	对 publication 的引用。
prrelid	oid	-	对表的引用。

5.3.82.PG_RANGE

PG_RANGE 系统表存储关于范围类型的信息。除了 PG_TYPE（参考：系统表和系统视图->系统表->PG_TYPE 章节）里类型的记录。

表 1 PG_RANGE 字段

名称	类型	引用	描述
rngtypid	oid	PG_TYPE.oid	范围类型的 OID。
rngsubtype	oid	PG_TYPE.oid	这个范围类型的元素类型（子类型）的 OID。
rngcollation	oid	PG_COLLATION.oid	用于范围比较的排序规则的 OID，如果没有则为零。
rngsubopc	oid	PG_OPCLASS.oid	用于范围比较的子类型的操作符类的 OID。
rngcanonical	regproc	PG_PROC.proname	转换范围类型为规范格式的函数名，如果没有则为 0。
rngsubdiff	regproc	PG_PROC.proname	返回两个 double precision 元素值

名称	类型	引用	描述
			的不同的函数名，如果没有则为 0。

rngsubopc（如果元素类型是可排序的，则加上 rngcollation）决定用于范围类型的排序顺序。当元素类型是离散的时使用 rngcanonical。

5.3.83.PG_REPLICATION_ORIGIN

PG_REPLICATION_ORIGIN 系统表包含所有已创建的复制源，该表为全局共享表，即在每个节点上只有一份 pg_replication_origin，而不是每个数据库一份。

表 1 PG_REPLICATION_ORIGIN 字段

名称 类型 描述 roident oid 一个集群范围内唯一的复制源标识符。
roname text 外部的由用户定义的复制源名称。

5.3.84.PG_RESOURCE_POOL

PG_RESOURCE_POOL 系统表提供了数据库资源池的信息。

表 1 PG_RESOURCE_POOL 字段

名称	类型	描述
oid	oid	行标识符（隐藏列，必须显式指定列名选择）。
respool_name	name	资源池名称。
mem_percent	integer	内存配置的百分比。
cpu_affinity	bigint	CPU 绑定 core 的数值。
control_group	name	资源池所在的 control group 名称。
active_statements	integer	资源池上最大的并发数。
max_dop	integer	最大并发度。用作扩容的接口，表示数据重分布时，扫描并发度。
memory_limit	name	资源池最大的内存。
parentid	oid	父资源池 OID。
io_limits	integer	每秒触发 IO 的次数上限。

名称	类型	描述
		行存单位是万次/s，列存是次/s。
io_priority	name	IO 利用率高达 90%时，重消耗 IO 作业进行 IO 资源管控时关联的优先级等级。
nodegroup	name	表示资源池所在的逻辑 openGauss 的名称。
is_foreign	boolean	表示资源池是否用于逻辑 openGauss 之外的用户。如果为 true，表示资源池用来控制不属于当前资源池的普通用户的资源。
max_worker	integer	只用于扩容的接口，表示扩容数据重分布时，表内并发度。

注：max_dop 和 max_worker 用户扩容，不适用于 PanWeiDB。

5.3.85.PG_REWRITE

PG_REWRITE 系统表存储为表和视图定义的重写规则。

表 1 PG_REWRITE 字段

名称	类型	描述
oid	oid	行标识符（隐藏列，必须显式指定列名选择）。
rulename	name	规则名称。
ev_class	oid	使用这条规则的表名称。
ev_attr	smallint	这条规则适用的字段（目前总是为零，表示整个表）。
ev_type	“char”	规则适用的事件类型：1 = SELECT。2 = UPDATE。3 = INSERT。4 = DELETE。
ev_enabled	“char”	用于控制复制的触发。O = “origin” 和 “local” 模式时触发。D =禁用触发。R =

名称	类型	描述
		“replica”时触发。A = 任何模式是都会触发。
is_instead	boolean	如果该规则是 INSTEAD 规则，则为真。
ev_qual	pg_node_tree	规则的资格条件的表达式树（以 nodeToString()形式存在）。
ev_action	pg_node_tree	规则动作的查询树（以 nodeToString()形式存在）。

5.3.86.PG_RLSPOLICY

PG_RLSPOLICY 系统表存储行级访问控制策略。

表 1 PG_RLSPOLICY 字段

名称	类型	描述
oid	oid	行标识符（隐藏列，必须显式指定列名选择）。
polname	name	行级访问控制策略的名称。
polrelid	oid	行级访问控制策略作用的表对象 oid。
polcmd	“char”	行级访问控制策略影响的 SQL 操作。
polpermissive	boolean	行级访问控制策略的属性，t 为表达式 OR 条件拼接，f 为表达式 AND 条件拼接。
polroles	oid[]	行级访问控制策略影响的用户 oid 列表，不指定表示影响所有的用户。
polqual	pg_node_tree	行级访问控制策略的表达式。

5.3.87.PG_SECLABEL

PG_SECLABEL 系统表存储数据对象上的安全标签。

PG_SHSECLABEL（参考：系统表和系统视图->系统表->PG_SHSECLABEL 章节）的作用类似，只是它是用于在一个 PanWeiDB 内共享的数据库对象的安全标签上的。

表 1 PG_SECLABEL 字段

名称	类型	引用	描述

名称	类型	引用	描述
objoid	oid	任意 OID 属性	这个安全标签所属的对象的 OID。
classoid	oid	PG_CLASS.oid	出现这个对象的系统目录的 OID。
objsubid	integer	-	出现在这个对象中的列的序号。
provider	text	-	与这个标签相关的标签提供程序。
label	text	-	应用于这个对象的安全标签。

5.3.88.PG_SHDEPEND

PG_SHDEPEND 系统表记录数据库对象和共享对象（比如角色）之间的依赖性关系。这些信息允许 PanWeiDB 保证在企图删除这些对象之前，这些对象是没有被引用的。

PG_DEPEND(参考：系统表和系统视图->系统表->PG_DEPEND 章节）的作用类似，只是它是用于在一个数据库内部的对象的依赖性关系的。

和其他大多数系统表不同，PG_SHDEPEND 是在 PanWeiDB 里面所有的数据库之间共享的：每个 PanWeiDB 只有一个 PG_SHDEPEND，而不是每个数据库一个。

表 1 PG_SHDEPEND 字段

名称	类型	引用	描述
dbid	oid	PG_DATABASE.oid	依赖对象所在的数据库的 OID，如果是共享对象，则为零。
classid	oid	PG_CLASS.oid	依赖对象所在的系统表的 OID。
objid	oid	任意 OID 属性	指定的依赖对象的 OID。
objsubid	integer	-	对于一个表字段，这是字段号（objid 和 classid 参考表本身）。对于所有其他对象类型，这个字段为零。
refclassid	oid	PG_CLASS.oid	被引用对象所在的系统表的 OID（必须是一个共享表）。
refobjid	oid	任意 OID 属性	指定的被引用对象的 OID。
deptype	“char”	-	一段代码，定义了这个依赖性关系的特定语义；参阅下文。
objfile	text	-	用户定义函数库文件路径。

在任何情况下，一条 PG_SHDEPEND 记录就表明这个被引用的对象不能在未删除依赖对象的前提下删除。不过，deptype 同时还标出了几种不同的子风格：

- SHARED_DEPENDENCY_OWNER (o)
  被引用的对象（必须是一个角色）是依赖对象的所有者。
- SHARED_DEPENDENCY_ACL (a)
  被引用的对象（必须是一个角色）在依赖对象的 ACL（访问控制列表，也就是权限列表）里提到。SHARED_DEPENDENCY_ACL 不会在对象的所有者头上添加的，因为所有者会有一个 SHARED_DEPENDENCY_OWNER 记录。
- SHARED_DEPENDENCY_PIN (p)
  没有依赖对象；这类记录标识系统自身依赖于该被依赖对象，因此这样的对象绝对不能被删除。这种类型的记录只是由 initdb 创建。这样的依赖对象的字段都是零。
- SHARED_DEPENDENCY_DBPRIV(d)
  被引用的对象（必须是一个角色）具有依赖对象所对应的 ANY 权限（指定的依赖对象的 OID 对应的是系统表 gs_db_privilege 中一行）。

5.3.89.PG_SHDESCRIPTION

PG_SHDESCRIPTION 系统表为共享数据库对象存储可选的注释。可以使用 [COMMENT 命令（参考：SQL 语法参考->SQL 语法->COMMENT 章节）操作注释的内容，使用 psql 的\d 命令查看注释内容。

PG_DESCRIPTION 提供了类似的功能，它记录了单个数据库中对象的注释。

不同于大多数系统表，PG_SHDESCRIPTION 是在 PanWeiDB 里面所有的数据库之间共享的：每个 PanWeiDB 只有一个 PG_SHDESCRIPTION，而不是每个数据库一个。

表 1 PG_SHDESCRIPTION 字段

名称	类型	引用	描述
objoid	oid	任意 OID 属性	这条描述所描述的对象的 OID。
classoid	oid	PG_CLASS.oid	这个对象出现的系统表的 OID。

名称	类型	引用	描述
description	text	-	作为对该对象的描述的任意文本。

5.3.90.PG_SHSECLABEL

PG_SHSECLABEL 系统表存储在共享数据库对象上的安全标签。安全标签可以用 SECURITY LABEL 命令操作。

查看安全标签的简单点的方法，参考：系统表和系统视图->系统视图->PG_SECLABELS。

PG_SECLABEL 的作用类似，只是它是用于在单个数据库内部的对象的安全标签的。

不同于大多数的系统表，PG_SHSECLABEL 在 PanWeiDB 中的所有数据库中共享：每个 PanWeiDB 只有一个 PG_SHSECLABEL，而不是每个数据库一个。

表 1 PG_SHSECLABEL 字段

名称	类型	引用	描述
objoid	oid	任意 OID 属性	这个安全标签所属的对象的 OID。
classoid	oid	PG_CLASS.oid	出现这个对象的系统目录的 OID。
provider	text	-	与这个标签相关的标签提供程序。
label	text	-	应用于这个对象的安全标签。

5.3.91.PG_STATISTIC

PG_STATISTIC 系统表存储有关该数据库中表和索引列的统计数据。默认只有系统管理员权限才可以访问此系统表，普通用户需要授权才可以访问。

表 1 PG_STATISTIC 字段

名称	类型	描述
starelid	oid	所描述的字段所属的表或者索引。
starelkind	“char”	所属对象的类型。
staattnum	smallint	所描述的字段在表中的编号，从 1 开始。
stainherit	Boolean	是否统计有继承关系的对象。

名称	类型	描述
stanullfrac	real	该字段中为 NULL 的记录的比率。
stawidth	integer	非 NULL 记录的平均存储宽度，以字节计。
stadistinct	real	标识全局统计信息中数据库节点上字段里唯一的非 NULL 数据值的数目。一个大于零的数值是独立数值的实际数目。一个小于零的数值是表中行数的分数的负数（比如，一个字段的数值平均出现概率为两次，则可以表示为 stadistinct=-0.5）。零值表示独立数值的数目未知。
stakindN	smallint	一个编码，表示这种类型的统计存储在 pg_statistic 行的第 n 个“槽位”。n 的取值范围：1～5
staopN	oid	一个用于生成这些存储在第 n 个“槽位”的统计信息的操作符。比如，一个柱面图槽位会显示<操作符，该操作符定义了该数据的排序顺序。n 的取值范围：1～5
stanumbersN	real[]	第 n 个“槽位”的相关类型的数值类型统计，如果该槽位和数值类型没有关系，则就是 NULL。n 的取值范围：1～5
stavaluesN	anyarray	第 n 个“槽位”类型的字段数据值，如果该槽位类型不存储任何数据值，则就是 NULL。每个数组的元素值实际上都是指定字段的数据类型，因此，除了把这些字段的类型定义成 anyarray 之外，没有更好地办法。n 的取值范围：1～5
stadndistinct	real	标识 dn1 上字段里唯一的非 NULL 数据值的数目。一个大于零的数值是独立数值的实际数目。一个小于零的数值是表中行数的分数的负数（比如，一个字段的数值平均出现概率为两次，则可以表示为 stadistinct=-0.5）。零值表示独立数值的数目未知。
staextinfo	text	统计信息的扩展信息。预留字段。

须知

PG_STATISTIC 系统表存储了统计对象的一些敏感信息，如高频值 MCV。系统管理员和授权后的其他用户可以通过访问 PG_STATISTIC 系统表查询到统计对象的这些敏感信息。

5.3.92.PG_STATISTIC_EXT

PG_STATISTIC_EXT 系统表存储有关该数据库中表的扩展统计数据，包括多列统计数据和表达式统计数据（后续支持）。收集哪些扩展统计数据是由用户指定的。需要有系统管理员权限才可以访问此系统表。

表 1 PG_STATISTIC_EXT 字段

名称	类型	描述
starelid	oid	所描述的字段所属的表或者索引。
starelkind	“char”	所属对象的类型，'c'表示普通表，'p'表示分区表。
stainherit	Boolean	是否统计有继承关系的对象。
stanullfrac	real	该字段中为 NULL 的记录的比率。
stawidth	integer	非 NULL 记录的平均存储宽度，以字节计。
stadistinct	real	标识全局统计信息中数据库节点上字段里唯一的非 NULL 数据值的数目。一个大于零的数值是独立数值的实际数目。一个小于零的数值是表中行数的分数的负数（比如，一个字段的数值平均出现概率为两次，则可以表示为 stadistinct=-0.5）。零值表示独立数值的数目未知。
stadndistinct	real	标识 dn1 上字段里唯一的非 NULL 数据值的数目。一个大于零的数值是独立数值的实际数目。一个小于零的数值是表中行数的分数的负数（比如，一个字段的数值平均出现概率为两次，则可以表示为 stadistinct=-0.5）。零值表示独立数值的数目未知。
stakindN	smallint	一个编码，表示这种类型的统计存储在 pg_statistic 行的第 n 个“槽位”。n 的取值范围：1～5
staopN	oid	一个用于生成这些存储在第 n 个“槽位”的统计信息的操作符。比如，一个柱面图槽位会显示<操作符，该操作符定义了该数据的排序顺序。n 的取值范围：1～5
stakey	int2vector	所描述的字段编号的数组。
stanumbersN	real[]	第 n 个“槽位”的相关类型的数值类型统计，如果该槽位和数值类型没有关系，则就是 NULL。n 的取值范围：1～5
stavaluesN	anyarray	第 n 个“槽位”类型的字段数据值，如果该槽位类型

名称	类型	描述
		不存储任何数据值，则就是 NULL。每个数组的元素值实际上都是指定字段的数据类型，因此，除了把这些字段的类型定义成 anyarray 之外，没有更好地办法。n 的取值范围：1～5
staexprs	pg_node_tree	扩展统计信息对应的表达式。

须知

PG_STATISTIC_EXT 系统表存储了统计对象的一些敏感信息，如高频值 MCV。系统管理员和授权后的其他用户可以通过访问 PG_STATISTIC_EXT 系统表查询到统计对象的这些敏感信息。

5.3.93.PG_SUBSCRIPTION

PG_SUBSCRIPTION 系统表包含所有现有的逻辑复制订阅。

【须知】

- 访问此系统表用户需要具有系统管理员权限。
- 和大部分系统表不同，PG_SUBSCRIPTION 在数据库实例的所有数据库之间共享，即在每个节点上有只有一份 PG_SUBSCRIPTION，而不是每个数据库一份。

表 1 PG_SUBSCRIPTION 字段

名称	类型	描述
oid	oid	oid 行标识符（隐藏列，必须显式指定列名选择）。
subdbid	oid	订阅所在的数据库的 OID。
subname	name	订阅的名称。
subowner	oid	订阅的拥有者。
subenabled	boolean	如果为真，订阅被启用并且应该被复制。

名称	类型	描述
subconninfo	text	到发布端数据库的连接信息。
subslotname	name	发布端数据库中复制槽的名称，空表示为 NONE。
subsynccommit	text	订阅 worker 的 synchronous_commit 设置的值。
subpublications	text[]	被订阅的 publication 名称的数组。这些引用的是发布者服务器上的 publication。
subbinary	boolean	如果为真，发布端和订阅端之间传输数据是以二进制格式进行传输。
submatchddlowner	boolean	当订阅时指定选项 matchddlowner 为 false 时，该值为 false，默认为 true。 该选项指示订阅端在应用 DDL 操作时，是否切换到 DDL 日志信息中指定 owner 用户。
subskiplsn	text	commit_lsn 为该 lsn 的事务会被跳过，不被应用。

5.3.94.PG_SUBSCRIPTION_REL

系统表 PG_SUBSCRIPTION_REL 包含每个订阅中每个被复制表的状态，是多对多的映射关系。

该系统表仅包含运行 CREATE SUBSCRIPTION 或 ALTER SUBSCRIPTION ... REFRESH PUBLICATION 后对订阅已知的表。

名称	类型	描述
srsubid	oid	订阅标识符。
srrelid	oid	订阅关系标识符。
srsubstate	"char"	订阅状态。 ❖ i：初始化 ❖ d：基础数据正在复制

名称	类型	描述
		❖ f：基础数据复制完成 ❖ s：已与增量复制进度同步 ❖ r：准备好增量复制
srcsn	bigint	基础数据复制过程中快照 csn。
srsublsn	text	在 s 或 r 状态中，用于同步增量复制进度的远端 LSN，否则为空。

5.3.95.PG_SYNONYM

PG_SYNONYM 系统表存储同义词对象名与其他数据库对象名间的映射信息。

表 1 PG_SYNONYM 字段

名称	类型	描述
oid	oid	数据库对象 id。
synname	name	同义词名称。
synnamespace	oid	包含该同义词的名字空间的 OID。
synowner	oid	同义词的所有者，通常是创建它的用户 OID。
synobjschema	name	关联对象指定的模式名。
synobjname	name	关联对象名。

5.3.96.PG_TABLESPACE

PG_TABLESPACE 系统表存储表空间信息。

表 1 PG_TABLESPACE 字段

名称	类型	描述
oid	oid	行标识符（隐藏列，必须显式指定列名选择）。
spcname	name	表空间名称。
spcowner	oid	表空间的所有者，通常是创建它的人。
spcacl	aclitem[]	参考：SQL 语法参考->SQL 语法->GRANT、REVOKE 章节。
spcoptions	text[]	表空间的选项。
spcmaxsize	text	可使用的最大磁盘空间大小，单位 Byte。

名称	类型	描述
relative	boolean	标识表空间指定的存储路径是否为相对路径。
spcthreshold	spcthreshold	表空间阈值。

5.3.97.PG_TRIGGER

PG_TRIGGER 系统表存储触发器信息。

表 1 PG_TRIGGER 字段

名称	类型	描述
oid	oid	行标识符（隐藏列，必须显式指定列名选择）。
tgrelid	oid	触发器所在表的 OID。
tgname	name	触发器名。
tgfoid	oid	要被触发器调用的函数。
tgtype	smallint	触发器类型。
tgenabled	“char”	O =触发器在“origin”和“local”模式下触发。D = 触发器被禁用。R =触发器在“replica”模式下触发。A =触发器始终触发。
tgisinternal	boolean	内部触发器标识，如果为 true 表示内部触发器。
tgconstrrelid	oid	完整性约束引用的表。
tgconstrindid	oid	完整性约束的索引。
tgconstraint	oid	约束触发器在 pg_constraint 中的 OID。
tgdeferrable	boolean	约束触发器是为 DEFERRABLE 类型。
tginitdeferred	boolean	约束触发器是否为 INITIALLY DEFERRED 类型。
tgnargs	smallint	触发器函数入参个数。
tgattr	int2vector	当触发器指定列时的列号，未指定则为空数组。
tgargs	bytea	传递给触发器的参数。
tgqual	pg_node_tree	表示触发器的 WHEN 条件，如果没有则为 null。
tgowner	oid	触发器的所有者。

5.3.98.PG_TS_CONFIG

PG_TS_CONFIG 系统表包含表示文本搜索配置的记录。一个配置指定一个特定的文本搜索解析器和一个为了每个解析器的输出类型使用的字典的列表。

解析器在 PG_TS_CONFIG 记录中显示，但是字典映射的标记是由 PG_TS_CONFIG_MAP（参考：系统表和系统视图->系统表->PG_TS_CONFIG_MAP 章节）里面的辅助记录定义的。

表 1 PG_TS_CONFIG 字段

名称	类型	引用	描述
oid	oid	-	行标识符（隐藏列，必须显式指定列名选择）。
cfgname	name	-	文本搜索配置名。
cfgnamespace	oid	PG_NAMESPACE.oid	包含这个配置的名称空间的 OID。
cfgowner	oid	PG_AUTHID.oid	配置的所有者。
cfgparser	oid	PG_TS_PARSER.oid	这个配置的文本搜索解析器的 OID。
cfoptions	text[]	-	分词相关配置选项。

5.3.99.PG_TS_CONFIG_MAP

PG_TS_CONFIG_MAP 系统表包含为每个文本搜索配置的解析器的每个输出符号类型，显示哪个文本搜索字典应该被咨询、以什么顺序搜索的记录。

表 1 PG_TS_CONFIG_MAP 字段

名称	类型	引用	描述
mapcfg	oid	PG_TS_CONFIG.oid	拥有这个映射记录的 PG_TS_CONFIG 记录的 OID。
maptokentype	integer	-	由配置的解析器产生的一个符号类型值。
mapseqno	integer	-	在相同 mapcfg 或 maptokentype 值的情况下，该符号类型的顺序号。
mapdict	oid	PG_TS_DICT.oid	要咨询的文本搜索字典的 OID。

5.3.100.PG_TS_DICT

PG_TS_DICT 系统表包含定义文本搜索字典的记录。字典取决于文本搜索模板，该模板声明所有需要的实现函数；字典本身提供模板支持的用户可设置的参数的值。

这种分工允许字典通过非权限用户创建。参数由文本字符串 dictinitoption 指定，参数的格式和意义取决于模板。

表 1 PG_TS_DICT 字段

名称	类型	引用	描述
oid	oid	-	行标识符（隐藏列，必须显式指定列名选择）。
dictname	name	-	文本搜索字典名。
dictnamespace	oid	PG_NAMESPACE.oid	包含这个字典的名称空间的 OID。
dictowner	oid	PG_AUTHID.oid	字典的所有者。
dicttemplate	oid	PG_TS_TEMPLATE.oid	这个字典的文本搜索模板的 OID。
dictinitoption	text	-	该模板的初始化选项字符串。

5.3.101.PG_TS_PARSER

PG_TS_PARSER 系统表包含定义文本解析器的记录。解析器负责分裂输入文本为词位，并且为每个词位分配标记类型。新解析器必须由数据库系统管理员创建。

表 1 PG_TS_PARSER 字段

名称	类型	引用	描述
oid	oid	-	行标识符（隐藏列，必须显式指定列名选择）。
prsname	name	-	文本搜索解析器名。
prsnamespace	oid	PG_NAMESPACE.oid	包含这个解析器的名称空间的 OID。
prsstart	regproc	PG_PROC.proname	解析器的启动函数名。
prstoken	regproc	PG_PROC.proname	解析器的下一个标记函数名。
prsend	regproc	PG_PROC.proname	解析器的关闭函数名。

名称	类型	引用	描述
prsheadline	regproc	PG_PROC.proname	解析器的标题函数名。
prslextype	regproc	PG_PROC.proname	解析器的 lextype 函数名。

5.3.102.PG_TS_TEMPLATE

PG_TS_TEMPLATE 系统表包含定义文本搜索模板的记录。模板是文本搜索字典的类的实现框架。因为模板必须通过 C 语言级别的函数实现，索引新模板的创建必须由数据库系统管理员创建。

表 1 PG_TS_TEMPLATE 字段

名称	类型	引用	描述
oid	oid	-	行标识符（隐藏列，必须显式指定列名选择）。
tmplname	name	-	文本搜索模板名。
tmplnamespace	oid	PG_NAMESPACE.oid	包含这个模板的名称空间的 OID。
tmplinit	regproc	PG_PROC.proname	模板的初始化函数名。
tmpllexize	regproc	PG_PROC.proname	模板的 lexize 函数名。

5.3.103.PG_TYPE

PG_TYPE 系统表存储数据类型的相关信息。

表 1 PG_TYPE 字段

名称	类型	描述
oid	oid	行标识符（隐藏列，必须显式指定列名选择）。
typname	name	数据类型名称。
typnamespace	oid	包含这个类型的名称空间的 OID。
typowner	oid	该类型的所有者。
typlen	smallint	对于定长类型是该类型内部表现形式的字节数目。对于变长类型是负数。 ❖ -1 表示一种“变长”（有长度字属性的数据）。 ❖ -2 表示这是一个 NULL 结尾的 C 字符串。

名称	类型	描述
typbyval	boolean	指定内部传递这个类型的数值时是传值（该值为 true）还是传引用（该值为 false）。如果该类型的 TYPLEN 不是 1、2、4、8， TYPBYVAL 最好为 false。变长类型通常是传引用。即使 TYPLEN 允许传值，TYPBYVAL 也可以为 false。
typtype	“char”	❖ 对于抽象数据类型是 a。 ❖ 对于基础类型是 b。 ❖ 对于复合类型是 c（比如，一个表的行类型）。 ❖ 对于域类型是 d。 ❖ 对于伪类型是 p。 ❖ 对于范围类型是 r。 ❖ 对于枚举类型是 e。 ❖ 对于未定义类型是 u。 ❖ 对于集合类型是 o。 ❖ 对于可变数组类型是 y。 参见 typrelid 和 typbasetype。
typcategory	“char”	是数据类型的模糊分类，可用于解析器做为数据转换的依据。
typispreferred	boolean	如果为真，则数据符合 TYPCATEGORY 所指定的转换规则时进行转换。
typisdefined	boolean	如果定义了类型则为真，如果是一种尚未定义的类型的占位符则为假。如果为假，则除了该类型名称，名称空间和 OID 之外没有可靠的信息。
typdelim	“char”	当分析数组输入时，分隔两个此类型数值的字符请注意该分隔符是与数组元素数据类型相关联的，而不是和数组数据类型关联。
typrelid	oid	如果是复合类型（请参见 typtype），则这个字段指向 pg_class 中定义该表的行。对于自由存在的复合类型，pg_class 记录并不表示一个表，但是总需要它来查找该类型连接的 pg_attribute 记录。对于非复合类型为零。
typelem	oid	如果不为 0，则它标识 pg_type 里面的另外一行。当前类型可以当做一个产生类型为 typelem 的数组来描述。一个“真正的”数组类型是变长的（typlen= -

名称	类型	描述
		1），但是一些定长的（typlen > 0）类型也拥有非零的 typelem（比如 name 和 point）。如果一个定长类型拥有一个 typelem，则他的内部形式必须是 typelem 数据类型的某个数目的个数值，不能有其他数据。变长数组类型有一个该数组子过程定义的头（文件）。
typarray	oid	如果不为 0，则表示在 pg_type 中有对应的类型记录。
typinput	regproc	输入转换函数（文本格式）。
typoutput	regproc	输出转换函数（文本格式）。
typreceive	regproc	输入转换函数（二进制格式），如果没有则为 0。
typsend	regproc	输出转换函数（二进制格式），如果没有则为 0。
typmodin	regproc	输入类型修改符函数，如果为 0，则不支持。
typmodout	regproc	输出类型修改符函数，如果为 0，则不支持。
typanalyze	regproc	自定义的 ANALYZE 函数，如果使用标准函数，则为 0。
typalign	“char”	当存储此类型的数值时要求的对齐性质。它应用于磁盘存储以及该值在 PanWeiDB 内部的大多数形式。如果数值是连续存放的，比如在磁盘上以完全的裸数据的形式存放时，则先在此类型的数据前填充空白，这样它就可以按照要求的界限存储。对齐引用是该序列中第一个数据的开头。可能的值包含： ❖ c = char 对齐，也就是不需要对齐。 ❖ s = short 对齐（在大多数机器上是 2 字节）。 ❖ i = int 对齐（在大多数机器上是 4 字节）。 ❖ d = double 对齐（在大多数机器上是 8 字节，但不一定是全部）。 须知：对于在系统表里使用的类型，在 pg_type 里定义的尺寸和对齐必须和编译器在一个表示表的一行的结构里的布局一样。
typstorage	“char”	指明一个变长类型（那些有 typlen = -1）是否准备好应付非常规值，以及对这种属性的类型的缺省策略是什么。可能的值包含： ❖ p：数值总是以简单方式存储。

名称	类型	描述
		❖ e：数值可以存储在一个“次要”关系中（如果该关系有这么一个，请参见 pg_class.reltoastrelid）。 ❖ m：数值可以以内联的压缩方式存储。 ❖ x：数值可以以内联的压缩方式或者在“次要”表里存储。 须知：m 域也可以移到从属表里存储，但只是最后的解决方法（e 和 x 域先移走）。
typenotnull	boolean	该类型是否存在 NOTNULL 约束。目前只用于域。
typbasetype	oid	如果这是一个衍生类型（请参见 typtype），则该标识作为这个类型的基础的类型。如果不是衍生类型则为零。
typtypmod	integer	域使用 typtypmod 记录要作用到它们的基础类型上的 typmod（如果基础类型不使用 typmod 则为-1）。如果这种类型不是域，则为-1。
typndims	integer	如果一个域是数组，则 typndims 是数组维数的数值（也就是说，typbasetype 是一个数组类型；域的 typelem 将匹配基本类型的 typelem）。非域非数组域为零。
typcollation	oid	指定类型的排序规则。取值参考：系统表和系统视图->系统表->PG_COLLATION 章节。如果为 0，则表示不支持排序。
typdefaultbin	pg_node_tree	如果为非 NULL，则它是该类型缺省表达式的 nodeToString()表现形式。目前这个字段只用于域。
typdefault	text	如果某类型没有相关缺省值，则取值是 NULL。 ❖ 如果 typdefaultbin 为非 NULL，则 typdefault 必须包含一个 typdefaultbin 代表的缺省表达式。 ❖ 如果 typdefaultbin 为 NULL 但 typdefault 不是，typdefault 则是该类型缺省值的外部表现形式，可以把它作为该类型的输入，转换器生成一个常量。
typacl	aclitem[]	访问权限。

5.3.104.PG_USER_MAPPING

PG_USER_MAPPING 系统表存储从本地用户到远程的映射。

需要有系统管理员权限才可以访问此系统表。普通用户可以使用视图 PG_USER_MAPPINGS（参考：系统表和系统视图->系统视图->PG_USER_MAPPINGS 章节）进行查询。

表 1 PG_USER_MAPPING 字段

名称	类型	引用	描述
oid	oid	-	行标识符（隐藏列，必须显式指定列名选择）。
umuser	oid	PG_AUTHID.oid	被映射的本地用户的 OID，如果用户映射是公共的则为 0。
umserver	oid	PG_FOREIGN_SERVER.oid	包含这个映射的外部服务器的 OID。
umoptions	text[]	-	用户映射指定选项，使用“keyword=value”格式的字符串。

5.3.105.PG_USER_STATUS

PG_USER_STATUS 系统表提供了访问数据库用户的状态。需要有系统管理员权限才可以访问此系统表。

表 1 PG_USER_STATUS 字段

名称	类型	描述
oid	oid	行标识符（隐含字段，必须明确选择）。
roloid	oid	角色的标识。
failcount	integer	尝试失败次数。
locktime	timestamp with time zone	角色被锁定的时间点。
rolstatus	smallint	角色的状态。 0：正常状态。 1：由于登录失败次数超过阈值被锁定了

名称	类型	描述
		一定的时间。 2：被管理员锁定。
permspace	bigint	角色已经使用的永久表存储空间大小。
tempspace	bigint	角色已经使用的临时表存储空间大小。
passwordexpired	smallint	密码是否失效。 0：密码有效。 1：密码失效。

5.3.106.PG_WORKLOAD_GROUP

PG_WORKLOAD_GROUP 系统表提供了数据库负载组的信息。

表 1 PG_WORKLOAD_GROUP 字段

名称	类型	描述
oid	oid	行标识符（隐藏列，必须显式指定列名选择）。
workload_gpname	name	负载组名称。
respool_oid	oid	绑定到的资源池的 id。
act_statements	integer	负载组内最大的活跃语句数。

5.3.107.PGXC_CLASS

PGXC_CLASS 系统表存储每张表的复制或分布信息。PGXC_CLASS 系统表仅在分布式场景下有具体含义，PanWeiDB 只能查询表定义。

表 1 PGXC_CLASS 字段

名称	类型	描述
pcrelid	oid	表的 OID。
pclocatortype	“char”	定位器类型。H：hashG：RangeL：ListM：ModuloN：Round RobinR：Replication
pchashalgorithm	smallint	使用哈希算法分布元组。
pchashbuckets	smallint	哈希容器的值。
pgroup	name	节点群的名称。
redistributed	“char”	表已经完成重分布。

名称	类型	描述
redis_order	integer	重分布的顺序。该值等于 0 的表在本轮重分布过程中不进行重分布。
pcattnum	int2vector	用作分布键的列标号。
nodeoids	oidvector_extend	表分布的节点 OID 列表。
options	text	系统内部保留字段，存储扩展状态信息。

5.3.108.PGXC_GROUP

PGXC_GROUP 系统表存储节点组信息。PGXC_GROUP 系统表仅在分布式场景下有具体含义，PanWeiDB 只能查询表定义。

表 1 PGXC_GROUP 字段

名称	类型	描述
oid	oid	行标识符（隐含字段，必须明确选择）。
group_name	name	节点组名称。
in_redistribution	“char”	是否需要重分布。取值包括 n，y，t。n：表示 NodeGroup 没有再进行重分布。y：表示 NodeGroup 是重分布过程中的源节点组。t：表示 NodeGroup 是重分布过程中的目的节点组。
group_members	oidvector_extend	节点组的节点 OID 列表。
group_buckets	text	分布数据桶的集合。
is_installation	Boolean	是否安装子集群。t（true）：表示安装。f（false）：表示不安装。
group_acl	aclitem[]	访问权限。
group_kind	“char”	node group 类型，取值包括 i, n, v, e。i：表示 installation node group。n：表示普通非逻辑集群 node group。v：表示逻辑集群 node group。e：表示弹性集群。
group_parent	oid	如果是子 node group，该字段表示父 node group 的 OID，如果是父 node group，该字段值为空。

5.3.109.PGXC_NODE

PGXC_NODE 系统表存储集群节点信息。PGXC_NODE 系统表仅在分布式场景下有具体含义，PanWeiDB 只能查询表定义。

表 1 PGXC_NODE 字段

名称	类型	描述
oid	oid	行标识符（隐含字段，必须明确选择）。
node_name	name	节点名称。
node_type	“char”	节点类型。 C：协调节点。 D：数据节点。 S：数据节点的备节点。
node_port	integer	节点的端口号。
node_host	name	节点的主机名称或者 IP（如配置为虚拟 IP，则为虚拟 IP）。
node_port1	integer	复制节点的端口号。
node_host1	name	复制节点的主机名称或者 IP（如配置为虚拟 IP，则为虚拟 IP）。
hostis_primary	Boolean	表明当前节点是否发生主备切换。 t（true）：表示发生。 f（false）：表示不发生。
nodeis_primary	Boolean	在 replication 表下，是否优选当前节点作为优先执行的节点进行非查询操作。 t（true）：表示优选。 f（false）：表示不优选。
nodeis_preferred	Boolean	在 replication 表下，是否优选当前节点作为首选的节点进行查询。 t（true）：表示优选。 f（false）：表示不优选。
node_id	integer	节点标志符。由 node_name 经过 hash 函数计算后得到。
sctp_port	integer	主节点使用 TCP 代理通信库或 SCTP 通信库的数据通道侦听端口。
control_port	integer	主节点使用 TCP 代理通信库或 SCTP 通信库的控制

名称	类型	描述
		通道侦听端口。
sctp_port1	integer	备节点使用 TCP 代理通信库或 SCTP 通信库的数据通道侦听端口。
control_port1	integer	备节点使用 TCP 代理通信库或 SCTP 通信库的控制通道侦听端口。
nodeis_central	Boolean	表明当前节点是否为中心控制节点，只用于 CN，对 DN 无效。 t（true）：表示是。 f（false）：表示不是。
nodeis_active	Boolean	表明当前节点是否是正常状态，用于标记 CN 是否被剔除，对 DN 无效。 t（true）：表示是。 f（false）：表示不是。

5.3.110.PGXC_SLICE

PGXC_SLICE 表是针对 range 范围分布和 list 分布创建的系统表，用来记录分布具体信息，当前不支持 range interval 自动扩展分片，不过在系统表中预留。

PGXC_SLICE 系统表仅在分布式场景下有具体含义，PanWeiDB 只能查询表定义。

表 1 PGXC_SLICE 字段

名称	类型	描述
relname	name	表名或者分片名，通过 type 区分
type	char	当为’t’时 relname 是表名，当为’s’时 relname 是分片的名字
strategy	char	‘r’：为 range 分布表‘l’：为 list 分布表后续 interval 分片会扩展该值
relid	oid	该 tuple 隶属的分布表 oid
referenceoid	oid	所参考分布表的 oid,用于 slice reference 建表语法
sindex	int	当为 list 分布表时，表示当前 boundary 在某个分片内的位置

名称	类型	描述
interval	text[]	预留字段
transitboundary	text[]	预留字段
transitno	int	预留字段
nodeoid	oid	当 relname 为分片名时，表示该分片的数据存放在哪个 DN 上，nodeoid 表示这个 DN 的 oid
boundaries	text[]	当 relname 为分片名时，对应该分片的边界值
specified	boolean	当前分片对应的 DN 是否是用户在 DDL 中显示指定的
sliceorder	int	用户定义分片的顺序

5.3.111.PW_DEPENDENCIES

该系统表主要记录依赖对象与引用对象的依赖关系。

表 1 PW_DEPENDENCIES 字段

名称	类型	描述
schemaname	name	依赖对象所在的 schema。
objectname	name	依赖对象的名称。
objecttype	text	依赖对象的类型。
refname	name	引用对象的名称。
refobjid	oid	引用对象的 oid。
refschemaname	name	引用对象的 schema。
refpackagename	name	引用对象所属的 package。
refobjectname	name	引用对象名。
refobjectsubname	name	引用对象的类型属性名。
refobjecttype	text	引用对象的类型（表、视图、存储过程、函数）。
dependedtype	text	引用对象与依赖对象的依赖关系，可取值包括： ❖ weak：代表弱依赖关系。 ❖ hard：代表强依赖关系。

5.3.112.PLAN_TABLE_DATA

PLAN_TABLE_DATA 存储了用户通过执行 EXPLAIN PLAN 收集到的计划信息。与 PLAN_TABLE 视图（参考：系统表和系统视图->系统视图-

>PLAN_TABLE 章节）不同的是 PLAN_TABLE_DATA 表存储了所有 session 和 user 执行 EXPLAIN PLAN 收集的计划信息。

表 1 PLAN_TABLE_DATA 字段

名称	类型	描述
session_id	text	表示插入该条数据的会话，由服务线程启动时间戳和服务线程 ID 组成。受非空约束限制。
user_id	oid	用户 ID，用于标识触发插入该条数据的用户。受非空约束限制。
statement_id	varchar2(30)	用户输入的查询标签。
plan_id	bigint	查询标识。该标识在计划生成阶段自动产生，供内核工程师调试使用。
id	int	计划中的节点编号。
operation	varchar2(30)	操作描述。
options	varchar2(255)	操作选项。
object_name	name	操作对应的对象名，来自于用户定义。
object_type	varchar2(30)	对象类型。
object_owner	name	对象所属 schema，来自于用户定义。
projection	varchar2(4000)	操作输出的列信息。

说明

- PLAN_TABLE_DATA 中包含了当前节点所有用户、所有会话的数据，仅管理员有访问权限。普通用户可以通过 PLAN_TABLE 视图（参考：系统表和系统视图->系统视图->PLAN_TABLE 章节）查看属于自己的数据。
- PLAN_TABLE_DATA 中的数据是用户通过执行 EXPLAIN PLAN 命令后由系统自动插入表中，因此禁止用户手动对数据进行插入或更新，否则会引起表中的数据混乱。需要对表中数据删除时，建议通过 PLAN_TABLE 视图（参考：系统表和系统视图->系统视图->PLAN_TABLE 章节）。

- statement_id、object_name、object_owner 和 projection 字段内容遵循用户定义的大小写存在系统库中查询到结果，用户库中无法查询。
- 对于此系统表查询有如下约束：
- 必须在 postgres 库内查询，其他库中不存数据。
- 此系统表受 track_stmt_stat_level 控制，默认为“OFF,L0”，第一部分控制 Full SQL，第二部分控制 Slow SQL，具体字段记录级别见下表。
- 对于 Slow SQL，当 track_stmt_stat_level 的值为非 OFF 时，且 SQL 执行时间超过 log_min_duration_statement，会记录为慢 SQL。
- 表 1 STATEMENT_HISTORY 字段储，其他字段内容采用大写存储。

5.3.113.STATEMENT_HISTORY

获得当前节点的执行语句的信息。查询系统表必须具有 sysadmin 权限。只可在系统库中查询到结果，用户库中无法查询。

对于此系统表查询有如下约束：

- 必须在 postgres 库内查询，其他库中不存数据。
- 此系统表受 track_stmt_stat_level 控制，默认为“OFF,L0”，第一部分控制 Full SQL，第二部分控制 Slow SQL，具体字段记录级别见下表。考虑性能影响，更改该参数的值时建议通过 set 方式设置，使该参数仅对当前会话生效。
- 对于 Slow SQL，当 track_stmt_stat_level 的值为非 OFF 时，且 SQL 执行时间超过 log_min_duration_statement，会记录为慢 SQL。
- 对于存储过程内的 SQL 语句(parent_unique_sql_id 非 0)，n_soft_parse、n_hard_parse、n_returned_rows 需要设置参数 instr_unique_sql_track_type 为 all 才能记录，n_tuples_fetched、n_tuples_returned、n_tuples_inserted、n_tuples_updated、n_tuples_deleted、n_blocks_fetched、n_blocks_hit、db_time、

pu_time、execution_time、parse_time、plan_time、rewrite_time、pl_execution_time、pl_compilation_time、data_io_time、net_send_info、net_recv_info、net_stream_send_info、net_stream_recv_info、lock_wait_count、lock_wait_time、lwlock_count、lwlock_wait_count、lwlock_ti、lwlock_wait_time、details、trace_id、advise 列不支持，记录的值没有实际意义。

名称	类型	描述	记录级别
db_name	name	数据库名称。	L0
schema_name	name	schema 名称。	L0
origin_node	integer	节点名称。	L0
user_name	name	用户名。	L0
application_name	text	用户发起的请求的应用程序名称。	L0
client_addr	text	用户发起的请求的客户端地址。	L0
client_port	integer	用户发起的请求的客户端端口。	L0
unique_query_id	bigint	归一化 SQL ID。	L0
debug_query_id	bigint	唯一 SQL ID。	L0
query	text	归一化 SQL。	L0
start_time	timestamp with time zone	语句启动的时间。	L0
finish_time	timestamp with time zone	语句结束的时间。	L0
slow_sql_threshold	bigint	语句执行时慢 SQL 的标准。	L0
transaction_id	bigint	事务 ID。	L0
thread_id	bigint	执行线程 ID。	L0
session_id	bigint	用户 session id。	L0
n_soft_parse	bigint	软解析次数，n_soft_parse + n_hard_parse 可能大于 n_calls，因为子查询未计入 n_calls。	L0
n_hard_parse	bigint	硬解析次数，n_soft_parse + n_hard_parse 可能大于 n_calls，因为子查询未计入	L0

名称	类型	描述	记录级别
		n_calls。	
query_plan	text	语句执行计划。	L1
n_returned_rows	bigint	SELECT 返回的结果集行数。	L0
n_tuples_fetched	bigint	随机扫描行。	L0
n_tuples_returned	bigint	顺序扫描行。	L0
n_tuples_inserted	bigint	插入行。	L0
n_tuples_updated	bigint	更新行。	L0
n_tuples_deleted	bigint	删除行。	L0
n_blocks_fetched	bigint	buffer 的块访问次数。	L0
n_blocks_hit	bigint	buffer 的块命中次数。	L0
db_time	bigint	有效的 DB 时间花费，多线程将累加（单位：微秒）。	L0
cpu_time	bigint	CPU 时间（单位：微秒）。	L0
execution_time	bigint	执行器内执行时间（单位：微秒）。	L0
parse_time	bigint	SQL 解析时间（单位：微秒）。	L0
plan_time	bigint	SQL 生成计划时间（单位：微秒）。	L0
rewrite_time	bigint	SQL 重写时间（单位：微秒）。	L0
pl_execution_time	bigint	plpgsql 上的执行时间（单位：微秒）。	L0
pl_compilation_time	bigint	plpgsql 上的编译时间（单位：微秒）。	L0
data_io_time	bigint	IO 上的时间花费（单位：微秒）。	L0
net_send_info	text	通过物理连接发送消息的网络状态，包含时间（微秒）、调用次数、吞吐量（字节）。通过该字段可以分析 SQL 在分布式系统下的网络开销，单机模式下不支持该字段。例如：{“time” :xxx, “n_calls” :xxx, “size” :xxx}。	L0
net_recv_info	text	通过物理连接接收消息的网络状态，包含时间（微秒）、调用次	L0

名称	类型	描述	记录级别
		数、吞吐量（字节）。通过该字段可以分析 SQL 在分布式系统下的网络开销，单机模式下不支持该字段。例如：{“time” :xxx, “n_calls” :xxx, “size” :xxx}。	
net_stream_send_info	text	通过逻辑连接发送消息的网络状态，包含时间（微秒）、调用次数、吞吐量（字节）。通过该字段可以分析 SQL 在分布式系统下的网络开销，单机模式下不支持该字段。例如：{“time” :xxx, “n_calls” :xxx, “size” :xxx}。	L0
net_stream_recv_info	text	通过逻辑连接接收消息的网络状态，包含时间（微秒）、调用次数、吞吐量（字节）。通过该字段可以分析 SQL 在分布式系统下的网络开销，单机模式下不支持该字段。例如：{“time” :xxx, “n_calls” :xxx, “size” :xxx}。	L0
lock_count	bigint	加锁次数。	L0
lock_time	bigint	加锁耗时。	L1
lock_wait_count	bigint	加锁等待次数。	L0
lock_wait_time	bigint	加锁等待耗时。	L1
lock_max_count	bigint	最大持锁数量。	L0
lwlock_count	bigint	轻量级加锁次数（预留）。	L0
lwlock_wait_count	bigint	轻量级等锁次数。	L0
lwlock_time	bigint	轻量级加锁时间（预留）。	L1
lwlock_wait_time	bigint	轻量级等锁时间。	L1
details	bytea	语句锁事件的列表，该列表按时间书序记录事件，记录的数量受参数 track_stmt_details_size 的影响。该字段为二进制，需要借助解析函数 pg_catalog.statement_detail_de	L2

名称	类型	描述	记录级别
		code 读取。 事件包括：加锁开始、加锁结束、等锁开始、等锁结束、放锁开始、放锁结束、轻量级等锁开始、轻量级等锁结束	
is_slow_sql	boolean	该 SQL 是否为 slow SQL。 ❖ t（true）：表示是。 ❖ f（false）：表示不是。	L0
trace_id	text	驱动传入的 trace id，与应用的一次请求相关联。	L0

5.4.系统视图

5.4.1.GS_AUDITING

GS_AUDITING 视图显示对数据库相关操作的所有审计信息。需要有系统管理员或安全策略管理员权限才可以访问此视图。

表 1 GS_AUDITING 字段

名称	类型	描述
polname	name	策略名称，需要唯一，不可重复。
pol_type	text	审计策略类型，值为 access 或者 privilege。 access：表示审计 DML 操作。 privilege：表示审计 DDL 操作。
polenabled	boolean	用来表示策略启动开关。 t（true）：表示启动。 f（false）：表示不启动。
access_type	bigint	DML 数据库操作相关类型。例如 SELECT、INSERT、DELETE 等。
label_name	name	资源标签名称。对应系统表 gs_auditing_policy 中的 polname 字段。
priv_object	name	用来描述数据库资产的路径。
filter_name	text	过滤条件的逻辑字符串。

5.4.2.GS_AUDITING_ACCESS

GS_AUDITING_ACCESS 视图显示对数据库 DML 相关操作的所有审计信息。需要有系统管理员或安全策略管理员权限才可以访问此视图。

表 1 GS_AUDITING_ACCESS 字段

名称	类型	描述
polname	name	策略名称，需要唯一，不可重复。
pol_type	text	审计策略类型，值为‘access’，表示审计 DML 操作。
polenabled	boolean	用来表示策略启动开关。
access_type	name	DML 数据库操作相关类型。例如 SELECT、INSERT、DELETE 等。
label_name	name	资源标签名称。对应系统表 gs_auditing_policy 中的 polname 字段。
access_object	text	用来描述数据库资产的路径。
filter_name	text	过滤条件的逻辑字符串。

5.4.3.GS_AUDITING_PRIVILEGE

GS_AUDITING_PRIVILEGE 视图显示对数据库 DDL 相关操作的所有审计信息。需要有系统管理员或安全策略管理员权限才可以访问此视图。

表 1 GS_AUDITING_PRIVILEGE 字段

名称	类型	描述
polname	name	策略名称，需要唯一，不可重复。
pol_type	text	审计策略类型，值为‘privilege’，表示审计 DDL 操作。
polenabled	boolean	用来表示策略启动开关。
access_type	name	DDL 数据库操作相关类型。例如 CREATE、ALTER、DROP 等。
label_name	name	资源标签名称。对应系统表 gs_auditing_policy 中的 polname 字段。
priv_object	text	带有数据库对象的全称域名。
filter_name	text	过滤条件的逻辑字符串。

5.4.4.GS_DB_PRIVILEGES

GS_DB_PRIVILEGES 系统视图记录 ANY 权限的授予情况，每条记录对应一条授权信息。

表 1 GS_DB_PRIVILEGES 字段

名称	类型	描述
rolename	name	用户名。
privilege_type	text	用户拥有的 ANY 权限，取值请参考：SQL 语法参考->SQL 语法->GRANT 章节。
admin_option	boolean	是否具有 privilege_type 列记录的 ANY 权限的再授权权限。 yes：表示具有。 no：表示不具有。

5.4.5.GS_FILE_STAT

GS_FILE_STAT 视图通过对数据文件 IO 的统计，反映数据的 IO 性能，用以发现 IO 操作异常等性能问题。

表 1 GS_FILE_STAT 字段

名称	类型	描述
filenum	oid	文件标识。
dbid	oid	数据库标识。
spcid	oid	表空间标识。
phyrds	bigint	读物理文件的数目。
phywrts	bigint	写物理文件的数目。
phyblkrd	bigint	读物理文件块的数目。
phyblkwrt	bigint	写物理文件块的数目。
readtim	bigint	读文件的总时长，单位微秒。

名称	类型	描述
writetim	bigint	写文件的总时长，单位微秒。
avgiotim	bigint	读写文件的平均时长，单位微秒。
lstiotim	bigint	最后一次读文件时长，单位微秒。
miniotim	bigint	读写文件的最小时长，单位微秒。
maxiowtm	bigint	读写文件的最大时长，单位微秒。

5.4.6.GS_GSC_MEMORY_DETAIL

GS_GSC_MEMORY_DETAIL 视图描述当前节点当前进程的全局 SysCache 的内存占用情况，仅在开启 GSC 的模式下有数据。需要注意的是，这个查询由于是以数据库内存上下文分隔的，因此会缺少一部分内存的统计，缺失的内存统计对应的内存上下文名称为 GlobalSysDBCache。

表 1 GS_GSC_MEMORY_DETAIL 字段

名称	类型	描述
db_id	text	数据库 id。
totalsize	numeric	共享内存总大小，单位 Byte。
freesize	numeric	共享内存剩余大小，单位 Byte。
usedsize	numeric	共享内存使用大小，单位 Byte。

5.4.7.GS_INSTANCE_TIME

提供当前集节点下的各种时间消耗信息，主要分为以下类型：

- DB_TIME：作业在多核下的有效时间花销 F。
- CPU_TIME：CPU 的时间花销。
- EXECUTION_TIME：执行器内的时间花销。
- PARSE_TIME：SQL 解析的时间花销。
- PLAN_TIME：生成 Plan 的时间花销。
- REWRITE_TIME：SQL 重写的时间花销。
- PL_EXECUTION_TIME：plpgsql（存储过程）执行的时间花销。

- PL_COMPILATION_TIME：plpgsql（存储过程）编译的时间花销。
- NET_SEND_TIME：网络上的时间花销。
- DATA_IO_TIME：IO 的时间花销。

表 1 GS_INSTANCE_TIME 字段

名称	类型	描述
stat_id	integer	统计编号。
stat_name	text	类型名称。
value	bigint	时间值（单位：微秒）。

5.4.8.GS_LABELS

GS_LABELS 视图显示所有已配置的资源标签信息。需要有系统管理员或安全策略管理员权限才可以访问此视图。

名称	类型	描述
labelname	name	资源标签的名称。
labeltype	name	资源标签的类型。对应系统表[GS_POLICY_LABEL（参考：系统表和系统视图->系统表->GS_POLICY_LABEL 章节）中的 labeltype 字段。
fqdntype	name	数据库资源的类型。如 table、schema、index 等。
schemaname	name	数据库资源所属的 schema 名称。
fqdnname	name	数据库资源名称。
columnname	name	数据库资源列名称，若标记的数据库资源不为表的列则该项为空。

5.4.9.GS_LSC_MEMORY_DETAIL

GS_LSC_MEMORY_DETAIL 视图统计所有的线程的本地 SysCache 内存使用情况，以 MemoryContext 节点来统计，仅在开启 GSC 的模式下有数据。

表 1 GS_LSC_MEMORY_DETAIL 字段

名称	类型	描述
threadid	text	线程启动时间+线程标识（字符串信息为

名称	类型	描述
		timestamp.sessionid）。
tid	bigint	线程标识。
thrdtype	text	线程类型。可以是系统内存在的任何线程类型，如postgresql、wlmmonitor 等。
contextname	text	内存上下文名称。
level	smallint	当前上下文在整体内存上下文中的层级。
parent	text	父内存上下文名称。
totalsize	bigint	当前内存上下文的内存总数，单位 Byte。
freesize	bigint	当前内存上下文中已释放的内存总数，单位 Byte。
usedsize	bigint	当前内存上下文中已使用的内存总数，单位 Byte。

5.4.10.GS_MASKING

GS_MASKING 视图显示所有已配置的动态脱敏策略信息。需要有系统管理员或安全策略管理员权限才可以访问此视图。

名称	类型	描述
polname	name	脱敏策略名称。
polenabled	boolean	脱敏策略开关。
maskaction	name	脱敏函数。
labelname	name	脱敏函数作用的标签名称。
masking_object	text	脱敏数据库资源对象。
filter_name	text	过滤条件的逻辑表达式。

5.4.11.GS_MATVIEWS

GS_MATVIEWS 视图提供了关于数据库中每一个物化视图的信息。

表 1 GS_MATVIEWS 字段

名称	类型	引用	描述
schemaname	name	PG_NAMESPACE.nspname	物化视图的模式名。
matviewname	name	PG_CLASS.relname	物化视图名。
matviewowner	name	PG_AUTHID.Erolname	物化视图的所有者。

名称	类型	引用	描述
tablespace	name	PG_TABLESPACE.spcname	物化视图的表空间名（如果使用数据库默认表空间则为空）。
cmindexes	boolean	-	如果物化视图有（或者最近有过）任何索引，则此列为真。
definition	text	-	物化视图的定义（一个重构的 SELECT 查询）。

5.4.12.GS_OS_RUN_INFO

GS_OS_RUN_INFO 视图显示当前操作系统运行的状态信息。

表 1 GS_OS_RUN_INFO 字段

名称	类型	描述
id	integer	编号。
name	text	操作系统运行状态名称。
value	numeric	操作系统运行状态值。
comments	text	操作系统运行状态注释。
cumulative	Boolean	操作系统运行状态的值是否为累加值。

5.4.13.GS_REDO_STAT

GS_REDO_STAT 视图用于统计会话线程日志回放情况。

表 1 GS_REDO_STAT 字段

名称	类型	描述
phywrts	bigint	日志回放过程中写数据的次数。
phyblkwrt	bigint	日志回放过程中写数据的块数。
writetim	bigint	日志回放过程中写数据所耗的总时间。
avgiotim	bigint	日志回放过程中写一次数据的平均消耗时间。
lstiotim	bigint	日志回放过程中最后一次写数据消耗的时间。
miniotim	bigint	日志回放过程中单次写数据消耗的最短时间。
maxiowtm	bigint	日志回放过程中单次写数据消耗的最长时间。

5.4.14.GS_SESSION_CPU_STATISTICS

GS_SESSION_CPU_STATISTICS 视图显示当前用户执行的正在运行的复杂作业的 CPU 使用的负载管理信息。

表 1 GS_SESSION_CPU_STATISTICS 字段

名称	类型	描述
datid	oid	连接后端的数据库 OID。
usename	name	登录到该后端的用户名。
pid	bigint	后端线程 ID。
start_time	timestamp with time zone	语句执行的开始时间。
min_cpu_time	bigint	语句在数据库节点上的最小 CPU 时间，单位为 ms。
max_cpu_time	bigint	语句在数据库节点上的最大 CPU 时间，单位为 ms。
total_cpu_time	bigint	语句在数据库节点上的 CPU 总时间，单位为 ms。
query	text	正在执行的语句。
node_group	text	该字段不支持。
top_cpu_dn	text	cpu 使用量信息。

5.4.15.GS_SESSION_MEMORY

GS_SESSION_MEMORY 视图统计 Session 级别的内存使用情况，包含执行作业在数据节点上 gaussdb 线程和 Stream 线程分配的所有内存。当 GUC 参数 enable_memory_limit 的值为 off 时，本视图不可用。

表 1 GS_SESSION_MEMORY 字段

名称	类型	描述
sessid	text	线程启动时间+线程标识。
init_mem	integer	当前正在执行作业进入执行器前已分配的内存，单位 MB。
used_mem	integer	当前正在执行作业已分配的内存，单位 MB。
peak_mem	integer	当前正在执行作业已分配的内存峰值，单位 MB。

5.4.16.GS_SESSION_MEMORY_CONTEXT

GS_SESSION_MEMORY_CONTEXT 视图统计所有的会话的内存使用情况，以 MemoryContext 节点来统计。该视图仅在开启线程池（enable_thread_pool = on）时生效。

其中内存上下文“TempSmallContextGroup”，记录当前线程中所有内存上下文字段“totalsize”小于 8192 字节的信息汇总，并且内存上下文统计计数记录到“usedsize”字段中。所以在视图中，“TempSmallContextGroup”内存上下文中的“totalsize”和“freesize”是该线程中所有内存上下文“totalsize”小于 8192 字节的汇总总和，usedsize 字段表示统计的内存上下文个数。

表 1 GS_SESSION_MEMORY_CONTEXT 字段

名称	类型	描述
sessid	text	会话启动时间+会话标识（字符串信息为 timestamp.sessionid）。
threadid	bigint	会话绑定的线程标识，如果未绑定线程，该值为-1。
contextname	text	内存上下文名称。
level	smallint	当前上下文在整体内存上下文中的层级。
parent	text	父内存上下文名称。
totalsize	bigint	当前内存上下文的内存总数，单位 Byte。
freesize	bigint	当前内存上下文中已释放的内存总数，单位 Byte。
usedsize	bigint	当前内存上下文中已使用的内存总数，单位 Byte；“TempSmallContextGroup”内存上下文中该字段含义为统计计数。

5.4.17.GS_SESSION_MEMORY_DETAIL

GS_SESSION_MEMORY_DETAIL 统计会话的内存使用情况，以 MemoryContext 节点来统计。当开启线程池（enable_thread_pool = on）时，该视图包含所有的线程和会话的内存使用情况。当 GUC 参数 enable_memory_limit 的值为 off 时，本视图不可用。

其中内存上下文“TempSmallContextGroup”，记录当前线程中所有内存上下文字段“totalsize”小于 8192 字节的信息汇总，并且内存上下文统计计数记录到“usedsize”字段中。所以在视图中，“TempSmallContextGroup”内存上下文中的“totalsize”和“freesize”是该线程中所有内存上下文“totalsize”小于 8192 字节的汇总总和，usedsize 字段表示统计的内存上下文个数。

可通过“select * from gs_session_memctx_detail(threadid, '');”将某个线程所有内存上下文信息记录到“$GAUSSLOG/pg_log/${node_name}/dumpmem”目录下的“threadid_timestamp.log”文件中。其中 threadid 可通过下表 sessid 中获得。

表 1 GS_SESSION_MEMORY_DETAIL 字段

名称	类型	描述
sessid	text	线程启动时间+线程标识（字符串信息为 timestamp.threadid）。
sesstype	text	线程名称。
contextname	text	内存上下文名称。
level	smallint	当前上下文在整体内存上下文中的层级。
parent	text	父内存上下文名称。
totalsize	bigint	当前内存上下文的内存总数，单位 Byte。
freesize	bigint	当前内存上下文中已释放的内存总数，单位 Byte。
usedsize	bigint	当前内存上下文中已使用的内存总数，单位 Byte；“TempSmallContextGroup”内存上下文中该字段含义为统计计数。

5.4.18.GS_SESSION_MEMORY_STATISTICS

GS_SESSION_MEMORY_STATISTICS 视图显示和当前用户执行复杂作业正在运行时的负载管理内存使用的信息。

表 1 GS_SESSION_MEMORY_STATISTICS 字段

名称	类型	描述
datid	oid	连接后端的数据库 OID。

名称	类型	描述
usename	name	登录到该后端的用户名。
pid	bigint	后端线程 ID。
start_time	timestamp with time zone	语句执行的开始时间。
min_peak_memory	integer	语句在数据库节点上的最小内存峰值大小，单位 MB。
max_peak_memory	integer	语句在数据库节点上的最大内存峰值大小，单位 MB。
spill_info	text	语句在数据库节点上的下盘信息：None：数据库节点均未下盘。All：数据库节点均下盘。[a:b]：数量为 b 个数据库节点中有 a 个数据库节点下盘。
query	text	正在执行的语句。
node_group	text	该字段不支持。
top_mem_dn	text	mem 使用量信息。

5.4.19.GS_SESSION_STAT

GS_SESSION_STAT 视图以会话线程或 AutoVacuum 线程为单位，统计会话状态信息。

表 1 GS_SESSION_STAT 字段

名称	类型	描述
sessid	text	线程标识+线程启动时间。
statid	integer	统计编号。
statname	text	统计会话名称。
statunit	text	统计会话单位。
value	bigint	统计会话值。

5.4.20.GS_SESSION_TIME

GS_SESSION_TIME 视图用于统计会话线程的运行时间信息，及各执行阶段所消耗时间。

表 1 GS_SESSION_TIME 字段

名称	类型	描述
sessid	text	线程标识+线程启动时间。
stat_id	integer	统计编号。
stat_name	text	会话类型名称。
value	bigint	会话值。

5.4.21.GS_SQL_COUNT

GS_SQL_COUNT 视图显示数据库当前节点当前时刻执行的五类语句（SELECT、INSERT、UPDATE、DELETE、MERGE INTO）统计信息。

- 普通用户查询 GS_SQL_COUNT 视图仅能看到该用户当前节点的统计信息；管理员权限用户查询 GS_SQL_COUNT 视图则能看到所有用户当前节点的统计信息。
- 当 PanWeiDB 或该节点重启时，计数将清零，并重新开始计数 。
- 计数以节点收到的查询数为准，包括 PanWeiDB 内部进行的查询。

表 1 GS_SQL_COUNT 字段

名称	类型	描述
node_name	text	节点名称。
user_name	text	用户名。
select_count	bigint	select 语句统计结果。
update_count	bigint	update 语句统计结果。
insert_count	bigint	insert 语句统计结果。
delete_count	bigint	delete 语句统计结果。
mergeinto_count	bigint	MERGE INTO 语句统计结果。
ddl_count	bigint	DDL 语句的数量。

名称	类型	描述
dml_count	bigint	DML 语句的数量。
dcl_count	bigint	DML 语句的数量。
total_select_elapse	bigint	总 select 的时间花费（单位：微秒）。
avg_select_elapse	bigint	平均 select 的时间花费（单位：微秒）。
max_select_elapse	bigint	最大 select 的时间花费（单位：微秒）。
min_select_elapse	bigint	最小 select 的时间花费（单位：微秒）。
total_update_elapse	bigint	总 update 的时间花费（单位：微秒）。
avg_update_elapse	bigint	平均 update 的时间花费（单位：微秒）。
max_update_elapse	bigint	最大 update 的时间花费（单位：微秒）。
min_update_elapse	bigint	最小 update 的时间花费（单位：微秒）。
total_insert_elapse	bigint	总 insert 的时间花费（单位：微秒）。
avg_insert_elapse	bigint	平均 insert 的时间花费（单位：微秒）。
max_insert_elapse	bigint	最大 insert 的时间花费（单位：微秒）。
min_insert_elapse	bigint	最小 insert 的时间花费（单位：微秒）。
total_delete_elapse	bigint	总 delete 的时间花费（单位：微秒）。
avg_delete_elapse	bigint	平均 delete 的时间花费（单位：微秒）。
max_delete_elapse	bigint	最大 delete 的时间花费（单位：微秒）。
min_delete_elapse	bigint	最小 delete 的时间花费（单位：微秒）。

5.4.22.GS_STAT_SESSION_CU

GS_STAT_SESSION_CU 视图查询 PanWeiDB 各个节点，当前运行 session 的 CU 命中情况。session 退出相应的统计数据会清零。PanWeiDB 重启后，统计数据也会清零。

表 1 GS_STAT_SESSION_CU 字段

名称	类型	描述
mem_hit	integer	内存命中次数。
hdd_sync_read	integer	硬盘同步读次数。
hdd_asyn_read	integer	硬盘异步读次数。

5.4.23.GS_THREAD_MEMORY_CONTEXT

GS_THREAD_MEMORY_CONTEXT 视图统计所有的线程的内存使用情况，以 MemoryContext 节点来统计。该视图在关闭线程池（enable_thread_pool = off）时等价于 GS_SESSION_MEMORY_DETAIL 视图。

其中内存上下文 “TempSmallContextGroup”，记录当前线程中所有内存上下文字段 “totalsize” 小于 8192 字节的信息汇总，并且内存上下文统计计数记录到 “usedsize” 字段中。所以在视图中，“TempSmallContextGroup” 内存上下文中的 “totalsize” 和 “freesize” 是该线程中所有内存上下文 “totalsize” 小于 8192 字节的汇总总和，usedsize 字段表示统计的内存上下文个数。

表 1 GS_THREAD_MEMORY_CONTEXT 字段

名称	类型	描述
threadid	text	线程启动时间+线程标识（字符串信息为 timestamp.sessionid）。
tid	bigint	线程标识。
thrdtype	text	线程类型。
contextname	text	内存上下文名称。

名称	类型	描述
level	smallint	当前上下文在整体内存上下文中的层级。
parent	text	父内存上下文名称。
totalsize	bigint	当前内存上下文的内存总数，单位 Byte。
freesize	bigint	当前内存上下文中已释放的内存总数，单位 Byte。
usedsize	bigint	当前内存上下文中已使用的内存总数，单位 Byte；“TempSmallContextGroup”内存上下文中该字段含义为统计计数。

5.4.24.GS_TOTAL_MEMORY_DETAIL

GS_TOTAL_MEMORY_DETAIL 视图统计当前数据库节点使用内存的信息，单位为 MB。当 GUC 参数 enable_memory_limit 的值为 off 时，本视图不可用。

表 1 GS_TOTAL_MEMORY_DETAIL 字段

名称	类型	描述
nodename	text	节点名称。
memorytype	text	内存类型，包括以下几种： ❖ max_process_memory：PanWeiDB 实例所占用的内存大小。 ❖ process_used_memory：PanWeiDB 进程所使用的内存大小。 ❖ max_dynamic_memory：最大动态内存。 ❖ dynamic_used_memory：已使用的动态内存。 ❖ dynamic_peak_memory：内存的动态峰值。 ❖ dynamic_used_shrctx：最大动态共享内存上下文。 ❖ dynamic_peak_shrctx：共享内存上下文的动态峰值。 ❖ max_shared_memory：最大共享内存。 ❖ shared_used_memory：已使用的共享内存。 ❖ max_cstore_memory：列存所允许使用的最大内存。

名称	类型	描述
		❖ cstore_used_memory：列存已使用的内存大小。 ❖ max_sctpcomm_memory：通信库所允许使用的最大内存。 ❖ sctpcomm_used_memory：通信库已使用的内存大小。 ❖ sctpcomm_peak_memory：通信库的内存峰值。 ❖ other_used_memory：其他已使用的内存大小。
memorymbytes	integer	内存类型分配内存的大小。

5.4.25.GS_WLM_CGROUP_INFO

GS_WLM_CGROUP_INFO 视图显示当前执行作业的控制组的信息。

表 1 GS_WLM_CGROUP_INFO 字段

名称	类型	描述
cgroup_name	text	控制组的名称。
priority	interger	作业的优先级。
usage_percent	interger	控制组占用的百分比。
shares	bigint	控制组分配的 CPU 资源配额。
cpuacct	bigint	CPU 配额分配。
cpuset	text	CPU 限额分配。
relpath	text	控制组的相对路径。
valid	text	该控制组是否有效。
node_group	text	逻辑数据库实例名称。

5.4.26.GS_WLM_EC_OPERATOR_STATISTICS

GS_WLM_EC_OPERATOR_STATISTICS 视图显示当前用户正在执行的 EC（Extension Connector）作业的算子相关信息。查询该视图需要 sysadmin 权限。

表 1 GS_WLM_EC_OPERATOR_STATISTICS 的字段

名称	类型	描述
queryid	bigint	EC 语句执行使用的内部 query_id。
plan_node_id	integer	EC 算子对应的执行计划的 plan node id。
start_time	timestamp with time zone	EC 算子处理第一条数据的开始时间。
ec_status	text	EC 作业的执行状态。 ❖ EC_STATUS_INIT：初始化。 ❖ EC_STATUS_CONNECTED：已连接。 ❖ EC_STATUS_EXECUTED：已执行。 ❖ EC_STATUS_FETCHING：获取中。 ❖ EC_STATUS_END：已结束。
ec_execute_datanode	text	执行 EC 作业的 DN 名称。
ec_dsn	text	EC 作业所使用的 DSN。
ec_username	text	EC 作业访问远端数据库实例的 USERNAME（远端集群为 SPARK 类型时该值为空）。
ec_query	text	EC 作业发送给远端数据库实例执行的语句。
ec_libodbc_type	text	EC 作业使用的 unixODBC 驱动类型。 类型 1：对应 libodbc.so.1。 类型 2：对应 libodbc.so.2。
ec_fetch_count	bigint	EC 作业当前处理的数据条数。

5.4.27.GS_WLM_OPERATOR_HISTORY

GS_WLM_OPERATOR_HISTORY 视图显示的是当前用户当前数据库主节点上执行作业结束后的算子的相关记录。查询该视图需要 sysadmin 权限。

记录的数据同 GS_WLM_OPERATOR_INFO 表 1（详细请参考：系统表和系统视图->系统表->GS_WLM_OPERATOR_INFO 章节）。

5.4.28.GS_WLM_OPERATOR_STATISTICS

GS_WLM_OPERATOR_STATISTICS 视图显示当前用户正在执行的作业的算子相关信息。查询该视图需要 sysadmin 权限。

表 1 GS_WLM_OPERATOR_STATISTICS 的字段

名称	类型	描述
queryid	bigint	语句执行使用的内部 query_id。
pid	bigint	后端线程 id。
plan_node_id	integer	查询对应的执行计划的 plan node id。
plan_node_name	text	对应于 plan_node_id 的算子的名称。
start_time	timestamp with time zone	该算子处理第一条数据的开始时间。
duration	bigint	该算子到结束时候总的执行时间（ms）。
status	text	当前算子的执行状态，包括 finished 和 running。
query_dop	integer	当前算子执行时的并行度。
estimated_rows	bigint	优化器估算的行数信息。
tuple_processed	bigint	当前算子返回的元素个数。
min_peak_memory	integer	当前算子在数据库实例上的最小内存峰值（MB）。
max_peak_memory	integer	当前算子在数据库实例上的最大内存峰值（MB）。
average_peak_memory	integer	当前算子在数据库实例上的平均内存峰值（MB）。
memory_skew_percent	integer	当前算子在数据库实例间的内存使用倾斜率。
min_spill_size	integer	若发生下盘，数据库实例上下盘的最小数据量（MB），默认为 0。
max_spill_size	integer	若发生下盘，数据库实例上下盘的最大数据量（MB），默认为 0。
average_spill_size	integer	若发生下盘，数据库实例上下盘的平均数据量（MB），默认为 0。
spill_skew_percent	integer	若发生下盘，数据库实例间下盘倾斜率。
min_cpu_time	bigint	该算子在数据库实例上的最小执行时间（ms）。

名称	类型	描述
max_cpu_time	bigint	该算子在数据库实例上的最大执行时间（ms)。
total_cpu_time	bigint	该算子在数据库实例上的总执行时间（ms）。
cpu_skew_percent	integer	数据库实例间执行时间的倾斜率。
warning	text	主要显示如下几类告警信息： ❖ Sort/SetOp/HashAgg/HashJoin spill ❖ Spill file size large than 256MB ❖ Broadcast size large than 100MB ❖ Early spill ❖ Spill times is greater than 3 ❖ Spill on memory adaptive ❖ Hash table conflict

5.4.29.GS_WLM_PLAN_OPERATOR_HISTORY

GS_WLM_PLAN_OPERATOR_HISTORY 视图显示的是当前用户数据库主节点上执行作业结束后的执行计划算子级的相关记录。

记录的数据同 GS_WLM_PLAN_OPERATOR_INFO 表 1（详细请参考：系统表和系统视图->系统表->GS_WLM_PLAN_OPERATOR_INFO 章节）。

5.4.30.GS_WLM_REBUILD_USER_RESOURCE_POOL

该视图用于在当前连接节点上重建内存中用户的资源池信息，无输出。只是用于资源池信息缺失或者错乱时用作补救措施。查询该视图需要 sysadmin 权限。

表 1 GS_WLM_REBUILD_USER_RESOURCE_POOL 的字段

名称	类型	描述
gs_wlm_rebuild_user_resource_pool	boolean	重建内存中用户资源池信息结果。 t 为成功，f 为失败。

5.4.31.GS_WLM_RESOURCE_POOL

这是资源池上的一些统计信息。

表 1 GS_WLM_RESOURCE_POOL 的字段

名称	类型	描述
rpoid	oid	资源池的 OID。
respool	name	资源池的名称。
control_group	name	该字段不支持。
parentid	oid	父资源池的 OID。
ref_count	integer	关联到该资源池上的作业数量。
active_points	integer	资源池上已经使用的点数。
running_count	integer	正在资源池上运行的作业数量。
waiting_count	integer	正在资源池上排队的作业数量。
io_limits	integer	资源池的 iops 上限。
io_priority	integer	资源池的 io 优先级。

5.4.32.GS_WLM_SESSION_INFO

GS_WLM_SESSION_INFO 视图显示数据库实例执行作业结束后的负载管理记录。查询该视图需要 sysadmin 权限。

具体的字段请参考：系统表和系统视图->系统视图->GS_WLM_SESSION_HISTORY 章节表 1 字段中的信息。

5.4.33.GS_WLM_SESSION_HISTORY

GS_WLM_SESSION_HISTORY 视图显示当前用户在数据库实例上执行作业结束后的负载管理记录。查询该视图需要 sysadmin 或者 monitor admin 权限。

表 1 GS_WLM_SESSION_HISTORY 的字段

名称	类型	描述
datid	oid	连接后端的数据库 OID。
dbname	text	连接后端的数据库名称。
schemaname	text	模式的名称。
nodename	text	语句执行的数据库实例名称。
username	text	连接到后端的用户名。
application_name	text	连接到后端的应用名。
client_addr	inet	连接到后端的客户端的 IP 地址。 如果此字段是 null，它表明通过服务器机器上

名称	类型	描述
		UNIX 套接字连接客户端或者这是内部进程，如 autovacuum。
client_hostname	text	客户端的主机名，这个字段是通过 client_addr 的反向 DNS 查找得到。这个字段只有在启动 log_hostname 且使用 IP 连接时才非空。
client_port	integer	客户端用于与后端通讯的 TCP 端口号，如果使用 Unix 套接字，则为-1。
query_band	text	用于标示作业类型，可通过 GUC 参数 query_band 进行设置，默认为空字符串。
block_time	bigint	语句执行前的阻塞时间，包含语句解析和优化时间，单位 ms。
start_time	timestamp with time zone	语句执行的开始时间。
finish_time	timestamp with time zone	语句执行的结束时间。
duration	bigint	语句实际执行的时间，单位 ms。
estimate_total_time	bigint	语句预估执行时间，单位 ms。
status	text	语句执行结束状态：正常为 finished，异常为 aborted。
abort_info	text	语句执行结束状态为 aborted 时显示异常信息。
resource_pool	text	用户使用的资源池。
control_group	text	语句所使用的 Cgroup。
estimate_memory	integer	语句估算内存大小。
min_peak_memory	integer	语句在数据库实例上的最小内存峰值，单位 MB。
max_peak_memory	integer	语句在数据库实例上的最大内存峰值，单位 MB。
average_peak_memory	integer	语句执行过程中的内存使用平均值，单位 MB。

名称	类型	描述
memory_skew_percent	integer	语句数据库实例间的内存使用倾斜率。
spill_info	text	语句在数据库实例上的下盘信息：None：数据库实例均未下盘。All：数据库实例均下盘。[a:b]：数量为 b 个数据库实例中有 a 个数据库实例下盘。
min_spill_size	integer	若发生下盘，数据库实例上下盘的最小数据量，单位 MB，默认为 0。
max_spill_size	integer	若发生下盘，数据库实例上下盘的最大数据量，单位 MB，默认为 0。
average_spill_size	integer	若发生下盘，数据库实例上下盘的平均数据量，单位 MB，默认为 0。
spill_skew_percent	integer	若发生下盘，数据库实例间下盘倾斜率。
min_dn_time	bigint	语句在数据库实例上的最小执行时间，单位 ms。
max_dn_time	bigint	语句在数据库实例上的最大执行时间，单位 ms。
average_dn_time	bigint	语句在数据库实例上的平均执行时间，单位 ms。
dntime_skew_percent	integer	语句在数据库实例间的执行时间倾斜率。
min_cpu_time	bigint	语句在数据库实例上的最小 CPU 时间，单位 ms。
max_cpu_time	bigint	语句在数据库实例上的最大 CPU 时间，单位 ms。
total_cpu_time	bigint	语句在数据库实例上的 CPU 总时间，单位 ms。
cpu_skew_percent	integer	语句在数据库实例间的 CPU 时间倾斜率。
min_peak_iops	integer	语句在数据库实例上的每秒最小 IO 峰值（列存单位是次/s，行存单位是万次/s）。
max_peak_iops	integer	语句在数据库实例上的每秒最大 IO 峰值（列存单位是次/s，行存单位是万次/s）。
average_peak_iops	integer	语句在数据库实例上的每秒平均 IO 峰值

名称	类型	描述
		（列存单位是次/s，行存单位是万次/s）。
iops_skew_percent	integer	语句在数据库实例间的 IO 倾斜率。
warning	text	主要显示如下几类告警信息： ❖ Spill file size large than 256MB ❖ Broadcast size large than 100MB ❖ Early spill ❖ Spill times is greater than 3 ❖ Spill on memory adaptive ❖ Hash table conflict
queryid	bigint	语句执行使用的内部 query id。
query	text	执行的语句。
query_plan	text	语句的执行计划。
node_group	text	语句所属用户对应的逻辑数据库实例。
cpu_top1_node_name	text	cpu 使用率第 1 的节点名称。
cpu_top2_node_name	text	cpu 使用率第 2 的节点名称。
cpu_top3_node_name	text	cpu 使用率第 3 的节点名称。
cpu_top4_node_name	text	cpu 使用率第 4 的节点名称。
cpu_top5_node_name	text	cpu 使用率第 5 的节点名称。
mem_top1_node_name	text	内存使用量第 1 的节点名称。
mem_top2_node_name	text	内存使用量第 2 的节点名称。
mem_top3_node_name	text	内存使用量第 3 的节点名称。
mem_top4_node_name	text	内存使用量第 4 的节点名称。
mem_top5_node_name	text	内存使用量第 5 的节点名称。
cpu_top1_value	bigint	cpu 使用率。
cpu_top2_value	bigint	cpu 使用率。
cpu_top3_value	bigint	cpu 使用率。
cpu_top4_value	bigint	cpu 使用率。
cpu_top5_value	bigint	cpu 使用率。
mem_top1_value	bigint	内存使用量。
mem_top2_value	bigint	内存使用量。
mem_top3_value	bigint	内存使用量。
mem_top4_value	bigint	内存使用量。

名称	类型	描述
mem_top5_value	bigint	内存使用量。
top_mem_dn	text	内存使用量 topN 信息。
top_cpu_dn	text	cpu 使用量 topN 信息。

5.4.34.GS_WLM_SESSION_INFO_ALL

GS_WLM_SESSION_INFO_ALL 视图显示在数据库实例上执行作业结束后的负载管理记录。查询该视图需要 sysadmin 或者 monitor admin 权限。

表 1 GS_WLM_SESSION_INFO_ALL 的字段

名称	类型	描述
datid	oid	连接后端的数据库 OID。
dbname	text	连接后端的数据库名称。
schemaname	text	模式的名称。
nodename	text	语句执行的 CN 名称。
username	text	连接到后端的用户名。
application_name	text	连接到后端的应用名。
client_addr	inet	连接到后端的客户端的 IP 地址。如果此字段是 null，它表明通过服务器机器上 UNIX 套接字连接客户端或者这是内部进程，如 autovacuum。
client_hostname	text	客户端的主机名，这个字段是通过 client_addr 的反向 DNS 查找得到。这个字段只有在启动 log_hostname 且使用 IP 连接时才非空。
client_port	integer	客户端用于与后端通讯的 TCP 端口号，如果使用 Unix 套接字，则为-1。
query_band	text	用于标示作业类型，可通过 GUC 参数 query_band 进行设置，默认为空字符串。
block_time	bigint	语句执行前的阻塞时间，包含语句解析和优化时间，单位 ms。
start_time	timestamp with time	语句执行的开始时间。

名称	类型	描述
	zone	
finish_time	timestamp with time zone	语句执行的结束时间。
duration	bigint	语句实际执行的时间，单位 ms。
estimate_total_time	bigint	语句预估执行时间，单位 ms。
status	text	语句执行结束状态：正常为 finished，异常为 aborted。
abort_info	text	语句执行结束状态为 aborted 时显示异常信息。
resource_pool	text	用户使用的资源池。
control_group	text	语句所使用的 Cgroup。
estimate_memory	integer	语句估算内存大小。
min_peak_memory	integer	语句在所有 DN 上的最小内存峰值，单位 MB。
max_peak_memory	integer	语句在所有 DN 上的最大内存峰值，单位 MB。
average_peak_memory	integer	语句执行过程中的内存使用平均值，单位 MB。
memory_skew_percent	integer	语句各 DN 间的内存使用倾斜率。
spill_info	text	语句在所有 DN 上的下盘信息：None：所有 DN 均未下盘。All：所有 DN 均下盘。[a:b]：数量为 b 个 DN 中有 a 个 DN 下盘。
min_spill_size	integer	若发生下盘，所有 DN 上下盘的最小数据量，单位 MB，默认为 0。
max_spill_size	integer	若发生下盘，所有 DN 上下盘的最大数据量，单位 MB，默认为 0。
average_spill_size	integer	若发生下盘，所有 DN 上下盘的平均数据量，单位 MB，默认为 0。
spill_skew_percent	integer	若发生下盘，DN 间下盘倾斜率。
min_dn_time	bigint	语句在所有 DN 上的最小执行时间，单位 ms。

名称	类型	描述
max_dn_time	bigint	语句在所有 DN 上的最大执行时间，单位 ms。
average_dn_time	bigint	语句在所有 DN 上的平均执行时间，单位 ms。
dntime_skew_percent	integer	语句在各 DN 间的执行时间倾斜率。
min_cpu_time	bigint	语句在所有 DN 上的最小 CPU 时间，单位 ms。
max_cpu_time	bigint	语句在所有 DN 上的最大 CPU 时间，单位 ms。
total_cpu_time	bigint	语句在所有 DN 上的 CPU 总时间，单位 ms。
cpu_skew_percent	integer	语句在 DN 间的 CPU 时间倾斜率。
min_peak_iops	integer	语句在所有 DN 上的每秒最小 IO 峰值（列存单位是次/s，行存单位是万次/s）。
max_peak_iops	integer	语句在所有 DN 上的每秒最大 IO 峰值（列存单位是次/s，行存单位是万次/s）。
average_peak_iops	integer	语句在所有 DN 上的每秒平均 IO 峰值（列存单位是次/s，行存单位是万次/s）。
iops_skew_percent	integer	语句在 DN 间的 IO 倾斜率。
warning	text	主要显示如下几类告警信息以及 SQL 自诊断调优相关告警： ❖ Spill file size large than 256MB ❖ Broadcast size large than 100MB ❖ Early spill ❖ Spill times is greater than 3 ❖ Spill on memory adaptive ❖ Hash table conflict
queryid	bigint	语句执行使用的内部 query id。
query	text	执行的语句。
query_plan	text	语句的执行计划。
node_group	text	语句所属用户对应的逻辑集群。
cpu_top1_node_name	text	cpu 使用率第 1 的节点名称。
cpu_top2_node_name	text	cpu 使用率第 2 的节点名称。

名称	类型	描述
cpu_top3_node_name	text	cpu 使用率第 3 的节点名称。
cpu_top4_node_name	text	cpu 使用率第 4 的节点名称。
cpu_top5_node_name	text	cpu 使用率第 5 的节点名称。
mem_top1_node_name	text	内存使用量第 1 的节点名称。
mem_top2_node_name	text	内存使用量第 2 的节点名称。
mem_top3_node_name	text	内存使用量第 3 的节点名称。
mem_top4_node_name	text	内存使用量第 4 的节点名称。
mem_top5_node_name	text	内存使用量第 5 的节点名称。
cpu_top1_value	bigint	cpu 使用率。
cpu_top2_value	bigint	cpu 使用率。
cpu_top3_value	bigint	cpu 使用率。
cpu_top4_value	bigint	cpu 使用率。
cpu_top5_value	bigint	cpu 使用率。
mem_top1_value	bigint	内存使用量。
mem_top2_value	bigint	内存使用量。
mem_top3_value	bigint	内存使用量。
mem_top4_value	bigint	内存使用量。
mem_top5_value	bigint	内存使用量。
top_mem_dn	text	内存使用量 topN 信息。
top_cpu_dn	text	cpu 使用量 topN 信息。
n_returned_rows	bigint	SELECT 返回的结果集行数。
n_tuples_fetched	bigint	随机扫描行。
n_tuples_returned	bigint	顺序扫描行。
n_tuples_inserted	bigint	插入行。
n_tuples_updated	bigint	更新行。
n_tuples_deleted	bigint	删除行。
n_blocks_fetched	bigint	buffer 的块访问次数。
n_blocks_hit	bigint	buffer 的块命中次数。
db_time	bigint	有效的 DB 时间花费，多线程将累加（单位：微秒）。
cpu_time	bigint	CPU 时间（单位：微秒）。
execution_time	bigint	执行器内执行时间（单位：微秒）。
parse_time	bigint	SQL 解析时间（单位：微秒）。

名称	类型	描述
plan_time	bigint	SQL 生成计划时间（单位：微秒）。
rewrite_time	bigint	SQL 重写时间（单位：微秒）。
pl_execution_time	bigint	plpgsql 上的执行时间（单位：微秒）。
pl_compilation_time	bigint	plpgsql 上的编译时间（单位：微秒）。
net_send_time	bigint	网络上的时间花费（单位：微秒）。
data_io_time	bigint	IO 上的时间花费（单位：微秒）。
is_slow_query	bigint	是否是慢 SQL 记录。

5.4.35.GS_WLM_SESSION_STATISTICS

GS_WLM_SESSION_STATISTICS 视图显示当前用户在数据库实例上正在执行的作业的负载管理记录。查询该视图需要 sysadmin 权限。

表 1 GS_WLM_SESSION_STATISTICS 的字段

名称	类型	描述
datid	oid	连接后端的数据 OID。
dbname	name	连接后端的数据库名称。
schemaname	text	模式的名称。
nodename	text	语句执行的数据库实例名称。
username	name	连接到后端的用户名。
application_name	text	连接到后端的应用名。
client_addr	inet	连接到后端的客户端的 IP 地址。如果此字段是 null，它表明通过服务器机器上 UNIX 套接字连接客户端或者这是内部进程，如 autovacuum。
client_hostname	text	客户端的主机名，这个字段是通过 client_addr 的反向 DNS 查找得到。这个字段只有在启动 log_hostname 且使用 IP 连接时才非空。
client_port	integer	客户端用于与后端通讯的 TCP 端口号，如果使用 Unix 套接字，则为-1。
query_band	text	用于标示作业类型，可通过 GUC 参数 query_band 进行设置，默认为空字符串。
pid	bigint	后端线程 ID。

名称	类型	描述
sessionid	bigint	会话 ID。
block_time	bigint	语句执行前的阻塞时间，单位 ms。
start_time	timestamp with time zone	语句执行的开始时间。
duration	bigint	语句已经执行的时间，单位 ms。
estimate_total_time	bigint	语句执行预估总时间，单位 ms。
estimate_left_time	bigint	语句执行预估剩余时间，单位 ms。
enqueue	text	工作负载管理资源状态。
resource_pool	name	用户使用的资源池。
control_group	text	语句所使用的 Cgroup。
estimate_memory	integer	语句预估使用内存，单位 MB。
min_peak_memory	integer	语句在数据库实例上的最小内存峰值，单位 MB。
max_peak_memory	integer	语句在数据库实例上的最大内存峰值，单位 MB。
average_peak_memory	integer	语句执行过程中的内存使用平均值，单位 MB。
memory_skew_percent	integer	语句在数据库实例间的内存使用倾斜率。
spill_info	text	语句在数据库实例上的下盘信息： ❖ None：数据库实例均未下盘。 ❖ All：数据库实例均下盘。 ❖ [a:b]：数量为 b 个数据库实例中有 a 个数据库实例下盘。
min_spill_size	integer	若发生下盘，数据库实例上下盘的最小数据量，单位 MB，默认为 0。
max_spill_size	integer	若发生下盘，数据库实例上下盘的最大数据量，单位 MB，默认为 0。
average_spill_size	integer	若发生下盘，数据库实例上下盘的平均数据量，单位 MB，默认为 0。
spill_skew_percent	integer	若发生下盘，数据库实例间下盘倾斜率。
min_dn_time	bigint	语句在数据库实例上的最小执行时间，单位 ms。

名称	类型	描述
max_dn_time	bigint	语句在数据库实例上的最大执行时间，单位ms。
average_dn_time	bigint	语句在数据库实例上的平均执行时间，单位ms。
dntime_skew_percent	integer	语句在数据库实例间的执行时间倾斜率。
min_cpu_time	bigint	语句在数据库实例上的最小 CPU 时间，单位 ms。
max_cpu_time	bigint	语句在数据库实例上的最大 CPU 时间，单位 ms。
total_cpu_time	bigint	语句在数据库实例上的 CPU 总时间，单位ms。
cpu_skew_percent	integer	语句在数据库实例间的 CPU 时间倾斜率。
min_peak_iops	integer	语句在数据库实例上的每秒最小 IO 峰值（列存单位是次/s，行存单位是万次/s）。
max_peak_iops	integer	语句在数据库实例上的每秒最大 IO 峰值（列存单位是次/s，行存单位是万次/s）。
average_peak_iops	integer	语句在数据库实例上的每秒平均 IO 峰值（列存单位是次/s，行存单位是万次/s）。
iops_skew_percent	integer	语句在数据库实例间的 IO 倾斜率。
warning	text	主要显示如下几类告警信息： ❖ Spill file size large than 256MB ❖ Broadcast size large than 100MB ❖ Early spill ❖ Spill times is greater than 3 ❖ Spill on memory adaptive ❖ Hash table conflict
queryid	bigint	语句执行使用的内部 query id。
query	text	正在执行的语句。
query_plan	text	语句的执行计划。
node_group	text	语句所属用户对应的逻辑数据库实例。
top_cpu_dn	text	cpu 使用量 topN 信息。
top_mem_dn	text	内存使用量 topN 信息。

5.4.36.GS_WLM_USER_INFO

用户统计信息视图。

表 1 GS_WLM_USER_INFO 字段

名称	类型	描述
userid	oid	用户 OID。
username	name	用户名。
sysadmin	boolean	是否是管理员用户。
rpoid	oid	关联的资源池的 OID。
respool	name	关联的资源池的名称。
parentid	oid	用户组的 OID。
totalspace	bigint	用户的可用空间上限。
spacelimit	bigint	用户表空间限制。
childcount	interger	子用户的个数。
childlist	text	子用户列表。

5.4.37.MPP_TABLES

MPP_TABLES 视图显示信息如下。

表 1 MPP_TABLES 字段

名称	类型	描述
schemaname	name	包含表的模式名。
tablename	name	表名。
tableowner	name	表的所有者。
tablespace	name	表所在的表空间。

名称	类型	描述
pgroup	name	节点群的名称。
nodeoids	oidvector_extend	表分布的节点 OID 列表。

5.4.38.PG_AVAILABLE_EXTENSION_VERSIONS

PG_AVAILABLE_EXTENSION_VERSIONS 视图显示数据库中某些特性的扩展版本信息。

表 1 PG_AVAILABLE_EXTENSION_VERSIONS 字段

名称	类型	描述
name	name	扩展名。
version	text	版本名。
installed	Boolean	如果这个扩展的版本是当前已经安装了的则为真。
superuser	Boolean	如果只允许系统管理员安装这个扩展则为真。
relocatable	Boolean	如果扩展可以重新加载到另一个模式则为真。
schema	name	扩展必须安装到的模式名，如果部分或全部可重新定位则为 NULL。
requires	name[]	先决条件扩展的名称，如果没有则为 NULL。
comment	text	扩展的控制文件中的评论字符串。

5.4.39.PG_AVAILABLE_EXTENSIONS

PG_AVAILABLE_EXTENSIONS 视图显示数据库中某些特性的扩展信息。

表 1 PG_AVAILABLE_EXTENSIONS 字段

名称	类型	描述
name	name	扩展名。
default_version	text	缺省版本的名称，如果没有指定则为 NULL。
installed_version	text	扩展当前安装版本，如果没有安装任何版本则为 NULL。
comment	text	扩展的控制文件中的评论字符串。

5.4.40.PG_COMM_DELAY

PG_COMM_DELAY 视图展示单个节点的通信库时延状态。

表 1 PG_COMM_DELAY 字段

名称	类型	描述
node_name	text	节点名称。
remote_name	text	连接对端节点名称。
remote_host	text	连接对端 IP 地址。
stream_num	integer	当前物理连接使用的 stream 逻辑连接数量。
min_delay	integer	当前物理连接一分钟内探测到的最小时延，单位微秒。说明：负数结果无效，请重新等待时延状态更新后再执行。
average	integer	当前物理连接一分钟内探测时延的平均值，单位微秒。
max_delay	integer	当前物理连接一分钟内探测到的最大时延，单位微秒。

5.4.41.PG_COMM_RECV_STREAM

PG_COMM_RECV_STREAM 视图展示节点上所有的通信库接收流状态。

表 1 PG_COMM_RECV_STREAM 字段

名称	类型	描述
node_name	text	节点名称。
local_tid	bigint	使用此通信流的线程 ID。
remote_name	text	连接对端节点名称。
remote_tid	bigint	连接对端线程 ID。
idx	integer	通信对端 DN 在本 DN 内的标识编号。
sid	integer	通信流在物理连接中的标识编号。
tcp_sock	integer	通信流所使用的 tcp 通信 socket。
state	text	通信流当前的状态。 ❖ UNKNOWN：表示当前逻辑连接状态未知。 ❖ READY：表示逻辑连接已就绪。

名称	类型	描述
		❖ RUN：表示逻辑连接发送报文正常。 ❖ HOLD：表示逻辑连接发送报文等待中。 ❖ CLOSED：表示关闭逻辑连接。 TO_CLOSED：表示将会关闭逻辑连接。
query_id	bigint	通信流对应的 debug_query_id 编号。
pn_id	integer	通信流所执行查询的 plan_node_id 编号。
send_smp	integer	通信流所执行查询 send 端的 smpid 编号。
recv_smp	integer	通信流所执行查询 recv 端的 smpid 编号。
recv_bytes	bigint	通信流接收的数据总量，单位 Byte。
time	bigint	通信流当前生命周期使用时长，单位 ms。
speed	bigint	通信流的平均接收速率，单位 Byte/s。
quota	bigint	通信流当前的通信配额值，单位 Byte。
buff_usize	bigint	通信流当前缓存的数据大小，单位 Byte。

5.4.42.PG_COMM_SEND_STREAM

PG_COMM_SEND_STREAM 视图展示节点上所有的通信库发送流状态。

表 1 PG_COMM_SEND_STREAM 字段

名称	类型	描述
node_name	text	节点名称。
local_tid	bigint	使用此通信流的线程 ID。
remote_name	text	连接对端节点名称。
remote_tid	bigint	连接对端线程 ID。
idx	integer	通信对端 DN 在本 DN 内的标识编号。
sid	integer	通信流在物理连接中的标识编号。
tcp_sock	integer	通信流所使用的 tcp 通信 socket。
state	text	通信流当前的状态。 ❖ UNKNOWN：表示当前逻辑连接状态未知。 ❖ READY：表示逻辑连接已就绪。 ❖ RUN：表示逻辑连接发送报文正常。

名称	类型	描述
		❖ HOLD：表示逻辑连接发送报文等待中。 ❖ CLOSED：表示关闭逻辑连接。 ❖ TO_CLOSED：表示将会关闭逻辑连接。
query_id	bigint	通信流对应的 debug_query_id 编号。
pn_id	integer	通信流所执行查询的 plan_node_id 编号。
send_smp	integer	通信流所执行查询 send 端的 smpid 编号。
recv_smp	integer	通信流所执行查询 recv 端的 smpid 编号。
send_bytes	bigint	通信流发送的数据总量，单位 Byte。
time	bigint	通信流当前生命周期使用时长，单位 ms。
speed	bigint	通信流的平均发送速率，单位 Byte/s。
quota	bigint	通信流当前的通信配额值，单位 Byte。
wait_quota	bigint	通信流等待 quota 值产生的额外时间开销，单位 ms。

5.4.43.PG_COMM_STATUS

PG_COMM_STATUS 视图展示节点的通信库状态。

表 1 PG_COMM_STATUS 字段

名称	类型	描述
node_name	text	节点名称。
rxpck_rate	integer	节点通信库接收速率，单位 Byte/s。
txpck_rate	integer	节点通信库发送速率，单位 Byte/s。
rxkbyte_rate	bigint	bigint 节点通信库接收速率，单位 KByte/s。
txkbyte_rate	bigint	bigint 节点通信库发送速率，单位 KByte/s。
buffer	bigint	cmailbox 的 buffer 大小。
memkbyte_libcomm	bigint	libcomm 进程通信内存大小，单位 Byte。
memkbyte_libpq	bigint	libpq 进程通信内存大小，单位 Byte。
used_pm	integer	postmaster 线程实时使用率。

名称	类型	描述
used_sflow	integer	gs_sender_flow_controller 线程实时使用率。
used_rflow	integer	gs_receiver_flow_controller 线程实时使用率。
used_rloop	integer	多个 gs_receivers_loop 线程中高的实时使用率。
stream	integer	当前使用的逻辑连接总数。

5.4.44.PG_CONTROL_GROUP_CONFIG

PG_CONTROL_GROUP_CONFIG 视图存储系统的控制组配置信息。查询该视图需要 sysadmin 权限。

表 1 PG_CONTROL_GROUP_CONFIG 字段

名称	类型	描述
pg_control_group_config	text	控制组的配置信息。

5.4.45.PG_CURSORS

PG_CURSORS 视图列出了当前可用的游标。

表 1 PG_CURSORS 字段

名称	类型	描述
name	text	游标名。
statement	text	声明改游标时的查询语句。
is_holdable	Boolean	如果该游标是持久的（就是在声明该游标的事务结束后仍然可以访问该游标）则为 TRUE，否则为 FALSE。
is_binary	Boolean	如果该游标被声明为 BINARY 则为 TRUE，否则为 FALSE。
is_scrollable	Boolean	如果该游标可以滚动（就是允许以不连续的方式检索）则为 TRUE，否则为 FALSE。
creation_time	timestamp with time	声明该游标的时间戳。

名称	类型	描述
	zone	

5.4.46.PG_EXT_STATS

PG_EXT_STATS 视图提供对存储在 PG_STATISTIC_EXT 表（参考：系统表和系统视图->系统表->PG_STATISTIC_EXT 章节）里面的扩展统计信息的访问。扩展统计信息目前包括多列统计信息。

表 1 PG_EXT_STATS 字段

名称	类型	引用	描述
schemaname	name	PG_NAMESPACE.nspname	包含表的模式名。
tablename	name	PG_CLASS.relname	表名。
attname	int2vector	PG_STATISTIC_EXT.stakey	统计信息扩展的多列信息。
inherited	Boolean	-	如果为真，则包含继承的子列，否则只是指定表的字段。
null_frac	real	-	记录中字段组合为空的百分比。
avg_width	integer	-	字段组合记录以字节记的平均宽度。
n_distinct	real	-	如果大于零，表示字段组合中独立数值的估计数目。 如果小于零，表示独立数值的数目被行数除的负数。 用负数形式是因为 ANALYZE 认为独立数值的数目是随着表增长而增长； 正数的形式用于在字段看上去好像有固定的可能值数目的情况下。比如，-1 表示一个字段组合中独立数值的个数和行数相同。 如果等于零，表示独立数值的数目未知。

名称	类型	引用	描述
n_dndistinct	real	-	标识 dn1 上字段组合中非 NULL 数据的唯一值的数目。如果大于零，表示独立数值的实际数目。 如果小于零，表示独立数值的数目被行数除的负数。（比如，一个字段组合的数值平均出现概率为两次，则可以表示为 n_dndistinct=-0.5）。 如果等于零，表示独立数值的数目未知。
most_common_vals	anyarray	-	一个字段组合里最常用数值的列表。如果该字段组合不存在最常用数值，则为 NULL。本列保存的多列常用数值均不为 NULL。
most_common_freqs	real[]	-	一个最常用数值组合的频率的列表，也就是说，每个出现的次数除以行数。如果 most_common_vals 是 NULL，则为 NULL。
most_common_vals_null	anyarray	-	一个字段组合里最常用数值的列表。如果该字段组合不存在最常用数值，则为 NULL。本列保存的多列常用数值中至少有一个值为 NULL。
most_common_freqs_null	real[]	-	一个最常用数值组合的频率的列表，也就是说，每个出现的次数除以行数。如果 most_common_vals_null 是 NULL，则为 NULL。
histogram_bounds	anyarray	-	直方图的边界值列表。

5.4.47.PG_GET_INVALID_BACKENDS

PG_GET_INVALID_BACKENDS 视图提供显示数据库主节点上连接到当前备机的后台线程信息。

表 1 PG_GET_INVALID_BACKENDS 字段

名称	类型	描述
pid	bigint	线程 ID。
node_name	text	后台线程中连接的节点信息。
dbname	name	当前连接的数据库。
backend_start	timestamp with time zone	后台线程启动的时间。
query	text	后台线程正在执行的查询语句。

5.4.48.PG_GET_SENDERS_CATCHUP_TIME

PG_GET_SENDERS_CATCHUP_TIME 视图显示数据库节点上当前活跃的主备发送线程的追赶信息。

表 1 PG_GET_SENDERS_CATCHUP_TIME 字段

名称	类型	描述
pid	bigint	当前 sender 的线程 ID。
lwpid	integer	当前 sender 的 lwpid。
local_role	text	本地的角色。
peer_role	text	对端的角色。
state	text	当前 sender 的复制状态。
type	text	当前 sender 的类型。
catchup_start	timestamp with time zone	catchup 启动的时间。
catchup_end	timestamp with time zone	catchup 结束的时间。

5.4.49.PG_GROUP

PG_GROUP 视图查看数据库认证角色及角色之间的成员关系。

表 1 PG_GROUP 字段

名称	类型	描述

名称	类型	描述
groname	name	组的名称。
grosysid	oid	组的 ID。
grolist	oid[]	一个数组，包含这个组里面所有角色的 ID。

5.4.50.PG_GTT_ATTACHED_PIDS

PG_GTT_ATTACHED_PIDS 视图查看哪些会话正在使用全局临时表，调用 pg_get_attached_pid 函数。

表 1 PG_GTT_ATTACHED_PIDS 字段

名称	类型	描述
schemaname	name	schema 名称。
tablename	name	全局临时表名称。
relid	oid	全局临时表的 oid。
pids	bigint[]	线程 pid 列表。

5.4.51.PG_GTT_RELSTATS

PG_GTT_RELSTATS 视图查看当前会话所有全局临时表基本信息，调用 pg_get_gtt_relstats 函数。

表 1 PG_GTT_RELSTATS 字段

名称	类型	描述
schemaname	name	schema 名称。
tablename	name	全局临时表名称。
relfilenode	oid	文件对象的 ID。
relpages	integer	全局临时表的磁盘页面数。
reltuples	real	全局临时表的记录数。
relallvisible	integer	被标识为全可见的页面数。
relfrozenxid	xid	该表中所有在这个之前的事务 ID 已经被一个固定的（frozen）事务 ID 替换。
relminmxid	xid	预留接口，暂未启用。

5.4.52.PG_GTT_STATS

PG_GTT_STATS 视图查看当前会话所有全局临时表单列统计信息，调用 pg_get_gtt_statistics 函数。

表 1 PG_GTT_STATS 字段

名称	类型	描述
schemaname	name	schema 名称。
tablename	name	全局临时表名称。
attname	name	属性名称。
inherited	boolean	是否统计有继承关系的对象。
null_frac	real	该字段中为 NULL 的记录的比率。
avg_width	integer	非 NULL 记录的平均存储宽度，以字节计算。
n_distinct	real	标识全局统计信息中字段里唯一的非 NULL 数据值的数目。
most_common_vals	text[]	高频值列表，按照出现的频率排序。
most_common_freqs	real[]	高频值的频率。
histogram_bounds	text[]	等频直方图描述列中的数据分布（不包含高频值）。
correlation	real	相关系数。
most_common_elems	text[]	类型高频值列表，用于数组类型或一些其他类型。
most_common_elem_freqs	real[]	类型高频值的频率。
elem_count_histogram	real[]	数组类型直方图。

5.4.53.PG_INDEXES

PG_INDEXES 视图提供对数据库中每个索引的有用信息的访问。

表 1 PG_INDEXES 字段

名称	类型	引用	描述
schemaname	name	PG_NAMESPACE.nspname	包含表和索引的模式名称。
tablename	name	PG_CLASS.relname	此索引所服务的表的名称。
indexname	name	PG_CLASS.relname	索引的名称。

名称	类型	引用	描述
tablespace	name	PG_TABLESPACE.nspname	包含索引的表空间名称。
indexdef	text	-	索引定义（一个重建的 CREATE INDEX 命令）。

5.4.54.PG_LOCKS

PG_LOCKS 视图存储各打开事务所持有的锁信息。

表 1 PG_LOCKS 字段

名称	类型	引用	描述
locktype	text	-	被锁定对象的类型：relation、extend、page、tuple、transactionid、virtualxid、object、userlock、advisory。
database	oid	PG_DATABASE.oid	被锁定对象所在数据库的 OID。 ❖ 如果被锁定的对象是共享对象，则 OID 为 0。 ❖ 如果是一个事务 ID，则为 NULL。
relation	oid	PG_CLASS.oid	关系的 OID，如果锁定的对象不是关系，也不是关系的一部分，则为 NULL。
page	integer	-	关系内部的页面编号，如果对象不是关系页或者不是行页，则为 NULL。
tuple	smallint	-	页面里边的行编号，如果对象不是行，则为 NULL。
bucket	integer	-	子表对应的 bucket number。如果目标不是表的话，则为 NULL。
virtualxid	text	-	事务的虚拟 ID，如果对象不是一个虚拟事务 ID，则为 NULL。
transactionid	xid	-	事务的 ID，如果对象不是一个事务 ID，则为 NULL。

名称	类型	引用	描述
classid	oid	PG_CLASS.oid	包含该对象的系统表的 OID，如果对象不是普通的数据库对象，则为 NULL。
objid	oid	-	对象在其系统表内的 OID，如果对象不是普通数据库对象，则为 NULL。
objsubid	smallint	-	❖ 对于表的一个字段，这是字段编号； ❖ 对于其他对象类型，这个字段是 0； ❖ 如果这个对象不是普通数据库对象，则为 NULL。
virtualtransaction	text	-	持有此锁或者在等待此锁的事务的虚拟 ID。
pid	bigint	-	持有或者等待这个锁的服务器线程的逻辑 ID。如果锁是被一个预备事务持有的，则为 NULL。
sessionid	bigint	-	持有或者等待这个锁的会话 ID。
mode	text	-	这个线程持有的或者是期望的锁模式。
granted	Boolean	-	❖ 如果锁是持有锁，则为 TRUE。 ❖ 如果锁是等待锁，则为 FALSE。
fastpath	Boolean	-	❖ 如果通过 fast-path 获得锁，则为 TRUE； ❖ 如果通过主要的锁表获得，则为 FALSE。
locktag	text	-	会话等待锁信息，可通过 locktag_decode()函数解析。
global_sessionid	text		全局会话 ID。

5.4.55.PG_NODE_ENV

PG_NODE_ENV 视图提供获取当前节点的环境变量信息。

表 1 PG_NODE_ENV 字段

名称	类型	描述
node_name	text	当前节点的名称。
host	text	当前节点的主机名称。
process	integer	当前节点的进程号。
port	integer	当前节点的端口号。
installpath	text	当前节点的安装目录。
datapath	text	当前节点的数据目录。
log_directory	text	当前节点的日志目录。

5.4.56.PG_OS_THREADS

PG_OS_THREADS 视图提供当前节点下所有线程的状态信息。

表 1 PG_OS_THREADS 字段

名称	类型	描述
node_name	text	当前节点的名称。
pid	bigint	当前节点进程中正在运行的线程号。
lwpid	integer	与 pid 对应的轻量级线程号。
thread_name	text	与 pid 对应的线程名称。
creation_time	timestamp with time zone	与 pid 对应的线程创建的时间。

5.4.57.PG_PREPARED_STATEMENTS

PG_PREPARED_STATEMENTS 视图显示当前会话所有可用的预备语句。

表 1 PG_PREPARED_STATEMENTS 字段

名称	类型	描述

名称	类型	描述
name	text	预备语句的标识符。
statement	text	创建该预备语句的查询字符串。对于从 SQL 创建的预备语句而言是客户端提交的 PREPARE 语句；对于通过前/后端协议创建的预备语句而言是预备语句自身的文本。
prepare_time	timestamp with time zone	创建该预备语句的时间戳。
parameter_types	regtype[]	该预备语句期望的参数类型，以 regtype 类型的数组格式出现。与该数组元素相对应的 OID 可以通过把 regtype 转换为 oid 值得到。
from_sql	Boolean	❖ 如果该预备语句是通过 PREPARE 语句创建的则为 true。 ❖ 如果是通过前/后端协议创建的则为 false。

5.4.58.PG_PREPARED_XACTS

PG_PREPARED_XACTS 视图显示当前准备好进行两阶段提交的事务的信息。

表 1 PG_PREPARED_XACTS 字段

名称	类型	引用	描述
transaction	xid	-	预备事务的数字事务标识。
gid	text	-	赋予该事务的全局事务标识。
prepared	timestamp with time zone	-	事务准备好提交的时间。
owner	name	PG_AUTHID.rolname	执行该事务的用户的名称。
database	name	PG_DATABASE.datname	执行该事务所在的数据库名。

5.4.59.PG_PUBLICATION_TABLES

视图 PG_PUBLICATION_TABLES 提供 publication 与其所发布的表之间的映射信息。和底层的系统表 pg_publication_rel 不同，这个视图展开了定义为 FOR ALL TABLES 的 publication，这样对这类 publication 来说，每一个合格的表都有一行。

表 1 PG_PUBLICATION_TABLES 字段

名称	类型	描述
pubname	name	发布的名称。
schemaname	name	包含表的模式名称。
tablename	name	表的名称。

5.4.60.PG_REPLICATION_ORIGIN_STATUS

获取复制源的复制状态。

表 1 PG_REPLICATION_ORIGIN_STATUS 字段

名称	类型	描述
local_id	oid	复制源 ID。
external_id	text	复制源名称。
remote_lsn	LSN	复制源的 lsn 位置。
local_lsn	LSN	本地的 lsn 位置。

5.4.61.PG_REPLICATION_SLOTS

PG_REPLICATION_SLOTS 视图查看复制节点的信息。

表 1 PG_REPLICATION_SLOTS 字段

名称	类型	描述
slot_name	text	复制节点的名称。

名称	类型	描述
plugin	text	逻辑复制槽对应的输出插件名。
slot_type	text	复制节点的类型。
datoid	oid	复制节点的数据库 OID。
database	name	复制节点的数据库名称。
active	Boolean	复制节点是否为激活状态。
xmin	xid	复制节点事务标识。
catalog_xmin	xid	逻辑复制槽对应的最早解码事务标识。
restart_lsn	text	复制节点的 Xlog 文件信息。
dummy_standby	Boolean	复制节点是否为假备。

5.4.62.PG_RLSPOLICIES

PG_RLSPOLICIES 视图提供查询行级访问控制策略。

表 1 PG_RLSPOLICIES 字段

名称	类型	描述
schemaname	name	行级访问控制策略作用的表对象所属模式名称。
tablename	name	行级访问控制策略作用的表对象名称。
policyname	name	行级访问控制策略名称。
policypermissive	text	行级访问控制策略属性。
policyroles	name[]	行级访问控制策略影响的用户列表，不指定表示影响所有的用户。
policycmd	text	行级访问控制策略影响的 SQL 操作。
policyqual	text	行级访问控制策略的表达式。

5.4.63.PG_ROLES

PG_ROLES 视图提供访问数据库角色的相关信息，初始化用户和具有 sysadmin 属性或 createrole 属性的用户可以查看全部角色的信息，其他用户只能查看自己的信息。

表 1 PG_ROLES 字段

名称	类型	引用	描述

名称	类型	引用	描述
rolname	name	-	角色名称。
rolsuper	Boolean	-	该角色是否是拥有最高权限的初始系统管理员。
rolinherit	Boolean	-	该角色是否继承角色的权限。
rolcreaterole	Boolean	-	该角色是否可以创建其他的角色。
rolcreatedb	Boolean	-	该角色是否可以创建数据库。
rolcatupdate	Boolean	-	该角色是否可以直接更新系统表。只有 usesysid=10 的初始系统管理员拥有此权限。其他用户无法获得此权限。
rolcanlogin	Boolean	-	该角色是否可以登录数据库。
rolreplication	Boolean	-	该角色是否可以复制。
rolauditadmin	Boolean	-	该角色是否为审计管理员。
rolsystemadmin	Boolean	-	该角色是否为系统管理员。
rolconnlimit	integer	-	对于可以登录的角色，这里限制了该角色允许发起的最大并发连接数。-1 表示无限制。
rolpassword	text	-	不是口令，总是****。
rolvalidbegin	timestamp with time zone	-	帐户的有效开始时间；如果没有设置有效开始时间，则为 NULL。
rolvaliduntil	timestamp with time zone	-	帐户的有效结束时间；如果没有设置有效结束时间，则为 NULL。
rolrespool	name	-	用户所能够使用的 resource pool。
rolparentid	oid	PG_AUTHID.rolparentid	用户所在组用户的 OID。
roltabspace	text	-	用户永久表存储空间限额。
roltempspace	text	-	用户临时表存储空间限额，单位为 KB。
rolspillspace	text	-	用户算子落盘空间限额，单位为 KB。

名称	类型	引用	描述
rolconfig	text[]	-	运行时配置变量的会话缺省。
oid	oid	PG_AUTHID.oid	角色的 ID。
roluseft	Boolean	PG_AUTHID.roluseft	角色是否可以操作外表。
rolkind	“char”	-	角色类型。
nodegroup	name	-	该字段不支持。
rolmonitoradmin	Boolean	-	该角色是否为监控管理员。
roloperatoradmin	Boolean	-	该角色是否为运维管理员。
rolpolicyadmin	Boolean	-	该角色是否为安全策略管理员。
rolssoadmin	Boolean	-	该角色是否为安全管理员。

5.4.64.PG_RULES

PG_RULES 视图提供对查询重写规则的有用信息访问的接口。

表 1 PG_RULES 字段

名称	类型	描述
schemaname	name	包含表的模式的名称。
tablename	name	规则作用的表的名称。
rulename	name	规则的名称。
definition	text	规则定义（一个重新构造的创建命令）。

5.4.65.PG_RUNNING_XACTS

PG_RUNNING_XACTS 视图主要功能是显示当前节点运行事务的信息。

表 1 PG_RUNNING_XACTS 字段

名称	类型	描述
handle	integer	事务对应的事务管理器中的槽位句柄，该值恒为-1。
gxid	xid	事务 id 号。
state	tinyint	事务状态（3：prepared 或者 0：starting）。

名称	类型	描述
node	text	节点名称。
xmin	xid	节点上当前数据涉及的最小事务号 xmin。
vacuum	Boolean	标志当前事务是否是 lazy vacuum 事务。 t（true）：表示是。 f（false）：表示否。
timeline	bigint	标志数据库重启次数。
prepare_xid	xid	处于 prepared 状态的事务的 id 号，若不在 prepared 状态，值为 0。
pid	bigint	事务对应的线程 id。
next_xid	xid	其余节点发送给当前节点的事务 id，该值恒为 0。

5.4.66.PG_SECLABELS

PG_SECLABELS 视图提供关于安全标签的信息。

表 1 PG_SECLABELS 字段

名称	类型	引用	描述
objoid	oid	任意 OID 属性	这个安全标签指向的对象的 OID。
classoid	oid	PG_CLASS.oid	这个对象出现的系统表的 OID。
objsubid	integer	-	对于一个在表字段上的安全标签，是字段编号（引用表本身的 objoid 和 classoid）。对于所有其他对象类型，这个字段为 0。
objtype	text	-	这个标签出现的对象的类型，文本格式。
objnamespace	oid	PG_NAMESPACE.oid	这个对象的名称空间的 OID，如果适用；否则为 NULL。
objname	text	-	这个标签适用的对象的名称，文本格式。
provider	text	PG_SECLABEL.provider	与这个标签相关的标签提供者。
label	text	PG_SECLABEL.label	适用于这个对象的安全标签。

5.4.67.PG_SESSION_IOSTAT

PG_SESSION_IOSTAT 视图显示当前用户执行作业正在运行时的 IO 负载管理相关信息。查询该视图需要 sysadmin 权限或者 monitor admin 权限。

以下涉及到 iops，对于行存，均以万次/s 为单位，对于列存，均以次/s 为单位。

表 1 PG_SESSION_IOSTAT 字段

名称	类型	描述
query_id	bigint	作业 id。
mincurriops	integer	该作业当前 io 在数据库实例中的最小值。
maxcurriops	integer	该作业当前 io 在数据库实例中的最大值。
minpeakiops	integer	在作业运行时，作业 io 峰值中，数据库实例的最小值。
maxpeakiops	integer	在作业运行时，作业 io 峰值中，数据库实例的最大值。
io_limits	integer	该作业所设 GUC 参数 io_limits。
io_priority	text	该作业所设 GUC 参数 io_priority。
query	text	作业。
node_group	text	该字段不支持。
curr_io_limits	integer	使用 io_priority 管控 io 时的实时 io_limits 值。

5.4.68.PG_SESSION_WLMSTAT

PG_SESSION_WLMSTAT 视图显示当前用户执行作业正在运行时的负载管理相关信息。查询该视图需要 sysadmin 权限。

表 1 PG_SESSION_WLMSTAT 字段

名称	类型	描述
datid	oid	连接后端的数据库 OID。
datname	name	连接后端的数据库名称。
threadid	bigint	后端线程 ID。
sessionid	bigint	会话 ID。

名称	类型	描述
processid	integer	后端线程的 pid。
usesysid	oid	登录后端的用户 OID。
appname	text	连接到后端的应用名。
usename	name	登录到该后端的用户名。
priority	bigint	语句所在 Cgroups 的优先级。
attribute	text	语句的属性： ❖ Ordinary：语句发送到数据库后被解析前的默认属性。 ❖ Simple：简单语句。 ❖ Complicated：复杂语句。 ❖ Internal：数据库内部语句。Unknown：未知。
block_time	bigint	语句当前为止的 pending 的时间，单位 s。
elapsed_time	bigint	语句当前为止的实际执行时间，单位 s。
total_cpu_time	bigint	语句在上一时间周期内的数据库实例上 CPU 使用的总时间，单位 s。
cpu_skew_percent	integer	语句在上一时间周期内的数据库实例上 CPU 使用的倾斜率。
statement_mem	integer	语句执行使用的 statement_mem，预留字段。
active_points	integer	语句占用的资源池并发点数。
dop_value	integer	语句的从资源池中获取的 dop 值。
control_group	text	该字段不支持。
status	text	语句当前的状态，包括： ❖ pending：执行前状态。 ❖ running：执行进行状态。 ❖ finished：执行正常结束。（当 enqueue 字段为 StoredProc 或 Transaction 时，仅代表语句中的部分作业已经执行完毕，该状态会持续到该语句完全执行完毕。）aborted：执行异常终止。 ❖ active：非以上四种状态外的正常状态。 ❖ unknown：未知状态。
enqueue	text	该字段不支持。
resource_pool	name	语句当前所在的资源池。

名称	类型	描述
query	text	该后端的最新查询。如果 state 状态是 active（活的），此字段显示当前正在执行的查询。所有其他情况表示上一个查询。
is_plana	boolean	该字段不支持。
node_group	text	该字段不支持。

5.4.69.PG_SETTINGS

PG_SETTINGS 视图显示数据库运行时参数的相关信息。

表 1 PG_SETTINGS 字段

名称	类型	描述
name	text	参数名称。
setting	text	参数当前值。
unit	text	参数的隐式结构。
category	text	参数的逻辑组。
short_desc	text	参数的简单描述。
extra_desc	text	参数的详细描述。
context	text	设置参数值的上下文，包括 internal、postmaster、sighup、backend、superuser、user。
vartype	text	参数类型，包括 bool、enum、integer、real、string。
source	text	参数的赋值方式。
min_val	text	参数最小值。如果参数类型不是数值型，那么该字段值为 null。
max_val	text	参数最大值。如果参数类型不是数值型，那么该字段值为 null。
enumvals	text[]	enum 类型参数合法值。如果参数类型不是 enum 型，那么该字段值为 null。
boot_val	text	数据库启动时参数默认值。
reset_val	text	数据库重置时参数默认值。
sourcefile	text	设置参数值的配置文件。如果参数不是通过配置文件赋值，那么该字段值为 null。

名称	类型	描述
sourceline	integer	设置参数值的配置文件的行号。如果参数不是通过配置文件赋值，那么该字段值为 null。

5.4.70.PG_SHADOW

PG_SHADOW 视图显示了所有在 PG_AUTHID 中标记了 rolcanlogin 的角色的属性。

这个视图的名称来自于该视图是不可读的，因为它包含口令。PG_USER（参考：系统表和系统视图->系统视图->PG_USER 章节）是一个在 PG_SHADOW 上公开可读的视图，只是把口令域填成了空白。

表 1 PG_SHADOW 字段

名称	类型	引用	描述
usename	name	.rolname	用户名。
usesysid	oid	.oid	用户的 ID。
usecreatedb	Boolean	-	用户可以创建数据库。
usesuper	Boolean	-	用户是系统管理员。
usecatupd	Boolean	-	用户可以更新视图。即使是系统管理员，如果这个字段不是真，也不能更新视图。
userepl	Boolean	-	用户可以初始化流复制和使系统处于或不处于备份模式。
passwd	text	-	获取加密的口令是如何存储的信息。
valbegin	timestamp with time zone	-	帐户的有效开始时间；如果没有设置有效开始时间，则为 NULL。
valuntil	timestamp with time zone	-	帐户的有效结束时间；如果没有设置有效结束时间，则为 NULL。

名称	类型	引用	描述
respool	name	-	用户使用的资源池。
parent	oid	-	父资源池。
spacelimit	text	-	永久表存储空间限额。
tempspacelimit	text	-	临时表存储空间限额。
spillspacelimit	text	-	算子落盘空间限额。
usemonitoradmin	boolean	-	用户是监控管理员。
useoperatoradmin	boolean	-	用户是运维管理员。
usepolicyadmin	boolean	-	用户是安全策略管理员。
useconfig	text[]	-	运行时配置变量的会话默认值。

5.4.71.PG_STAT_ACTIVITY

PG_STAT_ACTIVITY 视图显示和当前用户查询相关的信息，字段保存的是上一次执行的信息。

表 1 PG_STAT_ACTIVITY 字段

名称	类型	描述
datid	oid	用户会话在后台连接到的数据库 OID。
datname	name	用户会话在后台连接到的数据库名称。
pid	bigint	后台线程 ID。
sessionid	bigint	会话 ID。
usesysid	oid	登录该后台的用户 OID。
username	name	登录该后台的用户名。
application_name	text	连接到该后台的应用名。

名称	类型	描述
client_addr	inet	连接到该后台的客户端的 IP 地址。 如果此字段是 null，它表明通过服务器机器上 UNIX 套接字连接客户端或者这是内部进程，如 autovacuum。
client_hostname	text	客户端的主机名，这个字段是通过 client_addr 的反向 DNS 查找得到。这个字段只有在启动 log_hostname 且使用 IP 连接时才非空。
client_port	integer	客户端用于与后台通讯的 TCP 端口号，如果使用 Unix 套接字，则为-1。
backend_start	timestamp with time zone	该过程开始的时间，即当客户端连接服务器时。
xact_start	timestamp with time zone	启动当前事务的时间，如果没有事务是活跃的，则为 null。如果当前查询是首个事务，则这列等同于 query_start 列。
query_start	timestamp with time zone	开始当前活跃查询的时间， 如果 state 的值不是 active，则这个值是上一个查询的开始时间。
state_change	timestamp with time zone	上次状态改变的时间。
waiting	boolean	如果后台当前正等待锁则为 true。
enqueue	text	该字段不支持。
state	text	该后台当前总体状态。可能值是： ❖ active：后台正在执行一个查询。 ❖ idle：后台正在等待一个新的客户端命令。 ❖ idle in transaction：后台在事务中，但事务中没有语句在执行。 ❖ idle in transaction (aborted)：后台在事务中，但事务中有语句执行失败。 ❖ fastpath function call：后台正在执行一个 fast-path 函数。 ❖ disabled：如果后台禁用 track_activities，则报告这个状态。

名称	类型	描述
		说明：普通用户只能查看到自己帐户所对应的会话状态。即其他帐户的 state 信息为空。例如以 judy 用户连接数据库后，在 pg_stat_activity 中查看到的普通用户 joe 及初始用户 panweidb 的 state 信息为空： SELECT datname, usename, usesysid, state,pid FROM pg_stat_activity; datname | usename | usesysid | state | pid −−−−+−−−+−−−−+−−−+−−−−−− postgres | panweidb | 10 | | 139968752121616 postgres | panweidb | 10 | | 139968903116560 db_tpcc | judy | 16398 | active | 139968391403280 postgres | panweidb | 10 | | 139968643069712 postgres | panweidb | 10 | | 139968680818448 postgres | joe | 16390 | | 139968563377936 (6 rows)`
resource_pool	name	用户使用的资源池。
query_id	bigint	查询语句的 ID。
query	text	该后台的最新查询。如果 state 状态是 active（活跃的），此字段显示当前正在执行的查询。所有其他情况表示上一个查询。
connection_info	text	json 格式字符串，记录当前连接数据库的驱动类型、驱动版本号、当前驱动的部署路径、进程属主用户等信息。
unique_sql_id	bigint	语句的 unique sql id。
trace_id	text	驱动传入的 trace id，与应用的一次请求相关联。

5.4.72.PG_STAT_ACTIVITY_NG

PG_STAT_ACTIVITY_NG 视图显示在当前用户所属的逻辑数据库实例下，所有查询的相关信息。

表 1 PG_STAT_ACTIVITY_NG 字段

名称	类型	描述

名称	类型	描述
datid	oid	用户会话在后台连接到的数据库 OID。
datname	name	用户会话在后台连接到的数据库名称。
pid	bigint	后台线程 ID。
sessionid	bigint	会话 ID。
usesysid	oid	登录该后台的用户 OID。
usename	name	登录该后台的用户名。
application_name	text	连接到该后台的应用名。
client_addr	inet	连接到该后台的客户端的 IP 地址。如果此字段是 null，它表明通过服务器机器上 UNIX 套接字连接客户端或者这是内部进程，如 autovacuum。
client_hostname	text	客户端的主机名，这个字段是通过 client_addr 的反向 DNS 查找得到。这个字段只有在启动 log_hostname 且使用 IP 连接时才非空。
client_port	integer	客户端用于与后台通讯的 TCP 端口号，如果使用 Unix 套接字，则为-1。
backend_start	timestamp with time zone	该过程开始的时间，即当客户端连接服务器时。
xact_start	timestamp with time zone	启动当前事务的时间，如果没有事务是活跃的，则为 null。如果当前查询是首个事务，则这列等同于 query_start 列。
query_start	timestamp with time zone	开始当前活跃查询的时间，如果 state 的值不是 active，则这个值是上一个查询的开始时间。
state_change	timestamp with time zone	上次状态改变的时间。
waiting	boolean	如果后台当前正等待锁则为 true。否则为 false。
enqueue	text	语句当前排队状态。可能值是： ❖ waiting in queue：表示语句在排队中。 ❖ 空：表示语句正在运行。
state	text	该后台当前总体状态。可能值是： ❖ active：后台正在执行一个查询。 ❖ idle：后台正在等待一个新的客户端命令。

名称	类型	描述
		❖ idle in transaction：后台在事务中，但事务中没有语句在执行。 ❖ idle in transaction (aborted)：后台在事务中，但事务中有语句执行失败。 ❖ fastpath function call：后台正在执行一个 fast-path 函数。 disabled：如果后台禁用 track_activities，则报告这个状态。说明：普通用户只能查看到自己帐户所对应的会话状态。即其他帐户的 state 信息为空。例如以 judy 用户连接数据库后，在 pg_stat_activity 中查看到的普通用户 joe 及初始用户 panweidb 的 state 信息为空： SELECT datname, usename, usesysid, state,pid FROM pg_stat_activity_ng; datname | usename | usesysid | state | pid −−−−+−−−+−−−−+−−−+−−−−− postgres | panweidb | 10 | | 139968752121616 postgres | panweidb | 10 | | 139968903116560 db_tpcds | judy | 16398 | active | 139968391403280 postgres | panweidb | 10 | | 139968643069712 postgres | panweidb | 10 | | 139968680818448 postgres | joe | 16390 | | 139968563377936 (6 rows)
resource_pool	name	用户使用的资源池。
query_id	bigint	查询语句的 ID。
query	text	该后台的最新查询。如果 state 状态是 active（活跃的），此字段显示当前正在执行的查询。所有其他情况表示上一个查询。
node_group	text	语句所属用户对应的逻辑数据库实例。

5.4.73.PG_STAT_ALL_INDEXES

PG_STAT_ALL_INDEXES 视图将包含当前数据库中的每个索引行，显示访问特定索引的统计。

索引可以通过简单的索引扫描或“位图”索引扫描进行使用。位图扫描中几个索引的输出可以通过 AND 或者 OR 规则进行组合， 因此当使用位图扫描的时候，很难将独立堆行抓取与特定索引进行组合，因此，一个位图扫描增加 pg_stat_all_indexes.idx_tup_read 使用索引计数，并且增加 pg_stat_all_tables.idx_tup_fetch 表计数，但不影响 pg_stat_all_indexes.idx_tup_fetch。

表 1 PG_STAT_ALL_INDEXES 字段

名称	类型	描述
relid	oid	这个索引的表的 OID。
indexrelid	oid	索引的 OID。
schemaname	name	索引的模式名。
relname	name	索引的表名。
indexrelname	name	索引名。
idx_scan	bigint	索引上开始的索引扫描数。
idx_tup_read	bigint	通过索引上扫描返回的索引项数。
idx_tup_fetch	bigint	通过使用索引的简单索引扫描抓取的活表行数。

5.4.74.PG_STAT_ALL_TABLES

PG_STAT_ALL_TABLES 视图将包含当前数据库中每个表的一行（包括 TOAST 表），显示访问特定表的统计信息。

表 1 PG_STAT_ALL_TABLES 字段

名称	类型	描述
relid	oid	表的 OID。
schemaname	name	该表的模式名。
relname	name	表名。

名称	类型	描述
seq_scan	bigint	该表发起的顺序扫描数。
seq_tup_read	bigint	顺序扫描抓取的活跃行数。
idx_scan	bigint	该表发起的索引扫描数。
idx_tup_fetch	bigint	索引扫描抓取的活跃行数。
n_tup_ins	bigint	插入行数。
n_tup_upd	bigint	更新行数。
n_tup_del	bigint	删除行数。
n_tup_hot_upd	bigint	HOT 更新行数（比如没有更新所需的单独索引）。
n_live_tup	bigint	估计活跃行数。
n_dead_tup	bigint	估计死行数。
last_vacuum	timestamp with time zone	最后一次清理该表的时间。
last_autovacuum	timestamp with time zone	上次被 autovacuum 守护进程清理该表的时间。
last_analyze	timestamp with time zone	上次分析该表的时间。
last_autoanalyze	timestamp with time zone	上次被 autovacuum 守护进程分析该表的时间。
vacuum_count	bigint	这个表被清理的次数。
autovacuum_count	bigint	这个表被 autovacuum 清理的次数。

名称	类型	描述
analyze_count	bigint	这个表被分析的次数。
autoanalyze_count	bigint	这个表被 autovacuum 守护进程分析的次数。
last_data_changed	timestamp with time zone	记录这个表上一次数据发生变化的时间（引起数据变化的操作包括 INSERT/UPDATE/DELETE、EXCHANGE/TRUNCATE/DROP partition），该列数据仅在本地数据库主节点记录。

5.4.75.PG_STAT_BAD_BLOCK

PG_STAT_BAD_BLOCK 视图显示自节点启动后，读取数据时出现 Page/CU 校验失败的统计信息。

表 1 PG_STAT_BAD_BLOCK 字段

名称	类型	描述
nodename	text	节点名。
databaseid	integer	数据库 OID。
tablespaceid	integer	表空间 OID。
relfilenode	integer	文件对象 ID。
bucketid	smallint	一致性 hash bucket ID。
forknum	integer	文件类型。取值如下： 0，数据主文件。 1，FSM 文件。 2，VM 文件。 3，BCM 文件。 大于 4，列存表每个字段的数据文件。

名称	类型	描述
error_count	integer	出现校验失败的次数。
first_time	timestamp with time zone	第一次出现时间。
last_time	timestamp with time zone	最近一次出现时间。

5.4.76.PG_STAT_BGWRITER

PG_STAT_BGWRITER 视图显示关于后端写进程活动的统计信息。

表 1 PG_STAT_BGWRITER 字段

名称	类型	描述
checkpoints_timed	bigint	执行的定期检查点数。
checkpoints_req	bigint	执行的需求检查点数。
checkpoint_write_time	double precision	花费在检查点处理部分的时间总量，其中文件被写入到磁盘，以毫秒为单位。
checkpoint_sync_time	double precision	花费在检查点处理部分的时间总量，其中文件被同步到磁盘，以毫秒为单位。
buffers_checkpoint	bigint	检查点写缓冲区数量。
buffers_clean	bigint	后端写进程写缓冲区数量。
maxwritten_clean	bigint	后端写进程停止清理扫描时间数，因为它写了太多缓冲区。
buffers_backend	bigint	通过后端直接写缓冲区数。
buffers_backend_fsync	bigint	后端不得不执行自己的 fsync 调用的时间数（通常后端写进程处理这些

名称	类型	描述
		即使后端确实自己写）。
buffers_alloc	bigint	分配的缓冲区数量。
stats_reset	timestamp with time zone	这些统计被重置的时间。

5.4.77.PG_STAT_DATABASE

PG_STAT_DATABASE 视图将包含 PanWeiDB 中每个数据库的数据库统计信息。

表 1 PG_STAT_DATABASE 字段

名称	类型	描述
datid	oid	数据库的 OID。
datname	name	这个数据库的名称。
numbackends	integer	当前连接到该数据库的后端数。这是在返回一个反映目前状态值的视图中唯一的列；自上次重置所有其他列返回累积值。
xact_commit	bigint	此数据库中已经提交的事务数。
xact_rollback	bigint	此数据库中已经回滚的事务数。
blks_read	bigint	在这个数据库中读取的磁盘块的数量。
blks_hit	bigint	高速缓存中已经发现的磁盘块的次数， 这样读取是不必要的（这只包括 PanWeiDB 缓冲区高速缓存，没有操作系统的文件系统缓存）。
tup_returned	bigint	通过数据库查询返回的行数。
tup_fetched	bigint	通过数据库查询抓取的行数。
tup_inserted	bigint	通过数据库查询插入的行数。

名称	类型	描述
tup_updated	bigint	通过数据库查询更新的行数。
tup_deleted	bigint	通过数据库查询删除的行数。
conflicts	bigint	由于数据库恢复冲突取消的查询数量（只在备用服务器发生的冲突）。请参见 PG_STAT_DATABASE_CONFLICTS 获取更多信息。
temp_files	bigint	通过数据库查询创建的临时文件数量。计算所有临时文件， 不论为什么创建临时文件（比如排序或者哈希）， 而且不管 log_temp_files 设置。
temp_bytes	bigint	通过数据库查询写入临时文件的数据总量。计算所有临时文件，不论为什么创建临时文件，而且不管 log_temp_files 设置。
deadlocks	bigint	在该数据库中检索的死锁数。
blk_read_time	double precision	通过数据库后端读取数据文件块花费的时间，以毫秒计算。
blk_write_time	double precision	通过数据库后端写入数据文件块花费的时间，以毫秒计算。
stats_reset	timestamp with time zone	重置当前状态统计的时间。

5.4.78.PG_STAT_DATABASE_CONFLICTS

PG_STAT_DATABASE_CONFLICTS 视图显示数据库冲突状态的统计信息。

表 1 PG_STAT_DATABASE_CONFLICTS 字段

名称	类型	描述
datid	oid	数据库标识。

名称	类型	描述
datname	name	数据库名称。
confl_tablespace	bigint	冲突的表空间的数目。
confl_lock	bigint	冲突的锁数目。
confl_snapshot	bigint	冲突的快照数目。
confl_bufferpin	bigint	冲突的缓冲区数目。
confl_deadlock	bigint	冲突的死锁数目。

5.4.79.PG_STAT_REPLICATION

PG_STAT_REPLICATION 视图用于描述日志同步状态信息，例如发起端发送日志位置，接收端接收日志位置等。

表 1 PG_STAT_REPLICATION 字段

名称	类型	描述
pid	bigint	线程的 PID。
usesysid	oid	用户系统 ID。
usename	name	用户名。
application_name	text	程序名称。
client_addr	inet	客户端地址。
client_hostname	text	客户端名。
client_port	integer	客户端端口。
backend_start	timestamp with time zone	程序启动时间。
state	text	日志复制的状态：

名称	类型	描述
		❖ 追赶状态。 ❖ 一致的流状态。
sender_sent_location	text	发送端发送日志位置。
receiver_write_location	text	接收端 write 日志位置。
receiver_flush_location	text	接收端 flush 日志位置。
receiver_replay_location	text	接收端 replay 日志位置。
sync_priority	integer	同步复制的优先级（0 表示异步）。
sync_state	text	同步状态： ❖ sync：表示此备库作为同步备库，对日志进行同步复制。 ❖ async：表示此备库状态为异步备库，对日志进行异步复制。 ❖ potential：表示此备库为潜在同步备库，当前对日志进行异步复制。若当前同步备库中的一个出现故障后，则此潜在同步备库很有可能变为同步备库。 ❖ quorum：在同步与异步之间切换，保证备机中有大于一定数量的同步备机，同步备机数量一般为(n+1)/2-1，n 为总副本个数。是否为同步备机取决于是否先接到了日志。详情请参考《开发者指南》->GUC 参数说明->双机复制->主服务器->synchronous_standby_names 参数描述。

5.4.80.PG_STAT_SUBSCRIPTION

获取订阅的详细同步信息。

表 1 PG_STAT_SUBSCRIPTION 字段

名称	类型	描述
subid	oid	订阅的 OID。
subname	name	订阅的名称。
pid	thread_id	后台 apply 线程的 thread id。
received_lsn	LSN	从发布端接收到的最近的 lsn。
last_msg_send_time	timestamp	最近发布端发送消息的时间。
last_msg_receipt_time	timestamp	最新订阅端收到消息的时间。
latest_end_lsn	LSN	最近一次收到保活消息时发布端的 lsn。
latest_end_time	timestamp	最近一次收到保活消息的时间。

5.4.81.PG_STAT_SYS_INDEXES

PG_STAT_SYS_INDEXES 视图显示 pg_catalog、information_schema 模式中所有系统表的索引状态信息。

表 1 PG_STAT_SYS_INDEXES 字段

名称	类型	描述
relid	oid	这个索引的表的 OID。
indexrelid	oid	索引的 OID。
schemaname	name	索引的模式名。
relname	name	索引的表名。
indexrelname	name	索引名。
idx_scan	bigint	索引上开始的索引扫描数。
idx_tup_read	bigint	通过索引上扫描返回的索引项数。

名称	类型	描述
idx_tup_fetch	bigint	通过使用索引的简单索引扫描抓取的活表行数。

5.4.82.PG_STAT_SYS_TABLES

PG_STAT_SYS_TABLES 视图显示 pg_catalog、information_schema 模式的所有命名空间中系统表的统计信息。

表 1 PG_STAT_SYS_TABLES 字段

名称	类型	描述
relid	oid	表的 OID。
schemaname	name	该表的模式名。
relname	name	表名。
seq_scan	bigint	该表发起的顺序扫描数。
seq_tup_read	bigint	顺序扫描抓取的活跃行数。
idx_scan	bigint	该表发起的索引扫描数。
idx_tup_fetch	bigint	索引扫描抓取的活跃行数。
n_tup_ins	bigint	插入行数。
n_tup_upd	bigint	更新行数。
n_tup_del	bigint	删除行数。
n_tup_hot_upd	bigint	HOT 更新行数（比如没有更新所需的单独索引）。
n_live_tup	bigint	估计活跃行数。
n_dead_tup	bigint	估计死行数。
last_vacuum	timestamp with time zone	最后一次该表是手动清理的时间（不计算 VACUUM FULL）。
last_autovacuum	timestamp with time	上次被 autovacuum 守护进程清理

名称	类型	描述
	zone	的时间。
last_analyze	timestamp with time zone	上次手动分析这个表的时间。
last_autoanalyze	timestamp with time zone	上次被 autovacuum 守护进程分析的时间。
vacuum_count	bigint	这个表被手动清理的次数（不计算 VACUUM FULL）。
autovacuum_count	bigint	这个表被 autovacuum 清理的次数。
analyze_count	bigint	这个表被手动分析的次数。
autoanalyze_count	bigint	这个表被 autovacuum 守护进程分析的次数。
last_data_changed	timestamp with time zone	这个表数据最近修改时间。

5.4.83.PG_STAT_USER_FUNCTIONS

PG_STAT_USER_FUNCTIONS 视图显示命名空间中用户自定义函数（函数语言为非内部语言）的状态信息。

表 1 PG_STAT_USER_FUNCTIONS 字段

名称	类型	描述
funcid	oid	函数标识。
schemaname	name	模式的名称。
funcname	name	函数名称。
calls	bigint	函数被调用的次数。

名称	类型	描述
total_time	double precision	函数的总执行时长。
self_time	double precision	当前线程调用函数的总的时长。

5.4.84.PG_STAT_USER_INDEXES

PG_STAT_USER_INDEXES 视图显示数据库中用户自定义普通表和 toast 表的索引状态信息。

表 1 PG_STAT_USER_INDEXES 字段

名称	类型	描述
relid	oid	这个索引的表的 OID。
indexrelid	oid	索引的 OID。
schemaname	name	索引的模式名。
relname	name	索引的表名。
indexrelname	name	索引名。
idx_scan	bigint	索引上开始的索引扫描数。
idx_tup_read	bigint	通过索引上扫描返回的索引项数。
idx_tup_fetch	bigint	通过使用索引的简单索引扫描抓取的活表行数。

5.4.85.PG_STAT_USER_TABLES

PG_STAT_USER_TABLES 视图显示所有命名空间中用户自定义普通表和 toast 表的状态信息。

表 1 PG_STAT_USER_TABLES 字段

名称	类型	描述
relid	oid	表的 OID。
schemaname	name	该表的模式名。
relname	name	表名。
seq_scan	bigint	该表发起的顺序扫描数。

名称	类型	描述
seq_tup_read	bigint	顺序扫描抓取的活跃行数。
idx_scan	bigint	该表发起的索引扫描数。
idx_tup_fetch	bigint	索引扫描抓取的活跃行数。
n_tup_ins	bigint	插入行数。
n_tup_upd	bigint	更新行数。
n_tup_del	bigint	删除行数。
n_tup_hot_upd	bigint	HOT 更新行数（即没有更新所需的单独索引）。
n_live_tup	bigint	估计活跃行数。
n_dead_tup	bigint	估计死行数。
last_vacuum	timestamp with time zone	最后一次该表是手动清理的（不计算 VACUUM FULL）。
last_autovacuum	timestamp with time zone	上次被 autovacuum 守护进程清理的表。
last_analyze	timestamp with time zone	上次手动分析这个表。
last_autoanalyze	timestamp with time zone	上次被 autovacuum 守护进程分析的表。
vacuum_count	bigint	这个表被手动清理的次数（不计算 VACUUM FULL）。
autovacuum_count	bigint	这个表被 autovacuum 清理的次数。
analyze_count	bigint	这个表被手动分析的次数。

名称	类型	描述
autoanalyze_count	bigint	这个表被 autovacuum 守护进程分析的次数。
last_data_changed	timestamp with time zone	这个表数据最近修改时间。

5.4.86.PG_STAT_XACT_ALL_TABLES

PG_STAT_XACT_ALL_TABLES 视图显示命名空间中所有普通表和 toast 表的事务状态信息。

表 1 PG_STAT_XACT_ALL_TABLES 字段

名称	类型	描述
relid	oid	表的 OID。
schemaname	name	该表的模式名。
relname	name	表名。
seq_scan	bigint	该表发起的顺序扫描数。
seq_tup_read	bigint	顺序扫描抓取的活跃行数。
idx_scan	bigint	该表发起的索引扫描数。
idx_tup_fetch	bigint	索引扫描抓取的活跃行数。
n_tup_ins	bigint	插入行数。
n_tup_upd	bigint	更新行数。
n_tup_del	bigint	删除行数。
n_tup_hot_upd	bigint	HOT 更新行数（比如没有更新所需的单独索引）。

5.4.87.PG_STAT_XACT_SYS_TABLES

PG_STAT_XACT_SYS_TABLES 视图显示命名空间中系统表的事务状态信息。

表 1 PG_STAT_XACT_SYS_TABLES 字段

名称	类型	描述
relid	oid	表的 OID。
schemaname	name	该表的模式名。
relname	name	表名。

名称	类型	描述
seq_scan	bigint	该表发起的顺序扫描数。
seq_tup_read	bigint	顺序扫描抓取的活跃行数。
idx_scan	bigint	该表发起的索引扫描数。
idx_tup_fetch	bigint	索引扫描抓取的活跃行数。
n_tup_ins	bigint	插入行数。
n_tup_upd	bigint	更新行数。
n_tup_del	bigint	删除行数。
n_tup_hot_upd	bigint	HOT 更新行数（比如没有更新所需的单独索引）。

5.4.88.PG_STAT_XACT_USER_FUNCTIONS

PG_STAT_XACT_USER_FUNCTIONS 视图包含每个函数的执行的统计信息。

表 1 PG_STAT_XACT_USER_FUNCTIONS 字段

名称	类型	描述
funcid	oid	函数标识。
schemaname	name	模式的名称。
funcname	name	函数名称。
calls	bigint	函数被调用的次数。
total_time	double precision	函数的总执行时长。
self_time	double precision	当前线程调用函数的总的时长。

5.4.89.PG_STAT_XACT_USER_TABLES

PG_STAT_XACT_USER_TABLES 视图显示命名空间中用户表的事务状态信息。

表 1 PG_STAT_XACT_USER_TABLES 字段

名称	类型	描述
relid	oid	表的 OID。
schemaname	name	该表的模式名。
relname	name	表名。

名称	类型	描述
seq_scan	bigint	该表发起的顺序扫描数。
seq_tup_read	bigint	顺序扫描抓取的活跃行数。
idx_scan	bigint	该表发起的索引扫描数。
idx_tup_fetch	bigint	索引扫描抓取的活跃行数。
n_tup_ins	bigint	插入行数。
n_tup_upd	bigint	更新行数。
n_tup_del	bigint	删除行数。
n_tup_hot_upd	bigint	HOT 更新行数（比如没有更新所需的单独索引）。

5.4.90.PG_STATIO_ALL_INDEXES

PG_STATIO_ALL_INDEXES 视图将包含当前数据库中的每个索引行，显示特定索引的 I/O 的统计。

表 1 PG_STATIO_ALL_INDEXES 字段

名称	类型	描述
relid	oid	索引的表的 OID。
indexrelid	oid	该索引的 OID。
schemaname	name	该索引的模式名。
relname	name	该索引的表名。
indexrelname	name	索引名称。
idx_blks_read	bigint	从索引中读取的磁盘块数。
idx_blks_hit	bigint	索引命中缓存数。

5.4.91.PG_STATIO_ALL_SEQUENCES

PG_STATIO_ALL_SEQUENCES 视图包含当前数据库中每个序列的 I/O 的统计信息。

表 1 PG_STATIO_ALL_SEQUENCES 字段

名称	类型	描述
relid	oid	序列 OID。
schemaname	name	序列中模式名。
relname	name	序列名。
blks_read	bigint	从序列中读取的磁盘块数。
blks_hit	bigint	序列中缓存命中数。

5.4.92.PG_STATIO_ALL_TABLES

PG_STATIO_ALL_TABLES 视图将包含当前数据库中每个表（包括 TOAST 表）的 I/O 统计信息。

表 1 PG_STATIO_ALL_TABLES 字段

名称	类型	描述
relid	oid	表 OID。
schemaname	name	该表模式名。
relname	name	表名。
heap_blks_read	bigint	从该表中读取的磁盘块数。
heap_blks_hit	bigint	该表缓存命中数。
idx_blks_read	bigint	从表中所有索引读取的磁盘块数。
idx_blks_hit	bigint	表中所有索引命中缓存数。
toast_blks_read	bigint	该表的 TOAST 表读取的磁盘块数（如果存在）。
toast_blks_hit	bigint	该表的 TOAST 表命中缓冲区数（如果存在）。
tidx_blks_read	bigint	该表的 TOAST 表索引读取的磁盘块数（如果存在）。
tidx_blks_hit	bigint	该表的 TOAST 表索引命中缓冲区数（如果存在）。

5.4.93.PG_STATIO_SYS_INDEXES

PG_STATIO_SYS_INDEXES 视图显示命名空间中所有系统表索引的 IO 状态信息。

表 1 PG_STATIO_SYS_INDEXES 字段

名称	类型	描述
relid	oid	索引的表的 OID。
indexrelid	oid	该索引的 OID。
schemaname	name	该索引的模式名。
relname	name	该索引的表名。
indexrelname	name	索引名称。
idx_blks_read	bigint	从索引中读取的磁盘块数。
idx_blks_hit	bigint	索引命中缓存数。

5.4.94.PG_STATIO_SYS_SEQUENCES

PG_STATIO_SYS_SEQUENCES 视图显示命名空间中所有序列的 IO 状态信息。

表 1 PG_STATIO_SYS_SEQUENCES 字段

名称	类型	描述
relid	oid	序列 OID。
schemaname	name	序列中模式名。
relname	name	序列名。
blks_read	bigint	从序列中读取的磁盘块数。
blks_hit	bigint	序列中缓存命中数。

5.4.95.PG_STATIO_SYS_TABLES

PG_STATIO_SYS_TABLES 视图显示命名空间中所有系统表的 IO 状态信息。

表 1 PG_STATIO_SYS_TABLES 字段

名称	类型	描述

名称	类型	描述
relid	oid	表 OID。
schemaname	name	该表模式名。
relname	name	表名。
heap_blks_read	bigint	从该表中读取的磁盘块数。
heap_blks_hit	bigint	该表缓存命中数。
idx_blks_read	bigint	从表中所有索引读取的磁盘块数。
idx_blks_hit	bigint	表中所有索引命中缓存数。
toast_blks_read	bigint	该表的 TOAST 表读取的磁盘块数（如果存在）。
toast_blks_hit	bigint	该表的 TOAST 表命中缓冲区数（如果存在）。
tidx_blks_read	bigint	该表的 TOAST 表索引读取的磁盘块数（如果存在）。
tidx_blks_hit	bigint	该表的 TOAST 表索引命中缓冲区数（如果存在）。

5.4.96.PG_STATIO_USER_INDEXES

PG_STATIO_USER_INDEXES 视图显示命名空间中所有用户关系表索引的 IO 状态信息。

表 1 PG_STATIO_USER_INDEXES 字段

名称	类型	描述
relid	oid	索引的表的 OID。
indexrelid	oid	该索引的 OID。
schemaname	name	该索引的模式名。

名称	类型	描述
relname	name	该索引的表名。
indexrelname	name	索引名称。
idx_blks_read	bigint	从索引中读取的磁盘块数。
idx_blks_hit	bigint	索引命中缓存数。

5.4.97.PG_STATIO_USER_SEQUENCES

PG_STATIO_USER_SEQUENCES 视图显示命名空间中所有用户关系表类型为序列的 IO 状态信息。

表 1 PG_STATIO_USER_SEQUENCES 字段

名称	类型	描述
relid	oid	序列 OID。
schemaname	name	序列中模式名。
relname	name	序列名。
blks_read	bigint	从序列中读取的磁盘块数。
blks_hit	bigint	序列中缓存命中数。

5.4.98.PG_STATIO_USER_TABLES

PG_STATIO_USER_TABLES 视图显示命名空间中所有用户关系表的 IO 状态信息。

表 1 PG_STATIO_USER_TABLES 字段

名称	类型	描述
relid	oid	表 OID。
schemaname	name	该表模式名。
relname	name	表名。
heap_blks_read	bigint	从该表中读取的磁盘块数。

名称	类型	描述
heap_blks_hit	bigint	该表缓存命中数。
idx_blks_read	bigint	从表中所有索引读取的磁盘块数。
idx_blks_hit	bigint	表中所有索引命中缓存数。
toast_blks_read	bigint	该表的 TOAST 表读取的磁盘块数（如果存在）。
toast_blks_hit	bigint	该表的 TOAST 表命中缓冲区数（如果存在）。
tidx_blks_read	bigint	该表的 TOAST 表索引读取的磁盘块数（如果存在）。
tidx_blks_hit	bigint	该表的 TOAST 表索引命中缓冲区数（如果存在）。

5.4.99.PG_STATS

PG_STATS 视图提供对存储在 pg_statistic 表里面的单列统计信息的访问。该视图记录的统计信息更新时间间隔由参数 autovacuum_naptime 设置。

表 1 PG_STATS 字段

名称	类型	引用	描述
schemaname	name	PG_NAMESPACE.nspname	包含表的模式名。
tablename	name	PG_CLASS.relname	表名。
attname	name	PG_ATTRIBUTE.attname	字段的名称。
inherited	Boolean	-	如果为真，则包含继承的子列，否则只是指定表的字段。
null_frac	real	-	记录中字段为空的百分比。
avg_width	integer	-	字段记录以字节记的平均宽度。
n_distinct	real	-	❖ 如果大于零，表示字段中独立数值的估计数目。 ❖ 如果小于零，表示独立数值的数目被行数除的负数。 1、用负数形式是因为 ANALYZE 认为独立数值的数目是随着表增长而增长； 2、正数的形式用于在字段看上去好像有固定的可能值数目的情况下。比如，-1

名称	类型	引用	描述
			表示一个唯一字段，独立数值的个数和行数相同。
n_dndistinct	real	-	标识 dn1 上字段中非 NULL 数据的唯一值的数目。 ❖ 如果大于零，表示独立数值的实际数目。如果小于零，表示独立数值的数目被行数除的负数。（比如，一个字段的数值平均出现概率为两次，则可以表示为 n_dndistinct=-0.5）。 ❖ 如果等于零，表示独立数值的数目未知。
most_common_vals	anyarray	-	一个字段里最常用数值的列表。如果里面的字段数值是最常见的，则为 NULL。
most_common_freqs	real[]	-	一个最常用数值的频率的列表，也就是说，每个出现的次数除以行数。如果 most_common_vals 是 NULL，则为 NULL。
histogram_bounds	anyarray	-	一个数值的列表，它把字段的数值分成几组大致相同热门的组。如果在 most_common_vals 里有数值，则在这个饼图的计算中省略。如果字段数据类型没有<操作符或者 most_common_vals 列表代表了整个分布性，则这个字段为 NULL。
correlation	real	-	统计与字段值的物理行序和逻辑行序有关。它的范围从-1 到+1。在数值接近-1 或者+1 的时候，在字段上的索引扫描将被认为比它接近零的时候开销更少，因为减少了对磁盘的随机访问。如果字段数据类型没有<操作符，则这个字段为 NULL。
most_common_elems	anyarray	-	一个最常用的非空元素的列表。
most_common	real[]	-	一个最常用元素的频率的列表。

名称	类型	引用	描述
_elem_freqs			
elem_count_histogram	real[]	-	对于独立的非空元素的统计直方图。

5.4.100.PG_TABLES

PG_TABLES 视图提供了对数据库中每个表访问的有用信息。

表 1 PG_TABLES 字段

名称	类型	引用	描述
schemaname	name	PG_NAMESPACE.nspname	包含表的模式名。
tablename	name	PG_CLASS.relname	表名。
tableowner	name	pg_get_userbyid(PG_CLASS.relowner)	表的所有者。
tablespace	name	PG_TABLESPACE.spcname	包含表的表空间，默认为 NULL。
cmindexes	Boolean	PG_CLASS.relcmindex	如果表上有索引（或者最近拥有）则为 TRUE，否则为 FALSE。
cmrules	Boolean	PG_CLASS.relcmruls	如果表上有规则，则为 TRUE，否则为 FALSE。
cmtriggers	Boolean	PG_CLASS.RELCMTRIGGERS	如果表上有触发器，则为 TRUE，否则为 FALSE。
tablecreator	name	pg_get_userbyid(po.creator)	创建表的名称。
created	timestamp with time zone	pg_object.ctime	对象的创建时间。
last_ddl_time	timestamp with time zone	pg_object.mtime	对象的最后修改时间。

5.4.101.PG_TDE_INFO

PG_TDE_INFO 视图提供了 PanWeiDB 加密信息。

表 1 PG_TDE_INFO 字段

名称	类型	描述
is_encrypt	Boolean	是否加密 PanWeiDB。 ❖ f：非加密 PanWeiDB。 ❖ t：加密 PanWeiDB。
g_tde_algo	text	加密算法。 SM4-CTR-128。 AES-CTR-128。
remain	text	保留字段。

5.4.102.PG_THREAD_WAIT_STATUS

通过 PG_THREAD_WAIT_STATUS 视图可以检测当前实例中工作线程（backend thread）以及辅助线程（auxiliary thread）的阻塞等待情况。

表 1 PG_THREAD_WAIT_STATUS 字段

名称	类型	描述
node_name	text	当前节点的名称。
db_name	text	数据库名称。
thread_name	text	线程名称。
query_id	bigint	查询 ID，对应 debug_query_id。
tid	bigint	当前线程的线程号。
sessionid	bigint	当前会话 ID。
lwtid	integer	当前线程的轻量级线程号。
psessionid	bigint	父会话 ID。
tlevel	integer	streaming 线程的层级。
smpid	integer	并行线程的 ID。
wait_status	text	当前线程的等待状态。等待状态的详细信息请参见表 2。
wait_event	text	如果 wait_status 是 acquire lock、acquire lwlock、

名称	类型	描述
		wait io 三种类型，此列描述具体的锁、轻量级锁、IO 的信息。否则是空。
locktag	text	当前线程正在等待锁的信息。
lockmode	text	当前线程正等待获取的锁模式。包含表级锁、行级锁、页级锁下的各模式。
block_sessionid	bigint	阻塞当前线程获取锁的会话标识。
global_sessionid	text	全局会话 ID。

wait_status 列的等待状态有以下状态。

表 2 等待状态列表

wait_status 值	含义
none	没在等任意事件。
acquire lock	等待加锁，要么加锁成功，要么加锁等待超时。
acquire lwlock	等待获取轻量级锁。
wait io	等待 IO 完成。
wait cmd	等待完成读取网络通信包。
wait pooler get conn	等待 pooler 完成获取连接。
wait pooler abort conn	等待 pooler 完成终止连接。
wait pooler clean conn	等待 pooler 完成清理连接。
pooler create conn:[nodename], total N	等待 pooler 建立连接，当前正在与 nodename 指定节点建立连接，且仍有 N 个连接等待建立。
get conn	获取到其他节点的连接。
set cmd: [nodename]	在连接上执行 SET/RESET/TRANSACTION BLOCK LEVEL PARA SET/SESSION LEVEL PARA SET，当前正在 nodename 指定节点上执行。
cancel query	取消某连接上正在执行的 SQL 语句。

wait_status 值	含义
stop query	停止某连接上正在执行的查询。
wait node: [nodename](plevel), total N, [pcme]	等待接收与某节点的连接上的数据，当前正在等待 nodename 节点 plevel 线程的数据，且仍有 N 个连接的数据待返回。如果状态包含 pcme 信息，则可能的阶段状态有： ❖ begin：表示处于事务开始阶段。 ❖ commit：表示处于事务提交阶段。 ❖ rollback：表示处于事务回滚阶段。
wait transaction sync: xid	等待 xid 指定事务同步。
wait wal sync	等待特定 LSN 的 wal log 完成到备机的同步。
wait data sync	等待完成数据页到备机的同步。
wait data sync queue	等待把行存的数据页或列存的 CU 放入同步队列。
flush data: [nodename](plevel), [pcme]	等待向网络中 nodename 指定节点的 plevel 对应线程发送数据。如果状态包含 pcme 信息，则可能的阶段状态为 wait quota，即当前通信流正在等待 quota 值。
stream get conn: [nodename], total N	初始化 stream flow 时，等待与 nodename 节点的 consumer 对象建立连接，且当前有 N 个待建连对象。
wait producer \ready: [nodename](plevel), total N	初始化 stream flow 时，等待每个 producer 都准备好，当前正在等待 nodename 节点 plevel 对应线程的 producer 对象准备好，且仍有 N 个 producer 对象处于等待状态。
synchronize quit	stream plan 结束时，等待 stream 线程组内的线程统一退出。
wait stream nodegroup destroy	stream plan 结束时，等待销毁 stream node group。

wait_status 值	含义
wait active statement	等待作业执行，正在资源负载管控中。
analyze: [relname], [pcme]	当前正在对表 relname 执行 analyze。如果状态包含 pcme 信息，则为 autovacuum，表示是数据库自动开启 AutoVacuum 线程执行的 analyze 分析操作。
vacuum: [relname], [pcme]	当前正在对表 relname 执行 vacuum。如果状态包含 pcme 信息，则为 autovacuum，表示是数据库自动开启 AutoVacuum 线程执行的 vacuum 清理操作。
vacuum full: [relname]	当前正在对表 relname 执行 vacuum full 清理。
create index	当前正在创建索引。
HashJoin - [build hash | write file]	当前是 HashJoin 算子，主要关注耗时的执行阶段。 ❖ build hash：表示当前 HashJoin 算子正在建立哈希表。 ❖ write file：表示当前 HashJoin 算子正将数据写入磁盘。
HashAgg - [build hash | write file]	当前是 HashAgg 算子，主要关注耗时的执行阶段。 ❖ build hash：表示当前 HashAgg 算子正在建立哈希表。 ❖ write file：表示当前 HashAgg 算子正在将数据写入磁盘。
HashSetop - [build hash | write file]	当前是 HashSetop 算子，主要关注耗时的执行阶段。 ❖ build hash：表示当前 HashSetop 算子正在建立哈希表。 ❖ write file：表示当前 HashSetop 算子正在将数据写入磁盘。
Sort | Sort - [fetch tuple | write file]	当前是 Sort 算子做排序，fetch tuple 表示 Sort 算子正在获取 tuple，write file 表示 Sort 算子正在将数据写入磁盘。
Material | Material - write	当前是 Material 算子，write file 表示 Material 算

wait_status 值	含义
file	子正在将数据写入磁盘。
NestLoop	当前是 NestLoop 算子。
wait memory	等待内存获取。
wait sync consumer next step	Stream 算子等待消费者执行。
wait sync producer next step	Stream 算子等待生产者执行。

表 3 轻量级锁等待事件列表

wait_event 类型	类型描述
ShmemIndexLock	用于保护共享内存中的主索引哈希表。
OidGenLock	用于避免不同线程产生相同的 OID。
XidGenLock	用于避免两个事务获得相同的 xid。
ProcArrayLock	用于避免并发访问或修改 ProcArray 共享数组。
SInvalReadLock	用于避免与清理失效消息并发执行。
SInvalWriteLock	用于避免与其他写失效消息、清理失效消息并发执行。
WALInsertLock	用于避免与其他 WAL 插入操作并发执行。
WALWriteLock	用于避免并发 WAL 写盘。
ControlFileLock	用于避免 pg_control 文件的读写并发、写写并发。
CheckpointLock	用于避免多个 checkpoint 并发执行。
CLogControlLock	用于避免并发访问或者修改 Clog 控制数据结构。
SubtransControlLock	用于避免并发访问或者修改子事务控制数据结构。
MultiXactGenLock	用于串行分配唯一 MultiXact id。

wait_event 类型	类型描述
MultiXactOffsetControlLock	用于避免对 pg_multixact/offset 的写写并发和读写并发。
MultiXactMemberControlLock	用于避免对 pg_multixact/members 的写写并发和读写并发。
RelCacheInitLock	用于失效消息场景对 init 文件进行操作时加锁。
CheckpointerCommLock	用于向 checkpointer 发起文件刷盘请求场景，需要串行的向请求队列插入请求结构。
TwoPcmeStateLock	用于避免并发访问或者修改两阶段信息共享数组。
TablespaceCreateLock	用于确定 tablespace 是否已经存在。
BtreeVacuumLock	用于防止 vacuum 清理 B-tree 中还在使用的页面。
AutovacuumLock	用于串行化访问 autovacuum worker 数组。
AutovacuumScheduleLock	用于串行化分配需要 vacuum 的 table。
AutoanalyzeLock	用于获取和释放允许执行 Autoanalyze 的任务资源。
SyncScanLock	用于确定 heap 扫描时某个 relfilenode 的起始位置。
NodeTableLock	用于保护存放数据库节点信息的共享结构。
PoolerLock	用于保证两个线程不会同时从连接池里取到相同的连接。
RelationMappingLock	用于等待更新系统表到存储位置之间映射的文件。
AsyncCtlLock	用于避免并发访问或者修改共享通知状态。
AsyncQueueLock	用于避免并发访问或者修改共享通知信息队列。
SerializableXactHashLock	用于避免对于可串行事务共享结构的写写并发和读写并发。

wait_event 类型	类型描述
SerializableFinishedListLock	用于避免对于已完成可串行事务共享链表的写写并发和读写并发。
SerializablePredicateLockListLock	用于保护对于可串行事务持有的锁链表。
OldSerXidLock	用于保护记录冲突可串行事务的结构。
FileStatLock	用于保护存储统计文件信息的数据结构。
SyncRepLock	用于在主备复制时保护 xlog 同步信息。
DataSyncRepLock	用于在主备复制时保护数据页同步信息。
CStoreColspaceCacheLock	用于保护列存表的 CU 空间分配。
CStoreCUCacheSweepLock	用于列存 CU Cache 循环淘汰。
MetaCacheSweepLock	用于元数据循环淘汰。
ExtensionConnectorLibLock	用于初始化 ODBC 连接场景，在加载与卸载特定动态库时进行加锁。
SearchServerLibLock	用于 GPU 加速场景初始化加载特定动态库时，对读文件操作进行加锁。
LsnXlogChkFileLock	用于串行更新特定结构中记录的主备机的 xlog flush 位置点。
ReplicationSlotAllocationLock	用于主备复制时保护主机端的流复制槽的分配。
ReplicationSlotControlLock	用于主备复制时避免并发更新流复制槽状态。
ResourcePoolHashLock	用于避免并发访问或者修改资源池哈希表。
WorkloadStatHashLock	用于避免并发访问或者修改包含数据库主节点的 SQL

wait_event 类型	类型描述
	请求构成的哈希表。
WorkloadIoStatHashLock	用于避免并发访问或者修改用于统计当前数据库节点的 IO 信息的哈希表。
WorkloadCGroupHashLock	用于避免并发访问或者修改 Cgroup 信息构成的哈希表。
OBSGetPathLock	用于避免对 obs 路径的写写并发和读写并发。
WorkloadUserInfoLock	用于避免并发访问或修改负载管理的用户信息哈希表。
WorkloadRecordLock	用于避免并发访问或修改在内存自适应管理时对数据库主节点收到请求构成的哈希表。
WorkloadIOUtilLock	用于保护记录 iostat，CPU 等负载信息的结构。
WorkloadNodeGroupLock	用于避免并发访问或者修改内存中的 nodegroup 信息构成的哈希表。
JobShmemLock	用于定时任务功能中保护定时读取的全局变量。
OBSRuntimeLock	用于获取环境变量，如 GASSHOME。
LLVMDumpIRLock	用于导出动态生成函数所对应的汇编语言。
LLVMParseIRLock	用于在查询开始处从 IR 文件中编译并解析已写好的 IR 函数。
CriticalCacheBuildLock	用于从共享或者本地缓存初始化文件中加载 cache 的场景。
WaitCountHashLock	用于保护用户语句计数功能场景中的共享结构。
BufMappingLock	用于保护对共享缓冲映射表的操作。
LockMgrLock	用于保护常规锁结构信息。
PredicateLockMgrLock	用于保护可串行事务锁结构信息。
OperatorRealTLock	用于避免并发访问或者修改记录算子级实时数据的全局

wait_event 类型	类型描述
	结构。
OperatorHistLock	用于避免并发访问或者修改记录算子级历史数据的全局结构。
SessionRealTLock	用于避免并发访问或者修改记录 query 级实时数据的全局结构。
SessionHistLock	用于避免并发访问或者修改记录 query 级历史数据的全局结构。
CacheSlotMappingLock	用于保护 CU Cache 全局信息。
BarrierLock	用于保证当前只有一个线程在创建 Barrier。
dummyServerInfoCacheLock	用于保护缓存加速 PanWeiDB 连接信息的全局哈希表。
RPNumberLock	用于加速 PanWeiDB 的数据库节点对正在执行计划的任务线程的计数。
CBMParseXlogLock	Cbm 解析 xlog 时的保护锁。
RelfilenodeReuseLock	避免错误地取消已重用的列属性文件的链接。
RcvWriteLock	防止并发调用 WalDataRcvWrite。
PercentileLock	用于保护全局 PercentileBuffer。
CSNBufMappingLock	保护 csn 页面。
UniqueSQLMappingLock	用于保护 uniquesql hash table。
DelayDDLLock	防止并发 ddl。
CLOG Ctl	用于避免并发访问或者修改 Clog 控制数据结构。
Async Ctl	保护 Async buffer。
MultiXactOffset Ctl	保护 MultiXact offet 的 slru buffer。

wait_event 类型	类型描述
MultiXactMember Ctl	保护 MultiXact member 的 slrubuffer。
OldSerXid SLRU Ctl	保护 old xids 的 slru buffer。
ReplicationSlotLock	用于保护 ReplicationSlot。
PGPROCLock	用于保护 pgproc。
MetaCacheLock	用于保护 MetaCache。
DataCacheLock	用于保护 datacache。
InstrUserLock	用于保护 InstrUserHTAB。
BadBlockStatHashLock	用于保护 global_bad_block_stat hash 表。
BufFreelistLock	用于保证共享缓冲区空闲列表操作的原子性。
CUSlotListLock	用于控制列存缓冲区槽位的并发操作。
AddinShmemInitLock	保护共享内存对象的初始化。
AlterPortLock	保护协调节点更改注册端口号的操作。
FdwPartitionCaheLock	HDFS 分区表缓冲区的管理锁。
DfsConnectorCacheLock	DFSConnector 缓冲区的管理锁。
DfsSpaceCacheLock	HDFS 表空间管理缓冲区的管理锁。
FullBuildXlogCopyStartPtrLock	用于保护全量 Build 中 Xlog 拷贝的操作。
DfsUserLoginLock	用于 HDFS 用户登录以及认证。
LogicalReplicationSlotPersistentDataLock	用于保护逻辑复制过程中复制槽位的数据。
WorkloadSessionInfoLock	保护负载管理 session info 内存 hash 表访问。
InstrWorkloadLock	保护负载管理统计信息的内存 hash 表访问。

wait_event 类型	类型描述
PgfdwLock	用于管理实例向 Foreign server 建立连接。
InstanceTimeLock	用于获取实例中会话的时间信息。
XlogRemoveSegLock	保护 Xlog 段文件的回收操作。
DnUsedSpaceHashLock	用于更新会话对应的空间使用信息。
CsnMinLock	用于计算 CSNmin。
GPCCommitLock	用于保护全局 Plan Cache hash 表的添加操作。
GPCClearLock	用于保护全局 Plan Cache hash 表的清除操作。
GPCTimelineLock	用于保护全局 Plan Cache hash 表检查 Timeline 的操作。
TsTagsCacheLock	用于时序 tag 缓存管理。
InstanceRealTLock	用于保护共享实例统计信息 hash 表的更新操作。
CLogBufMappingLock	用于提交日志缓存管理。
GPCMappingLock	用于全局 Plan Cache 缓存管理。
GPCPrepareMappingLock	用于全局 Plan Cache 缓存管理。
BufferIOLock	保护共享缓冲区页面的 IO 操作。
BufferContentLock	保护共享缓冲区页面内容的读取、修改。
CSNLOG Ctl	用于 CSN 日志管理。
DoubleWriteLock	用于双写的管理操作。
RowPageReplicationLock	用于管理行存储的数据页复制。
extension	其他轻量锁。

表 4 IO 等待事件列表

wait_event 类型	类型描述

wait_event 类型	类型描述
BufFileRead	从临时文件中读取数据到指定 buffer。
BufFileWrite	向临时文件中写入指定 buffer 中的内容。
ControlFileRead	读取 pg_control 文件。主要在数据库启动、执行 checkpoint 和主备校验过程中发生。
ControlFileSync	将 pg_control 文件持久化到磁盘。数据库初始化时发生。
ControlFileSyncUpdate	将 pg_control 文件持久化到磁盘。主要在数据库启动、执行 checkpoint 和主备校验过程中发生。
ControlFileWrite	写入 pg_control 文件。数据库初始化时发生。
ControlFileWriteUpdate	更新 pg_control 文件。主要在数据库启动、执行 checkpoint 和主备校验过程中发生。
CopyFileRead	copy 文件时读取文件内容。
CopyFileWrite	copy 文件时写入文件内容。
DataFileExtend	扩展文件时向文件写入内容。
DataFileFlush	将表数据文件持久化到磁盘。
DataFileImmediateSync	将表数据文件立即持久化到磁盘。
DataFilePrefetch	异步读取表数据文件。
DataFileRead	同步读取表数据文件。
DataFileSync	将表数据文件的修改持久化到磁盘。
DataFileTruncate	表数据文件 truncate。
DataFileWrite	向表数据文件写入内容。
LockFileAddToDataDirRead	读取“postmaster.pid”文件。
LockFileAddToDataDirSync	将“postmaster.pid”内容持久化到磁盘。

wait_event 类型	类型描述
LockFileAddToDataDirWrite	将 pid 信息写到“postmaster.pid”文件。
LockFileCreateRead	读取 LockFile 文件“%s.lock”。
LockFileCreateSync	将 LockFile 文件“%s.lock”内容持久化到磁盘。
LockFileCreateWRITE	将 pid 信息写到 LockFile 文件“%s.lock”。
RelationMapRead	读取系统表到存储位置之间的映射文件。
RelationMapSync	将系统表到存储位置之间的映射文件持久化到磁盘。
RelationMapWrite	写入系统表到存储位置之间的映射文件。
ReplicationSlotRead	读取流复制槽文件。重新启动时发生。
ReplicationSlotRestoreSync	将流复制槽文件持久化到磁盘。重新启动时发生。
ReplicationSlotSync	checkpoint 时将流复制槽临时文件持久化到磁盘。
ReplicationSlotWrite	checkpoint 时写流复制槽临时文件。
SLRUFlushSync	将 pg_clog、pg_subtrans 和 pg_multixact 文件持久化到磁盘。主要在执行 checkpoint 和数据库停机时发生。
SLRURead	读取 pg_clog、pg_subtrans 和 pg_multixact 文件。
SLRUSync	将脏页写入文件 pg_clog、pg_subtrans 和 pg_multixact 并持久化到磁盘。主要在执行 checkpoint 和数据库停机时发生。
SLRUWrite	写入 pg_clog、pg_subtrans 和 pg_multixact 文件。
TimelineHistoryRead	读取 timeline history 文件。在数据库启动时发生。
TimelineHistorySync	将 timeline history 文件持久化到磁盘。在数据库启动时发生。

wait_event 类型	类型描述
TimelineHistoryWrite	写入 timeline history 文件。在数据库启动时发生。
TwopcmeFileRead	读取 pg_twopcme 文件。在两阶段事务提交、两阶段事务恢复时发生。
TwopcmeFileSync	将 pg_twopcme 文件持久化到磁盘。在两阶段事务提交、两阶段事务恢复时发生。
TwopcmeFileWrite	写入 pg_twopcme 文件。在两阶段事务提交、两阶段事务恢复时发生。
WALBootstrapSync	将初始化的 WAL 文件持久化到磁盘。在数据库初始化发生。
WALBootstrapWrite	写入初始化的 WAL 文件。在数据库初始化发生。
WALCopyRead	读取已存在的 WAL 文件并进行复制时产生的读操作。在执行归档恢复完后发生。
WALCopySync	将复制的 WAL 文件持久化到磁盘。在执行归档恢复完后发生。
WALCopyWrite	读取已存在 WAL 文件并进行复制时产生的写操作。在执行归档恢复完后发生。
WALInitSync	将新初始化的 WAL 文件持久化磁盘。在日志回收或写日志时发生。
WALInitWrite	将新创建的 WAL 文件初始化为 0。在日志回收或写日志时发生。
WALRead	从 xlog 日志读取数据。两阶段文件 redo 相关的操作产生。
WALSyncMethodAssign	将当前打开的所有 WAL 文件持久化到磁盘。
WALWrite	写入 WAL 文件。
WALBufferAccess	WAL Buffer 访问（出于性能考虑，内核代码里只统计访问次数，未统计其访问耗时）。

wait_event 类型	类型描述
WALBufferFull	WAL Buffer 满时，写 wal 文件相关的处理。
DoubleWriteFileRead	双写 文件读取。
DoubleWriteFileSync	双写 文件强制刷盘。
DoubleWriteFileWrite	双写 文件写入。
PredoProcessPending	并行日志回放中当前记录回放等待其他记录回放完成。
PredoApply	并行日志回放中等待当前工作线程等待其他线程回放至本线程 LSN。
DisableConnectFileRead	HA 锁分片逻辑文件读取。
DisableConnectFileSync	HA 锁分片逻辑文件强制刷盘。
DisableConnectFileWrite	HA 锁分片逻辑文件写入。

当 wait_status 值为 acquire lock（事务锁）时对应的 wait_event 等待事件类型与描述信息如下。

表 5 事务锁等待事件列表

wait_event 类型	类型描述
relation	对表加锁。
extend	对表扩展空间时加锁。
partition	对分区表加锁。
partition_seq	对分区表的分区加锁。
page	对表页面加锁。
tuple	对页面上的 tuple 加锁。

wait_event 类型	类型描述
transactionid	对事务 ID 加锁。
virtualxid	对虚拟事务 ID 加锁。
object	加对象锁。
cstore_freespace	对列存空闲空间加锁。
userlock	加用户锁。
advisory	加 advisory 锁。

5.4.103.PG_TIMEZONE_ABBREVS

PG_TIMEZONE_ABBREVS 视图提供了显示了所有可用的时区信息。

表 1 PG_TIMEZONE_ABBREVS 字段

名称	类型	描述
abbrev	text	时区名缩写。
utc_offset	interval	相对于 UTC 的偏移量。
is_dst	Boolean	如果当前正处于夏令时范围则为 TRUE，否则为 FALSE。

5.4.104.PG_TIMEZONE_NAMES

PG_TIMEZONE_NAMES 视图提供了显示了所有能够被 SET TIMEZONE 识别的时区名及其缩写、UTC 偏移量、是否夏时制。

表 1 PG_TIMEZONE_NAMES 字段

名称	类型	描述
abbrev	text	时区名缩写。
utc_offset	interval	相对于 UTC 的偏移量。

名称	类型	描述
is_dst	Boolean	如果当前正处于夏令时范围则为 TRUE，否则为 FALSE。

5.4.105.PG_TOTAL_MEMORY_DETAIL

PG_TOTAL_MEMORY_DETAIL 视图显示某个数据库节点内存使用情况。

表 1 PG_TOTAL_MEMORY_DETAIL 字段

名称	类型	描述
nodename	text	节点名称。
memorytype	text	内存的名称。 ❖ max_process_memory：PanWeiDB 实例所占用的内存大小。 ❖ process_used_memory：进程所使用的内存大小。 ❖ max_dynamic_memory：最大动态内存。 ❖ dynamic_used_memory：已使用的动态内存。 ❖ dynamic_peak_memory：内存的动态峰值。 ❖ dynamic_used_shrctx：最大动态共享内存上下文。 ❖ dynamic_peak_shrctx：共享内存上下文的动态峰值。 ❖ max_shared_memory：最大共享内存。 ❖ shared_used_memory：已使用的共享内存。 ❖ max_cstore_memory：列存所允许使用的最大内存。 ❖ cstore_used_memory：列存已使用的内存大小。 ❖ max_sctpcomm_memory：sctp 通信所允

名称	类型	描述
		许使用的最大内存。 ❖ sctpcomm_used_memory：sctp 通信已使用的内存大小。 ❖ sctpcomm_peak_memory：sctp 通信的内存峰值。 ❖ other_used_memory：其他已使用的内存大小。 ❖ gpu_max_dynamic_memory：GPU 最大动态内存。 ❖ gpu_dynamic_used_memory：GPU 已使用的动态内存。 ❖ gpu_dynamic_peak_memory：GPU 内存的动态峰值。 ❖ pooler_conn_memory：链接池申请内存计数。 ❖ pooler_freeconn_memory：链接池空闲连接的内存计数。 ❖ storage_compress_memory：存储模块压缩使用的内存大小。 ❖ udf_reserved_memory：UDF 预留的内存大小。
memorymbytes	integer	内存使用的大小，单位为 MB。

5.4.106.PG_TOTAL_USER_RESOURCE_INFO

PG_TOTAL_USER_RESOURCE_INFO 视图显示所有用户资源使用情况，需要使用管理员用户进行查询。此视图在参数 use_workload_manager（参考：GUC 参数说明->负载管理章节）为 on 时才有效。其中，IO 相关监控项在参数 enable_logical_io_statistics 为 on 时才有效。

表 1 PG_TOTAL_USER_RESOURCE_INFO 字段

名称	类型	描述
username	name	用户名。
used_memory	integer	正在使用的内存大小，单位 MB。
total_memory	integer	可以使用的内存大小，单位 MB。值为 0 表示未限制最大可用内存，其限制取决于数据库最大可用内存。
used_cpu	double precision	正在使用的 CPU 核数（仅统计复杂作业 CPU 使用 情况，且该值为相关控制组的 CPU 使用统计值）。
total_cpu	integer	在该机器节点上，用户关联控制组的 CPU 核数总和。
used_space	bigint	已使用的永久表存储空间大小，单位 KB。
total_space	bigint	可使用的永久表存储空间大小，单位 KB，值为-1 表示未限制最大存储空间。
used_temp_space	bigint	已使用的临时空间大小，单位 KB。
total_temp_space	bigint	可使用的临时空间总大小，单位 KB，值为-1 表示未限制。
used_spill_space	bigint	已使用的算子落盘空间大小，单位 KB。
total_spill_space	bigint	可使用的算子落盘空间总大小，单位 KB，值为-1 表示未限制。
read_kbytes	bigint	数据库主节点：过去 5 秒内，该用户在数据库节点上复杂作业 read 的字节总数（单位 KB）。 数据库节点：实例启动至当前时间为止，该用户复杂作业 read 的字节总数（单位 KB）。 仅分布式可用。
write_kbytes	bigint	数据库主节点：过去 5 秒内，该用户在数

名称	类型	描述
		据库节点上复杂作业 write 的字节总数（单位 KB）。 数据库节点：实例启动至当前时间为止，该用户复杂作业 write 的字节总数（单位 KB）。 仅分布式可用。
read_counts	bigint	数据库主节点：过去 5 秒内，该用户在数据库节点上复杂作业 read 的次数之和（单位次）。 数据库节点：实例启动至当前时间为止，该用户复杂作业 read 的次数之和（单位次）。 仅分布式可用。
write_counts	bigint	数据库主节点：过去 5 秒内，该用户在数据库节点上复杂作业 write 的次数之和（单位次）。 数据库节点：实例启动至当前时间为止，该用户复杂作业 write 的次数之和（单位次）。 仅分布式可用。
read_speed	double precision	数据库主节点：过去 5 秒内，该用户在单个数据库节点上复杂作业 read 平均速率（单位 KB/s）。 数据库节点：过去 5 秒内，该用户在该数据库节点上复杂作业 read 平均速率（单位 KB/s）。 仅分布式可用。
write_speed	double precision	数据库主节点：过去 5 秒内，该用户在单个数据库节点上复杂作业 write 平均速率（单位 KB/s）。 数据库节点：过去 5 秒内，该用户在该数

名称	类型	描述
		据库节点上复杂作业 write 平均速率（单位 KB/s）。 仅分布式可用。

5.4.107.PG_TOTAL_USER_RESOURCE_INFO_OID

PG_TOTAL_USER_RESOURCE_INFO_OID 视图显示所有用户资源使用情况，需要使用管理员用户进行查询。此视图在参数 use_workload_manager（参考：GUC 参数说明->负载管理章节）为 on 时才有效。

表 1 PG_TOTAL_USER_RESOURCE_INFO_OID 字段

名称	类型	描述
userid	oid	用户 ID。
used_memory	integer	正在使用的内存大小，单位 MB。
total_memory	integer	可以使用的内存大小，单位 MB。值为 0 表示未限制最大可用内存，其限制取决于数据库最大可用内存。
used_cpu	double precision	正在使用的 CPU 核数。
total_cpu	integer	在该机器节点上，用户关联控制组的 CPU 核数总和。
used_space	bigint	已使用的存储空间大小，单位 KB。
total_space	bigint	可使用的存储空间大小，单位 KB，值为-1 表示未限制最大存储空间。
used_temp_space	bigint	已使用的临时空间大小，单位 KB。
total_temp_space	bigint	可使用的临时空间总大小，单位 KB，值为-1 表示未限制。
used_spill_space	bigint	已使用的下盘空间大小。单位 KB。

名称	类型	描述
total_spill_space	bigint	可使用的下盘空间总大小，单位 KB，值为-1 表示未限制。
read_kbytes	bigint	读磁盘数据量，单位 KB。
write_kbytes	bigint	写磁盘数据量，单位 KB。
read_counts	bigint	读磁盘次数。
write_counts	bigint	写磁盘次数。
read_speed	double precision	读磁盘速率，单位 B/ms。
write_speed	double precision	写磁盘速率，单位 B/ms。

5.4.108.PG_USER

PG_USER 视图提供了访问数据库用户的信息，默认只有初始化用户和具有 sysadmin 属性的用户可以查看，其余用户需要赋权后才可以查看。

表 1 PG _USER 字段

名称	类型	描述
usename	name	用户名。
usesysid	oid	此用户的 ID。
usecreatedb	boolean	用户是否可以创建数据库。
usesuper	boolean	用户是否是拥有最高权限的初始系统管理员。
usecatupd	boolean	用户是否可以直接更新系统表。只有 usesysid=10 的初始系统管理员拥有此权限。其他用户无法获得此权限。
userepl	boolean	用户是否可以复制数据流。
passwd	text	密文存储后的用户口令，始终为

名称	类型	描述
		********。
valbegin	timestamp with time zone	帐户的有效开始时间；如果没有设置有效开始时间，则为 NULL。
valuntil	timestamp with time zone	帐户的有效结束时间；如果没有设置有效结束时间，则为 NULL。
respool	name	用户所在的资源池。
parent	oid	父用户 OID。
spacelimit	text	永久表存储空间限额。
tempspacelimit	text	临时表存储空间限额。
spillspacelimit	text	算子落盘空间限额。
useconfig	text[]	运行时配置参数的会话缺省。
nodegroup	name	用户关联的逻辑 PanWeiDB 名称，如果该用户没有管理逻辑 PanWeiDB，则该字段为空。
usemonitoradmin	boolean	用户是否是监控管理员。 t（true）：表示是。 f（false）：表示否。
useoperatoradmin	boolean	用户是否是运维管理员。 t（true）：表示是。 f（false）：表示否。
usepolicyadmin	boolean	用户是否是安全策略管理员。 t（true）：表示是。 f（false）：表示否。

5.4.109._PG_USER_MAPPINGS

存储从本地用户到远程的映射。该视图只有 sysadmin 权限可以查看。

表 1 _PG_USER_MAPPINGS 字段

名称	类型	引用	描述
umid	oid	PG_USER_MAPPING.oid	用户映射的 OID。
srvid	oid	PG_FOREIGN_SERVER.oid	包含这个映射的外部服务器的 OID。
srvname	name	PG_FOREIGN_SERVER.srvname	外部服务器的名称。
umuser	oid	PG_AUTHID.oid	被映射的本地角色的 OID，如果用户映射是公共的则为 0。
usename	name	-	被映射的本地用户的名称。
umoptions	text[ ]	-	如果当前用户是外部服务器的所有者，则为用户映射指定选项，使用“keyword=value”字符串，否则为 null。

5.4.110.PG_VARIABLE_INFO

PG_VARIABLE_INFO 视图用于查询 PanWeiDB 中当前节点的 xid、oid 的状态。

表 1 PG_VARIABLE_INFO 字段

名称	类型	描述
node_name	text	节点名称。
next_oid	oid	该节点下一次生成的 oid。
next_xid	xid	该节点下一次生成的事务号。
oldest_xid	xid	该节点最老的事务号。
xid_vac_limit	xid	强制 autovacuum 的临界点。

名称	类型	描述
oldest_xid_db	oid	该节点 datafrozenxid 最小的数据库 oid。
last_extend_csn_logpage	xid	最后一次扩展 csnlog 的页面号。
start_extend_csn_logpage	xid	csnlog 扩展的起始页面号。
next_commit_seqno	xid	该节点下次生成的 csn 号。
latest_completed_xid	xid	该节点提交或者回滚后节点上的最新事务号。
startup_max_xid	xid	该节点关机前的最后一个事务号。

5.4.111.PG_VIEWS

PG_VIEWS 视图提供访问数据库中每个视图的有用信息。

表 1 PG_VIEWS 字段

名称	类型	引用	描述
schemaname	name	PG_NAMESPACE.nspname	包含视图的模式名。
viewname	name	PG_CLASS.relname	视图名。
viewowner	name	PG_AUTHID.Erolname	视图的所有者。
definition	text	-	视图的定义。

5.4.112.PG_WLM_STATISTICS

PG_WLM_STATISTICS 视图显示作业结束后或已被处理异常后的负载管理相关信息。查询该视图需要 sysadmin 权限。

表 1 PG_WLM_STATISTICS 字段

名称	类型	描述
statement	text	执行了异常处理的语句。
block_time	bigint	语句执行前的阻塞时间。
elapsed_time	bigint	语句的实际执行时间。
total_cpu_time	bigint	语句执行异常处理时数据库实例上 CPU 使用的总时

名称	类型	描述
		间。
qualification_time	bigint	语句检查倾斜率的时间周期。
cpu_skew_percent	integer	语句在执行异常处理时数据库实例上 CPU 使用的倾斜率。
control_group	text	语句执行异常处理时所使用的 Cgroups。
status	text	语句执行异常处理后的状态，包括： pending：执行前预备状态。 Running：执行进行状态。 Finished：执行正常结束。 Abort：执行异常终止。
action	text	语句执行的异常处理动作，包括：abort：执行终止操作。adjust：执行 Cgroups 调整操作，目前只有降级操作。finish：正常结束。

5.4.113.PGXC_PREPARED_XACTS

PGXC_PREPARED_XACTS 视图显示当前处于 prepared 阶段的两阶段事务。只有 system admin 和 monitor admin 用户有权限查看。

表 1 PGXC_PREPARED_XACTS 字段

名称	类型	描述
pgxc_prepared_xact	text	查看当前处于 prepared 阶段的两阶段事务。

5.4.114.PLAN_TABLE

PLAN_TABLE 显示用户通过执行 EXPLAIN PLAN 收集到的计划信息。计划信息的生命周期是 session 级别，session 退出后相应的数据将被清除。同时不同 session 和不同 user 间的数据是相互隔离的。

表 1 PLAN_TABLE 字段

名称	类型	描述
statement_id	varchar2(30)	用户输入的查询标签。
plan_id	bigint	查询标识。

名称	类型	描述
id	int	查询生成的计划中的每一个执行算子的编号。
operation	varchar2(30)	计划中算子的操作描述。
options	varchar2(255)	操作选项。
object_name	name	操作对应的对象名，非查询中使用到的对象别名。来自于用户定义。
object_type	varchar2(30)	对象类型。
object_owner	name	对象所属 schema，来自于用户定义。
projection	varchar2(4000)	操作输出的列信息。

说明

- object_type 取值范围为 PG_CLASS（参考：系统表和系统视图->系统表->PG_CLASS 章节）中定义的 relkind 类型（TABLE 普通表，INDEX 索引，SEQUENCE 序列，VIEW 视图，COMPOSITE TYPE 复合类型，TOASTVALUE TOAST 表）和计划使用到的 rtekind(SUBQUERY,JOIN,FUNCTION,VALUES,CTE,REMOTE_QUERY)。
- object_owner 对于 RTE 来说是计划中使用的对象描述，非用户定义的类型不存在 object_owner。
- statement_id、object_name、object_owner、projection 字段内容遵循用户定义的大小写存储，其他字段内容采用大写存储。
- 支持用户对 PLAN_TABLE 进行 SELECT 和 DELETE 操作，不支持其他 DML 操作。

5.4.115.STATEMENT_HISTORY

获得当前节点的执行语句的信息。查询视图必须具有 sysadmin 权限或者 monitor admin 权限。只可在系统库中查询到结果，用户库中无法查询。

表 1 STATEMENT_HISTORY 字段

名称	类型	描述
db_name	name	数据库名称。
schema_name	name	schema 名称。
origin_node	integer	节点名称。
user_name	name	用户名。
application_name	text	用户发起的请求的应用程序名称。
client_addr	text	用户发起的请求的客户端地址。
client_port	integer	用户发起的请求的客户端端口。
unique_query_id	bigint	归一化 SQL ID。
debug_query_id	bigint	唯一 SQL ID。
query	text	归一化 SQL。
start_time	timestamp with time zone	语句启动的时间。
finish_time	timestamp with time zone	语句结束的时间。
slow_sql_threshold	bigint	语句执行时慢 SQL 的标准。
transaction_id	bigint	事务 ID。
thread_id	bigint	执行线程 ID。
session_id	bigint	用户 session id。
n_soft_parse	bigint	软解析次数，n_soft_parse +

名称	类型	描述
		n_hard_parse 可能大于 n_calls，因为子查询未计入 n_calls。
n_hard_parse	bigint	硬解析次数，n_soft_parse + n_hard_parse 可能大于 n_calls，因为子查询未计入 n_calls。
query_plan	text	语句执行计划。
n_returned_rows	bigint	SELECT 返回的结果集行数。
n_tuples_fetched	bigint	随机扫描行。
n_tuples_returned	bigint	顺序扫描行。
n_tuples_inserted	bigint	插入行。
n_tuples_updated	bigint	更新行。
n_tuples_deleted	bigint	删除行。
n_blocks_fetched	bigint	buffer 的块访问次数。
n_blocks_hit	bigint	buffer 的块命中次数。
db_time	bigint	有效的 DB 时间花费，多线程将累加（单位：微秒）。
cpu_time	bigint	CPU 时间（单位：微秒）。
execution_time	bigint	执行器内执行时间（单位：微秒）。
parse_time	bigint	SQL 解析时间（单位：微秒）。
plan_time	bigint	SQL 生成计划时间（单位：微秒）。
rewrite_time	bigint	SQL 重写时间（单位：微秒）。
pl_execution_time	bigint	plpgsql 上的执行时间（单位：微秒）。
pl_compilation_time	bigint	plpgsql 上的编译时间（单位：微秒）。
data_io_time	bigint	IO 上的时间花费（单位：微秒）。
net_send_info	text	通过物理连接发送消息的网络状态，包含时

名称	类型	描述
		间（微秒）、调用次数、吞吐量（字节）。通过该字段可以分析 SQL 在分布式系统下的网络开销，单机模式下不支持该字段。例如：{“time” :xxx，“n_calls” :xxx，“size” :xxx}。
net_recv_info	text	通过物理连接接收消息的网络状态，包含时间（微秒）、调用次数、吞吐量（字节）。通过该字段可以分析 SQL 在分布式系统下的网络开销，单机模式下不支持该字段。例如：{“time” :xxx，“n_calls” :xxx，“size” :xxx}。
net_stream_send_info	text	通过逻辑连接发送消息的网络状态，包含时间（微秒）、调用次数、吞吐量（字节）。通过该字段可以分析 SQL 在分布式系统下的网络开销，单机模式下不支持该字段。例如：{“time” :xxx，“n_calls” :xxx，“size” :xxx}。
net_stream_recv_info	text	通过逻辑连接接收消息的网络状态，包含时间（微秒）、调用次数、吞吐量（字节）。通过该字段可以分析 SQL 在分布式系统下的网络开销，单机模式下不支持该字段。例如：{“time” :xxx，“n_calls” :xxx，“size” :xxx}。
lock_count	bigint	加锁次数。
lock_time	bigint	加锁耗时。
lock_wait_count	bigint	加锁等待次数。
lock_wait_time	bigint	加锁等待耗时。
lock_max_count	bigint	最大持锁数量。
lwlock_count	bigint	轻量级加锁次数（预留）。

名称	类型	描述
lwlock_wait_count	bigint	轻量级等锁次数。
lwlock_time	bigint	轻量级加锁时间（预留）。
lwlock_wait_time	bigint	轻量级加锁时间。
details	bytea	语句锁事件的列表，该列表按时间书序记录事件，记录的数量受参数 track_stmt_details_size 的影响。 事件包括： 加锁开始 加锁结束 等锁开始 等锁结束 放锁开始 放锁结束 轻量级等锁开始 轻量级等锁结束
is_slow_sql	boolean	该 SQL 是否为 slow SQL
trace_id	text	驱动传入的 trace id，与应用的一次请求相关联。

相关特性

对应系统表 statement_history，主要目的是记录数据库运行中产生的 sql 与其运行信息，保证即便数据库重启，SQL 信息也依然可以查询到。

一般使用形式：

```
select * from DBE_PERF.statement_history;
```

主要受到以下参数控制：

- log_duration：是否记录慢查询。
- log_min_duration_statement：单位毫秒，标记 SQL 的慢查询时间，0 记录所有 SQL，-1 则不记录任何信息。

- track_stmt_stat_level：默认为`OFF,L0`。参数第一部分为非 OFF 情况下，会记录所有 SQL，第一部分为 OFF，第二部分为非 OFF 情况下，仅记录慢 SQL。
- track_stmt_parameter：追踪语句更详细内容。

此处代码判断逻辑为（以下各个条件为或判定，满足其一即可）：

1、打开了动态语句追踪功能：采用 dynamic_func_control 追踪 STMT。

2、track_stmt_stat_level 追踪第一个 level 为 L0 或者更高。

3、track_stmt_stat_level 追踪第二个 level 为 L0 或者更高，且语句运行时间大于 log_min_duration_statement 设定值，且 log_min_duration_statement 大于等于 0，并且没有打开 track_stmt_parameter。

4、打开 track_stmt_parameter，并且时间模式第一个值（消耗的 DBTIME）大于 0。

5.4.116.SUMMARY_STATIO_ALL_INDEXES

SUMMARY_STATIO_ALL_INDEXES 视图包含 PanWeiDB 内汇聚的数据库中的每个索引行， 显示特定索引的 I/O 的统计。

名称	类型	描述
relid	oid	该索引的表 ID。
indexrelid	oid	索引 ID
schemaname	name	该索引的模式名。
relname	name	该索引的表名。
indexrelname	name	索引名称。
idx_blks_read	numeric	从索引中读取的磁盘块数。
idx_blks_hit	numeric	索引命中缓存数。

6. 物化视图

物化视图是一种特殊的物理表，物化视图是相对普通视图而言的。普通视图是虚拟表，应用的局限性较大，任何对视图的查询实际上都是转换为对 SQL 语句的查询，性能并没有实际上提高。物化视图实际上就是存储 SQL 执行语句的结果，起到缓存的效果。

目前 Ustore 引擎不支持创建、使用物化视图。

6.1.全量物化视图

6.1.1.概述

全量物化视图仅支持对已创建的物化视图进行全量更新，而不支持进行增量更新。创建全量物化视图语法和 CREATE TABLE AS 语法类似。

6.1.2.使用

语法格式

❖ 创建全量物化视图

```
CREATE MATERIALIZED VIEW [ view_name ] AS { query_block };
```

❖ 全量刷新物化视图

```
REFRESH MATERIALIZED VIEW [ view_name ];
```

❖ 删除物化视图

```
DROP MATERIALIZED VIEW [ view_name ];
```

❖ 查询物化视图

```
SELECT * FROM [ view_name ];
```

示例

```
--准备数据。
panweidb=# CREATE TABLE t1(c1 int, c2 int);
panweidb=# INSERT INTO t1 VALUES(1, 1);
panweidb=# INSERT INTO t1 VALUES(2, 2);
```

```
--创建全量物化视图。
panweidb=# CREATE MATERIALIZED VIEW mv AS select count(*) from t1;
CREATE MATERIALIZED VIEW

--查询物化视图结果。
panweidb=# SELECT * FROM mv;
 count
-------
     2
(1 row)

--向物化视图中基表插入数据。
panweidb=# INSERT INTO t1 VALUES(3, 3);

--对全量物化视图做全量刷新。
panweidb=# REFRESH MATERIALIZED VIEW mv;
REFRESH MATERIALIZED VIEW

--查询物化视图结果。
panweidb=# SELECT * FROM mv;
 count
-------
     3
(1 row)

--删除物化视图。
panweidb=# DROP MATERIALIZED VIEW mv;
DROP MATERIALIZED VIEW
```

6.1.3.支持和约束

支持场景

- 通常全量物化视图所支持的查询范围与 CREATE TABLE AS 语句一致。
- 全量物化视图上支持创建索引。

❖ 支持 analyze、explain。

不支持场景

物化视图不支持增删改操作，只支持查询语句。

约束

全量物化视图的刷新、删除过程中会给基表加高级别锁，若物化视图的定义涉及多张表，需要注意业务逻辑，避免死锁产生。

6.2.增量物化视图

6.2.1.概述

增量物化视图可以对物化视图增量刷新，需要用户手动执行语句完成对物化视图在一段时间内的增量数据刷新。与全量创建物化视图的不同在于目前增量物化视图所支持场景较小。目前物化视图创建语句仅支持基表扫描语句或者 UNION ALL 语句。

6.2.2.使用

语法格式

❖ 创建增量物化视图

```
CREATE INCREMENTAL MATERIALIZED VIEW [ view_name ] AS { query_block };
```

❖ 全量刷新物化视图

```
REFRESH MATERIALIZED VIEW [ view_name ];
```

❖ 增量刷新物化视图

```
REFRESH INCREMENTAL MATERIALIZED VIEW [ view_name ];
```

❖ 删除物化视图

```
DROP MATERIALIZED VIEW [ view_name ];
```

❖ 查询物化视图

```
SELECT * FROM [ view_name ];
```

示例

```
--准备数据。
```

```
panweidb=# CREATE TABLE t1(c1 int, c2 int);
panweidb=# INSERT INTO t1 VALUES(1, 1);
panweidb=# INSERT INTO t1 VALUES(2, 2);

--创建增量物化视图。
panweidb=# CREATE INCREMENTAL MATERIALIZED VIEW mv AS SELECT * FROM t1;
CREATE MATERIALIZED VIEW

--插入数据。
panweidb=# INSERT INTO t1 VALUES(3, 3);
INSERT 0 1

--增量刷新物化视图。
panweidb=# REFRESH INCREMENTAL MATERIALIZED VIEW mv;
REFRESH MATERIALIZED VIEW

--查询物化视图结果。
panweidb=# SELECT * FROM mv;
 c1 | c2
----+----
  1 |  1
  2 |  2
  3 |  3
(3 rows)

--插入数据。
panweidb=# INSERT INTO t1 VALUES(4, 4);
INSERT 0 1

--全量刷新物化视图。
panweidb=# REFRESH MATERIALIZED VIEW mv;
REFRESH MATERIALIZED VIEW

--查询物化视图结果。
panweidb=# select * from mv;
```

```
 c1 | c2
----+----
  1 |  1
  2 |  2
  3 |  3
  4 |  4
(4 rows)

--删除物化视图。
panweidb=# DROP MATERIALIZED VIEW mv;
DROP MATERIALIZED VIEW
```

6.2.3.支持和约束

支持场景

- 单表查询语句。
- 多个单表查询的 UNION ALL。
- 物化视图上支持创建索引。
- 物化视图支持 Analyze 操作。

不支持场景

- 物化视图中不支持多表 Join 连接计划以及 subquery 计划。
- 除少部分 ALTER 操作外，不支持对物化视图中基表执行绝大多数 DDL 操作。
- 物化视图不支持增删改操作，只支持查询语句。
- 不支持用临时表/hashbucket/unlog/分区表创建物化视图。
- 不支持物化视图嵌套创建（即物化视图上创建物化视图）。
- 不支持视图上创建增量物化视图。
- 仅支持行存表，不支持列存表。
- 不支持 UNLOGGED 类型的物化视图，不支持 WITH 语法。

约束

- 物化视图定义如果为 UNION ALL，则其中每个子查询需使用不同的基表。

- 增量物化视图的创建、全量刷新、删除过程中会给基表加高级别锁，若物化视图的定义为 UNION ALL，需要注意业务逻辑，避免死锁产生。

7. PL/pgSQL

7.1.概述

7.1.1.概念

PL/pgSQL 是一种程序语言,叫作过程化 SQL 语言。

PL/pgSQL 是结合了结构化查询与数据库自身过程控制为一体的强大语言，不但支持更多的数据类型，拥有自身的变量声明、赋值语句，而且还有条件、循环等流程控制语句。过程控制结构与 SQL 数据处理能力无缝的结合形成了强大的编程语言，可以创建过程和函数以及程序包。

本章介绍的 PL/pgSQL 程序包括：存储过程、自定义函数。

7.1.2.优势

支持 SQL

SQL 是访问数据库的标准语言，通过 SQL 命令，用户可以操纵数据库中的数据。PL/pgSQL 支持所有的 SQL 数据操纵命令、游标控制命令、事务控制命令、SQL 函数、运算符和伪列。同时 PL/pgSQL 和 SQL 语言紧密集成，PL/pgSQL 支持所有的 SQL 数据类型和 NULL 值。

支持面向对象编程

PL/pgSQL 支持面向对象的编程，在 PL/pgSQL 中可以创建类型，可以对类型进行继承，可以在子程序中重载方法等。

更好的性能

SQL 是非过程语言，只能一条一条执行，而 PL/pgSQL 把一个 PL/pgSQL 块统一进行编译后执行，同时还可以把编译好的 PL/pgSQL 块存储起来，以备重用，减少了应用程序和服务器之间的通信时间，PL/pgSQL 是快速而高效的。

可移植性

使用 PL/pgSQL 编写的应用程序，可以移植到任何操作系统平台上的 Oracle 服务器，同时还可以编写可移植程序库，在不同环境中重用。

安全性

可以通过存储过程对客户机和服务器之间的应用程序逻辑进行分隔，这样可以限制对 Oracle 数据库的访问，数据库还可以授权和撤销其他用户访问的能力。

7.1.3.PL/pgSQL 块

PL/pgSQL 是一种块结构的语言，一个 PL/pgSQL 程序包含了一个或者多个逻辑块，逻辑块中可以声明变量，变量在使用之前必须先声明。除了正常的执行程序外，PL/pgSQL 还提供了专门的异常处理部分进行异常处理。

7.2.存储过程

存储过程（Stored Procedure）是一组为了完成特定功能的 SQL 语句集，它存储在数据库中，一次编译后永久有效，用户通过指定存储过程的名字并给出参数（如果该存储过程带有参数）来执行它。存储过程是数据库中的一个重要对象。在数据量特别庞大的情况下利用存储过程能达到倍速的效率提升。

简单地说，存储过程是一种封装了 SQL 的语句集，并能实现相应的逻辑功能。当存储过程执行成功后会被存储在数据库服务器中，并允许客户端直接调用，这大大提高了 SQL 语句的执行效率和安全性。

7.2.1.动态语句

7.2.1.1.动态调用存储过程

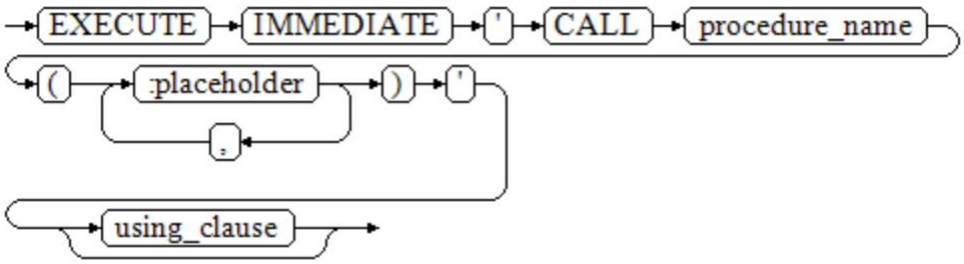
功能描述

动态调用存储过程就是在程序运行的时候调用存储过程。使用 EXECUTE IMMEDIATE...CALL 语句执行。

语法格式

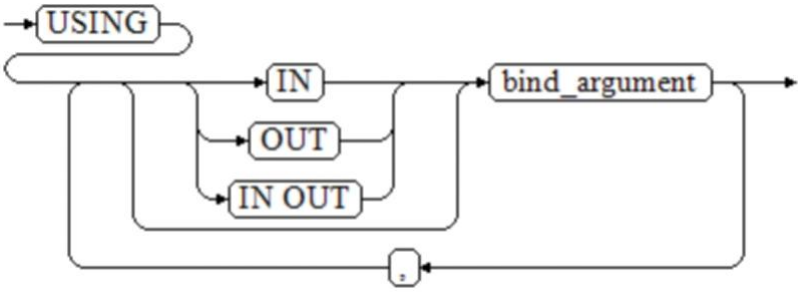
语法请参见下图。

call_procedure::=



using_clause 子句的语法参见下图。

using_clause::=



参数说明

- CALL procedure_name: 调用存储过程。
- [:placeholder1，:placeholder2，...]：存储过程参数占位符列表。占位符个数与参数个数相同。
- USING [IN|OUT|IN OUT] bind_argument: 用于指定存放传递给存储过程参数值的变量。bind_argument 前的修饰符与对应参数的修饰符一致。

注意事项

动态调用存储过程必须使用匿名的语句块将存储过程或语句块包在里面，使用 EXECUTE IMMEDIATE...USING 语句后面带 IN、OUT 来输入、输出参数。

7.2.1.2.动态调用匿名块

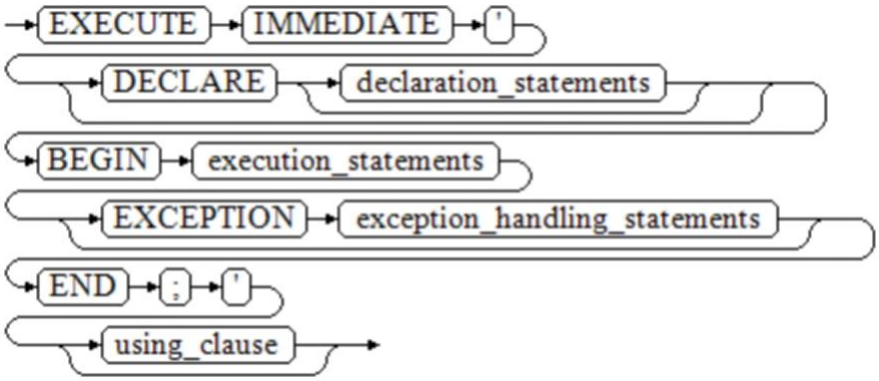
功能描述

动态调用匿名块是指在动态语句中执行匿名块，使用 EXECUTE IMMEDIATE... USING 语句后面带 IN、OUT 来输入、输出参数。

语法格式

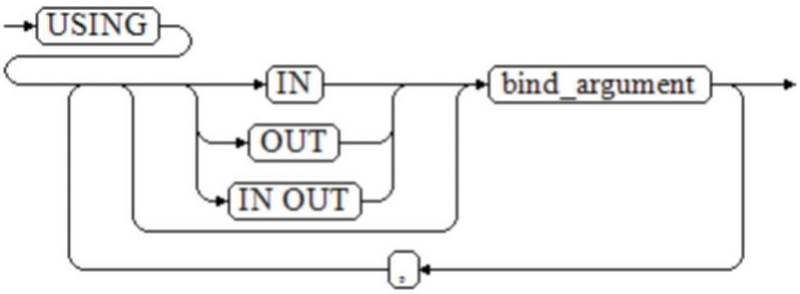
语法请参见下图。

call_anonymous_block::=



using_clause 子句的语法参见下图。

using_clause::=



对以上语法格式的解释如下：

- 匿名块程序实施部分，以 BEGIN 语句开始，以 END 语句停顿，以一个分号结束。
- USING [IN|OUT|IN OUT] bind_argument，用于指定存放传递给存储过程参数值的变量。bind_argument 前的修饰符与对应参数的修饰符一致。
- 匿名块中间的输入输出参数使用占位符来指明，要求占位符个数与参数个数相同，并且占位符所对应参数的顺序和 USING 中参数的顺序一致。

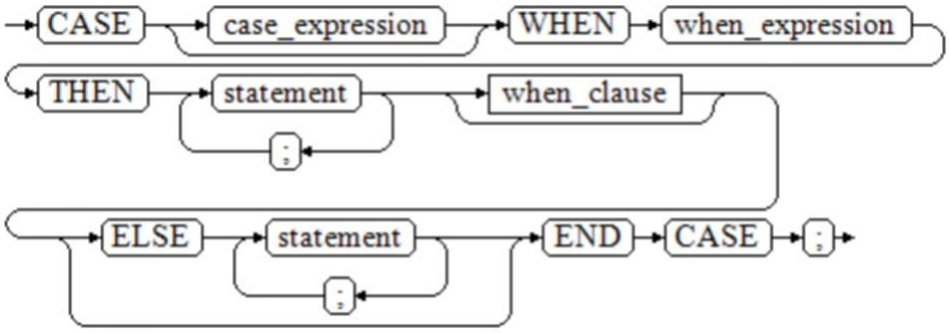
- 目前 PanWeiDB 在动态语句调用匿名块时，EXCEPTION 语句中暂不支持使用占位符进行输入输出参数的传递。

7.2.1.3.分支语句

语法格式

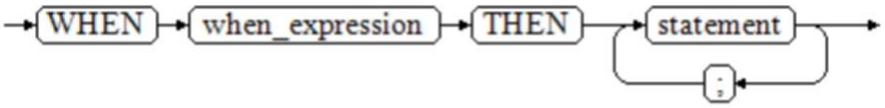
分支语句的语法请参见下图。

case_when::=



when_clause 子句的语法图参见下图。

when_clause::=



参数说明

- case_expression：变量或表达式。
- when_expression：常量或者条件表达式。
- statement：执行语句。

示例

1、创建并调用存储过程。

```
CREATE OR REPLACE PROCEDURE proc_case_branch(pi_result in integer, pi_return o
ut integer)
AS
 BEGIN
   CASE pi_result
     WHEN 1 THEN
       pi_return := 111;
      WHEN 2 THEN
       pi_return := 222;
     WHEN 3 THEN
       pi_return := 333;
     WHEN 6 THEN
       pi_return := 444;
     WHEN 7 THEN
       pi_return := 555;
     WHEN 8 THEN
       pi_return := 666;
     WHEN 9 THEN
       pi_return := 777;
     WHEN 10 THEN
       pi_return := 888;
     ELSE
       pi_return := 999;
   END CASE;
   raise info 'pi_return : %',pi_return ;
END;
/

CALL proc_case_branch(3,0);
```

当结果显示如下信息，则表示调用成功。

```
 pi_return
-------------
     333
(1 row)
```

2、删除存储过程

```
DROP PROCEDURE proc_case_branch;
```

7.2.1.4.执行动态查询语句

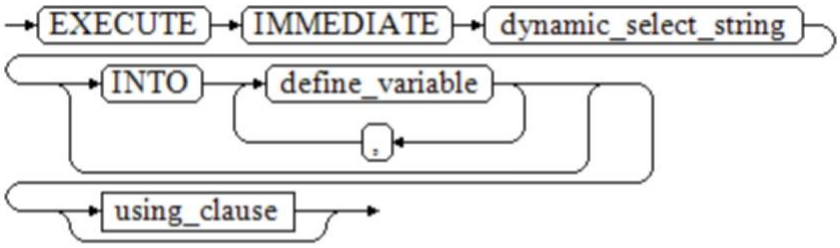
功能描述

动态查询就是在程序运行的时候来进行查询。PanWeiDB 提供两种方式：使用 EXECUTE IMMEDIATE、OPEN FOR 实现动态查询。前者通过动态执行 SELECT 语句，后者结合了游标的使用。当需要将查询的结果保存在一个数据集用于提取时，可使用 OPEN FOR 实现动态查询。

EXECUTE IMMEDIATE

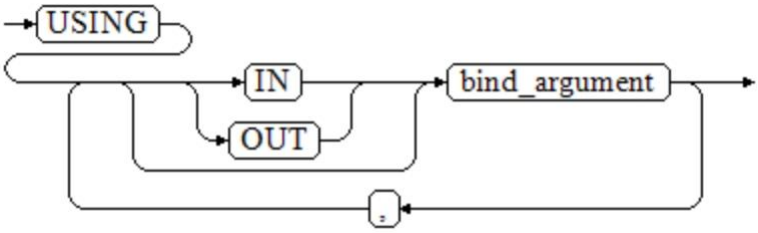
语法图请参见下图。

EXECUTE IMMEDIATE dynamic_select_clause::=



using_clause 子句的语法图参见下图。

using_clause::=



对以上语法格式的解释如下：

- define_variable：用于指定存放单行查询结果的变量。
- USING IN bind_argument：用于指定存放传递给动态 SQL 值的变量，即在 dynamic_select_string 中存在占位符时使用。
- USING OUT bind_argument：用于指定存放动态 SQL 返回值的变量。

须知

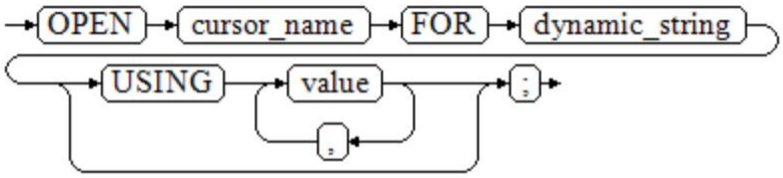
- 查询语句中，into 和 out 不能同时存在；
- 占位符命名以“:”开始，后面可跟数字、字符或字符串，与 USING 子句的 bind_argument 一一对应；
- bind_argument 只能是值、变量或表达式，不能是表名、列名、数据类型等数据库对象，即不支持使用 bind_argument 为动态 SQL 语句传递模式对象。如果存储过程需要通过声明参数传递数据库对象来构造动态 SQL 语句（常见于执行 DDL 语句时），建议采用连接运算符“||”拼接 dynamic_select_clause；
- 动态 PL/pgSQL 块允许出现重复的占位符，即相同占位符只能与 USING 子句的一个 bind_argument 按位置对应。

OPEN FOR

动态查询语句还可以使用 OPEN FOR 打开动态游标来执行。

语法参见下图。

open_for::=



参数说明：

- cursor_name：要打开的游标名。
- dynamic_string：动态查询语句。
- USING value：在 dynamic_string 中存在占位符时使用。

游标的使用请参考：PL/pgSQL->游标章节。

7.2.1.5.执行动态非查询语句

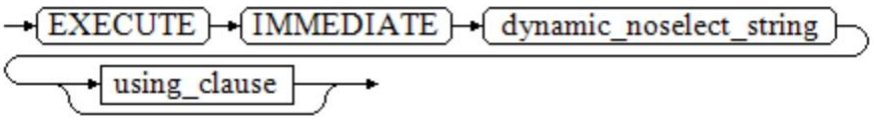
功能描述

动态非查询就是在程序运行的时候来进行非查询操作。

语法格式

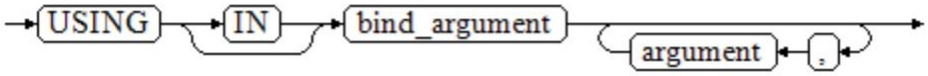
语法格式请参见下图。

noselect::=



using_clause 子句的语法参见图下图。

using_clause::=



参数说明

USING IN bind_argument：用于指定存放传递给动态 SQL 值的变量，在 dynamic_noselect_string 中存在占位符时使用，即动态 SQL 语句执行时，bind_argument 将替换相对应的占位符。

bind_argument 只能是值、变量或表达式，不能是表名、列名、数据类型等数据库对象。如果存储过程需要通过声明参数传递数据库对象来构造动态 SQL 语句（常见于执行 DDL 语句时），建议采用连接运算符“||”拼接 dynamic_select_clause。另外，动态语句允许出现重复的占位符，相同占位符只能与唯一一个 bind_argument 按位置一一对应。

示例

1、创建测试表。

```
CREATE TABLE sections_t1
(
 section    NUMBER(4) ,
 section_name  VARCHAR2(30),
 manager_id   NUMBER(6),
 place_id    NUMBER(4)
);
```

2、声明变量。

```
DECLARE
 section    NUMBER(4) := 280;
 section_name  VARCHAR2(30) := 'Info support';
 manager_id   NUMBER(6) := 103;
 place_id    NUMBER(4) := 1400;
 new_colname  VARCHAR2(10) := 'sec_name';
BEGIN
```

3、执行查询 。

```
EXECUTE IMMEDIATE 'insert into sections_t1 values(:1, :2, :3, :4)'
USING section, section_name, manager_id,place_id;
```

4、执行查询（重复占位符）。

```
EXECUTE IMMEDIATE 'insert into sections_t1 values(:1, :2, :3, :1)'
USING section, section_name, manager_id;
```

5、执行 ALTER 语句（建议采用“||”拼接数据库对象构造 DDL 语句）。

```
EXECUTE IMMEDIATE 'alter table sections_t1 rename section_name to ' || new_colna
me;
END;
/
```

6、查询数据。

```
SELECT * FROM sections_t1;
```

7、删除表。

```
DROP TABLE sections_t1;
```

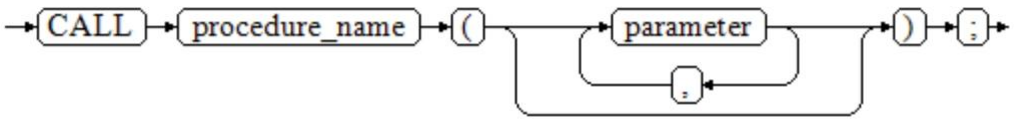
7.2.2.基本语句

7.2.2.1.调用语句

语法

调用一个语句的语法请参见下图。

call_clause::=



对以上语法格式的解释如下：

- procedure_name：存储过程名。
- parameter：存储过程的参数，可以没有或者有多个参数。

示例

1、创建表并插入测试数据。

```
create table staffs(
section_id int,
salary int
);
insert into staffs values(1,1000);
insert into staffs values(2,1000);
insert into staffs values(3,1000);
insert into staffs values(4,1000);
insert into staffs values(5,1000);
insert into staffs values(6,1000);
insert into staffs values(7,1000);
insert into staffs values(8,1000);
```

2、创建存储过程 proc_staffs 。

```
CREATE OR REPLACE PROCEDURE proc_staffs
(
section   NUMBER(6),
salary_sum out NUMBER(8,2),
staffs_count out INTEGER
)
IS
BEGIN
SELECT sum(salary), count(*) INTO salary_sum, staffs_count FROM staffs where sect
ion_id = section;
END;
/
```

3、调用存储过程 proc_return。

```
CALL proc_staffs(2,8,6);
```

结果显示如下：

```
 salary_sum | staffs_count
------------+--------------
       1000 |            1
```

```
(1 row)
```

4、清除存储过程 。

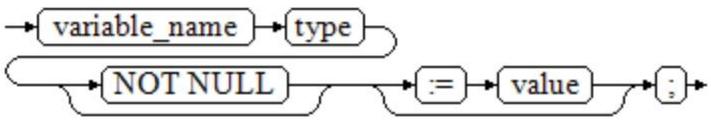
```
DROP PROCEDURE proc_staffs;
```

7.2.2.2.定义变量

介绍 PL/pgSQL 中变量的声明，以及该变量在代码中的作用域。

变量声明

变量声明语法请参见下图。



declare_variable::=

对以上语法格式的解释如下：

- variable_name：变量名。
- type：变量类型。
- value：该变量的初始值（如果不给定初始值，则初始为 NULL）。value 也可以是表达式。

变量类型除了支持基本类型，还可以使用%TYPE 和%ROWTYPE 去声明一些与其他表字段或表结构本身相关的变量。

- %TYPE 属性

%TYPE 主要用于声明某个与其他变量类型（例如，表中某列的类型）相同的变量。假如我们想定义一个 my_name 变量，它的变量类型与 employee 的 firstname 类型相同，我们可以通过如下定义：

```
my_name employee.firstname%TYPE
```

这样定义可以带来两个好处，首先，我们不用预先知道 employee 表的 firstname 类型具体是什么。其次，即使之后 firstname 类型有了变化，我们也不需要再次修改 my_name 的类型。

- %ROWTYPE 属性

%ROWTYPE 属性主要用于对一组数据的类型声明，用于存储表中的一行数据或从游标匹配的结果。假如，我们需要一组数据，该组数据的字段名称与字段类型都与 employee 表相同。我们可以通过如下定义：

```
my_employee employee%ROWTYPE
```

变量作用域

变量的作用域表示变量在代码块中的可访问性和可用性。只有在它的作用域内，变量才有效。

- 变量必须在 declare 部分声明，即必须建立 BEGIN-END 块。块结构也强制变量必须先声明后使用，即变量在过程内有不同作用域、不同的生存期。
- 同一变量可以在不同的作用域内定义多次，内层的定义会覆盖外层的定义。
- 在外部块定义的变量，可以在嵌套块中使用。但外部块不能访问嵌套块中的变量。

示例

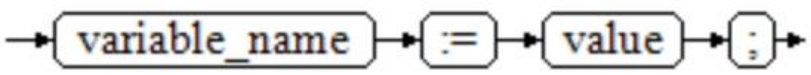
```
DECLARE
 emp_id  INTEGER := 7788;  --定义变量并赋值
BEGIN
 emp_id := 5*7784;  --变量赋值
END;
/
```

7.2.2.3.赋值语句

语法

给变量赋值的语法请参见下图。

assignment_value::=



对以上语法格式的解释如下：

- variable_name：变量名。
- value：可以是值或表达式。值 value 的类型需要和变量 variable_name 的类型兼容才能正确赋值。

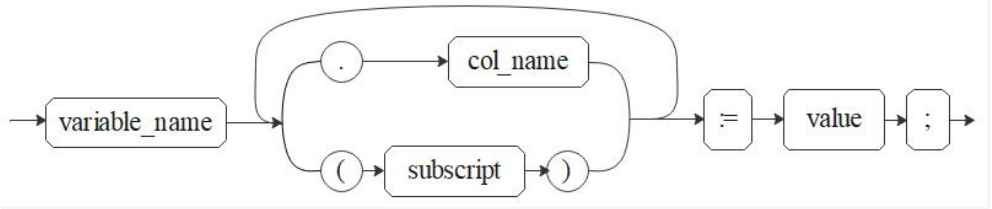
示例

```
DECLARE
 emp_id  INTEGER := 7788;--赋值
BEGIN
 emp_id := 5;--赋值
 emp_id := 5*7784;
END;
/
```

嵌套赋值

给变量嵌套赋值的语法请参见下图

nested_assignment_value::=



对以上语法格式的解释如下：

- variable_name：变量名。
- col_name：列名。
- subscript：下标，针对数组变量使用，可以是值或表达式，类型必须为 int。
- value：可以是值或表达式。值 value 的类型需要和变量 variable_name 的类型兼容才能正确赋值。

示例：

```
panweidb=#CREATE TYPE o1 as (a int, b int);
openGauss=# DECLARE
```

```
  TYPE r1 is VARRAY(10) of o1;
  emp_id  r1;
BEGIN
  emp_id(1).a := 5;--赋值
  emp_id(1).b := 5*7784;
END;
/
```

须知

INTO 方式赋值仅支持对第一层列赋值，且不支持二维及以上数组；

引用嵌套的列值时，若存在数组下标，目前仅支持在前三层列中只存在一个小括号情况，建议使用方括号[ ]引用下标；

INTO/BULK COLLECT INTO

将存储过程内语句返回的值存储到变量内，BULK COLLECT INTO 允许将部分或全部返回值暂存到数组内部。

示例：

```
panweidb=# DECLARE
  my_id integer;
BEGIN
  select id into my_id from customers limit 1; -- 赋值
END;
/

panweidb=# DECLARE
  type id_list is varray(6) of customers.id%type;
  id_arr id_list;
BEGIN
  select id bulk collect into id_arr from customers order by id DESC limit 20; -- 批量赋值
END;
/
```

须知

- BULK COLLECT INTO 只支持批量赋值给数组。合理使用 LIMIT 字段避免操作过量数据导致性能下降。
- 对于数组变量，小括号"()"将优先识别为下标，因此对于带括号的表达式，不支持写在数组变量后面。如对于 select (1+3) into va(5)，不支持写为 select into va(5) (1+3)或 select into va[5] (1+3)。

7.2.3.控制语句

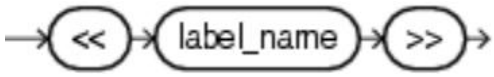
7.2.3.1.GOTO 语句

功能描述

GOTO 语句可以实现从 GOTO 位置到目标语句的无条件跳转。GOTO 语句会改变原本的执行逻辑，因此应该慎重使用，或者也可以使用 EXCEPTION 处理特殊场景。当执行 GOTO 语句时，目标 Label 必须是唯一的。

语法格式

label declaration ::=



goto statement ::=

注意事项

- 不支持有多个相同的 GOTO labels 目标场景，无论是否在同一个 block 中。

```
BEGIN
  GOTO pos1;
  <<pos1>>
```

```
  SELECT * FROM ...
  <<pos1>>
  UPDATE t1 SET ...
 END;
/
```

❖ 不支持 GOTO 跳转到 IF 语句、CASE 语句、LOOP 语句中。

```
BEGIN
   GOTO pos1;
   IF valid THEN
     <<pos1>>
     SELECT * FROM ...
   END IF;
  END;
/
```

❖ 不支持 GOTO 语句从一个 IF 子句跳转到另一个 IF 子句，或从一个 CASE 语句的 WHEN 子句跳转到另一个 WHEN 子句。

```
BEGIN
   IF valid THEN
     GOTO pos1;
     SELECT * FROM ...
   ELSE
     <<pos1>>
     UPDATE t1 SET ...
   END IF;
  END;
/
```

❖ 不支持从外部块跳转到内部的 BEGIN-END 块。

```
BEGIN
   GOTO pos1;
   BEGIN
     <<pos1>>
     UPDATE t1 SET ...
   END;
  END;
```

```
/
```

❖ 不支持从异常处理部分跳转到当前的 BEGIN-END 块。但可以跳转到上层 BEGIN-END 块。

```
BEGIN
   <<pos1>>
   UPDATE t1 SET ...
   EXCEPTION
    WHEN condition THEN
      GOTO pos1;
  END;
/
```

❖ 如果从 GOTO 到一个不包含执行语句的位置，需要添加 NULL 语句。

```
DECLARE
   done  BOOLEAN;
 BEGIN
   FOR i IN 1..50 LOOP
     IF done THEN
       GOTO end_loop;
     END IF;
     <<end_loop>>  -- not allowed unless an executable statement follows
     NULL; -- add NULL statement to avoid error
   END LOOP;  -- raises an error without the previous NULL
 END;
/
```

示例

1、创建测试表 test。

```
CREATE TABLE test(id,int);
```

2、创建匿名块进行测试，执行过程将从 GOTO 语句位置跳转到'pos1'标记位置执行。

```
BEGIN
GOTO pos1;
INSERT INTO test(id) VALUES(1);
```

```
<<pos1>>
INSERT INTO test(id) VALUES(2);
END;
/
```

3、验证结果。

```
SELECT * FROM test;
```

当结果返回如下信息，则表示跳转成功。

```
 id
----
  2
(1 row)
```

7.2.3.2.错误捕获语句

功能描述

缺省时，当 PL/pgSQL 函数执行过程中发生错误时退出函数执行，并且周围的事务也会回滚。可以用一个带有 EXCEPTION 子句的 BEGIN 块捕获错误并且从中恢复。

语法格式

```
[<<label>>]
[DECLARE
  declarations]
BEGIN
  statements
EXCEPTION
  WHEN condition [OR condition ...] THEN
    handler_statements
  [WHEN condition [OR condition ...] THEN
    handler_statements
  ...]
END;
```

参数说明

- condition：可以是 SQL 标准错误码编号说明的任意值。特殊的条件名 OTHERS 匹配除了 QUERY_CANCELED 之外的所有错误类型。
- handler_statements：捕获到错误语句后执行的操作。

注意事项

- 如果没有发生错误，这种形式的块儿只是简单地执行所有语句，然后转到 END 之后的下一个语句。但是如果在执行的语句内部发生了一个错误，则这个语句将会回滚，然后转到 EXCEPTION 列表。寻找匹配错误的第一个条件。若找到匹配，则执行对应的 handler_statements，然后转到 END 之后的下一个语句。如果没有找到匹配，则会向事务的外层报告错误，和没有 EXCEPTION 子句一样。错误码可以捕获同一类的其他错误码。也就是说该错误可以被一个包围块用 EXCEPTION 捕获，如果没有包围块，则进行退出函数处理。
- 如果在选中的 handler_statements 里发生了新错误，则不能被这个 EXCEPTION 子句捕获，而是向事务的外层报告错误。一个外层的 EXCEPTION 子句可以捕获它。
- 如果一个错误被 EXCEPTION 捕获，PL/pgSQL 函数的局部变量保持错误发生时的原值，但是所有该块中想写入数据库中的状态都回滚。

示例

- 示例 1：当控制到达给 y 赋值的地方时，会有一个 division_by_zero 错误失败。这个错误将被 EXCEPTION 子句捕获。而在 RETURN 语句里返回的数值将是 x 的增量值。

1、创建测试表并插入数据。

```
CREATE TABLE mytab(id INT,firstname VARCHAR(20),lastname VARCHAR(20)) ;
INSERT INTO mytab(firstname, lastname) VALUES('Tom', 'Jones');
```

2、创建函数并调用。

```
CREATE FUNCTION fun_exp() RETURNS INT
AS $$
DECLARE
```

```
  x INT :=0;
  y INT;
BEGIN
  UPDATE mytab SET firstname = 'Joe' WHERE lastname = 'Jones';
  x := x + 1;
  y := x / 0;
EXCEPTION
  WHEN division_by_zero THEN
    RAISE NOTICE 'caught division_by_zero';
    RETURN x;
END;$$
LANGUAGE plpgsql;

call fun_exp();
```

当结果显示如下信息，则表示捕获到错误语句。

```
NOTICE:  caught division_by_zero
 fun_exp
----------------
     1
(1 row)
```

3、查询表数据验证结果。

```
select * from mytab;
```

结果显示如下：

```
 id | firstname | lastname
----+-----------+----------
  | Tom       | Jones
(1 row)
```

4、删除测试表和函数。

```
DROP FUNCTION fun_exp();
DROP TABLE mytab;
```

- 进入和退出一个包含 EXCEPTION 子句的块要比不包含的块开销大的多。因此，不必要的时候不要使用 EXCEPTION。
- 在下列场景中，无法捕获处理异常，整个存储过程回滚：节点故障、网络故障引起的存储过程参与节点线程退出以及 COPY FROM 操作中源数据与目标表的表结构不一致造成的异常。
- 示例 2：UPDATE/INSERT 异常，这个例子根据使用异常处理器执行恰当的 UPDATE 或 INSERT 。

1、创建测试表。

```
CREATE TABLE db (a INT, b TEXT);
```

2、创建测试函数。

```
CREATE FUNCTION merge_db(key INT, data TEXT) RETURNS VOID AS
$$
BEGIN
  LOOP
    UPDATE db SET b = data WHERE a = key;                    --第一次尝试更新 key
    IF found THEN
      RETURN;
    END IF;
    BEGIN               --不存在，所以尝试插入 key，如果其他人同时插入相同的 key，
我们可能得到唯一 key 失败。
      INSERT INTO db(a,b) VALUES (key, data);
      RETURN;
    EXCEPTION WHEN unique_violation THEN
    END;                --什么也不做，并且循环尝试再次更新。
  END LOOP;
END;
$$
LANGUAGE plpgsql;
```

3、调用函数并查询数据进行对比。

```
SELECT merge_db(1, 'david');
SELECT * FROM db;
```

结果显示如下：

```
 a |  b
---+-------
 1 | david
(1 row)
```

再次调用并查询。

```
SELECT merge_db(1, 'dennis');
SELECT * FROM db;
```

结果显示如下：

```
 a |  b
---+--------
 1 | dennis
(1 row)
```

4、删除 FUNCTION 和 TABLE。

```
DROP FUNCTION merge_db;
DROP TABLE db;
```

7.2.3.3.RETURN 语句

功能描述

PanWeiDB 提供两种方式用于返回数据：RETURN，RETURN NEXT 或 RETURN QUERY。

注意事项

RETURN NEXT 和 RETURN QUERY 只适用于函数，不适用存储过程。

语法格式

- RETURN：用于终止函数并把 expression 的值返回给调用者。

```
    RETURN [expression];
```

- RETURN NEXT 及 RETURN QUERY：当需要函数返回一个集合时，使用 RETURN NEXT 或者 RETURN QUERY，可以向结果集追加结果，然后继续执行函数的下一条语句。

```
    RETURN NEXT expression;

    RETURN QUERY query;

    RETURN QUERY EXECUTE command-string [ USING expression [, ... ] ];
```

【须知】

RETURN QUERY 有一种变体 RETURN QUERY EXECUTE，它可以动态指定要被执行的查询。可以通过 USING 向计算出的查询字符串插入参数表达式，这和在 EXECUTE 命令中的方式相同。

示例

示例 1：在函数中使用 RETURN 语句。

1、创建函数。

```
create function test_f_1166349(i1 int, i2 int) return int

as

x varchar;

begin

select i1 + i2 into x;

return x;

end;

/
```

2、调用函数。

```
select test_f_1166349(1,55);

返回结果为：

 test_f_1166349

---------------

          56
```

```
(1 row)
```

示例 2：在函数中使用 RETURN QUERY 语句。

1、创建测试表并插入数据。

```
create table t_1166343(col1 int,col2 varchar);

insert into t_1166343 values(1, 'test'),(2, 'test2');
```

2、创建函数。

```
create function test_f_1166343() returns table(col1 int, col2 varchar)
as $$
begin
return query select * from t_1166343;
end;$$
LANGUAGE 'plpgsql';
```

3、调用函数。

```
select test_f_1166343();
返回结果为：
 test_f_1166343
----------------
 (1,test)
 (2,test2)
(2 rows)
```

7.2.3.4.空语句

功能描述

在 PL/pgSQL 程序中，可以用 NULL 语句来说明“不用做任何事情”，相当于一个占位符，可以使某些语句变得有意义，提高程序的可读性。

语法格式

```
DECLARE
 …
BEGIN
 …
 IF v_num IS NULL THEN
   NULL; -- 不需要处理任何数据。
 END IF;
END;
```

7.2.4.条件语句

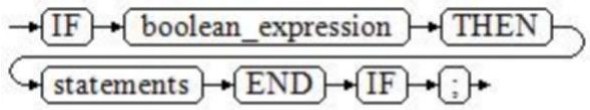
功能描述

条件语句的主要作用是判断参数或者语句是否满足已给定的条件，根据判定结果执行相应的操作。

语法格式

PanWeiDB 中支持如下多种形式的条件语句。

IF_THEN



IF_THEN 语句是 IF 的最简单形式。如果条件为真，statements 将被执行。否则，将忽略它们的结果使该 IF_THEN 语句执行结束。

示例：

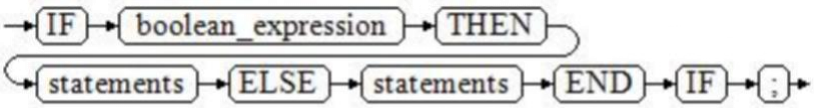
```
    declare
    n int;
    begin
    n :=95;
    raise info 'n:%',n;
```

```
if n>85 then raise info '优秀';
end if;
end;
/
```

返回结果为：

```
INFO:  n:95
INFO:  优秀
ANONYMOUS BLOCK EXECUTE
```

IF_THEN_ELSE



IF_THEN_ELSE 语句增加了 ELSE 的分支，可以声明在条件为假的时候执行的语句。

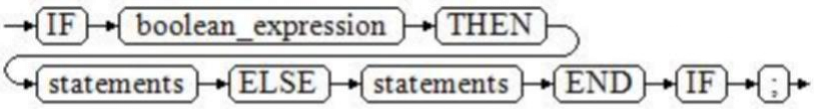
该语法还可以进行嵌套使用，嵌套方式如下所示：

```
IF sex = 'm' THEN
  pretty_sex := 'man';
ELSE
  IF sex = 'f' THEN
    pretty_sex := 'woman';
  END IF;
END IF;
```

【须知】

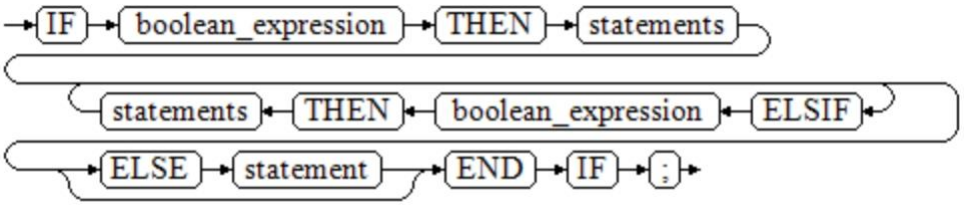
这种形式实际上就是在一个 IF 语句的 ELSE 部分嵌套了另一个 IF 语句。

IF_THEN_ELSE_IF



IF_THEN_ELSE 语句增加了 ELSE 的分支，可以声明在条件为假的时候执行的语句。

IF_THEN_ELSIF_ELSE



IF_THEN_ELSE 语句增加了 ELSE 的分支，可以声明在条件为假的时候执行的语句。

IF_THEN_ELSEIF_ELSE

ELSEIF 是 ELSIF 的别名。语法格式和用法均与 IF_THEN_ELSIF_ELSE 相同。

示例

示例 1：在函数中使用 IF 语句。

1、创建函数。

```
CREATE OR REPLACE FUNCTION FUN_1165174(n int )
return varchar2
AS
begin
raise info 'n:%',n;
if n>85 then raise info '优秀';
```

```
elseif n>70 then raise info '良好';
elseif n>=60 then raise info '合格';
else raise info '不合格';
end if;
return '标记情况结束！';
end;
/
```

2、调用函数。

```
call FUN_1165174(100);
call FUN_1165174(60);
call FUN_1165174(20);
```

返回结果分别为：

```
INFO：n:100
INFO：优秀
 fun_1165174
----------------
 标记情况结束！
(1 row)


INFO：n:60
INFO：合格
 fun_1165174
----------------
```

```
 标记情况结束!
(1 row)


INFO:  n:20
INFO:  不合格
 fun_1165174
----------------
 标记情况结束!
(1 row)
```

示例 2： 在匿名块中使用 IF 语句。

```
declare
n int;
begin
n :=95;
raise info 'n:%',n;
if n>85 then raise info '优秀';
elseif n>70 then raise info '良好';
elseif n>60 then raise info '合格';
else raise info '不合格';
end if;
raise info '标记情况结束！';
end;
/
```

返回结果为：

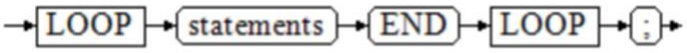
```
INFO:  n:95

INFO:  优秀

INFO:  标记情况结束！

ANONYMOUS BLOCK EXECUTE
```

7.2.4.1.循环语句

7.2.4.1.1. 简单 LOOP 语句

语法图

loop::=



示例

创建存储过程 proc_loop 并调用。

```
CREATE OR REPLACE PROCEDURE proc_loop(i in integer, count out integer)
AS
 BEGIN
  count:=0;
  LOOP
  IF count > i THEN
   raise info 'count is %. ', count;
    EXIT;
  ELSE
    count:=count+1;
  END IF;
  END LOOP;
  END;
```

```
/
CALL proc_loop(10,5);
```

当结果显示如下信息，则表示 loop 验证成功。

```
INFO:  count is 11.
 count
---------------------
   11
(1 row)
```

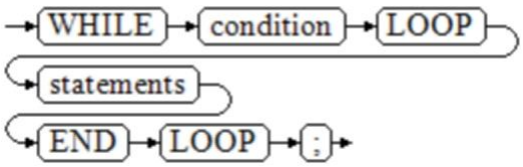
须知

该循环必须要结合 EXIT 使用，否则将陷入死循环。

7.2.4.1.2. WHILE_LOOP 语句

语法图

while_loop::=



只要条件表达式为真，WHILE 语句就会不停的在一系列语句上进行循环，在每次进入循环体的时候进行条件判断。

示例

创建存储过程并调用。

```
CREATE TABLE integertable(c1 integer) ;
CREATE OR REPLACE PROCEDURE proc_while_loop(maxval in integer)
AS
 DECLARE
```

```
 i int :=1;
 BEGIN
  WHILE i < maxval LOOP
    INSERT INTO integertable VALUES(i);
    i:=i+1;
  END LOOP;
 END;
/

CALL proc_while_loop(10);
```

查询表数据验证结果。

```
select * from integertable;
```

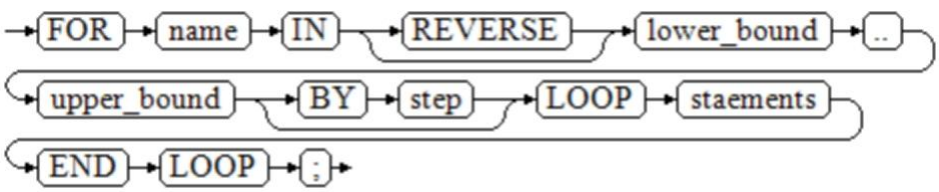
当结果显示如下信息，则表示循环插入成功。

```
 c1
-------
  1
  2
  3
  4
  5
  6
  7
  8
  9
(9 rows)
```

7.2.4.1.3. FOR_LOOP（integer 变量）语句

语法图

for_loop::=



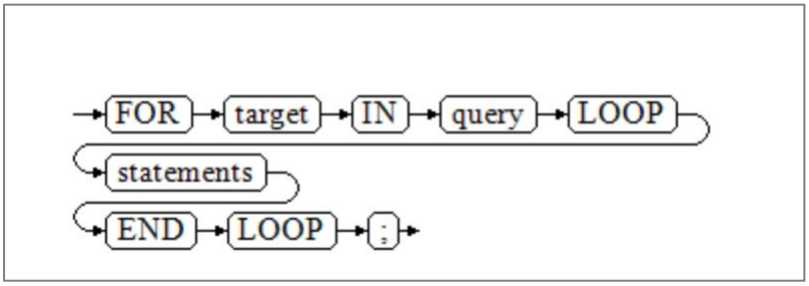
说明

- 变量 name 会自动定义为 integer 类型并且只在此循环里存在。变量 name 介于 lower_bound 和 upper_bound 之间。
- 当使用 REVERSE 关键字时，lower_bound 必须大于等于 upper_bound，否则循环体不会被执行。

7.2.4.1.4. FOR_LOOP 查询语句

语法图

for_loop_query::=



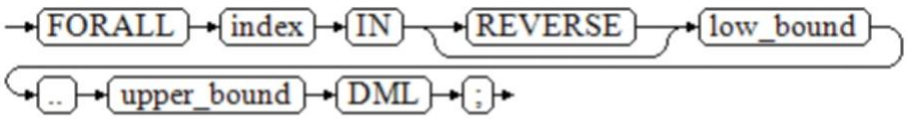
说明

变量 target 会自动定义，类型和 query 的查询结果的类型一致，并且只在此循环中有效。target 的取值就是 query 的查询结果。

7.2.4.1.5. FORALL 批量查询语句

语法图

forall::=



【说明】

变量 index 会自动定义为 integer 类型并且只在此循环里存在。index 的取值介于 low_bound 和 upper_bound 之间。

示例

1、创建测试表并插入数据。

```
CREATE TABLE hdfs_t1 (
 title NUMBER(6),
 did VARCHAR2(20),
 data_peroid VARCHAR2(25),
 kind VARCHAR2(25),
 interval VARCHAR2(20),
 time DATE,
 isModified VARCHAR2(10)
);

INSERT INTO hdfs_t1 VALUES( 8, 'Donald', 'OConnell', 'DOCONNEL', '650.507.9833', t
o_date('21-06-1999', 'dd-mm-yyyy'), 'SH_CLERK' );
```

2、创建存储过程并调用。

```
CREATE OR REPLACE PROCEDURE proc_forall()
AS
BEGIN
 FORALL i IN 100..120
 update hdfs_t1 set title = title + 100*i;
END;
/

CALL proc_forall();
```

3、查询存储过程调用结果

```
SELECT * FROM hdfs_t1 ;
```

当结果显示如下信息，则表示循环调用成功。

```
title  | did   | data_peroid |   kind   |  interval  |      time       | ismodified
--------+--------+-------------+----------+-------------+---------------------+------------
----
 231008 | Donald | OConnell    | DOCONNEL | 650.507.9833 | 1999-06-21 00:00:00 | SH
_CLERK
(1 row)
```

4、删除存储过程和表

```
DROP PROCEDURE proc_forall;
DROP TABLE hdfs_t1;
```

7.2.4.1.6. LABEL_LOOP 语句

语法格式

```
[label_begin:] LOOP
  statements
END LOOP [label_end]
```

【说明】

- 在简单 loop 语句的基础上增加了 label 标签的用法，标签规则如下：
- label_begin 可以单独出现（不加 label_end），但是使用 label_end，就必须有与之相同的 label_begin。
- 标签可以被 continue 或 exit 语句引用，特别的是，在数据库模式为'B'时，也可用 iterate 或 leave 语句。

【须知】

该循环只在数据库兼容模式为'B'时使用，其他模式下报错。必须要结合 EXIT 使用（'B'模式下可使用'LEAVE',与 EXIT 效果相同；'B'模式下可使用'ITERATE',与 CONTINUE 效果相同），否则将陷入死循环。

示例

```
CREATE OR REPLACE PROCEDURE label_loop(i in integer, count out integer)
AS
  BEGIN
    count:=0;
    label:
```

```
    LOOP
    IF count > i THEN
      raise info 'count is %. ', count;
      LEAVE;
    ELSE
      count:=count+1;
    END IF;
    END LOOP label;
  END;
/

CALL proc_loop(10,5);
```

7.2.5.锁操作

PanWeiDB 提供了多种锁模式用于控制对表中数据的并发访问。这些模式可以用在 MVCC（多版本并发控制）无法给出期望行为的场合。同样，大多数 PanWeiDB 命令自动施加恰当的锁，以保证被引用的表在命令的执行过程中不会以一种不兼容的方式被删除或者修改。比如，在存在其他并发操作的时候，ALTER TABLE 是不能在同一个表上执行的。

7.2.6.游标

为了处理 SQL 语句，存储过程进程分配一段内存区域来保存上下文联系。游标是指向上下文区域的句柄或指针。借助游标，存储过程可以控制上下文区域的变化。

当游标作为存储过程的返回值时，如果使用 JDBC 调用该存储过程，返回的游标将不可用。

游标的使用分为显式游标和隐式游标。对于不同的 SQL 语句，游标的使用情况不同，详细信息请参见下表

SQL 语句	游标
非查询语句	隐式的
结果是单行的查询语句	隐式的或显式的

SQL 语句	游标
结果是多行的查询语句	显式的

7.2.6.1.显式游标

功能描述

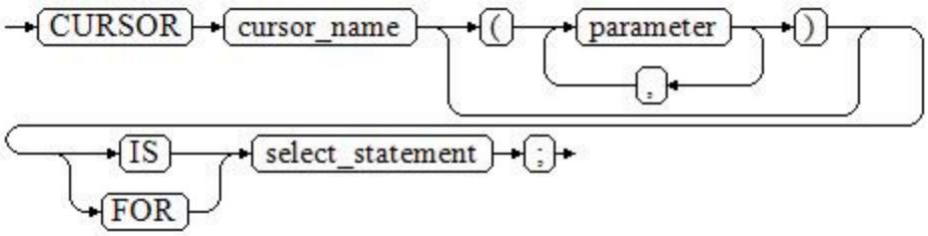
显式游标主要用于对查询语句的处理，尤其是在查询结果为多条记录的情况下。

处理步骤

显式游标处理需六个 PL/pgSQL 步骤（定义静态游标->打开静态游标->提取游标数据->对该记录进行处理->继续处理->关闭游标）如下所示：

1、定义静态游标：就是定义一个游标名，以及与其相对应的 SELECT 语句。

定义静态游标的语法图，请参见下图：



static_cursor_define::=

参数说明：

- cursor_name：

  定义的游标名。

- parameter：

  游标参数，只能为输入参数，其格式为：

  ```
  parameter_name datatype。
  ```

- select_statement：

  查询语句。

  【说明】

  - 根据执行计划的不同，系统会自动判断该游标是否可以用于以倒序的方

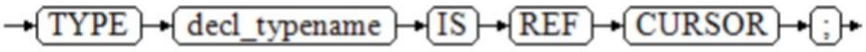
式检索数据行。

- 语法上支持 parameter 为输出参数，但其行为与输入参数保持一致。

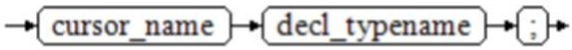
定义动态游标：指 ref 游标，可以通过一组静态的 SQL 语句动态的打开游标。首先定义 ref 游标类型，然后定义该游标类型的游标变量，在打开游标时通过 OPEN FOR 动态绑定 SELECT 语句。

定义动态游标的语法图，请参见下图。

cursor_typename::=



dynamic_cursor_define::=

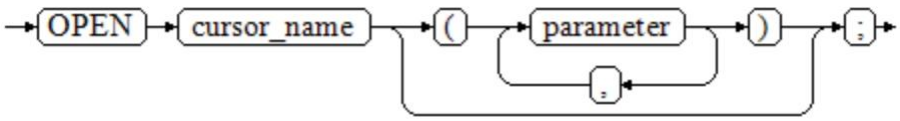


2、打开静态游标：

- 在 A 兼容模式下，就是执行游标所对应的 SELECT 语句，将其查询结果放入工作区，并且指针指向工作区的首部，标识游标结果集合。如果游标查询语句中带有 FOR UPDATE 选项，OPEN 语句还将锁定数据库表中游标结果集合对应的数据行。
- 在其他兼容模式下，仅对表加 RowShareLock，如果游标查询语句中带有 FOR UPDATE 选项，打开静态游标不会锁定游标结果集中对应的数据行，提取游标数据时才会对结果集的数据行加排它锁。

打开静态游标的语法图，请参见下图

open_static_cursor::=



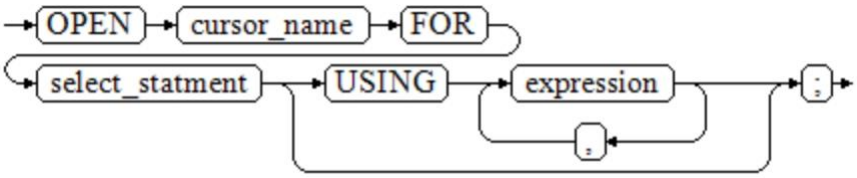
打开动态游标：可以通过 OPEN FOR 语句打开动态游标，动态绑定 SQL 语句。

打开动态游标的语法图，请参见下图。

open_dynamic_cursor::=

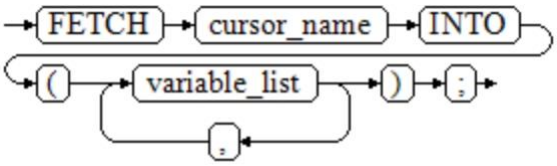
PL/pgSQL 程序不能用 OPEN 语句重复打开一个游标。

3、提取游标数据：检索结果集合中的数据行，放入指定的输出变量中。



提取游标数据的语法图，请参见下图。

fetch_cursor::=



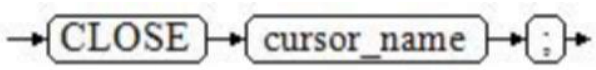
4、对该记录进行处理。

5、继续处理，直到活动集合中没有记录。

6、关闭游标：当提取和处理完游标结果集合数据后，应及时关闭游标，以释放该游标所占用的系统资源，并使该游标的工作区变成无效，不能再使用 FETCH 语句获取其中数据。关闭后的游标可以使用 OPEN 语句重新打开。

关闭游标的语法图，请参见下图。

close_cursor::=



属性

游标的属性用于控制程序流程或者了解程序的状态。当运行 DML 语句时，PL/pgSQL 打开一个内建游标并处理结果，游标是维护查询结果的内存中的一个区域，游标在运行 DML 语句时打开，完成后关闭。显式游标的属性为：

- %FOUND 布尔型属性：当最近一次读记录时成功返回，则值为 TRUE。
- %NOTFOUND 布尔型属性：与%FOUND 相反。
- %ISOPEN 布尔型属性：当游标已打开时返回 TRUE。
- %ROWCOUNT 数值型属性：返回已从游标中读取的记录数。

7.2.6.2.隐式游标

功能描述

对于非查询语句，如修改、删除操作，则由系统自动地为这些操作设置游标并创建其工作区，这些由系统隐含创建的游标称为隐式游标，隐式游标的名称为 SQL，这是由系统定义的。

对于隐式游标的操作，如定义、打开、取值及关闭操作，都由系统自动地完成，无需用户进行处理。用户只能通过隐式游标的相关属性，来完成相应的操作。在隐式游标的工作区中，所存放的数据是最新处理的一条 SQL 语句所包含的数据，与用户自定义的显式游标无关。

语法格式

格式调用为： SQL%

说明

- INSERT，UPDATE，DELETE，SELECT 语句中不必明确定义游标。

- 兼容 O 模式下，GUC 参数 behavior_compat_options 为 compat_cursor 时，隐式游标跨存储过程有效。

属性

隐式游标属性为：

- SQL%FOUND 布尔型属性：当最近一次读记录时成功返回，则值为 TRUE。
- SQL%NOTFOUND 布尔型属性：与%FOUND 相反。
- SQL%ROWCOUNT 数值型属性：返回已从游标中读取得记录数。
- SQL%ISOPEN 布尔型属性：取值总是 FALSE。SQL 语句执行完毕立即关闭隐式游标。

示例

1、创建测试表 staffs。

```
create table staffs(
section_id int,
salary int
);
insert into staffs values(1,1000);
insert into staffs values(2,1000);
insert into staffs values(3,1000);
insert into staffs values(4,1000);
insert into staffs values(5,1000);
insert into staffs values(6,1000);
insert into staffs values(7,1000);
insert into staffs values(8,1000);
```

2、创建测试表 sections。

```
create table sections(
section_id int,
salary int
);
insert into sections values(1,1000);
```

```
insert into sections values(2,1000);
insert into sections values(3,1000);
insert into sections values(4,1000);
insert into sections values(5,1000);
insert into sections values(6,1000);
insert into sections values(7,1000);
```

3、删除员工表 staffs 中某部门的所有员工，如果该部门中已没有员工，则在部门表 section 中删除该部门。

```
CREATE OR REPLACE PROCEDURE proc_cursor1()
AS
 DECLARE
 V_DEPTNO NUMBER(4) := 100;
 BEGIN
DELETE FROM staffs WHERE section_ID = V_DEPTNO;
  --根据游标状态做进一步处理
  IF SQL%NOTFOUND THEN
  DELETE FROM sections WHERE section_ID = V_DEPTNO;
 END IF;
END;
/
CALL proc_cursor1();
```

4、删除存储过程和表 。

```
DROP PROCEDURE proc_cursor1;
DROP TABLE sections;
DROP TABLE staffs;
```

7.2.6.3.游标循环

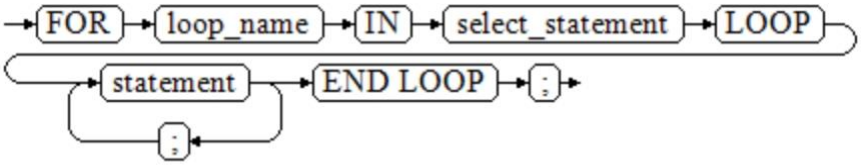
功能描述

游标在 WHILE 语句、LOOP 语句中的使用称为游标循环，一般这种循环都需要使用 OPEN、FETCH 和 CLOSE 语句。下面要介绍的一种循环不需要这些操作，可以简化游标循环的操作，这种循环方式适用于静态游标的循环，不用执行静态游标的四个步骤。

语法

FOR AS 循环的语法请参见下图。

FOR_AS_loop::=



注意事项

- 不能在该循环语句中对查询的表进行更新操作。
- 变量 loop_name 会自动定义且只在此循环中有效，类型和 select_statement 的查询结果类型一致。loop_name 的取值就是 select_statement 的查询结果。
- 游标的属性中%FOUND、%NOTFOUND、%ROWCOUNT 在 PanWeiDB 数据库中都是访问同一个内部变量，事务和匿名块不支持多个游标同时访问。

7.2.7.record

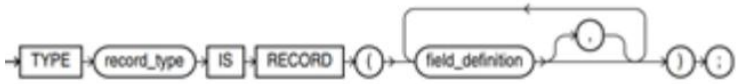
record 类型的变量

创建一个 record 变量的方式：定义一个 record 类型，然后使用该类型来声明一个变量。

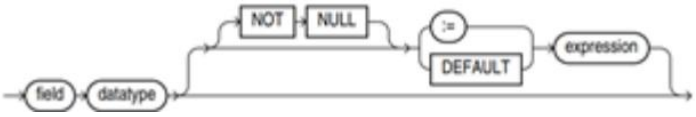
语法

record 类型的语法参见下图：

record_type_definition ::=



field_definition ::=



对以上语法格式的解释如下：

- record_type：声明的类型名称。
- field：record 类型中的成员名称。
- datatype：record 类型中成员的类型。
- expression：设置默认值的表达式。

说明

在 PanWeiDB 中：

- record 类型变量的赋值支持：
  - 在函数或存储过程的声明阶段，声明一个 record 类型，并且可以在该类型中定义成员变量。
  - 一个 record 变量到另一个 record 变量的赋值。
  - SELECT INTO 和 FETCH 向一个 record 类型的变量中赋值。
  - 将一个 NULL 值赋值给一个 record 变量。
- 不支持 INSERT 和 UPDATE 语句使用 record 变量进行插入数据和更新数据。

- 如果成员有复合类型，在声明阶段不支持指定默认值，该行为同声明阶段的变量一样。

示例

1、创建测试表。

```
CREATE TABLE emp_rec(empno numeric(4,0),ename character varying(10), job char
acter varying(9), mgr numeric(4,0),hiredate timestamp(0) without time zone,sal nu
meric(7,2),comm numeric(7,2),deptno numeric(2,0));
```

2、查看表定义。

```
\d emp_rec
```

结果返回如下：

```
      Table "public.emp_rec"
 Column  |       Type         | Modifiers
---------+----------------------------------+-----------
 empno   | numeric(4,0)       | not null
 ename   | character varying(10) |
 job     | character varying(9)  |
 mgr     | numeric(4,0)       |
 hiredate | timestamp(0) without time zone |
 sal     | numeric(7,2)       |
 comm    | numeric(7,2)       |
 deptno  | numeric(2,0)       |
```

3、演示在存储过程中对数组进行操作。

```
CREATE OR REPLACE FUNCTION regress_record(p_w VARCHAR2)
RETURNS
CREATE OR REPLACE FUNCTION regress_record(p_w VARCHAR2)
RETURNS
VARCHAR2  AS $$
DECLARE
```

```
--声明一个 record 类型.
type rec_type is record (name  varchar2(100), epno int);
employer rec_type;

--使用%type 声明 record 类型
type rec_type1 is record (name  emp_rec.ename%type, epno int not null :=10);
employer1 rec_type1;

--声明带有默认值的 record 类型
type rec_type2 is record (name varchar2 not null := 'SCOTT',epno int not null :=10);
employer2 rec_type2;
CURSOR C1 IS  select ename,empno from emp_rec order by 1 limit 1;

BEGIN
--对一个 record 类型的变量的成员赋值。
employer.name := 'WARD';
employer.epno = 18;
raise info 'employer name: % , epno:%', employer.name, employer.epno;

-将一个 record 类型的变量赋值给另一个变量。
employer1 := employer;
raise info 'employer1 name: % , epno: %',employer1.name, employer1.epno;

 --将一个 record 类型变量赋值为 NULL。
 employer := NULL;
raise info 'employer name: % , epno: %',employer.name, employer.epno;

    --获取 record 变量的默认值。
 raise info 'employer2 name: % ,epno: %', employer2.name, employer2.epno;

 --在 for 循环中使用 record 变量
 for employer in select ename,empno from emp_rec order by 1  limit 1
  loop
raise info 'employer name: % , epno: %', employer.name, employer.epno;
```

```
 end loop;

 --在 cursor 中使用 record 变量。
OPEN C1;
FETCH C1 INTO employer2;
raise info 'employer name: % , epno: %', employer2.name, employer2.epno;
CLOSE C1;
RETURN employer.name;
END;
$$
LANGUAGE plpgsql;
```

4、调用该存储过程。

```
CALL regress_record('abc');
```

结果显示如下：

```
INFO:  employer name: WARD , epno:18
INFO:  employer1 name: WARD , epno: 18
INFO:  employer name: <NULL> , epno: <NULL>
INFO:  employer2 name: SCOTT ,epno: 10
INFO:  employer name: SCOTT , epno: 10
 regress_record
----------------


(1 row)
```

5、删除存储过程。

```
DROP PROCEDURE regress_record;
```

7.2.8.Retry 管理

功能描述

Retry 是数据库在 SQL 或存储过程（包含匿名块）执行失败时，在数据库内部进行重新执行的过程，以提高执行成功率和用户体验。数据库内部通过检查发生错误时的错误码及 Retry 相关配置，决定是否进行重试。

注意事项

失败时回滚之前执行的语句，并重新执行存储过程进行 Retry。

示例

```
CREATE OR REPLACE PROCEDURE retry_basic ( IN  x INT)
AS
BEGIN
    INSERT INTO t1 (a) VALUES (x);
    INSERT INTO t1 (a) VALUES (x+1);
END;
/

CALL retry_basic(1);
```

7.2.9.事务管理

功能描述

存储过程本身就处于一个事务中，开始调用最外围存储过程时会自动开启一个事务，在调用结束时自动提交或者发生异常时回滚。除了系统自动的事务控制外，也可以使用 COMMIT/ROLLBACK 来控制存储过程中的事务。在存储过程中调用 COMMIT/ROLLBACK 命令，将提交/回滚当前事务并自动开启一个新的事务，后续的所有操作都会在此新事务中运行。

保存点 SAVEPOINT 是事务中的一个特殊记号，它允许将那些在它建立后执行的命令全部回滚，把事务的状态恢复到保存点所在的时刻。存储过程中允许使用保存点来进行事务管理，当前支持保存点的创建、回滚和释放操作。存储过程中使用回滚保存点只是回退当前事务的修改，而不会改变存储过程的执行流程，也不会回退存储过程中的局部变量值等。

语法格式

❖ 定义保存点。

```
SAVEPOINT savepoint_name;
```

❖ 回滚保存点。

```
ROLLBACK TO [SAVEPOINT] savepoint_name;
```

- 释放保存点。

```
RELEASE [SAVEPOINT] savepoint_name;
```

使用场景

支持调用的上下文环境：

- 支持在 PLSQL 的存储过程内使用 COMMIT/ROLLBACK/SAVEPOINT。
- 支持含有 EXCEPTION 的存储过程使用 COMMIT/ROLLBACK/SAVEPOINT。
- 支持在存储过程的 EXCEPTION 语句内使用 COMMIT/ROLLBACK/SAVEPOINT。
- 支持在事务块里调用含有 COMMIT/ROLLBACK/SAVEPOINT 的存储过程，即通过/BEGIN/START/END 等开启控制的外部事务。
- 支持在子事务中调用含有 SAVEPOINT 的存储过程，即存储过程中使用外部定义的 SAVEPOINT，回退事务状态到存储过程外定义的 SAVEPOINT 位置。
- 支持存储过程外部对存储过程内定义的 SAVEPOINT 可见，即存储过程外可以将事务修改回滚到存储过程中定义 SAVEPOINT 的位置。
- 支持多数 PLSQL 的上下文和语句内调用 COMMIT/ROLLBACK/SAVEPOINT，包括常用的 IF/FOR/CURSOR LOOP/WHILE。
- 支持存储过程返回值与简单表达式计算中调用含有 COMMIT/ROLLBACK/SAVEPOINT 的存储过程或者函数。

支持提交/回滚的内容：

- 支持 DDL 在 COMMIT/ROLLBACK 后的提交/回滚。
- 支持 DML 的 COMMIT/ROLLBACK 后的提交。
- 支持存储过程内 GUC 参数的回滚提交。

注意事项

不支持调用的上下文环境：

- 不支持除 PLSQL 的其他存储过程中调用 COMMIT/ROLLBACK/SAVEPOINT，例如 PLJAVA、PLPYTHON 等。
- 不支持函数中调用 COMMIT/ROLLBACK/SAVEPOINT，包括函数调用含有 COMMIT/ROLLBACK/SAVEPOINT 的存储过程。
- 不支持事务块中调用了 SAVEPOINT 后，调用含有 COMMIT/ROLLBACK 的存储过程。
- 不支持 TRIGGER 中调用含有 COMMIT/ROLLBACK/SAVEPOINT 语句的存储过程。
- 不支持 EXECUTE 语句中调用 COMMIT/ROLLBACK/SAVEPOINT 语句。
- 不支持在 CURSOR 语句中打开一个含有 COMMIT/ROLLBACK/SAVEPOINT 的存储过程。
- 不支持带有 IMMUTABLE 以及 SHIPPABLE 的存储过程调用 COMMIT/ROLLBACK/SAVEPOINT，或调用带有 COMMIT/ROLLBACK/SAVEPOINT 语句的存储过程。
- 不支持 SQL 中调用含有 COMMIT/ROLLBACK/SAVEPOINT 语句的存储过程，除了 SELECT PROC 以及 CALL PROC。
- 存储过程头带有 GUC 参数设置的不允许调用 COMMIT/ROLLBACK/SAVEPOINT 语句。
- 不支持 CURSOR/EXECUTE 语句，以及各类表达式内调用 COMMIT/ROLLBACK/SAVEPOINT。
- 自治事务和存储过程事务是两个独立的事务，不能互相使用对方事务中定义的保存点。
- 不支持存储过程中释放存储过程外部定义的保存点。

不支持提交回滚的内容：

- 不支持存储过程内声明变量以及传入变量的提交/回滚。
- 不支持存储过程内必须重启生效的 GUC 参数的提交/回滚。

示例

示例 1：支持在 PLSQL 的存储过程内使用 COMMIT/ROLLBACK。

```
CREATE TABLE EXAMPLE1(COL1 INT);
CREATE OR REPLACE PROCEDURE TRANSACTION_EXAMPLE()
AS
BEGIN
  FOR i IN 0..20 LOOP
    INSERT INTO EXAMPLE1(COL1) VALUES (i);
    IF i % 2 = 0 THEN
      COMMIT;
    ELSE
      ROLLBACK;
    END IF;
  END LOOP;
END;
/
```

示例 2：

支持含有 EXCEPTION 的存储过程使用 COMMIT/ROLLBACK。

支持在存储过程的 EXCEPTION 语句内使用 COMMIT/ROLLBACK。

支持 DDL 在 COMMIT/ROLLBACK 后的提交/回滚。

```
CREATE OR REPLACE PROCEDURE TEST_COMMIT_INSERT_EXCEPTION_ROLLBACK()
AS
BEGIN
  DROP TABLE IF EXISTS TEST_COMMIT;
  CREATE TABLE TEST_COMMIT(A INT, B INT);
  INSERT INTO TEST_COMMIT SELECT 1, 1;
  COMMIT;
    CREATE TABLE TEST_ROLLBACK(A INT, B INT);
  RAISE EXCEPTION 'RAISE EXCEPTION AFTER COMMIT';
 EXCEPTION
  WHEN OTHERS THEN
  INSERT INTO TEST_COMMIT SELECT 2, 2;
  ROLLBACK;
END;
/
```

示例 3：支持在事务块里调用含有 COMMIT/ROLLBACK 的存储过程，即通过/BEGIN/START/END 等开启控制的外部事务。

```
BEGIN;
  CALL TEST_COMMIT_INSERT_EXCEPTION_ROLLBACK();
END;
```

示例 4：支持多数 PLSQL 的上下文和语句内调用 COMMIT/ROLLBACK，包括常用的 IF/FOR/CURSOR LOOP/WHILE。

```
CREATE OR REPLACE PROCEDURE TEST_COMMIT2()
IS
BEGIN
  DROP TABLE IF EXISTS TEST_COMMIT;
  CREATE TABLE TEST_COMMIT(A INT);
  FOR I IN REVERSE 3..0 LOOP
  INSERT INTO TEST_COMMIT SELECT I;
  COMMIT;
  END LOOP;
  FOR I IN REVERSE 2..4 LOOP
  UPDATE TEST_COMMIT SET A=I;
  COMMIT;
  END LOOP;
EXCEPTION
WHEN OTHERS THEN
  INSERT INTO TEST_COMMIT SELECT 4;
  COMMIT;
END;
/
```

示例 5：支持存储过程返回值与简单表达式计算。

```
CREATE OR REPLACE PROCEDURE exec_func3(RET_NUM OUT INT)
AS
BEGIN
 RET_NUM := 1+1;
COMMIT;
END;
/
```

```
CREATE OR REPLACE PROCEDURE exec_func4(ADD_NUM IN INT)
AS
SUM_NUM INT;
BEGIN
SUM_NUM := ADD_NUM + exec_func3();
COMMIT;
END;
/
```

示例 6：支持存储过程内 GUC 参数的回滚提交。

1、查询参数值。

```
SHOW explain_perf_mode;
SHOW enable_force_vector_engine;
```

2、创建存储过程修改参数。

```
CREATE OR REPLACE PROCEDURE GUC_ROLLBACK()
AS
BEGIN
  SET enable_force_vector_engine = on;
  COMMIT;
  SET explain_perf_mode TO pretty;
  ROLLBACK;
END;
/
```

3、调用存储过程并查询。

```
call GUC_ROLLBACK();
SHOW explain_perf_mode;
SHOW enable_force_vector_engine;
```

结果显示如下：

```
 explain_perf_mode
-------------------
 normal
(1 row)
```

```
enable_force_vector_engine
---------------------------
 on
(1 row)
```

4、将参数改回。

```
SET enable_force_vector_engine = off;
```

示例 7：函数（Function）中不允许调用 commit/rollback 语句

```
CREATE OR REPLACE FUNCTION FUNCTION_EXAMPLE1() RETURN INT
AS
EXP INT;
BEGIN
  FOR i IN 0..20 LOOP
    INSERT INTO EXAMPLE1(col1) VALUES (i);
    IF i % 2 = 0 THEN
      COMMIT;
    ELSE
      ROLLBACK;
    END IF;
  END LOOP;
  SELECT COUNT(*) FROM EXAMPLE1 INTO EXP;
  RETURN EXP;
END;
/
```

示例 8：函数（Fucntion）中不允许调用带有 commit/rollback 语句的存储过程。

```
CREATE OR REPLACE FUNCTION FUNCTION_EXAMPLE2() RETURN INT
AS
EXP INT;
BEGIN
  --transaction_example 为存储过程，带有 commit/rollback 语句
  CALL transaction_example();
  SELECT COUNT(*) FROM EXAMPLE1 INTO EXP;
  RETURN EXP;
END;
```

```
/
```

示例 9：不允许 Trigger 的存储过程包含 commit/rollback 语句，或调用带有 commit/rollback 语句的存储过程。

```
CREATE OR REPLACE FUNCTION FUNCTION_TRI_EXAMPLE2() RETURN TRIGGER
AS
EXP INT;
BEGIN
  FOR i IN 0..20 LOOP
    INSERT INTO EXAMPLE1(col1) VALUES (i);
    IF i % 2 = 0 THEN
      COMMIT;
    ELSE
      ROLLBACK;
    END IF;
  END LOOP;
  SELECT COUNT(*) FROM EXAMPLE1 INTO EXP;

END;
/
CREATE TRIGGER TRIGGER_EXAMPLE AFTER DELETE ON EXAMPLE1
FOR EACH ROW EXECUTE PROCEDURE FUNCTION_TRI_EXAMPLE2();

DELETE FROM EXAMPLE1;
```

示例 10：不支持带有 IMMUABLE 以及 SHIPPABLE 的存储过程调用 commit/rollback，或调用带有 commit/rollback 语句的存储过程。

```
CREATE OR REPLACE PROCEDURE TRANSACTION_EXAMPLE1()
IMMUTABLE
AS
BEGIN
  FOR i IN 0..20 LOOP
    INSERT INTO EXAMPLE1 (col1) VALUES (i);
    IF i % 2 = 0 THEN
      COMMIT;
    ELSE
```

```
      ROLLBACK;
    END IF;
  END LOOP;
END;
/
```

示例 11：不支持出现在 SQL 中的调用（除了 Select Procedure）。

```
CREATE OR REPLACE PROCEDURE TRANSACTION_EXAMPLE3()
AS
BEGIN
  FOR i IN 0..20 LOOP
    INSERT INTO EXAMPLE1 (col1) VALUES (i);
    IF i % 2 = 0 THEN
      EXECUTE IMMEDIATE 'COMMIT';
    ELSE
      EXECUTE IMMEDIATE 'ROLLBACK';
    END IF;
  END LOOP;
END;
/
```

示例 12：存储过程头带有 GUC 参数设置的不允许调用 commit/rollback 语句。

```
CREATE OR REPLACE PROCEDURE TRANSACTION_EXAMPLE4()
SET ARRAY_NULLS TO "ON"
AS
BEGIN
  FOR i IN 0..20 LOOP
    INSERT INTO EXAMPLE1 (col1) VALUES (i);
    IF i % 2 = 0 THEN
      COMMIT;
    ELSE
      ROLLBACK;
    END IF;
  END LOOP;
END;
/
```

示例 13：游标 open 的对象不允许为带有 commit/rollback 语句的存储过程。

```
CREATE OR REPLACE PROCEDURE TRANSACTION_EXAMPLE5(INTIN IN INT, INTOUT OU
T INT)
AS
BEGIN
INTOUT := INTIN + 1;
COMMIT;
END;
/

CREATE OR REPLACE PROCEDURE TRANSACTION_EXAMPLE6()
AS
CURSOR CURSOR1(EXPIN INT)
IS SELECT TRANSACTION_EXAMPLE5(EXPIN);
INTEXP INT;
BEGIN
  FOR i IN 0..20 LOOP
    OPEN CURSOR1(i);
    FETCH CURSOR1 INTO INTEXP;
    INSERT INTO EXAMPLE1(COL1) VALUES (INTEXP);
    IF i % 2 = 0 THEN
      COMMIT;
    ELSE
      ROLLBACK;
    END IF;
    CLOSE CURSOR1;
  END LOOP;
END;
/
```

示例 14：不支持 CURSOR/EXECUTE 语句，以及各类表达式内调用 COMMIT/ROLLBACK。

```
CREATE OR REPLACE PROCEDURE exec_func1()
AS
```

```
BEGIN
  CREATE TABLE TEST_exec(A INT);
COMMIT;
END;
/
CREATE OR REPLACE PROCEDURE exec_func2()
AS
BEGIN
EXECUTE exec_func1();
COMMIT;
END;
/
```

示例 15：存储过程使用保存点回退事务部分修改。

```
CREATE OR REPLACE PROCEDURE STP_SAVEPOINT_EXAMPLE1()
AS
BEGIN
  INSERT INTO EXAMPLE1 VALUES(1);
  SAVEPOINT s1;
  INSERT INTO EXAMPLE1 VALUES(2);
  ROLLBACK TO s1; -- 回退插入记录 2
  INSERT INTO EXAMPLE1 VALUES(3);
END;
/
```

示例 16：存储过程中使用保存点回退到存储过程外部定义的保存点。

```
CREATE OR REPLACE PROCEDURE STP_SAVEPOINT_EXAMPLE2()
AS
BEGIN
  INSERT INTO EXAMPLE1 VALUES(2);
  ROLLBACK TO s1; -- 回退插入记录 2
  INSERT INTO EXAMPLE1 VALUES(3);
END;
/

BEGIN;
INSERT INTO EXAMPLE1 VALUES(1);
```

```
SAVEPOINT s1;
CALL STP_SAVEPOINT_EXAMPLE2();
SELECT * FROM EXAMPLE1;
COMMIT;
```

示例 17：存储过程外部回退到存储过程中定义的保存点。

```
CREATE OR REPLACE PROCEDURE STP_SAVEPOINT_EXAMPLE3()
AS
BEGIN
  INSERT INTO EXAMPLE1 VALUES(1);
  SAVEPOINT s1;
  INSERT INTO EXAMPLE1 VALUES(2);
END;
/

BEGIN;
INSERT INTO EXAMPLE1 VALUES(3);
CALL STP_SAVEPOINT_EXAMPLE3();
ROLLBACK TO SAVEPOINT s1; --回退存储过程中插入记录 2
SELECT * FROM EXAMPLE1;
COMMIT;
```

示例 18：不支持存储过程中释放存储过程外部定义的保存点。

```
CREATE OR REPLACE PROCEDURE STP_SAVEPOINT_EXAMPLE3()
 AS
BEGIN
  INSERT INTO EXAMPLE1 VALUES(2);
  RELEASE SAVEPOINT s1; -- 释放存储过程外部定义的保存点
  INSERT INTO EXAMPLE1 VALUES(3);
  END;
/

BEGIN;
INSERT INTO EXAMPLE1 VALUES(1);
SAVEPOINT s1;
 CALL STP_SAVEPOINT_EXAMPLE3();
COMMIT;
```

7.2.10.数据类型

数据类型是一组值的集合以及定义在这个值集上的一组操作。PanWeiDB 数据库是由表的集合组成的，而各表中的列定义了该表，每一列都属于一种数据类型，PanWeiDB 根据数据类型有相应函数对其内容进行操作，例如 PanWeiDB 可对数值型数据进行加、减、乘、除操作。

数据类型转换

PanWeiDB 数据库中允许有些数据类型进行隐式类型转换（赋值、函数调用的参数等）；有些数据类型不允许进行隐式数据类型转换，可尝试使用 PanWeiDB 提供的类型转换函数，例如 CAST 进行数据类型强转。

PanWeiDB 数据库常见的隐式类型转换如下表所示：

【须知】

PanWeiDB 支持的 DATE 的效限范围是：公元前 4713 年到公元 294276 年。

原始数据类型	目标数据类型	备注
CHAR	VARCHAR2	-
CHAR	NUMBER	原数据必须由数字组成。
CHAR	DATE	原数据不能超出合法日期范围。
CHAR	RAW	-
CHAR	CLOB	-
VARCHAR2	CHAR	-
VARCHAR2	NUMBER	原数据必须由数字组成。
VARCHAR2	DATE	原数据不能超出合法日期范围。
VARCHAR2	CLOB	-
NUMBER	CHAR	-
NUMBER	VARCHAR2	-
DATE	CHAR	-
DATE	VARCHAR2	-
RAW	CHAR	-
RAW	VARCHAR2	-
CLOB	CHAR	-
CLOB	VARCHAR2	-
CLOB	NUMBER	原数据必须由数字组成。

原始数据类型	目标数据类型	备注
INT4	CHAR	-
INT4	BOOLEAN	-
BOOLEAN	INT4	

7.2.11.数组

数组类型的使用

在使用数组之前，需要自定义一个数组类型。

在存储过程中紧跟 AS 关键字后面定义数组类型。定义方法为：

```
TYPE array_type IS VARRAY(size) OF data_type;
```

其中：

- array_type：要定义的数组类型名。
- VARRAY：表示要定义的数组类型。
- size：取值为正整数，表示可以容纳的成员的最大数量。
- data_type：要创建的数组中成员的类型。
- 在 PanWeiDB 中，数组会自动增长，访问越界会返回一个 NULL，不会报错。
- 在存储过程中定义的数组类型，其作用域仅在该存储过程中。
- 建议选择上述定义方法的一种来自定义数组类型，当同时使用两种方法定义同名的数组类型时，PanWeiDB 会优先选择存储过程中定义的数组类型来声明数组变量。

PanWeiDB 支持使用圆括号来访问数组元素，且还支持一些特有的函数，如 extend，count，first，last 来访问数组的内容。

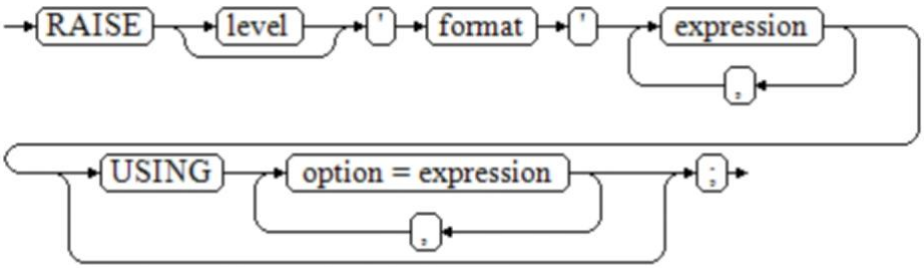
存储过程中如果有 DML 语句（SELECT、UPDATE、INSERT、DELETE），DML 语句只能使用中括号来访问数组元素，使用小括号默认识别为数组访问，若数组不存在，则识别为函数表达式。

7.2.12.调试

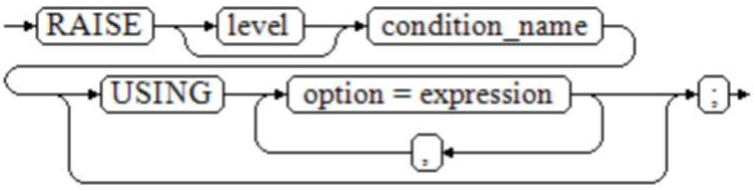
语法格式

RAISE 有以下五种语法格式：

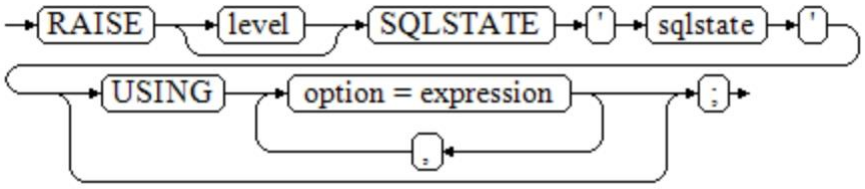
raise_format::=



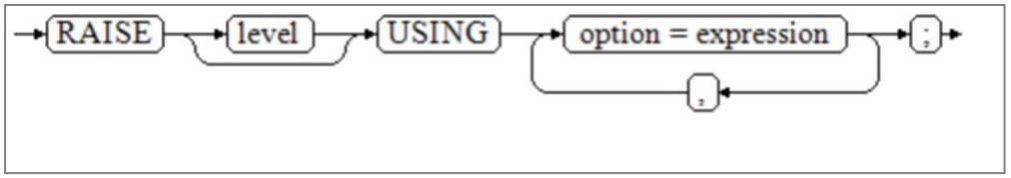
raise_condition::=



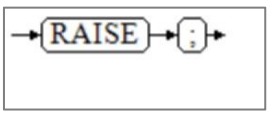
raise_sqlstate::=



raise_option::=



raise::=



参数说明：

- level 选项用于指定错误级别，有 DEBUG，LOG，INFO，NOTICE，WARNING 以及 EXCEPTION（默认值）。EXCEPTION 抛出一个正常终止当前事务的异常，其他的仅产生不同异常级别的信息。特殊级别的错误信息是否报告到客户端、写到服务器日志由 log_min_messages 和 client_min_messages 这两个配置参数控制。
- format：格式字符串，指定要报告的错误消息文本。格式字符串后可跟表达式，用于向消息文本中插入。在格式字符串中，%由 format 后面跟着的参数的值替换，%%用于打印出%。例如：

```
--v_job_id 将替换字符串中的 %：
RAISE NOTICE 'Calling cs_create_job(%)',v_job_id;
```

- option = expression：向错误报告中添加另外的信息。关键字 option 可以是 MESSAGE、DETAIL、HINT 以及 ERRCODE，并且每一个 expression 可以是任意的字符串。
  - MESSAGE，指定错误消息文本，这个选项不能用于在 USING 前包含一个格式字符串的 RAISE 语句中。
  - DETAIL，说明错误的详细信息。
  - HINT，用于打印出提示信息。
  - ERRCODE，向报告中指定错误码（SQLSTATE）。可以使用条件名称或者直接用五位字符的 SQLSTATE 错误码。
- condition_name：错误码对应的条件名。
- sqlstate：错误码。
- 如果在 RAISE EXCEPTION 命令中既没有指定条件名也没有指定 SQLSTATE，默认用 RAISE EXCEPTION (P0001)。如果没有指定消息文本，默认用条件名或者 SQLSTATE 作为消息文本。

- 图 5 的 raise 所示的语法不接任何参数。这种形式仅用于一个 BEGIN 块中的 EXCEPTION 语句，它使得错误重新被处理。
- 当由 SQLSTATE 指定了错误码，则不局限于已定义的错误码，可以选择任意包含五个数字或者大写的 ASCII 字母的错误码，而不是 00000。建议避免使用以三个 0 结尾的错误码，因为这种错误码是类别码，会被整个种类捕获。
- 兼容 O 模式下，SQLCODE 等于 SQLSTATE。

示例

示例 1：终止事务时，给出错误和提示信息：

```
CREATE OR REPLACE PROCEDURE proc_raise1(user_id in integer)
AS
BEGIN
RAISE EXCEPTION 'Noexistence ID --> %',user_id USING HINT = 'Please check your us
er ID';
END;
/

call proc_raise1(300011);
```

结果显示如下：

```
ERROR:  Noexistence ID --> 300011
HINT:  Please check your user ID
```

示例 2：两种设置 SQLSTATE 的方式：

```
CREATE OR REPLACE PROCEDURE proc_raise2(user_id in integer)
AS
BEGIN
RAISE 'Duplicate user ID: %',user_id USING ERRCODE = 'unique_violation';
END;
/

\set VERBOSITY verbose
```

```
call proc_raise2(300011);
```

结果显示如下：

```
ERROR:  Duplicate user ID: 300011
SQLSTATE: 23505
```

如果主要的参数是条件名或者是 SQLSTATE，可以使用：

```
RAISE division_by_zero;
RAISE SQLSTATE '22012';
```

例如：

```
CREATE OR REPLACE PROCEDURE division(div in integer, dividend in integer)
AS
DECLARE
res int;
 BEGIN
 IF dividend=0 THEN
  RAISE division_by_zero;
   RETURN;
 ELSE
   res := div/dividend;
   RAISE INFO 'division result: %', res;
   RETURN;
 END IF;
 END;
/

call division(3,0);
```

结果显示如下

```
ERROR:  division_by_zero
SQLSTATE: 22012
```

或者另一种方式：

```
RAISE unique_violation USING MESSAGE = 'Duplicate user ID: ' || user_id;
```

7.2.13.声明语法

结构

PL/pgSQL 块中可以包含子块，子块可以位于 PL/pgSQL 中任何部分。
PL/pgSQL 块的结构如下：

- 声明部分：声明 PL/pgSQL 用到的变量、类型、游标、局部的存储过程和函数。

```
DECLARE
```

不涉及变量声明时声明部分可以没有。

- 对匿名块来说，没有变量声明部分时，可以省去 DECLARE 关键字。
- 对存储过程来说，没有 DECLARE， AS 相当于 DECLARE。即便没有变量声明的部分，关键字 AS 也必须保留。
- 执行部分：过程及 SQL 语句，程序的主要部分。必选。

```
BEGIN
```

- 执行异常部分：错误处理。可选。

```
EXCEPTION
```

- 结束

```
END;
/
```

禁止在 PL/pgSQL 块中使用连续的 Tab，连续的 Tab 可能会造成在使用 gsql 工具带“-r”参数执行 PL/pgSQL 块时出现异常。

分类

PL/pgSQL 块可以分为以下几类：

- 匿名块：动态构造，只能执行一次。语法请参考图 1。
- 子程序：存储在数据库中的存储过程、函数、操作符和高级包等。当在数据库上建立好后，可以在其他程序中调用它们。

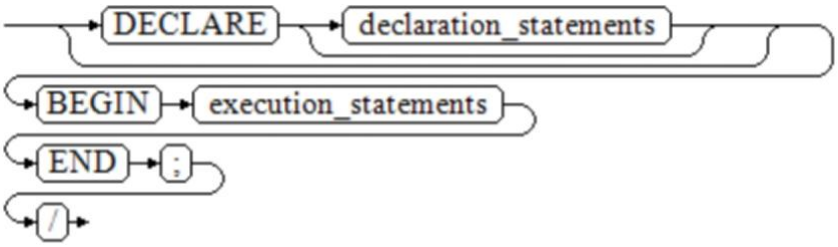
匿名块

匿名块（Anonymous Block）一般用于不频繁执行的脚本或不重复进行的活动。它们在一个会话中执行，并不被存储。

语法

匿名块的语法参见图 1。

图 1. anonymous_block::=



对以上语法图的解释如下：

- 匿名块程序实施部分，以 BEGIN 语句开始，以 END 语句停顿，以一个分号结束。输入“/”按回车执行。

最后的结束符“/”必须独占一行，不能直接跟在 END 后面。

- 声明部分包括变量定义、类型、游标定义等。
- 最简单的匿名块不执行任何命令。但一定要在任意实施块里至少有一个语句，甚至是一个 NULL 语句。

子程序

存储在数据库中的存储过程、函数、操作符和高级包等。当在数据库上建立好后，可以在其他程序中调用它们。

7.3.自定义函数

7.3.1.PL/pgSQL 语言函数

PL/pgsql 是一种可载入的过程语言。

用 PL/pgsql 创建的函数可以被用在任何可以使用内建函数的地方。例如，可以创建复杂条件的计算函数并且后面用它们来定义操作符或把它们用于索引表达式。

SQL 被大多数数据库用作查询语言。它是可移植的并且容易学习。但是每一个 SQL 语句必须由数据库服务器单独执行。

这意味着客户端应用必须发送每一个查询到数据库服务器、等待它被处理、接收并处理结果、做一些计算，然后发送更多查询给服务器。如果客户端和数据库服务器不在同一台机器上，则会引起进程间通信并且将带来网络负担。

通过 PL/pgsql，可以将一整块计算和一系列查询分组在数据库服务器内部，这样就有了一种过程语言的能力并且使 SQL 更易用，同时能节省客户端/服务器通信开销。

- 客户端和服务器之间的额外往返通信被消除。
- 客户端不需要的中间结果不必被整理或者在服务器和客户端之间传送。
- 多轮的查询解析可以被避免。

PL/pgsql 可以使用 SQL 中所有的数据类型、操作符和函数。

应用 PL/pgsql 创建函数的语法为 CREATE FUNCTION。PL/pgsql 是一种可载入的过程语言。其应用方法与存储过程相似，只是存储过程无返回值，函数有返回值。

8. 并发控制

8.1.事务管理

8.1.1.管理事务

事务是用户定义的一个数据库操作序列，这些操作要么全做要么全不做，是一个不可分割的工作单位。PanWeiDB 数据库支持的事务控制命令有启动、设置、提交、回滚事务。PanWeiDB 数据库支持的事务隔离级别有读已提交和可重复读。

事务控制

以下是数据库支持的事务命令：

- **启动事务**

  用户可以使用 START TRANSACTION 和 BEGIN 语法启动事务,具体参考：SQL 语法参考->SQL 语法->START TRANSACTION 和 BEGIN 章节。

- **设置事务**

  用户可以使用 SET TRANSACTION 或者 SET LOCAL TRANSACTION 语法设置事务特性，详细操作请参考：SQL 语法参考->SQL 语法->START TRANSACTION 章节。

- **提交事务**

  用户可以使用 CpanweidbIT 或者 END 完成提交事务的功能，即提交事务的所有操作，详细操作请参考：SQL 语法参考->SQL 语法->COMMIT | END 章节。

- **回滚事务**

  回滚是在事务运行的过程中发生了某种故障，事务不能继续执行，系统将事务中对数据库的所有已完成的操作全部撤销，详细操作请参考：SQL 语法参考->SQL 语法->ROLLBACK 章节。

【说明】

- 数据库中收到的一次执行请求（不在事务块中），如果含有多条语句，将会被打包成一个事务，如果其中有一个语句失败，那么整个请求都将会被回滚。
- 支持在存储过程中使用 COMMIT 和 ROLLBACK。
- 未执行事务控制语句进行事务控制时，事务默认自动提交，不支持修改 AUTOCOMMIT 参数。

事务隔离级别

事务隔离级别，它决定多个事务并发操作同一个对象时的处理方式。

【说明】

在事务中第一个数据修改语句（SELECT、INSERT、DELETE、UPDATE、FETCH、COPY）执行之后，事务隔离级别就不能再次设置。

- **READ COMMITTED**：读已提交隔离级别，事务只能读到已提交的数据而不会读到未提交的数据，这是缺省值。

  实际上，SELECT 查询会查看到在查询开始运行的瞬间该数据库的一个快照。不过，SELECT 能查看到其自身所在事务中先前更新的执行结果。即使先前更新尚未提交。请注意，在同一个事务里两个相邻的 SELECT 命令可能会查看到不同的快照，因为其他事务会在第一个 SELECT 执行期间提交。

  因为在读已提交模式里，每个新的命令都是从一个新的快照开始的，而这个快照包含所有到该时刻为止已提交的事务，因此同一事务中后面的命令将看到任何已提交的其他事务的效果。这里关心的问题是在单个命令里是否看到数据库里绝对一致的视图。

  读已提交模式提供的部分事务隔离对于许多应用而言是足够的，并且这个模式速度快，使用简单。不过，对于做复杂查询和更新的应用，可能需要保证数据库有比读已提交模式更加严格的一致性视图。
- **REPEATABLE READ**：事务可重复读隔离级别，事务只能读到事务开始之前已提交的数据，不能读到未提交的数据以及事务执行期间其他并发事务提交的修改（但是，查询能查看到自身所在事务中先前更新的执行结果，即使先前更新尚未提交）。

这个级别和读已提交是不一样的，因为可重复读事务中的查询看到的是事务开始时的快照，不是该事务内部当前查询开始时的快照，就是说，单个事务内部的 select 命令总是查看到同样的数据，查看不到自身事务开始之后其他并发事务修改后提交的数据。使用该级别的应用必须准备好重试事务，因为可能会发生串行化失败。

8.1.2.自治事务

功能描述

该功能支持在定义 PL/pgSQL 定义函数/存储过程中，在声明部分增加指令：PRAGMA AUTONOMOUS_TRANSACTION，用来控制对当前存储过程中的内部事务可以进行独立提交，而不影响其他的事务。 该执行方式与普通方式执行函数/存储过程相同。 此外，自治事务还支持如下功能：

- 支持 PL/pgSQL 块内的 SQL 语句使用函数/存储过程参数和定义的变量。
- 允许在匿名块内声明自治事务。
- 支持自治事务的嵌套使用。

注意事项

声明为自治事务的函数/存储过程对返回值以及参数有如下限制：

- 不支持集合作为返回值。
- 不支持组合类型作为返回类型。
- 不支持游标类型。
- 不支持 out 参数。

示例

- 创建测试表。

```
create table at_tb2(id int, val varchar(64));
```

- 支持声明为自治事务的存储过程。

1、创建存储过程

```
create or replace procedure at_test3(i int)
AS
```

```
DECLARE
 PRAGMA AUTONOMOUS_TRANSACTION;
BEGIN
 insert into at_tb2 values(1, 'before s1');
 insert into at_tb2 values(2, 'after s1');
 if i > 10 then
 rollback;
 else
 commit;
 end if;
end;
/
```

如果数据库未开启默认自动提交，则需要手动提交。

```
commit;
```

2、调用存储过程。

```
call at_test3(6);
```

3、查询结果。

```
select * from at_tb2;
```

当结果显示如下信息，则说明写入成功。

```
id |  val
----+-----------
 1 | before s1
 2 | after s1
(2 rows)
```

❖ 支持 plpgsql 块内的 sql 语句使用函数/存储过程参数和定义的变量

1、创建存储过程。

```
create or replace procedure at_proc(i int)
as
declare
strval varchar;
```

```
pragma autonomous_transaction;
begin
strval:= 'begin:insert by autonomous transaction procedure<at_proc>';
insert into at_tb2 values(i,strval);
commit;
end;
/
```

如果数据库未开启默认自动提交，则需要手动提交。

```
commit;
```

2、调用存储过程。

```
call at_proc(1);
```

3、查询结果。

```
select * from at_tb2;
```

当结果显示如下信息，则说明验证成功。

```
id |   val
----+-----------
 1 | before s1
 2 | after s1
 3 | begin:insert by autonomous transaction procedure<at_proc>
(3 rows)
```

❖ 支持在触发器函数内声明自治事务

1、创建源表及触发表。

```
create table test_trigger_src_tbl(id1 int, id2 int, id3 int);
create table test_trigger_des_tbl(id1 int, id2 int, id3 int);
```

2、创建触发器函数。

```
create or replace function tri_insert_func() returns trigger as
$$
declare
pragma autonomous_transaction;
```

```
begin
insert into test_trigger_des_tbl values(new.id1, new.id2, new.id3);
return new;
end
$$ language plpgsql;
```

3、创建 insert 触发器。

```
create trigger insert_trigger
before insert on test_trigger_src_tbl
for each row
execute procedure tri_insert_func();
```

4、执行 insert 触发事件。

```
insert into test_trigger_src_tbl values(100,200,300);
```

5、检查触发结果。

```
select * from test_trigger_src_tbl;
```

结果显示如下：

```
 id1 | id2 | id3
-----+-----+-----
 100 | 200 | 300
(1 row)
```

6、查看触发操作是否生效。

```
select * from test_trigger_des_tbl;
```

结果显示如下：

```
 id1 | id2 | id3
-----+-----+-----
 100 | 200 | 300
(1 row)
```

❖ 支持在匿名块内声明自治事务

1、清空测试表数据。

```
truncate at_tb2;
```

2、匿名块声明自治事务。

```
declare
pragma autonomous_transaction;
strval varchar;
begin strval:='insert by autonomous transaction by anonymous block';
insert into at_tb2 values(1,strval);
commit;
end;
/
```

3、查询结果。

```
select * from at_tb2;
```

结果显示如下：

```
id |              val
1 | insert by autonomous transaction by anonymous block
(1 row)
```

❖ 支持自治事务嵌套使用

1、创建嵌套自治事务的存储过程。

```
create or replace procedure at_proc_inner(i int)
as
declare
pragma autonomous_transaction;
begin
insert into at_tb2 values(1,'insert by autonomous transaction procedure<at_proc_in
ner>');
if i>5 then
rollback;
else
commit;
end if;
end;
```

```
/
create or replace procedure at_proc_outer(i int)
as
declare
pragma autonomous_transaction;
begin
insert into at_tb2 values(1,'insert by autonomous transaction procedure<at_proc_o
uter>');
at_proc_inner(i);
if i>10 then rollback;
else
commit;
end if;
end;
/
```

2、调用存储过程。

❖ 调用存储过程（入参为 11）

```
truncate at_tb2;
call at_proc_outer(11);
```

验证结果：

```
select * from at_tb2;
```

结果显示如下：

```
----+-----
(0 rows)
```

❖ 调用存储过程（入参为 8）

```
truncate at_tb2;
call at_proc_outer(8);
```

验证结果：

```
select * from at_tb2;
```

结果返回如下：

```
id |                    val
----+------------------------------------------------------------
 1 | insert by autonomous transaction procedure<at_proc_outer>
```

❖ 调用存储过程（入参为 2）

```
truncate at_tb2;
call at_proc_outer(2);
```

验证结果：

```
select * from at_tb2;
```

结果返回如下：

```
id |                    val
----+------------------------------------------------------------
 1 | insert by autonomous transaction procedure<at_proc_outer>
 1 | insert by autonomous transaction procedure<at_proc_inner>
(2 rows)
```

9. 插件管理

本章介绍 PanWeiDB 支持的数据库插件，也叫数据库扩展程序。

用户可以使用 CREATE EXTENSION 将新的扩展加载到当前数据库中。不得加载的同名的扩展名。

加载扩展本质上等于运行扩展脚本文件。执行 DROP EXTENSION 时可以删除扩展。

9.1.citext 插件

citext 插件提供了 citext 数据类型及相关的操作符和函数。 citext 类型是一种特殊的字符串类型，用于存储大小写不敏感的文本。当在一个字段上使用 citext 类型时，所有基于这个字段的比较都会自动忽略字母的大小写差异，这意味着 "Example" 和 "example" 将被视为相同的值。
citext 类型对于需要存储或检索含有字母的字段时不希望区分大小写的场景下。例如，创建索引或进行搜索操作时，希望避免大小写引起的复杂性。

功能描述

通常情况下，不区分大小写匹配的标准方法是使用 `lower()` 函数，例如：

```
SELECT * FROM tab WHERE lower(col) = LOWER(?);
```

这种调用方式存在以下缺点：

- 使 SQL 语句冗长，并且需要同时使用 lower() 函数修饰列与查询表达式。
- 如果创建索引时，相关列上没有使用 lower() 函数，则使用 lower() 表达式进行查询时，也不会使用该索引。
- 如果查询列被声明为 UNIQUE 或者 PRIMARY KEY，则隐式产生的索引仍然对大小写敏感。因此这种调用方式对不区分大小写的查询是无效的，并且在判断唯一性时区分大小写。

citext 数据类型可以在 SQL 查询中不调用 lower() 函数，而实现主键不区分大小写。

citext 数据类型的大写字母和小写字母字符的匹配依赖于数据库的 LC_CTYPE 设置规则，这种操作与查询中 `lower()` 的使用相同。

注意事项

- 本文提及的特性仅在数据库兼容模式为 Oracle 或 PostgreSQL 时支持（即数据库实例初始化时指定 DBCOMPATIBILITY='A' 或 'PG'）。
- citext 类型的大小写转换取决于数据库的字符分类（即 LC_CTYPE 设置），LC_CTYPE 在创建数据库时指定。如果数据库中的数据使用一种语言存储，而查询的字符集是另一种语言，则查询结果会与预期不一致。
- citext 类型不适用于局部的大小写不敏感的比较。若要在局部执行大小写不敏感的比较，建议使用 lower() 函数对 text 类型进行比较。
- 包含 citext 操作符的模式必须为当前的 search_path，否则会使用大小写敏感的 text 类型操作符。

字符串比较操作

citext 类型会先将字符串转换为小写，然后比较结果。因此如果 `lower()` 对两个字符串产生了相同的结果，则认为这两个字符串相等。

对于正则表达式函数，如果需要匹配大小写敏感，可以声明"c"标记以来强制使用大小写匹配，或者在函数或操作符运算前转换为 text 类型。

对于 citext 类型，以下函数或操作符对大小写不敏感：

- !~ 操作符
- !~* 操作符
- ~~ 操作符
- ~~* 操作符
- !~~ 操作符
- !~~* 操作符
- regexp_replace() 函数
- regexp_split_to_array() 函数
- regexp_split_to_table() 函数
- replace() 函数
- split_part() 函数
- strpos() 函数
- translate() 函数

示例

1、安装 citext 插件。

```
CREATE EXTENSION citext;
```

2、创建测试表。

```
CREATE TABLE users (
   nick CITEXT PRIMARY KEY,
   pass TEXT   NOT NULL
);

INSERT INTO users VALUES ( 'larry',  md5(random()::text) );
INSERT INTO users VALUES ( 'Tom',    md5(random()::text) );
INSERT INTO users VALUES ( 'Damian', md5(random()::text) );
INSERT INTO users VALUES ( 'NEAL',   md5(random()::text) );
INSERT INTO users VALUES ( 'Bjørn',  md5(random()::text) );
```

3、测试查询是否大小写不敏感。

```
SELECT * FROM users WHERE nick = 'Larry';
```

返回结果为：

```
 nick  | pass
-------+----------------------------------
 larry | beffa8d3febac980c47348daff4d843
(1 row)
```

9.2.ltree 插件

功能描述

ltree 插件实现了一种数据类型，用于表示存储在一个层次树状结构中的数据的标签，并且提供了在标签树中搜索的扩展功能。

- 标签是由字符组成的序列。在 C 区域中允许标签使用以下字符：大写字母（A-Z）、小写字母（a-z）、数字（0-9）、连字符（-）、下划线（_）。标签支持的字符与数据库当前的区域设置有关。标签长度必须少于 1000 字符。

  支持在初始化数据库时通过--locale 选项指定区域。如果不加--locale 选项，则采用系统默认的区域。系统默认区域和编码格式可以使用 locale 如下命令查看：

```
locale|grep LC_CTYPE
```

- 一个标签路径是由点号分隔的零个或者更多个标签的序列，例如 L1.L2.L3，它表示一个从层次树的根到一个特定节点的路径。一个标签路径的长度必须小于 65535 个标签。

注意事项

- 默认支持的 ltree 插件版本为 V1.2。
- 创建此扩展要求用户在当前数据库中具有 create 权限。
- 不同版本的插件支持的功能范围不同，支持使用如下语句查看当前插件的版本：

```
select * from pg_available_extensions WHERE name like 'ltree';
```

支持范围

ltree 插件提供了多种数据类型：

- ltree 存储一个标签路径。

例如：

```
CREATE TABLE test (path ltree);
INSERT INTO test VALUES ('Top');
INSERT INTO test VALUES ('Top.Science');
INSERT INTO test VALUES ('Top.Science.Astronomy');
INSERT INTO test VALUES ('Top.Science.Astronomy.Astrophysics');
INSERT INTO test VALUES ('Top.Science.Astronomy.Cosmology');
```

- lquery 表示一个用于匹配 ltree 值的类正则表达式。

一个简单词可以匹配一个路径中的标签，星号可以匹配零个或多个标签。星号和简单词都可以被限定来限制它能匹配多少标签。它们可以用点连接起来，以形成一个必须匹配整个标签路径的模式。例如：

模式	描述
foo	正好匹配标签路径 foo。
.foo.	匹配任何包含标签 foo 的路径。
*.foo	匹配任何最后一个标签是 foo 的标签路径。

模式	描述
*{n}	匹配正好 n 个标签。
*{n, }	匹配至少 n 个标签。
*{n,m}	匹配至少 n 个最多 m 个标签。
*{,m}	匹配至多 m 个标签。
foo{n,m}	匹配至少 n 个最多 m 个 foo 标签。
foo{,}	匹配任意数量的 foo 标签。

在 lquery 中，可以在一个非星号标签的末尾用如下的修饰符，使它能匹配除了精确匹配之外更多的匹配::

修饰符	描述
@	不区分大小写匹配，例如 a@匹配 A。
*	匹配带此前缀的任何标签，例如 foo*匹配 foobar。
%	匹配开头以下划线分隔的词。

除此之外，还可以使用 |(OR)、! (NOT)修饰符进行匹配。

❖ ltxtquery 表示一种用于匹配 ltree 值的类全文搜索的模式。

一个 ltxtquery 值包含词，也可能在末尾带有修饰符@、*、%，修饰符具有和 lquery 中相同的含义。词可以用&（AND）、|（OR）、!（NOT）以及圆括号组合。例如:

```
CREATE TABLE test (path ltree);
INSERT INTO test VALUES ('Top.Science');
SELECT path~'Science'::lquery from test;  --返回 f
SELECT path@'Science'::ltxtquery from test;   --返回 t
```

或者:

```
Europe & Russia*@ & !Transportation
```

上述语句表示匹配包含标签 Europe 以及任何以 Russia 开始（大小写不敏感）的标签的路径，但是不匹配包含标签 Transportation 的路径。这些词在路径中的位置并不重要。

【说明】

- ltxtquery 允许符号之间存在空白，但是 ltree 和 lquery 不允许。
- lquery 和 ltxtquery 的关键区别是后者匹配词时不考虑它们在标签路径中的位置。

操作符

ltree 支持普通比较操作符=、<>、<、>、<=、>=。比较会按照树遍历的顺序排序，一个节点的子女按照标签文本排序。除此以外，还支持下表列出的特殊操作符：

操作符	返回值	描述
ltree@>ltree	boolean	判断左参数是不是右参数的祖先。
ltree<@ ltree	boolean	判断左参数是不是右参数的后代。
ltree ~ lquery	boolean	判断 ltree 是否匹配 lquery。
lquery ~ ltree	boolean	
ltree? lquery[]	boolean	判断数组中是否存在与 ltree 匹配的 lquery。
lquery[]? ltree	boolean	
ltree @ ltxtquery	boolean	判断 ltree 是否匹配 ltxtquery。
ltxtquery @ ltree	boolean	
ltree || ltree	ltree	串接 ltree 路径。
ltree || text	ltree	将 text 文本转换成 ltree 并且串接。
text || ltree	ltree	
ltree[] @>ltree	boolean	判断数组中是否包含右参数 ltree 的一个祖先。

操作符	返回值	描述
ltree <@ ltree[]	boolean	判断数组中是否包含左参数 ltree 的一个祖先。
ltree[]<@ ltree	boolean	判断数组是否包含右参数 ltree 的一个后代。
ltree@> ltree[]	boolean	判断数组是否包含左参数 ltree 的一个后代。
ltree[]~ lquery	boolean	判断数组是否包含匹配 lquery 的路径。
lquery~ltree[]	boolean	
ltree[] ? lquery[]	boolean	判断 ltree 数组是否包含匹配任意 lquery 的路径。
lquery[]? ltree[]	boolean	
ltree[] @ ltxtquery	boolean	判断数组是否包含匹配 ltxtquery 的路径。
ltxtquery @ltree[]	boolean	
ltree[]?@ >ltree	ltree	返回数组中第一个是右参数祖先的项，不存在返回 NULL。
ltree[]?<@ ltree	ltree	返回数组中第一个是左参数祖先的项，不存在返回 NULL。
ltree[]?~lquery	ltree	返回数组中第一个匹配 lquery 的项，不存在返回 NULL。
ltree[]?@ ltxtquery	ltree	

函数

函数	描述
subltree(ltree,int start,int end)	返回从位置 start 到位置 end-1（从 0 开始计算）的子路径。
subpath(ltree,int offset,int len)	返回从位置 offset 开始长度为 len 的子路径。 如果 offset 为负，则子路径从距离路径终点的远端开始。 如果 len 为负，则从路径的尾部开始丢弃 len 个标签。
subpath(ltree,int	返回从位置 offset 开始一直延伸到路径末尾的子路径。

函数	描述
offset)	如果 offset 为负，则子路径从距离路径终点的远端开始。
nlevel(ltree)	返回路径中的标签数量。
index(a ltree ,b ltree)	a 中第一次出现 b 的位置，没有找到则返回-1，从 0 开始。
index(a ltree ,b ltree,offset int)	a 中第一次出现 b 的位置，搜索从 offset 开始。offset 为负，则从尾部-offset 个标签开始。
text2ltree(text)	将 text 转换成 ltree。
ltree2text(text)	将 ltree 转换成 text。
lca(ltree,ltree,...)	最长公共路径的前缀，最多支持 8 个参数。
lca(ltree[])	最长公共路径的前缀。

索引

ltree 支持一些能加速上述操作符的索引类型：

- ltree 数据类型支持 B-树索引：<、<=、=、>=、>
- ltree 数据类型支持 GIST 索引：<、<=、=、>=、>、@>、<@、@、~、?
- ltree[]数据类型支持 GIST 索引：ltree[]<@ltree、ltree@>ltree[]、@、~、?

【说明】

在 PG 数据库中 ltree 的 gist 索引在使用时可以加选项带参数，比如：

```
create index tstidx on treetest using gist (t gist_ltree _ops(siglen=2025));
```

PanWeiDB 不支持该功能，在 PanWeiDB 数据库中使用方法如下：

```
create index tstidx on ltreetest using gist (t gist_ltree_ops);
```

示例

示例 1：ltree 数据类型的列支持增删改查。

1、创建 ltree 插件。

```
create extension ltree;
```

2、创建包含数据类型 ltree 的表。

```
create table tb_1101588(id int, path ltree);
```

3、向表中插入数据。

```
insert into tb_1101588 values(1, 'GangTai.NanGeShou.ZhouJieLun.QiLiXiang');
insert into tb_1101588 values(2, 'GangTai.NanGeShou.ZhouJieLun.QingTian');
insert into tb_1101588 values(3, 'NeiDi.NanGeShou');
insert into tb_1101588 values(4, 'NeiDi.NvGeShou.ZhangLiangYing');
insert into tb_1101588 values(5, 'NeiDi');
```

4、查询表中得数据。

```
select * from tb_1101588;
```

返回结果为：

```
 id |              path
----+----------------------------------------
  1 | GangTai.NanGeShou.ZhouJieLun.QiLiXiang
  2 | GangTai.NanGeShou.ZhouJieLun.QingTian
  3 | NeiDi.NanGeShou
  4 | NeiDi.NvGeShou.ZhangLiangYing
  5 | NeiDi
(5 rows)
```

5、更新表中数据。

```
update tb_1101588 set path = 'GangTai.NanGeShou.LinJunJie' where path ~ '*.Qing
Tian';
```

查询数据：

```
select * from tb_1101588;
```

返回结果为：

```
 id |              path
----+----------------------------------------
  1 | GangTai.NanGeShou.ZhouJieLun.QiLiXiang
  3 | NeiDi.NanGeShou
```

```
 4 | NeiDi.NvGeShou.ZhangLiangYing
 5 | NeiDi
 2 | GangTai.NanGeShou.LinJunJie
(5 rows)
```

6、删除表中数据。

```
delete from tb_1101588 where path ~ '*.QiLiXiang';
```

7、查询表中数据。

```
select * from tb_1101588;
```

返回结果为：

```
 id |          path
----+-------------------------------
  3 | NeiDi.NanGeShou
  4 | NeiDi.NvGeShou.ZhangLiangYing
  5 | NeiDi
  2 | GangTai.NanGeShou.LinJunJie
(4 rows)
```

8、删除测试表。

```
drop table tb_1101588;
```

示例 2：数据类型 lquery 匹配 ltree 值的类正则表达式（foo 正好匹配标签路径 foo）。

1、创建测试表。

```
create table tb_1101393(id int, path ltree);
```

2、向表中插入数据。

```
insert into tb_1101393 values(1, 'GangTai.NanGeShou');
insert into tb_1101393 values(2, 'GangTai.NvGeShou');
```

3、匹配标签路径。

```
select path from tb_1101393 where path ~ 'GangTai.NvGeShou';
```

返回结果为：

```
    path
-----------------
 GangTai.NvGeShou
(1 row)
```

4、删除测试表。

```
drop table tb_1101393;
```

示例 3：数据类型 lquery 匹配 ltree 值的类正则表达式（*.foo.*匹配任何包含标签 foo 的标签路径）。

1、创建测试表。

```
create table tb_1101394(id int, path ltree);
```

2、向表中插入数据。

```
insert into tb_1101394 values(1, 'GangTai.NanGeShou.ZhouJieLun.QiLiXiang');
insert into tb_1101394 values(2, 'GangTai.NvGeShou.CaiYiLin.RiBuLuo.Test');
insert into tb_1101394 values(3, 'GangTai.NvGeShou.CaiYiLin.GuaiMeiDe');
```

3、匹配标签路径。

```
select path from tb_1101394 where path ~ '*.CaiYiLin.*';
```

返回结果为：

```
             path
-------------------------------------
 GangTai.NvGeShou.CaiYiLin.RiBuLuo.Test
 GangTai.NvGeShou.CaiYiLin.GuaiMeiDe
(2 rows)
```

4、删除测试表。

```
drop table tb_1101394;
```

示例 4： 操作符 ltree[] ~ lquery 和 lquery ~ ltree[]（判断数组是否包含匹配 lquery 的路径）。

```
select '{"NeiDi", "GangTai"}'::ltree[] ~ 'NeiDi*';
```

返回结果为：

```
 ?column?
----------
 t
(1 row)
```

示例 5： 调用函数 index(a ltree,b ltree)返回 a 中第一次出现 b 的位置，没有找到则返回-1，从 0 开始。

1、创建表中数据类型为 ltree。

```
create table tb_1101506(id int, path ltree);
```

2、向表中插入数据。

```
insert into tb_1101506 values(1, 'GangTai.NanGeShou.ZhouJieLun.QiLiXiang');
insert into tb_1101506 values(2, 'GangTai.NanGeShou.ZhouJieLun.QingTian');
insert into tb_1101506 values(3, 'NeiDi.NanGeShou');
insert into tb_1101506 values(4, 'NeiDi.NvGeShou.ZhangLiangYing');
insert into tb_1101506 values(5, 'NeiDi');
```

3、调用 index 函数。

```
select index(path, 'NanGeShou'::ltree) from tb_1101506;
```

返回结果为：

```
 index
-------
     1
     1
     1
    -1
    -1
(5 rows)
```

4、删除表。

```
drop table tb_1101506;
```

示例 6： 调用函数 lca(ltree[])计算数组中的路径的最长公共祖先。

```
select lca(array['1.2.3'::ltree, '1.2.3.4']);
```

返回结果为：

```
 lca
-----
 1.2
(1 row)
```

示例 7： ltree 上的 B-树索引。

1、创建表中数据类型为 ltree。

```
create table tb_1101523(id int, path ltree);
```

2、向表中插入数据。

```
insert into tb_1101523 values(1, 'GangTai.NanGeShou.ZhouJieLun.QiLiXiang');
insert into tb_1101523 values(2, 'GangTai.NanGeShou.ZhouJieLun.QingTian');
insert into tb_1101523 values(3, 'NeiDi.NanGeShou');
insert into tb_1101523 values(4, 'NeiDi.NvGeShou.ZhangLiangYing');
insert into tb_1101523 values(5, 'NeiDi');
```

3、创建索引。

```
create index idx_1101523 on tb_1101523(path);
```

4、验证操作符<、<=、=、>=、>。

```
select path from tb_1101523 where path < 'NeiDi.NvGeShou';
```

返回结果分别为：

```
             path
-------------------------------------
 GangTai.NanGeShou.ZhouJieLun.QiLiXiang
 GangTai.NanGeShou.ZhouJieLun.QingTian
```

```
 NeiDi.NanGeShou
 NeiDi
(4 rows)
```

5、验证操作符<=。

```
select path from tb_1101523 where path <= 'NeiDi.NvGeShou';
```

返回结果为：

```
             path
-------------------------------------
 GangTai.NanGeShou.ZhouJieLun.QiLiXiang
 GangTai.NanGeShou.ZhouJieLun.QingTian
 NeiDi.NanGeShou
 NeiDi
(4 rows)
```

6、验证操作符=。

```
select path from tb_1101523 where path = 'NeiDi';
```

返回结果为：

```
 path
-------
 NeiDi
(1 row)
```

7、验证操作符>=。

```
select path from tb_1101523 where path >= 'NeiDi.NvGeShou';
```

返回结果为：

```
         path
------------------------------
 NeiDi.NvGeShou.ZhangLiangYing
(1 row)
```

8、验证操作符>。

```
select path from tb_1101523 where path > 'NeiDi.NvGeShou';
```

返回结果为：

```
         path
------------------------------
 NeiDi.NvGeShou.ZhangLiangYing
(1 row)
```

9、关闭顺序扫描，验证查询时走索引扫描。

```
set enable_seqscan =off;
set enable_bitmapscan =off;
explain select path from tb_1101523 where path < 'NeiDi.NvGeShou';
```

返回结果为：

```
                          QUERY PLAN
-------------------------------------------------------------------------------
 Index Only Scan using idx_1101523 on tb_1101523  (cost=0.00..8.27 rows=1 width=
32)
   Index Cond: (path < 'NeiDi.NvGeShou'::ltree)
(2 rows)
```

9.3.PostGIS 插件

功能描述

PostGIS 是在对象关系型数据库 PostgreSQL 上增加了存储管理空间数据的能力的开源 GIS 数据库，它在 PostgreSQL 的基础上增加了表达地理信息的空间数据类型和操作这些类型的函数。

PanWeiDB V2.0-S3.0.1_B01 版本可使用 PostGIS 的如下模块：

- 支持矢量分析的 postgis 模块。
- 支持计算几何的 sfcgal 模块。
- 支持栅格分析的 raster 模块。
- 支持拓扑分析的 topology 模块。
- 支持扩展组件如下：

- Fuzzystrmatch 扩展，是一个判断字符串之间相似性和距离的一个插件，主要是提供 Soundex、Levenshtein、Metaphone 和 Double Metaphone 四个功能，开源 PostgreSQL 的 contrib 中包含此扩展。PostGIS 中的 postgis_tiger_geocoder 依赖此扩展。
- postgis_tiger_geocoder 扩展，TIGER 指的是拓扑集成地理编码和参考，这个扩展提供了 TIGER 数据的地理编码支持。

【须知】

注意此扩展启用前，需要先启用 fuzzystrmatch（字符串模糊查询）插件，以及可选的 address_standardizer、address_standardizer_data_us。

- address_standardizer 扩展， TIGER 数据地址规则化，是基于 PAGC 标准的地名标准化的扩展。
- address_standardizer_data_us 扩展，TIGER 地址规则化示例数据集，包含了基于 PAGC 标准的地名标准化的美国样例数据。

注意事项

- PanWeiDB 创建函数不支持添加 WINDOW 关键字，所以不支持函数 ST_ClusterDBSCAN 和 ST_ClusterKMeans。
- PanWeiDB 不支持 BRIN 索引，所以不支持 geog_brin_inclusion_add_value 和 geom2d_brin_inclusion_add_value 等相关函数。
- PanWeiDB 不支持 WITH ORDINALITY 功能，所以 topology.ValidateTopology 提供的功能与 PostGIS 3.2 版本不一致。
- 不支持 PostGIS 3.2 版本中的 ST_AsGeoJson 函数。

示例

示例 1：安装 PostGIS 插件。

1、联系工程师获取 PostGIS 插件安装包。

2、使用 root 用户解压 PanWeiDB 专用 PostGIS 插件安装包，将解压得到 postgis 文件夹下的内容复制到 GAUSSHOME 目录中（$GAUSSHOME 为数据库的安装路径）。

```
cp -r postgis/* $GAUSSHOME
chown -R panweidb:panweidb $GAUSSHOME
```

3、yum 安装相关依赖（如果不能使用 yum 可下载对应版本的安装包进行安装）。

```
yum install -y gmp gmp-devel          （版本 6.0.0）
yum install -y mpfr mpfr-devel        （版本 3.1.1）
yum install -y boost boost-devel      （版本 1.53.0）
```

4、启动数据库并设置 guc 参数 behavior_compat_options（PGDATA 为数据库的实例路径，panweidb 为数据库安装用户）。

```
su - panweidb
pw_ctl start
pw_guc reload -D $PGDATA -c " behavior_compat_options = 'bind_procedure_sear
chpath, display_leading_zero' "
```

5、重启并连接数据库（可能需要指定路径或修改连接端口等）。

```
pw_ctl restart
gsql panweidb -r
```

6、创建插件。

```
create  extension  postgis;
create  extension  postgis_sfcgal;
create  extension  postgis_raster;
create  extension  postgis_topology;
```

7、查询测试。

```
SELECT Box3D(ST_GeomFromEWKT('LINESTRING(1 2 3, 3 4 5, 5 6 5)'));
```

结果返回如下：

```
Box3d
---------
BOX3D(1 2 3,5 6 5)
(1 row)
```

8、删除插件

```
drop extension postgis cascade;
```

示例 2：创建 address_standardizer_data_us 扩展。

1、创建扩展（前提条件：已创建 PostGIS 插件）。

```
create extension address_standardizer_data_us;
```

2、查询是否安装成功。

```
\dx
```

示例 3：创建 fuzzystrmatch 和 postgis_tiger_geocoder 扩展。

1、创建扩展（前提条件：已创建 PostGIS 插件）。

```
create extension fuzzystrmatch;
create extension postgis_tiger_geocoder;
```

2、查看已创建扩展。

```
\dx
```

示例 4：创建 address_standardizer 扩展

1、创建扩展（前提条件：已创建 PostGIS 插件）。

```
create extension address_standardizer;
```

2、查询 address_standardizer 信息。

```
\dx address_standardizer
```

9.4.pgBadger 插件

功能描述

pgBadger 是一个快速、简便的日志分析工具。

pgBadger 能够基于 PanWeiDB 数据库的日志文件输出 HTML 页面，通过动态图的形式进行展示，利于阅读。pgBadger 是了解 PanWeiDB 服务器的行为并确定需要优化哪些 SQL 查询的便捷工具。

使用方法

在使用 pgBadger 之前，需要在 postgresql.conf 中配置 PanWeiDB 的日志记录行为。参考重设参数表 1 对以下 GUC 参数进行设置：

步骤 1： 启用 SQL 日志记录：

```
log_min_duration_statement=0
```

参数值设置为 0 时，每个语句都将被记录。在繁忙的服务器上，调高此参数的值可以实现仅记录持续时间较长的查询。

【说明】

请勿启用 log_statement，因为 pgBadger 无法解析它的日志格式。如果将 log_statement 设置为'all'，则不会通过 log_min_duration_statement 指令记录任何内容。

步骤 2： 设置日志的前缀信息

pgBadger 支持在 postgresql.conf 文件的 log_line_prefix 指令中设置的任何自定义格式，前提是 log_line_prefix 至少指定了一个时间转义序列（%t， %m 或%n）和进程相关的转义序列（%p 或%c）。

【说明】

日志格式由参数 log_destination 控制，有效值为 stderr、csvlog、syslog。

例如，对于“stderr”日志格式，log_line_prefix 必须至少为：

```
log_line_prefix = '%t [%p]: '
```

日志行前缀可以添加用户、数据库名称、应用程序名称和客户端 ip 地址，如下所示：

```
log_line_prefix = '%t [%p]: user=%u,db=%d,app=%a,client=%h '
```

stderr 输出的日志行前缀也可以是：

```
log_line_prefix = '%t [%p]: db=%d,user=%u,app=%a,client=%h '
```

对于“syslog”日志文件格式，log_line_prefix 可以是：

```
log_line_prefix = 'user=%u,db=%d,app=%a,client=%h '
```

或者：

```
log_line_prefix = 'db=%d,user=%u,app=%a,client=%h '
```

步骤 3：开启其他日志记录

需要开启一些基础记录功能，以便 pgBadger 能够从日志文件中获取更多信息：

```
log_checkpoints = on
log_connections = on
log_disconnections = on
log_lock_waits = on
log_temp_files = 0
log_autovacuum_min_duration = 0
log_error_verbosity = default
```

步骤 4： 设置字符集

确保 PanWeiDB 的日志消息应该是英文，设置语言环境：

```
lc_messages='en_US.UTF-8'
lc_messages='C'
```

pgBadger 解析器不支持其他语言环境，例如 'fr_FR.UTF-8' 。

附：更多日志相关参数配置说明

关于 log_min_duration_statement、log_duration 和 log_statement 配置的注意事项：

- 如果希望查询统计信息包含实际的查询字符串，则必须将 log_min_duration_statement 设置为 0 或更多毫秒。
- 如果只需要查看报告查询的持续时间和数量，而不想要查询的所有细节，将 log_min_duration_statement 设置为 -1 以禁用它，并在 postgresql.conf 文件中启用 log_duration。如果要添加最常见的请求报告，可以选择将 log_min_duration_statement 设置为更高的值，或者选择启用 log_statement。
- 启用 log_min_duration_statement 将添加关于最慢查询和占用时间最多的查询的报告。注意，如果将 log_statement 设置为'all'，则不

会使用 log_min_duration_statement 记录任何内容。

- 不要同时启用 log_min_duration_statement、log_duration 和 log_statement，这将导致错误的计数器值。注意，这也会大大增加日志的大小。应该始终优先选择 Log_min_duration_statement。

步骤 5： 构建报告

使用 pgBadger 查看报告的前提是 PanWeiDB 中执行了 SQL 并生成了相应日志文件。

执行类似如下的操作系统命令即可构建 pgBadger 报告，更多用法示例请参考后文[样例](#样例)，各选项介绍详见[参数说明](#参数说明)：

```
pgbadger -f stderr $OM_GAUSSLOG/postgresql-20xx-xx-xxxxx.log
```

步骤 6： 查看报告

为指定日志文件生成的 HTML 报告默认存放在当前登录用户的用户主文件夹下。

```
cd ~
ll
```

查看用户目录下的文件列表，其中`out.html`即为 pgBadger 生成的详细日志报告。

参数说明

pgBadger 提供了大量参数，用于控制报告生成的效果。

【须知】

PanWeiDB 暂不支持未介绍的参数。

- **-f | --format logtype**

  指定数据库日志的输出格式，可能的取值为：syslog、stderr、csv。

- **-o | --outfile filename**

  指定输出的文件名称。

  默认值取决于输出模式：out.html、out.txt、out.csv、或 out.json。此选项可多次使用以输出多种格式。

  如果使用 json 格式输出，则必须安装 Perl 模块 JSON::XS。

- **-a | --average minutes****

  构建查询和连接的平均图所需的分钟数。默认为 5 分钟。

- **-A | --histo-average min**

  构建查询直方图的分钟数。默认为 60 分钟。

- **-b | --begin datetime**

  要在日志中解析的数据的开始日期/时间（timestamp 或者 time）。

- **-c | --dbclient host**

  仅报告指定客户端主机的记录。

- **-C | --nocomment**

  从查询 SQL 中移除形如`/* ... */`的注释内容。

- **-d | --dbname database**

  分析指定的数据库。

- **-D | --dns-resolv**

  将客户端 IP 地址替换为其 DNS 名称。

  【说明】

  指定此选项会降低 pgBadger 的速度。

- **-e | --end datetime**

  要在日志中解析的数据的结束日期/时间。

- **-E | --explode**

  通过为每个数据库生成一个报表来扩展主报告。与数据库无关的全局信息将被添加到 postgres 库的报告中。

- **-G | --nograph**

  在 HTML 输出中禁用图表。默认启用图表。

- **-h | --help**

  展示帮助信息并退出。

- **-H | --html-outdir path**

  在增量模式下写入 HTML 报告的目录路径，二进制文件保留在用 `-O, --outdir` 选项定义的目录中。

- **-I | --incremental**

  使用增量模式，报告将按天生成，必须设置--outdir。

- **-j | --jobs number**

  并行解析日志的进程数量。

  默认情况下为单并发。

- **-J | --Jobs number**

  并行执行解析的日志文件数量。

  默认情况下为单并发。

- **-L | --logfile-list file**

  在文件中包含待解析的日志文件列表。

- **-m | --maxlength size**

  查询的最大长度，超过指定长度将被截断。

  默认截断长度为 100000。

- **-M | --no-multiline**

  不收集多行语句，以避免出现垃圾信息。

  对于会产生大量错误的报告非常有用。

- **-N | --appname name**

  只报告指定应用程序名称的记录。

- **-O | --outdir path**

  指定输出文件保存的目录。

- **-q | --quiet**

  不向 stdout 打印任何内容，包括进度条。

- **-Q | --query-numbering**

  在输出中添加查询编号。

- **-S | --select-only**

  报告中只分析 SELECT 查询。

- **-t | --top number**

  要存储/显示的查询次数。

  默认值：20。

- **-T | --title string**

  指定 HTML 报告页面的标题。

- **-u | --dbuser username**

  只分析给定用户的记录。

- **-U | --exclude-user username**

  排除指定用户的记录。

- **-V | --version**

  显示 pgBadger 版本号并退出。

- **-w | --watch-mode**

  只显示错误报告。

- **-x | --extension**

  输出格式。可能的值：txt，html，csv 或 json。

  默认值：html

- **--disable-type**

  不按类型、数据库或用户生成查询报告。

- **--disable-query**

  不生成查询报告（最慢查询、最频繁查询、按用户查询、按数据库查询.....）。

- **--disable-session****

  不生成会话报告。

- **--disable-connection**

  不生成连接报告。

- **--disable-lock**

  不生成锁报告。

- **--disable-temporary**

  不生成临时报告。

- **--disable-checkpoint**

  不生成检查点报告。

- **--disable-autovacuum**

不展示自动清理报告。

- **--csv-separator**

  用于设置 CSV 字段分隔符，默认为：`,`。

- **--exclude-db name**

  从报告中排除指定数据库的记录。

  可多次使用此选项。

- **--exclude-appname name**

  从报告中排除指定数据库的记录。

  可以多次使用此选项。

- **--exclude-client name**

  排除指定客户端 IP 的日志记录。

  可多次使用此选项。

构建语句样例

- **指定并发数，快速高效地分析 10GB 大文件：**

```
pgbadger -j 8 /pglog/postgresql-10.1-main.log
```

- **指定输出文件：**

```
pgbadger postgresql-2022-04-13_000000.csv  -o p.html
```

- **分析多个日志文件：**

```
pgbadger /var/log/postgresql/postgresql-2012-05-*
```

- **分析多个文件、多种格式：**

```
pgbadger /var/log/postgres.log.2.gz /var/log/postgres.log.1.gz /var/log/postgres.log
```

- **分析时段：**

```
pgbadger -b "2012-06-25 10:56:11" -e "2012-06-25 10:59:11" /var/log/postgresql.log
```

- **分析前排除程序 pw_dump：**

```
pgbadger --exclude-appname "pw_dump"  postgresql.log
```

- **每周报告错误：**

```
30 23 * * 1 /usr/bin/pgbadger -q -w /var/log/postgresql.log  -o /var/reports/pg_errors.html
```

- **每周使用增量行为生成报告：**

```
0 4 * * 1 /usr/bin/pgbadger -q `find /var/log/ -mtime -7 -name "postgresql.log*"` -o /var/reports/pg_errors-`date +\%F`.html -l /var/reports/pgbadger_incremental_file.dat
```

❖ **自动产生增量报告：**

```
0 4 * * * /usr/bin/pgbadger -I -q /var/log/postgresql/postgresql.log.1  -O /var/www/pg_reports/
```

示例

此处给出一个简单的示例，指导用户快速使用 pgBadger 构建一份日志报告。

【说明】

使用数据库安装用户执行以下操作。

1、设置必要的日志相关 GUC 参数。

```
echo "log_statement = 'all'
log_line_prefix = '%t [%p]: user=%u,db=%d,app=%a,client=%h '
log_destination='stderr'
log_min_duration_statement=0
log_duration = on
log_connections = on
log_disconnections = on" >>$PGDATA/postgresql.conf
```

2、重启数据库使设置生效。

```
pw_ctl restart
```

3、执行若干 SQL，确保$OM_GAUSSLOG 目录中存在用于日志分析的日志文件。

4、执行如下命令构建日志分析报告。其中`postgresql-20xx-xx-xxxxx.log`是数据库日志文件的名称。

```
pgbadger -f stderr $OM_GAUSSLOG/postgresql-20xx-xx-xxxxx.log
```

报告构建成功的显示信息如下：

```
[==========================>] Parsed 217879 bytes of 217879 (100.00%), queries: 358, events: 35
LOG: Ok, generating html report...
```

参考链接

pgBadger 官方文档：https://pgbadger.darold.net/documentation.html

9.5.pg_repack 插件

功能描述

pg_repack 插件可以用来重新组织和压缩数据库中的 astore，行存表和索引。其主要用于在线重建表和索引，以解决表或索引膨胀的问题，减少碎片并提高性能。

pg_repack 通过复制表配合触发器，实现数据库的在线表重组功能，实际是实现了和 vacuum full 类似的功能，“重组表”和“清理磁盘碎片”的原理类似，原表的元组之间可能存在一些空洞，pg_repack 负责重新组织一个元组排列紧密的新表。

pg_repack 与 VACUUM FULL 相比，后者在做清理时会长时间锁表阻塞 DML，而 pg_repack 不会。pg_repack 在原表的基础上，把原表数据拷贝到新表上，拷贝过程中，会通过建立触发器（期间会锁表）记录原表的 DML 操作，数据拷贝完成后，锁定表并将 log 更新到新表上，并互换新表（重组后的表）和原表。

注意事项

- 暂不支持在 MySQL 兼容模式下使用此插件。
- 使用 pg_repack 前必须确保空间充足。例如表和索引一共占用了 1GB 的空间，那么执行重组操作就至少需要额外的 2GB 的空间，才能执行成功，否则会由于空间不足而报错。
- 不支持列存，ustore 行存表，unlogged 表的重组。
- PanWeiDB 与 PostgreSQL 对于分区表的底层实现存在差异，PG 通过参数--parent-table 指定分区表，而 PanWeiDB 通过--table 参数来指定，会自动 repack 分区表的每个子分区。

使用方法

1、在使用 pg_repack 插件之前，需要在对应数据库中执行如下命令安装 pg_repack 插件：

```
create extension pg_repack;
```

2、创建完成插件之后使用特定的命令来运行 pg_repack，语法格式如下所示：

```
pg_repack [OPTION]... [DBNAME]
```

参数说明

【说明】
PanWeiDB 暂不支持以下未介绍的参数。

OPTION 选项

- **-a,--all**

  开启该选项，则会重组所有数据库（所有表），默认为关闭状态。未安装 pg_repack 插件的数据库将被跳过。

  【说明】
  该选项仅在大范围 repack 时才会生效，因此不能与 -t /--table 选项，-l /--parent-table 选项， -c /--schema 选项同时使用。

- **-t, --table=TABLE**

  仅重新组织指定的表。可以通过编写多个-t 开关来重组多个表。默认情况下，目标数据库中所有符合条件的表都会被重组。

- **-I, --parent-table=TABLE**

  重新组织指定的表及其继承者。可以通过编写多个-I 开关来重新组织多个表层次结构。

  【说明】
  由于继承表的概念目前只在 PostgreSQL 兼容模式下实现，所以该选项仅在 PanWeiDB 的 PG 兼容模式下可以使用，其他兼容模式下都会作为普通表处理。

- **-c, --schema=SCHEMA**

  仅重新打包指定模式中的表。可以通过指定多个-c 打包多个模式。可以与--tablespace 结合使用以将表移动到不同的表空间。

该选项仅在大范围 repack 时才会生效，因此不能与 -t /--table 选项，-l/--parent-table 选项， -a / --all 选项同时使用。

- **-s, --tablespace=TBLSPC**

  默认为空，该选项指定重组后的新表处于哪个表空间。本质上是 ALTER TABLE ... SET TABLESPACE 的在线版本。除非同时指定 --moveidx，否则表的索引将保留在原始表空间中。

- **-S, --moveidx**

  将重新打包的表的索引移动到 --tablespace 选项指定的表空间。该选项只在-s 选项指定后开启才会生效。

- **-o, --order-by=COLUMNS**

  默认为空，指定重组后的表元组根据某列进行排序。

  【说明】
  -o/--order-by 选项不能与-n/--no-order 选项同时指定。

- **-n, --no-order**

  默认关闭，若开启该选项，则会无序重组新表元组。

- **-N, --dry-run**

  指定该选项会对重组做一次预演，该选项开启后不会真正执行 repack，但会输出 repack 的预计相关表信息并退出。

- **-j, --jobs=NUM**

  默认为 0，如果原表存在多个索引的时候，使用该参数可以并发重建索引，提升 repack 的效率（只在表上索引较多的时候有用）。

  仅全表重新打包支持并行索引构建，不支持 --index 或 --only-indexes 选项。如果除 PanWeiDB 服务器外还有额外的内核和可用的磁盘 I/O，这可能是加速 pg_repack 的有用方法。

- **-i, --index=INDEX**

仅重新打包指定的索引。可以通过指定多个 -i 重新打包多个索引。可以与 --tablespace 结合使用以将索引移动到不同的表空间。

【注意】

- 该选项直接指明了特定的索引表，所以不能同时指定 -t/--table 选项或者 -l/--parent-table 选项（这两个是数据表相关的选项），相反， -x /--only-indexes 是配合数据表选项的附属选项，所以必须要指定 -t 或者 -l 其中之一的选项。
- -i 和 -x 本身是冲突的选项，不能同时使用，在-i 或 -x 其中之一开启后，需要注意以下情况：
  - 不能同时指定 -a / --all 选项：repack 索引表则必须指定特定索引表。
  - 不能同时指定 -C/--exclude-extension 选项：由于 -C 在只在类似 -a 这种大范围 repack 才会生效，所以也会和 -i/-x 指定特定索引表冲突。
  - 不能指定 -o /--order-by 或者 -n/--no-order 两个和排序相关的选项：这两个选项只能排序数据表的元组，而非索引表。
  - 需要显式指定 -Z 即 --no-analyze 选项：analyze 命令无法分析索引表，所以需要开启该选项。
  - 不能指定 -j/--jobs 即并行相关选项：无法并行处理特定索引表。

- **-x, --only-indexes**

  仅重新打包指定表的索引，表必须使用 --table 或 --parent-table 选项指定。当选项开启后，只重组--table 或者 --parent-table 所指定的数据表的索引表，但不重组数据表。

- **-T, --wait-timeout=SECS**

  pg_repack 过程需要获取共享锁和排它锁，这可能会和其他进程产生冲突。此选项指定了获取锁的超时时间，默认为 60 秒。

如果在此持续时间后无法获取锁且未指定 --no-kill-backend 选项，则 pg_repack 将强制取消冲突查询。

- **-D, --no-kill-backend**

  该选项需要结合--wait-timeout 选项进行理解，当获取锁的时间超过 timeout 值，工具会强制关闭冲突进程以完成 pg_repack 操作，若开启--no-kill-backend 选项则不会关闭，而是放弃 pg_repack 操作。

  此选项默认为关闭状态。

- **-Z, --no-analyze**

  该选项指定在重组完成后，是否对表进行分析。

  【注意】

  PanWeiDB 当前必须显式指定此选项，否则会报错。

- **-k, --no-superuser-check**

  默认情况下，pg_repack 需要在连接选项的 --username 选项指定超级用户才能执行，但开启该选项可以忽略超级用户检查。此设置对于在支持以非超级用户身份运行的平台上使用 pg_repack 很有用。

- **-C, --exclude-extension**

  默认为空，该选项指定一个拓展插件，在大范围重组的情况下，会忽略该拓展插件所涉及或者所需的系统表和普通表。

  【说明】

  该选项仅在大范围 repack 时才会生效，因此不能与 -t /--table 选项，-l /--parent-table 选项同时使用，同理 -i/-x 也不行。

连接选项

【说明】

连接到服务器的选项，不能同时使用--all 和 --dbname 或 --table 和 --parent-table。

- **-d, --dbname=DBNAME**

  指定要重组的数据库的名称。

- **-h, --host=HOSTNAME**

  指定运行服务器的机器主机名。如果该值以斜杠开头，则将其用作 Unix 域套接字的目录。

- **-p, --port=PORT**

  指定服务器正在侦听连接的 TCP 端口或本地 Unix 域套接字文件扩展名。

- **-U, --username=USERNAME**

  指定要连接的用户名。

- **-w, --no-password**

  不出现输入密码提示。如果主机要求密码认证并且密码没有通过其他形式给出，则连接尝试将会失败。

- **-W, --password**

  指定-U 参数所指定的用户密码。

通用选项

- **-e, --echo**

  回显命令发送到服务器。

- **-E, --elevel=LEVEL**

  指定输出的消息的等级。

  取值范围：DEBUG，INFO，NOTICE，WARNING，ERROR，LOG，FATAL 和 PANIC。

  默认值：INFO

- **--help**

  打印帮助信息并退出。

- **--version**

  打印版本信息并退出。

示例

【说明】
本示例通过函数 pg_total_relation_size 查询得到的数据表所用磁盘空间作为指标，来判断执行 pg_repack 前后，数据表占用的磁盘空间是否减少。查询得到的磁盘空间以及重组前后的差值因环境不同而存在差异，请知悉。

前置步骤： 执行如下命令安装 pg_repack 插件。

```
create extension pg_repack;
```

示例 1： 对普通表进行重组。

1、创建测试表。

```
CREATE TABLE repack_test (
  id SERIAL PRIMARY KEY,
  table_data TEXT,
  created_time TIMESTAMP DEFAULT CURRENT_TIMESTAMP
);
```

2、插入数据。

```
DO $$
DECLARE
  i INT;
BEGIN
  FOR i IN 1..10000 LOOP
    -- 插入 10000 行随机元组
    INSERT INTO repack_test (table_data) VALUES (md5(random()::text));
  END LOOP;
END;
$$;
```

3、删除部分数据以模拟碎片化空洞。

```
DELETE FROM repack_test WHERE id % 10 = 0;  -- 删除其中 10%的数据
```

4、检测表大小。

```
SELECT pg_size_pretty(pg_total_relation_size('repack_test'));
```

返回结果为：

```
pg_size_pretty
---------------
1216 kB
(1 row)
```

5、按\q 退出 gsql，执行 pg_repack 工具。

```
pg_repack --dbname=panweidb --table=repack_test --no-superuser-check --no-analyze
```

执行成功，返回如下内容：

```
INFO:repacking table "public.repack_test"
WARNING: Trigger function with non-plpgsql type is not recommended.
DETAIL:Non-plpgsql trigger function are not shippable by default.
HINT: Unshippable trigger may lead to bad performance.
INFO:repacking table "public.repack_test" success
```

6、再次连接 gsql 并检测表大小。

```
gsql -d panweidb -r
SELECT pg_size_pretty(pg_total_relation_size('repack_test'));
```

返回结果如下，数据表占用的磁盘空间已减少：

```
pg_size_pretty
---------------
1104 kB
(1 row)
```

示例 2： 对继承表进行重组。

【说明】
由于继承表的概念目前只在 PostgreSQL 兼容模式下实现，所以该示例执行环境为 PG 兼容模式。

1、创建父表。

```
CREATE TABLE vehicle (
  id SERIAL PRIMARY KEY,
  brand VARCHAR(50),
  model VARCHAR(50)
);
```

2、创建子表。

```
-- 创建子表 car
CREATE TABLE car (
   id SERIAL PRIMARY KEY,
   brand VARCHAR(50),
   model VARCHAR(50),
   car_type VARCHAR(50)
) INHERITS (vehicle);

-- 创建子表 truck
CREATE TABLE truck (
   id SERIAL PRIMARY KEY,
   brand VARCHAR(50),
   model VARCHAR(50),
   truck_type VARCHAR(50)
) INHERITS (vehicle);
```

3、向表中插入数据。

```
-- 插入大量数据到父表 vehicle
INSERT INTO vehicle (brand, model) SELECT 'BrandA', 'ModelA' || i FR
OM generate_series(1, 10000) AS i;

-- 插入大量数据到子表 car
INSERT INTO car (brand, model, car_type) SELECT 'BrandB', 'ModelB' |
| i, 'Sedan' FROM generate_series(1, 10000) AS i;

-- 插入大量数据到子表 truck
INSERT INTO truck (brand, model, truck_type) SELECT 'BrandC', 'Mod
elC' || i, 'Pickup' FROM generate_series(1, 10000) AS i;
```

4、删除测试表的部分数据。

```
DELETE FROM vehicle WHERE id % 10 = 0;
```

```
DELETE FROM car WHERE id % 10 = 1;
DELETE FROM truck WHERE id % 10 = 2;
```

5、检测每个表的空间占用。

```
SELECT 'vehicle', pg_size_pretty(pg_total_relation_size('vehicle'));
SELECT 'car', pg_size_pretty(pg_total_relation_size('car'));
SELECT 'truck', pg_size_pretty(pg_total_relation_size('truck'));
```

返回结果为：

```
?column?   |pg_size_pretty
-----------+--------------
vehicle    | 904 kB
(1 row)

?column?   |pg_size_pretty
-----------+--------------
car        | 1008 kB
(1 row)

?column?   |pg_size_pretty
-----------+--------------
truck      | 1008 kB
(1 row)
```

6、按\q 退出 gsql，执行 pg_repack。

```
pg_repack --dbname=panweidb --parent-table=vehicle --no-super
user-check --no-analyze --elevel=DEBUG
```

7、再次连接 gsql 并检测重组后表的大小。

```
gsql -d panweidb -r
SELECT 'vehicle', pg_size_pretty(pg_total_relation_size('vehicle'));
SELECT 'car', pg_size_pretty(pg_total_relation_size('car'));
SELECT 'truck', pg_size_pretty(pg_total_relation_size('truck'));
```

从如下返回结果可以看出，父表和子表占用的磁盘空间均已减少：

```
?column?  |pg_size_pretty
------------+---------------
vehicle   | 816 kB
(1 row)

?column?  |pg_size_pretty
------------+---------------
car       | 816 kB
(1 row)

?column?  |pg_size_pretty
------------+---------------
truck     | 816 kB
(1 row)
```

9.6.pg_zhtrgm 插件

功能描述

pg_zhtrgm 插件的主要作用是做相似度匹配。该插件提供了多种用于字母数字文本相似度匹配的函数和操作符，可以简化全文索引的使用方式。

该扩展不仅支持对英文字符的全文检索，还允许对中文字符进行全文检索。

默认支持的 pg_zhtrgm 插件版本为 V1.4。

注意事项

- 创建此扩展要求用户在当前数据库中具有 create 权限。
- 若安装 PanWeiDB 时采取非实例化安装，使用 initdb 工具初始化实例时需注意：initdb 指定的 -E 和--locale 参数对应的编码规则必须一致，否则将无法使用 pg_zhtrgm 插件。

- 若在升级数据库版本前，使用了在 V2.0-S3.0.0_B01 及更早的内核版本的 pg_zhtrgm 插件，则在升级数据库版本后，需要执行以下操作重载该插件。

```
DROP EXTENSION zhparser CASCADE;
CREATE EXTENSION zhparser;
```

支持范围

表 1 pg_zhtrgm 支持的函数

函数	返回类型	描述
similarity(text, text)	real	返回一个数字指示两个参数有多相似。该结果的范围是 0（指示两个字符串完全不相似）到 1（指示两个字符串完全一样）。
show_trgm(text)	text	返回一个给定字符串中所有的 trigram 组成的一个数组（实际上除了调试很少有用）。
show_limit()	real	返回%操作符使用的当前相似性阈值。
set_limit(real)	real	设置%操作符使用的当前相似性阈值。
show_trgm3(text)	text[]	返回三元匹配的 trigram 数组。
similarity3(text,text)	real	返回三元匹配的 similarity 相似度。
show_limit(integer)	real	integer 参数指定的是匹配方式，支持 2 和 3。 如果是 2 等价于 show_limit()，如果是 3 则返回%%操作符使用的当前词相似度阈值。
set_limit(real,integer)	real	integer 参数指定的是匹配方式，支持 2 和 3。 如果是 2 等价于 set_limit(real)，如果是 3 设置%%操作符使用的当前词相似度阈值，取值范围[0,1]。
word_similarity(text,text)	real	返回二元匹配的 word_similarity 相似度。
word_similarit	real	返回三元匹配的 word_similarity 相似度。

函数	返回类型	描述
y3(text,text)		
show_word_limit()	real	返回<%和%>操作符使用的当前词相似度阈值。
set_word_limit(real)	real	设置<%和%>操作符使用的当前词相似度阈值，取值范围[0,1]。
show_word_limit(integer)	real	integer 参数指定的是匹配方式,支持 2 和 3。 如果是 2 等价于 show_word_limit()，如果是 3 则返回<%%和%%>操作符使用的当前词相似度阈值。
set_word_limit(real,integer)	real	imteger 参数指定的是匹配方式，支持 2 和 3。 如果是 2 等价于 set_word_limit(real)，如果是 3 设置<%% 和 %%>操作符使用的当前词相似度阀值，取值范围[0,1]。
strict_word_similarity(text,text)	real	返回二元匹配的 strict_word_similarity 相似度。
strict_word_similarity3(text,text)	real	返回三元匹配的 strict_word_similarity 相似度。
show_strict_word_limit()	real	返回<<%和%>>操作符使用的当前词相似度阈值。
set_strict_word_limit(real)	real	设置<<%和%>>操作符使用的当前词相似度阈值，取值范围[0,1]。
show_strict_word_limit(integer)	real	integer 参数指定的是匹配方式，支持 2 和 3。 如果是 2 等价于 show_strict_word_limit()；如果是 3，则返回<<%%和%%>>操作符使用的当前词相似度阈值。
set_strict_word_limit(real,int	real	integer 参数指定的是匹配方式，支持 2 和 3。 如果是 2 等价于 set_strict_word limit(real)；如果是 3

函数	返回类型	描述
eger)		设置<<%%和%%>>操作符使用的当前词相似度阈值，取值范围[0,1]。

pg_zhtrgm 模块提供了 Gist 和 Gin 索引操作符类，此类全文索引仅支持模糊查询，不支持等值匹配。对应的操作符如表 2 所示：

表 2 pg_zhtrgm 支持的操作符

操作符	返回类型	描述
text <-> text	real	返回参数之间的“距离”，即 1-similarity()的值。
text %% text	boolean	如果参数的 similarity3 计算超过 show_limit()阈值，则返回 true。
		如果参数的 similarity3 计算超过 show_limit(3)阈值，则返回 true。
text <<->> text	boolean	返回参数之间的距离，即 1-similarity3()的值。
text <% text	boolean	如果第一个参数中的 bigram 集合与第二个参数中有序 bigram 集合的一个连续部分之间的相似度超过 show_word_limit()阈值，则返回 true。
text %> text	boolean	<%的交换子。
text <%% text	boolean	<%对应的三元匹配操作。如果第一个参数中的 trigram 集合与第二个参数中有序 trigram 集合的一个连续部分之间的相似度超过 show_word_limit(3)阈值，则返回 true。
text %%> text	boolean	<%%的交换子。
text <<%	boolean	如果第二个参数中包含序 bigram 集合的一个连续部分匹配

操作符	返回类型	描述
text		词边界，并且其与第一个参数的 bigram 集合的相似度超过 show_strict_word_limit()阈值，则返回 true。
text %>> text	boolean	<<%的交换子。
text <<%% text	boolean	<<%对应的三元匹配操作。如果第二个参数包含有序 bigram 集合的一个连续部分匹配词边界，并且其与第一个参数的 bigram 集合的相似度超过 show_strict_word_limit(3)阈值，则返回 true。
text %%>> text	boolean	<<%%的交换子。
text <<-> text	real	返回参数之间的距离，即 1-word_similarity()的值。
text <->> text	real	<<->的交换子。
text <<<- >> text	real	返回参数之间的“距离”，即 1-word_similarity3()的值。
text <<- >>> text	real	<<<->>的交换子。
text <<<-> text	real	返回参数之间的距离，即 1-strict_word_similarity()的值。
text <->>> text	real	<<<->的交换子。
text <<<<- >> text	real	返回参数之间的距离，即 1-strict_word_similarity3()的值
text <<- >>> text	real	<<<<->>的交换子。

pg_zhtrgm 提供了 gist 和 gin 索引操作符类，该操作符类支持在文本上创建一个索引，使用匹配操作提升检索速度或性能。

表 3 索引支持

索引	描述
gist_zhtrgm_ops	gist 索引操作符。 支持%、<->、like、ilike 操作符。
	支持~、~*、%>、<->>、%>>、<->>>操作符。
gin_zhtrgm_ops	gin 索引操作符。 支持 like、ilike、%操作符。
	支持~、~*、%>、%>>操作符。
gist_zhtrgm3_ops	三元匹配的 gist 索引操作符。 支持 like、ilike、%%、<<->>操作符。
	支持~、~*、%%>、<<->>>、%%>>、<<->>>>操作符。
gin_zhtrgm3_ops	三元匹配的 gin 索引操作符。 支持 like、ilike、%%操作符。
	支持~、~*、%%>、%%>>操作符。

示例

1、创建插件。

```
create extension pg_zhtrgm;
```

2、创建测试表。

```
create table t_full_text(id int, info text);
```

3、插入如下中文数据：

```
insert into t_full_text values (1, ' PanWeiDB 是一种特性非常齐全的自由软件的对象-关系型数据库管理系统（ORDBMS），PanWeiDB 支持大部分的 SQL 标准并且提供了很多其他现代特性，如复杂查询、外键、触发器、视图、事务完整性、多版本并发控制等。同样，PostgreSQL 也可以用许多方法扩展，例如通过增加新的数据类型、函数、操作符、聚集函数、索引方法、过程语言等。另外，因为许可证的灵活，任何人都可以以任何目的免费使用、修改和分发 PostgreSQL。');
```

4、在此表上创建以 pg_zhgrtm 提供的 gin_zhtrgm_ops、gist_zhtrgm_ops 操作符类所支持的 gin 索引或 gist 索引，该索引创建不再需要借助 to_tsvector 函数，同样地，使用该索引进行查询时也不再需要借助 to_tsquery 函数。

```
create index idx_full_text1 on t_full_text using gin(info gin_zhtrgm_ops);
create index idx_full_text2 on t_full_text using gist(info gist_zhtrgm_ops);
```

5、可以进行单字、词语、条件、短句的全文搜索：

❖ 使用单字搜索。

```
select * from t_full_text where info like '%函%';
```

结果显示如下：

```
 id |

      info

 ---+-----------------------------------------------------------------------------
------
 ---------------------------------------------------------------------------------
------
 ------------
  1 | PanWeiDB 是一种特性非常齐全的自由软件的对象-关系型数据库管理系统（ORDB
MS），PanWeiDB 支持大部分的 SQL 标准并且提供了很多其他现代特性，如复杂查询、
外键、触发器、视图、事务完整性、多版本并发控制等。同样，PostgreSQL 也可以用许
多方法扩展，例如通过增加新的数据类型、函数、操作符、聚集函数、索引方法、过程语
言等。另外，因为许可证的灵活，任何人都可以以任何目的免费使用、
 改和分发 PostgreSQL。
 (1 row)
```

❖ 使用词语搜索。

```
select * from t_full_text where info like '%视图%';
```

结果显示如下：

```
 id |

      info

 --+-------------------------------------------------------------------------------
------
 ---------------------------------------------------------------------------------
------
 -----------
  1 | PanWeiDB 是一种特性非常齐全的自由软件的对象-关系型数据库管理系统（ORDB
MS），PanWeiDB 支持大部分的 SQL 标准并且提供了很多其他现代特性，如复杂查询、
外键、触发器、视图、事务完整性、多版本并发控制等。同样，PostgreSQL 也可以用许
多方法扩展，例如通过增加新的数据类型、函数、操作符、聚集函数、索引方法、过程语
言等。另外，因为许可证的灵活，任何人都可以以任何目的免费使用、
 改和分发 PostgreSQL。
 (1 row)
```

❖ not like 条件搜索。

```
 select * from t_full_text where info like '%查询%' and info not like '%外键%';
```

结果显示如下：

```
 id | info
 --+----
 (0 rows)
```

❖ 使用语句搜索。

```
 select * from t_full_text where info like '%PanWeiDB 是一种特性非常齐全的自由软
件的对象%';
```

结果显示如下：

```
 id |

      info
```

```
 --+-------------------------------------------------------------------------------
------
 -------------------------------------------------------------------------------
------
 ------------
 1| PanWeiDB 是一种特性非常齐全的自由软件的对象-关系型数据库管理系统（ORDB
MS），PanWeiDB 支持大部分的 SQL 标准并且提供了很多其他现代特性，如复杂查询、
外键、触发器、视图、事务完整性、多版本并发控制等。同样，PostgreSQL 也可以用许
多方法扩展，例如通过增加新的数据类型、函数、操作符、聚集函数、索引方法、过程语
言等。另外，因为许可证的灵活，任何人都可以以任何目的免费使用、
 改和分发 PostgreSQL。
 (1 row)
```

9.7.PGroonga 插件

功能描述

PGroonga 是一个全文检索插件，PanWeiDB 已经内置了 PGroonga 扩展，您只需要创建扩展，即可使用 PGroonga 支持的语言进行超高速全文检索。

语法格式

1、使用 pgroonga()创建单字段全文索引。

```
CREATE INDEX pgroonga_index ON memos USING pgroonga(content1）;
```

❖ 使用&@操作符进行全文检索。

```
SELECT * FROM memos WHERE content &@ '全文搜索关键字';
```

❖ 使用&@~操作符执行含 pgroonga 查询语法的全文检索。

```
SELECT * FROM memos WHERE content2 &@~ 'PGroonga OR PostgreSQL';
```

❖ 使用 like 操作符（数据库本身的一种查询语法，区分大小写）执行全文检索。

```
SELECT * FROM memos WHERE content2 LIKE '%全文搜索关键字%';
```

❖ 使用 ilike 操作符（数据库本身的一种查询语法，不区分大小写）执行全文检索。

```
SELECT * FROM memos WHERE content2 ILIKE '%全文搜索关键字%';
```

2、使用 pgroonga()创建多字段全文索引。

```
CREATE INDEX pgroonga_index_1 ON memos USING pgroonga(content1, content2);
```

❖ 使用&@操作符进行多字段全文检索。

```
SELECT * FROM memos WHERE content1 &@ '全文搜索关键字' and content2 &@ '全文搜索关键字';
```

❖ 使用 like 操作符（数据库本身的一种查询语法，区分大小写）执行多字段全文检索。

```
SELECT * FROM memos WHERE content1 like '%全文搜索关键字%' and content2 like '%全文搜索关键字%';
```

❖ 使用 ilike 操作符（数据库本身的一种查询语法，不区分大小写）执行多字段全文检索。

```
SELECT * FROM memos WHERE content1 ilike '%全文搜索关键字%' and content2 ilike '%全文搜索关键字%';
```

使用 pgroonga_vacuum()函数清理临时数据。

```
select pgroonga_vacuum();
```

示例

1、创建插件。

修改 postgresql.conf 中配置 enable_pgroonga=true。

```
CREATE EXTENSION pgroonga;
```

2、关闭全表扫描，为了在少量数据下体现索引效果。

```
set enable_seqscan=off;
```

3、创建表，并插入数据。

```
CREATE TABLE memos (
id integer,
content text
);
INSERT INTO memos VALUES (1, 'PostgreSQL is a relational database management system.');
```

```
INSERT INTO memos VALUES (2, 'Groonga is a fast full text search engine that supp
orts all languages.');
INSERT INTO memos VALUES (3, 'PGroonga is a PostgreSQL extension that uses Gro
onga as index.');
INSERT INTO memos VALUES (4, 'There is groonga command.');
```

4、创建全文索引。

```
CREATE INDEX idxA ON memos USING pgroonga (content pgroonga_text_full_text_s
earch_ops_v2);
```

查看执行计划。

```
explain select * from memos where content LIKE '%groonga%';
```

结果显示如下：

```
                QUERY PLAN
-----------------------------------
 Index Scan using idxa on memos  (cost=0.00..4.01 rows=1 width=36)
   Index Cond: (content ~~ '%groonga%'::text)
(2 rows)
```

5、查看表中内容。

```
select * from memos where content LIKE '%groonga%';
```

结果显示如下：

```
 id |        content
----+---------------
  4 | There is groonga command.
(1 row)
```

使用 ILIKE 查询。

```
select * from memos where content ILIKE '%roonga%';
```

结果显示如下：

```
 id | content
----+-----
```

```
(0 rows)
```

6、查询验证。

❖ 使用&@操作符。

```
SELECT * FROM memos WHERE content &@ 'languages';
```

结果显示如下：

```
id | content
---+-----------------------------------
2 | Groonga is a fast full text search engine that supports all languages.
(1 row)
```

❖ 使用&@~操作符。

```
SELECT * FROM memos WHERE content &@~ 'PGroonga OR PostgreSQL';
```

结果显示如下：

```
id | content
---+-------------------------------
3 | PGroonga is a PostgreSQL extension that uses Groonga as index.
1 | PostgreSQL is a relational database management system.
(2 rows)
```

❖ 使用 like 操作符。

```
SELECT * FROM memos WHERE content LIKE '%groonga%';
```

结果显示如下：

```
id | content
---+---------------
4 | There is groonga command.
(1 row)
```

❖ 使用 ilike 操作符。

```
SELECT * FROM memos WHERE content ILIKE '%groonga%';
```

结果显示如下：

```
 id |                    content
----+-----------------------------------
```

```
 2 | Groonga is a fast full text search engine that supports all languages.
 3 | PGroonga is a PostgreSQL extension that uses Groonga as index.
 4 | There is groonga command.
(3 rows)
```

9.8.TimescaleDB 插件

概述

功能描述

PanWeiDB 数据库支持 TimescaleDB 时序插件，TimescaleDB 能够以插件化的形式，方便用户处理时序数据。

- TimescaleDB 是一个开源的时序数据库扩展。TimescaleDB 专门用于高性能和可扩展的时间序列存储和分析。它结合了关系型数据库的功能优势，以及时间序列数据库的特性，提供了一套强大的功能来处理大规模时间序列数据，TimescaleDB 在以下场景中非常适用：
- 物联网（IoT）应用：物联网应用通常会产生大量的时间序列数据，例如传感器数据、设备监控数据等。TimescaleDB 的高性能和数据分区功能可以有效地处理这些数据，并支持快速地实时查询和分析。
- 金融和交易数据：金融行业需要对加以数据进行高效地存储和分析。TimescaleDB 的连续聚合和数据保留策略功能可以方便地计算和维护聚合数据，同时自动删除过期的数据。
- 日志和监控数据：在日志和监控系统中，需要对大量的时间和指标数据进行存储和分析。TimescaleDB 的数据连续性和数据压缩功能可以满足高并发的写入需求，并减少存储空间的使用。
- 时间序列分析：对于需要进行时间序列分析场景，TimescaleDB 提供 SQL 接口和丰富的时间序列函数，可以方便地进行复杂的查询和分析操作。

TimescaleDB 超表和数据块

- **Hypertables（超表）**

  数据交互的重点是超表（Hypertables），跨越空间和时间间隔的单个连续表的抽象，使得可以通过普通的 SQL 查询它。

几乎所有与 TimescaleDB 的用户交互都是与超表。创建表和索引，更改表，插入数据，选择数据等可以（并且应该）全部在 hypertable 上执行。

超表由具有列名称和类型的标准模式定义，至少有一列指定时间值，另一列用于分区键(partitioning key)。

单个 TimescaleDB 部署可以存储多个超表，每个具有不同的结构。

创建一个 Hypertable 需要的两个简单的 SQL 命令：CREATE TABLE，其次是 SELECT create_hypertable()。

超表会自动创建时间和主键（Primary Key）的索引,也可以创建其他类型的索引。

- **Chunks(块)**

在内部，TimescaleDB 自动将 Hypertable 分割成块，每个块对应于一个特定的时间间隔和一个分区键。这些分区是不重合的，这有助于查询器进行查询.

每个块都使用标准数据库表实现。

合适的块大小，确保表索引的所有 B 树可以在插入期间驻留在内存中。这样可以避免在修改这些树中的任意位置时出现 thrashing。

此外，通过避免过大的块，我们可以根据自动保留策略，删除的数据时避免昂贵的 VACUUM 操作。运行时可以通过简单地删除块（内部表）来执行此类操作，而不是删除单个行。

使用指南

注意事项

- TimescaleDB 插件依赖于 PUBLIC SCHEMA，因此不支持使用 DROP SCHEMA 的方式删除 PUBLIC SCHEMA。
- TimescaleDB 创建的超表需要使用 DROP TABLE CASCADE 进行删除，会同时删除超表。
- TimescaleDB 不支持压缩表、压缩函数。

- 该功能仅在数据库兼容模式为 PG 时能够使用（即创建初始化数据库实例时指定 DBCOMPATIBILITY=PG），在其他数据库兼容模式下暂不支持使用该特性。

添加 TimescaleDB 插件

1、使用 TimescaleDB 插件前，需要在数据库配置文件（$PGDATA/postgresql.conf）中添加如下参数：

【说明】

对于没有配置环境隔离的集群，需要手动配置 $PGDATA 变量。

```
echo "shared_preload_libraries=\$libdir/timescaledb" >> $PGDATA/postgresql.conf
```

2、在数据库集群的各数据节点执行以上操作。

3、重启数据库集群，然后登录数据库主节点。

```
pw_ctl restart
```

4、在创建 TimescaleDB 插件。

```
CREATE EXTENSION TimescaleDB;
```

返回结果为：

添加时序表

1、创建标准表 conditions。

```
CREATE TABLE conditions (
 time      TIMESTAMPTZ      NOT NULL,
 location  TEXT             NOT NULL,
```

```
 temperature DOUBLE PRECISION  NULL,
 humidity    DOUBLE PRECISION  NULL
);
```

2、创建时序表。

```
SELECT create_hypertable('conditions', 'time');
```

返回结果为：

```
WARNING:  Trigger function with non-plpgsql type is not recommended.
DETAIL:  Non-plpgsql trigger function are not shippable by default.
HINT:  Unshippable trigger may lead to bad performance.
CONTEXT:  referenced column: create_hypertable
  create_hypertable
-------------------------
(19,public,conditions,t)
(1 row)
```

高效写入

1、使用标准 SQL 命令将数据插入超表（Hypertables）。

```
INSERT INTO conditions(time, location, temperature, humidity)
 VALUES (NOW(), 'office', 70.0, 50.0);
```

2、一次将多行数据插入到超表中。

```
INSERT INTO conditions
 VALUES
  (NOW(), 'office', 70.0, 50.0),
  (NOW(), 'basement', 66.5, 60.0),
  (NOW(), 'garage', 77.0, 65.2);
```

数据检索

可以使用高级 SQL 查询检索数据，例如：

```
--过去 3 小时内，每 15 分钟采集一次数据，按时间和温度排序。
SELECT time_bucket('15 minutes', time) AS fifteen_min,
  location, COUNT(*),
  MAX(temperature) AS max_temp,
```

```
  MAX(humidity) AS max_hum
 FROM conditions
 WHERE time > NOW() - interval '3 hours'
 GROUP BY fifteen_min, location
 ORDER BY fifteen_min DESC, max_temp DESC;
```

返回结果为：

```
    fifteen_min      | location | count | max_temp | max_hum
-----------------------+----------+-------+----------+---------
 2024-07-08 17:00:00+08 | garage   |     1 |       77 |    65.2
 2024-07-08 17:00:00+08 | office   |     2 |       70 |      50
 2024-07-08 17:00:00+08 | basement |     1 |     66.5 |      60
(3 rows)
```

也可以使用固有的函数进行分析查询，例如：

```
--均值查询（Median）
SELECT percentile_cont(0.5)
 WITHIN GROUP (ORDER BY temperature)
 FROM conditions;
```

返回结果为：

```
 percentile_cont
-----------------
              70
(1 row)
```

或者：

```
--移动平均数（Moving Average）
SELECT time, AVG(temperature) OVER(ORDER BY time
    ROWS BETWEEN 9 PRECEDING AND CURRENT ROW)
  AS smooth_temp
 FROM conditions
 WHERE location = 'garage' and time > NOW() - interval '1 day'
 ORDER BY time DESC;
```

返回结果为：

```
        time          | smooth_temp
------------------------------+-------------
 2024-07-08 17:00:24.843729+08 |        77
(1 row)
```

时间窗口聚集查询

time_bucket 函数可以对值为时间的字段分 bucket。即将时间戳数据划分到固定宽度的时间间隔内。

- 时间类型(DATE/TIMESTAMP/TIMESTAMP WITH TIME ZONE):

```
time_bucket(bucket_width interval,ts xxx)
time_bucket(bucket_width interval,ts xxx, "offset" xxx)
time_bucket(bucket_width xxx,ts xxx, origin xxx)
```

入参:

- bucket_width：必填参数，表示每个 bucket 的时间间隔。
- ts xxx：必填参数，xxx 表示同一字段类型，表示 bucket 的时间戳，函数对该值进行分 bucket。
- timezone：可选参数，用于计算 bucket 开始和结束时间的时区，只能与 timestamp with time zone 结合使用，默认为 UTC。
- origin：可选参数，bucket 相对于此时间戳对齐。
  - 若间隔值不包括月份或年份，则默认为 2000 年 1 月 3 日午夜。
  - 对于月份、年份和世纪存储 bucket，则默认为 2001 年 1 月 1 日午夜。
- offset：可选参数，bucket 偏移，表示要偏移所有时间段的时间间隔。

- 整数类型（integer/smallint/bigint 类型）：

```
time_bucket(bucket_width xxx,ts xxx)
time_bucket(bucket_width xxx,ts xxx, "offset" xxx)
```

入参:

- bucket_width xxx：必填参数，xxx 表示同一字段类型，表示每个 bucket 的时间间隔。
- ts xxx：必填参数，xxx 表示同一字段类型，表示 bucket 的时间戳。函数对该值进行分 bucket。
- offset：可选参数，bucket 偏移，表示要偏移所有时间段的时间间隔。

函数参考

❖ create_hypertable

创建超表，时序元数据表缺省以时间列分区。同时具备按照多列组合分区能力。

注意事项

- 仅支持在 Astore 存储引擎的表上创建超表。
- 不支持和线程池同时使用。

语法格式

```
create_hypertable( main_table
                 , time_column_name
                 [, partitioning_column
                  , number_partitions]
                 [, associated_schema_name]
                 [, associated_table_prefix]
                 [, chunk_time_interval]
                 [, create_default_indexes]
                 [, if_not_exists]
                 [, partitioning_func]
                 [, migrate_data]
                 [, chunk_target_size]
                 [, chunk_sizing_func]
                 [, time_partitioning_func])
```

参数说明

- main_table

  时序元数据表关联的物理表。

- time_column_name

  包含时间的主分区列。

- partitioning_column

  分区列，与 number_partitions 配合使用。

- number_partitions

  分区 partitioning_column 分区个数，需大于 0。

- chunk_time_interval

分区覆盖时间范围，需大于 0，缺省值为 NULL。

- create_default_indexes

  布尔值，用来确定是否在分区列上创建缺省索引。缺省为 TRUE。

- if_not_exists

  布尔值，用来确定当时序元数据表已经创建时，是否打印告警信息。缺省值 FALSE。

- partitioning_func

  分区计算函数。

- associated_schema_name

  时序元数据表内部名称。

- associated_table_prefix

  内部表分区前缀名，缺省值为 NULL。

- migrate_data

  布尔值，设置成 true 时，把 main_table 数据迁移到新的时序元数据表的分区数据块中，缺省值 false。

- chunk_target_size

  块的目标大小，例如 1000MB, estimate 或 off。

- chunk_sizing_func

  用于计算新块的块时间间隔的函数。

- time_partitioning_func

  用于 time 分区的分区函数。

示例

1、创建标准表 conditions_create。

```
CREATE TABLE conditions_create (
 time       TIMESTAMPTZ      NOT NULL,
 location    TEXT            NOT NULL,
 temperature DOUBLE PRECISION  NULL,
 humidity    DOUBLE PRECISION  NULL
);
```

2、创建时序表。

```
SELECT create_hypertable('conditions_create', 'time');
```

返回结果为：

```
HINT:  Unshippable trigger may lead to bad performance.
CONTEXT:  referenced column: create_hypertable
    create_hypertable
-------------------------------
 (7,public,conditions_create,t)
(1 row)
```

❖ set_chunk_time_interval

设置超表数据块的时间间隔。

语法格式

```
set_chunk_time_interval ( main_table
                , chunk_time_intelval
               [, dimension_name ])
```

参数说明

- main_table

  时序元数据表表名。

- chunk_time_interval

  数据块覆盖的时间区间，需大于 0。

- dimension_name

  时间分区名，有且只有当时序元数据表有多个分区时使用。

示例

1、创建测试表。

```
CREATE TABLE conditions_chunk (
time TIMESTAMPTZ NOT NULL,
location TEXT NOT NULL,
temperature DOUBLE PRECISION NULL,
humidity DOUBLE PRECISION NULL
);
```

2、创建超表，并插入数据。

```
SELECT create_hypertable('conditions_chunk', 'time');
INSERT INTO conditions_chunk (time, location, temperature, humidity)
```

```
VALUES ('2024/04/10 18:15:11.21', 'office4', 36.0, 90.65);
INSERT INTO conditions_chunk (time, location, temperature, humidity)
VALUES ('2024/04/09 18:05:04.221', 'office2', 37.5, 50.0);
INSERT INTO conditions_chunk (time, location, temperature, humidity)
VALUES ('2024/04/08 18:05:04.221', 'office3', 20, 30.0);
INSERT INTO conditions_chunk (time, location, temperature, humidity)
VALUES ('2024/04/08 16:05:04.221', 'office3', 22.5, 30.0);
INSERT INTO conditions_chunk (time, location, temperature, humidity)
VALUES ('2024/04/10 18:05:11.21', 'office4', -10.0, -1.3);
```

3、设置超表数据块的时间间隔。

```
SELECT set_chunk_time_interval('conditions_chunk' , INTERVAL '24 hours');
```

返回结果为：

```
 set_chunk_time_interval
-------------------------


(1 row)
```

❖ **set_number_partitions**

设置超表上空间维度的分区(片)数量。

语法格式

```
set_number_partitions ( main_table
                      , number_partitions
                     [, dimension_name])
```

参数说明

- main_table

  时序元数据表表名。

- number_partitions

  分区数量，取值范围： 1 ~ 32767 之间。

- dimension_name

  时间主分区外的其他分区键名。有且只有时序元数据表有多个分区键时使用。

示例

1、创建测试表。

```
CREATE TABLE conditions_part (
time TIMESTAMPTZ NOT NULL,
location TEXT NOT NULL,
temperature DOUBLE PRECISION NULL,
humidity DOUBLE PRECISION NULL
);
```

2、创建超表，并插入数据。

```
SELECT create_hypertable('conditions_part' , 'time');
```

3、为超表添加维度。

```
SELECT add_dimension('conditions_part', 'location', number_partitions =>
4);
```

返回结果为：

```
        add_dimension
--------------------------------------
 (12,public,conditions_part,location,t)
(1 row)
```

4、设置超表上空间维度的分区。

```
SELECT set_number_partitions('conditions_part', 5,'location');
```

返回结果为：

```
 set_number_partitions
----------------------


(1 row)
```

- **drop_chunks**

删除时间范围完全落在指定时间之前(或之后)的数据块，可以跨所有超表操作，也可以针对特定的超表操作。返回删除的块列表。

语法格式

```
drop_chunks( main_table
        [, older_than]
        [, table_name]
        [, schema_name]
        [, cascade]
        [, newer_than]
```

```
    [, verbose]
    [, cascade_to_materialization])
```

参数说明

- relation

  超表或连续聚合，从其中删除块。

- older_than

  删除任何比此时间戳更早的完整块，缺省表示选择所有时间。

- table_name

  删除与 table_name 表关联的块，缺省表示选择所有表。

- schema_name

  删除与 schema_name 模式关联的块，缺省表示选择所有模式。

- cascade

  级联删除，缺省为 FALSE。

- newer_than

  删除任何比此时间戳更晚的完整块。

- verbose

  设置为 true 将显示有关重新排序命令进度的信息。缺省为 false。

- cascade_to_materialization

  级联删除物化视图，缺省为 NULL。

示例

1、创建测试表

```
CREATE TABLE conditions_drop (
time TIMESTAMPTZ primary key,
location TEXT NOT NULL,
temperature DOUBLE PRECISION NULL,
humidity DOUBLE PRECISION NULL
);
```

2、创建超表，并插入数据。

```
SELECT create_hypertable('conditions_drop', 'time');
INSERT INTO conditions_drop(time, location, temperature, humidity)
VALUES ('2024/04/10 18:15:11.21', 'office4', 36.0, 90.65);
INSERT INTO conditions_drop(time, location, temperature, humidity)
```

```
VALUES ('2024/04/09 18:05:04.221', 'office2', 37.5, 50.0);
INSERT INTO conditions_drop(time, location, temperature, humidity)
VALUES ('2024/04/08 18:05:04.221', 'office3', 20, 30.0);
INSERT INTO conditions_drop(time, location, temperature, humidity)
VALUES ('2024/04/08 16:05:04.221', 'office3', 22.5, 30.0);
INSERT INTO conditions_drop(time, location, temperature, humidity)
VALUES ('2024/04/10 18:05:11.21', 'office4', -10.0, -1.3);
```

3、查询超表。

```
SELECT location FROM conditions_drop;
```

4、插入要在接下来的步骤中删除的数据。

```
INSERT INTO conditions_drop(time, location, temperature, humidity)
VALUES (now()+interval '4 months', 'office5', -10.0, -1.3);
```

5、删除块。

```
SELECT drop_chunks(newer_than => now() + interval '3 months');
SELECT location FROM conditions_drop;
```

返回结果为：

```
 location
----------
 office4
 office2
 office3
 office3
 office4
(5 rows)
```

❖ **show_chunks**

获取与超表关联的块的列表。

语法格式

```
show_chunks([  hyptertable]
        [, older_than]
        [, newer_than])
```

参数说明

➢ hyertable

展示指定超表的块，设置为 NULL 表示展示数据库中所有超表的块。

- older_than

  显示所有早于此时间戳的块。

- newer_than

  显示所有晚于此时间戳的块。

❖ **add_dimension**

添加额外的维度，选择作为维度的列既可以是 Interval 分区，也可以是 Hash 分区。

语法格式

```
add_dimension( main_table
        , column_name
        [, number_partitions]
        [, chunk_time_interval]
        [, partitioning_func]
        [, if_not_exist])
```

参数说明

- main_table

  添加新分区的时序元数据表。

- column_name

  分区列名。

- number_partitions

  column_name 列的分区个数，需大于 0。

- chunk_time_interval

  每个分区覆盖范围需大于 0。

- partitioning_func

  分区计算函数。

- if_not_exists

  布尔值，用来确定当时序元数据表已经创建时，是否打印告警信息，缺省值 FALSE。

示例

1、创建测试表。

```
CREATE TABLE conditions_dimn (
```

```
time TIMESTAMPTZ NOT NULL,
location TEXT NOT NULL,
temperature DOUBLE PRECISION NULL,
humidity DOUBLE PRECISION NULL
);
```

2、创建超表。

```
SELECT create_hypertable('conditions_dimn', 'time');
```

3、为超表添加维度。

```
SELECT add_dimension('conditions_dimn', 'location', number_partitions =>
 4);
```

❖ attach_tablespace

为超表关联表空间，并使用它来存储块。

TimescaleDB 可以为每个超表管理一组表空间，自动将块扩展到附加到超表的表空间集。如果超表为 Hash 分区，TimescaleDB 会将尝试将属于同一分区的块放在同一表空间中。未指定表空间的超表会使用缺省表空间。

语法格式

```
attach_tablespace( tablespace
          , hypertable
         [,  if_not_attached ])
```

参数说明

- tablespace

  为超表指定的表空间。

- hypertable

  关联表空间的超表。

- if_not_attached

  如果表空间已经附加到表，则设置为 true 可避免抛出错误，而是发出通知。缺省为 false。

示例

1、创建超表。

```
CREATE TABLE conditions_attach (
time TIMESTAMPTZ NOT NULL,
```

```
location TEXT NOT NULL,
temperature DOUBLE PRECISION NULL,
humidity DOUBLE PRECISION NULL
);
SELECT create_hypertable('conditions_attach', 'time');
```

2、为超表关联表空间。

```
CREATE TABLESPACE tbs01 RELATIVE LOCATION 'tablespace/tablespace_1';
SELECT attach_tablespace('tbs01', 'conditions_attach');
```

3、删除测试表空间。

```
SELECT detach_tablespace('tbs01', 'conditions_attach');
DROP TABLESPACE tbs01;
```

❖ detach_tablespace

与已关联的表空间分离，这只意味着新的块不会被放置在分离的表空间上。分离的表空间本身和上面有数据的任何现有块都将保持不变，并将继续像以前一样使用，包括可用于查询。

新插入的数据行仍然可以插入到被分离的表空间上已有的块中，因为已有数据不会从分离表空间中清除。如果需要再次考虑块放置，需要重新关联该表空间。

语法格式

```
attach_tablespace(  tablespace
            [, hypertable]
            [,  if_not_attached])
```

参数说明

- tablespace

  与超表分离的表空间。

- hypertable

  分离表空间的超表。

- if_not_attached

  如果表空间已经附加到表，则设置为 true 可避免抛出错误，而是发出通知。缺省为 false。

示例

1、创建超表。

```
CREATE TABLE conditions_detach (
time TIMESTAMPTZ NOT NULL,
location TEXT NOT NULL,
temperature DOUBLE PRECISION NULL,
humidity DOUBLE PRECISION NULL
);
SELECT create_hypertable('conditions_detach', 'time');
```

2、关联、分离表空间。

```
CREATE TABLESPACE tbs01 RELATIVE LOCATION 'tablespace/tablespace_1';
SELECT attach_tablespace('tbs01', 'conditions_detach');
SELECT detach_tablespace('tbs01', 'conditions_detach');
DROP TABLESPACE tbs01;
```

❖ detach_tablespaces

从超表中分离所有表空间。在超表上发出此命令后，它不再附加任何表空间。新的块被放置在数据库的缺省表空间中。

语法格式

```
detach_tablespaces(hypertable)
```

参数说明

- hypertable

  解除关联表空间的超表。

❖ show_tablespaces

展示与超表关联的表空间。

语法格式

```
show_tablespaces(hypertable)
```

参数说明

- hypertable

  要展示表空间信息的的超表。

示例

```
CREATE TABLE conditions_show (
time TIMESTAMPTZ NOT NULL,
location TEXT NOT NULL,
```

```
temperature DOUBLE PRECISION NULL,
humidity DOUBLE PRECISION NULL
);
SELECT create_hypertable('conditions_show', 'time');
CREATE TABLESPACE tbs01 RELATIVE LOCATION 'tablespace/tablespace_1';
SELECT attach_tablespace('tbs01', 'conditions_show');
SELECT show_tablespaces('conditions_show');
```

返回结果为：

```
 show_tablespaces
------------------
 tbs01
(1 row)
```

- **time_bucket**

该函数是 data_trunc 函数的增强版本。该函数允许任意的时间间隔。

语法格式

```
--对于 INTERVAL 类型的 bucket 宽度
 time_bucket(  bucket_width
          , ts
         [, origin]
         [, offset])

--对于整型的 bucket 宽度
 time_bucket(  bucket_width
          , ts
         [, offset])
```

参数说明

- bucket_width

  bucket 宽度，支持的数据类型为 INTERVAL、SMALLINT、INT 与 BIGINT。

- ts

  bucket 的时间戳，支持的数据类型为 TIMESTAMP、TIMESTAMPTZ、DATE、SMALLINT、INT 与 BIGINT。

- origin

bucket 相对于该参数指定的时间戳对齐。仅 INTERVAL 类型的可以指定该参数

- offset

  所有 bucket 要偏移的数量。

示例

1、创建测试表。

```
CREATE TABLE conditions_bucket (
      time       TIMESTAMPTZ      NOT NULL,
      location    TEXT            NOT NULL,
      temperature DOUBLE PRECISION  NULL,
      humidity    DOUBLE PRECISION  NULL);

SELECT create_hypertable('conditions_bucket', 'time');
```

2、一次将多行数据插入到超表中。

```
INSERT INTO conditions_bucket
 VALUES (NOW(), 'office', 70.0, 50.0),
  (NOW(), 'office', 70.0, 50.0),
  (NOW(), 'basement', 66.5, 60.0),
  (NOW(), 'garage', 77.0, 65.2);
```

3、查询过去 3 小时内，每 15 分钟采集一次数据，按时间和温度排序。

```
SELECT time_bucket('15 minutes', time) AS fifteen_min,
  location, COUNT(*),
  MAX(temperature) AS max_temp,
  MAX(humidity) AS max_hum
 FROM conditions_bucket
 WHERE time > NOW() - interval '3 hours'
 GROUP BY fifteen_min, location
 ORDER BY fifteen_min DESC, max_temp DESC;
```

返回结果为：

```
    fifteen_min      | location | count | max_temp | max_hum
-----------------------+----------+-------+----------+---------
 2024-07-04 14:45:00+08 | garage   |     1 |       77 |    65.2
 2024-07-04 14:45:00+08 | office   |     2 |       70 |      50
```

```
2024-07-04 14:45:00+08 | basement |    1 |    66.5 |    60
(3 rows)
```

❖ hypertable_relation_size

获得时序元数据表的数据块关系。返回值是一张表。

语法格式

```
hypertable_relation_size(main_table);
```

参数说明

➢ main_table

时序元数据表表名。

示例

1、创建表。

```
CREATE TABLE conditions_size (
time TIMESTAMPTZ primary key,
location TEXT NOT NULL,
temperature DOUBLE PRECISION NULL,
humidity DOUBLE PRECISION NULL
);
```

2、创建超表，并插入数据。

```
SELECT create_hypertable('conditions_size', 'time');
INSERT INTO conditions_size(time, location, temperature, humidity)
VALUES ('2024/04/10 18:15:11.21', 'office4', 36.0, 90.65);
INSERT INTO conditions_size(time, location, temperature, humidity)
VALUES ('2024/04/09 18:05:04.221', 'office2', 37.5, 50.0);
INSERT INTO conditions_size(time, location, temperature, humidity)
VALUES ('2024/04/08 18:05:04.221', 'office3', 20, 30.0);
INSERT INTO conditions_size(time, location, temperature, humidity)
VALUES ('2024/04/08 16:05:04.221', 'office3', 22.5, 30.0);
INSERT INTO conditions_size(time, location, temperature, humidity)
VALUES ('2024/04/10 18:05:11.21', 'office4', -10.0, -1.3);
```

3、查询时序元数据表的数据块关系。

```
select hypertable_relation_size('conditions_size');
```

返回结果为：

```
 hypertable_relation_size
```

```
-------------------------
 (8192,16384,8192,32768)
(1 row)
```

❖ hypertable_relation_size_pretty

与 hypertable_relation_size()功能相同，显示格式不同。

语法格式

```
hypertable_relation_size_pretty(main_table);
```

参数说明

➢ main_table

时序元数据表表名。

示例

1、创建测试表。

```
CREATE TABLE conditions_size_prty (
time TIMESTAMPTZ NOT NULL,
location TEXT NOT NULL,
temperature DOUBLE PRECISION NULL,
humidity DOUBLE PRECISION NULL
);
```

2、创建超表，并插入数据。

```
SELECT create_hypertable('conditions_size_prty', 'time');
INSERT INTO conditions_size_prty(time, location, temperature, humidity)
VALUES ('2024/04/10 18:15:11.21', 'office4', 36.0, 90.65);
INSERT INTO conditions_size_prty(time, location, temperature, humidity)
VALUES ('2024/04/09 18:05:04.221', 'office2', 37.5, 50.0);
INSERT INTO conditions_size_prty(time, location, temperature, humidity)
VALUES ('2024/04/08 18:05:04.221', 'office3', 20, 30.0);
INSERT INTO conditions_size_prty(time, location, temperature, humidity)
VALUES ('2024/04/08 16:05:04.221', 'office3', 22.5, 30.0);
INSERT INTO conditions_size_prty(time, location, temperature, humidity)
VALUES ('2024/04/10 18:05:11.21', 'office4', -10.0, -1.3);
```

3、查询时序元数据表的关系表。

```
select hypertable_relation_size_pretty('conditions_size_prty');
```

返回结果为：

```
    hypertable_relation_size_pretty
-----------------------------------------
("8192 bytes","16 kB","8192 bytes","32 kB")
(1 row)
```

❖ chunk_relation_size

获得时序元数据表的数据块关系。返回值是一张表。

语法格式

```
chunk_relation_size(main_table);
```

参数说明

➢ main_table

时序元数据表表名。

示例

1、创建表。

```
CREATE TABLE conditions_rel_size (
time TIMESTAMPTZ primary key,
location TEXT NOT NULL,
temperature DOUBLE PRECISION NULL,
humidity DOUBLE PRECISION NULL
);
```

2、创建超表

```
SELECT create_hypertable('conditions_rel_size', 'time');
INSERT INTO conditions_rel_size(time, location, temperature, humidity)
VALUES ('2024/04/10 18:15:11.21', 'office4', 36.0, 90.65);
INSERT INTO conditions_rel_size(time, location, temperature, humidity)
VALUES ('2024/04/09 18:05:04.221', 'office2', 37.5, 50.0);
INSERT INTO conditions_rel_size(time, location, temperature, humidity)
VALUES ('2024/04/08 18:05:04.221', 'office3', 20, 30.0);
INSERT INTO conditions_rel_size(time, location, temperature, humidity)
VALUES ('2024/04/08 16:05:04.221','office3', 22.5, 30.0);
INSERT INTO conditions_rel_size(time, location, temperature, humidity)
VALUES ('2024/04/10 18:05:11.21', 'office4', -10.0, -1.3);
```

3、查询时序元数据表的数据块关系。

```
select chunk_relation_size('conditions_rel_size');
```

返回结果为：

```
                    chunk_relation_size
-------------------------------------------------------------------------------
 (8,_timescaledb_internal._hyper_13_8_chunk,{time},"{""timestamp with ti
me zone""}",{NULL},"{""[1712188800000000,1712793600000000)""}",8192,1
6384,81
92,32768)
(1 row)
```

❖ chunk_relation_size_pretty

与 chunk_relation_size()功能相同，显示格式不同。

语法格式

```
chunk_relation_size_pretty(main_table)
```

参数说明

- main_table

  时序元数据表表名。

示例

1、创建表。

```
CREATE TABLE conditions_rel_prty (
time TIMESTAMPTZ primary key,
location TEXT NOT NULL,
temperature DOUBLE PRECISION NULL,
humidity DOUBLE PRECISION NULL
);
```

2、创建超表，插入数据。

```
SELECT create_hypertable('conditions_rel_prty', 'time');
INSERT INTO conditions_rel_prty(time, location, temperature, humidity)
VALUES ('2024/04/10 18:15:11.21', 'office4', 36.0, 90.65);
INSERT INTO conditions_rel_prty(time, location, temperature, humidity)
VALUES ('2024/04/09 18:05:04.221', 'office2', 37.5, 50.0);
INSERT INTO conditions_rel_prty(time, location, temperature, humidity)
VALUES ('2024/04/08 18:05:04.221', 'office3', 20, 30.0);
INSERT INTO conditions_rel_prty(time, location, temperature, humidity)
VALUES ('2024/04/08 16:05:04.221', 'office3', 22.5, 30.0);
```

```
INSERT INTO conditions_rel_prty(time, location, temperature, humidity)
VALUES ('2024/04/10 18:05:11.21', 'office4', -10.0, -1.3);
```

3、查询时序元数据表的数据块关系。

```
select chunk_relation_size_pretty('conditions_rel_prty');
```

返回结果为：

```
                chunk_relation_size_pretty
-------------------------------------------------------------------
 (2,_timescaledb_internal._hyper_6_2_chunk,{time},"{""timestamp with tim
e zone""}",{NULL},"{""[2024-04-04 08:00:00+08,2024-04-11 08:00:00+08)""}
","8192 bytes","16 kB","8192 bytes","32 kB")
(1 row)
```

❖ indexes_relation_size

获得时序元数据表索引大小。

语法格式

```
indexes_relation_size(main_table);
```

参数说明

➢ main_table

时序元数据表表名。

示例

1、创建测试表。

```
CREATE TABLE conditions_idx_size (
time TIMESTAMPTZ NOT NULL,
location TEXT NOT NULL,
temperature DOUBLE PRECISION NULL,
humidity DOUBLE PRECISION NULL
);
```

2、创建超表，并插入数据。

```
SELECT create_hypertable('conditions_idx_size', 'time');
INSERT INTO conditions_idx_size(time, location, temperature, humidity)
VALUES ('2024/04/10 18:15:11.21', 'office4', 36.0, 90.65);
INSERT INTO conditions_idx_size(time, location, temperature, humidity)
VALUES ('2024/04/09 18:05:04.221', 'office2', 37.5, 50.0);
INSERT INTO conditions_idx_size(time, location, temperature, humidity)
```

```
VALUES ('2024/04/08 18:05:04.221', 'office3', 20, 30.0);
INSERT INTO conditions_idx_size(time, location, temperature, humidity)
VALUES ('2024/04/08 16:05:04.221', 'office3', 22.5, 30.0);
INSERT INTO conditions_idx_size(time, location, temperature, humidity)
VALUES ('2024/04/10 18:05:11.21', 'office4', -10.0, -1.3);
```

3、创建索引。

```
create index ix_conditions_time on conditions_idx_size(time);
create unique index ix_conditions_time_uni on conditions_idx_size(time);
```

4、查看序元数据表索引大小。

```
select indexes_relation_size('conditions_idx_size');
```

返回结果为：

```
        indexes_relation_size
-------------------------------------------
 (public.ix_conditions_time,16384)
 (public.conditions_idx_size_time_idx,16384)
 (public.ix_conditions_time_uni,16384)
(3 rows)
```

❖ indexes_relation_size_pretty

与 indexes_relation_size()功能相同，显示格式不同。

语法格式

```
indexes_relation_size_pretty(main_table);
```

参数说明

- main_table

  时序元数据表表名。

示例

1、创建测试表。

```
CREATE TABLE conditions_idx_prty (
time TIMESTAMPTZ NOT NULL,
location TEXT NOT NULL,
temperature DOUBLE PRECISION NULL,
humidity DOUBLE PRECISION NULL
);
```

2、创建超表，并插入数据。

```
SELECT create_hypertable('conditions_idx_prty', 'time');
INSERT INTO conditions_idx_prty(time, location, temperature, humidity)
VALUES ('2024/04/10 18:15:11.21', 'office4', 36.0, 90.65);
INSERT INTO conditions_idx_prty(time, location, temperature, humidity)
VALUES ('2024/04/09 18:05:04.221', 'office2', 37.5, 50.0);
INSERT INTO conditions_idx_prty(time, location, temperature, humidity)
VALUES ('2024/04/08 18:05:04.221', 'office3', 20, 30.0);
INSERT INTO conditions_idx_prty(time, location, temperature, humidity)
VALUES ('2024/04/08 16:05:04.221', 'office3', 22.5, 30.0);
INSERT INTO conditions_idx_prty(time, location, temperature, humidity)
VALUES ('2024/04/10 18:05:11.21', 'office4', -10.0, -1.3);
```

3、创建索引。

```
create index idx_conditions_time on conditions_idx_prty(time);
create unique index idx_conditions_time_uni on conditions_idx_prty(time);
```

4、查看序元数据表索引大小。

```
select indexes_relation_size_pretty('conditions_idx_prty');
```

返回结果为：

```
      indexes_relation_size_pretty
-----------------------------------------------
(public.conditions_idx_prty_time_idx,"16 kB")
(public.idx_conditions_time_uni,"16 kB")
(public.idx_conditions_time,"16 kB")
(3 rows)
```

❖ hypertable_approximate_row_count

获取超表或连续聚合使用的大致总磁盘空间，即表本身的大小之和，包括块、表上的任何索引和任何 toast 表。报告的大小以单位为字节。

该函数相当于从 hypertable_approximate_tailed_size 函数的输出计算 total_bytes 列的和。

语法格式

```
hypertable_approximate_row_count([main_table])
```

参数说明

- main_table

  要计算总磁盘空间的超表。

示例

1、创建测试表。

```
CREATE TABLE conditions_row_cnt (
time TIMESTAMPTZ NOT NULL,
location TEXT NOT NULL,
temperature DOUBLE PRECISION NULL,
humidity DOUBLE PRECISION NULL
);
```

2、创建超表，并插入数据。

```
SELECT create_hypertable('conditions_row_cnt', 'time');
INSERT INTO conditions_row_cnt(time, location, temperature, humidity)
VALUES ('2024/04/10 18:15:11.21', 'office4', 36.0, 90.65);
INSERT INTO conditions_row_cnt(time, location, temperature, humidity)
VALUES ('2024/04/09 18:05:04.221', 'office2', 37.5, 50.0);
INSERT INTO conditions_row_cnt(time, location, temperature, humidity)
VALUES ('2024/04/08 18:05:04.221', 'office3', 20, 30.0);
INSERT INTO conditions_row_cnt(time, location, temperature, humidity)
VALUES ('2024/04/08 16:05:04.221', 'office3', 22.5, 30.0);
INSERT INTO conditions_row_cnt(time, location, temperature, humidity)
VALUES ('2024/04/10 18:05:11.21', 'office4', -10.0, -1.3);
```

3、查询超表使用的总磁盘空间。

```
SELECT hypertable_approximate_row_count('conditions_row_cnt');
```

返回结果为：

```
 hypertable_approximate_row_count
----------------------------------
 (public,conditions_row_cnt,0)
(1 row)
```

9.9.uuid-ossp 插件

功能描述

uuid-ossp 模块提供了一些函数用来生成通用唯一识别码（UUID），兼容 uuid-ossp1.1，使用该模块提供的函数，采用几种标准算法之一产生通用唯一标识符（UUID）。

兼容的 uuid-ossp 模块的函数分为如下两类：

❖ 用于产生 uuid 的函数：

函数	描述
uuid_generate_v1()	这个函数产生一个版本 1 的 UUID。这涉及到计算机的 MAC 地址和一个时间戳。注：这种 UUID 会泄露产生该标识符的计算机标识以及产生的时间，因此它不适合某些对安全性很敏感的应用。
uuid_generate_v1mc()	这个函数产生一个版本 1 的 UUID，但是使用一个随机广播 MAC 地址而不是该计算机真实的 MAC 地址。
uuid_generate_v3(namespace uuid, name text)	这个函数使用指定的输入名称在给定名字的空间中产生一个版本 3 的 UUID。该名字空间应该是由 uuid_ns_*()函数产生的特殊常量之一（理论上它可以是任意 UUID）。名称是选择的名字空间中的一个标识符。例如：panweidb=# SELECT uuid_generate_v3(uuid_ns_url(), 'http://www.postgresql.org'); 名称参数将使用 MD5 进行哈希，因此从产生的 UUID 中得不到明文。采用这种方法的 UUID 生成没有随机性并且不涉及依赖于环境的元素，因此是可以重现的。
uuid_generate_v4()	这个函数产生一个版本 4 的 UUID，它完全从随机数产生。
uuid_generate_v5(namespace uuid, name text)	这个函数产生一个版本 5 的 UUID，它和版本 3 的 UUID 相似，但是采用的是 SHA-1 作为哈希方法。注：版本 5 比版本 3 更好，因为 SHA-1 比 MD5 更安全。

❖ 返回 uuid 常量的函数：

函数	描述
uuid_nil()	一个"nil" UUID 常量，它不作为一个真正的 UUID 发生。
uuid_ns_dns()	为 UUID 指定 DNS 名字空间的常量。
uuid_ns_url()	为 UUID 指定 URL 名字空间的常量。

函数	描述
uuid_ns_oid()	为 UUID 指定 ISO 对象标识符（OID） 名字空间的常量（这属于 ASN.1 OID，它与 PostgreSQL 使用的 OID 无关）。
uuid_ns_x500()	为 UUID 指定 X.500 可识别名（DN）名字空间的常量。Constant designating the X.500 distinguished name (DN) namespace for UUIDs。

注意事项

- PanWeiDB V2.0-S3.0.1_B01 版本兼容 uuid-ossp 插件目前仅兼容 ossp，对 bsd 和 e2fs 不做硬性要求。
- PanWeiDB V2.0-S3.0.1_B01 已经内置了 uuid-ossp 扩展，您只需要创建扩展即可使用。

示例

示例一：

1、创建 uuid-ossp 扩展。

```
create extension "uuid-ossp";
```

2、使用产生 uuid 的函数。

- 测试 uuid_generate_v1()。

```
select uuid_generate_v1();
```

结果显示如下：

```
          uuid_generate_v1
--------------------
 bdf064b8-baef-11ec-8838-000c2948c478
 (1 row)
```

- 测试 uuid_generate_v1mc()。

```
select uuid_generate_v1mc();
```

结果显示如下：

```
        uuid_generate_v1mc
--------------------
```

```
 ce8693b0-baef-11ec-8839-83581df53531
(1 row)
```

❖ 测试 uuid_generate_v3(namespace uuid, name text)。

```
SELECT uuid_generate_v3(uuid_ns_url(),'http://www.postgresql.org');
```

结果显示如下：

```
          uuid_generate_v3
--------------------
 cf16fe52-3365-3a1f-8572-288d8d2aaa46
(1 row)
```

❖ 测试 uuid_generate_v4()。

```
select uuid_generate_v4();
```

结果显示如下：

```
          uuid_generate_v4
--------------------
 3cad2542-a2a5-43ed-9df4-0dacb2f1d52f
(1 row)
```

❖ 测试 uuid_generate_v5(namespace uuid, name text)。

```
SELECT uuid_generate_v5(uuid_ns_url(),'http://www.postgresql.org');
```

结果显示如下：

```
         uuid_generate_v5
--------------------
 e1ee1ad4-cd4e-5889-962a-4f605a68d94e
(1 row)
```

示例二：返回 uuid 常量的函数

❖ 测试 uuid_nil()函数。

```
select uuid_nil();
```

结果显示如下：

```
              uuid_nil
--------------------
```

```
 00000000-0000-0000-0000-000000000000
(1 row)
```

❖ 测试 uuid_ns_dns()函数。

```
select uuid_ns_dns();
```

结果显示如下：

```
           uuid_ns_dns
--------------------
 6ba7b810-9dad-11d1-80b4-00c04fd430c8
(1 row)
```

❖ 测试 uuid_ns_url()函数。

```
select uuid_ns_url();
```

结果显示如下：

```
           uuid_ns_url
--------------------
 6ba7b811-9dad-11d1-80b4-00c04fd430c8
(1 row)
```

❖ 测试 uuid_ns_oid()函数。

```
select uuid_ns_oid();
```

结果显示如下：

```
           uuid_ns_oid
--------------------
 6ba7b812-9dad-11d1-80b4-00c04fd430c8
(1 row)
```

❖ 测试 uuid_ns_x500()函数。

```
select uuid_ns_x500();
```

结果显示如下：

```
           uuid_ns_x500
--------------------
 6ba7b814-9dad-11d1-80b4-00c04fd430c8
(1 row)
```

9.10.wal2json 插件

功能描述

wal2json 是逻辑解码插件，使用该插件可以访问 INSERT 和 UPDATE 生成的元组，解析 WAL 中的内容。wal2json 插件会在每个事务中生成一个 JSON 对象。JSON 对象总提供了所有新/旧元组，额外选项还可以包括事务时间戳，限定架构，数据类型，事务 ID 等属性。

wal2json 支持的参数列表如下：

选项	描述
include-xids	添加 xid 信息，默认 false。
include-timestamp	添加 timestamp 信息，默认 false。
include-schemas	添加 schemas 信息，默认 true。
include-types	添加 type 信息，默认 true。
include-typmod	添加 typmod 信息，默认 true。
include-type-oids	添加 type oids 信息，默认 false。
include-domain-data-type	将域名替换成数据类型名，默认 false。
include-column-positions	添加 column position 信息，默认 false。
include-origin	添加 origin 信息，默认 false。

include-not-null	添加 not nul1 信息，默认 false。
include-default	添加默认值表达式，默认为 false。
include-pk	添加主键信息，默认为 false。
pretty-print	更好的输出 ison 结构，默认为 false。
include-lsn	添加 nextlsn 信息，默认为 false。
write-in-chunks	在每次更改后写入，而不是在每次更改集之后写入。只在 format-version 为 1 的时候有用，默认为 false。
filter-origins	排除指定 origin 的更改，默认为空不过滤，多个 origin 用逗号分隔。
filter-tables	排除指定 table 的更改，默认为空不过滤，多个 table 用逗号分隔。
add_tables	仅包含指定 table 的更改，默认值是所有的表，与 filter-tables 值具有相同的规则。
format-version	格式化方式，可选值为 1 和 2，默认为 1。 ❖ 设置为 1 表示将整个事务内的所有 change 变更组织为一个 json 格式字符串进行输出。 ❖ 设置为 2 表示 表示对于每一个 change 都输出独立的 json 格式字符串。
actions	解析的 actions，默认是所有的 action（insert，update，delete）。

注意事项

❖ GUC 参数 wal_level 需要设置为 logical。

❖ 设置合适的 max_wal_senders 和 max_replication_slots 值。

❖ 暂不支持 truncate 和 message 操作。

使用流程

使用逻辑复制槽和解码插件解析数据库中 wal 日志记录的操作流程如下：

1、修改配置文件相关参数。

2、创建逻辑复制槽。

3、执行 SQL 语句。

4、解析 wal 内容。

该流程中涉及的函数请参见：《PanWeiDB V2.0-S3.0.1_B01 开发者指南》-->SQL 语法参考->函数和操作符->逻辑复制函数

【注意】

- 若不指定 write-in-chunks，则使用 format-version=1 会将整个事务内的所有 change 变更组织为一个 json 格式字符串进行输出，超过代码内部字符串长度限制（1G）时将出现内存溢出的错误。
- 为避免内存溢出的问题，可以指定 fromat-version=2，表示对于每一个 change 都输出独立的 json 格式字符串。

示例

1、修改相关参数（修改后需重启数据库实例）。

```
ALTER SYSTEM SET wal_level to logical;
ALTER SYSTEM SET max_wal_senders to 10;
ALTER SYSTEM SET max_replication_slots to 10;
```

2、创建逻辑复制槽。

```
select 'init' from pg_create_logical_replication_slot('regression_slot_1098463','wal2json');
```

3、执行 SQL 语句。

```
create table tb_1098463(col1 int ,col2 bigint default '1',col3 text default 'a',col4 int );
```

```
insert into tb_1098463 values(generate_series(1,3),1,'a');
insert into tb_1098463(col1) values(4);
insert into tb_1098463 values(5,4,'a');
insert into tb_1098463 select * from tb_1098463;
```

4、解析 wal 内容。

```
select data from pg_logical_slot_peek_changes('regression_slot_1098463',null,null,
'actions','insert');
```

返回结果为：

```
                                                              data
---------------------------------------------------------------------
 {"change":[]}
 {"change":[{"kind":"insert","schema":"public","table":"tb_1098463","columnnames
":["col1","col2","col3","col4"],"columntype
s":["integer","bigint","text","integer"],"columnvalues":[1,1,"a",null]},{"kind":"insert",
"schema":"public","table":"tb_1098
463","columnnames":["col1","col2","col3","col4"],"columntypes":["integer","bigint",
"text","integer"],"columnvalues":[2,1,"a
",null]},{"kind":"insert","schema":"public","table":"tb_1098463","columnnames":["c
ol1","col2","col3","col4"],"columntypes":
["integer","bigint","text","integer"],"columnvalues":[3,1,"a",null]}]}
 {"change":[{"kind":"insert","schema":"public","table":"tb_1098463","columnnames
":["col1","col2","col3","col4"],"columntype
s":["integer","bigint","text","integer"],"columnvalues":[4,1,"a",null]}]}
 {"change":[{"kind":"insert","schema":"public","table":"tb_1098463","columnnames
":["col1","col2","col3","col4"],"columntype
s":["integer","bigint","text","integer"],"columnvalues":[5,4,"a",null]}]}
 {"change":[{"kind":"insert","schema":"public","table":"tb_1098463","columnnames
":["col1","col2","col3","col4"],"columntype
s":["integer","bigint","text","integer"],"columnvalues":[1,1,"a",null]},{"kind":"insert",
"schema":"public","table":"tb_1098
463","columnnames":["col1","col2","col3","col4"],"columntypes":["integer","bigint",
"text","integer"],"columnvalues":[2,1,"a
```

```
",null]},{"kind":"insert","schema":"public","table":"tb_1098463","columnnames":["c
ol1","col2","col3","col4"],"columntypes":
["integer","bigint","text","integer"],"columnvalues":[3,1,"a",null]},{"kind":"insert","sc
hema":"public","table":"tb_1098463
","columnnames":["col1","col2","col3","col4"],"columntypes":["integer","bigint","tex
t","integer"],"columnvalues":[4,1,"a",n
ull]},{"kind":"insert","schema":"public","table":"tb_1098463","columnnames":["col1
","col2","col3","col4"],"columntypes":["i
nteger","bigint","text","integer"],"columnvalues":[5,4,"a",null]}]}
(5 rows)
```

5、解析 wal 内容添加 xid（include-xids 选项测试）。

```
select data from pg_logical_slot_peek_changes('regression_slot_1098463',null,null,
'include-xids','true');
```

返回结果为：

```
                    data
-------------------------------------------------------------

 {"xid":15504,"change":[]}
 {"xid":15505,"change":[{"kind":"insert","schema":"public","table":"tb_1098463","co
lumnnames":["col1","col2","col3","col4"]
,"columntypes":["integer","bigint","text","integer"],"columnvalues":[1,1,"a",null]},{"k
ind":"insert","schema":"public","tab
le":"tb_1098463","columnnames":["col1","col2","col3","col4"],"columntypes":["inte
ger","bigint","text","integer"],"columnval
ues":[2,1,"a",null]},{"kind":"insert","schema":"public","table":"tb_1098463","column
names":["col1","col2","col3","col4"],"c
olumntypes":["integer","bigint","text","integer"],"columnvalues":[3,1,"a",null]}]}
 {"xid":15506,"change":[{"kind":"insert","schema":"public","table":"tb_1098463","co
lumnnames":["col1","col2","col3","col4"]
,"columntypes":["integer","bigint","text","integer"],"columnvalues":[4,1,"a",null]}]}
 {"xid":15507,"change":[{"kind":"insert","schema":"public","table":"tb_1098463","co
lumnnames":["col1","col2","col3","col4"]
,"columntypes":["integer","bigint","text","integer"],"columnvalues":[5,4,"a",null]}]}
```

```
{"xid":15508,"change":[{"kind":"insert","schema":"public","table":"tb_1098463","co
lumnnames":["col1","col2","col3","col4"]
,"columntypes":["integer","bigint","text","integer"],"columnvalues":[1,1,"a",null]},{"k
ind":"insert","schema":"public","tab
le":"tb_1098463","columnnames":["col1","col2","col3","col4"],"columntypes":["inte
ger","bigint","text","integer"],"columnval
ues":[2,1,"a",null]},{"kind":"insert","schema":"public","table":"tb_1098463","column
names":["col1","col2","col3","col4"],"c
olumntypes":["integer","bigint","text","integer"],"columnvalues":[3,1,"a",null]},{"kin
d":"insert","schema":"public","table"
:"tb_1098463","columnnames":["col1","col2","col3","col4"],"columntypes":["integer
","bigint","text","integer"],"columnvalues
":[4,1,"a",null]},{"kind":"insert","schema":"public","table":"tb_1098463","columnna
mes":["col1","col2","col3","col4"],"colu
mntypes":["integer","bigint","text","integer"],"columnvalues":[5,4,"a",null]}]}
(5 rows)
```

6、删除逻辑复制槽。

```
select 'stop' from pg_drop_replication_slot('regression_slot_1098463');
drop table tb_1098463;
```

9.11.WalMiner

功能描述

WalMiner 是从 WAL(write ahead logs)日志中解析出执行的 SQL 语句的工具，并能生成对应的 undo SQL 语句。与传统的 logical decode 插件相比，WalMiner 不要求 logical 日志级别且解析方式较为灵活。目前 WalMiner 支持解析系统表的变化，可以分析出 DDL 语句引起的系统表的变化。详细功能如下：

- 从产生 WAL 日志的数据库中直接执行解析。包含以下两个方面：
  - 能够恢复 DATABASE 删除后的数据(数据字典完整，WAL 完整，操作发生后即刻停库)，即被删除 DATABASE 内的所有用户或部分数据(根据可恢复条件)。
  - 能够恢复 SCHEMA 删除后的数据(数据字典完整，WAL 完整，操作发

生后即刻停库)，即被删除 SCHEMA 下属的所有用户数据(根据可恢复条件)。

- 从非 WAL 产生的数据库中执行 WAL 日志解析。
- DDL 解析。支持 DROP TABLE，VACUUM FULL,TRUNCATETABLE 等 DDL 语句。对应场景能够识别此类 DDL，并能够恢复因此操作而被删除的用户数据表数据。
- 支持解析双写模式的 WAL 日志，要求 wal_level 设置为 logical 且目标表具有主键。

注意事项

- 不支持在 MySQL 兼容模式下（即数据库实例初始化时指定 DBCOMPATIBILITY='B'）创建 WalMiner 插件。
- 针对数据字典不一致的情况，无法保证恢复完整性。
- 不支持开启线程池模式下运行 WalMiner 插件，即需要在配置文件中配置参数 enable_thread_pool=off。

语法格式

- 添加需解析的 WAL 日志文件。

```
SELECT walminer_wal_add('<WAL 日志文件路径>');
```

- 移除 WAL 日志文件。

```
SELECT walminer_wal_remove('<WAL 日志文件路径>');
```

- 查询日志文件。

```
SELECT walminer_wal_list();
```

- 解析添加的全部 WAL 日志。

```
SELECT walminer_all();
```

或使用

```
SELECT wal2sql();
```

- 在添加的 WAL 日志中查找对应时间范围的 WAL 记录。

```
SELECT walminer_by_time(starttime,endtime,['true']);
```

或使用

```
SELECT wal2sql(starttime,endtime,['true']);
```

- 在添加的 WAL 日志中查找对应 LSN 范围的 WAL 记录。

```
SELECT walminer_by_lsn(startlsn,endlsn,['true']);
```

或使用

```
SELECT wal2sql(startlsn,endlsn,['true']);
```

❖ 在添加的 WAL 日志中查找对应 XID 的 WAL 记录。

```
SELECT walminer_by_xid(xid,['true']);
```

或使用

```
SELECT wal2sql(xid,['true']);
```

【说明】

'true'表示进行精确解析。

WalMiner 的构建基础是，checkpoint 之后对每一个 page 的更改会产生全页写(FPW)，因此一个 checkpoint 之后的所有 WAL 日志可以完美解析。注意 checkpoint 是指 checkpoint 开始的点，而不是 checkpoint 的 wal 记录的点。

普通解析会直接解析给定范围内的 WAL 日志，因为可能没有找到之前的 checkpoint 点，所以会出现有些记录解析不全导致出现空的解析结果。

精确解析是指 WalMiner 程序会界定需要解析的 WAL 范围，并在给定的 WAL 范围之前探索一个 checkpoint 开始点 c1，从 c1 点开始记录 FPI，然后就可以完美解析指定的 WAL 范围。如果在给定的 WAL 段内没有找到 c1 点，那么此次解析会报错停止。

❖ 替身映射

```
SELECT walminer_table_avatar(avatar_table_name,missed_relfilenode);
```

如果一个表被 DROP 或者被 TRUNCATE 等操作，导致新产生的数据字典不包含旧的数据库中所包含的 relfilenode，那么使用新的数据字典无法解析出旧的 WAL 日志中包含的的某些内容。在知晓旧表的表结构的前提下，可以使用替身解析模式。替身模式目前只适用于从 WAL 日志产生的数据库中直接执行解析的场景。

❖ **查看解析结果。**

```
SELECT * FROM walminer_contents;
```

字段名	数据类型	描述
sqlno	Integer	本条 SQL 在其事务中的序号。
xid	bigint	事务 ID。
topxid	bigint	父事务 ID，为 0 表示该事务非子事务。
sqlkind	integer	SQL 类型 1:INSERT 2:UPDATE 3:DELETE
minerd	boolean	解析结果是否完整（缺失 checkpoint 情况下可能无法解析出正确结果。
timestamp	timestamp with time zone	SQL 所在事务的提交时间。
op_text	text	SQL
undo_text	text	UNDO SQL
complete	boolean	SQL 所在事务是否完整解析。
schema	text	目标表所在模式。
relation	text	目标表表名。
start_lsn	text	该记录的开始 LSN。
commit_lsn	text	事务的提交 LSN。

❖ 坏块修复

数据库的数据页挽回（或称为坏块修复）是指在数据库管理系统中修复损坏的数据页或块的过程。数据页是数据库中存储数据的最小单位，通常包

含多个数据库记录。当数据页发生物理损坏或逻辑错误时，可能导致其中存储的数据不可读或不一致。数据页挽回是通过一系列技术和方法来修复损坏的数据页，以尽可能地恢复其中的数据。

```
SELECT page_collect(relfilenode,reloid,pages);
```

【说明】

relfilenode 和 reloid 值可从系统表 pg_catalog.pg_class 中指定表名获取。

❖ 结束 WalMiner 操作

用于释放内存，结束日志分析。

```
SELECT walminer_stop();
```

示例

1、创建插件。

```
CREATE EXTENSION walminer;
```

2、手动归档老的日志，切换到新的日志。

```
SELECT pg_switch_xlog();
```

3、执行检查点，创建测试数据。

```
CHECKPOINT;
CREATE TABLE test_xx(col int);
INSERT INTO test_xx VALUES(998);
SELECT * FROM test_xx;
```

结果返回如下：

```
col
-----
 998
(1 row)
```

4、模拟数据丢失，删除数据。

```
DELETE FROM test_xx;
```

5、添加日志文件到 WalMiner 准备解析。

```
SELECT walminer_wal_add('/home/panweidb/data/oracle/pg_xlog');
```

结果返回如下：

```
 walminer_wal_add
--------------------
 14 file add success
(1 row)
```

6、查询 WalMiner 中的日志文件。

```
SELECT walminer_wal_list();
```

结果返回如下：

```
(/home/panweidb/data/oracle/pg_xlog/000000010000000000000001)
(/home/panweidb/data/oracle/pg_xlog/000000010000000000000002)
(/home/panweidb/data/oracle/pg_xlog/000000010000000000000003)
(/home/panweidb/data/oracle/pg_xlog/000000010000000000000004)
(/home/panweidb/data/oracle/pg_xlog/000000010000000000000005)
(/home/panweidb/data/oracle/pg_xlog/000000010000000000000006)
(/home/panweidb/data/oracle/pg_xlog/000000010000000000000007)
(/home/panweidb/data/oracle/pg_xlog/000000010000000000000008)
(/home/panweidb/data/oracle/pg_xlog/000000010000000000000009)
(/home/panweidb/data/oracle/pg_xlog/00000001000000000000000A)
(/home/panweidb/data/oracle/pg_xlog/00000001000000000000000B)
(/home/panweidb/data/oracle/pg_xlog/00000001000000000000000C)
(/home/panweidb/data/oracle/pg_xlog/00000001000000000000000D)
(/home/panweidb/data/oracle/pg_xlog/00000001000000000000000E)
(14 rows)
```

7、执行全部解析。

```
SELECT walminer_all();
```

结果返回如下：

```
NOTICE:  table "walminer_contents" does not exist, skipping
CONTEXT:  SQL statement "DROP TABLE IF EXISTS walminer_contents"
PL/pgSQL function public.walminer_contents_check() line 3 at SQL statement
referenced column: walminer_contents_check
SQL function "walminer_all" statement 1
referenced column: walminer_all
NOTICE:  Switch wal to 000000010000000000000001 on time 2023-09-05 10:55:
42.137507+08
CONTEXT:  referenced column: wal2sql_internal
```

```
SQL function "walminer_all" statement 2
referenced column: walminer_all
NOTICE:  Switch wal to 000000010000000000000002 on time 2023-09-05 10:55:
42.170179+08
CONTEXT:  referenced column: wal2sql_internal
SQL function "walminer_all" statement 2
referenced column: walminer_all
NOTICE:  Switch wal to 000000010000000000000003 on time 2023-09-05 10:55:
42.19022+08
CONTEXT:  referenced column: wal2sql_internal
SQL function "walminer_all" statement 2
referenced column: walminer_all
NOTICE:  Switch wal to 000000010000000000000004 on time 2023-09-05 10:55:
42.204866+08
CONTEXT:  referenced column: wal2sql_internal
SQL function "walminer_all" statement 2
referenced column: walminer_all
  walminer_all
---------------------
 pg_minerwal success
(1 row)
```

8、查询解析数据。

```
SELECT * FROM walminer_contents;
```

结果返回如下：

```
sqlno | xid  | topxid | sqlkind | minerd |      timestamp       |              op_text
       |              undo_text
    | complete | schema | relation | start_lsn | commit_lsn
-------+-------+--------+---------+--------+----------------------+-------------
------------------------------+--------------------------------------
------+----------+--------+----------+-----------+------------
   1 | 17095 |     0 |      1 | t     | 2000-01-01 08:00:00+08 | INSERT INTO public.test
_xx(col) VALUES(998) | DELETE FROM public.test_xx WHERE col=9
98   | t       | public | test_xx  | 0/4002528 | 0/4002608
   1 | 17096 |     0 |      3 | t     | 2000-01-01 08:00:00+08 | DELETE FROM public.te
st_xx WHERE col=998   | INSERT INTO public.test_xx(col) VALUES
```

```
(998) | t      | public | test_xx  | 0/4002678 | 0/4002748
(2 rows)
```

9.12.python3 环境配置

本文指导用户进行 python3 环境配置，用户可根据实际需要选择如下两种配置方式：

- 在数据库安装用户下配置 python3 环境，此方式仅在该用户的文件夹下单独安装 python 及相关库，不会影响其他用户的使用环境。
- 在 root 用户下配置 python3 环境，此方式可以安装多个不同版本的 python，方便用户按需切换使用。

PanWeiDB 创建 plpython3u 插件可按照上述方法配置 python3 环境，并设置相关环境变量，具体请参考创建 plpython3u 插件。

前置条件

安装 python 需要的依赖（执行如下命令前请确保 yum 源可用）。

```
yum install -y zlib-devel bzip2-devel openssl-devel ncurses-devel sqlite-devel rea
dline-devel tk-devel libffi-devel gcc make
```

在数据库安装用户下配置 python3 环境

【说明】

如下命令均在数据库安装用户下执行，本小节以 panweidb 为例。

步骤 1 切换至数据库安装用户。

```
su - panweidb
```

步骤 2 创建 python 安装路径，并进入。（本文安装路径以 /home/panweidb/python 为例）

```
mkdir -p /home/panweidb/python
cd /home/panweidb/python
```

步骤 3 登录下载地址，下载对应版本的安装包，上传至步骤 2 创建的安装路径中。

步骤 4 解压 python 安装包（安装包名称以实际为准）。

```
tar xf Python-3.****.tar.xz
```

步骤 5 切换进入解压后获得的 python-3.**目录下。

```
cd /home/panweidb/python/Python-3.**
```

步骤 6 运行 configure 脚本，为编译安装做准备。

【须知】

--prefix 用于指定 Python 的安装路径，即步骤 2 中所创建路径。

```
./configure --prefix=/home/panweidb/python --enable-shared --with-ssl-default-suites=openssl --with-python3
```

步骤 7 编译安装 python3。

```
make&make install
```

【须知】

- 若返回结果显示 Successfully，且无 failed 提示则表示编译安装成功。
- 若返回结果中出现 zipimport.ZipImportError: can't decompress data; zlib not available 报错信息，则可执行如下命令解决：

  yum install -y zlib*

步骤 8 拷贝 so 库文件到数据库目录下。

【须知】

- 如下命令以 python3.8.so 所在路径为例，用户应以实际安装的 python 版本对应库文件所在路径为准。例如 Python3.6.8 则需要拷贝 libpython3.6m.so 库文件。
- 拷贝命令也可在 root 用户下执行。
- /home/panweidb/local/panweidb/为数据库安装路径，即$GAUSSHOME 环境变量对应路径。用户可先执行 echo $GAUSSHOME 确定数据库安装目录。

```
cp /home/panweidb/python/lib/libpython3.8.so /home/panweidb/local/panweidb/lib
cp /home/panweidb/python/lib/libpython3.8.so.1.0 /home/panweidb/local/panweidb/lib
```

步骤 9 查看安装好的 python3。

【须知】

如上步骤为在数据库安装用户 panweidb 下安装 python3 环境，因此以 root 用户或其他用户身份执行如下命令将报错。

```
python3 -V
```

在 root 用户下配置 python3 环境

【须知】

如下命令均在 root 下执行。

步骤 1 创建 python 安装路径（路径可以为系统的任意目录，根据实际情况创建，本文安装路径以/home/panweidb/python 为例）。

```
mkdir /home/panweidb/python
```

步骤 2 登录下载地址，下载对应版本的安装包，上传至步骤 1 创建的安装路径中。

步骤 3 解压 python 安装包（安装包名称以实际为准）。

```
cd /home/panweidb/python
tar -zxvf Python-3.**.tar.xz
```

步骤 4 切换至 python 安装路径。

```
cd /home/panweidb/python/Python-3.**
```

步骤 5 运行 configure 脚本，为编译安装做准备。

【须知】

--prefix 是 Python 的安装路径，即步骤 1 中所创建路径。

```
./configure --prefix=/home/panweidb/python --enable-shared --with-ssl-default-suites=openssl --with-python3
```

步骤 6 编译安装 python3。

```
make&make install
```

步骤 7 创建软链接并查看。(如下命令以 python3.8 为例，用户应与实际安装的 python 版本对应文件所在路径为准)

```
ln -s /home/panweidb/python/bin/python3.8 /usr/local/python3
cd /usr/local
ll
```

当返回结果中有 python3 -> /home/panweidb/python/bin/python3.*表示创建软连接成功。

步骤 8 配置环境变量（root 用户下）。

【须知】

- PYTHONHOME：python3 安装路径，即运行 configure 脚本时通过--prefix 指定的路径（步骤 1 中创建的路径）。
- PATH：python3 安装路径的 bin 目录。

1、打开配置文件。

```
vi ~/.bash_profile
```

2、添加如下环境变量。

```
export PYTHON_HOME=/home/panweidb/python
export PATH=$PYTHON_HOME/bin:$PATH
```

3、使环境变量生效。

```
source ~/.bash_profile
```

【说明】

echo $PYTHON_HOME 可用于检查环境变量是否生效。若返回上述所设置的值，则表示设置成功。

步骤 9 拷贝库文件。

【须知】

如下命令以 python3.8.so.1.0 所在路径为例，用户应以实际安装的 python 版本对应库文件所在路径为准。例如 Python3.6.8 则需要拷贝 libpython3.6m.so.1.0 库文件。

```
cp /home/panweidb/python/lib/libpython3.8.so.1.0 /usr/local/lib64/
cp /home/panweidb/python/lib/libpython3.8.so.1.0 /usr/lib/
cp /home/panweidb/python/lib/libpython3.8.so.1.0 /usr/lib64/
```

步骤 10 查看安装好的 python3。

```
python3 -V
```

创建 plpython3u 插件

步骤 1 切换到 PanWeiDB 安装用户，打开配置文件。

```
vi ~/.PanWeiDB
```

步骤 2 添加如下环境变量。

```
export PYTHONHOME=/home/panweidb/python
export LD_LIBRARY_PATH=$LD_LIBRARY_PATH:/home/panweidb/python/lib
```

```
export PATH=$PATH:/home/panweidb/python/bin
```

【说明】

- PYTHONHOME：python3 安装路径，即运行 configure 脚本时通过--prefix 指定的路径（步骤 2 中创建的路径）。
- LD_LIBRARY_PATH：python3 安装路径下的 lib 目录。
- PATH：python3 安装路径的 bin 目录。

【须知】

在.PanWeiDB 文件中，若 LD_LIBRARY_PATH 有多个，请参照上述命令修改靠后的一行。

步骤 3 重启数据库。

```
pw_ctl restart
```

步骤 4 登录 panweidb 数据库，创建 plpython3u 插件。

```
gsql -r
create extension plpython3u;
```

10. 外部数据封装器

FDW 是 Foreign Data Wrappers 的简称,叫做外部数据封装。

外部数据是通过外部数据封装器的帮助来访问的。一个外部数据封装器是一个可以与外部数据源沟通的库，隐藏与外部数据源连接的细节并且从外部数据源获得数据。本章介绍了 PanWeiDB 支持的外部数据封装器。

10.1.Oracle_FDW

功能描述

支持创建外部数据封装器 oracle_fdw，连接 Oracle 数据库，并能在外部表上进行查询、插入、更新和删除操作。

编译 Oracle_FDW

编译 oracle_fdw 需要安装 Oracle 的开发库和头文件。

1、选择合适的运行环境和版本，下载 Basic Package 和 SDK Package 并安装。另外 SQLPlus Package 是 Oracle 的客户端工具，也可以根据需要安装，用于连接 Oracle Server 进行测试。

2、安装好开发包后，就可以开始编译 oracle_fdw 了。编译时需要在执行 configure 时，加入 -enable-oracle-fdw 选项。后续按照正常的 Vstbase 安装方式安装即可。（PanWeiDB 的编译参考章节"字符安装"）

3、编译完成后，编译产物为 oracle_fdw.so，位于安装目录的 lib/postgresql/下。oracle_fdw 相关的 sql 文件和 control 文件，位于安装目录的 share/postgresql/extension/下。

4、如果编译安装时，没有加入 -enable-oracle-fdw 选项，可以在 PanWeiDB 安装完成后，再次编译 oracle_fdw，然后手动将编译产物 oracle_fdw.so 放到对应的安装目录 lib/postgresql/ ，将 oracle_fdw-1.0-1.1.sql、oracle_fdw-1.1.sql、oracle_fdw.control 放到对应的安装目录 share/postgresql/extension/即可。

使用 Oracle_FDW

- 使用 oracle_fdw 需要连接 Oracle，Oracle server 请自行安装。
- 加载 oracle_fdw 扩展：

```
CREATE EXTENSION oracle_fdw
```

- 创建服务器对象：

```
CREATE SERVER
```

- 创建用户映射：

```
CREATE USER MAPPING
```

- 创建外表： 外表的表结构需要与 Oracle 数据库中的表结构保持一致。注意 Oracle server 侧的表的第一个字段必须具有唯一性约束（如 PRIMARY KEY、UNIQUE 等）。

```
CREATE FOREIGN TABLE
```

- 对外表做正常的操作，如 INSERT、UPDATE、DELETE、SELECT、EXPLAIN、ANALYZE、COPY 等。
- 删除外表：

```
DROP FOREIGN TABLE
```

- 删除用户映射：

```
DROP USER MAPPING
```

- 删除服务器对象：

```
DROP SERVER
```

- 删除扩展：

```
DROP EXTENSION oracle_fdw
```

注意事项

- 两个 Oracle 外表间的 SELECT JOIN 不支持下推到 Oracle server 执行，会被分成两条 SQL 语句传递到 Oracle 执行，然后在 PanWeiDB 处汇总处理结果。
- 不支持 IMPORT FOREIGN SCHEMA 语法。
- 不支持对外表进行 CREATE TRIGGER 操作。
- Oracle 数据库支持的 v$session.osuser 最多为 30 个字符，因此如果操作系统用户名超过该限制，则会出现错误。因此需要确保数据库连接的操作系统用户（或执行 vsql 命令的用户）用户名长度不超过 30 个字符。

示例

1、创建扩展。

```
create extension oracle_fdw;
```

2、创建 FDW Server 用于识别外部数据源。

```
create server ora_fdw_server foreign data wrapper oracle_fdw
options(dbserver '172.16.103.104:1521/orcl');
```

3、新建用户并授权。

```
create user use_ora password 'Bigdata@123';
grant usage on foreign data wrapper ora_fdw_server to use_ora;
```

4、创建 USER MAPPING 用于映射 Oracle 数据库用户。

```
create user mapping for use_ora server ora_fdw_server options(user 'system',pass
word 'root');
```

5、在 oracle 创建表。

```
create table system.emp_fdw(empno int,ename varchar(30));
insert into emp_fdw values(1,'foo');
insert into emp_fdw values(2,'bar');
```

6、在 panweidb 创建外部表，切换到用户 use_ora。

```
create foreign table emp_fdw_ora(empno int,ename varchar(30))
server ora_fdw_server
options(schema 'SYSTEM',table 'EMP_FDW');
\c - use_ora
```

7、插入外部表数据。

```
insert into emp_fdw_ora values(3,'bar3');
```

8、查询外部表数据。

```
select * from emp_fdw_ora;
```

10.2.dblink

功能描述

dblink 是一个可以在一个 PanWeiDB 数据库会话中连接到其他 PanWeiDB 数据库的工具。

本章主要介绍 dblink 插件的功能。

关于 PanWeiDB 数据库兼容 Oracle 语法 CREATE DATABASE LINK 的详细信息，请参见《PanWeiDB V2.0-S3.0.1_B01 Oracle 兼容性手册》的 DATABASE LINK 章节。

注意事项

因 PanWeiDB 与 PostgreSQL 依赖冲突，因此无法连接至 PostgreSQL 数据库。

语法格式

- 加载 dblink 扩展。

```
CREATE EXTENSION dblink;
```

- 打开一个到远程数据库的持久连接。

  如果不指定 connname，则会开启一个未命名连接。可以同时打开多个不同名称的连接，仅支持同时打开一个未命名的连接。

  连接会持续到手动关闭，或者数据库会话结束。

```
SELECT dblink_connect(constr);
SELECT dblink_connect(connname,connstr);
```

- 关闭一个到远程数据库的持久连接。

  如果不指定 connname，则表示关闭未命名连接。

```
SELECT dblink_disconnect();
SELECT dblink_disconnect(connname);
```

- 在远程数据库执行查询。

  可以通过指定 conname 或 connstr 来指定连接执行查询，如果不指定，则表示使用未命名连接。

```
SELECT * FROM dblink(connname, sql_statement [, fail_on_error]);
SELECT * FROM dblink(connstr, sql_statement [, fail_on_error]);
```

```
SELECT * FROM dblink(sql_statement [, fail_on_error])
```

dblink 会返回 record，而不是指定任何特定的列集合，也就是说，FROM 子句的别名部分必须指定函数将返回的列名和类型。例如：

```
SELECT *
FROM dblink('dbname=postgres', 'select proname, prosrc from pg_proc')
 AS t1(proname name, prosrc text)
WHERE proname LIKE 'bytea%';
```

在运行时，如果来自远程数据库的实际查询结果的列数与 from 子句中显示的列数不同，则会引发错误。因此，建议将 dblink 与视图配合使用，避免每次调用。因此，可以将上述业务修改为以下形式：

```
CREATE VIEW view_16384 AS
 SELECT *
FROM dblink('dbname=postgres', 'select proname, prosrc from pg_proc')
AS t1(proname name, prosrc text);
SELECT * FROM view_16384 WHERE proname LIKE 'bytea%';
```

- 在远程数据库执行命令。

```
SELECT dblink_exec(connname, sql_statement [, fail_on_error]);
SELECT dblink_exec(connstr, sql_statement) [, fail_on_error];
SELECT dblink_exec(sql_statement [, fail_on_error]);
```

- 在远程数据库打开游标。

```
SELECT dblink_open(cursorname, sql_statement [, fail_on_error]);
SELECT dblink_open(connname, cursorname, sql_statement [, fail_on_error]);
```

- 在远程数据库从打开的游标返回行。

```
SELECT dblink_fetch(cursorname, howmany [, fail_on_error])
SELECT dblink_fetch(connname, cursorname, howmany [, fail_on_error])
```

- 在远程数据库关闭游标。

```
SELECT dblink_close(cursorname [, fail_on_error]);
SELECT dblink_close(connname, cursorname [, fail_on_error]);
```

- 返回所有打开的命名 dblink 连接的名称。

```
SELECT dblink_get_connections();
```

- 查询指定连接最近的错误消息。

```
SELECT dblink_error_message(connname);
```

- 向远程数据库发送一个异步查询。

```
SELECT dblink_send_query(connname, sql_statement);
```

- 检查连接是否正在忙于一个异步查询。

```
SELECT dblink_is_busy(connname);
```

- 检索连接上的异步通知。

```
SELECT dblink_get_notify();
SELECT dblink_get_notify(connname);
```

- 获取异步查询结果。

```
SELECT dblink_get_result(connname[, fail_on_error]);
```

- 取消连接上运行的所有查询。

```
SELECT dblink_cancel_query(connname);
```

- 查询远程数据库表的主键。

```
SELECT dblink_get_pkey(relname)
```

- 删除 dblink 扩展。

```
DROP EXTENSION dblink;
```

参数说明

- **connname**

  连接名称，可自定义。

- **connstr**

  连接信息，例如：hostaddr=172.16.103.92 port=6036 dbname=panweidb user=panweidb password='Bigdata@123'

- **fail_on_error**

  布尔值，设置为 True 表示远端如果发生错误，则也会返回到本地；如果设置为 False，远端的错误则会返回提示，函数返回值为 ERROR。

- **cursorname**

  打开的游标名称。

- **sql_statement**

  执行的语句。

- **relname**

  远端表名。

注意事项

- 不同的用户可以创建同名的 private dblink，且用户只能访问自己创建的 DBLINK。
- 同一用户可以创建同名的 private dblink 和 public dblink，在访问时优先访问 private dblink。
- 为支持不同用户创建同名 dblink，需禁用 GRANT FOREIGN SERVER，不再允许将 server 权限给予其他用户，以防一个用户能访问多个同名 server，造成混乱。
- 不支持访问远程 Oracle 包变量和常量。（注：数据也不支持）
- 不支持将远程函数和存储过程创建为同义词方式使用。
- 创建 private dblink 或 public dblink 时，必须提前创建 jdbc_fdw，并具备 DBLINK 相关权限。

示例

示例 1：常用函数操作

1、创建扩展。

```
create extension dblink;
```

2、先执行 dblink_connect 保持连接。

```
SELECT dblink_connect('mycoon','hostaddr=172.16.103.92 port=6036 dbname=panweidb user=lst password=Bigdata@123');
```

3、执行 BEGIN 命令。

```
SELECT dblink_exec('mycoon', 'BEGIN');
```

4、执行数据操作。

```
SELECT dblink_exec('mycoon', 'create table people(id int,info varchar(10))');
SELECT dblink_exec('mycoon', 'insert into people values(1,''foo'')');
SELECT dblink_exec('mycoon', 'insert into people values(2,''foo'')');
SELECT dblink_exec('mycoon', 'update people set info=''bar'' where id=1');
```

5、执行事务提交。

```
SELECT dblink_exec('mycoon', 'COMMIT');
```

6、执行查询。

```
select * from dblink('mycoon','select * from people') as testTable (id int,info varcha
r(10));
```

7、解除连接。

```
SELECT dblink_disconnect('mycoon');
```

10.3.JDBC_FDW

功能描述

支持基于 JDBC 驱动的 FDW，可以通过不同数据源的 JDBC 驱动在外部表访问数据。

注意事项

- 支持 DATABASE LINK 访问 Oracle 中数据库的 Sequence 序列的值。
  - 支持通过 NEXTVAL 获取下一个序列值。
  - 支持通过 CURRVAL 获取当前序列值。
- 支持 DATABASE LINK 与 synonym 结合访问 Oracle 的存储过程。
  - 支持远程调用无参的存储过程。
  - 支持调用带输入参数的存储过程。
  - 支持调用带 out 参数的存储过程。
  - 传入参数和 out 参数，支持的数据类型如下：

数据类型名称	数据类型
数字类型	NUMBER
变长字符串类型	VARCHAR2，NVARCHAR2 VARCHAR，NVARCHAR
定长字符串类型	CHAR，NCHAR
单精度精度二进制浮点类型	BINARY_FLOAT
双精度二进制浮点类型	BINARY_DOUBLE
二进制数据类型	RAW
二进制大对象类型	BLOB
字符大对象类型	CLOB

数据类型名称	数据类型
XML 数据类型	XMLType
日期时间类型	DATE
时间戳数据类型	TIMESTAMP， TIMESTAMP WITH TIME ZONE
本地时间戳数据类型	TIMESTAMP WITH LOCAL TIME ZONE

- 支持 DATABASE LINK 与 synonym 结合访问 Oracle 的自定义函数。
  - 支持远程调用无参的自定义函数。
  - 支持远程调用带输入参数的自定义函数。
  - 支持远程调用 out 参数的自定义函数。
  - 支持调用的函数 return 是%rowtype 行数据类型。
  - 传入参数和 out 参数和 return 函数返回，支持的数据类型如下：

数据类型名称	数据类型
数字类型	NUMBER
变长字符串类型	VARCHAR2，NVARCHAR2 VARCHAR，NVARCHAR
定长字符串类型	CHAR，NCHAR
单精度精度二进制浮点类型	BINARY_FLOAT
双精度二进制浮点类型	BINARY_DOUBLE
二进制数据类型	RAW
二进制大对象类型	BLOB
字符大对象类型	CLOB
XML 数据类型	XMLType
日期时间类型	DATE
时间戳数据类型	TIMESTAMP， TIMESTAMP WITH TIME ZONE
本地时间戳数据类型	TIMESTAMP WITH LOCAL TIME ZONE

- 支持 DATABASE LINK 与 synonym 结合访问 Oracle 的包。
- 支持在远端执行 JOIN、AGG、SORT 操作。
- 支持通过 analyze tablename@dblinkname 语法可获得更多建议，优化建议仅供参考。

- Oracle 数据库支持的 v$session.osuser 最多为 30 个字符，因此如果操作系统用户名超过该限制，则会出现错误。因此需要确保数据库连接的操作系统用户（或执行 vsql 命令的用户）用户名长度不超过 30 个字符。

示例

前置条件

1、安装 JDK1.8，并配置好 JAVA_HOME 环境变量。

2、数据库环境变量 LD_LIBRARY_PATH 动态库路径加入 JAVA_HOME 配置。

通过如下命令配置环境变量：

```
x86：export LD_LIBRARY_PATH=$GAUSSHOME/jre/lib/amd64/server/:$LD_LIBRARY_PATH
ARM：export LD_LIBRARY_PATH=$GAUSSHOME/jre/lib/aarch64/server/:$LD_LIBRARY_PATH
```

3、修改数据库配置参数：在 shared_preload_libraries 参数配置中加上 jdbc_fdw，然后重启数据库。

4、驱动包 ojdbc7.jar 放置到$GAUSSHOME/lib/postgresql/下并执行如下命令授权

```
chmod 777 $GAUSSHOME/lib/postgresql/ojdbc7.jar
```

示例 1：创建 jdbc_fdw 扩展。

1、加载 jdbc_fdw 扩展。

```
create extension jdbc_fdw;
```

2、在创建测试表。

```
create table emp_fdw(empno int,ename varchar(30));
insert into emp_fdw values(1,'foo');
insert into emp_fdw values(2,'bar');
```

3、创建连接。

```
create server ora_jdbc foreign data wrapper jdbc_fdw options(
drivername 'oracle.jdbc.driver.OracleDriver',
url 'jdbc:oracle:thin:@//172.16.103.104:1521/orcl',
querytimeout '100',
jarfile '/home/panweidb/bin/ojdbc7.jar',
maxheapsize '200'
);
```

4、修改 pwadmin 用户密码。

```
alter role pwadmin identified by 'Panwei@admin';
```

5、创建到 oracle 的映射。

```
create user mapping for use_ora_jdbc server ora_jdbc options(username 'system',
password 'root');
```

6、创建需要访问的 oracle 中对应表的结构。

```
create foreign table emp_fdw_ora(empno int,ename varchar(30))
server ora_jdbc options(table_name 'EMP_FDW');
```

7、将权限赋予 emp_fdw_ora。

```
grant all on emp_fdw_ora to use_ora_jdbc;
```

8、查看外部表。

```
\c - use_ora_jdbc
select * from emp_fdw_ora;
```

当结果显示如下信息，则表示外部表访问成功。

```
id | info
---+-------
1 | bar
2 | foo
(2 row)
```

示例 2：远程访问 Oracle 函数。

1、修改 postgresql.conf 参数，将 jdbc_fdw 配置到 shared_preload_libraries 参数中。

2、驱动包 ojdbc7.jar 放置到$GAUSSHOME/lib/postgresql/下并修改权限。

```
chmod 777 $GAUSSHOME/lib/postgresql/ojdbc7.jar
```

3、在 oracle 中创建对应的表和存储过程。

```
create or replace function ora_func01(param1 in binary_float,param2 in out binary
_float ,param3 out binary_float)
return binary_float
as
v_num1 binary_float;
begin
v_num1 :=param1*param2;
param2:=2;
param3:=3;
return v_num1;
end;
/
```

4、在 PanWeiDB 安装插件。

```
create extension jdbc_fdw;
```

5、在 panweidb 中创建 dblink。

```
CREATE public DATABASE LINK dblink_104 CONNECT TO SYSTEM IDENTIFIED BY 'root'
 USING jdbc_fdw(
url 'jdbc:oracle:thin:@//172.16.103.104:1521/orcl',
jarfile '/home/lmh/local/panweidb/lib/postgresql/ojdbc7.jar'
);
```

6、在 panweidb 中创建同义词进行访问 Oracle 的带参数的函数。

```
create public SYNONYM syn_f01 for ora_func01@dblink_104;
```

7、在 PanWeiDB 直接调用函数。

```
call ora_func01@dblink_104(11,2,3);
```

8、在 PanWeiDB 通过同义词调用函数。

```
call sync_f01(11,2,3);
```

10.4.MySQL_FDW

功能描述

支持创建外部数据封装器 mysql_fdw，连接 MariaDB 或 MySQL 数据库，并能在外部表上进行查询、插入、更新和删除操作。

mysql_fdw 插件默认参与编译，用户只需要在服务器上安装 MariaDB 开发包和 PanWeiDB 后，即可直接使用 mysql_fdw，无须其他操作。

使用 MySQL_FDW

- 下载 MariaDB-common、MariaDB-compat、MariaDB-shared、MariaDB-devel 文件并进行安装。推荐从 MariaDB 的官方源进行下载。
- 使用 mysql_fdw 需要连接 MariaDB 或者 MySQL Server，MariaDB 或 MySQL Server 请自行安装。
- 加载 mysql_fdw 扩展。

```
CREATE EXTENSION mysql_fdw;
```

- 创建服务器对象。

```
CREATE SERVER
```

- 创建用户映射。

```
CREATE USER MAPPING
```

- 创建外表：外表的表结构需要与 MySQL/MariaDB 侧的表结构保持一致。注意 MySQL/MariaDB 侧的表的第一个字段必须具有唯一性约束（如 PRIMARY KEY、UNIQUE 等）。

```
CREATE FOREIGN TABLE
```

- 对外表做正常的操作，如 INSERT、UPDATE、DELETE、SELECT、EXPLAIN、ANALYZE、COPY 等。
- 删除外表。

```
DROP FOREIGN TABLE
```

- 删除用户映射。

```
DROP USER MAPPING
```

- 删除服务器对象：

```
DROP SERVER
```

❖ 删除扩展。

```
DROP EXTENSION mysql_fdw
```

注意事项

❖ 两个 mysql 外表间的 SELECT JOIN 不支持下推到 MariaDB/MySQL Server 执行，会被分成两条 SQL 语句传递到 MariaDB/MySQL Server 执行，然后在 PanWeiDB 处汇总处理结果。

❖ 不支持 IMPORT FOREIGN SCHEMA 语法。

❖ 不支持对外表进行 CREATE TRIGGER 操作。

示例

1、创建扩展。

```
create extension mysql_fdw;
```

2、创建 FDW Server 用于识别外部数据源。

```
create server mysql_fdw_server foreign data wrapper mysql_fdw options(HOST '17
2.16.103.108',PORT '20001');
```

3、新建用户并授权。

```
create user use_mysql password 'Bigdata@123';
grant usage on foreign server mysql_fdw_server to use_mysql;
```

4、创建 USER MAPPING 用于映射 MySQL 数据库用户。

```
create user mapping for use_mysql server mysql_fdw_server options(username 'ro
ot',password 'root');
```

5、在 MySQL 创建表。

```
create table test.emp_fdw(empno int,ename varchar(30));
insert into emp_fdw values(1,'foo');
insert into emp_fdw values(2,'bar');
```

6、在 panweidb 创建外部表，切换到用户 use_mysql。

```
create foreign table emp_fdw_mysql(empno int,ename varchar(30)) server mysql_f
dw_server options(DBNAME 'test',table_name 'emp_fdw');
\c - use_mysql
```

7、查询外部表数据。

```
select * from emp_fdw_mysql;
```

当结果显示如下信息，则表示外部表访问成功。

```
empno | ename
-------+-------
     1 | foo
     2 | bar
(2 rows)
```

10.5.POSTGRE_FDW

功能描述

postgres_fdw 功能提供了外部数据封装器 postgres_fdw 插件，它可以被用来访问存储在外部 PanWeiDB 或 PostgreSQL 服务器中的数据。

用户可以通过指定参数 remote_kind 的值来控制远端连接的数据库为 PanWeiDB 或是 PostgreSQL 11。

注意事项

- 两个 postgres_fdw 外表间的 SELECT JOIN 不支持下推到远端 PanWeiDB 执行，会被分成两条 SQL 语句传递到远端 PanWeiDB 执行，然后在本地汇总处理结果。
- 支持使用 PostgreSQL 兼容性手册的 IMPORT FOREIGN SCHEMA 命令从外部服务器导入表定义。
- 不支持对外表进行 CREATE TRIGGER 操作。
- 若需要 PanWeiDB 访问远端 PostgreSQL 11 的分区表，需要设置 GUC 参数 sql_beta_feature 的值为 partition_fdw_on。关于此参数的详细说明及配置方法可以参考《PanWeiDB V2.0-S3.0.1_B01 开发者指南》->sql_beta_feature。

语法格式

1、加载 postgres_fdw 扩展。

```
CREATE EXTENSION [ IF NOT EXISTS ] extension_name
 [ WITH ] [ SCHEMA schema_name ]
     [ VERSION version ]
     [ FROM old_version ]
```

2、创建一个新的外部服务器。

```
CREATE SERVER server_name
FOREIGN DATA WRAPPER fdw_name
   OPTIONS ( { option_name ' value ' } [, ...] ) ;
```

3、创建一个用户到一个外部服务器的新映射。

```
CREATE USER MAPPING FOR { user_name | USER | CURRENT_USER | PUBLIC }
 SERVER server_name
[ OPTIONS ( option 'value' [ , ... ] ) ]
```

4、创建外表。

```
CREATE FOREIGN TABLE [ IF NOT EXISTS ] table_name ( [
column_name type_name [ OPTIONS ( option 'value' [, ... ] ) ] [ COLLATE
collation ] [ column_constraint [ ... ] ]
 [, ... ]
] )
 SERVER server_name
[ OPTIONS ( option 'value' [, ... ] ) ]
```

【说明】

当在 OPTIONS 中出现 password 选项时，需要保证 astbase 每节点的 SGAUSSHOME/bin 目录下存在 usermapping.key.cipher 和 usermapping.keyrand 文件，如果不存在这两个文件，需要通过如下命令创建这两个文件，并将这两个文件放入各节点目录 GAUSSHOME/bin，且确保具有读权限。

```
pw_guc generate -o usermapping -S default -D $GAUSSHOME/bin
```

这里 column_constraint 可以是：

```
[ CONSTRAINT constraint_name ]
{ NOT NULL |
NULL |
DEFAULT default_expr }
```

参数说明

- **server_name**

  server 的名称。

  取值范围：长度必须小于等于 63。

- **fdw_name**

  指定外部数据封装器的名称。

  取值范围：oracle_fdw，mysql_fdw，postgres_fdw，mot_fdw。

- **CREATE SERVER 中 OPTIONS 参数包括：**
  - host（必选）：要连接的主机名，本功能填写远端服务器的 IP 地址。
  - port（必选）：主机服务器的端口号，本功能填写远端服务器上集群的端口号。
  - dbname（必选）：数据库名，本功能填写远端服务器上的数据库名。
  - remote_kind（非必选）：远程连接类型，可选取值为 panweidb（默认）或 postgres。指定为 postgres 时可以连接到远端 PostgreSQL

11 数据库。如果不指定此参数，默认按照远程连接到 PanWeiDB 数据库的逻辑进行处理。

- hostaddr(非必选)：与之链接的主机的 IP 地址，是标准的 IPv4 地址格式，比如，172.28.40.9。如果机器支持 IPv6，那么也可以使用 IPv6 的地址。如果声明了一个非空的字符串，那么使用 TCP/IP 通讯机制。
- connect_timeout(非必选)：链接的最大等待时间，以秒计（用十进制整数字符串书写），0 或者不声明表示无穷。不建议把链接超时的值设置得小于 2 秒。
- options(非必选)：添加命令行选项以在运行时发送到服务器。
- keepalives(非必选)：控制客户端侧的 TCP 保持激活是否使用。缺省值是 1，意思为打开，但是如果不想要保持激活，你可以更改为 0，意思为关闭。通过 Unix 域套接字做的链接忽略这个参数。
- keepalives_idle(非必选)：在 TCP 应该发送一个保持激活的信息给服务器之后，控制不活动的秒数。0 值表示使用系统缺省。通过 Unix 域套接字做的链接或者如果禁用了保持激活则忽略这个参数。
- keepalives_interval(非必选)：在 TCP 保持激活信息没有被应该传播的服务器承认之后，控制秒数。0 值表示使用系统缺省。通过 Unix 域套接字做的链接或者如果禁用了保持激活则忽略这个参数。
- keepalives_count(非必选)：添加命令行选项以在运行时发送到服务器。例如，设置为-c comm_debug_mode=off 设置 guc 参数 comm_debug_mode 参数的会话的值为 off。
- use_remote_estimate(非必选)：控制 postgres_fdw 是否发出 EXPLAIN 命令以获取运行消耗估算。默认值为 false。
- fdw_startup_cost(非必选)：执行一个外表扫描时的启动耗时估算。这个值通常包含建立连接、远端对请求的分析和生成计划的耗时。默认值为 100。
- fdw_typle_cost(非必选)：在远端服务器上对每一个元组进行扫描时的额外消耗。这个值通常表示数据在 server 间传输的额外消耗。默认值为 0.01。

- **user_name**

  要映射到外部服务器的一个现有用户的名称。 CURRENT_USER 和 USER 匹配当前用户的名称。 当 PUBLIC 被指定时，一个所谓的公共映射会被创建，当没有特定用户的映射可用时将会使用它。

- **server_name**

  将为其创建用户映射的现有服务器的名称。

- **CREATE USER MAPPING 中 OPTIONS 参数包括：**
  - user(必选)：远端服务器上 PanWeiDB 的用户名，属于普通用户。
  - password（必选）：远端服务器上 PanWeiDB 用户对应的密码

- **table_name**

  外表的表名。取值范围：字符串，要符合标识符的命名规范。

- **column_name**

  外表中的字段名。取值范围：字符串，要符合标识符的命名规范。

- **type_name**

  字段的数据类型。

- **SERVER server_name**

  外表的 server 名称。

- **OPTIONS 参数包括**
  - schema_name（必选）：远端 server 的 schema 名称。如果不指定的话，将使用外表自身的 schema 名称作为远端的 schema 名称。
  - table_name（必选）：远端 server 的表名。如果不指定的话，将使用外表自身的表名作为远端的表名。
  - column_name（非必选）：远端 server 的表的列名。如果不指定的话，将使用外表自身的列名作为远端的的表的列名。

示例

示例 1：使用 postgres_fdw 连接到远程 panweidb 数据库，并访问外表。

前置条件：

在本地服务器和远端服务器创建数据库用户之前，建议关闭远端数据库和本地数据库的强制修改密码功能。（参数默认启用，修改后重启生效。）

```
alter system set password_force_alter=off;
```

1、在本地服务器创建用户。

```
create user hzy with sysadmin password '123456Aa';
grant all on database panweidb to hzy;
```

2、远端服务器修改配置文件 pg_hba.conf 的配置参数，新增下列参数。

```
host  all   all     0.0.0.0/0       sha256
```

3、远端服务器修改配置文件 postgresql.conf 中的 listen_addresses 参数，值为本地服务器的 IP 地址。

4、远端服务器中创建用户。

```
create user hzy with sysadmin password '123456Aa';
grant all on database panweidb to hzy;
```

5、远端服务器使用 hzy 用户创建测试表。

```
create table student( student_no int4 primary key, student_name varchar(30), age int2 );
```

```
insert into student values(1,'xiaoming',12);
```

6、在本地服务器中，使用新建的用户登录系统。

```
\c - hzy
```

【须知】
以下操作均在本地服务器上执行。

7、加载 postgres_fdw 扩展。

```
create extension postgres_fdw;
```

8、创建外部服务器。

```
create server server_to_200 foreign data wrapper postgres_fdw options (host '172.16.105.54',port '5432',dbname 'panweidb');
```

9、创建一个用户到外部服务器的映射。

```
create user MAPPING FOR hzy SERVER server_to_200 OPTIONS (user 'hzy',password '123456Aa');
```

10、创建外表。

```
create foreign table local_foreign_table_student( student_no int4 , student_name varchar(30), age int2) server server_to_200 options(schema_name 'hzy', table_name 'student');
```

11、访问外表。

```
select * from local_foreign_table_student;
```

访问外表成功，返回结果为：

```
 student_no | student_name | age
------+--------+---
      1 | xiaoming    | 12
(1 row)
```

12、对外表执行 DML 操作并再次查看外表。

```
insert into local_foreign_table_student values (2,'xiaohong',11);
update local_foreign_table_student set student_name = 'xiaoqing' where student_no = 1;
delete from local_foreign_table_student where age = 11;
select * from local_foreign_table_student ;
```

操作成功，返回结果为：

```
 student_no | student_name | age
------+--------+---
      1 | xiaoqing    | 12
(1 row)
```

示例 2：使用 postgres_fdw 连接到远程 PostgreSQL 11 数据库，并执行查询表的操作。

前置条件：以下步骤在远端服务器执行。

1、具备远端 PostgreSQL 11 的数据库环境。

2、在远端数据库中创建数据库用户 u1。

```
create user u1 password 'Test1234';
```

3、创建并切换至测试数据库 dtest 下。

```
create database dtest;
\c dtest;
```

4、以用户 u1 的身份在远端测试数据库中创建测试表 t1，并插入数据。

```
\c - u1
create table t1(id int,name text);
insert into t1 values(101,'zhangsan');
insert into t1 values(102,'lisi');
```

示例步骤：以下步骤在本地服务器中执行。

1、创建扩展 postgres_fdw。

```
create extension postgres_fdw;
```

2、创建 DBLINK，指定参数 remote_kind 为'postgres'。

```
create database link link1 connect to u1 identified by 'Test1234' using
postgres_fdw(dbname 'dtest', host '127.0.0.1', port '5432', remote_kind 'postgres');
```

【说明】

DBLINK 加密时需要使用 usermapping.key.cipher 和 usermapping.key,.and 文件作为加密密码文件和加密因子。首次使用前需要通过如下命令创建这两个文件，并将这两个文件放入各节点目录 SGAUSSHOME/bin，且确保具有读权限。

```
pw_guc generate -o usermapping -S default -D $GAUSSHOME/bin
```

3、查询远端 PG 数据库中的 t1 表。

```
select * from public.t1@link1;
```

返回结果如下：查询到 t1 中的数据，且数据正确。

```
id | name
---+------
101 | zhangsan
102 | lisi
(2 rows)
```

示例 3：使用 postgres_fdw 连接到远程 PostgreSQL 11 数据库，并查询分区表的数据。

前置条件：以下步骤在远端服务器执行。

【须知】

远程连接前请确认远端 PostgreSQL 数据库已经在配置文件 pg_hba.conf 中允许其他用户访问此数据库。

1、具备远端 PostgreSQL 11 的数据库环境。

2、在远端数据库中创建数据库用户 u2。

```
create user u2 password 'Test1234';
```

3、创建并切换至测试数据库 dtest 下。

```
create database dtest;
\c dtest;
```

4、以用户 u2 的身份在远端测试数据库中创建分区表。

```
\c - u2
create table test_partition(col1 int ,col2 text) partition by list(col1);
create table test_subpartition1 partition of test_partition for values in (1);
create table test_subpartition2 partition of test_partition for values  in (2);
create table test_subpartition3 partition of test_partition for values  in (3);
create table test_subpartition4 partition of test_partition for values  in (4);
create table test_subpartition5 partition of test_partition for values  in (5);
```

5、向分区表 test_partition 中插入数据。

```
insert into test_partition values (1,'test1');
insert into test_partition values (2,'test2');
insert into test_partition values (3,'test3');
insert into test_partition values (4,'test4');
insert into test_partition values (5,'test5');
```

示例步骤：以下步骤在本地服务器中执行。

1、创建扩展 postgres_fdw。

```
create extension postgres_fdw;
```

2、创建 DBLINK，指定参数 remote_kind 为'postgres'。

```
create database link link2 connect to u2 identified by 'Test1234' using
postgres_fdw(dbname 'dtest', host '127.0.0.1', port '5432', remote_kind 'postgres');
```

【说明】

DBLINK 加密时需要使用 usermapping.key.cipher 和 usermapping.key,.and 文件作为加密密码文件和加密因子。首次使用前需要通过如下命令创建这

两个文件，并将这两个文件放入各节点目录 SGAUSSHOME/bin，且确保具有读权限。

```
pw_guc generate -o usermapping -S default -D $GAUSSHOME/bin
```

3、开启分区表查询开关。

```
set sql_beta_feature=PARTITION_FDW_ON;
```

4、查询远端 PG 数据库中的分区表。

```
select * from public.test_partition@link2;
```

返回结果如下：查询到 test_partition 表中的数据，且数据正确。

```
col1 | col2
---+-----
  1 | test1
  2 | test2
  3 | test3
  4 | test4
  5 | test5
(5 rows)
```

11. AI特性

人工智能技术最早可以追溯到上世纪 50 年代，甚至比数据库系统的发展历史还要悠久。但是，由于各种各样客观因素的制约，在很长的一段时间内，人工智能技术并没有得到大规模的应用，甚至还经历了几次明显的低谷期。到了近些年，随着信息技术的进一步发展，从前限制人工智能发展的因素已经逐渐减弱，所谓的 ABC（AI、Big data、Cloud computing）技术也随之而诞生。

PanWeiDB 的 AI 特性子模块大致可分为 DB4AI 以及 AI in DB 两个部分。

- DB4AI 就是指打通数据库到人工智能应用的端到端流程，通过数据库来驱动 AI 任务，统一人工智能技术栈，达到开箱即用、高性能、节约成本等目的。例如通过 SQL-like 语句实现推荐系统、图像检索、时序预测等功能，充分发挥数据库的高并行、列存储等优势，既可以避免数据和碎片化存储的代价，又可以避免因信息泄漏造成的安全风险。
- AI in DB 就是对数据库内核进行修改，实现原有数据库架构模式下无法实现的功能，如利用 AI 算法改进数据库的优化器，实现更精确的代价估计等。

本章节所涉及的功能独立存在于数据库安装目录($GAUSSHOME)的 bin/dbmind 目录中，各个子功能存在于 dbmind 的子目录 components 中。提供 gs_dbmind 命令行供用户调用。与此同时，对于数据库内置 AI 的功能（如 DB4AI），以 SQL 语法和系统函数的形式呈现。

本章介绍的内容包括：

- DB4AI：数据库驱动 AI
- AI in DB：数据库内 AI 功能

11.1.DB4AI: 数据库驱动 AI

DB4AI 是指利用数据库的能力驱动 AI 任务，实现数据存储、技术栈的同构。通过在数据库内集成 AI 算法，令 PanWeiDB 具备数据库原生 AI 计算引擎、模型管理、AI 算子、AI 原生执行计划的能力，为用户提供普惠 AI 技术。不

同于传统的 AI 建模流程，DB4AI “一站式” 建模可以解决数据在各平台的反复流转问题，同时简化开发流程，并可通过数据库规划出最优执行路径，让开发者更专注于具体业务和模型的调优上，具备同类产品不具备的易用性与性能优势。

11.1.1.原生 DB4AI 引擎

PanWeiDB 当前版本支持了原生 DB4AI 能力，通过引入原生 AI 算子，简化操作流程，充分利用数据库优化器、执行器的优化与执行能力，获得高性能的数据库内模型训练能力。更简化的模型训练与预测流程、更高的性能表现，让开发者在更短时间内能更专注于模型的调优与数据分析上，而避免了碎片化的技术栈与冗余的代码实现。

关键字解析

表 1 DB4AI 语法及关键字

名称		描述
语法	CREATE MODEL	创建模型并进行训练，同时保存模型。
	PREDICT BY	利用已有模型进行推断。
	DROP MODEL	删除模型。
关键字	TARGET	训练/推断任务的目标列名。
	FEATURES	训练/推断任务的数据特征列名。
	MODEL	训练任务的模型名称。

使用指导

1、 本版本支持的算法概述。

当前版本的 DB4AI 新增支持算法如下：

表 2 支持算法

优化算法	算法
GD	logistic_regression
	linear_regression
	svm_classification
	PCA
	multiclass
Kmeans	kmeans
xgboost	xgboost_regression_logistic
	xgboost_binary_logistic
	xgboost_regression_squarederror
	xgboost_regression_gamma

2、 模型训练语法说明。

❖ CREATE MODEL

使用“CREATE MODEL”语句可以进行模型的创建和训练。模型训练 SQL 语句，选用公开数据集鸢尾花数据集 iris。

❖ 以 multiclass 为例，训练一个模型。从 tb.*iris 训练集中指定 sepal.*length, sepal.*width,petal.*length,petal.*widt 为特征列，使用 multiclass 算法，创建并保存模型 iris.*classification._model。

```
panweidb=# CREATE MODEL iris_classification_model USING xgboost_regression_lo
gistic FEATURES sepal_length, sepal_width,petal_length,petal_width TARGET target
_type < 2 FROM tb_iris_1 WITH nthread=4, max_depth=8;
MODEL CREATED. PROCESSED 1
```

上述命令中：

❖ “CREATE MODEL”语句用于模型的训练和保存。

❖ USING 关键字指定算法名称。

- FEATURES 用于指定训练模模型的特征，需根据训练数据表的列名添加。
- TARGET 指定模型的训练目标，它可以是训练所需数据表的列名，也可以是一个表达式，例如：price .> 10000。
- WITH 用于指定训练模型时的超参数。当超参未被用户进行设置的时候，框架会使用默认数值。

针对不同的算子，框架支持不同的超参组合：

表 3 算子支持的超参

算子	超参
GD (logistic_regression、linear_regression、svm_classification)	optimizer(char); verbose(bool); max_iterations(int); max_seconds(double); batch_size(int); learning_rate(double); decay(double); tolerance(double) 其中，SVM 限定超参 lambda(double)
Kmeans	max_iterations(int); num_centroids(int); tolerance(double); batch_size(int); num_features(int); distance_function(char); seeding_function(char); verbose(int);seed(int)
GD(pca)	batch_size(int);max_iterations(int);max_seconds(int);tolerance(float8);verbose(bool);number_components(int);seed(int)
GD(multiclass)	classifier(char) 注意：multiclass 的其他超参种类取决于选择的分类器中类
xgboost_regression_logistic、xgboost_binary_logistic、xgboost_regression_squarederror、	batch_size(int);booster(char);tree_method(char);eval_metric(char*);seed(int);nthread(int);max_depth(int);gamma(float8);eta(float8);min_child_weight(int);verbosity(int)

算子	超参
xgboost_regression_gamma	

当前各个超参数设置的默认值和取值范围如下：

表 4 超参的默认值以及取值范围

算子	超参(默认值)	取值范围	超参描述
GD：logistic_regression、linear_regression、svm_classification、	optimizer = gd（梯度下降法）	gd/ngd（自然梯度下降）	优化器
	verbose = false	T/F	日志显示
	max_iterations = 100	(0, 10000]	最大迭代次数
	max_seconds = 0 (不对运行时长设限制)	[0,INT_MAX_VALUE]	运行时长
	batch_size = 1000	(0, 1048575]	一次训练所选取的样本数
	learning_rate = 0.8	(0, DOUBLE_MAX_VALUE]	学习率
	decay = 0.95	(0, DOUBLE_MAX_VALUE]	权值衰减率
	tolerance = 0.0005	(0, DOUBLE_MAX_VALUE]	公差
	seed = 0（对 seed 取随机值）	[0, INT_MAX_VALUE]	种子
	just for linear、SVM：	linear/gaussian/polynomial	核函数

算子	超参(默认值)	取值范围	超参描述
pca	kernel = “linear”		
	just for linear、SVM: components = MAX(2*features, 128)	[0, INT_MAX_VALUE]	高维空间维数
	just for linear、SVM: gamma = 0.5	(0, DOUBLE_MAX_VALUE]	gaussian 核函数参数
	just for linear、SVM: degree = 2	[2, 9]	polynomial 核函数参数
	just for linear、SVM: coef0 = 1.0	[0, DOUBLE_MAX_VALUE]	polynomial 核函数的参数
	just for SVM：lambda = 0.01	(0, DOUBLE_MAX_VALUE)	正则化参数
	just for pca: number_components	（0，INT_MAX_VALUE]	降维的目标维度
GD: multiclass	classifier= “svm_classification”	svm_classification\logistic_regression	多分类任务的分类器
Kmeans	max_iterations = 10	[1, 10000]	最大迭代次数
	num_centroids = 10	[1, 1000000]	簇的数目

算子	超参(默认值)	取值范围	超参描述
	tolerance = 0.00001	(0,1]	中心点误差
	batch_size = 10	[1,1048575]	一次训练所选取的样本数
	num_features = 2	[1, INT_MAX_VALUE]	输入样本特征数
	distance_function = “L2_Squared”	L1\L2\L2_Squared\Linf	正则化方法
	seeding_function = “Random++”	“Random++” \ “KMeans‖”	初始化种子点方法
	verbose = 0U	{ 0, 1, 2 }	冗长模式
	seed = 0U	[0, INT_MAX_VALUE]	种子
xgboost:xgboost_regression_lo	n_iter=10	(0, 10000]	迭代次数
	batch_size=10000	(0, 1048575]	一次训练所选取的样本数
	booster= “gbtree”	gbtree\gblinear\dart	booster种类
	tree_method= “auto”	auto\exact\approx\hist\gpu_hist 注意：gpu_hist 参数需要相应的库 GPU 版本，否则 DB4AI 平台不支持该值。	树构建算法

算子	超参(默认值)	取值范围	超参描述
gistic、xgboost_binary_logistic、xgboost_regression_gamma、xgboost_regression_squar	eval_metric= “rmse”	rmse\rmsle\map\mae\auc\aucpr	验证数据的评估指标
	seed=0	[0, 100]	种子
	nthread=1	(0, MAX_MEMORY_LIMIT]	并发量
	max_depth=5	(0, MAX_MEMORY_LIMIT]	树的最大深度，该超参仅对树型booster生效。
	gamma=0.0	[0, 1]	叶节点上进行进一步分区所需的最小损失减少
	eta=0.3	[0, 1]	更新中使用的步长收缩，以防止过拟合
	min_child_weight=1	[0, INT_MAX_VALUE]	孩子节点中所需的实例权重的最小总和
	verbosity=1	0 (silent)\1 (warning)\2 (info)\3	打印信息

算子	超参(默认值)	取值范围	超参描述
ed err or		(debug)	的详细程度
MAX_MEMORY_LIMIT = 最大内存加载的元组数量			
GS_MAX_COLS = 数据库单表最大属性数量			

- 模型保存成功，则返回创建成功信息：

```
MODEL CREATED. PROCESSED x
```

3、 查看模型信息。

当训练完成后模型会被存储到系统表 gs.*model*.warehouse 中。系统表 gs.*model*.warehouse 可以查看到关于模型本身和训练过程的相关信息。

关于模型的详细描述信息以二进制的形式存储在系统表中，用户可用过使用函数 gs.*explain*.model 完成对模型的查看，语句如下：

```
panweidb=# select * from gs_explain_model("iris_classification_model");
  DB4AI MODEL
------------------------------
  Name: iris_classification_model
  Algorithm: xgboost_regression_logistic
  Query: CREATE MODEL iris_classification_model
  USING xgboost_regression_logistic
  FEATURES sepal_length, sepal_width,petal_length,petal_width
  TARGET target_type < 2
  FROM tb_iris_1
  WITH nthread=4, max_depth=8;
  Return type: Float64
  Pre-processing time: 0.000000
  Execution time: 0.001443
  Processed tuples: 78
```

```
  Discarded tuples: 0
  n_iter: 10
  batch_size: 10000
  max_depth: 8
  min_child_weight: 1
  gamma: 0.0000000000
  eta: 0.3000000000
  nthread: 4
  verbosity: 1
  seed: 0
  booster: gbtree
  tree_method: auto
  eval_metric: rmse
  rmse: 0.2648450136
  model size: 4613
```

4、 利用已存在的模型做推断任务。

使用“SELECT”和“PREDICT BY”关键字利用已有模型完成推断任务。

查询语法：SELECT...PREDICT BY....(FEATURES....)...FROM...;

```
panweidb=# SELECT id, PREDICT BY iris_classification (FEATURES sepal_length,sepa
l_width,petal_length,petal_width) as "PREDICT" FROM tb_iris limit 3;

id |PREDICT
---+-----
  84|     2
  85|     0
  86|     0
(3 rows)
```

针对相同的推断任务，同一个模型的结果是大致稳定的。且基于相同的超参数和训练集训练的模型也具有稳定性，同时 AI 模型训练存在随机成分（每个 batch 的数据分布、随机梯度下降），所以不同的模型间的计算表现、结果允许存在小的差别。

5、 查看执行计划。

使用 explain 语句可对“CREATE MODEL”和“PREDICT BY”的模型训练或预测过程中的执行计划进行分析。Explain 关键字后可直接拼接 CREATE MODEL/ PREDICT BY 语句（子句），也可接可选的参数，支持的参数如下：

表 5 EXPLAIN 支持的参数

参数名	描述
ANALYZE	布尔型变量，追加运行时间、循环次数等描述信息
VERBOSE	布尔型变量，控制训练的运行信息是否输出到客户端
COSTS	布尔型变量
CPU	布尔型变量
DETAIL	布尔型变量，不可用。
NODES	布尔型变量，不可用
NUM_NODES	布尔型变量，不可用
BUFFERS	布尔型变量
TIMING	布尔型变量
PLAN	布尔型变量
FORMAT	可选格式类型：TEXT / XML / JSON / YAML

示例：

```
panweidb=# Explain CREATE MODEL patient_logisitic_regression USING logistic_reg
ression FEATURES second_attack, treatment TARGET trait_anxiety > 50 FROM patient
s WITH batch_size=10, learning_rate = 0.05;
                          QUERY PLAN
-----------------------------------
  Train Model - logistic_regression  (cost=0.00..0.00 rows=0 width=0)
  ->  Materialize  (cost=0.00..41.08 rows=1776 width=12)
        ->  Seq Scan on patients  (cost=0.00..32.20 rows=1776 width=12)
```

```
(3 rows)
```

6、 异常场景。

训练阶段

场景一：当超参数的设置超出取值范围，模型训练失败，返回 ERROR，并提示错误，例如：

```
panweidb=# CREATE MODEL patient_linear_regression USING linear_regression FEATURES second_attack,treatment TARGET trait_anxiety  FROM patients WITH optimizer='aa';
ERROR:  Invalid hyperparameter value for optimizer. Valid values are: gd, ngd.
```

场景二：当模型名称已存在，模型保存失败，返回 ERROR，并提示错误原因，例如：

```
panweidb=# CREATE MODEL patient_linear_regression USING linear_regression FEATURES second_attack,treatment TARGET trait_anxiety  FROM patients;
ERROR:  The model name "patient_linear_regression" already exists in gs_model_warehouse.
```

场景三：FEATURE 或者 TARGETS 列是.*，返回 ERROR，并提示错误原因，例如：

```
panweidb=# CREATE MODEL patient_linear_regression USING linear_regression FEATURES *  TARGET trait_anxiety  FROM patients;
ERROR:  FEATURES clause cannot be *
-----------------------------------------------------------
panweidb=# CREATE MODEL patient_linear_regression USING linear_regression FEATURES second_attack,treatment TARGET *  FROM patients;
ERROR:  TARGET clause cannot be *
```

场景四：对于无监督学习方法使用 TARGET 关键字，或者在监督学习方法中不适用 TARGET 关键字，均会返回 ERROR，并提示错误原因，例如：

```
panweidb=# CREATE MODEL patient_linear_regression USING linear_regression FEATURES second_attack,treatment FROM patients;
ERROR:  Supervised ML algorithms require TARGET clause
-------------------------------------------------------------
CREATE MODEL patient_linear_regression USING linear_regression TARGET trait_anxiety  FROM patients;
```

```
ERROR:  Supervised ML algorithms require FEATURES clause
```

场景五：当进行分类任务时 TARGET 列的分类只有 1 种情况，会返回 ERROR，并提示错误原因，例如：

```
panweidb=# CREATE MODEL ecoli_svmc USING multiclass FEATURES f1, f2, f3, f4, f5, f6, f7 TARGET cat FROM (SELECT * FROM db4ai_ecoli WHERE cat='cp');
ERROR:  At least two categories are needed
```

场景六：DB4AI 在训练过程中会过滤掉含有空值的数据，当参与训练的模型数据为空的时候，会返回 ERROR，并提示错误原因，例如：

```
panweidb=# create model iris_classification_model using xgboost_regression_logistic features message_regular target error_level from error_code;
ERROR:  Training data is empty, please check the input data.
```

场景七：DB4AI 的算法对于支持的数据类型是有限制的。当数据类型不在支持白名单中，会返回 ERROR，并提示非法的 oid，可通过 pg._type 查看 OID 确定非法的数据类型，例如：

```
panweidb=# CREATE MODEL ecoli_svmc USING multiclass FEATURES f1, f2, f3, f4, f5, f6, f7, cat TARGET cat FROM db4ai_ecoli ;
ERROR:  Oid type 1043 not yet supported
```

场景八：当 GUC 参数 statement.*timeout 设置了时长，训练超时执行的语句将被终止：执行 CREATE MODEL 语句。训练集的大小、训练轮数.(iteration.)、提前终止条件.(tolerance、max.*seconds.)、并行线程数.(nthread.)等参数都会影响训练时长。当时长超过数据库限制，语句被终止模型训练失败。

模型解析

场景九：当模型名在系统表中查找不到，数据库会报 ERROR，例如：

```
panweidb=# select gs_explain_model("ecoli_svmc");
ERROR:  column "ecoli_svmc" does not exist
```

推断阶段

场景十：当模型名在系统表中查找不到，数据库会报 ERROR，例如：

```
panweidb=# select id, PREDICT BY patient_logistic_regression (FEATURES second_attack,treatment) FROM patients;
ERROR:  There is no model called "patient_logistic_regression".
```

场景十一：当做推断任务 FEATURES 的数据维度和数据类型与训练集存在不一致，将报 ERROR，并提示错误原因，例如：

```
panweidb=# select id, PREDICT BY patient_linear_regression (FEATURES second_attack) FROM patients;
ERROR:  Invalid number of features for prediction, provided 1, expected 2
CONTEXT:  referenced column: patient_linear_regression_pred
-----------------------------------------------------------------
panweidb=# select id, PREDICT BY patient_linear_regression (FEATURES 1,second_attack,treatment) FROM patients;
ERROR:  Invalid number of features for prediction, provided 3, expected 2
CONTEXT:  referenced column: patient_linear_regression_pre
```

说明：

DB4AI 特性需要读取数据参与计算，不适用于密态数据库等情况。

11.1.2.全流程 AI

传统的 AI 任务往往具有多个流程，如数据的收集过程包括数据的采集、数据清洗、数据存储等，在算法的训练过程中又包括数据的预处理、训练、模型的保存与管理等。其中，对于模型的训练过程，又包括超参数的调优过程。诸如此类机器学习模型生命周期的全过程，可大量集成于数据库内部。在距离数据存储侧最近处进行模型的训练、管理、优化等流程，在数据库端提供 SQL 语句式的开箱即用的 AI 全声明周期管理的功能，称之为全流程 AI.

PanWeiDB 实现了部分全流程 AI 的功能，将在本章节中详细展开。

11.1.2.1.PLPython Fenced 模式

在 fenced 模式中添加 plpython 非安全语言。在数据库编译时需要将 python 集成进数据库中，在 configure 阶段加入--with-python 选项。同时也可指定安装 plpython 的 python 路径，添加选项--with-includes='/python-dir=path'。

在启动数据库之前配置 GUC 参数 unix_socket_directory，指定 unix_socket 进程间通讯的文件地址。用户需要提前在 user-set-dir-path 下创建文件夹，并将文件夹权限修改为可读可写可执行状态。

```
unix_socket_directory = '/user-set-dir-path'
```

配置完成，启动数据库。

将 plpython 加入数据库编译，并设置好 GUC 参数 unix_socket_directory 后，在启动数据库的过程中，自动创建 fenced-Master 进程。在数据库不进行 python 编译的情况下，fenced 模式需要手动拉起 master 进程，在 GUC 参数设置完成后，输入创建 master 进程命令。

启动 fenced-Master 进程，命令为：

```
gaussdb --fenced -k /user-set-dir-path -D /user-set-dir-path &
```

完成 fence 模式配置，针对 plpython-fenced UDF 数据库将在 fenced-worker 进程中执行 UDF 计算。

使用指导

创建 extension

❖ 当编译的 plpython 为 python2 时：

```
panweidb=### create extension plpythonu;
CREATE EXTENSION
```

❖ 当编译的 plpython 为 python3 时：

```
panweidb=### create extension plpython3u;
CREATE EXTENSION
```

下面示例是以 python2 为例。

❖ 创建 plpython-fenced UDF

```
create or replace function pymax(a int, b int)
returns INT
language plpythonu fenced
as $$
import numpy
if a > b:
```

```
return a;
else:
return b;
$$;
CREATE FUNCTION
```

❖ 查看 UDF 信息

```
select * from pg_proc where proname='pymax';

-[ RECORD 1 ]---+--------
proname         | pymax
pronamespace    | 2200
proowner        | 10
prolang         | 16388
procost         | 100
prorows         | 0
provariadic     | 0
protransform    | -
proisagg        | f
proiswindow     | f
prosecdef       | f
proleakproof    | f
proisstrict     | f
proretset       | f
provolatile     | v
pronargs        | 2
pronargdefaults | 0
prorettype      | 23
proargtypes     | 23 23
proallargtypes  |
proargmodes     |
proargnames     | {a,b}
proargdefaults  |
prosrc          |
            | import numpy
            | if a > b:
```

```
        |    return a;
        | else:
        |    return b;
        |
probin        |
proconfig     |
proacl        |
prodefaultargpos |
fencedmode    | t
proshippable  | f
propackage    | f
prokind       | f
proargsrc     |
```

运行 UDF

❖ 创建一个数据表：

```
panweidb=# create table temp (a int ,b int) ;
CREATE TABLE
panweidb=# insert into temp values (1,2),(2,3),(3,4),(4,5),(5,6);
INSERT 0 5
```

❖ 运行 UDF：

```
panweidb=# select pymax(a,b) from temp;
  pymax
-------
      2
      3
      4
      5
      6
(5 rows)
```

11.1.2.2.DB4AI-Snapshots 数据版本管理

DB4AI-Snapshots 是 DB4AI 模块用于管理数据集版本的功能。通过 DB4ai-Snapshots 组件，开发者可以简单、快速地进行特征筛选、类型转换等数据预

处理操作，同时还可以像 git 一样对训练数据集进行版本控制。数据表快照创建成功后可以像视图一样进行使用，但是一经发布后，数据表快照便固化为不可变的静态数据，如需修改该数据表快照的内容，需要创建一个版本号不同的新数据表快照。

DB4AI-Snapshots 的生命周期

DB4AI-Snapshots 的状态包括 published、archived 以及 purged。其中，published 可以用于标记该 DB4AI-Snapshots 已经发布，可以进行使用。archived 表示当前 DB4AI-Snapshots 处于“存档期”，一般不进行新模型的训练，而是利用旧数据对新的模型进行验证。purged 则是该 DB4AI-Snapshots 已经被删除的状态，在数据库系统中无法再检索到。

需要注意的是快照管理功能是为了给用户提供统一的训练数据，不同团队成员可以使用给定的训练数据来重新训练机器学习模型，方便用户间协同。为此私有用户和三权分立状态(enableSeparationOfDuty=ON)等涉及不支持用户数据转写等情况将不支持 Snapshot 特性。

用户可以通过“CREATE SNAPSHOT”语句创建数据表快照，创建好的快照默认即为 published 状态。可以采用两种模式创建数据表快照，即为 MSS 以及 CSS 模式，它们可以通过 GUC 参数 db4ai_snapshot_mode 进行配置。对于 MSS 模式，它是采用物化算法进行实现的，存储了原始数据集的数据实体；CSS 则是基于相对计算算法实现的，存储的是数据的增量信息。数据表快照的元信息存储在 DB4AI 的系统目录中。可以通过 db4ai.snapshot 系统表查看到。

可以通过“ARCHIVE SNAPSHOT”语句将某一个数据表快照标记为 archived 状态，可以通过“PUBLISH SNAPSHOT”语句将其再度标记为 published 状态。标记数据表快照的状态，是为了帮助数据科学家进行团队合作使用的。

当一个数据表快照已经丧失存在价值时，可以通过“PURGE SNAPSHOT”语句删除它，以便永久删除其数据并恢复存储空间。

DB4AI-Snapshots 使用指导

1、 创建表以及插入表数据。

数据库内存在已有的数据表，可根据该已有的数据表创建对应的数据表快照。为了后续演示，在此处新建一个名为 t1 的数据表，并向其中插入测试数据。

```
create table t1 (id int, name varchar);
insert into t1 values (1, 'zhangsan');
insert into t1 values (2, 'lisi');
insert into t1 values (3, 'wangwu');
insert into t1 values (4, 'lisa');
insert into t1 values (5, 'jack');
```

通过 SQL 语句，查询搭配数据表内容。

```
SELECT * FROM t1;
id |  name
---+------
  1 | zhangsan
  2 | lisi
  3 | wangwu
  4 | lisa
  5 | jack
(5 rows)
```

2、 使用 DB4AI-Snapshots。

❖ 创建 DB4AI-Snapshots

示例 1：CREATE SNAPSHOT...AS

示例如下，其中，默认版本分隔符为“@”，默认子版本分割符为“.”，该分割符可以分别通过 GUC 参数 db4ai_snapshot_version_delimiter 以及 db4ai_snapshot_version_separator 进行设置。

```
create snapshot s1@1.0 comment is 'first version' as select * from t1;
schema |  name
----+----
  public | s1@1.0
(1 row)
```

上述结果提示已经创建了数据表 s1 的快照，版本号为 1.0。创建好后的数据表快照可以像使用一般视图一样进行查询，但不支持通过“INSERT INTO”语句进行更新。例如下面几种语句都可以查询到数据表快照 s1 的对应版本 1.0 的内容：

```
SELECT * FROM s1@1.0;
SELECT * FROM public.s1@1.0;
SELECT * FROM public . s1 @ 1.0;
id |  name
---+------
  1 | zhangsan
  2 | lisi
  3 | wangwu
  4 | lisa
  5 | jack
(5 rows)
```

可以通过下列 SQL 语句修改数据表 t1 的内容：

```
UPDATE t1 SET name = 'tom' where id = 4;
insert into t1 values (6, 'john');
insert into t1 values (7, 'tim');
```

再检索数据表 t1 的内容时，发现虽然数据表 t1 的内容已经发生变化，但是数据表快照 s1@1.0 版本的查询结果并未发生变化。由于数据表 t1 的数据已经发生了改变，如果将当前数据表的内容作为版本 2.0，则可创建快照 **s1@2.0**,创建的 SQL 语句如下：

```
create snapshot s1@2.0 as select * from t1;
```

通过上述例子，我们可以发现，数据表快照可以固化数据表的内容，避免中途对数据的改动造成机器学习模型训练时的不稳定，同时可以避免多用户同时访问、修改同一个表时造成的锁冲突。

示例 2：CREATE SNAPSHOT...FROM

SQL 语句可以对一个已经创建好的数据表快照进行继承，利用在此基础上进行的数据修改产生一个新的数据表快照。例如：

```
create snapshot s1@3.0 from @1.0 comment is 'inherits from @1.0' using (INSERT V
ALUES(6, 'john'), (7, 'tim'); DELETE WHERE id = 1);
schema | name
----+----
  public | s1@3.0
(1 row)
```

其中，“@”为数据表快照的版本分隔符，from 子句后加上已存在的数据表快照，用法为“@”+版本号，USING 关键字后加入可选的几个操作关键字（INSERT .../UPDATE .../DELETE .../ALTER ...）,其中“INSERT INTO”以及“DELETE FROM”语句中的“INTO”、“FROM”等与数据表快照名字相关联的子句可以省略。

示例中，基于前述 s1@1.0 快照，插入 2 条数据，删除 1 条新的数据，新生成的快照 s1@3.0，检索该 s1@3.0：

```
SELECT * FROM s1@3.0;
id | name
--+------
  2 | lisi
  3 | wangwu
  4 | lisa
  5 | jack
  6 | john
  7 | tim
(7 rows)
```

❖ 删除数据表快照 SNAPSHOT

```
purge snapshot s1@3.0;
schema | name
----+----
  public | s1@3.0
(1 row)
```

此时，已经无法再从 s1@3.0 中检索到数据了，同时该数据表快照在 db4ai.snapshot 视图中的记录也会被清除。删除该版本的数据表快照不会影响其他版本的数据表快照。

❖ 从数据表快照中采样

示例：从 snapshot s1 中抽取数据，使用 0.5 抽样率。

```
sample snapshot s1@2.0 stratify by name as nick at ratio .5;
schema |   name
--------+------
  public | s1nick@2.0
(1 row)
```

可以利用该功能创建训练集与测试集，例如：

```
SAMPLE SNAPSHOT s1@2.0  STRATIFY BY name AS _test AT RATIO .2, AS _train AT RA
TIO .8 COMMENT IS 'training';
schema |    name
--------+--------
  public | s1_test@2.0
  public | s1_train@2.0
(2 rows)
```

❖ 发布数据表快照

采用下述 SQL 语句将数据表快照 s1@2.0 标记为 published 状态：

```
publish snapshot s1@2.0;
schema | name
--------+----
  public | s1@2.0
(1 row)
```

❖ 存档数据表快照

采用下述语句可以将数据表快照标记为 archived 状态：

```
archive snapshot s1@2.0;
schema | name
--------+----
  public | s1@2.0
(1 row)
```

可以通过 db4ai-snapshots 提供的视图查看当前数据表快照的状态以及其他信息：

```
select * from db4ai.snapshot;
id | parent_id | matrix_id | root_id | schema |    name    | owner |              commands
           | comment | published | archived |        created         | row_count
----+-----------+-----------+---------+--------+------------+-------+-----------------------+---------+-----------+----------+-
-------------+-------
  1 |           |           |       1 | public | s1@2.0     | omm | {"select *","from t1 where id > 3",N
ULL} |         | t         | f       | 2021-04-17 09:24:11.139868 |         2
  2 |         1 |           |       1 | public | s1nick@2.0 | omm | {"SAMPLE nick .5 {name}"}
   |         | f         | f       | 2021-04-17 10:02:31.73923  |         0
```

3、 异常场景

❖ 数据表或 db4ai-snapshots 不存在时。

```
purge snapshot s1nick@2.0;
publish snapshot s1nick@2.0;
-----
ERROR:  snapshot public."s1nick@2.0" does not exist
CONTEXT:  PL/pgSQL function db4ai.publish_snapshot(name,name) line 11 at assi
gnment

archive snapshot s1nick@2.0;
------
ERROR:  snapshot public."s1nick@2.0" does not exist
CONTEXT:  PL/pgSQL function db4ai.archive_snapshot(name,name) line 11 at assi
gnment
```

❖ 删除 snapshot 时，有依赖该快照的其他 snapshot，需先确保删除对本快照所依赖的其他快照。

```
purge snapshot s1@1.0;
ERROR:  cannot purge root snapshot 'public."s1@1.0"' having dependent snapshot
s
HINT:  purge all dependent snapshots first
CONTEXT:  referenced column: purge_snapshot_internal
SQL statement "SELECT db4ai.purge_snapshot_internal(i_schema, i_name)"
```

```
PL/pgSQL function db4ai.purge_snapshot(name,name) line 71 at PERFORM
```

4、 相关 GUC 参数

- db4ai_snapshot_mode：

  Snapshot 有 2 种模式：MSS（物化模式，存储数据实体）和 CSS（计算模式，存储增量信息）。Snapshot 可在 MSS 和 CSS 之间切换快照模式，默认是 MSS 模式。

- db4ai_snapshot_version_delimiter：

  该参数为数据表快照版本分隔符。“@”为数据表快照的默认版本分隔符。

- db4ai_snapshot_version_separator

  该参数为数据表快照子版本分隔符。“.”为数据表快照的默认版本分隔符。

5、 DB4AI Schema 下的数据表快照详情 db4ai.snapshot。

```
panweidb=# d db4ai.snapshot
                Table "db4ai.snapshot"
  Column   |         Type          |       Modifiers
-------+---------------+---------------
  id       | bigint                |
  parent_id | bigint                |
  matrix_id | bigint                |
  root_id   | bigint                |
  schema    | name                  | not null
  name      | name                  | not null
  owner     | name                  | not null
  commands  | text[]                | not null
  comment   | text                  |
  published | boolean               | not null default false
  archived  | boolean               | not null default false
  created   | timestamp without time zone | default pg_systimestamp()
  row_count | bigint                | not null
Indexes:
  "snapshot_pkey" PRIMARY KEY, btree (schema, name) TABLESPACE pg_default
  "snapshot_id_key" UNIQUE CONSTRAINT, btree (id) TABLESPACE pg_default
```

说明：

命名空间 DB4AI 是本功能的私有域，不支持在 DB4AI 的命令空间下创建函数索引（functional index）。

11.2.AI in DB: 数据库内 AI 功能

11.2.1.智能 Explain: SQL 语句查询时间预测

11.2.1.1.概述

本功能名为 Predictor，是基于机器学习且具有在线学习能力的查询时间预测工具。通过不断学习数据库内收集的历史执行信息，实现计划的执行时间预测功能。

本特性需要拉起 python 进程 AIEngine，用于模型的训练和推理。

该功能所在目录为 $GAUSSHOME/bin/dbmind/components/predictor. 由于该模块中某些功能涉及到相对复杂的搭建，因此，需要用户切换到该目录中寻找对应文件，并按照本章说明进行部署。

11.2.1.2.环境部署

前提条件

需要保证 PanWeiDB 处于正常状态，用户通过身份验证成功登录 PanWeiDB；用户执行的 SQL 语法正确无报错，且不会导致数据库异常等；历史性能数据窗口内 PanWeiDB 并发量稳定，表结构、表数量不变，数据量无突变，涉及查询性能的 guc 参数不变；进行预测时，需要保证模型已训练并收敛；AiEngine 运行环境稳定。

请求样例

AiEngine 进程与内核进程使用 https 发送请求进行通信，请求样例如下：

```
curl -X POST -d '{"modelName":"modelname"}' -H 'Content-Type: application/json' 'https://IP-address:port/request-API'
```

表 1： AI-Engine 对外接口

Request-API	功能
/check	检查模型是否被正常拉起
/configure	设置模型参数
/train	模型训练
/track_process	查看模型训练日志
/setup	加载历史模型
/predict	模型预测

证书生成

请参考:《PanWeiDB V2.0-S3.0.1_B01 管理员指南->数据库使用->连接数据库->身份认证与通信加密》章节。

环境准备

1、 将工具代码文件夹拷贝至目标环境。

- 假设安装路径为$INSTALL_FOLDER
- 假设目标环境路径为/home/ai_user

```
scp -r $INSTALL_FOLDER/bin/dbmind/predictor ai_user@127.0.0.1:path_to_Predictor
```

2、 拷贝 CA 证书文件夹至 aiEngine 环境中某路径下。

```
cp -r $GAUSSHOME/CA ai_user@127.0.0.1:path_to_CA
```

3、 安装 predictor/install/requirements(-gpu).txt 工具（**该功能比较特殊，与其他 AI 功能不同，需要独立安装依赖**）。

```
有 GPU：pip install -r requirements-gpu.txt
无 GPU：pip install -r requirements.txt
```

拉起 AiEngine

1、 切换至 aiEngine 环境（即拷贝 predictor 的目标环境 ai_user）。

设置 predictor/python/settings.py 中的相关参数。

```
DEFAULT_FLASK_SERVER_HOST = '127.0.0.1'（aiEngine 运行 IP 地址）
DEFAULT_FLASK_SERVER_PORT = '5000'（aiEngine 运行端口号）
```

2、 运行 aiEngine 启动脚本。

```
python path_to_Predictor/python/run.py
```

此时，aiEngine 即在相应端口保持拉起状态，等待内核侧时间预测功能的请求指令。

至此，aiEngine 工具部署完成。

11.2.1.3.使用指导

数据收集

1、 打开数据收集。

a、 设置 ActiveSQL operator 信息相关参数：

```
enable_resource_track=on
resource_track_level=operator
enable_resource_record=on
resource_track_cost=10（默认值为 100000）
```

📖 说明：

- resource_track_cost 需设置为小于需要收集的查询总代价，满足条件的信息才能被收集。
- Cgroups 功能正常加载。

---

b、 信息收集：

执行业务查询语句。

查看实时收集数据：

```
select * from gs_wlm_plan_operator_history;
```

预期：满足 resource_track_duration 和 resource_track_cost 的作业被全量收集。

2、 关闭数据收集。

a、 设置 ActiveSQL operator 信息相关参数：

```
enable_resource_track=off 或
resource_track_level=none 或
resource_track_level=query
```

b、 执行业务查询语句。

等待三分钟之后查看当前节点上的数据：

```
select * from gs_wlm_plan_operator_info;
```

预期：所查表和视图无新增数据。

3、 数据持久化保存。

a、 设置 ActiveSQL operator 信息相关参数：

```
enable_resource_track=on
resource_track_level=operator
enable_resource_record=on
resource_track_duration=0（默认值为 60s）
resource_track_cost=10（默认值为 100000）
```

说明：

- resource_track_cost 需设置为小于需要收集的查询总代价，满足条件的信息才能被收集。
- Cgroups 功能正常加载。

---

b、 执行业务查询语句。

等待三分钟之后查看当前节点上的数据：

```
select * from gs_wlm_plan_operator_info;
```

预期：满足 resource_track_duration 和 resource_track_cost 的作业被全量收集。

模型管理（系统管理员用户）

说明：

模型管理操作需要在数据库正常的状态下进行。

1、 新增模型：

INSERT INTO gs_opt_model values('.....');

示例：

```
INSERT INTO gs_opt_model values('rlstm', 'model_name', 'datname', '127.0.0.1', 5000, 2000, 1, -1, 64, 512, 0 , false, false, '{S, T}', '{0,0}', '{0,0}', 'Text');
```

说明：

- 具体模型参数设置请参考 GS_OPT_MODEL。
- 目前 "template_name" 列只支持 "rlstm";
- "datname" 列请和用于模型使用和训练的数据库保持一致，否则无法使用。
- "model_name" 一列需要满足 unique 约束。

2、 修改模型参数：

```
UPDATE gs_opt_model SET <attribute> = <value> WHERE model_name = <target_model_name>;
```

3、 删除模型：

```
DELETE FROM gs_opt_model WHERE model_name = <target_model_name>;
```

4、 查询现有模型及其状态：

```
SELECT * FROM gs_opt_model;
```

模型训练（系统管理员用户）

1、 配置/添加模型训练参数：参考模型管理（系统管理员用户）

例：

模型添加：

```
INSERT INTO gs_opt_model values('rlstm', 'default', 'postgres', '127.0.0.1', 5000, 2000, 1, -1, 64, 512, 0 , false, false, '{S, T}', '{0,0}', '{0,0}', 'Text');
```

训练参数更新：

```
UPDATE gs_opt_model SET <attribute> = <value> WHERE model_name = <target_model_name>;
```

2、 前提条件为数据库状态正常且历史数据正常收集：

删除原有 encoding 数据：

```
DELETE FROM gs_wlm_plan_encoding_table;
```

进行数据编码，需要指定数据库名：

```
SELECT gather_encoding_info('postgres');
```

开始训练：

```
SELECT model_train_opt('rlstm', 'default');
```

3、 获取 AI Engine 侧模型训练日志相对路径：

```
SELECT * FROM track_model_train_opt('rlstm', 'default');
```

模型预测

📖 说明：

- 模型预测功能需在数据库状态正常、指定模型已被训练且收敛的条件下进行。
- 目前，模型训练参数的标签设置中需要包含“S”标签，explain 中才可显示“p-time”预测值。
- 例：INSERT INTO gs_opt_model values('rlstm', 'default', 'postgres', '127.0.0.1', 5000, 1000, 1, -1, 50, 500, 0 , false, false, '{S, T}', '{0,0}', '{0,0}', 'Text');

1、 调用 explain 接口：

```
explain (analyze on, predictor <model_name>)
SELECT ...
```

预期结果：

```
例：Row Adapter（cost=110481.35..110481.35 rows=100 p-time=99..182 width=100)
(actual time=375.158..375.160 rows=2 loops=1)
其中，“p-time”列为标签预测值。
```

其他功能

1、 检查 AiEngine 是否可连接：

```
panweidb=# select check_engine_status('aiEngine-ip-address',running-port);
```

2、 查看模型对应日志在 AiEngine 侧的保存路径：

```
panweidb=# select track_model_train_opt('template_name', 'model_name');
```

11.2.1.4.最佳实践

相关参数解释参考表**表 1** GS_OPT_MODEL。

表 1 GS_OPT_MODEL

模型参数	参数建议
template_name	‘rlstm’
model_name	自定义，如‘open_ai’，需满足 unique 约束。
datname	所服务 database 名称，如‘postgres’。
ip	aiEngine-ip 地址，如‘127.0.0.1’。
port	aiEngine 侦听端口，如‘5000’。
max_epoch	迭代次数，推荐较大数值，保证收敛效果，如‘2000’。
learning_rate	(0, 1]浮点数，推荐较大的学习率，助于加快收敛速度。
dim_red	特征值降维系数：‘-1’：不采用 PCA 降维，全量特征；‘（0，1]’区间浮点数：越小，训练维度越小，收敛速度越快，但影响训练准确率。
hidden_units	特征值维度较高时，建议适度增大此参数，提高模型复杂度，如‘64，128……’
batch_size	根据编码数据量，较大数据量推荐适度增大此参数，加快模型收

模型参数	参数建议
	敛，如‘256，512……’
其他参数	参考表 GS_OPT_MODEL

推荐参数配置：

```
INSERT INTO gs_opt_model values('rlstm', 'open_ai', 'postgres', '127.0.0.1', 5000, 20
00，1, -1, 64, 512, 0 , false, false, '{S, T}', '{0,0}', '{0,0}', 'Text');
```

11.2.1.5.常见问题处理

AI Engine 配置问题

- **AiEngine 启动失败**：请检查 ip 地址，端口是否可用；CA 证书路径是否存在。
- **发起请求 AiEngine 无响应**：请检查通信双方 CA 证书是否一致。
- **训练，测试场景失败**：请检查模型文件保存路径是否存在；训练预测文件是否在正确下载。

数据库内部报错问题

问题：AiEngine 链接失败。

```
ERROR:  AI engine connection failed.
CONTEXT:  referenced column: model_train_opt
```

处理方法：检查 AIEngine 是否正常拉起或重启 AIEngine；检查通信双方 CA 证书是否一致；检查模型配置信息中的 ip 和端口是否匹配；

问题：模型不存在。

```
ERROR:  OPT_Model not found for model name XXX
CONTEXT:  referenced column: track_model_train_opt
```

处理方法：检查 GS_OPT_MODEL 表中是否存在执行语句中“model_name”对应的模型；使用预测功能报错时，检查模型是否已被训练。

12. GUC参数说明

用户可以通过 GUC(Grand Unified Configuration)参数来控制 PanWeiDB 数据库的部署形态和运行行为。GUC 参数种类繁多，本章主要介绍这些 GUC 参数的含义、使用场景以及配置方法等。

12.1.GUC 使用说明

功能描述

数据库提供了许多运行参数，配置这些参数可以影响数据库系统的行为。在修改这些参数时请确保用户理解了这些参数对数据库的影响，否则可能会导致无法预料的结果。

注意事项

- 参数中如果取值范围为字符串，此字符串应遵循操作系统的路径和文件名命名规则。
- 取值范围最大值为 INT_MAX 的参数，此选项最大值跟所在的操作系统有关。
- 取值范围最大值为 DBL_MAX 的参数，此选项最大值跟所在的操作系统有关。

12.2.文件位置

数据库安装后会自动生成三个配置文件（postgresql.conf、pg_hba.conf 和 pg_ident.conf），并统一存放在数据目录（data）下。用户可以使用本节介绍的方法修改配置文件的名称和存放路径。

修改任意一个配置文件的存放目录时，postgresql.conf 里的 data_directory 参数必须设置为实际数据目录（data）。

【须知】
考虑到配置文件修改一旦出错对数据库的影响很大，不建议安装后再修改本节的配置文件。

data_directory

参数说明：设置 PanWeiDB 的数据目录（data 目录）。此参数可以通过如下方式指定。

- 在安装 PanWeiDB 时指定。
- 该参数属于 POSTMASTER 类型参数，请参考：配置运行参数->重设参数章节中表 1 对应设置方法进行设置。

取值范围：字符串，长度大于 0

默认值：安装时指定，如果在安装时不指定，则默认不初始化数据库。

config_file

参数说明：设置主服务器配置文件名称（postgresql.conf）。

该参数属于 POSTMASTER 类型参数，请参考：配置运行参数->重设参数章节中表 1 对应设置方法进行设置，不支持使用：配置运行参数->重设参数章节中表 2 的方式四进行修改。

取值范围：字符串，长度大于 0

默认值：postgresql.conf（实际安装可能带有绝对目录）

hba_file

参数说明：设置基于主机认证（HBA）的配置文件（pg_hba.conf）。此参数只能在配置文件 postgresql.conf 中指定。

该参数属于 POSTMASTER 类型参数，请参考：配置运行参数->重设参数章节中表 1 对应设置方法进行设置。

取值范围：字符串

默认值：pg_hba.conf（实际安装可能带有绝对目录）

ident_file

参数说明：设置用于客户端认证的配置文件的名称（pg_ident.conf）。

该参数属于 POSTMASTER 类型参数，请参考：配置运行参数->重设参数章节中表 1 对应设置方法进行设置。

取值范围：字符串

默认值：pg_ident.conf（实际安装可能带有绝对目录）

external_pid_file

参数说明：声明可被服务器管理程序使用的额外 PID 文件。

该参数属于 POSTMASTER 类型参数，请参考：配置运行参数->重设参数章节中表 1 对应设置方法进行设置。

【须知】
这个参数只能在数据库服务重新启动后生效。

取值范围：字符串

默认值：空

12.3.连接和认证

12.3.1.连接设置

介绍设置客户端和服务器连接方式相关的参数。

listen_addresses

参数说明：声明服务器侦听客户端的 TCP/IP 地址。

该参数指定 PanWeiDB 服务器使用哪些 IP 地址进行侦听，如 IPV4 或 IPV6（若支持）。服务器主机上可能存在多个网卡，每个网卡可以绑定多个 IP 地址，该参数就是控制 PanWeiDB 到底绑定在哪个或者哪几个 IP 地址上。而

客户端则可以通过该参数中指定的 IP 地址来连接 PanWeiDB 或者给 PanWeiDB 发送请求。

该参数属于 POSTMASTER 类型参数，请参考：配置运行参数->重设参数章节中表 1 对应设置方法进行设置。

取值范围：

- 主机名或 IP 地址，多个值之间用英文逗号分隔。
- 星号“”或“0.0.0.0”表示侦听所有 IP 地址。配置侦听所有 IP 地址存在安全风险，不推荐用户使用。必须与有效地址结合使用（比如本地 IP 等），否则，可能造成 Build 失败的问题。同时，主备环境下配置为“”或“0.0.0.0”时，主节点数据库路径下 postgresql.conf 文件中的 localport 端口号不能为数据库 dataPortBase+1，否则会导致数据库无法启动。
- 若存在非法 IP 时，进程启动阶段会报错退出。

默认值：数据库实例安装好后，根据 XML 配置文件中不同实例的 IP 地址配置不同默认值。DN 的默认参数值为：listen_addresses = 'x.x.x.x'。

local_bind_address

参数说明：声明当前节点连接 PanWeiDB 其他节点绑定的本地 IP 地址。

该参数属于 POSTMASTER 类型参数，请参考：配置运行参数->重设参数章节中表 1 对应设置方法进行设置。

默认值：数据库实例安装好后，根据 XML 配置文件中不同实例的 IP 地址配置不同默认值。DN 的默认参数值为：local_bind_address = 'x.x.x.x'。

port

参数说明：PanWeiDB 服务侦听的 TCP 端口号。

该参数属于 POSTMASTER 类型参数，请参考：配置运行参数->重设参数章节中表 1 对应设置方法进行设置。

【说明】

该参数由安装时的配置文件指定，请勿轻易修改，否则修改后会影响数据库正常通信。

取值范围：整型，1～65535

【说明】

- 设置端口号时，请设置一个未被占用的端口号。设置多个实例的端口号，不可冲突。
- 1~1023 为操作系统保留端口号，请不要使用。
- 通过配置文件安装数据库实例时，配置文件中的端口号需要注意通信矩阵预留端口。如：DN 还需保留 dataPortBase+1 作为内部工具使用端口，保留 dataPortBase+6 作为流引擎消息队列通信端口等。故数据库实例安装阶段，port 最大值为：DN 可设置 65529，同时需要保证端口号不冲突。
- 通过 gs_guc set 的方式修改了 port 值之后，需要手动修改下静态配置文件 static_config_files 里面的端口信息后，才可以生效。

默认值：5432（实际值由安装时的配置文件指定）static_config_files

max_connections

参数说明：允许和数据库连接的最大并发连接数。此参数会影响 PanWeiDB 的并发能力。

该参数属于 POSTMASTER 类型参数，请参考：配置运行参数->重设参数章节中表 1 对应设置方法进行设置。

取值范围：整型。最小值为 10（要大于 max_wal_senders），理论最大值为 262143，实际最大值为动态值，计算公式为“262143 - job_queue_processes - autovacuum_max_workers - AUXILIARY_BACKENDS - AV_LAUNCHER_PROCS - max_inner_tool_connections”。

job_queue_processes、autovacuum_max_workers 和 max_inner_tool_connections 的值取决于对应 GUC 参数的设置。

AUXILIARY_BACKENDS 为预留辅助线程数，固定为 20。
AV_LAUNCHER_PROCS 为预留 autovacuum 的 lancher 线程数，固定为 2。

在资源池化模式下，除去以上规则外，最大值还需要满足不超过 16000。

默认值：

- 200：编译安装数据库或极简安装数据库的情况下。
- 5000：使用 om 安装数据库的情况下。

如果该默认值超过内核支持的最大值（在执行 pw_initdb 的时候判断），系统会提示错误。

设置建议：

- 数据库主节点中此参数建议保持默认值。数据库节点中此参数建议设置为数据库主节点的个数乘以数据库主节点中此参数的值。
- 增大这个参数可能导致 PanWeiDB 要求更多的 SystemV 共享内存或者信号量，可能超过操作系统缺省配置的最大值。这种情况下，请酌情对数值加以调整。

max_inner_tool_connections

参数说明：允许和数据库连接的工具的最大并发连接数。此参数会影响 PanWeiDB 的工具连接并发能力。

该参数属于 POSTMASTER 类型参数，请参考：配置运行参数->重设参数章节中表 1 对应设置方法进行设置。

取值范围：整型，最小值为 1，最大值为 262143。

默认值：数据库节点为 50。如果该默认值超过内核支持的最大值（在执行 pw_initdb 的时候判断），系统会提示错误。

设置建议：

数据库主节点中此参数建议保持默认值。

增大此参数可能导致 PanWeiDB 要求更多的 SystemV 共享内存或者信号量，可能超过操作系统缺省配置的最大值。这种情况下，请酌情对数值加以调整。

sysadmin_reserved_connections

参数说明：为管理员用户预留的最少连接数，不建议设置过大。该参数和 max_connections 参数配合使用，管理员用户的最大连接数等于 max_connections + sysadmin_reserved_connections。

该参数属于 POSTMASTER 类型参数，请参考：配置运行参数->重设参数章节中表 1 对应设置方法进行设置。

取值范围：整型，最小值为 0，最大值为 MIN(262143, max_connections)，max_connections 的计算方法见上文。

默认值：3

注意：启用线程池功能时，若线程池占满将形成处理瓶颈，导致管理员预留连接无法正常建立，作为逃生手段，此时可使用 gsql 通过主端口+1 端口号连入，清理无用会话，即可正常连入。

unix_socket_directory

参数说明：设置 PanWeiDB 服务器侦听客户端连接的 Unix 域套接字目录。

该参数属于 POSTMASTER 类型参数，请参考：配置运行参数->重设参数章节中表 1 对应设置方法进行设置。

该参数的长度限制于操作系统的长度，超过该限制将会导致 Unix-domain socket path "xxx" is too long 的问题。

取值范围：字符串

默认值：空字符串（实际值由安装时配置文件指定）

unix_socket_group

参数说明：设置 Unix 域套接字的所属组（套接字的所属用户总是启动服务器的用户）。可以与选项 unix_socket_permissions 一起用于对套接字进行访问控制。

该参数属于 POSTMASTER 类型参数，请参考：配置运行参数->重设参数章节中表 1 对应设置方法进行设置。

取值范围：字符串，其中空字符串表示当前用户的缺省组。

默认值：空字符串

unix_socket_permissions

参数说明：设置 Unix 域套接字的访问权限。

Unix 域套接字使用普通的 Unix 文件系统权限集。这个参数的值应该是数值的格式（chmod 和 umask 命令可接受的格式）。如果使用自定义的八进制格式，数字必须以 0 开头。

建议设置为 0770（只有当前连接数据库的用户和同组的人可以访问）或者 0700（只有当前连接数据库的用户自己可以访问，同组或者其他人都没有权限）。

该参数属于 POSTMASTER 类型参数，请参考：配置运行参数->重设参数章节中表 1 对应设置方法进行设置。

取值范围：0000-0777

默认值：0777

【说明】
在 Linux 中，文档具有十个属性，其中第一个属性为文档类型，后面九个为权限属性，分别为 Owner、Group 及 Others 这三个组别的 read、write、execute 属性。
文档的权限属性分别简写为 r、w、x，这九个属性三个为一组，也可以使用数字来表示文档的权限，对照表如下：
❖ r：4

- w：2
- x：1
- -：0

同一组（owner/group/others）的三个属性是累加的。

例如，-rwxrwx---表示这个文档的权限为：

- owner = rwx = 4+2+1 = 7
- group = rwx = 4+2+1 = 7
- others = --- = 0+0+0 = 0

所以其权限为 0770。

application_name

参数说明：当前连接请求当中，所使用的客户端名称。

该参数属于 USERSET 类型参数，请参考：配置运行参数->重设参数章节中表 1 对应设置方法进行设置。

在备机请求主机进行日志复制时，如果该参数非空串，那么会被用来作为备机在主机上的流复制槽名字。此时，如果该参数长度超过 61 个字节，那么流复制槽名字只会截取使用前 61 个字节的字符。

取值范围：字符串。

默认值：空字符串（连接到后端的应用名，以实际安装为准）

connection_info

参数说明：连接数据库的驱动类型、驱动版本号、当前驱动的部署路径和进程属主用户。

该参数属于 USERSET 类型参数，属于运维类参数，不建议用户设置。

取值范围：字符串。

默认值：空字符串

【说明】

- 空字符串，表示当前连接数据库的驱动不支持自动设置 connection_info 参数或应用程序未设置。
- 驱动连接数据库的时候自行拼接的 connection_info 参数格式如下：

{"driver_name":"ODBC","driver_version": "(PanWeiDB X.X.X build 13b34b53) compiled at 2023-05-08 02:59:43 commit 2143 last mr 131 debug","driver_path":"/usr/local/lib/psqlodbcw.so","os_user":"omm"}

默认显示 driver_name 和 driver_version，driver_path 和 os_user 的显示由用户控制。

tcp_user_timeout

参数说明： 在支持 TCP_USER_TIMEOUT 套接字选项的操作系统上，指定传输的数据在 TCP 连接被强制关闭连接之前，指定传输的数据可以保持未确认状态的最大时长。

该参数属于 SIGHUP 类型参数，请参考重设参数表 1 中对应设置方法进行设置。

取值范围： 0 ~ 3600000，单位 ms

默认值： 0

enable_dolphin_proto

参数说明： 是否开启 dolphin 数据库协议功能。

该参数属于 POSTMASTER 类型参数，请参考重设参数表 1 中对应设置方法进行设置。

取值范围： 布尔型

- on:表示开启 dolphin 数据库协议。
- off:表示关闭 dolphin 数据库协议。

默认值： off

dolphin_server_port

参数说明： dophin 协议插件监听的 TCP 端口号。

该参数属于 POSTMASTER 类型参数，请参考重设参数表 1 中对应设置方法进行设置。

取值范围： 整型，1024～65535

默认值： 3308

【说明】

- 当加载了 dophin 插件，并且开启了 dolphin 数据库协议后，可以使用此功能。
- 设置端口号时，请设置一个未被占用的端口号，不能同 PanWeiDB 数据库协议的端口号冲突。

panwei_login_info

参数说明：控制用户登录时是否记录其访问信息，并在成功登录后显示访问历史信息。

该参数属于 SIGHUP 类型参数，请参考重设参数表 1 中对应设置方法进行设置。

取值范围： 布尔型

- on：记录访问信息。
- off：不记录访问信息。

默认值： off

light_comm

参数说明：指定服务器是否使用轻量通信方式。该参数指定服务器是否使用基于轻量锁和非阻塞 socket 的通信方式。

该参数属于 POSTMASTER 类型参数，请参考重设参数表 1 中对应设置方法进行设置。

取值范围： 布尔型

- on：表示使用轻量通信方式。
- off：表示不使用轻量通信方式。

默认值：off

comm_shm

参数说明： 控制是否开启共享内存连接，共享内存连接是指使用共享内存的一块区域作为客户端和数据库服务器之间通信的通道。

该参数属于 POSTMASTER 类型参数，请参考重设参数表 1 中对应设置方法进行设置。

取值范围： 布尔型

- on：开启共享内存连接。
- off：关闭共享内存连接。

默认值： off

12.3.2.安全和认证（postgresql.conf）

介绍设置客户端和服务器的安全认证方式的相关参数。

authentication_timeout

参数说明：完成客户端认证的最长时间。如果一个客户端没有在这段时间里完成与服务器端的认证，则服务器自动中断与客户端的连接，这样就避免了出问题的客户端无限制地占用连接数。

该参数属于 SIGHUP 类型参数，请参考：配置运行参数->重设参数章节中表 1 对应设置方法进行设置。

取值范围：整型，最小值为 1，最大值为 600，最小单位为 s。

默认值：1min

auth_iteration_count

参数说明：认证加密信息生成过程中使用的迭代次数。

该参数属于 SIGHUP 类型参数，请参考：配置运行参数->重设参数章节中表 1 对应设置方法进行设置。

取值范围：整型，2048-134217728。

默认值：10000

【须知】
迭代次数设置过小会降低口令存储的安全性，设置过大会导致认证、用户创建等涉及口令加密的场景性能劣化，请根据实际硬件条件合理设置迭代次数，推荐采用默认迭代次数。

session_authorization

参数说明：当前会话的用户标识。

该参数属于 USERSET 类型参数，此参数为内部参数，只能通过 SET SESSION AUTHORIZATION 语法设置，该语法请参考：SQL 语法参考->SQL 语法->SET SESSION AUTHORIZATION 章节，不支持直接设置。

取值范围：字符串。

默认值：NULL

session_timeout

参数说明：表明与服务器建立链接后，不进行任何操作的最长时间。

该参数属于 USERSET 类型参数，请参考：配置运行参数->重设参数章节中表 1 对应设置方法进行设置。

取值范围：整型，0-86400，最小单位为 s，0 表示关闭超时设置。

默认值：10min

【须知】
PanWeiDB gsql 客户端中有自动重连机制，所以针对初始化用户本地连接，超时后 gsql 表现的现象为断开后重连。

idle_in_transaction_session_timeout

参数说明：表明与服务器建立链接后，如果当前会话处于事务中，不进行任何操作的最长时间。

该参数属于 USERSET 类型参数，请参考重设参数表 1 中对应设置方法进行设置。

取值范围：整型，0～86400000，最小单位为 ms，0 表示关闭超时设置。

默认值：0

【须知】
PanWeiDB gsql 客户端中有自动重连机制，所以针对初始化用户本地连接，超时后 gsql 表现的现象为断开后重连。

ssl

参数说明：启用 SSL 连接。请在使用这个选项之前阅读：《PanWeiDB V2.0-S3.0.1_B01 管理员指南->数据库使用->连接数据库->使用 gsql 连接》章节。

该参数属于 POSTMASTER 类型参数，请参考：配置运行参数->重设参数章节中表 1 对应设置方法进行设置。

取值范围：布尔型

- on 表示启用 SSL 连接。
- off 表示不启用 SSL 连接。

【须知】

开启此参数需要同时配置 ssl_cert_file、ssl_key_file 和 ssl_ca_file 等参数及对应文件，不正确的配置可能会导致 PanWeiDB 无法正常启动。

默认值：off

require_ssl

参数说明：设置服务器端是否强制要求 SSL 连接，该参数只有当参数 ssl 为 on 时才有效。请在使用这个选项之前阅读：《PanWeiDB V2.0-S3.0.1_B01 管理员指南->数据库使用->连接数据库->使用 gsql 连接》章节。

该参数属于 SIGHUP 类型参数，请参考：配置运行参数->重设参数章节中表 1 对应设置方法进行设置。

取值范围：布尔型

- on 表示服务器端强制要求 SSL 连接。
- off 表示服务器端对是否通过 SSL 连接不作强制要求。

【须知】

PanWeiDB 目前支持 SSL 的场景为客户端连接数据库主节点场景，该参数目前建议只在数据库主节点中开启。

默认值：off

ssl_ciphers

参数说明：指定 SSL 支持的加密算法列表。ssl_ciphers 设置错误会导致数据库不能正常启动。

该参数属于 POSTMASTER 类型参数，请参考：配置运行参数->重设参数章节中表 1 对应设置方法进行设置。

取值范围：字符串，如果指定多个加密算法，加密算法之间需要以分号分割。详细请参考：《PanWeiDB V2.0-S3.0.1_B01 管理员指南->数据库使用->连接数据库->身份认证与通信加密》章节，获取支持的加密算法。

默认值：ALL

ssl_renegotiation_limit

参数说明：指定在会话密钥重新协商之前，通过 SSL 加密通道可以传输的流量。这个重新协商流量限制机制可以减少攻击者针对大量数据使用密码分析法破解密钥的几率，但是也带来较大的性能损失。流量是指发送和接受的流量总和。使用 SSL 重协商机制可能引入其他风险，因此已禁用 SSL 重协商机制，为保持版本兼容保留此参数，修改参数配置不再起作用。

该参数属于 USERSET 类型参数，请参考：配置运行参数->重设参数章节中表 1 对应设置方法进行设置。

取值范围：整型，最小值为 0，最大值为 2147483647。单位为 KB。其中 0 表示禁用重新协商机制。

默认值：0

ssl_cert_file

参数说明：指定包含 SSL 服务器证书的文件的名称，其相对路径是相对于数据目录的。

该参数属于 POSTMASTER 类型参数，请参考：配置运行参数->重设参数章节中表 1 对应设置方法进行设置。

取值范围：字符串

默认值：server.crt

ssl_key_file

参数说明：指定包含 SSL 私钥的文件名称，其相对路径是相对于数据目录的。

该参数属于 POSTMASTER 类型参数，请参考：配置运行参数->重设参数章节中表 1 对应设置方法进行设置。

取值范围：字符串

默认值：server.key

ssl_ca_file

参数说明：指定包含 CA 信息的文件的名称，其相对路径是相对于数据目录的。

该参数属于 POSTMASTER 类型参数，请参考：配置运行参数->重设参数章节中表 1 对应设置方法进行设置。

取值范围：字符串，其中空字符串表示没有 CA 文件被加载，不进行客户端证书验证。

默认值：空

ssl_crl_file

参数说明：证书吊销列表，如果客户端证书在该列表中，则当前客户端证书被视为无效证书。必须使用相对路径，相对路径是相对于数据目录的。

该参数属于 POSTMASTER 类型参数，请参考：配置运行参数->重设参数章节中表 1 对应设置方法进行设置。

取值范围：字符串，空字符串表示没有吊销列表。

默认值：空

krb_server_keyfile

参数说明：指定 Kerberos 服务主配置文件的位置，详细请参见：《PanWeiDB V2.0-S3.0.1_B01 管理员指南->数据库使用->连接数据库->配置客户端接入认证》章节。

该参数属于 SIGHUP 类型参数，请参考：配置运行参数->重设参数章节中表 1 对应设置方法进行设置。

取值范围：字符串

默认值：空

krb_srvname

参数说明：设置 Kerberos 服务名，详细请参见：《PanWeiDB V2.0-S3.0.1_B01 管理员指南->数据库使用->连接数据库->配置客户端接入认证》章节。

该参数属于 SIGHUP 类型参数，请参考：配置运行参数->重设参数章节中表 1 对应设置方法进行设置。

取值范围：字符串

默认值：postgres

krb_caseins_users

参数说明：设置 Kerberos 用户名是否大小写敏感。

该参数属于 SIGHUP 类型参数，请参考：配置运行参数->重设参数章节中表 1 对应设置方法进行设置。

取值范围：布尔型

- on 表示大小写不敏感。
- off 表示大小写敏感。

默认值：off

modify_initial_password

参数说明：当 PanWeiDB 安装成功后，数据库中仅存在一个初始用户（UID 为 10 的用户）。客户通过该帐户初次登录数据库进行操作时，该参数决定是否要对该初始帐户的密码进行修改。

该参数属于 SIGHUP 类型参数，请参考：配置运行参数->重设参数章节中表 1 对应设置方法进行设置。

如果安装过程中未指定初始用户密码，则安装后初始用户密码默认为空，执行其他操作前需要先通过 gsql 客户端设置初始用户的密码。此参数功能不再生效，保留此参数仅为兼容升级场景。

取值范围：布尔型

- on 表示 PanWeiDB 安装成功后初始用户首次登录操作前需要修改初始密码。
- off 表示 PanWeiDB 安装成功后初始用户无需修改初始密码即可进行操作。

默认值：off

password_force_alter

参数说明：设置是否强制新建用户首次登陆修改密码。

该参数属于 SIGHUP 类型参数，请参考：配置运行参数->重设参数章节中表 1 对应设置方法进行设置。

取值范围：布尔型

- on 表示当用户密码为新时，会被强制修改。
- off 表示不强制修改用户密码。

默认值：off

password_policy

参数说明：在使用 CREATE ROLE/USER 或者 ALTER ROLE/USER 命令创建或者修改 PanWeiDB 帐户时，该参数决定是否进行密码复杂度检查。关于密码复杂度检查策略请参见：《PanWeiDB V2.0-S3.0.1_B01 管理员指南->安全性->管理用户及权限->设置安全策略->设置密码安全策略》章节。

该参数属于 SIGHUP 类型参数，请参考：配置运行参数->重设参数章节中表 1 对应设置方法进行设置。

【须知】
从安全性考虑，请勿关闭密码复杂度策略。

取值范围：0、1、2

- 0 表示不采用密码复杂度校验策略。
- 1 表示采用默认密码复杂度校验策略。
- 2 表示启用密码复杂度校验，且密码需满足四类字符中的四类。

默认值：1

password_reuse_time

参数说明：在使用 ALTER USER 或者 ALTER ROLE 修改用户密码时，该参数指定是否对新密码进行可重用天数检查。关于密码可重用策略请参见：《PanWeiDB V2.0-S3.0.1_B01 管理员指南->安全性->管理用户及权限->设置安全策略->设置密码安全策略》章节。

该参数属于 SIGHUP 类型参数，请参考：配置运行参数->重设参数章节中表 1 对应设置方法进行设置。

【须知】

修改密码时会检查配置参数 password_reuse_time 和 password_reuse_max。

- 当 password_reuse_time 和 password_reuse_max 都为正数时，只要满足其中任一个，即可认为密码可以重用。
- 当 password_reuse_time 为 0 时，表示不限制密码重用天数，仅限制密码重用次数。

取值范围：浮点型（天），最小值为 30，最大值为 90。

正数表示新密码不能为该值指定的天数内使用过的密码。

默认值：90

password_reuse_max

参数说明：在使用 ALTER USER 或者 ALTER ROLE 修改用户密码时，该参数指定是否对新密码进行可重用次数检查。关于密码可重用策略请参见：

《PanWeiDB V2.0-S3.0.1_B01 管理员指南->安全性->管理用户及权限->设置安全策略->设置密码安全策略》章节。

该参数属于 SIGHUP 类型参数，请参考：配置运行参数->重设参数章节中表 1 对应设置方法进行设置。

【须知】

修改密码时会检查配置参数 password_reuse_time 和 password_reuse_max。

- 当 password_reuse_time 和 password_reuse_max 都为正数时，只要满足其中任一个，即可认为密码可以重用。
- 当 password_reuse_time 为 0 时，表示不限制密码重用天数，仅限制密码重用次数。

取值范围：整型，最小值为 1，最大值为 100。

正整数表示新密码不能为该值指定的次数内使用过的密码。

默认值：3

password_lock_time

参数说明：该参数指定帐户被锁定后自动解锁的时间。关于帐户自动锁定策略请参见：《PanWeiDB V2.0-S3.0.1_B01 管理员指南->安全性->管理用户及权限->设置安全策略->设置密码安全策略》章节。

该参数属于 SIGHUP 类型参数，请参考：配置运行参数->重设参数章节中表 1 对应设置方法进行设置。

【须知】

password_lock_time 和 failed_login_attempts 必须都为正数时锁定和解锁功能才能生效。

取值范围：整型，最小值为 1，最大值为 525600，单位为分钟。

- 正数表示帐户被锁定后，当锁定时间超过 password_lock_time 设定的值时，帐户将会被自行解锁。

❖ 系统管理员可以通过 ALTER USER joe ACCOUNT UNLOCK;手动解锁。

默认值：1440（即 1 天）

failed_login_attempts

参数说明：在任意时候，如果输入密码错误的次数达到 failed_login_attempts 参数设定的值，则当前帐户会被锁定。password_lock_time 参数设定的天数过后，帐户自动解锁。例如，登录时输入密码失败，ALTER USER 时修改密码失败等。关于帐户自动锁定策略请参见：《PanWeiDB V2.0-S3.0.1_B01 管理员指南->安全性->管理用户及权限->设置安全策略->设置密码安全策略》章节。

该参数属于 SIGHUP 类型参数，请参考：配置运行参数->重设参数章节中表 1 对应设置方法进行设置。

【须知】
failed_login_attempts 和 password_lock_time 必须都为正数时锁定和解锁功能才能生效。
不建议将该参数设置为 0，会存在安全隐患。

取值范围：整型，最小值为 0，最大值为 1000。

❖ 正整数表示当错误密码次数达到 failed_login_attempts 设定的值时，当前帐户将被锁定。
❖ 配置为 0 表示当错误密码次数达到 failed_login_attempts 设定的值，也不会锁定当前用户。
❖ 系统管理员可以通过 ALTER USER joe ACCOUNT UNLOCK;手动解锁。

默认值：5

password_encryption_type

参数说明：该字段决定采用何种加密方式对用户密码进行加密存储。修改此参数的配置不会自动触发已有用户密码加密方式的修改，只会影响新创建用户或修改用户密码操作。

该参数属于 SIGHUP 类型参数，请参考：配置运行参数->重设参数章节中表 1 对应设置方法进行设置。

取值范围：0、1、2、3

- 0 表示采用 md5 方式对密码加密。
- 1 表示采用 sha256 和 md5 两种方式分别对密码加密。
- 2 表示采用 sha256 方式对密码加密。
- 3 表示采用 sm3 方式对密码加密。

【须知】

md5 加密算法安全性低，存在安全风险，不建议用户使用。

默认值：1

password_min_length

参数说明：该字段决定帐户密码的最小长度。

该参数属于 SIGHUP 类型参数，请参考：配置运行参数->重设参数章节中表 1 对应设置方法进行设置。

取值范围：整型，6~999 个字符。

默认值：8

password_max_length

参数说明：该字段决定帐户密码的最大长度。

该参数属于 SIGHUP 类型参数，请参考：配置运行参数->重设参数章节中表 1 对应设置方法进行设置。

取值范围：整型，6~999 个字符。

默认值：32

password_min_uppercase

参数说明：该字段决定帐户密码中至少需要包含大写字母个数。

该参数属于 SIGHUP 类型参数，请参考：配置运行参数->重设参数章节中表 1 对应设置方法进行设置。

取值范围：整型，0~999

- 0 表示没有限制。
- 1~999 表示创建账户所指定的密码中至少需要包含大写字母个数。

默认值：0

password_min_lowercase

参数说明：该字段决定帐户密码中至少需要包含小写字母的个数。

该参数属于 SIGHUP 类型参数，请参考：配置运行参数->重设参数章节中表 1 对应设置方法进行设置。

取值范围：整型，0~999

- 0 表示没有限制。
- 1~999 表示创建帐户所指定的密码中至少需要包含小写字母个数。

默认值：0

password_min_digital

参数说明：该字段决定帐户密码中至少需要包含数字的个数。

该参数属于 SIGHUP 类型参数，请参考：配置运行参数->重设参数章节中表 1 对应设置方法进行设置。

取值范围：整型，0~999

- 0 表示没有限制。
- 1~999 表示创建帐户所指定的密码中至少需要包含数字个数。

默认值：0

password_min_special

参数说明：该字段决定帐户密码中至少需要包含特殊字符个数。

该参数属于 SIGHUP 类型参数，请参考：配置运行参数->重设参数章节中表 1 对应设置方法进行设置。

取值范围：整型，0~999

- 0 表示没有限制。
- 1~999 表示创建帐户所指定的密码中至少需要包含特殊字符个数。

默认值：0

password_effect_time

参数说明：该字段决定帐户密码的有效时间。

该参数属于 SIGHUP 类型参数，请参考：配置运行参数->重设参数章节中表 1 对应设置方法进行设置。

取值范围：浮点型，最小值为 30，最大值为 100*365，单位为天。

30~100*365 表示创建帐户所指定的密码有效期，临近或超过有效期系统会提示用户修改密码。

默认值：36500

password_notify_time

参数说明：该字段决定帐户密码到期前提醒的天数。

该参数属于 SIGHUP 类型参数，请参考：配置运行参数->重设参数章节中表 1 对应设置方法进行设置。

取值范围：整型，最小值为 1，最大值为 90，单位为天。

1~90 表示帐户密码到期前提醒的天数。

默认值：7

enable_stat_mask_password

参数说明： 该参数用于控制是否需要密码脱敏操作。

该参数属于 SIGHUP 类型参数，请参考重设参数表 1 中对应设置方法进行设置。

取值范围： 布尔型

- on：开启密码脱敏功能。
- off：关闭密码脱敏功能。

默认值： on

12.3.3.通信库参数

本节介绍通信库相关的参数设置及取值范围等内容。

tcp_keepalives_idle

参数说明：在支持 TCP_KEEPIDLE 套接字选项的系统上，设置发送活跃信号的间隔秒数。不设置发送保持活跃信号，连接就会处于闲置状态。

该参数属于 USERSET 类型参数，为固定参数，用户无法修改此参数，只能查看。

【须知】

- 如果操作系统不支持 TCP_KEEPIDLE 选项，这个参数的值必须为 0。
- 在通过 Unix 域套接字进行的连接的操作系统上，这个参数将被忽略。
- 将该值设置为 0 时，将使用系统的值。
- 该参数在不同的会话之间不共享，也就是说不同的会话连接可能有不同的值。
- 查看该参数时查出来的是当前会话连接内的参数值，而不是 guc 副本的值。

取值范围：0-3600，单位为 s。

默认值：0

tcp_keepalives_interval

参数说明：在支持 TCP_KEEPINTVL 套接字选项的操作系统上，以秒数声明在重新传输之间等待响应的时间。

该参数属于 USERSET 类型参数，请参考：配置运行参数->重设参数章节中表 1 对应设置方法进行设置。

取值范围：0-180，单位为 s。

默认值：0

【须知】

- 如果操作系统不支持 TCP_KEEPINTVL 选项，这个参数的值必须为 0。
- 在通过 Unix 域套接字进行的连接的操作系统上，这个参数将被忽略。
- 将该值设置为 0 时，将使用系统的值。
- 该参数在不同的会话之间不共享，也就是说不同的会话连接可能有不同的值。
- 查看该参数时查出来的是当前会话连接内的参数值，而不是 guc 副本的值。

tcp_keepalives_count

参数说明：在支持 TCP_KEEPCNT 套接字选项的操作系统上，设置 PanWeiDB 服务端在断开与客户端连接之前可以等待的保持活跃信号个数。

该参数属于 USERSET 类型参数，请参考：配置运行参数->重设参数章节中表 1 对应设置方法进行设置。

【须知】

- 如果操作系统不支持 TCP_KEEPCNT 选项，这个参数的值必须为 0。
- 在通过 Unix 域套接字进行连接的操作系统上，这个参数将被忽略。
- 将该值设置为 0 时，将使用系统的值。
- 该参数在不同的会话之间不共享，也就是说不同的会话连接可能有不同的值。
- 查看该参数时查出来的是当前会话连接内的参数值，而不是 guc 副本的值。

取值范围：0-100，其中 0 表示 PanWeiDB 未收到客户端反馈的保持活跃信号则立即断开连接。

默认值：0

comm_proxy_attr

参数说明：通信代理库相关参数配置。

【须知】

- 该参数仅支持欧拉 2.9 系统下的集中式 ARM 单机。
- 本功能在线程池开启状态下生效，即 enable_thread_pool 为 on。
- 配置该参数时需同步配置 GUC 参数 local_bind_address 为 libos_kni 的网卡 IP。
- 参数模板：comm_proxy_attr = '{enable_libnet:true, enable_dfx:false, numa_num:4, numa_bind:[[30,31],[62,63],[94,95],[126,127]]}'
- 可配置参数说明。
  - enable_libnet：是否开启用户态协议，取值范围：true、false。
  - enable_dfx：是否开启通信代理库视图，取值范围：true、false。
  - numa_num：机器环境中 numa 的数量，支持 2P、4P 服务器，取值范围：4、8。

- numa_bind：代理线程绑核参数，每个 numa 两个 CPU 绑核，共 numa_num 组，取值范围：[0，cpu 数-1]。

该参数属于 POSTMASTER 类型参数，请参考：配置运行参数->重设参数章节中表 1 对应设置方法进行设置。

取值范围：字符串，长度大于 0。

默认值：'none'

12.4.资源消耗

12.4.1.内存

介绍与内存相关的参数设置。

【须知】
这些参数只能在数据库服务重新启动后生效。

memorypool_enable

参数说明：设置是否允许使用内存池。

该参数属于 POSTMASTER 类型参数，请参考：配置运行参数->重设参数章节中表 1 对应设置方法进行设置。

取值范围：布尔型

- on 表示允许使用内存池。
- off 表示不允许使用内存池。

默认值：off

memorypool_size

参数说明：设置内存池大小。

该参数属于 POSTMASTER 类型参数，请参考：配置运行参数->重设参数章节中表 1 对应设置方法进行设置。

取值范围：整型，128*1024～INT_MAX/2，单位为 KB。

默认值：512MB

enable_memory_limit

参数说明：启用逻辑内存管理模块。

该参数属于 POSTMASTER 类型参数，请参考：配置运行参数->重设参数章节中表 1 对应设置方法进行设置。

取值范围：布尔型

- on 表示启用逻辑内存管理模块。
- off 表示不启用逻辑内存管理模块。

默认值：on

【注意】

- 若 max_process_memory-shared_buffers-cstore_buffers-元数据少于 2G，PanWeiDB 强制把 enable_memory_limit 设置为 off。其中元数据是 PanWeiDB 内部使用的内存和部分并发参数，如 max_connections、thread_pool_attr、max_prepared_transactions 等参数相关。
- 当该值为 off 时，不对数据库使用的内存做限制，在大并发或者复杂查询时，使用内存过多，可能导致操作系统 OOM 问题。

max_process_memory

参数说明：设置一个数据库节点可用的最大物理内存。

该参数属于 POSTMASTER 类型参数，为固定参数，用户无法修改此参数，只能查看。

取值范围：整型，2*1024*1024～INT_MAX，单位为 KB。

默认值：12GB

设置建议：

数据库节点上该数值需要根据系统物理内存及单节点部署主数据库节点个数决定。建议计算公式如下：(物理内存大小 - vm.min_free_kbytes) * 0.7 / (1 + 主节点个数)。该系数的目的是尽可能保证系统的可靠性，不会因数据库内存膨胀导致节点 OOM。这个公式中提到 vm.min_free_kbytes，其含义是预留操作系统内存供内核使用，通常用作操作系统内核中通信收发内存分配，至少为 5%内存。即，max_process_memory = 物理内存 * 0.665 / (1 + 主节点个数)。

【注意】
当该值设置不合理，即大于服务器物理内存，可能导致操作系统 OOM 问题。

enable_memory_context_control

参数说明：启用检查内存上下文是否超过给定限制的功能。仅适用于 DEBUG 版本。

该参数属于 SIGHUP 类型参数，请参考：配置运行参数->重设参数章节中表 1 对应设置方法进行设置。

取值范围：布尔型

- on 表示启用最大内存上下文限制检查功能。
- off 表示关闭最大内存上下文限制检查功能。

默认值：off

uncontrolled_memory_context

参数说明：启用检查内存上下文是否超过给定限制的功能时，设置不受此功能约束。仅适用于 DEBUG 版本。

该参数属于 USERSET 类型参数，请参考：配置运行参数->重设参数章节中表 1 对应设置方法进行设置。

查询时会在参数值的最前面添加标题含义字符串“MemoryContext white list:”。

取值范围：字符串

默认值：空

shared_buffers

参数说明：设置 PanWeiDB 使用的共享内存大小。增加此参数的值会使 PanWeiDB 比系统默认设置需要更多的 System V 共享内存。

该参数属于 POSTMASTER 类型参数，请参考：配置运行参数->重设参数章节中表 1 对应设置方法进行设置。

取值范围：整型，16～1073741823，单位为 8KB。

shared_buffers 需要设置为 BLCKSZ 的整数倍，BLCKSZ 目前设置为 8KB，即 shared_buffers 需要设置为 8KB 整数倍。改变 BLCKSZ 的值会改变最小值。

默认值：8MB

设置建议：

1、建议设置 shared_buffers 值为内存的 40%以内。行存列存分开对待。行存设大，列存设小。列存：(单服务器内存/单服务器数据库节点个数)*0.4*0.25。

2、如果设置较大的 shared_buffers 需要同时增加 checkpoint_segments 的值，因为写入大量新增、修改数据需要消耗更多的时间周期。

3、如果调整 shared_buffers 参数之后，导致进程重启失败，请参考启动失败的报错信息，采用以下解决方案之一：

- 对应调整操作系统 kernel.shmall、kernel.shmmax、kernel.shmmin 参数。
- 执行 free -g 观察操作系统可用内存和 swap 空间是否足够，如果内存明显不足，请手动停止其他比较占用内存的用户程序。
- 避免设置明显不合理（过大或过小）的 shared_buffers 值。

4、在资源池化模式下，如果主备机同时运行业务，应适当增大本参数。避免由于参数设置过小导致主备机页面交互频繁、网络负载增大，严重情况下可能出现数据库响应较慢甚至无响应的情况。

segment_buffers

参数说明：设置 PanWeiDB 段页式元数据页的内存大小。

该参数属于 POSTMASTER 类型参数，请参考：配置运行参数->重设参数章节中表 1 对应设置方法进行设置。

取值说明：整型，16～1073741823，单位为 8KB。

segment_buffers 需要设置为 BLCKSZ 的整数倍，BLCKSZ 目前设置为 8KB，即 segment_buffers 需要设置为 8KB 整数倍。改变 BLCKSZ 的值会改变最小值。

默认值：8MB

设置建议：

segment_buffers 用来缓存段页式段头的内容，属于关键元数据信息，为了提高性能建议常用的表的段头都能缓存在 buffer 中，不被置换出去。建议按照表的个数（包括索引和 toast 表）* 分区数 * 3 + 128 来设置。乘以 3 是因为每个表（分区）会有一些额外的元数据段，一般一个表有 3 个段。最后+128 因为段页式表空间管理需要一定数量的 buffer。

bulk_write_ring_size

参数说明：大批量数据写入触发时（例如 copy 动作），该操作使用的环形缓冲区大小。

该参数属于 USERSET 类型参数，请参考：配置运行参数->重设参数章节中表 1 对应设置方法进行设置。

取值范围：整型，16384～2147483647，单位为 KB。

默认值：2GB

设置建议：建议导入压力大的场景中增加数据库节点中此参数配置。

standby_shared_buffers_fraction

参数说明：备实例所在服务器使用 shared_buffers 内存缓冲区大小的比例。

该参数属于 SIGHUP 类型参数，请参考：配置运行参数->重设参数章节中表 1 对应设置方法进行设置。

取值范围：双精度浮点型，0.1~1.0

默认值：0.3

temp_buffers

参数说明：设置每个数据库会话使用的 LOCAL 临时缓冲区的大小。

该参数属于 USERSET 类型参数，请参考：配置运行参数->重设参数章节中表 1 对应设置方法进行设置。

在每个会话的第一次使用临时表之前可以改变 temp_buffers 的值，之后的设置将是无效的。

一个会话将按照 temp_buffers 给出的限制，根据需要分配临时缓冲区。如果在一个并不需要大量临时缓冲区的会话里设置一个大的数值，其开销只是一个缓冲区描述符的大小。当缓冲区被使用，就会额外消耗 8192 字节。

取值范围：整型，100~1073741823，单位为 8KB。

默认值：1MB

max_prepared_transactions

参数说明：设置可以同时处于“预备”状态的事务的最大数目。增加此参数的值会使 PanWeiDB 比系统默认设置需要更多的 System V 共享内存。

当 PanWeiDB 部署为主备双机时，在备机上此参数的设置必须要高于或等于主机上的，否则无法在备机上进行查询操作。

该参数属于 POSTMASTER 类型参数，请参考：配置运行参数->重设参数章节中表 1 对应设置方法进行设置。

取值范围：整型，0~262143。

默认值：10

【说明】
一般不需要对事务显式进行 PREPARE 操作，如果业务对事务进行显示 PREPARE 操作，为避免在准备步骤失败，需要调大该值，大于需要进行 PREPARE 业务的并发数。

work_mem

参数说明：设置内部排序操作和 Hash 表在开始写入临时磁盘文件之前使用的内存大小。ORDER BY、DISTINCT 和 merge joins 都要用到排序操作。Hash 表在散列连接、散列为基础的聚集、散列为基础的 IN 子查询处理中都要用到。

对于复杂的查询，可能会同时并发运行好几个排序或者散列操作，每个都可以使用此参数所声明的内存量，不足时会使用临时文件。同样，好几个正在运行的会话可能会同时进行排序操作。因此使用的总内存可能是 work_mem 的好几倍。

该参数属于 USERSET 类型参数，请参考：配置运行参数->重设参数章节中表 1 对应设置方法进行设置。

取值范围：整型，64~2147483647，单位为 KB。

默认值：4096

设置建议：

依据查询特点和并发来确定，一旦 work_mem 限定的物理内存不够，算子运算数据将写入临时表空间，带来 5-10 倍的性能下降，查询响应时间从秒级下降到分钟级。

- 对于串行无并发的复杂查询场景，平均每个查询有 5-10 关联操作，建议 work_mem=50%内存/10。

- 对于串行无并发的简单查询场景，平均每个查询有 2-5 个关联操作，建议 work_mem=50%内存/5。
- 对于并发场景，建议 work_mem=串行下的 work_mem/物理并发数。
- 对于 BitmapScan 的哈希表也会受到 work_mem 的限制，但不会被严格管控下盘。完全 Lossify 的情况下，哈希表每占用 1MB 的内存，对应一次 BitmapHeapScan 的 16GB 的页面（Ustore 为 32GB），达到 work_mem 上限后，会按此比例随数据访问量线性增长。

query_mem

参数说明：设置执行作业所使用的内存。

该参数属于 USERSET 类型参数，请参考：配置运行参数->重设参数章节中表 1 对应设置方法进行设置。

取值范围：0，或大于 32M 的整型，默认单位为 KB。

默认值：0

- 如果设置的 query_mem 值大于 0，在生成执行计划时，优化器会将作业的估算内存调整为该值。
- 如果设置值为负数或小于 32MB，将设置为默认值 0，此时优化器不会根据该值调整作业的估算内存。

query_max_mem

参数说明：设置执行作业所能够使用的最大内存。

该参数属于 USERSET 类型参数，请参考：配置运行参数->重设参数章节中表 1 对应设置方法进行设置。

取值范围：0，或大于 32M 的整型，默认单位为 KB。

默认值：0

【须知】

- 如果设置的 query_max_mem 值大于 0，当作业执行时所使用内存超过该值时，将报错退出。
- 如果设置值为负数或小于 32M，将设置为默认值 0，此时不会根据该值限制作业的内存使用。

maintenance_work_mem

参数说明：设置在维护性操作（比如 VACUUM、CREATE INDEX、ALTER TABLE ADD FOREIGN KEY 等）中可使用的最大的内存。该参数的设置会影响 VACUUM、VACUUM FULL、CLUSTER、CREATE INDEX 的执行效率。

该参数属于 USERSET 类型参数，请参考：配置运行参数->重设参数章节中表 1 对应设置方法进行设置。

取值范围：整型，1024~INT_MAX，单位为 KB。

默认值：16MB

设置建议：

- 建议设置此参数的值大于 work_mem 可以改进清理和恢复数据库转储的速度。因为在一个数据库会话里，任意时刻只有一个维护性操作可以执行，并且在执行维护性操作时不会有太多的会话。
- 当自动清理线程(参考：GUC 参数说明->自动清理章节)运行时，autovacuum_max_workers 倍数的内存将会被分配，所以此时设置 maintenance_work_mem 的值应该不小于 work_mem。
- 如果进行大数据量的 cluster 等，可以在 session 中调大该值。

psort_work_mem

参数说明：设置列存表在进行局部排序中，在开始写入临时磁盘文件之前使用的内存大小。带 partial cluster key 的表、带索引的表插入、创建表索引、删除表和更新表都会用到。

该参数属于 USERSET 类型参数，请参考：配置运行参数->重设参数章节中表 1 对应设置方法进行设置。

【须知】
同样，多个正在运行的会话可能会同时进行表的局部排序操作。因此，使用的总内存可能是 psort_work_mem 的几倍。

取值范围：整型 64~2147483647，单位为 KB。

默认值：512MB

max_loaded_cudesc

参数说明：设置列存表在做扫描时，每列缓存 cudesc 信息的个数。增大设置会提高查询性能，但也会增加内存占用，特别是当列存表的列非常多时。

该参数属于 USERSET 类型参数，请参考：配置运行参数->重设参数章节中表 1 对应设置方法进行设置。

【须知】
max_loaded_cudesc 设置过高时，有可能引起内存分配不足。

取值范围：100~INT_MAX/2048。

默认值：1024

max_stack_depth

参数说明：设置 PanWeiDB 执行堆栈的最大安全深度。需要这个安全界限是因为在服务器里，并非所有程序都检查了堆栈深度，只是在可能递规的过程，比如表达式计算这样的过程里面才进行检查。

该参数属于 SUSET 类型参数，请参考：配置运行参数->重设参数章节中表 1 对应设置方法进行设置。

取值范围：整型，100~INT_MAX，单位为 KB。

默认值：2MB

设置原则：

- 数据库需要预留 640KB 堆栈深度，因此，此参数的最佳设置是=操作系统内核允许的最大值（就是 ulimit -s 的设置）- 640KB。
- 如果设置此参数的值大于实际的内核限制，则一个正在运行的递归函数可能会导致一个独立的服务器线程崩溃。在 PanWeiDB 能够检测内核限制的操作系统上，将自动限制设置为一个不安全的值。
- 因为并非所有的操作都能够检测，所以建议用户在此设置一个明确的值。
- 默认值 2MB，这个值相对比较小，不容易导致系统崩溃。

cstore_buffers

参数说明：设置列存所使用的共享缓冲区的大小。

该参数属于 POSTMASTER 类型参数，请参考：配置运行参数->重设参数章节中表 1 对应设置方法进行设置。

取值范围：整型，16384～1073741823，单位为 KB。

默认值：128MB

设置建议：

列存表使用 cstore_buffers 设置的共享缓冲区，几乎不用 shared_buffers。因此在列存表为主的场景中，应减少 shared_buffers，增加 cstore_buffers。

bulk_read_ring_size

参数说明：大批量数据查询时（例如大表扫描），该操作使用的环形缓冲区大小。

该参数属于 USERSET 类型参数，请参考：配置运行参数->重设参数章节中表 1 对应设置方法进行设置。

取值范围：整型，256~2147483647，单位为 KB。

默认值：16MB

enable_early_free

参数说明：控制是否可以实现算子内存的提前释放。

该参数属于 USERSET 类型参数，请参考：配置运行参数->重设参数章节中表 1 对应设置方法进行设置。

取值范围：布尔型

- on 表示支持算子内存提前释放。
- off 表示不支持算子内存提前释放。

默认值：on

local_syscache_threshold

参数说明：系统表 cache 在单个 session 缓存的大小。

该参数属于 PG_SIGHUP 类型参数，请参考：配置运行参数->重设参数章节中表 1 对应设置方法进行设置。

如果 enable_global_plancache 已打开，为保证 GPC 生效，local_syscache_threshold 设置值小于 16MB 时不会生效，最小为 16MB。

如果 enable_global_plancache 和 enable_thread_pool 打开，该参数描述的是当前线程和绑定到当前线程上的 session 缓存的总大小。

取值范围：整型，1*1024*~*512*1024，单位 KB。

默认值：256MB

resilience_memory_reject_percent

参数说明： 用于控制内存过载逃生的动态内存占用百分比。该参数仅在 GUC 参数 use_workload_manager 和 enable_memory_limit 打开时生效。

该参数属于 SIGHUP 类型参数，请参考重设参数表 1 中对应设置方法进行设置。

取值范围： 字符串，长度大于 0

该参数分为 recover_memory_percent,overload_memory_percent 两部分，这两个部分的具体含义如下：

- recover_memory_percent：内存从过载状态恢复正常状态的动态内存使用占最大动态内存的百分比，当动态内存使用小于最大动态内存乘以该值对应的百分比后，停止过载逃生并放开新连接接入，取值为 0~100，设置为多少表示百分之多少。
- overload_memory_percent：内存过载时动态内存使用占最大动态内存的百分比，当动态内存使用大于最大动态内存乘以该值对应的百分比后，表示当前内存已经过载，触发过载逃生 kill 会话并禁止新连接接入，取值为 0~100，设置为多少表示百分之多少。

默认值： '0,0'，表示关闭内存过载逃生功能。

示例：

```
resilience_memory_reject_percent = '70,90'
```

上述设置表示内存使用超过最大内存上限的 90%后禁止新连接接入并 kill 堆积的会话，kill 会话过程中内存恢复到最大内存的 70%以下时停止 kill 会话并允许新连接接入。

【说明】

- 最大动态内存和已使用的动态内存可以通过 gs_total_memory_detail 视图查询获得，最大动态内存：max_dynamic_memory，已使用的动态内存：dynamic_used_memory。
- 该参数如果设置的百分比过小，则会频繁触发内存过载逃生流程，会使正在执行的会话被强制退出，新连接短时间接入失败，需要根据实际内存使用情况慎重设置。

12.4.2.磁盘空间

介绍与磁盘空间相关的参数，用于限制临时文件所占用的磁盘空间。

sql_use_spacelimit

参数说明：限制单个 SQL 在单个数据库节点上、触发落盘操作时、落盘文件的空间大小、管控的空间包括普通表、临时表以及中间结果集落盘占用的空间，对初始用户不生效。

该参数属于 USERSET 类型参数，请参考：配置运行参数->重设参数章节中表 1 对应设置方法进行设置。

取值范围：整型，-1~2147483647，单位为 KB。其中-1 表示没有限制。

默认值：-1

temp_file_limit

参数说明：限制一个会话中，触发下盘操作时，下盘文件占用的空间大小。例如一次会话中，排序和哈希表使用的临时文件，或者游标占用的临时文件。

此设置为会话级别的下盘文件控制。

该参数属于 SUSET 类型参数，请参考：配置运行参数->重设参数章节中表 1 对应设置方法进行设置。

【须知】
SQL 查询执行时使用的临时表空间不在此限制。

取值范围：整型，-1~2147483647，单位为 KB。其中-1 表示没有限制。

默认值：-1

12.4.3.内核资源使用

介绍与操作系统内核相关的参数，这些参数是否生效依赖于操作系统的设置。

max_files_per_process

参数说明：设置每个服务器进程允许同时打开的最大文件数目。如果操作系统内核强制一个合理的数目，则不需要设置。

但是在一些平台上（特别是大多数 BSD 系统），内核允许独立进程打开比系统真正可以支持的数目大得多的文件数。如果用户发现有的“Too many open files”这样的失败现象，请尝试缩小这个设置。通常情况下需要满足，系统 FD（file descriptor）数量>=最大并发数*数据库节点个数* max_files_per_process*3。

该参数属于 POSTMASTER 类型参数，请参考：配置运行参数->重设参数章节中表 1 对应设置方法进行设置。

取值范围：整型，25~2147483647。

默认值：1000

shared_preload_libraries

参数说明：此参数用于声明一个或者多个在服务器启动的时候预先装载的共享库，多个库名称之间用逗号分隔。比如 '$libdir/mylib' 会在加载标准库目录中的库文件之前预先加载 mylib.so（某些平台上可能是 mylib.sl）库文件。

可以用这个方法预先装载 PanWeiDB 的存储过程库，通常是使用 '$libdir/plXXX' 语法。XXX 只能是 pgsql、perl、tcl、python 之一。

通过预先装载一个共享库并在需要的时候初始化它，可以避免第一次使用这个库的加载时间。但是启动每个服务器进程的时间可能会增加，即使进程从来没有使用过这些库。因此建议对那些将被大多数会话使用的库才使用这个选项。

该参数属于 POSTMASTER 类型参数，请参考：配置运行参数->重设参数章节中表 1 对应设置方法进行设置。

【须知】

❖ 如果被声明的库不存在，PanWeiDB 服务将会启动失败。

❖ 每一个支持 PanWeiDB 的库都有一个特殊的标记用于保证兼容性。因此，不支持 PanWeiDB 的库不能用这种方法加载。

取值范围：字符串

默认值：空

12.4.4.基于开销的清理延迟

这个特性的目的是允许管理员减少 VACUUM 和 ANALYZE 语句在并发活动的数据库上的 I/O 影响。比如，像 VACUUM 和 ANALYZE 这样的维护语句并不需要迅速完成，并且不希望他们严重干扰系统执行其他的数据库操作。基于开销的清理延迟为管理员提供了一个实现这个目的手段。

【须知】

有些清理操作会持有关键的锁，这些操作应该尽快结束并释放锁。所以 PanWeiDB 的机制是，在这类操作过程中，基于开销的清理延迟不会发生作用。为了避免在这种情况下的长延时，实际的开销限制取下面两者之间的较大值：

❖ vacuum_cost_delay*accumulated_balance/vacuum_cost_limit

❖ vacuum_cost_delay*4

背景信息

在：SQL 语法参考->SQL 语法->ANALYZE|ANALYSE 章节和 SQL 语法参考->SQL 语法->VACUUM 章节，语句执行过程中，系统维护一个内部的记数器，跟踪所执行的各种 I/O 操作的近似开销。如果积累的开销达到了 vacuum_cost_limit 声明的限制，则执行这个操作的进程将睡眠 vacuum_cost_delay 指定的时间。然后它会重置记数器然后继续执行。

这个特性是缺省关闭的。如需开启，需要把 vacuum_cost_delay 变量设置为一个非零值。

vacuum_cost_delay

参数说明：指定开销超过 vacuum_cost_limit 的值时，进程睡眠的时间。

要注意在许多系统上，睡眠的有效分辨率是 10 毫秒。因此把 vacuum_cost_delay 设置为一个不是 10 的整数倍的数值与将它设置为下一个 10 的整数倍作用相同。

此参数一般设置较小，常见的设置是 10 或 20 毫秒。调整此特性资源占用率时，最好是调整其他参数，而不是此参数。

该参数属于 USERSET 类型参数，请参考：配置运行参数->重设参数章节中表 1 对应设置方法进行设置。

取值范围：整型，0~100，正数值表示打开基于开销的清理延迟特性；0 表示关闭基于开销的清理延迟特性。

默认值：0

vacuum_cost_page_hit

参数说明：清理一个在共享缓存里找到的缓冲区的预计开销。它代表锁住缓冲池、查找共享的 Hash 表、扫描页面内容的开销。

该参数属于 USERSET 类型参数，请参考：配置运行参数->重设参数章节中表 1 对应设置方法进行设置。

取值范围：整型，0~10000。

默认值：1

vacuum_cost_page_miss

参数说明：清理一个要从磁盘上读取的缓冲区的预计开销。它代表锁住缓冲池、查找共享 Hash 表、从磁盘读取需要的数据块、扫描它的内容的开销。

该参数属于 USERSET 类型参数，请参考：配置运行参数->重设参数章节中表 1 对应设置方法进行设置。

取值范围：整型，0~10000。

默认值：10

vacuum_cost_page_dirty

参数说明：清理修改一个原先是干净的块的预计开销。它代表把一个脏的磁盘块再次刷新到磁盘上的额外开销。

该参数属于 USERSET 类型参数，请参考：配置运行参数->重设参数章节中表 1 对应设置方法进行设置。

取值范围：整型，0~10000

默认值：20

vacuum_cost_limit

参数说明：设置清理进程休眠的开销限制。

该参数属于 USERSET 类型参数，为固定参数，用户无法修改此参数，只能查看。

取值范围：整型，1~10000。

默认值：10000

12.4.5.后端写进程

介绍后端写（background writer）进程的参数配置。后端写进程的功能就是把共享缓冲区中的脏数据（指共享缓冲区中新增或者修改的内容）写入到磁盘。目的是让数据库进程在进行用户查询时可以很少或者几乎不等待写动作的发生（写动作由后端写进程完成）。

此机制同样也减少了检查点造成的性能下降。后端写进程将持续的把脏页面刷新到磁盘上，所以在检查点到来的时候，只有几个页面需要刷新到磁盘上。但是这样还是增加了 I/O 的总净负荷，因为以前的检查点间隔里，一个重复弄脏的页面可能只会冲刷一次，而同一个间隔里，后端写进程可能会写好几次。

在大多数情况下，连续的低负荷要比周期性的尖峰负荷好，但是在本节讨论的参数可以用于按实际需要调节其行为。

bgwriter_delay

参数说明：设置后端写进程写“脏”共享缓冲区之间的时间间隔。每一次，后端写进程都会为一些脏的缓冲区发出写操作（用 bgwriter_lru_maxpages 参数控制每次写的量），然后休眠 bgwriter_delay 毫秒后才再次启动。

在许多系统上，休眠延时的有效分辨率是 10 毫秒。因此，设置一个不是 10 的倍数的数值与把它设置为下一个 10 的倍数是一样的效果。

该参数属于 SIGHUP 类型参数，请参考：配置运行参数->重设参数章节中表 1 对应设置方法进行设置。

取值范围：整型，10~10000，单位为毫秒。

默认值：2s

设置建议：在数据写压力比较大的场景中可以尝试减小该值以降低 checkpoint 的压力。

candidate_buf_percent_target

参数说明：设置用于增量检查点打开时，候选 buffer 链中可用 buffer 数目占据 shared_buffer 内存缓冲区百分比的期望值，当前候选链中的数目少于目标值时，bgwriter 线程会启动将满足条件的脏页刷盘。

该参数属于 SIGHUP 类型参数，请参考：配置运行参数->重设参数章节中表 1 对应设置方法进行设置。

取值范围：双精度浮点类型，0.1 ~ 0.85

默认值：0.3

bgwriter_lru_maxpages

参数说明：设置后端写进程每次可写入磁盘的“脏”缓存区的个数。

该参数属于 SIGHUP 类型参数，请参考：配置运行参数->重设参数章节中表 1 对应设置方法进行设置。

取值范围：整型，0～1000

【说明】
此参数设置为 0 表示禁用后端写功能，禁用后端写功能不会对 checkpoints 产生影响。

默认值：100

bgwriter_lru_multiplier

参数说明：通过与已使用缓存区数目的乘积评估下次服务器需要的缓存区数目。

写“脏”缓存区到磁盘的数目取决于服务器最近几次使用的缓存区数目。最近的 buffers 数目的平均值乘以 bgwriter_lru_multiplier 是为了评估下次服务器进程需要的 buffers 数目。在有足够多的干净的、可用的缓存区之前，后端写进程会一直写“脏”缓存区的（每次写的缓存区数目不会超过 bgwriter_lru_maxpages 的值）。

设置 bgwriter_lru_multiplier 的值为 1.0 表示一种“实时”策略，其作用是精准预测下次写“脏”缓冲区的数目。设置为较大的值可以应对突然的需求高峰，而较小的值则可以让服务器进程执行更多的写操作。

设置较小的 bgwriter_lru_maxpages 和 bgwriter_lru_multiplier 会减小后端写进程导致的额外 I/O 开销，但是服务器进程必须自己发出写操作，增加了对查询的响应时间。

该参数属于 SIGHUP 类型参数，请参考：配置运行参数->重设参数章节中表 1 对应设置方法进行设置。

取值范围：浮点型，0~10。

默认值：2

pagewriter_thread_num

参数说明：设置用于增量检查点打开后后台刷页的线程数，主要是按照脏页置脏的顺序刷盘，用于推进 recovery 点。

该参数属于 POSTMASTER 类型参数，请参考：配置运行参数->重设参数章节中表 1 对应设置方法进行设置对应设置方法进行设置。

取值范围：整型，1～16

默认值：4

dirty_page_percent_max

参数说明：设置用于增量检查点打开后脏页数量占 shared_buffers 的百分比。达到这个设定值时，后台刷页线程将以设置的 max_io_capacity 计算出的最大值刷脏页。

该参数属于 SIGHUP 类型参数，请参考：配置运行参数->重设参数章节中表 1 对应设置方法进行设置。

取值范围：浮点型，0.1～1

默认值：0.9

pagewriter_sleep

参数说明：设置用于增量检查点打开后，pagewrite 线程每隔 pagewriter_sleep 的时间刷一批脏页下盘。当脏页占据 shared_buffers 的比例达到 dirty_page_percent_max 时，每批页面数量以设定的 max_io_capacity 计算出的值刷页，其余情况每批页面数量按比例相对减少。

该参数属于 SIGHUP 类型参数，请参考：配置运行参数->重设参数章节中表 1 对应设置方法进行设置。

取值范围：整型，0～3600000（毫秒）

默认值：5000ms

max_io_capacity

参数说明：设置后端写进程批量刷页每秒的 IO 上限，需要根据具体业务场景和机器磁盘 IO 能力进行设置。要求 RTO 很短时间或者数据量比共享内存大多倍的情况，业务访问数据量又是随机访问时，该值不宜过小。该参数设置较小会减小后端写进程刷页个数，如果业务触发页面淘汰多时，该值设置小会影响业务。

该参数属于 SIGHUP 类型参数，请参考：配置运行参数->重设参数章节中表 1 对应设置方法进行设置。

取值范围：整型，30720~10485760。单位是 KB。

默认值：512000KB（500MB）

max_logical_replication_workers

参数说明： 订阅端 apply worker 线程的最大数量。

该参数属于 POSTMASTER 类型参数，请参考重设参数表 1 中对应设置方法进行设置。

取值范围： 0 ~ 262143

默认值： 4

max_sync_workers_per_subscription

参数说明： 订阅端每个订阅的 tablesync worker 线程的最大数量。

该参数属于 SIGHUP 类型参数，请参考重设参数表 1 中对应设置方法进行设置。

取值范围： 0 ~ 262143

默认值： 2

enable_consider_usecount

参数说明：设置 backend 线程在页面置换时是否考虑页面热度，建议大容量场景下开启此参数。

该参数属于 SIGHUP 类型参数，请参考：配置运行参数->重设参数章节中表 1 对应设置方法进行设置。

取值范围：布尔型

- on/true 表示考虑页面热度。
- off/false 表示不考虑页面热度。

默认值：off

dw_file_num

参数说明：设置批量双写文件的数量，该值与 pagewriter_thread_num 有关，不会大于 pagwriter_thread_num，如果设置过大，内部会纠正为 pagewriter_thread_num。

该参数属于 POSTMASTER 类型参数，请参考：配置运行参数->重设参数章节中表 1 对应设置方法进行设置。

取值范围：整型，1～16

默认值：1

dw_file_size

参数说明：设置每个批量双写文件的大小。

该参数属于 POSTMASTER 类型参数，请参考：配置运行参数->重设参数章节中表 1 对应设置方法进行设置。

取值范围：整型，32～256

默认值：256

12.4.6.异步 IO

enable_adio_debug

参数说明：允许维护人员输出一些与 ADIO 相关的日志，便于定位 ADIO 相关问题。开发人员专用，不建议普通用户使用。

该参数属于 SUSET 类型参数，为固定参数，用户无法修改此参数，只能查看。

取值范围：布尔型

- on/true 表示开启此日志开关。
- off/false 表示关闭此日志开关。

默认值：off

【说明】
当前版本暂不支持打开该开关，即使用户手动设置为打开，系统内部也会自动设置为关闭状态。

enable_adio_function

参数说明：是否开起 ADIO 功能。

该参数属于 INTERNAL 类型参数，为固定参数，用户无法修改此参数，只能查看。

【说明】
当前版本暂不支持开启异步 IO 功能，默认该功能关闭。

取值范围：布尔型

- on/true 表示开启此功能。
- off/false 表示关闭此功能。

默认值：off

enable_fast_allocate

参数说明：磁盘空间快速分配开关。

该参数属于 SUSET 类型参数，请参考：配置运行参数->重设参数章节中表 1 对应设置方法进行设置。只有在 XFS 文件系统上才能开启该开关。

取值范围：布尔型

- on/true 表示开启此功能。
- off/false 表示关闭此功能。

默认值：off

prefetch_quantity

参数说明：描述行存储使用 ADIO 预读取 IO 量的大小。

该参数属于 USERSET 类型参数，请参考：配置运行参数->重设参数章节中表 1 对应设置方法进行设置。

取值范围：整型，128～131072，单位为 8KB。

默认值：32MB (4096 * 8KB)

backwrite_quantity

参数说明：描述行存储使用 ADIO 写入 IO 量的大小。

该参数属于 USERSET 类型参数，请参考：配置运行参数->重设参数章节中表 1 对应设置方法进行设置。

取值范围：整型，128～131072，单位为 8KB。

默认值：8MB (1024 * 8KB)

cstore_prefetch_quantity

参数说明：描述列存储使用 ADIO 预取 IO 量的大小。

该参数属于 USERSET 类型参数，请参考：配置运行参数->重设参数章节中表 1 对应设置方法进行设置。

取值范围：整型，1024～1048576，单位为 KB。

默认值：32MB

cstore_backwrite_quantity

参数说明：描述列存储使用 ADIO 写入 IO 量的大小。

该参数属于 USERSET 类型参数，请参考：配置运行参数->重设参数章节中表 1 对应设置方法进行设置。

取值范围：整型，1024～1048576，单位为 KB。

默认值：8MB

cstore_backwrite_max_threshold

参数说明：描述列存储使用 ADIO 写入数据库可缓存最大的 IO 量。

该参数属于 USERSET 类型参数，请参考：配置运行参数->重设参数章节中表 1 对应设置方法进行设置。

取值范围：整型，4096～INT_MAX/2，单位为 KB。

默认值：2GB

fast_extend_file_size

参数说明：描述列存储使用 ADIO 预扩展磁盘的大小。

该参数属于 SUSET 类型参数，请参考：配置运行参数->重设参数章节中表 1 对应设置方法进行设置。

取值范围：整型，1024～1048576，单位为 KB。

默认值：8MB

effective_io_concurrency

参数说明：磁盘子系统可以同时有效处理的请求数。对于 RAID 阵列，此参数应该是阵列中驱动器主轴的数量。

该参数属于 USERSET 类型参数，请参考：配置运行参数->重设参数章节中表 1 对应设置方法进行设置。

取值范围：整型，0~1000

默认值：1

checkpoint_flush_after

参数说明：设置 checkpointer 线程刷页个数超过设定的阈值时，告知操作系统开始将操作系统缓存中的页面异步刷盘。PanWeiDB 中，磁盘页大小为 8KB。

该参数属于 SIGHUP 类型参数，请参考：配置运行参数->重设参数章节中表 1 对应设置方法进行设置。

取值范围：整型，0~256（0 表示关闭异步刷盘功能）。例如，取值 32，表示 checkpointer 线程连续写 32 个磁盘页，即 32*8=256KB 磁盘空间后会进行异步刷盘。

默认值：32

bgwriter_flush_after

参数说明：设置 background writer 线程刷页个数超过设定的阈值时，告知操作系统开始将操作系统缓存中的页面异步刷盘。PanWeiDB 中，磁盘页大小为 8KB。

该参数属于 SIGHUP 类型参数，请参考：配置运行参数->重设参数章节中表 1 对应设置方法进行设置。

取值范围：整型，0~256（0 表示关闭异步刷盘功能），单位页面（8KB）。例如，取值 64，表示 background writer 线程连续写 64 个磁盘页，即 64*8=512KB 磁盘空间后会进行异步刷盘。

默认值：512KB（即 64 个页面）

backend_flush_after

参数说明：设置 backend 线程刷页个数超过设定的阈值时，告知操作系统开始将操作系统缓存中的页面异步刷盘。PanWeiDB 中，磁盘页大小为 8KB。

该参数属于 USERSET 类型参数，请参考：配置运行参数->重设参数章节中表 1 对应设置方法进行设置。

取值范围：整型，0~256（0 表示关闭异步刷盘功能），单位页面（8KB）。例如，取值 64，表示 backend 线程连续写 64 个磁盘页，即 64*8=512KB 磁盘空间后会进行异步刷盘。

默认值：0

12.5.并行导入

PanWeiDB 提供了并行导入功能，以快速、高效地完成大量数据导入。介绍 PanWeiDB 并行导入的相关参数。

raise_errors_if_no_files

参数说明：导入时是否区分“导入文件记录数为空”和“导入文件不存在”。raise_errors_if_no_files=TRUE，则“导入文件不存在”的时候，PanWeiDB 将抛出“文件不存在的”错误。

该参数属于 SUSET 类型参数，请参考：配置运行参数->重设参数章节中表 1 对应设置方法进行设置。

取值范围：布尔型

- on 表示导入时区分“导入文件记录数为空”和“导入文件不存在”。
- off 表示导入时不区分“导入文件记录数为空”和“导入文件不存在”。

默认值：off

partition_mem_batch

参数说明：为了优化对列存分区表的批量插入，在批量插入过程中会对数据进行缓存后再批量写盘。通过 partition_mem_batch 可指定缓存个数。该值设置过大，将消耗较多系统内存资源；设置过小，将降低系统列存分区表批量插入性能。

该参数属于 USERSET 类型参数，请参考：配置运行参数->重设参数章节中表 1 对应设置方法进行设置。

取值范围：1~ 65535

默认值：256

partition_max_cache_size

参数说明：为了优化对列存分区表的批量插入，在批量插入过程中会对数据进行缓存后再批量写盘。通过 partition_max_cache_size 可指定数据缓存区大小。该值设置过大，将消耗较多系统内存资源；设置过小，将降低列存分区表批量插入性能。

该参数属于 USERSET 类型参数，请参考：配置运行参数->重设参数章节中表 1 对应设置方法进行设置。

取值范围：列存分区表：4096~ INT_MAX / 2，最小单位为 KB。

默认值：2GB

enable_delta_store

参数说明：为了增强列存单条数据导入的性能和解决磁盘冗余问题，可通过此参数选择是否开启支持列存 delta 表功能。该参数开启时，数据导入列存表，会根据表定义时指定的 DELTAROW_THRESHOLD 决定数据进入 delta 表存

储还是主表 CU 存储，当数据量小于 DELTAROW_THRESHOLD 时，数据进入 delta 表。该参数影响的操作包括 insert，copy，vacuum，vacuum full，vacuum deltamerge 重分布等所有涉及列存表数据移动的操作。

该参数属于 POSTMASTER 类型参数，请参考：配置运行参数->重设参数章节中表 1 对应设置方法进行设置。

取值范围：布尔型

- on 表示开启列存 delta 表功能。
- off 表示不开启列存 delta 表功能。

默认值：off

12.6.预写式日志

12.6.1.设置

wal_level

参数说明：设置写入 WAL 信息量的级别。

该参数属于 POSTMASTER 类型参数，请参考：配置运行参数->重设参数章节中表 1 对应设置方法进行设置。

【须知】

- 如果需要启用 WAL 日志归档和主备机的数据流复制，必须将此参数设置为 archive 或者 hot_standby。
- 如果此参数设置为 minimal, archive_mode 必须设置为 off, hot_standby 必须设置为 off，max_wal_senders 参数设置为 0, 且需为单机环境，否则将导致数据库无法启动。
- 如果此参数设置为 archive，hot_standby 必须设置为 off，否则将导致数据库无法启动。但是，hot_standby 在双机环境中不能设置为 off，具体参见 hot_standby 参数说明。

取值范围：枚举类型

- minimal

- 优点：一些重要操作（包括创建表、创建索引、簇操作和表的复制）都能安全的跳过，这样就可以使操作变得更快。
- 缺点：WAL 仅提供从数据库服务器崩溃或者紧急关闭状态恢复时所需要的基本信息，无法用 WAL 归档日志恢复数据。

❖ archive

这个参数增加了 WAL 归档需要的日志信息，从而可以支持数据库的归档恢复。

❖ hot_standby

- 这个参数进一步增加了在备机上运行的 SQL 查询的信息，这个参数只能在数据库服务重新启动后生效。
- 为了在备机上开启只读查询，wal_level 必须在主机上设置成 hot_standby，并且备机必须打开 hot_standby 参数。hot_standby 和 archive 级别之间的性能只有微小的差异，如果它们的设置对产品的性能影响有明显差异，欢迎反馈。

❖ logical

这个参数表示 WAL 日志支持逻辑复制。

默认值：hot_standby

fsync

参数说明：设置 PanWeiDB 服务器是否使用 fsync()系统函数（请参见 wal_sync_method）确保数据的更新及时写入物理磁盘中。

该参数属于 SIGHUP 类型参数，请参考：配置运行参数->重设参数章节中表 1 对应设置方法进行设置。

【须知】

❖ 使用 fsync()系统函数可以保证在操作系统或者硬件崩溃的情况下将数据恢复到一个已知的状态。

❖ 如果将此参数关闭，可能会在系统崩溃时无法恢复原来的数据，导致数据库不可用。

取值范围：布尔型

- on 表示使用 fsync()系统函数。
- off 表示不使用 fsync()系统函数。

默认值：on

synchronous_commit

参数说明：设置当前事务的同步方式。开启后支持从库在主库 commit 后可直接读取已提交的数据。

该参数属于 USERSET 类型参数，请参考：配置运行参数->重设参数章节中表 1 对应设置方法进行设置。

通常情况下，一个事务产生的日志的同步顺序如下：

1、主机将日志内容写入本地内存。

2、主机将本地内存中的日志写入本地文件系统。

3、主机将本地文件系统中的日志内容刷盘。

4、主机将日志内容发送给备机。

5、备机接受到日志内容，存入备机内存。

6、备机将备机内存中的日志写入备机文件系统。

7、备机将备机文件系统中的日志内容刷盘。

8、备机回放日志，完成对数据文件的增量更新。

取值范围：枚举类型

- on：表示主机事务提交需要等待备机将对应日志刷新到磁盘。
- off：表示主机事务提交无需等待主机自身将对应日志刷新到磁盘，通常也称为异步提交。
- local：表示主机事务提交需要等待主机自身将对应日志刷新到磁盘，通常也称为本地提交。

- remote_write：表示主机事务提交需要等待备机将对应日志写到文件系统（无需刷新到磁盘）。
- remote_receive：表示主机事务提交需要等待备机接收到对应日志数据（无需写入文件系统）。
- remote_apply：表示主机事务提交需要等待备机完成对应日志的回放操作。
- true：同 on。
- false：同 off。
- yes：同 on。
- no：同 off。
- 1：同 on。
- 0：同 off。
- 2：同 remote_apply。

【说明】

synchronous_commit 为 off 或 local，并不等价于同步状态为 async，而是与 synchronous_standby_names 的值有关，当 synchronous_standby_names 为空时同步状态为 async。

默认值：on

wal_sync_method

参数说明：设置向磁盘强制更新 WAL 数据的方法。

该参数属于 SIGHUP 类型参数，请参考：配置运行参数->重设参数章节中表 1 对应设置方法进行设置。

【须知】

如果将 fsync 关闭，这个参数的设置就没有意义，因为所有数据更新都不会强制写入磁盘。

取值范围：枚举类型

- open_datasync 表示用带 O_DSYNC 选项的 open()打开“WAL”文件。

- fdatasync 表示每次提交的时候都调用 fdatasync()。（支持 suse10 和 suse11）。
- fsync_writethrough 表示每次提交的时候调用 fsync()强制把缓冲区任何数据写入磁盘。
- 由于历史原因，我们允许在 Windows 平台上将 wal_sync_method 设置为 fsync_writethrough，尽管它和 fsync 等效。
- fsync 表示每次提交的时候调用 fsync()。（支持 suse10 和 suse11）
- open_sync 表示用带 O_SYNC 选项的 open()写“WAL”文件。（支持 suse10 和 suse11）

默认值：fdatasync

full_page_writes

参数说明：设置 PanWeiDB 服务器在检查点之后对页面的第一次修改时，是否将每个磁盘页面的全部内容写到 WAL 日志中。当增量检查点开关和 enable_double_write 同时打开时，则不使用 full_page_writes。

该参数属于 SIGHUP 类型参数，请参考：配置运行参数->重设参数章节中表 1 对应设置方法进行设置。

取值范围：布尔型

- on 表示启用此特性。
- off 表示关闭此特性。

默认值：off

【须知】

不支持将 full_page_writes 设置为 on。

wal_log_hints

参数说明：设置在检查点之后对页面的第一次修改为页面上元组 hint bits 的修改时，是否将整个页面的全部内容写到 WAL 日志中。不推荐用户修改此设置。

该参数属于 POSTMASTER 类型参数，请参考：配置运行参数->重设参数章节中表 1 对应设置方法进行设置。

取值范围：布尔型

- on 表示整个页面全部内容写到 WAL 日志中。
- off 表示整个页面内容不会写到 WAL 日志中。

默认值：on

wal_buffers

参数说明：设置用于存放 WAL 数据的共享内存空间的 XLOG_BLCKSZ 数，XLOG_BLCKSZ 的大小默认为 8KB。

该参数属于 POSTMASTER 类型参数，请参考：配置运行参数->重设参数章节中表 1 对应设置方法进行设置。

取值范围：-1~218

- 如果设置为-1，表示 wal_buffers 的大小随着参数 shared_buffers 自动调整，为 shared_buffers 的 1/32，最小值为 8 个 XLOG_BLCKSZ，最大值为 2048 个 XLOG_BLCKSZ。
- 如果设置为其他值，当小于 4 时，会被默认设置为 4。

默认值：16MB

设置建议：每次事务提交时，WAL 缓冲区的内容都写入到磁盘中，因此设置为很大的值不会带来明显的性能提升。如果将它设置成几百兆，就可以在有很多即时事务提交的服务器上提高写入磁盘的性能。根据经验来说，默认值可以满足大多数的情况。

wal_writer_delay

参数说明：WalWriter 进程的写间隔时间。

该参数属于 SIGHUP 类型参数，请参考：配置运行参数->重设参数章节中表 1 对应设置方法进行设置。

须知：如果时间过长可能造成 WAL 缓冲区的内存不足，时间过短会引起 WAL 不断写入，增加磁盘 I/O 负担。

取值范围：整型， 1～10000（毫秒）

默认值：200ms

commit_delay

参数说明：表示一个已经提交的数据在 WAL 缓冲区中存放的时间。

该参数属于 USERSET 类型参数，请参考：配置运行参数->重设参数章节中表 1 对应设置方法进行设置。

须知：

- 设置为非 0 值时事务执行 commit 后不会立即写入 WAL 中，而仍存放在 WAL 缓冲区中，等待 WalWriter 进程周期性写入磁盘。
- 如果系统负载很高，在延迟时间内，其他事务可能已经准备好提交。但如果没有事务准备提交，这个延迟就是在浪费时间。

取值范围：整型， 0～100000（微秒），其中 0 表示无延迟。

默认值：0

commit_siblings

参数说明：当一个事务发出提交请求时，如果数据库中正在执行的事务数量大于此参数的值，则该事务将等待一段时间（commit_delay 的值），否则该事务则直接写入 WAL。

该参数属于 USERSET 类型参数，请参考：配置运行参数->重设参数章节中表 1 对应设置方法进行设置。

取值范围：整型， 0～1000

默认值：5

wal_block_size

参数说明：说明 WAL 日志段文件中日志页面的大小。

该参数属于 INTERNAL 类型参数，为固定参数，用户无法修改此参数，只能查看。

取值范围：整型，单位为 Byte。

默认值：8192

wal_segment_size

参数说明：说明 WAL 日志段文件的大小。

该参数属于 INTERNAL 类型参数，为固定参数，用户无法修改此参数，只能查看。

取值范围：整型，单位为 8KB。

默认值：16MB (2048 * 8KB)

walwriter_cpu_bind

参数说明：绑定到 WAL 写入线程的 CPU 核，与 thread pool 配合使用。

该参数属于 POSTMASTER 类型参数，请参考：配置运行参数->重设参数章节中表 1 对应设置方法进行设置。

取值范围：整型，-1~核数减 1。

默认值：-1

walwriter_sleep_threshold

参数说明：xlogflusher 进入 sleep 之前空闲 xlog 刷新的次数，达到阈值会休眠。

该参数属于 SIGHUP 类型参数，请参考：配置运行参数->重设参数章节中表 1 对应设置方法进行设置。

取值范围：整型，1~50000。

默认值：500

wal_file_init_num

参数说明：WAL 编写器将创建的 xlog 段文件的数量。

该参数属于 POSTMASTER 类型参数，请参考：配置运行参数->重设参数章节中表 1 对应设置方法进行设置。

取值范围：整型，0~1000000。

默认值：10

force_promote

参考说明：备机强切功能开关。

备机强切在集群故障状态下，以丢失部分数据为代价换取集群尽可能快的恢复服务；是集群状态为不可用时的一个逃生方法，不建议频繁触发。如果操作者不清楚备机强切后丢失数据对业务的影响，请勿使用本功能。

取值范围：整型，0 或 1

0 表示关闭，1 表示开启

默认值：0

wal_flush_timeout

参数说明：Xlog 刷盘自适应控制的刷盘 IO 遍历 WalInsertStatusTable 等待的最大时间，即遍历 WalInsertStatusTable 的超时时间。

该参数属于 SIGHUP 类型参数，请参考：配置运行参数->重设参数章节中表 1 对应设置方法进行设置。

须知：

- 如果时间过长可能造成 Xlog 刷盘频率降低，降低 Xlog 处理性能。
- 遍历 WalInsertStatusTable 是数据库内部的实现，所以通过合理设置 wal_flush_timeout 的值来控制 IO 频率比较困难。

取值范围：整型， 0～90000000（微秒）

默认值：2（微秒）

wal_flush_delay

参数说明：遍历 WalInsertStatusTable 时，遇到 WAL_NOT_COPIED 状态 entry 时等待的时间间隔。

该参数属于 SIGHUP 类型参数，请参考：配置运行参数->重设参数章节中表 1 对应设置方法进行设置。

须知：参数 wal_flush_delay 值建议不要配置过大，原因如下：

- wal_flush_delay 和 wal_flush_timeout 两个参数综合作用影响事务的提交。
- 一旦触发 WALWriter 休眠逻辑后，因为事务提交的等待时长为该参数配置的值，则当 wal_flush_delay 值设置过大时等待时长将过长。

取值范围：整型， 0～90000000（微秒）

默认值：1（微秒）

pw_wal_directory

参数说明：xlog 日志的存放路径。

xlog 的默认存放路径在$PGDATA/pg_xlog 下，如有更改路径的需求，可以修改 postgresql.conf 中的参数 pw_wal_directory。新的存放路径应是存在的且可访问的。

默认值：$PGDATA/pg_xlog

autocommit

参数说明：设置事务是否自动提交

该参数属于 USERSET 类型参数，为固定参数，用户无法修改此参数，只能查看.

取值范围：布尔型

- on 表示开启事务自动提交
- off 表示关闭事务自动提交

默认值: on

xlog_lock_file_path

参数说明： 双数据库实例共享存储场景下，xlog 日志共享盘抢占的锁文件的路径。本参数在数据库系统初始化时由 OM 进行配置，不建议用户自行修改。

该参数属于 POSTMASTER 类型参数，请参考重设参数表 1 中对应设置方法进行设置。

取值范围： 字符串

默认值： NULL

xlog_file_path

参数说明： 双数据库实例共享存储场景下，xlog 日志共享盘的路径。本参数在数据库系统初始化时由 OM 进行配置，不建议用户自行修改。

该参数属于 POSTMASTER 类型参数，请参考重设参数表 1 中对应设置方法进行设置。

取值范围： 字符串

默认值： NULL

xlog_file_size

参数说明： 双数据库实例共享存储场景下，xlog 日志共享盘的大小。本参数在数据库系统初始化时由 OM 进行配置，不建议用户自行修改。

该参数属于 POSTMASTER 类型参数，请参考重设参数表 1 中对应设置方法进行设置。

取值范围： 长整型，134217728~576460752303423487，单位是字节

默认值： 549755813888

12.6.2.检查点

checkpoint_segments

参数说明：设置 checkpoint_timeout 周期内所保留的最少 WAL 日志段文件数量。每个日志文件大小为 16MB（资源池化模式下为 1GB）。

该参数属于 SIGHUP 类型参数，请参考：配置运行参数->重设参数章节中表 1 对应设置方法进行设置。

取值范围：整型，1~2147483646

提升此参数可加快大数据的导入速度，但需要结合 checkpoint_timeout、shared_buffers 这两个参数统一考虑。这个参数同时影响 WAL 日志段文件复用数量，通常情况下 pg_xlog 文件夹下最大的复用文件个数为 2 倍的 checkpoint_segments 个，复用的文件被改名为后续即将使用的 WAL 日志段文件，不会被真正删除。

默认值：64

checkpoint_timeout

参数说明：设置自动 WAL 检查点之间的最长时间。

该参数属于 SIGHUP 类型参数，请参考：配置运行参数->重设参数章节中表 1 对应设置方法进行设置。

取值范围：整型，30～3600（秒）

在提升 checkpoint_segments 以加快大数据导入的场景也需将此参数调大，同时这两个参数提升会加大 shared_buffers 的负担，需要综合考虑。

默认值：15min

checkpoint_completion_target

参数说明：指定检查点完成的目标。

该参数属于 SIGHUP 类型参数，请参考：配置运行参数->重设参数章节中表 1 对应设置方法进行设置。

取值范围：双精度浮点类型，0.0～1.0

默认值：0.5

【说明】
默认值 0.5 表示每个 checkpoint 需要在 checkpoints 间隔时间的 50%内完成。

checkpoint_warning

参数说明：如果由于填充检查点段文件导致检查点发生的时间间隔接近这个参数表示的秒数，就向服务器日志发送一个建议增加 checkpoint_segments 值的消息。

该参数属于 SIGHUP 类型参数，请参考：配置运行参数->重设参数章节中表 1 对应设置方法进行设置。

取值范围：整型，1~INT_MAX，单位秒。

默认值：5min

推荐值：5min

checkpoint_wait_timeout

参数说明：设置请求检查点等待 checkpointer 线程启动的最长时间。

该参数属于 SIGHUP 类型参数，请参考：配置运行参数->重设参数章节中表 1 对应设置方法进行设置。

取值范围：整型，2～3600（秒）

默认值：1min

enable_incremental_checkpoint

参数说明：增量检查点开关。默认 checkpoint 为增量 checkpoint，手动执行 checkpoint 时，会选择当前最后一条 WAL 日志的 LSN 为 checkpoint.redo 位点，等待 pagewritetr 线程执行刷脏和 fsync 完成后，更新 pg_control 文件。

【注意】
PanWeiDB 不支持全量检查点，因此不允许关闭增量检查点，即该参数不支持设置为 OFF。

该参数属于 POSTMASTER 类型参数，为固定参数，用户无法修改此参数，只能查看。

取值范围：布尔型

默认值：on

enable_double_write

参数说明：双写开关。当增量检查点开关打开时，同时 enable_double_write 打开，则使用 enable_double_write 双写特性保护，不再使用 full_page_writes 防止半页写问题。

该参数属于 POSTMASTER 类型参数，请参考：配置运行参数->重设参数章节中表 1 对应设置方法进行设置。

取值范围：布尔型

默认值：on

incremental_checkpoint_timeout

参数说明：增量检查点开关打开之后，设置自动 WAL 检查点之间的最长时间。

该参数属于 SIGHUP 类型参数，请参考：配置运行参数->重设参数章节中表 1 对应设置方法进行设置。

取值范围：整型，1~3600（秒）

默认值：1min

enable_xlog_prune

参数说明：设置在任一备机断联时，主机是否根据 xlog 日志的大小超过参数 max_size_for_xlog_prune 的值而回收日志。

该参数属于 SIGHUP 类型参数，请参考：配置运行参数->重设参数章节中表 1 对应设置方法进行设置。

取值范围：布尔型

- 设置为 on 时，如果任一备机断联时，主机回收日志。

❖ 设置为 off 时，如果任一备机断联时，主机不回收日志。

默认值：off

max_redo_log_size

参数说明：备 DN 表示当前回放的最新检查点位置和当前日志回放位置之间日志量的期望值，主 DN 表示恢复点到当前最新日志之间日志量的期望值，关注 RTO 的情况下，这个值建议不宜过大。

该参数属于 SIGHUP 类型参数，请参考：配置运行参数->重设参数章节中表 1 对应设置方法进行设置。

取值范围：整型，163840～2147483647，单位为 KB。

默认值：1GB

max_size_for_xlog_prune

参数说明：在 enable_xlog_prune、synchronous_commit 都打开时生效，如果有备机断连且 xlog 日志大小大于此阈值，则回收日志。所有备机断联且无逻辑复制槽时，不回收日志。

该参数属于 SIGHUP 类型参数，请参考：配置运行参数->重设参数章节中表 1 对应设置方法进行设置。

取值范围：整型，0～2147483647，单位为 KB

默认值：2147483647，单位 KB

12.6.3.日志回放

recovery_time_target

参数说明：设置 recovery_time_target 秒能够让备机完成日志写入和回放。

该参数属于 SIGHUP 类型参数，请参考：配置运行参数->重设参数章节中表 1 对应设置方法进行设置。

取值范围：整型，0~3600（秒）

- 0 是指不开启日志流控。
- 1~3600 是指备机能够在 recovery_time_target 时间内完成日志的写入和回放，可以保证主机与备机切换时能够在 recovery_time_target 秒完成日志写入和回放，保证备机能够快速升主机。

默认值：0

【须知】

- recovery_time_target 设置时间过小会影响主机的性能，设置过大会失去流控效果。
- 另外，由于极致 RTO 自带流控，所以同时开启极致 RTO 与流控时会以极致 RTO 优先，在运行期间使流控不生效。

recovery_max_workers

参数说明：设置最大并行回放线程个数。

该参数属于 POSTMASTER 类型参数，请参考：配置运行参数->重设参数章节中表 1 对应设置方法进行设置。

取值范围：整型，0~20

默认值：1（安装工具默认设置为 4，以获得更好的性能）

recovery_parallelism

参数说明：查询实际回放线程个数，该参数为只读参数，无法修改。

该参数属于 POSTMASTER 类型参数，受 recovery_max_workers 以及 recovery_parse_workers 参数影响，任意一值大于 1 时，recover_parallelism 将被重新计算。

取值范围：整型，1~2147483647

默认值：1

enable_page_lsn_check

参数说明：数据页 lsn 检查开关。回放时，检查数据页当前的 lsn 是否是期望的 lsn。

该参数属于 POSTMASTER 类型参数，请参考：配置运行参数->重设参数章节中表 1 对应设置方法进行设置。

取值范围：布尔型

默认值：on

【说明】
该参数是保留参数，无实际用途。

recovery_min_apply_delay

参数说明：设置备节点回放的延迟时间。

该参数属于 SIGHUP 类型参数，请参考：配置运行参数->重设参数章节中表 1 对应设置方法进行设置。

【须知】
- 此参数主节点设置无效，必须设置在需要延迟的备节点上，推荐设置在异步备上，设置了延时的异步备如果升主 RTO 时间会比较长。
- 延迟时间是根据主服务器上事务提交的时间戳与备机上的当前时间来计算，因此需要保证主备系统时钟一致。
- 延迟时间设置过长时，可能会导致该备机 XLOG 文件所在的磁盘满，需要平衡考虑磁盘大小来设置延迟时间。
- 没有事务的操作不会被延迟。
- 主备切换之后，原主机若需延迟，需要再手动配置此参数。
- 当 synchronous_commit 被设置为 remote_apply 时，同步复制会受到这个延时的影响，每一个 COMMIT 都需要等待备机回放结束后才会返回。

因此，当主机的 synchronous_commit 被设置为 remote_apply 时，禁止设置该参数为非 0 值，否则主机的业务会被严重阻塞。

- 使用这个特性也会让 hot_standby_feedback 被延迟，这可能导致主服务器的膨胀，两者一起使用时要小心。
- 主机执行了持有 AccessExclusive 锁的 DDL 操作，比如 DROP 和 TRUNCATE 操作，在备机延迟回放该条记录期间，在备机上对该操作对象执行查询操作会等待锁释放之后才会返回。
- 不支持 MOT 表。

取值范围：整型，0~INT_MAX，单位为毫秒。

默认值：0（不增加延迟）

redo_bind_cpu_attr

参数说明：用于控制回放线程的绑核操作，仅 sysadmin 用户可以访问。该参数属于 POSTMASTER 类型参数，请参考：配置运行参数->重设参数章节中表 1 对应设置方法进行设置。

取值范围：字符串，长度大于 0，该参数不区分大小写。

- 'nobind'：线程不做绑核。
- 'nodebind: 1, 2'：利用 NUMA 组 1,2 中的 CPU core 进行绑核。
- 'cpubind: 0-30'：利用 0-30 号 CPU core 进行绑核。

默认值：'nobind'

enable_time_report

参数说明：用于控制回放时是否记录回放统计信息。

该参数属于 POSTMASTER 类型参数，请参考重设参数表 1 中对应设置方法进行设置。

取值范围：布尔型

- on：记录回访统计信息。

❖ off：不记录回访统计信息。

默认值：off

enable_batch_dispatch

参数说明：表级并行回放时，由 startup 线程读取 wal 日志并决定每一个 wal 记录的实际 redo 线程，然后再将读取到的 wal 记录分发给具体的 redo 线程。此参数用于控制表级并行回放时，是否启用批量分发的功能，即允许积攒一定数量的 wal 记录，再分发给各个 redo 线程。

该参数属于 POSTMASTER 类型参数，请参考重设参数表 1 中对应设置方法进行设置。

取值范围：布尔型

❖ on：表级并行回放时，启用批量分发的功能。
❖ off：表级并行回放时，不启用批量分发的功能。

默认值：off

parallel_recovery_batch

参数说明：参照 enable_batch_dispatch 参数对批量分发功能的描述，此参数控制表级并行回放时 startup 线程暂存 wal 记录的数量。

该参数属于 SIGHUP 类型参数，请参考重设参数表 1 中对应设置方法进行设置。

取值范围：整型，1~100000，意义为 wal 记录的个数

默认值：1000

parallel_recovery_timeout

参数说明：在表级别并行回放时，如果当在一段时间内没有积攒够 parallel_recovery_batch 数量，则应该立即分发当前已经暂存的 wal 记录。此参数用于控制 wal 分发的 timeout 时间。

该参数属于 SIGHUP 类型参数，请参考重设参数表 1 中对应设置方法进行设置。

取值范围：整型，1~1000，单位为毫秒

默认值：300

12.6.4.归档

archive_mode

参数说明：表示是否进行归档操作。

该参数属于 SIGHUP 类型参数，请参考：配置运行参数->重设参数章节中表 1 对应设置方法进行设置。

【须知】
当 wal_level 设置成 minimal 时，archive_mode 参数无法使用。

取值范围：布尔型

- on 表示进行归档。
- off 表示不进行归档。

默认值：off

archive_command

参数说明：由管理员设置的用于归档 WAL 日志的命令，建议归档路径为绝对路径。

该参数属于 SIGHUP 类型参数，请参考：配置运行参数->重设参数章节中表 1 对应设置方法进行设置。

【须知】

- 当 archive_dest 和 archive_command 同时配置时，WAL 日志优先保存到 archive_dest 所设置的目录中，archive_command 配置的命令不生效。
- 字符串中任何%p 都被要归档的文件的绝对路径代替，而任何%f 都只被该文件名代替（相对路径都相对于数据目录的）。如果需要在命令里嵌入%字符就必须双写%。
- 这个命令当且仅当成功的时候才返回零。示例如下：

```
archive_command = 'cp --remove-destination %p /mnt/server/archivedir/%f'
```

--remove-destination 选项作用为：拷贝前如果目标文件已存在，会先删除已存在的目标文件，然后执行拷贝操作。

- 如果归档命令有多条，则需将其写入 SHELL 脚本文件中，然后将 archive_command 配置为执行该脚本的命令。示例如下：

```
--假设多条命令如下。
test ! -f dir/%f && cp %p dir/%f
--则 test.sh 脚本内容如下。
test ! -f dir/$2 && cp $1 dir/$2
--归档命令如下。
archive_command='sh dir/test.sh %p %f'
```

取值范围：字符串

默认值：(disabled)

archive_dest

参数说明：由管理员设置的用于归档 WAL 日志的目录，建议归档路径为绝对路径。

该参数属于 SIGHUP 类型参数，请参考：配置运行参数->重设参数章节中表 1 对应设置方法进行设置。

【须知】

- 当 archive_dest 和 archive_command 同时配置时，WAL 日志优先保存到 archive_dest 所设置的目录中，archive_command 配置的命令不生效。
- 字符串中如果是相对路径为相对于数据目录的。示例如下。

```
archive_dest = '/mnt/server/archivedir/'
```

取值范围：字符串

默认值：空字符串

archive_timeout

参数说明：表示归档周期。

该参数属于 SIGHUP 类型参数，请参考：配置运行参数->重设参数章节中表 1 对应设置方法进行设置。

【须知】

- 超过该参数设定的时间时强制切换 WAL 段。
- 由于强制切换而提早关闭的归档文件仍然与完整的归档文件长度相同。因此，将 archive_timeout 设为很小的值将导致占用巨大的归档存储空间，建议将 archive_timeout 设置为 60 秒。

取值范围：整型，0 ~ INT_MAX，单位为秒。其中 0 表示禁用该功能。

默认值：0

time_to_target_rpo

参数说明： 双数据库实例异地灾备模式下，设置主数据库实例发生异常发生时到已归档到 OBS 的恢复点所允许的 time_to_target_rpo 秒。

该参数属于 SIGHUP 类型参数，请参考重设参数表 1 中对应设置方法进行设置。

取值范围：字符串

- 双数据库实例异地灾备模式下，主数据库实例日志将被归档到 OBS。
- 0 是指不开启日志流控。
- 1~3600 是指设置主数据库实例发生异常发生时到已归档到 OBS 的恢复点所允许的 time_to_target_rpo 秒，保证主数据库实例因灾难崩溃时，最多可能丢失的数据的时长在允许范围内。

time_to_target_rpo 设置时间过小会影响主机的性能，设置过大会失去流控效果。

默认值：0

max_archive_directory_size

参数说明：用于控制 archive_dest 指定的归档文件所在挂载点或文件系统的空间限制操作。若开启本参数，则将在一个归档日志落盘后检查归档目录所在挂载点或文件系统已用空间大小，判断是否进行告警或删除操作。

【注意】

- 推荐在归档目录独享设备挂载点的场景下使用本参数。
- 不建议在生产环境开启此参数。因为开启此参数可能会影响数据库异常后的数据恢复完整性。当进行恢复时，若所需归档日志已被删除，则数据无法被恢复。
- 删除过程与归档过程是串行的，因此删除过多文件的耗时会影响到归档的进度。

- 当归档目录所在挂载点或文件系统已用空间大小达到了 max_archive_directory_size 设置值的 85%时将触发告警动作，系统将告警信息写入数据库日志，警示用户尽快手动删除归档目录下的无用日志

或扩充目录空间。

- 当归档目录所在挂载点或文件系统已用空间大小达到了max_archive_directory_size 设置值的 95%时将触发删除动作，所删除文件的 LSN 不大于主库和备库的日志文件的最小 LSN，删除逻辑如下：

1、获取最近 REDO 点的 LSN 号记为 A，若 A>wal_keep_segments 则 B=A-wal_keep_segment；否则 B=1。

2、获取复制槽中需要的最小 LSN 号记作 C，若 C 为 0 则 C=1；否则 C 为原值。

3、获取全量或增量 BUILD 开始的 LSN 号记作 D，若 D 为 0 则 C=1；否则 D 为原值。

4、获取符合 enable_xlog_prune 参数处理的需要保留的最小的段文件的编号记作 E。

5、获取 ABCDE 的最小值记作 H，当 H=1 或 2 时不删除归档文件；当 H>2 时，删除文件的范围是[F,((H-2)-F)/2+F]。（其中 F 为 archive_dest 目录中归档文件的最小段文件编号）

该参数属于 POSTMASTER 类型参数，请参考重设参数表 1 中对应设置方法进行设置。

取值范围：0~当前 archive_dest 指定的目录挂载点大小，可设置的最大值为 252-1。本参数设置取值无单位时默认为 KB，且支持指定单位 KB、MB、GB。

默认值：0，表示不开启本功能。

12.7.双机复制

12.7.1.发送端服务器

max_wal_senders

参数说明：指定事务日志发送进程的并发连接最大数量。不可大于等于 max_connections，参考：GUC 参数说明->连接和认证->连接设置章节。

该参数属于 POSTMASTER 类型参数，请参考：配置运行参数->重设参数章节中表 1 对应设置方法进行设置。

【须知】
wal_level 必须设置为 archive 或者 hot_standby 以允许备机的连接。

取值范围：整型，0～1024

- 建议取值范围为 8～100。
- 只有当使用单 DN 实例无主备场景下才可以设置为 0。

默认值：单机环境默认值为 4；主备环境默认值为 8。

wal_keep_segments

参数说明：Xlog 日志文件段数量。设置“pg_xlog”目录下保留事务日志文件的最小数目，备机通过获取主机的日志进行流复制。

该参数属于 SIGHUP 类型参数，请参考：配置运行参数->重设参数章节中表 1 对应设置方法进行设置。

取值范围：整型，2～INT_MAX

默认值：16

设置建议：

- 当服务器开启日志归档或者从检查点恢复时，保留的日志文件数量可能大于 wal_keep_segments 设定的值。
- 如果此参数设置过小，则在备机请求事务日志时，此事务日志可能已经被产生的新事务日志覆盖，导致请求失败，主备关系断开。
- 当双机为异步传输时，以 COPY 方式连续导入 4G 以上数据需要增大 wal_keep_segments 配置。以 T6000 单板为例，如果导入数据量为

50G，建议调整参数为 1000。您可以在导入完成并且日志同步正常后，动态恢复此参数设置。

- ❖ 若 synchronous_commit 级别小于 LOCAL_FLUSH，重建备机时，建议调大改参数为 1000，避免重建过程中，主机日志回收导致重建失败。
- ❖ 资源池化模式下，单个 xlog 文件大小为 1G 而非 16M，建议适当缩小该数值，避免 xlog 堆积。

wal_sender_timeout

参数说明：设置本端等待事务日志接收端接收日志的最大等待时间。

该参数属于 SIGHUP 类型参数，请参考：配置运行参数->重设参数章节中表 1 对应设置方法进行设置。

【须知】

- ❖ 如果主机数据较大，重建备机数据库时需要增大此参数的值，主机数据在 500G 时，此参数的参考值为 600s。
- ❖ 此值不能大于 wal_receiver_timeout 或数据库重建时的超时参数。

取值范围：整型，0 ~ INT_MAX，单位为毫秒（ms）。

默认值：6s

max_replication_slots

参数说明：设置主机端的日志复制 slot 个数。

该参数属于 POSTMASTER 类型参数，请参考：配置运行参数->重设参数章节中表 1 对应设置方法进行设置。

取值范围：整型，0~1024（建议取值范围：8~100）

默认值：8

设置建议：

当使用双机复制、逻辑解码时，该参数值建议设为：当前物理流复制槽数+所需的逻辑复制槽数。如果实际设置值比上述建议值要小，那么可能造成这些功能不可用或异常。

- 物理流复制槽提供了一种自动化的方法来确保主节点在所有备节点或从备节点收到 xlog 之前，xlog 不会被移除。也就是说物理流复制槽用于支撑主备 HA。数据库所需要的物理流复制槽数为备节点加从备的和与主节点之间的比例。例如，假设数据库高可用方案为 1 主、1 备、1 从备，则所需物理流复制槽数为 2。假设数据库的高可用方案为 1 主 3 备，则所需物理流复制槽数为 3。
- 目前默认不支持主备从部署方式。
- 关于逻辑复制槽数，请按如下规则考虑：
  - 一个逻辑复制槽只能解码一个数据库的修改，如果需要解码多个数据库，则需要创建多个逻辑复制槽。
  - 如果需要多路逻辑复制同步给多个目标数据库，在源端数据库需要创建多个逻辑复制槽，每个逻辑复制槽对应一条逻辑复制链路。

enable_slot_log

参数说明：是否开启逻辑复制槽主备同步特性。

该参数属于 USERSET 类型参数，请参考：配置运行参数->重设参数章节中表 1 对应设置方法进行设置。

取值范围：布尔型

- on 表示开启逻辑复制槽主备同步特性。
- off 表示不开启逻辑复制槽主备同步特性。

默认值：off

max_changes_in_memory

参数说明：逻辑解码时单条事务在内存中缓存的 DML 语句数量上限。

该参数属于 POSTMASTER 类型参数，请参考：配置运行参数->重设参数章节中表 1 对应设置方法进行设置。

取值范围：整型，1~2147483647

默认值：4096

max_cached_tuplebufs

参数说明：逻辑解码时总元组信息在内存中缓存的数量上限。建议设置为 max_changes_in_memory 的两倍以上。

该参数属于 POSTMASTER 类型参数，请参考：配置运行参数->重设参数章节中表 1 对应设置方法进行设置。

取值范围：整型，1~2147483647

默认值：8192

enable_wal_shipping_compression

参数说明：在流式容灾模式下设置启动跨数据库实例日志压缩功能。

该参数属于 SIGHUP 类型参数，请参考：配置运行参数->重设参数章节中表 1 对应设置方法进行设置。

该参数仅作用于流式容灾中跨数据库实例传输的一对 walsender 与 walreceiver 中，在主数据库实例上配置。

取值范围：布尔型

- true 表示打开流式容灾跨数据库实例日志压缩
- false 表示关闭流式容灾跨数据库实例日志压缩

默认值：false

logical_decode_options_default

参数说明： 指定逻辑解码启动时未指定解码选项的全局默认值。

该参数属于 SIGHUP 类型参数，请参考重设参数表 1 中对应设置方法进行设置。

当前支持指定的逻辑解码选项包括：parallel-decode-num, parallel-queue-size, max-txn-in-memory, max-reorderbuffer-in-memory。

- parallel-decode-num：指定并行解码的 decoder 线程数量。其取值范围为 1~20 的 int 型，取 1 表示按照原有的串行逻辑进行解码，取其余值即为开启并行解码。默认值为 1。当该选项配置为 1 时，禁止配置解码格式 decode-style。
- parallel-queue-size：指定并行逻辑解码线程间进行交互的队列长度。取值范围是[2,1024]，且必须为 2 的幂数，默认值为 128。队列长度和解码过程的内存使用量正相关。
- max-txn-in-memory：指定单个事务解码中间结果缓存的内存阈值，单位为 MB，范围[0,100]，默认值为 0，表示不管控内存使用。
- max-reorderbuffer-in-memory：指定所有事务解码中间结果缓存的内存阈值，单位为 GB，范围[0,100]，默认值为 0，表示不管控内存使用。

取值范围： 通过逗号分隔的 key=value 字符串，例如：'parallel-decode-num=4,parallel-queue-size=128'。其中空字符串表示采用程序硬编码的默认值。

默认值： 空字符串

replconninfoN

参数说明：设置本端侦听和鉴权的第 N 个节点信息，其中 N 表示第几个节点，取值为 1~18。

该参数是用来复制连接信息，用于将主服务器连接到备用服务器上，或将主服务器或备用服务器连接到辅助服务器上。该参数包含如下几部分内容：

- localhost：本地服务器 IP 地址。
- localport：本地端口号，同步日志传输端口。
- localheartbeatport：集群心跳端口号。
- localservice：主备通讯端口。

- remotehost：远程服务器 IP 地址
- remoteport：远程端口号，同步日志传输端口。
- remoteheartbeatport：集群心跳端口。
- remoteservice：主备通讯端口。

例如，replconninofo1 可配置为如下内容：

```
replconninfo1='localhost=192.168.1.134 localport=26001 localheartbeatport=26005 localservice=26004 remotehost=192.168.1.135 remoteport=26001 remoteheartbeatport=26005 remoteservice=26004'
```

该参数属于 SIGHUP 类型参数，请参考：配置运行参数->重设参数章节中表 1 对应设置方法进行设置。

取值范围：字符串。其中空字符串表示没有配置第 N 个节点信息。

默认值：空字符串

cross_cluster_replconninfo1

参数说明：设置跨集群的本端侦听和鉴权的第一个节点信息。

该参数属于 SIGHUP 类型参数，请参考：配置运行参数->重设参数章节中表 1 对应设置方法进行设置。

取值范围：字符串。其中空字符串表示没有配置第一个节点信息。

默认值：空字符串

cross_cluster_replconninfo2

参数说明：设置跨集群的本端侦听和鉴权的第二个节点信息。

该参数属于 SIGHUP 类型参数，请参考：配置运行参数->重设参数章节中表 1 对应设置方法进行设置。

取值范围：字符串。其中空字符串表示没有配置第二个节点信息。

默认值：空字符串

cross_cluster_replconninfo3

参数说明：设置跨集群的本端侦听和鉴权的第三个节点信息。

该参数属于 SIGHUP 类型参数，请参考：配置运行参数->重设参数章节中表 1 对应设置方法进行设置。

取值范围：字符串。其中空字符串表示没有配置第三个节点信息。

默认值：空字符串

cross_cluster_replconninfo4

参数说明：设置跨集群的本端侦听和鉴权的第四个节点信息。

该参数属于 SIGHUP 类型参数，请参考：配置运行参数->重设参数章节中表 1 对应设置方法进行设置。

取值范围：字符串。其中空字符串表示没有配置第四个节点信息。

默认值：空字符串

cross_cluster_replconninfo5

参数说明：设置跨集群的本端侦听和鉴权的第五个节点信息。

该参数属于 SIGHUP 类型参数，请参考：配置运行参数->重设参数章节中表 1 对应设置方法进行设置。

取值范围：字符串。其中空字符串表示没有配置第五个节点信息。

默认值：空字符串

cross_cluster_replconninfo6

参数说明：设置跨集群的本端侦听和鉴权的第六个节点信息。

该参数属于 SIGHUP 类型参数，请参考：配置运行参数->重设参数章节中表 1 对应设置方法进行设置。

取值范围：字符串。其中空字符串表示没有配置第六个节点信息。

默认值：空字符串

cross_cluster_replconninfo7

参数说明：设置跨集群的本端侦听和鉴权的第七个节点信息。

该参数属于 SIGHUP 类型参数，请参考：配置运行参数->重设参数章节中表 1 对应设置方法进行设置。

取值范围：字符串。其中空字符串表示没有配置第七个节点信息。

默认值：空字符串

cross_cluster_replconninfo8

参数说明：设置跨集群的本端侦听和鉴权的第八个节点信息。

该参数属于 SIGHUP 类型参数，请参考：配置运行参数->重设参数章节中表 1 对应设置方法进行设置。

取值范围：字符串。其中空字符串表示没有配置第八个节点信息。

默认值：空字符串

available_zone

参数说明：设置本端节点所在区域信息。

该参数属于 POSTMASTER 类型参数，请参考：配置运行参数->重设参数章节中表 1 对应设置方法进行设置。

取值范围：字符串。其中空字符串表示没有配置节点信息。

默认值：空字符串

enable_availablezone

参数说明： 设置本端级联备节点能否连接跨 available_zone 的备机。

该参数属于 POSTMASTER 类型参数，请参考重设参数表 1 中对应设置方法进行设置。

取值范围：布尔型

- on：表示级联备只能连接相同 available_zone 中的备机。
- off：表示级联备可以连接不同 available_zone 中的备机。

默认值： off

max_keep_log_seg

参数说明： 流控参数，逻辑复制在 DN 本地会解析物理日志转换成逻辑日志，当未被解析的物理日志文件数量大于该参数时会触发限流。此参数为 0 表示关闭限流功能。

该参数属于 USERSET 类型参数，请参考重设参数表 1 中对应设置方法进行设置。

取值范围： 整型，0 ~ 2147483647

默认值： 0

cluster_run_mode

参数说明： 基于 dorado 同步复制双集群参数，标志主集群或备集群。

该参数属于 POSTERMASTER 类型参数，，请参考重设参数表 1 中对应设置方法进行设置。

取值范围：字符串，“cluster_primary”，“cluster_standby”

默认值： 空字符串

12.7.2.主服务器

synchronous_standby_names

参数说明：潜在同步复制的备机名称列表，每个名称用逗号分隔。

该参数属于 SIGHUP 类型参数，请参考：配置运行参数->重设参数章节中表 1 对应设置方法进行设置。

【须知】

- 当前连接的同步备机是列表中的第一个名称。如果当前同步备机失去连接，则它会立即更换下一个优先级更高的备机，并将此备机的名称放入列表中。
- 备机名称可以通过设置环境变量 PGAPPNAME 指定。

取值范围：字符串。当取值为*，表示匹配任意提供同步复制的备机名称。支持按如下格式配置：

```
ANY num_sync (standby_name [, ...]) [, ANY num_sync (standby_name [, ...])]
[FIRST] num_sync (standby_name [, ...])
standby_name [, ...]
```

【说明】

- 其中 num_sync 是事务需要等待其回复的同步复制的备机的数量，standby_name 是备机的名称，FIRST 以及 ANY 指定从所列服务器中选取同步复制的备机的策略。
- ANY N (node1,node2,...) 表示在括号内任选 N 个主机名称作为同步复制的备机名称列表。例如，ANY 1 (node1,node2) 表示在 node1 和 node2 中任选一个作为同步复制的备机名称。
- ANY N1 (node1,node2,...), ANY N2 (node3,node4,...) 表示分组潜在同步复制的备机名称列表，在第一组括号内任选 N1 个主机名称作为第一组同步复制的备机名称列表，在第二组括号内任选 N2 个主机名称作为第二组同步复制的备机名称列表。此时两个分组之间为且关系，必须两个分组均达到各自需求的同步备机数，本地事务才可以被提交。

- FIRST N (node1,node2,...)表示在括号内按出现顺序的先后作为优先级选择前 N 个主机名称作为同步复制的备机名称列表。例如，FIRST 1 (node1, node2)表示选择 node1 作为同步复制的备机名称。
- node1,node2,...和 FIRST 1 (node1,node2,...) 具有的含义相同。
- 若使用 pw_guc 工具设置该参数，需要如下设置：

```
pw_guc reload -Z datanode -D @DN_PATH@ -c "synchronous_standby_names='ANY NODE 1(dn_instanceId1, dn_instanceId2)'";
```

或者：

```
pw_guc reload -Z datanode -D @DN_PATH@ -c "synchronous_standby_names='ANY 1(AZ1, AZ2)'";
```

默认值：*

【说明】

- 备机名称列表中不可出现重复的名称，配置中 num_sync 不可大于备机列表数量。
- 多分组同步备机配置 如 ANY N1 (node1,node2,...), ANY N2 (node3,node4,...) 的时候，多个分组之间为且关系，当前仅支持多 ANY 分组。不允许使用 * 来作为模糊匹配，不允许出现配置重复的备机。

most_available_sync

参数说明：主机最大可用模式开关，当有同步备机故障且与主机断开连接时，主机事务是否不因同步备机故障而被阻塞。

该参数属于 SIGHUP 类型参数，请参考：配置运行参数->重设参数章节中表 1 对应设置方法进行设置。

取值范围：布尔型

- **on：**表示在有同步备机故障且与主机断开连接时，主机不因同步备故障而阻塞。比如有两个同步备机，一个故障，另一个正常，这个时候主机事务只会等好的这个同步备，而不被故障的同步备所阻塞；再比如走 quorum

协议时，一主三备，配置 ANY 2(node2,node3,node4)，当 node2、node4 故障，node3 正常时，主机业务同样不被阻塞。

- **off：**表示在有同步备机故障时，阻塞主机。注意: 如果在同步备机故障，又关闭了主机的最大可用模式时，可能由于主机的后台业务线程(比如 WDR 等)产生的事务所造成的阻塞，进而导致 checkpoint 相关的操作也同时等待。如果需要避免该情况，请打开最大可用或者将同步备机删除。

默认值：off

enable_save_confirmed_lsn

参数说明： 启用该参数后，主机会将每次事务操作时与当前同步备达成多数派一致性的位置持久化到磁盘上。当主机发生故障后，原主作为备机发起 build 时，检测源端（新主）是否存在相同的 confirmed LSN。如果不存在，build 失败，避免原主的数据被 build 覆盖。

该参数属于 POSTMASTER 类型参数，请参考重设参数表 1 中对应设置方法进行设置。

取值范围： 布尔型

- on: 表示启用该功能。在一主多备且配置有同步备机的场景下，在主机每次执行数据变化的事务操作时（DML/DDL），且主机等待与同步备机达成多数派一致性位置时，将与当前同步备达成多数派一致性的位置持久化到磁盘上。持久化的文件对应同步备的复制槽的状态文件。该参数打开后同时影响不指定模式的自动 build 和增量 build，当主机发生故障后，原主作为备机发起 build 时，检测源端（新主）是否存在相同的 confirmed LSN。如果不存在，build 失败，避免原主的数据被 build 覆盖。
- off: 表示不启用该功能，主机事务提交时的行为与原来保持一致。自动 build 和增量 build 的行为与原来保持一致。此时在一主多备且配置有异步备机的场景下，如果主机突然发生故障宕机，而主机此刻达成的多数派一致性位置（比如 LSN100）又没有同步到异步备机时，如果强行将异步备机作为新主机启动，且在新主上执行一些事务操作，那么新主上的数据会覆盖 LSN100，此时再将原主作为备机发起 build，主机上会丢失自己最近一次达成多数派一致性位置 LSN100 的业务数据。

默认值： off

【须知】

- 如果最大可用模式 most_available_sync 配置为 on，且所有同步备机都故障时，该功能不生效。因为没有同步备可以触发该 LSN 的持久化。
- 该功能只会影响增量 build 或不指定 build 模式的自动 build, 如果用户强制指定全量 build 模式，该功能不生效。
- 如果在执行 build 前，主机的 pg_replslot 下的文件被人为删除或破坏，本功能不生效。
- 该功能开启后，如果主机在等待同步备机达成多数派一致性的过程中被主动停止，不会提示“该事务已在本地提交，可能未同步到远端”，避免上层业务以为数据已经达成一致。
- 该功能开启后，因为等待同步的时间会由于持久化数据而变长，带有同步备的主备集群的性能会受到影响。测试数据显示，与不开启该功能相比，性能约下降 20%。

keep_sync_window

参数说明：延迟进入最大可用模式的时间。

- 当最大可用模式 most_available_sync 配置为 on，在主备场景下，当存在同步备发生故障，导致不满足当前所配置的同步备数量(详细可参考 synchonous_standby_name 的含义)时，如果配置了 keep_sync_window 参数, 则在 keep_sync_window 设置的时间窗口内，继续保持最大保护模式，即阻塞主机的事务提交, 延缓进入最大可用模式的时间。
- 若在 keep_sync_window 超时窗口内，同步备机故障恢复，且满足当前所配置的同步备数量, 则不阻塞事务, 恢复到正常状态.
- 如果设置 keep_sync_window，推荐最小配置为 5s，以避免监控系统监控到网络不稳定的误报。

该参数属于 SIGHUP 类型参数，请参考：配置运行参数->重设参数章节中表 1 对应设置方法进行设置。

取值范围：整形，范围 0~INT_MAX，单位为秒。

- 0 表示不设置 keep_sync_window 超时时间窗口，即直接进入最大可用模式。
- 其余表示 keep_sync_window 超时时间窗口的大小。

默认值：0

【须知】

配置该参数可能会对 RPO 造成影响，若主机在所配置的超时时间窗口内发生故障，则从开始阻塞到主机故障这段时间窗口内的数据可能丢失。

enable_stream_replication

参数说明：控制主备、主从是否进行数据和日志同步。

该参数属于 SIGHUP 类型参数，请参考：配置运行参数->重设参数章节中表 1 对应设置方法进行设置。

【须知】

- 此参数属于性能测试参数，用于测试带有备机和不带备机的性能参数。关闭参数后，不能进行切换、故障等异常场景测试，否则会出现主备从不一致的情况。
- 此参数属于受控参数，不建议正常业务场景下关闭此参数。
- 当前版本默认不支持主备从部署模式。

取值范围：布尔型

- on 表示打开主备、主从同步。
- off 表示关闭主备、主从同步。

默认值：on

enable_mix_replication

参数说明：控制主备、主从之间 WAL 日志及数据复制的方式。

该参数属于 INTERNAL 类型参数，默认值为 off，不允许外部修改。

【须知】

- 此参数目前不允许正常业务场景下改变其值，即关闭 WAL 日志、数据页混合复制模式。
- 当前版本默认不支持主备从部署模式。

取值范围：布尔型

- on 表示打开 WAL 日志、数据页混合复制模式。
- off 表示关闭 WAL 日志、数据页混合复制模式。

默认值：off

vacuum_defer_cleanup_age

参数说明：指定 VACUUM 使用的事务数，VACUUM 会延迟清除无效的行存表记录，延迟的事务个数通过 vacuum_defer_cleanup_age 进行设置。即 VACUUM 和 VACUUM FULL 操作不会立即清理刚刚被删除元组。

该参数属于 SIGHUP 类型参数，请参考：配置运行参数->重设参数章节中表 1 对应设置方法进行设置。

取值范围：整型，0～1000000，值为 0 表示不延迟。

默认值：0

data_replicate_buffer_size

参数说明：发送端与接收端传递数据页时，队列占用内存的大小。此参数会影响主备之间复制的缓冲大小。

该参数属于 POSTMASTER 类型参数，请参考：配置运行参数->重设参数章节中表 1 对应设置方法进行设置。

取值范围：整型，4096~1072693248，单位为 KB。

默认值：16MB（即 16384KB）

walsender_max_send_size

参数说明：设置主机端日志或数据发送缓冲区的大小。

该参数属于 POSTMASTER 类型参数，请参考：配置运行参数->重设参数章节中表 1 对应设置方法进行设置。

取值范围：整型，8~INT_MAX，单位为 KB。

默认值：8M（即 8192KB）

enable_data_replicate

参数说明：当数据库在数据导入行存表时，主机与备机的数据同步方式可以进行选择。

该参数属于 USERSET 类型参数，请参考：配置运行参数->重设参数章节中表 1 对应设置方法进行设置。

取值范围：布尔型

- on 表示导入数据行存表时主备数据采用数据页的方式进行同步。当 replication_type 参数为 1 时，不允许设置为 on，如果此时用 guc 工具设置成 on，会强制改为 off。
- off 表示导入数据行存表时主备数据采用日志（Xlog）方式进行同步。

默认值：off

ha_module_debug

参数说明：用于查看数据复制时具体数据块的复制状态日志。

该参数属于 USERSET 类型参数，请参考：配置运行参数->重设参数章节中表 1 对应设置方法进行设置。

取值范围：布尔型

- on 表示日志中将打印数据复制时每个数据块的状态。

❖ off 表示日志中不打印数据复制时每个数据块的状态。

默认值：off

enable_incremental_catchup

参数说明：控制主备之间数据追赶（catchup）的方式，目前默认不支持主备从部署模式。

该参数属于 SIGHUP 类型参数，请参考：配置运行参数->重设参数章节中表 1 对应设置方法进行设置。

取值范围：布尔型

❖ on 表示备机 catchup 时用增量 catchup 方式，即从从备本地数据文件扫描获得主备差异数据文件列表，进行主备之间的 catchup。
❖ off 表示备机 catchup 时用全量 catchup 方式，即从主机本地所有数据文件扫描获得主备差异数据文件列表，进行主备之间的 catchup。

默认值：on

wait_dummy_time

参数说明：同时控制增量数据追赶（catchup）时，PanWeiDB 主备从按顺序启动时等待从备启动的最长时间以及等待从备发回扫描列表的最长时间。

该参数属于 SIGHUP 类型参数，请参考：配置运行参数->重设参数章节中表 1 对应设置方法进行设置。

取值范围：整型，范围 1~INT_MAX，单位为秒

默认值：300

【说明】

❖ 单位只能设置为秒。
❖ 当前版本默认不支持主备从部署模式。

catchup2normal_wait_time

参数说明：打开最大可用模式 most_available_sync，主备场景下，控制备机数据追赶（catchup）阻塞主机的最长时间。该时间为估算值，实际结果可能与参数值有偏差。

该参数属于 SIGHUP 类型参数，请参考：配置运行参数->重设参数章节中表 1 对应设置方法进行设置。

取值范围：整型，范围-1~10000，单位为毫秒。

- -1 表示主机阻塞直到备机数据追赶完成。
- 0 表示备机数据追赶时始终不阻塞主机。
- 其余值表示备机数据追赶时阻塞主机的最长时间。例如，取值 5000，表示当备机数据追赶完成时间还剩 5s 时，阻塞主机等待其完成。

默认值：-1

sync_config_strategy

参数说明：主机和备机、备机和级联备之间配置文件的同步策略。

该参数属于 POSTMASTER 类型参数，请参考：配置运行参数->重设参数章节中表 1 对应设置方法进行设置。

取值范围：枚举类型

- all_node: 主机配置为 all_node 时，表示允许主机向所有备机主动同步配置文件；备机配置为 all_node 时，表示允许当前备机向其主机发送同步请求，允许当前备机向其所有级联备主动同步配置文件；级联备配置为 all_node 时，表示允许当前级联备向上一级备机发送同步请求。
- only_sync_node: 主机配置为 only_sync_node 时，表示仅允许主机向所有同步备机主动同步配置文件；备机配置为 only_sync_node 时，表示允许当前备机向其主机发送同步请求，不允许当前备机向其所有级联备机主动同步配置文件；级联备配置为 only_sync_node 时，表示允许当前级联备向其备机发送同步请求。

- none_node: 主机配置为 none_node 时，表示不允许主机向任何备机主动同步配置文件；备机配置为 none_node 时，表示不允许当前备机向其主机发送同步请求，不允许当前备机向其所有级联备主动同步配置文件；级联备配置为 none_node 时，表示不允许当前级联备向其备机发送同步请求。

默认值：all_node

【须知】

- 在一个包含了主机、备机和级联备的 PanWeiDB 集群中，主机相对于备机是发送端，备机相对于主机是接收端，备机相对于级联备是发送端，级联备相对于备机是接收端。
- 发送端主动向接收端同步配置文件、接收端请求发送端同步配置文件是两个独立的事件，均会使得配置文件同步。若不希望配置文件同步，则需要将集群中所有节点的 sync_config_strategy 参数配置为 none_node；若仅希望主机与同步备机同步配置文件，则需要将主机的 sync_config_strategy 参数配置为 only_sync_node，其余节点配置为 none_node；若希望所有节点同步配置文件，则需要将所有节点的 sync_config_strategy 参数配置为 all_node。目前暂不支持自定义指定任意节点间的同步策略。
- 配置参数同步的具体表现为，发送端发送配置文件，对接收端配置文件中的对应参数直接覆盖。若设置了配置文件需要同步的策略，则修改接收端配置参数后，发送端会立刻覆盖接收端的配置参数，使得接收端修改不生效。
- 即使设置了配置文件需要同步的策略，仍有部分配置参数不会被同步。它们是："application_name"、"archive_command"、"audit_directory"、"available_zone"、"comm_control_port"、"comm_sctp_port"、"listen_addresses"、"log_directory"、"port"、"replconninfoN"、"ssl"、"ssl_ca_file"、"ssl_cert_file"、"ssl_ciphers"、"ssl_crl_file"、"ssl_key_file"、"ssl_renegotiation_limit"、"ssl_cert_notify_time"、"synchronous_standby_names"、"local_bind_address"、"perf_directory"、"query_log_directory"、"asp_log_directory"、"streaming_router_port"、

"enable_upsert_to_merge"、"archive_dest"、"recovery_min_apply_delay"、"sync_config_strategy"。

hadr_recovery_time_target

参数说明：在流式容灾模式下设置 hadr_recovery_time_target 能够让备数据库实例完成日志写入和回放。

该参数属于 SIGHUP 类型参数，请参考：配置运行参数->重设参数章节中表 1 对应设置方法进行设置。

取值范围：整型，0~3600（秒）

- 0 是指不开启日志流控。
- 1~3600 是指备机能够在 hadr_recovery_time_target 时间内完成日志的写入和回放，可以保证主数据库实例与备数据库实例切换时能够在 hadr_recovery_time_target 秒完成日志写入和回放，保证备数据库实例能够快速升主。

hadr_recovery_time_target 设置时间过小会影响主机的性能，设置过大会失去流控效果。

默认值：0

hadr_recovery_point_target

参数说明：在流式容灾模式下设置 hadr_recovery_point_target 能够让备数据库实例完成日志刷盘的 RPO 时间。

该参数属于 SIGHUP 类型参数，请参考：配置运行参数->重设参数章节中表 1 对应设置方法进行设置。

取值范围：整型，0~3600（秒）

- 0 是指不开启日志流控。
- 1~3600 是指备机能够在 hadr_recovery_point_target 时间内完成日志的刷盘，可以保证主数据库实例与备数据库实例切换时日志差距能够在 hadr_recovery_point_target 秒内，保障备数据库实例升主日志量。

hadr_recovery_point_target 设置时间过小会影响主机的性能，设置过大会失去流控效果。

默认值：0

hadr_super_user_record_path

参数说明：该参数为流式异地容灾参数，表示备数据库实例中 hadr_disaster 用户的加密文件存放路径。

该参数属于 SIGHUP 类型参数，请参考：配置运行参数->重设参数章节中表 1 对应设置方法进行设置。

修改建议：由流式容灾密码传递工具自动设置，不需要用户手动添加。

取值范围：字符串

默认值：NULL

【须知】

- 在一个包含了主机、备机和级联备的数据库实例中，主机相对于备机是发送端，备机相对于主机是接收端，备机相对于级联备是发送端，级联备相对于备机是接收端。
- 发送端主动向接收端同步配置文件、接收端请求发送端同步配置文件是两个独立的事件，均会使得配置文件同步。若不希望配置文件同步，则需要在接收端配置为 none_node，发送端若为备机只能配置为 none_node，发送端若为主机，配置为 none_node 时主机与所有备机都不同步，为 only_sync_node 时仅与同步备同步，不与异步备同步。
- 配置参数同步的具体表现为，发送端发送配置文件，对接收端配置文件中的对应参数直接覆盖。若设置了配置文件需要同步的策略，则修改接收端配置参数后，发送端会立刻覆盖接收端的配置参数，使得接收端修改不生效。
- 即使设置了配置文件需要同步的策略，仍有部分配置参数不会被同步。包括：“application_name”，“archive_command”，“audit_directory”，“available_zone”，“comm_control_port”，“comm_sctp_port”，“listen_addresses”，“log_directory”，“port”，“replconninfoN”，“ssl”，

“ssl_ca_file”，“ssl_cert_file”，“ssl_ciphers”，“ssl_crl_file”，“ssl_key_file”，“ssl_renegotiation_limit”，“ssl_cert_notify_time”，“synchronous_standby_names”，“local_bind_address”，“perf_directory”，“query_log_directory”，“asp_log_directory”，“streaming_router_port”，“enable_upsert_to_merge”，“archive_dest”，“recovery_min_apply_delay”，“sync_config_strategy”。

repl_auth_mode

参数说明： 设置主备复制和备机重建的验证模式。

该参数属于 SIGHUP 类型参数，请参考重设参数表 1 中对应设置方法进行设置。

【说明】

- 如果主机上开启了 UUID 验证功能、且配置了非空字符串的 repl_uuid 验证码，那么备机也需要开启 UUID 验证功能、且配置相同的 repl_uuid 验证码，否则主备日志复制和备机重建请求将被主机拒绝。
- 该参数支持 SIGHUP 动态加载新值。修改之后，不影响已建连的主备连接，对后续主备复制请求和主备重建请求生效。
- 支持 Quorum、DCF 协议下的备机重建验证；支持 Quorum 协议下的主备复制验证；不支持 DCF 协议下的主备复制验证。
- UUID 验证功能主要为了防止主、备误连导致的数据串扰和污染，不是用于安全目的。
- 该参数不支持主、备间自动同步。

取值范围： 枚举类型

- off：表示关闭 UUID 验证功能。
- default：表示关闭 UUID 验证功能。
- uuid：表示开启 UUID 验证功能。

默认值： default

logical_sender_timeout

参数说明： 设置本端等待逻辑日志接收端接收日志的最大等待时间。

该参数属于 USERSET 类型参数，请参考重设参数表 1 中对应设置方法进行设置。

取值范围：整型， 0～2147483647，单位为毫秒（ms）。

默认值：30000

output_granularity_level

功能描述： 控制普通用户 COPY 导出数据的粒度。

该参数属于 SIGHUP 类型参数，请参考重设参数表 1 中对应设置方法进行设置。

取值范围： 整型，取值范围：为 0～3。

默认值： 0

ignore_standby_lsn_window

功能描述： 该参数为忽略 lsn 位点长时间不推进的同步备的超时窗口，表示当同步备返回的 lsn 位点在一个超时窗口未推进时，主机的事务提交不等待该备机同步。

该参数属于 SIGHUP 类型参数，请参考重设参数表 1 中对应设置方法进行设置。

取值范围： 整型，范围 0~INT_MAX，单位为毫秒。

- 0：表示不设置 ignore_standby_lsn_window 超时时间窗口，即无论同步备的 lsn 位点多久未推进，主机事务提交均等待该备机同步。
- 非零值：表示 ignore_standby_lsn_window 超时时间窗口的大小。

默认值： 0

【须知】

- 仅在 most_available_sync 配置为 on 时，该参数才生效。

❖ 注意取值的合理性，比如当 synchronous_commit=remote_apply，ignore_standby_lsn_window 配置的值小于延迟回放时间 recovery_min_apply_delay，则由于延迟回放导致备机返回的 apply 位点不推进，ignore_standby_lsn_window 生效，导致主机事务提交不等待该备机回放完成，remote_apply 失效。

ignore_feedback_xmin_window

功能描述： 该参数为打开 hot_standby_feedback 后，忽略备机长时间不推进的 xmin 的超时窗口，表示当备机反馈的 oldestXmin 在一个超时窗口未推进时，主机的 vacuum 不考虑该 xmin。

该参数属于 SIGHUP 类型参数，请参考重设参数表 1 中对应设置方法进行设置。

取值范围： 整型，范围 0~INT_MAX，单位为毫秒。

❖ 0：表示不设置 ignore_feedback_xmin_window 超时时间窗口，即无论备机的 oldestXmin 多久未推进，主机的 vacuum 都需要考虑该 xmin。
❖ 非零值：表示 ignore_feedback_xmin_window 超时时间窗口的大小。

默认值： 0

12.7.3.备服务器

hot_standby

参数说明：设置是否允许备机在恢复过程中连接和查询。

该参数属于 POSTMASTER 类型参数，请参考：配置运行参数->重设参数章节中表 1 对应设置方法进行设置。

【须知】

❖ 如果此参数设置为 on，wal_level 的级别必须设置为 host_standby 及以上。

- 如果 hot_standby 参数曾经被关闭，且 wal_level 参数曾被设置低于 hot_standby 等级，那么，再次打开 hot_standby 参数之前，为了确保主备环境下备机上待回放的日志都可以支持备机查询功能，需要进行如下操作：

  1、将主、备的 wal_level 参数调整到 hot_standby 等级或以上，并重启实例生效。

  2、在主机上执行 checkpoint 操作，并通过查询 pg_stat_get_wal_senders()系统函数，确认各个备机的 receiver_replay_location 追上主机当前的 sender_flush_location，保证 wal_level 的调整同步到备机并生效，且备机不需要再回放之前低等级的日志。

  3、将主、备的 hot_standby 参数打开（设为 on），并重启实例生效。

取值范围：布尔型

- on 表示允许备机在恢复过程中连接和查询。
- off 表示不允许备机在恢复过程中连接和查询。

默认值：on

max_standby_archive_delay

参数说明：当开启双机热备模式时，如果备机正处理归档 WAL 日志数据，这时进行查询就会产生冲突，此参数就是设置备机取消查询之前所等待的时间。

该参数属于 SIGHUP 类型参数，请参考：配置运行参数->重设参数章节中表 1 对应设置方法进行设置。

【须知】
-1 表示允许备机一直等待冲突的查询完成。

取值范围：整型，范围：-1~INT_MAX，单位为毫秒。

默认值：3s（即 3000ms）

max_standby_streaming_delay

参数说明：当开启双机热备模式时，如果备机正通过流复制接收 WAL 日志数据，这时进行查询就会产生冲突，这个参数就是设置备机取消查询之前所等待的时间。

该参数属于 SIGHUP 类型参数，请参考：配置运行参数->重设参数章节中表 1 对应设置方法进行设置。

【须知】
-1 表示允许备机一直等待冲突的查询完成。

取值范围：整型（毫秒），范围：-1~INT_MAX。

默认值：3s（即 3000ms）

wal_receiver_status_interval

参数说明：设置 WAL 日志接收进程的状态通知给主机的最大时间间隔。

该参数属于 SIGHUP 类型参数，请参考：配置运行参数->重设参数章节中表 1 对应设置方法进行设置。

取值范围：整型，范围：0 ~ INT_MAX，单位为秒。

默认值：5s（即 5000ms）

【须知】
当该参数设置为 0 时，表示关闭备机向主机反馈日志接收位置等信息，可能会导致主机事务提交阻塞、switchover 操作失败等异常现象。正常业务场景，不建议将该参数设置为 0。

hot_standby_feedback

参数说明：设置是否允许将备机上执行查询的结果反馈给主机，这可以避免查询冲突。

该参数属于 SIGHUP 类型参数，请参考：配置运行参数->重设参数章节中表 1 对应设置方法进行设置。

取值范围：布尔型

- on 表示允许将备机上执行查询的最小事务号反馈给主机。
- off 表示不允许将备机上执行查询的最小事务号反馈给主机。

默认值：off

【须知】
当该参数为 on 时，主机的旧版本数据的清理会受限于备机正在读的事务，即主机只允许清理小于备机反馈回来的事务所作的更改。
所以，若该参数开启时，会影响主机的性能。

wal_receiver_timeout

参数说明：设置从主机接收数据的最大等待时间。

该参数属于 SIGHUP 类型参数，请参考：配置运行参数->重设参数章节中表 1 对应设置方法进行设置。

取值范围：整型，0 ~ INT_MAX，单位为毫秒。

默认值：6s（即 6000ms）

wal_receiver_connect_timeout

参数说明：设置连接主机的最大等待超时时间。

该参数属于 SIGHUP 类型参数，请参考：配置运行参数->重设参数章节中表 1 对应设置方法进行设置。

取值范围：整型，0 ~ INT_MAX / 1000，单位为秒。

默认值：2s

wal_receiver_connect_retries

参数说明：设置连接主机的最大尝试次数。

该参数属于 SIGHUP 类型参数，请参考：配置运行参数->重设参数章节中表 1 对应设置方法进行设置。

取值范围：整型，1~ INT_MAX，单位为毫秒。

默认值：1

wal_receiver_buffer_size

参数说明：备机与从备接收 Xlog 存放到内存缓冲区的大小，目前默认不支持主备从部署模式。

该参数属于 POSTMASTER 类型参数，请参考：配置运行参数->重设参数章节中表 1 对应设置方法进行设置。

取值范围：整型，4096~1047552，单位为 KB。

默认值：64MB（即 65536KB）

primary_slotname

参数说明：设置备机对应主机的 slot name，用于主备校验，与 wal 日志删除机制。

该参数属于 SIGHUP 类型参数，请参考：配置运行参数->重设参数章节中表 1 对应设置方法进行设置。

取值范围：字符型

默认值：空字符串

max_logical_replication_workers

参数说明：订阅端 apply worker 线程的最大数量。

该参数属于 POSTMASTER 类型参数，请参考：配置运行参数->重设参数章节中表 1 对应设置方法进行设置。

取值范围：整型，0~262143

默认值：4

track_stmt_standby_chain_size

参数说明： 组合参数，控制备机快/慢 SQL 记录的最大占用内存与磁盘空间。以 60 秒为周期读取该参数，并执行清理超过保留时间的记录，仅 sysadmin 用户可以访问。

该参数属于 SIGHUP 类型参数，请参考重设参数表 1 中对应设置方法进行设置。

取值范围： 字符型

该参数分为四部分，形式为'fast sql memory size, fast sql disk size, slow sql memory size, slow sql disk size' 在主机上，full sql 为全量 sql，存储在一张 unlogged 表上，slow sql 为其中慢的那部分 sql。在备机中我们将非 slow 的那部分称为 fast sql，slow 与 fast 分开存放于不同位置，因此额外使用了四个值进行控制。

- fast sql memory size 为保留的快 SQL 的最大内存占用空间，取值范围为 [16, 1024]，单位为 MB。

- fast sql disk size 为保留的快 SQL 的最大磁盘占用空间，取值范围为 [512, 1048576]，单位为 MB。
- slow sql memory size 为保留的慢 SQL 的最大内存占用空间，取值范围为 [16, 1024]，单位为 MB。
- slow sql disk size 为保留的慢 SQL 的最大磁盘占用空间，取值范围为 [512, 1048576]，单位为 MB。

【须知】
- 注意其中快慢 SQL 各自对应的内存值不可大于磁盘值。
- 清理时按照每 16M 数据的粒度进行清理，因此最大会有 16M 数据量的延迟误差。

默认值： 32, 1024, 16, 512

subscription_conflict_resolution

参数说明：控制订阅端遇到主键或唯一键重复冲突时的处理方式。

该参数属于 SIGHUP 类型参数，请参考重设参数表 1 中对应设置方法进行设置。

【须知】
如果设置为 apply_remote，当同步过来的新元组存在多行因为不同索引发生冲突时，尝试应用该元组时仍会报错。

取值范围：枚举类型

- error：表示冲突时直接报错。
- apply_remote：表示冲突时应用发布端版本。
- keep_local：表示冲突时保留本地版本。

默认值：error

12.8.查询规划

介绍查询优化器方法配置、开销常量、规划算法以及一些配置参数。

【说明】

优化器中涉及的两个参数：

- INT_MAX 数据类型 INT 的最大值，其值为 2147483647。
- DBL_MAX 数据类型 FLOAT 的最大值。

12.8.1.优化器方法配置

这些配置参数提供了影响查询优化器选择查询规划的原始方法。如果优化器为特定的查询选择的缺省规划并不是最优的，可以通过使用这些配置参数强制优化器选择一个不同的规划来临时解决这个问题。更好的方法包括调节优化器开销常量、手动运行 ANALYZE、增加配置参数 default_statistics_target 的值、增加使用 ALTER TABLE SET STATISTICS 为指定列增加收集的统计信息。

enable_inner_unique_opt

参数说明：控制优化器对 Inner Unique 优化使用。

该参数属于 USERSET 类型参数，请参考重设参数表 1 中对应设置方法进行设置。

取值范围：布尔型

- on 表示使用。
- off 表示不使用。

默认值：off

enable_broadcast

参数说明：控制优化器对 stream 代价估算时对 broadcast 分布方式的使用。

该参数属于 USERSET 类型参数，请参考：配置运行参数->重设参数章节中表 1 对应设置方法进行设置。

取值范围：布尔型

- on 表示使用。
- off 表示不使用。

默认值：on

enable_bitmapscan

参数说明：控制优化器对位图扫描规划类型的使用。

该参数属于 USERSET 类型参数，请参考：配置运行参数->重设参数章节中表 1 对应设置方法进行设置。

取值范围：布尔型

- on 表示使用。
- off 表示不使用。

默认值：on

force_bitmapand

参数说明：控制优化器强制使用 bitmapand 规划类型的使用。

该参数属于 USERSET 类型参数，请参考：配置运行参数->重设参数章节中表 1 对应设置方法进行设置。

取值范围：布尔型

- on 表示使用。
- off 表示不使用。

默认值：off

enable_hashagg

参数说明：控制优化器对 Hash 聚集规划类型的使用。

该参数属于 USERSET 类型参数，请参考：配置运行参数->重设参数章节中表 1 对应设置方法进行设置。

取值范围：布尔型

- on 表示使用。
- off 表示不使用。

默认值：on

enable_hashjoin

参数说明：控制优化器对 Hash 连接规划类型的使用。

该参数属于 USERSET 类型参数，请参考：配置运行参数->重设参数章节中表 1 对应设置方法进行设置。

取值范围：布尔型

- on 表示使用。
- off 表示不使用。

默认值：on

enable_indexscan

参数说明：控制优化器对索引扫描规划类型的使用。

该参数属于 USERSET 类型参数，请参考：配置运行参数->重设参数章节中表 1 对应设置方法进行设置。

取值范围：布尔型

- on 表示使用。
- off 表示不使用。

默认值：on

enable_indexonlyscan

参数说明：控制优化器对仅索引扫描规划类型的使用。

该参数属于 USERSET 类型参数，请参考：配置运行参数->重设参数章节中表 1 对应设置方法进行设置。

取值范围：布尔型

- on 表示使用。
- off 表示不使用。

默认值：on

enable_material

参数说明：控制优化器对实体化的使用。消除整个实体化是不可能的，但是可以关闭这个变量以防止优化器插入实体节点。

该参数属于 USERSET 类型参数，请参考：配置运行参数->重设参数章节中表 1 对应设置方法进行设置。

取值范围：布尔型

- on 表示使用。
- off 表示不使用。

默认值：on

enable_mergejoin

参数说明：控制优化器对融合连接规划类型的使用。

该参数属于 USERSET 类型参数，请参考：配置运行参数->重设参数章节中表 1 对应设置方法进行设置。

取值范围：布尔型

- on 表示使用。
- off 表示不使用。

默认值：on

enable_nestloop

参数说明：控制优化器对内表全表扫描嵌套循环连接规划类型的使用。完全消除嵌套循环连接是不可能的，但是关闭这个变量就会让优化器在存在其他方法的时候优先选择其他方法。

该参数属于 USERSET 类型参数，请参考：配置运行参数->重设参数章节中表 1 对应设置方法进行设置。

取值范围：布尔型

- on 表示使用。
- off 表示不使用。

默认值：on

enable_index_nestloop

参数说明：控制优化器对内表参数化索引扫描嵌套循环连接规划类型的使用。

该参数属于 USERSET 类型参数，请参考：配置运行参数->重设参数章节中表 1 对应设置方法进行设置。

取值范围：布尔型

- on 表示使用。
- off 表示不使用。

默认值：on

enable_seqscan

参数说明：控制优化器对顺序扫描规划类型的使用。完全消除顺序扫描是不可能的，但是关闭这个变量会让优化器在存在其他方法的时候优先选择其他方法。

该参数属于 USERSET 类型参数，请参考：配置运行参数->重设参数章节中表 1 对应设置方法进行设置。

取值范围：布尔型

- on 表示使用。
- off 表示不使用。

默认值：on

enable_sort

参数说明：控制优化器使用的排序步骤。完全消除明确的排序是不可能的，但是关闭这个变量可以让优化器在存在其他方法的时候优先选择其他方法。

该参数属于 USERSET 类型参数，请参考：配置运行参数->重设参数章节中表 1 对应设置方法进行设置。

取值范围：布尔型

- on 表示使用。
- off 表示不使用。

默认值：on

enable_tidscan

参数说明：控制优化器对 TID 扫描规划类型的使用。

该参数属于 USERSET 类型参数，请参考：配置运行参数->重设参数章节中表 1 对应设置方法进行设置。

取值范围：布尔型

- on 表示使用。

❖ off 表示不使用。

默认值：on

enable_kill_query

参数说明：CASCADE 模式删除用户时，会删除此用户拥有的所有对象。此参数标识是否允许在删除用户的时候，取消锁定此用户所属对象的 query。

该参数属于 SUSET 类型参数，请参考：配置运行参数->重设参数章节中表 1 对应设置方法进行设置。

取值范围：布尔型

❖ on 表示允许取消锁定。
❖ off 表示不允许取消锁定。

默认值：off

enforce_a_behavior

参数说明：控制正则表达式的规则匹配模式。

该参数属于 USERSET 类型参数，请参考：配置运行参数->重设参数章节中表 1 对应设置方法进行设置。

取值范围：布尔型

❖ on 表示正则表达式采用 A 格式的匹配规则。
❖ off 表示正则表达式采用 POSIX 格式的匹配规则。

默认值：on

max_recursive_times

参数说明：控制 with recursive 的最大迭代次数。

该参数属于 USERSET 类型参数，请参考：配置运行参数->重设参数章节中表 1 对应设置方法进行设置。

取值范围：整型，0～INT_MAX。

默认值：1000

enable_vector_engine

参数说明：控制优化器对向量化执行引擎的使用。

该参数属于 USERSET 类型参数，请参考：配置运行参数->重设参数章节中表 1 对应设置方法进行设置。

取值范围：布尔型

- on 表示使用。
- off 表示不使用。

默认值：on

enable_change_hjcost

参数说明：控制优化器在 Hash Join 代价估算路径选择时，是否使用将内表运行时代价排除在 Hash Join 节点运行时代价外的估算方式。如果使用，则有利于选择条数少，但运行代价大的表做内表。

该参数属于 SUSET 类型参数，请参考：配置运行参数->重设参数章节中表 1 对应设置方法进行设置。

取值范围：布尔型

- on 表示使用。
- off 表示不使用。

默认值：off

enable_absolute_tablespace

参数说明：控制表空间是否可以使用绝对路径。

该参数属于 USERSET 类型参数，请参考：配置运行参数->重设参数章节中表 1 对应设置方法进行设置。

取值范围：布尔型

- on 表示可以使用绝对路径。
- off 表示不可以使用绝对路径。

默认值：on

enable_valuepartition_pruning

参数说明：控制是否对 DFS 分区表进行静态/动态优化。

该参数属于 USERSET 类型参数，请参考：配置运行参数->重设参数章节中表 1 对应设置方法进行设置。

取值范围：布尔型

- on 表示对 DFS 分区表进行静态/动态优化。
- off 表示不对 DFS 分区表进行静态/动态优化。

默认值：on

qrw_inlist2join_optmode

参数说明：控制是否使用 inlist-to-join 查询重写。

该参数属于 USERSET 类型参数，请参考：配置运行参数->重设参数章节中表 1 对应设置方法进行设置。

取值范围：字符串

- disable：关闭 inlist2join 查询重写。
- cost_base：基于代价的 inlist2join 查询重写。

- rule_base：基于规则的 inlist2join 查询重写，即强制使用 inlist2join 查询重写。
- 任意正整数：inlist2join 查询重写阈值，即 list 内元素个数大于该阈值，进行 inlist2join 查询重写。

默认值：cost_base

skew_option

参数说明：控制是否使用优化策略。

该参数属于 USERSET 类型参数，请参考：配置运行参数->重设参数章节中表 1 对应设置方法进行设置。

取值范围：字符串

- off：关闭策略。
- normal：采用激进策略。对于不确定是否出现倾斜的场景，认为存在倾斜，并进行相应优化。
- lazy：采用保守策略。对于不确定是否出现倾斜场景，认为不存在倾斜，不进行优化。

默认值：normal

default_limit_rows

参数说明：设置生成 genericplan 的缺省 limit 估算行数。此参数设置为正数时表示直接将设置的值作为优化阶段估算 limit 的行数（但如果判断出语句实际返回的行数是小于该参数值的，以实际返回的行数为准作为估算值），设置为负数时表示使用百分比的形式设置默认的估算值，负数转换为默认百分比，即-5 代表 5%。

该参数属于 USERSET 类型参数，请参考：配置运行参数->重设参数章节中表 1 对应设置方法进行设置。

取值范围：浮点型，-100 ~ DBL_MAX。

默认值：-10

check_implicit_conversions

参数说明：控制是否对查询中有隐式类型转换的索引列是否会生成候选索引路径进行检查。

该参数属于 USERSET 类型参数，请参考：配置运行参数->重设参数章节中表 1 对应设置方法进行设置。

取值范围：布尔型

- on 表示对查询中有隐式类型转换的索引列是否会生成候选索引路径进行检查。
- off 表示不进行相关检查。

默认值：off

cost_weight_index

参数说明：设置 index_scan 的代价权重。

该参数属于 USERSET 类型参数，请参考：配置运行参数->重设参数章节中表 1 对应设置方法进行设置。

取值范围：浮点型，1e-10~1e+10。

默认值：1

try_vector_engine_strategy

参数说明：设置行存表走向量化执行引擎的策略。通过设置该参数，可以使包含行存表的查询可以转换为向量化的执行计划执行计算，从而提升类 AP 场景的复杂查询的执行性能。

该参数属于 USERSET 类型参数，请参考：配置运行参数->重设参数章节中表 1 对应设置方法进行设置。

取值范围：枚举型

- off，为默认取值，表示关闭本功能，即行存表不会转换为向量的执行计划执行。
- force，表示只要查询中不包含向量化引擎不支持的类型或者表达式，则不论查询的基表为行存表、列存表，还是行列混合存储的，强制将查询转换为向量化的执行计划执行计算。在这种情况下，针对不同的查询场景可能出现性能下降。
- optimal，表示在 force 的基础上，由优化器根据查询的复杂度进行选择是否将查询语句转换为向量化的执行计划，尽可能避免转换为向量化的执行计划后出现性能下降。

默认值：off

partition_iterator_elimination

参数说明： 控制分区表在分区剪枝结果为一个分区时，是否通过消除分区迭代算子来提升执行效率。

该参数属于 USERSET 类型参数，请参考重设参数表 1 中对应设置方法进行设置。

取值范围： 布尔型

- on：表示消除分区迭代算子。
- off：表示不消除分区迭代算子。

默认值： off

partition_page_estimation

参数说明： 分区表页面是否通过剪枝结果进行页面估算优化。只包括分区表和 local 索引页面，不包括全局索引页面。估算公式为：估算后页面 = 分区表总页面数 *（剪枝后分区数/总分区数）

该参数属于 USERSET 类型参数，请参考重设参数表 1 中对应设置方法进行设置。

取值范围： 布尔型

- on：表示使用剪枝结果进行页面估算优化。
- off：表示不使用剪枝结果进行页面估算优化。

默认值： off

expr_postpone_mode

参数说明： expr_postpone_mode 用于控制是否启用 limit 与 sort 算子的延迟投影。

该参数属于 USERSET 类型参数，请参考中对应设置方法进行设置。

取值范围： 枚举型

- expr_postpone_mode 为 off 时，不进行延迟投影。
- expr_postpone_mode 为 cost_base 时，代价大于 cpu_operator_cost * 10，则进行延迟投影。
- expr_postpone_mode 为 force 时，代价大于 0 则进行延迟投影。

默认值： off

12.8.2.优化器开销常量

介绍优化器开销常量。这里描述的开销可以按照任意标准度量。只关心其相对值，因此以相同的系数缩放它们将不会对优化器的选择产生任何影响。缺省时，它们以抓取顺序页的开销为基本单位。也就是说将 seq_page_cost 设为 1.0，同时其他开销参数以它为基准设置。也可以使用其他基准，比如以毫秒计的实际执行时间。

seq_page_cost

参数说明：设置优化器计算一次顺序磁盘抓取页面的开销。

该参数属于 USERSET 类型参数，请参考：配置运行参数->重设参数章节中表 1 对应设置方法进行设置。

取值范围：浮点型，0～DBL_MAX。

默认值：1

random_page_cost

参数说明：设置优化器计算一次非顺序抓取磁盘页面的开销。

该参数属于 USERSET 类型参数，请参考：配置运行参数->重设参数章节中表 1 对应设置方法进行设置。

须知

虽然服务器允许将 random_page_cost 设置的比 seq_page_cost 小，但是物理上实际不受影响。如果所有数据库都位于随机访问内存中时，两者设置为相等很合理。因为在此种情况下，非顺序抓取页并没有副作用。同样，在缓冲率很高的数据库上，应该相对于 CPU 参数同时降低这两个值，因为获取内存中的页要比通常情况下开销小很多。

取值范围：浮点型，0～DBL_MAX。

默认值：4

说明

- 对于特别表空间中的表和索引，可以通过设置同名的表空间的参数来覆盖这个值。
- 相对于 seq_page_cost，减少这个值将导致系统更倾向于使用索引扫描，而增加这个值使得索引扫描开销比较高。可以通过同时增加或减少这两个值来调整磁盘 I/O 相对于 CPU 的开销。

cpu_tuple_cost

参数说明：设置优化器计算在一次查询中处理每一行数据的开销。

该参数属于 USERSET 类型参数，请参考：配置运行参数->重设参数章节中表 1 对应设置方法进行设置。

取值范围：浮点型，0～DBL_MAX。

默认值：0.01

cpu_index_tuple_cost

参数说明：设置优化器计算在一次索引扫描中处理每条索引的开销。

该参数属于 USERSET 类型参数，请参考：配置运行参数->重设参数章节中表 1 对应设置方法进行设置。

取值范围：浮点型，0～DBL_MAX。

默认值：0.005

cpu_operator_cost

参数说明：设置优化器计算一次查询中执行一个操作符或函数的开销。

该参数属于 USERSET 类型参数，请参考：配置运行参数->重设参数章节中表 1 对应设置方法进行设置。

取值范围：浮点型，0～DBL_MAX。

默认值：0.0025

effective_cache_size

参数说明：设置优化器在一次单一的查询中可用的磁盘缓冲区的有效大小。

设置这个参数，还要考虑 PanWeiDB 的共享缓冲区以及内核的磁盘缓冲区。另外，还要考虑预计的在不同表之间的并发查询数目，因为它们将共享可用的空间。

这个参数对 PanWeiDB 分配的共享内存大小没有影响，它也不会使用内核磁盘缓冲，它只用于估算。数值是用磁盘页来计算的，通常每个页面是 8192 字节。

该参数属于 USERSET 类型参数，请参考：配置运行参数->重设参数章节中表 1 对应设置方法进行设置。

取值范围：整型，1～INT_MAX，单位为 8KB。

比默认值高的数值可能会导致使用索引扫描，更低的数值可能会导致选择顺序扫描。

默认值：128MB

allocate_mem_cost

参数说明：设置优化器计算 Hash Join 创建 Hash 表开辟内存空间所需的开销，供 Hash join 估算不准时调优使用。

该参数属于 USERSET 类型参数，请参考：配置运行参数->重设参数章节中表 1 对应设置方法进行设置。

取值范围：浮点型，0～DBL_MAX。

默认值：0

12.8.3.基因查询优化器

介绍基因查询优化器相关的参数。基因查询优化器（GEQO）是一种启发式的查询规划算法。这个算法减少了对复杂查询规划的时间，而且生成规划的开销有时也小于正常的详尽的查询算法。

geqo

参数说明：控制基因查询优化的使用。

该参数属于 USERSET 类型参数，请参考：配置运行参数->重设参数章节中表 1 对应设置方法进行设置。

须知

通常情况下在执行过程中不要关闭，geqo_threshold 变量提供了更精细的控制 GEQO 的方法。

取值范围：布尔型

- on 表示使用。

❖ off 表示不使用。

默认值：on

geqo_threshold

参数说明：如果执行语句的数量超过设计的 FROM 的项数，则会使用基因查询优化来执行查询。

该参数属于 USERSET 类型参数，请参考：配置运行参数->重设参数章节中表 1 对应设置方法进行设置。

须知

❖ 对于简单的查询，通常用详尽搜索方法，当涉及多个表的查询的时候，用 GEQO 可以更好的管理查询。

❖ 一个 FULL OUTER JOIN 构造仅作为一个 FROM 项。

取值范围：整型，2～INT_MAX。

默认值：12

geqo_effort

参数说明：控制 GEQO 在规划时间和规划质量之间的平衡。

该参数属于 USERSET 类型参数，请参考：配置运行参数->重设参数章节中表 1 对应设置方法进行设置。

须知

geqo_effort 实际上并没有直接做任何事情，只是用于计算其他影响 GEQO 的变量的默认值。如果愿意，可以手工设置其他参数。

取值范围：整型，1～10。

须知

比默认值大的数值增加了查询规划的时间，但是也增加了选中有效查询的几率。

默认值：5

geqo_pool_size

参数说明：控制 GEQO 使用池的大小，也就是基因全体中的个体数量。

该参数属于 USERSET 类型参数，请参考：配置运行参数->重设参数章节中表 1 对应设置方法进行设置。

取值范围：整型，0～INT_MAX。

须知

至少是 2，且有用的值一般在 100 到 1000 之间。设置为 0，表示使用系统自适应方式，PanWeiDB 会基于 geqo_effort 和表的个数选取合适的值。

默认值：0

geqo_generations

参数说明：控制 GEQO 使用的算法的迭代次数。

该参数属于 USERSET 类型参数，请参考：配置运行参数->重设参数章节中表 1 对应设置方法进行设置。

取值范围：整型，0～INT_MAX。

须知

必须至少是 1，且有用的值介于 100 和 1000 之间。如果设置为 0，则基于 geqo_pool_size 选取合适的值。

默认值：0

geqo_selection_bias

参数说明：控制 GEQO 的选择性偏好，即就是一个种群中的选择性压力。

该参数属于 USERSET 类型参数，请参考：配置运行参数->重设参数章节中表 1 对应设置方法进行设置。

取值范围：浮点型，1.5 ~ 2.0。

默认值：2

geqo_seed

参数说明：控制 GEQO 使用的随机数生产器的初始化值，用来从顺序连接在一起的查询空间中查找随机路径。

该参数属于 USERSET 类型参数，请参考：配置运行参数->重设参数章节中表 1 对应设置方法进行设置。

取值范围：浮点型，0.0 ~ 1.0。

【须知】
不同的值会改变搜索的连接路径，从而影响了所找路径的优劣。

默认值：0

12.8.4.其他优化器选项

explain_dna_file

参数说明： 指定 explain_perf_mode 为 run，导出的 csv 信息的目标文件。

该参数属于 USERSET 类型参数，请参考：配置运行参数->重设参数章节中表 1 对应设置方法进行设置。

【须知】

- 这个参数的取值必须是绝对路径加上.csv 格式的文件名。
- 查询计划需为 stream 计划，才可以导出导出的 csv 信息的目标文件。

取值范围： 字符串

默认值： 空

explain_perf_mode

参数说明：此参数用来指定 explain 的显示格式。

该参数属于 USERSET 类型参数，请参考：配置运行参数->重设参数章节中表 1 对应设置方法进行设置。

取值范围：normal、pretty、summary、run

- normal：代表使用默认的打印格式。
- pretty：代表使用 PanWeiDB 改进后的新显示格式。新的格式层次清晰，计划包含了 plan node id，性能分析简单直接。
- summary：代表是在 pretty 的基础上增加了对打印信息的分析。
- run：代表在 summary 的基础上，将统计的信息输出到 csv 格式的文件中，以便于进一步分析。

默认值：normal（当前版本参数取值仅 normal 生效，若设置为非 normal，显示格式依然为 normal）。

analysis_options

参数说明：通过开启对应选项中所对应的功能选项使用相应的定位功能，包括数据校验、性能统计等，参见取值范围中的选项说明。

该参数属于 USERSET 类型参数，请参考：配置运行参数->重设参数章节中表 1 对应设置方法进行设置。设置时，选择开启或者关闭的选项请使用'on()'或

'off()'包括，未被显示指定的功能选项会维持原来的值，参考格式：'on(option1, option2, ...)'。

取值范围：字符串

- LLVM_COMPILE 表示在开启并行执行，且 explain 的显示格式设置为非 normal 的条件下，explain 显示界面中显示线程的 codegen 编译时间。
- HASH_CONFLICT 表示在数据库节点进程的 pg_log 目录中的 log 日志中显示 hash 表的统计信息，包括 hash 表大小、hash 链长、hash 冲突情况。
- STREAM_DATA_CHECK 表示对网络传输前后的数据进行 CRC 校验。

默认值：ALL,on(),off(LLVM_COMPILE、HASH_CONFLICT、STREAM_DATA_CHECK)，不开启任何定位功能。

cost_param

参数说明：该参数用于控制在特定的客户场景中，使用不同的估算方法使得估算值与真实值更接近。此参数可以同时控制多种方法，与某一方法对应的位做与操作，不为 0 表示该方法被选择。

当 cost_param & 1 不为 0，表示对于求不等值连接选择率时选择一种改良机制，此方法在自连接（两个相同的表之间连接）的估算中更加准确。目前，已弃用 cost_param & 1 不为 0 时的路径，默认选择更优的估算公式；

当 cost_param & 2 不为 0，表示求多个过滤条件（Filter）的选择率时，选择最小的作为总的选择率，而非两者乘积，此方法在过滤条件的列之间关联性较强时估算更加准确；

该参数属于 USERSET 类型参数，请参考：配置运行参数->重设参数章节中表 1 对应设置方法进行设置。

取值范围：整型，0～INT_MAX

默认值：0

enable_partitionwise

参数说明：分区表连接操作是否选择智能算法。

该参数属于 USERSET 类型参数，请参考：配置运行参数->重设参数章节中表 1 对应设置方法进行设置。

取值范围：布尔型

- on 表示选择智能算法。
- off 表示不选择智能算法。

默认值：off

var_eq_const_selectivity

参数说明：整型 const 选择率是否使用新型选择率模型进行估算

该参数属于 USERSET 类型参数，请参考重设参数表 1 中对应设置方法进行设置。

取值范围：布尔型

ON 表示使用新型选择率模型计算整型 const 的选择率：若整型不落入 MCV 且不为 NULL 值，但落入直方图，则利用直方图左右边界情况进行估算，也不落入直方图，则使用表的行数进行估算；若整型为 NULL 值或者 MCV 值，使用原逻辑计算选择率。

OFF 表示使用原有的选择率计算模型。

默认值：off

partition_page_estimation

参数说明：分区表页面是否通过剪枝结果进行页面估算优化

该参数属于 USERSET 类型参数，请参考重设参数表 1 中对应设置方法进行设置。

取值范围：布尔型

on 表示使用剪枝结果进行页面估算优化。

off 表示不使用剪枝结果进行页面估算优化。

默认值：off

partition_iterator_elimination

参数说明：分区表在分区剪枝结果为一个分区时，是否消除分区迭代算子来提升执行效率。

该参数属于 USERSET 类型参数，请参考重设参数表 1 中对应设置方法进行设置。

取值范围：布尔型

on 表示消除分区迭代算子。

off 表示不消除分区迭代算子。

默认值：off

enable_functional_dependency

参数说明：ANALYZE 生成的多列统计信息是否包含函数依赖统计信息，是否应用函数依赖统计信息计算选择率。

该参数属于 USERSET 类型参数，请参考重设参数表 1 中对应设置方法进行设置。

取值范围：布尔型

- on 包含两个功能：1. 执行 ANALYZE 生成的多列统计信息包含函数依赖统计信息。2. 计算选择率会使用函数依赖统计信息。

- off 包含两个功能：1. 执行 ANALYZE 生成的多列统计信息不包含函数依赖统计信息。2. 计算选择率不会使用函数依赖统计信息。

默认值：off

rewrite_rule

参数说明：标识开启的可选查询重写规则。有部分查询重写规则是可选的，开启它们并不能总是对查询效率有提升效果。在特定的客户场景中，通过此 GUC 参数对查询重写规则进行设置，使得查询效率最优。

此参数可以控制查询重写规则的组合，比如有多个重写规则：rule1、rule2、rule3、rule4。可以设置：

- 启用查询重写规则 rule1

```
set rewrite_rule=rule1;
```

- 启用查询重写规则 rule2 和 rule3

```
set rewrite_rule=rule2,rule3;
```

- 关闭所有可选查询重写规则

```
set rewrite_rule=none;
```

该参数属于 USERSET 类型参数，请参考：配置运行参数->重设参数章节中表 1 对应设置方法进行设置。

取值范围：字符串

- none：不使用任何可选查询重写规则。
- lazyagg：使用 Lazy Agg 查询重写规则（消除子查询中的聚集运算）。
- magicset：使用 Magic Set 查询重写规则（从主查询中下推条件到子查询）。
- partialpush：使用 Partial Push 查询重写规则。
- uniquecheck：使用 Unique Check 查询重写规则（提升目标列中无 agg 的子查询语句，在执行时检查返回行数是否为 1 行）。
- disablerep：使用 Disable Replicate 查询重写规则。
- intargetlist：使用 In Target List 查询重写规则（提升目标列中的子查

询）。

- predpushnormal：使用 Predicate Push 查询重写规则（下推谓词条件到子查询中）。
- predpushforce：使用 Predicate Push 查询重写规则（下推谓词条件到子查询中，尽可能的利用索引加速）。
- predpush：在 predpushnormal 和 predpushforce 中根据代价选择最优计划。
- disable_pullup_expr_sublink：禁止优化器将 expr_sublink 类型的子连接提升，关于 sublink 的分类和提升原理详见子查询调优。
- enable_sublink_pullup_enhanced：使用增强后的 sublink 查询重写规则，包括 where、 having 子句的非相关子链接提升和 winmagic 重写优化。
- rownumenhance：可以将 rownum < const 或 rownum <= const 的场景转为 limit 常量算子。
- enable_any_sublink_pullup_enhanced：计划器尝试将相关子连接进行提升，生成更好的执行计划。详细用法可参见《PanWeiDB V2.0-S3.0.1_B01 管理员指南》-> 性能调优 -> SQL 调优指南-> 典型 SQL 调优点->ANY 子连接性能优化章节。
- remove_redundant_diatinct_group_by：去除 ANY_sublink 子查询中多余的 distinct 和 group by 子句，以支持子查询提升。

默认值：magicset

enable_pbe_optimization

参数说明：设置优化器是否对以 PBE（Parse Bind Execute）形式执行的语句进行查询计划的优化。

该参数属于 SUSET 类型参数，请参考：配置运行参数->重设参数章节中表 1 对应设置方法进行设置。

取值范围：布尔型。

- on 表示优化器将优化 PBE 语句的查询计划。
- off 表示不使用优化。

默认值：on

enable_global_plancache

参数说明：设置是否对 PBE 查询的执行计划进行缓存共享，开启该功能可以节省高并发下数据库节点的内存使用。

在打开 enable_global_plancache 的情况下，为保证 GPC 生效，默认 local_syscache_threshold 不小于 16MB。即如当前 local_syscache_threshold 小于 16MB，则设置为 16MB，如大于 16MB，则不改变。

该参数属于 POSTMASTER 类型参数，请参考：配置运行参数->重设参数章节中表 1 对应设置方法进行设置。

取值范围：布尔型。

- on 表示对 PBE 查询的执行计划进行缓存共享。
- off 表示不共享。

默认值：off

【说明】
enable_global_plancache 不支持与 pwplsql_check 同时设置为 on，如果同事设置为 on 则出现报错提示：此参数不能和 pwplsql_check 同时设置为 on。

gpc_clean_timeout**

参数说明：开启 enable_global_plancache 的情况下，如果共享计划列表里的计划超过 gpc_clean_timeout 的时间没有被使用，则会被清理掉。本参数用于控制没有使用的共享计划的保留时间。

该参数属于 SIGHUP 类型参数，请参考：配置运行参数->重设参数章节中表 1 对应设置方法进行设置。

取值范围：整型，300～86400，单位为秒

默认值：1800，即 30min

【须知】
由于内部的 GPC 清理周期为 5 分钟，也就是 5 分钟检查清理一次，所以并非一超时就被清理，而是超时后的下一个检查周期被清理，即超时后 5 分钟内被清理。

enable_global_stats

参数说明：标识当前统计信息模式，区别采用全局统计信息收集模式还是单节点统计信息收集模式，默认创建为采用全局统计信息模式。当关闭该参数时，则默认收集 PanWeiDB 第一个节点的统计信息，此时可能会影响生成查询计划的质量，但信息收集性能较优，建议客户谨慎考虑。该参数当前版本已废弃，请勿设置。

该参数属于 SUSET 类型参数，请参考：配置运行参数->重设参数章节中表 1 对应设置方法进行设置。

取值范围：布尔型

- on/true 表示全局统计信息。
- off/false 表示数据库节点统计信息。

默认值：on

sql_beta_feature

参数说明：标识开启的可选 SQL 引擎 Beta 特性，其中包括对行数估算、查询等价估算等优化。

开启它们可以对特定的场景进行优化，但也可能会导致部分没有被测试覆盖的场景发生性能劣化。在特定的客户场景中，通过此 GUC 参数对查询重写规则进行设置，使得查询效率最优。

此参数可以控制 SQL 引擎 Beta 特性的组合，比如有多个 Beta 特性：feature1、feature2、feature3、feature4。可以设置：

```
--启用 SQL 引擎 Beta 特性 feature1。
set sql_beta_feature=feature1;

--启用 SQL 引擎 Beta 特性 feature2 和 feature3。
set sql_beta_feature=feature2,feature3;

--关闭所有可选 SQL 引擎 Beta 特性。
set sql_beta_feature=none;
```

该参数属于 USERSET 类型参数，请参考：配置运行参数->重设参数章节中表 1 对应设置方法进行设置。

取值范围：字符串

- none：不使用任何 Beta 优化器特性。
- sel_semi_poisson：使用泊松分布对等值的半连接和反连接选择率进行校准。
- sel_expr_instr: 使用字符串匹配的行数估算方法对 instr(col, 'const') > 0、= 0、 = 1 进行更准确的估算。
- param_path_gen：生成更多可能的参数化路径。
- rand_cost_opt：对小数据量表的随机读取代价进行优化。
- param_path_opt：利用表的膨胀系数优化索引 analyze 信息。
- page_est_opt：优化对非列存表索引 analyze 信息的 relpages 估算。
- no_unique_index_first：关闭主键索引扫描路径优先的优化。
- join_sel_with_cast_func：估算 join 行数的时候支持类型转换函数。
- canonical_pathkey：正则化 pathkey 生成置后（pathkey 是指标记数据有序性键值的集合）。该参数打开之后，可能会导致带 order by 等语句，在有外连接的情况下，输出数据语义和标准不一样。

- index_cost_with_leaf_pages_only：估算索引代价时考虑索引叶子结点。
- partition_opfusion：开启分区表优化。
- a_style_coerce：开启 Decode 类型转换规则兼容 O，详见 UNION CASE 和相关构造，请参考：SQL 语法参考->类型转换->UNION CASE 和相关构造章节。
- plpgsql_stream_fetchall：在存储过程中 for loop 或 cursor 上执行的 sql 走 stream 场景下，开启获取所有 tuple 结果。
- partition_fdw_on：支持基于分区表创建 postgres foreign table 下的相关 SQL。
- predpush_same_level：开启 predpush hint 控制同层参数化路径的功能。
- disable_bitmap_cost_with_lossy_pages：关闭 bitmap 路径代价中对 lossy pages 代价的计算。
- enable_plsql_smp：开启存储过程中的查询支持并行执行的功能。目前在同一时刻仅支持一条 query 使用并行执行，且 cursor 相关操作、自治事务、exception 中的查询不会生成并行执行计划。
- enable_upsert_execute_gplan：pbe 场景下 on duplicate key update 语句中 update 子句带有参数时设置 enable_upsert_execute_gplan 允许通过 gplan 执行。
- disable_merge_append_partition：对于分区表，禁止生成 Merge Append 路径。

- enable_plsql_smp：开启存储过程中的查询支持并行执行的功能。目前在同一时刻仅支持一条 query 使用并行执行，且 cursor 相关操作、自治事务、exception 中的查询不会生成并行执行计划。
- disable_bitmap_cost_with_lossy_pages：关闭 bitmap 路径代价中对 lossy pages 代价的计算。

默认值：canonical_pathkey

ngram_gram_size

参数说明：ngram 解析器分词的长度。

该参数属于 USERSET 类型参数，请参考：配置运行参数->重设参数章节中表 1 对应设置方法进行设置。

取值范围：整型，1～4

默认值：2

ngram_grapsymbol_ignore

参数说明：ngram 解析器是否忽略图形化字符。

该参数属于 USERSET 类型参数，请参考：配置运行参数->重设参数章节中表 1 对应设置方法进行设置。

取值范围：布尔型

- on 表示忽略图形化字符。
- off 表示不忽略图形化字符。

默认值：off

ngram_punctuation_ignore

参数说明：ngram 解析器是否忽略标点符号。

该参数属于 USERSET 类型参数，请参考：配置运行参数->重设参数章节中表 1 对应设置方法进行设置。

取值范围：布尔型

- on 表示忽略标点符号。
- off 表示不忽略标点符号。

默认值：on

default_statistics_target

参数说明：为没有用 ALTER TABLE SET STATISTICS 设置字段目标的表设置缺省统计目标。此参数设置为正数时，代表统计信息的样本数量；设置为负数时，代表使用百分比的形式设置统计目标，负数转换为对应的百分比，即-5 代表 5%。

该参数属于 USERSET 类型参数，请参考：配置运行参数->重设参数章节中表 1 对应设置方法进行设置。

取值范围：整型，-100～10000。

【须知】

- ❖ 比默认值大的正数数值增加了 ANALYZE 所需的时间，但是可能会改善优化器的估计质量。
- ❖ 调整此参数可能存在性能劣化的风险，如果某个查询劣化，可以考虑：

1、恢复默认的统计信息。

2、使用 plan hint 来调整到之前的查询计划。（详细参见：《PanWeiDB V2.0-S3.0.1_B01 管理员指南->性能调优->SQL 调优指南->使用 Plan Hint 进行调优》章节。

- ❖ 当此 guc 参数设置为负数时，如果计算的采样样本数大于等于总数据量的 2%，且用户表的数据量小于 1600000 时，ANALYZE 所需时间相比 guc 参数为默认值的时间会有所增加。
- ❖ 当此 guc 参数设置为负数时，则 autoanalyze 不生效。

默认值：100

constraint_exclusion

参数说明：控制查询优化器使用表约束查询的优化。

该参数属于 USERSET 类型参数，请参考：配置运行参数->重设参数章节中表 1 对应设置方法进行设置。

取值范围：枚举类型

- on 表示检查所有表的约束。
- off 表示不检查约束。
- partition 表示只检查继承的子表和 UNION ALL 子查询。

【须知】

当 constraint_exclusion 为 on，优化器用查询条件和表的 CHECK 约束比较，并且在查询条件和约束冲突的时候忽略对表的扫描。

默认值：partition

【说明】

目前， constraint_exclusion 缺省被打开，通常用来实现表分区。如果所有的表都打开它，对于简单的查询强加了额外的规划，并且对简单查询没有益处。如果不用分区表，可以关掉它。

cursor_tuple_fraction

参数说明：优化器估计游标获取行数在总行数中的占比。

该参数属于 USERSET 类型参数，请参考：配置运行参数->重设参数章节中表 1 对应设置方法进行设置。

取值范围：浮点型，0.0～1.0。

【须知】

比默认值小的值与使用“fast start”为游标规划的值相偏离，从而使得前几行恢复的很快而抓取全部的行需要很长的时间。比默认值大的值加大了总的估计的时间。在最大的值 1.0 处，像正常的查询一样规划游标，只考虑总的估计时间和传送第一行的时间。

默认值：0.1

from_collapse_limit

参数说明：根据生成的 FROM 列表的项数来判断优化器是否将把子查询合并到上层查询，如果 FROM 列表项个数小于等于该参数值，优化器会将子查询合并到上层查询。

该参数属于 USERSET 类型参数，请参考：配置运行参数->重设参数章节中表 1 对应设置方法进行设置。

取值范围：整型，1～INT_MAX。

【须知】

比默认值小的数值将降低规划时间，但是可能生成差的执行计划。

默认值：8

join_collapse_limit

参数说明：根据得出的列表项数来判断优化器是否执行把除 FULL JOINS 之外的 JOIN 构造重写到 FROM 列表中。

该参数属于 USERSET 类型参数，请参考：配置运行参数->重设参数章节中表 1 对应设置方法进行设置。

取值范围：整型，1～INT_MAX。

【须知】

- 设置为 1 会避免任何 JOIN 重排。这样就使得查询中指定的连接顺序就是实际的连接顺序。查询优化器并不是总能选取最优的连接顺序，高级用户可以选择暂时把这个变量设置为 1，然后指定它们需要的连接顺序。
- 比默认值小的数值减少规划时间但也降低了执行计划的质量。

默认值：8

plan_mode_seed

参数说明：该参数为调测参数，目前仅支持 OPTIMIZE_PLAN 和 RANDOM_PLAN 两种。其中：

OPTIMIZE_PLAN 表示通过动态规划算法进行代价估算的最优 plan，参数值设置为 0；RANDOM_PLAN 表示随机生成的 plan；如果设置 guc 参数值为-1，表示用户不指定随机数的种子标识符 seed 值，由优化器随机生成[1, 2147483647]范围整型值的随机数，并根据随机数生成随机的执行计划。

如果设置 guc 参数值为[1, 2147483647]范围的整型值，表示指定的生成随机数的种子标识符 seed，优化器需要根据 seed 值生成随机的执行计划。

该参数属于 USERSET 类型参数，请参考：配置运行参数->重设参数章节中表 1 对应设置方法进行设置。

取值范围：整型，-1~ 2147483647

默认值：0

【须知】

- 当该参数设置为随机执行计划模式时，优化器会生成不同的随机执行计划，该执行计划可能不是最优计划。因此在随机计划模式下，会对查询性能产生影响，所以建议在升级、扩容、缩容等正常业务操作或运维过程中将该参数保持为默认值 0。
- 当该参数不为 0 时，查询指定的 plan hint 不会生效。

hashagg_table_size

参数说明：用于设置执行 HASH JOIN 操作时 HASH 表的大小。

该参数属于 USERSET 类型参数，请参考：配置运行参数->重设参数章节中表 1 对应设置方法进行设置。

取值范围：整型，0 ~ INT_MAX/2。

默认值：0

enable_codegen

参数说明：标识是否允许开启代码生成优化，目前代码生成使用的是 LLVM 优化。

该参数属于 USERSET 类型参数，请参考：配置运行参数->重设参数章节中表 1 对应设置方法进行设置。

取值范围：布尔型

- on 表示允许开启代码生成优化。
- off 表示不允许开启代码生成优化。

【须知】

目前 LLVM 优化仅支持向量化执行引擎特性，在其他场景下建议关闭此参数。

默认值：off

codegen_strategy

参数说明：标识在表达式 codegen 化过程中所使用的代码生成优化策略。

该参数属于 USERSET 类型参数，请参考：配置运行参数->重设参数章节中表 1 对应设置方法进行设置。

取值范围：枚举类型

- partial 表示当所计算表达式中即使包含部分未被 codegen 化的函数时，仍可借助表达式全 codegen 框架调用 LLVM 动态编译优化策略。
- pure 表示当所计算表达式整体可被 codegen 化时，才考虑调用 LLVM 动态编译优化策略。

【须知】

在开启代码生成优化会导致查询性能下降的场景下可以设置此参数为 pure，其他场景下建议不改变此参数的默认值 partial。

默认值：partial

enable_codegen_print

参数说明：标识是否允许在 log 日志中打印所生成的 LLVM IR 函数。

该参数属于 USERSET 类型参数，请参考：配置运行参数->重设参数章节中表 1 对应设置方法进行设置。

取值范围：布尔型

- on 表示允许在 log 日志中打印 IR 函数。
- off 表示不允许在 log 日志中打印 IR 函数。

默认值：off

codegen_cost_threshold

参数说明：由于 LLVM 编译生成最终的可执行机器码需要一定时间，因此只有当实际执行的代价大于编译生成机器码所需要的代码和优化后的执行代价之和时，利用代码生成才有收益。codegen_cost_threshold 标识代价的阈值，当执行估算代价大于该代价时，使用 LLVM 优化。

该参数属于 USERSET 类型参数，请参考：配置运行参数->重设参数章节中表 1 对应设置方法进行设置。

取值范围：整型，0 ~ 2147483647。

默认值：10000

enable_bloom_filter

参数说明：标识是否允许使用 BloomFilter 优化。

该参数属于 USERSET 类型参数，请参考：配置运行参数->重设参数章节中表 1 对应设置方法进行设置。

取值范围：布尔型

- on 表示允许使用 BloomFilter 优化。
- off 表示不允许使用 BloomFilter 优化。

默认值：on

enable_extrapolation_stats

参数说明：标识对于日期类型是否允许基于历史统计信息使用推理估算的逻辑。使用该逻辑对于未及时收集统计信息的表可以增大估算准确的可能性，但也存在错误推理导致估算过大的可能性，需要对于日期类型数据定期插入的场景开启此开关。

该参数属于 SUSET 类型参数，请参考：配置运行参数->重设参数章节中表 1 对应设置方法进行设置。

取值范围：布尔型

- on 表示允许基于历史统计信息使用推理估算的逻辑。
- off 表示不允许基于历史统计信息使用推理估算的逻辑。

默认值：off

autoanalyze

参数说明：标识是否允许在生成计划的时候，对于没有统计信息的表进行统计信息自动收集。对于外表和临时表，不支持 autoanalyze，如果需要收集统计信息，用户需手动执行 analyze 操作。如果在 auto analyze 某个表的过程中数据库发生异常，当数据库正常运行之后再执行语句有可能仍提示需要收集此表的统计信息。此时需要用户对该表手动执行一次 analyze 操作，以同步统计信息数据。

该参数属于 SUSET 类型参数，请参考：配置运行参数->重设参数章节中表 1 对应设置方法进行设置。

取值范围：布尔型

- on 表示允许自动进行统计信息收集。
- off 表示不允许自动进行统计信息收集。

默认值：off

enable_analyze_check

参数说明：标识是否允许在生成计划的时候，对于在 pg_class 中显示 reltuples 和 relpages 均为 0 的表，检查该表是否曾进行过统计信息收集。

该参数属于 SUSET 类型参数，请参考：配置运行参数->重设参数章节中表 1 对应设置方法进行设置。

取值范围：布尔型

- on 表示允许检查。
- off 表示不允许检查。

默认值：off

enable_sonic_hashagg

参数说明：标识是否依据规则约束使用基于面向列的 hash 表设计的 Hash Agg 算子。

该参数属于 USERSET 类型参数，请参考：配置运行参数->重设参数章节中表 1 对应设置方法进行设置。

取值范围：布尔型

- on 表示在满足约束条件时使用基于面向列的 hash 表设计的 Hash Agg 算子。
- off 表示不使用面向列的 hash 表设计的 Hash Agg 算子。

【说明】

- 在开启 enable_sonic_hashagg，且查询达到约束条件使用基于面向列的 hash 表设计的 Hash Agg 算子时，查询对应的 Hash Agg 算子内存使用通常可获得精简。但对于代码生成技术可获得显著性能提升的场景 enable_codegen 打开后获得较大性能提升，对应的算子查询性能可能会出现劣化。

- 开启 enable_sonic_hashagg，且查询达到约束条件使用基于面向列的 hash 表设计的 Hash Agg 算子时，在 Explain Analyze/Performance 的执行计划和执行信息中，算子显示为“Sonic Hash Aggregation”，而未达到该约束条件时，算子名称将显示为“Hash Aggregation”，Explain 详解请参见：SQL 语法参考->SQL 语法->EXPLAIN 章节。

默认值：on

enable_sonic_hashjoin

参数说明：标识是否依据规则约束使用基于面向列的 hash 表设计的 Hash Join 算子。

该参数属于 USERSET 类型参数，请参考：配置运行参数->重设参数章节中表 1 对应设置方法进行设置。

取值范围：布尔型

- on 表示在满足约束条件时使用基于面向列的 hash 表设计的 Hash Join 算子。
- off 表示不使用面向列的 hash 表设计的 Hash Join 算子。

【说明】

- 当前开关仅适用于 Inner Join 的场景。
- 在开启 enable_sonic_hashjoin，查询对应的 Hash Inner 算子内存使用通常可获得精简。但对于代码生成技术可获得显著性能提升的场景，对应的算子查询性能可能会出现劣化。
- 开启 enable_sonic_hashjoin，且查询达到约束条件使用基于面向列的 hash 表设计的 Hash Join 算子时，在 Explain Analyze/Performance 的执行计划和执行信息中，算子显示为“Sonic Hash Join”，而未达到该约束条件时，算子名称将显示为“Hash Join”，Explain 详解请参见：SQL 语法参考->SQL 语法->EXPLAIN 章节。

默认值：on

enable_sonic_optspill

参数说明：标识是否对面向列的 hash 表设计的 Hash Join 算子进行下盘文件数优化。该参数打开时，在 Hash Join 算子下盘文件较多的时候，下盘文件数不会显著增加。

该参数属于 USERSET 类型参数，请参考：配置运行参数->重设参数章节中表 1 对应设置方法进行设置。

取值范围：布尔型

- on 表示优化面向列的 hash 表设计的 Hash Join 算子的下盘文件数。
- off 表示不优化面向列的 hash 表设计的 Hash Join 算子的下盘文件数。

默认值：on

log_parser_stats

参数说明：控制优化器输出 parser 模块的性能日志。

该参数属于 SUSET 类型参数，请参考：配置运行参数->重设参数章节中表 1 对应设置方法进行设置。

取值范围：布尔型

- on 表示使用。
- off 表示不使用。

默认值：off

log_planner_stats

参数说明：控制优化器输出 planner 模块的性能日志。

该参数属于 SUSET 类型参数，请参考：配置运行参数->重设参数章节中表 1 对应设置方法进行设置。

取值范围：布尔型

- on 表示使用。
- off 表示不使用。

默认值：off

log_executor_stats

参数说明：控制优化器输出 executor 模块的性能日志。

该参数属于 SUSET 类型参数，请参考：配置运行参数->重设参数章节中表 1 对应设置方法进行设置。

取值范围：布尔型

- on 表示使用。
- off 表示不使用。

默认值：off

log_statement_stats

参数说明：控制优化器输出该语句的性能日志。

该参数属于 SUSET 类型参数，请参考：配置运行参数->重设参数章节中表 1 对应设置方法进行设置。

取值范围：布尔型

- on 表示使用。
- off 表示不使用。

默认值：off

plan_cache_mode

参数说明：标识在 prepare 语句中，选择生成执行计划的策略。

该参数属于 USERSET 类型参数，请参考：配置运行参数->重设参数章节中表 1 对应设置方法进行设置。

取值范围：枚举类型

- auto 表示按照默认的方式选择 custom plan 或者 generic plan。
- force_generic_plan 表示强制走 generic plan。
- force_custom_plan 表示强制走 custom plan。

【说明】

- 此参数只对 prepare 语句生效，一般用在 prepare 语句中参数化字段存在比较严重的数据倾斜的场景下。
- custom plan 是指对于 preapre 语句，在执行 execute 的时候，把 execute 语句中的参数嵌套到语句之后生成的计划。custom plan 会根据 execute 语句中具体的参数生成计划，这种方案的优点是每次都按照具体的参数生成优选计划，执行性能比较好；缺点是每次执行前都需要重新生成计划，存在大量的重复的优化器开销。
- generic plan 是指对于 preapre 语句生成计划，该计划策略会在执行 execute 语句的时候把参数 bind 到 plan 中，然后执行计划。这种方案的优点是每次执行可以省去重复的优化器开销；缺点是当 bind 参数字段上数据存在倾斜时该计划可能不是最优的，部分 bind 参数场景下执行性能较差。

默认值：auto

enable_hypo_index

参数说明：控制优化器执行 EXPLAIN 命令时是否考虑虚拟索引。

该参数属于 USERSET 类型参数，请参考：配置运行参数->重设参数章节中表 1 对应设置方法进行设置。

取值范围：布尔型

- on 表示使用。
- off 表示不使用。

默认值：off

enable_force_vector_engine

参数说明：对于支持向量化的执行器算子，如果其子节点是非向量化的算子，通过设置此参数为 on，强制生成向量化的执行计划。

该参数属于 USERSET 类型参数，请参考：配置运行参数->重设参数章节中表 1 对应设置方法进行设置。

取值范围：布尔型

- on 表示可以向量化的算子强制生成向量化。
- off 表示由向量化算子优化器决定是否向量化。

默认值：off

enable_auto_explain

参数说明：控制是否开启自动打印执行计划。该参数是用来定位慢存储过程或慢查询，只对当前连接的数据库主节点有效。

该参数属于 USERSET 类型参数，请参考：配置运行参数->重设参数章节中表 1 对应设置方法进行设置。

取值范围：布尔型

- true 表示开启。
- false 表示关闭。

默认值：false

auto_explain_level

参数说明：控制自动打印执行计划的日志等级。

该参数属于 USERSET 类型参数，请参考：配置运行参数->重设参数章节中表 1 对应设置方法进行设置。

取值范围：枚举型，LOG 或 NOTICE。NOTICE 表示以提示知的形式打印出计划。

- LOG 表示在日志中打印执行计划。
- NOTICE 表示以提示知的形式打印出计划。

默认值：LOG

auto_dop

参数说明： 是否自动降级 DOP（并发计划数）。

该参数属于 USERSET 类型参数，请参考重设参数表 1 中对应设置方法进行设置。

取值范围： 布尔型

- on：开启自动降级功能。
- off：关闭自动降级功能。

默认值： off

auto_dop_freeprocs

参数说明： 该参数用于设置是否根据空闲进程的数量自动调整 query_dop（当前算子执行时的并行度）。

该参数属于 USERSET 类型参数，请参考重设参数表 1 中对应设置方法进行设置。

取值范围： 布尔型

- on：开启自动调整功能。

❖ off：关闭自动调整功能。

默认值： off

auto_dop_freeprocs_threshold

参数说明： 禁用 DOP（并发计划数）的最小空闲进程数量。

该参数属于 USERSET 类型参数，请参考重设参数表 1 中对应设置方法进行设置。

取值范围： 0～2147483647，integer 类型

默认值： 10

auto_dop_join_threshold

参数说明： 将 DOP（并发计划数）降低到 2 的最大连接阈值。

该参数属于 USERSET 类型参数，请参考重设参数表 1 中对应设置方法进行设置。

取值范围： -1～2147483647，integer 类型

默认值： 4

enable_seqscan_dopcost

参数说明： 启用顺序扫描的 DOP 成本计算。

该参数属于 USERSET 类型参数，请参考重设参数表 1 中对应设置方法进行设置。

取值范围： 布尔型

❖ on，表示按并行度计算 SeqScan 的 IO 代价。
❖ off，表示优化器不再按并行度计算 SeqScan 的 IO 代价。

默认值: on

enable_startwith_debug

参数说明：该参数为 start with/connect by 用于 debug 的参数，打开参数可以显示 start with 特性所有涉及的尾列相关信息。

该参数属于 USERSET 类型参数，请参考：配置运行参数->重设参数章节中表 1 对应设置方法进行设置。

取值范围：布尔型

- on 表示开启。
- off 表示关闭。

默认值：off

show_fdw_remote_plan

参数说明： 该参数控制在 explain 中是否打印 fdw 获取远端数据的方法内容。 当查询中存在外表时，数据库需要使用 ForeignScan 算子从远端服务器获取实际的数据。此时开启此参数，则在 explain 中，会为使用到的 ForeignScan 算子进行编号，并按顺序将每个 ForeignScan 算子获取远端数据的方法内容追加打印至 explain 的结果中。具体打印内容会调用所使用的 FDW 的远程计划打印专用接口，由 FDW 自己进行组织填写。若 FDW 不支持此接口，则会提示无相关计划信息。

该参数属于 USERSET 类型参数，请参考重设参数表 1 中对应设置方法进行设置。

取值范围： 布尔型

- on：表示开启该参数。
- off：表示关闭该参数。

默认值： off

max_index_parallel_workers

参数说明：该参数用于配置索引最大并行度，当参数值为 0 时表示不开启并行。实际创建索引时的并行度取 max_index_parallel_workers 参数值和目标表本身的并行度 parallel_workers 值中的最小值。

取值范围：整型，1~1024。

默认值：4

【说明】
目标表本身的并行度 parallel_workers 由用户通过表的存储参数指定，表示创建索引时启动的 bgworker 线程数量，详见 SQL 语法参考->SQL 语法->ALTER TABLE 章节。

enable_indexscan_optimization

参数说明：控制是否对 astore 存储引擎下的 btree 索引扫描 (IndexScan 和 IndexOnlyScan) 进行优化。

该参数属于 USERSET 类型参数，请参考重设参数表 1 中对应设置方法进行设置。

取值范围：布尔型

- on：进行优化。
- off：不进行优化。

默认值： off

force_parallel_build_index

参数说明： 目前并行创建索引需要满足几个条件：

- 分区表或者设置了并行度 parallel_workers。

- 对于普通表或者分区表本地索引，通过 plan_create_index_workers 计算的并行度优先并且不大于 a.parallel_workers 以及系统限制的最大值。

通过设置此参数，可以直接使用设置的 parallel_workers（小于等于系统限制值）并行创建索引。对于分区表的全局索引不生效，继续使用原来的逻辑。

该参数属于 USERSET 类型参数，请参考重设参数表 1 中对应设置方法进行设置。

取值范围： 布尔型

- on：设置此参数为 on，可以直接使用设置的 parallel_workers（小于等于系统限制值）并行创建索引。
- off：关闭此参数，表示采用通过 plan_create_index_workers 计算的并行度。

默认值： off

max_stream_workers

参数说明： 给 dop 并发线程预留的最大数量。

该参数属于 POSTMASTER 类型参数，请参考重设参数表 1 中对应设置方法进行设置。

取值范围： 整型，0-262143

默认值： 64

enable_streaming_plan_degrade

参数说明： 该参数用于控制当并发的总线程数量超过 max_stream_workers 时，是否将当前无法获取到足够子线程数量的 SMP 计划降级为普通的串行计划。

该参数属于 USERSET 类型参数，请参考重设参数表 1 中对应设置方法进行设置。

取值范围： 布尔型

- on：降级。
- off：不降级。

默认值： on

12.9.错误报告和日志

12.9.1.记录日志的位置

log_destination

参数说明：PanWeiDB 支持多种方法记录服务器日志，log_destination 的取值为一个逗号分隔开的列表（如 log_destination="stderr,csvlog"）。

该参数属于 SIGHUP 类型参数，请参考：配置运行参数->重设参数章节中表 1 对应设置方法进行设置。

取值范围：字符串

有效值为 stderr、csvlog、syslog。

- 取值为 stderr，表示日志打印到屏幕。
- 取值为 csvlog，表示日志的输出格式为“逗号分隔值”即 CSV（Comma Separated Value）格式。使用 csvlog 记录日志的前提是将 logging_collector 设置为 on，请参见：GUC 参数说明->错误报告和日志->使用 CSV 格式写日志章节。
- 取值为 syslog，表示通过操作系统的 syslog 记录日志。PanWeiDB 使用 syslog 的 LOCAL0～LOCAL7 记录日志，请参见 syslog_facility。使用 syslog 记录日志需在操作系统后台服务配置文件中添加代码：

```
local0.* /var/log/omm
```

默认值：stderr

logging_collector

参数说明：控制开启后端日志收集进程 logger 进行日志收集。该进程捕获发送到 stderr 或 csvlog 的日志消息并写入日志文件。

这种记录日志的方法比将日志记录到 syslog 更加有效，因为某些类型的消息在 syslog 的输出中无法显示。例如动态链接库加载失败消息和脚本（例如 archive_command）产生的错误消息。

该参数属于 POSTMASTER 类型参数，为固定参数，用户无法修改此参数，只能查看。

须知

将服务器日志发送到 stderr 时可以不使用 logging_collector 参数，此时日志消息会被发送到服务器的 stderr 指向的空间。这种方法的缺点是日志回滚困难，只适用于较小的日志容量。

取值范围：布尔型

- on 表示开启日志收集功能。
- off 表示关闭日志收集功能。

默认值：on

log_directory

参数说明：logging_collector 设置为 on 时，log_directory 决定存放服务器日志文件的目录。它可以是绝对路径或者是相对路径（相对于数据目录的路径）。log_directory 支持动态修改，可以通过 pw_guc reload 实现。

该参数属于 SIGHUP 类型参数，请参考：配置运行参数->重设参数章节中表 1 对应设置方法进行设置。

须知

- 当配置文件中 log_directory 的值为非法路径时，会导致 PanWeiDB 无法重新启动。
- 通过 pw_guc reload 动态修改 log_directory 时，当指定路径为合法路径时，日志输出到新的路径下。当指定路径为非法路径时，日志输出到上

一次合法的日志输出路径下而不影响数据库正常运行。此时即使指定的 log_directory 的值非法，也会写入到配置文件中。

- 在沙箱环境，路径中不可以包含/var/chroot，例如 log 的绝对路径是 /var/chroot/var/lib/log/Ruby/pg_log/cn_log，则只需要设置为 /var/lib/log/Ruby/pg_log/cn_log。

说明

- 合法路径：用户对此路径有读写权限。
- 非法路径：用户对此路径无读写权限。

取值范围：字符串

默认值：安装时指定。

log_filename

参数说明：logging_collector 设置为 on 时，log_filename 决定服务器运行日志文件的名称。通常日志文件名是按照 strftime 模式生成，因此可以用系统时间定义日志文件名，用%转义字符实现。

该参数属于 SIGHUP 类型参数，请参考：配置运行参数->重设参数章节中表 1 对应设置方法进行设置。

须知

- 建议使用%转义字符定义日志文件名称，否则难以对日志文件进行有效的管理。
- 当 log_destination 设为 csvlog 时，系统会生成附加了时间戳的日志文件名，文件格式为 csv 格式，例如“server_log.1093827753.csv”。

取值范围：字符串

默认值：postgresql-%Y-%m-%d_%H%M%S.log

log_file_mode

参数说明：logging_collector 设置为 on 时，log_file_mode 设置服务器日志文件的权限。通常 log_file_mode 的取值是能够被 chmod 和 umask 系统调用接受的数字。

该参数属于 SIGHUP 类型参数，请参考：配置运行参数->重设参数章节中表 1 对应设置方法进行设置。

须知

- 使用此选项前请设置 log_directory，将日志存储到数据目录之外的地方。
- 因日志文件可能含有敏感数据，故不能将其设为对外可读。

取值范围：整型，0000～0777（8 进制计数，转化为十进制 0～511）。

说明

- 0600 表示只允许服务器管理员读写日志文件。
- 0640 表示允许管理员所在用户组成员只能读日志文件。

默认值：0600

log_truncate_on_rotation

参数说明：logging_collector 设置为 on 时，log_truncate_on_rotation 设置日志消息的写入方式。

该参数属于 SIGHUP 类型参数，请参考：配置运行参数->重设参数章节中表 1 对应设置方法进行设置。

示例如下：

假设日志需要保留 7 天，每天生成一个日志文件，日志文件名设置为 server_log.Mon、server_log.Tue 等。第二周的周二生成的日志消息会覆盖写入到 server_log.Tue。设置方法：将 log_filename 设置为 server_log.%a，log_truncate_on_rotation 设置为 on，log_rotation_age 设置为 1440，即日志有效时间为 1 天。

取值范围：布尔型

- on 表示 PanWeiDB 以覆盖写入的方式写服务器日志消息。
- off 表示 PanWeiDB 将日志消息附加到同名的现有日志文件上。

默认值：off

log_rotation_age

参数说明：logging_collector 设置为 on 时，log_rotation_age 决定创建一个新日志文件的时间间隔。当现在的时间减去上次创建一个服务器日志的时间超过了 log_rotation_age 的值时，将生成一个新的日志文件。

该参数属于 SIGHUP 类型参数，请参考：配置运行参数->重设参数章节中表 1 对应设置方法进行设置。

取值范围：整型，0 ~ 35791394，单位为 min。其中 0 表示关闭基于时间的新日志文件的创建。

默认值：1440(min)

log_rotation_size

参数说明：logging_collector 设置为 on 时，log_rotation_size 决定服务器日志文件的最大容量。当日志消息的总量超过日志文件容量时，服务器将生成一个新的日志文件。

该参数属于 SIGHUP 类型参数，请参考：配置运行参数->重设参数章节中表 1 对应设置方法进行设置。

取值范围：整型，0 ~ INT_MAX / 1024kB，支持单位为 kB/MB/GB。

0 表示关闭基于容量的新日志文件的创建。

建议该值大小设置级别至少为 MB 级，利于日志文件的及时划分。

默认值：10MB

syslog_facility

参数说明：log_destination 设置为 syslog 时，syslog_facility 配置使用 syslog 记录日志的“设备”。

该参数属于 SIGHUP 类型参数，请参考：配置运行参数->重设参数章节中表 1 对应设置方法进行设置。

取值范围：枚举类型，有效值有 local0、local1、local2、local3、local4、local5、local6、local7。

默认值：local0

syslog_ident

参数说明：log_destination 设置为 syslog 时，syslog_ident 设置在 syslog 日志中 PanWeiDB 日志消息的标识。

该参数属于 SIGHUP 类型参数，请参考：配置运行参数->重设参数章节中表 1 对应设置方法进行设置。

取值范围：字符串

默认值：postgres

event_source

参数说明：该参数仅在 windows 环境下生效，PanWeiDB 暂不支持。log_destination 设置为 eventlog 时，event_source 设置在日志中 PanWeiDB 日志消息的标识。

该参数属于 POSTMASTER 类型参数，请参考：配置运行参数->重设参数章节中表 1 对应设置方法进行设置。

取值范围：字符串

默认值：PostgreSQL

12.9.2.记录日志的时间

client_min_messages

参数说明：控制发送到客户端的消息级别。每个级别都包含排在它后面的所有级别中的信息。级别越低，发送给客户端的消息就越少。

该参数属于 USERSET 类型参数，请参考：配置运行参数->重设参数章节中表 1 对应设置方法进行设置。

须知

当 client_min_messages 和 log_min_messages 取相同值时，其值所代表的级别不同。

取值范围：枚举类型，有效值有 debug、debug5、debug4、debug3、debug2、debug1、info、log、notice、warning、error、fatal、panic。参数的详细信息请参见：配置运行参数->重设参数章节中表 1。在实际设置过程中，如果设置的级别大于 error，为 fatal 或 panic，系统会默认将级别转为 error。

默认值：notice

log_min_messages

参数说明：控制写到服务器日志文件中的消息级别。每个级别都包含排在它后面的所有级别中的信息。级别越低，服务器运行日志中记录的消息就越少。

该参数属于 SUSET 类型参数，请参考：配置运行参数->重设参数章节中表 1 对应设置方法进行设置。

须知

当 client_min_messages 和 log_min_messages 取相同值 log 时所代表的消息级别不同。

部分日志信息的打印需要同时配置该参数与 logging_module，即设置该参数打开后可能还需要设置 logging_module 打开对应模块的日志打印开关。

取值范围：枚举类型，有效值有 debug、debug5、debug4、debug3、debug2、debug1、info、log、notice、warning、error、fatal、panic。参数的详细信息请参见：配置运行参数->重设参数章节中表 1。

默认值：warning

log_min_error_statement

参数说明：控制在服务器日志中记录错误的 SQL 语句。

该参数属于 SUSET 类型参数，请参考：配置运行参数->重设参数章节中表 1 对应设置方法进行设置。

取值范围：枚举类型，有效值有 debug、debug5、debug4、debug3、debug2、debug1、info、log、notice、warning、error、fatal、panic。参数的详细信息请参见：配置运行参数->重设参数章节中表 1。

说明

- 设置为 error，表示导致错误、日志消息、致命错误、panic 的语句都将被记录。
- 设置为 panic，表示关闭此特性。

默认值：error

log_min_duration_statement

参数说明：当某条语句的持续时间大于或者等于特定的毫秒数时，log_min_duration_statement 参数用于控制记录每条完成语句的持续时间。

设置 log_min_duration_statement 可以很方便地跟踪需要优化的查询语句。对于使用扩展查询协议的客户端，语法分析、绑定、执行每一步所花时间被独立记录。

该参数属于 SUSET 类型参数，请参考：配置运行参数->重设参数章节中表 1 对应设置方法进行设置。

须知

当此选项与 log_statement 同时使用时，已经被 log_statement 记录的语句文本不会被重复记录。在没有使用 syslog 情况下，推荐使用 log_line_prefix 记录 PID 或会话 ID，方便将当前语句消息连接到最后的持续时间消息。

取值范围：整型，-1 ~ INT_MAX，单位为毫秒。

- 设置为 250，所有运行时间不短于 250ms 的 SQL 语句都会被记录。
- 设置为 0，输出所有语句的持续时间。
- 设置为-1，关闭此功能。

默认值：5s

backtrace_min_messages

参数说明：控制当产生该设置参数级别相等或更高级别的信息时，会打印函数的堆栈信息到服务器日志文件中。

该参数属于 SUSET 类型参数，请参考：配置运行参数->重设参数章节中表 1 对应设置方法进行设置。

须知

该参数作为客户现场问题定位手段使用，且由于频繁的打印函数栈会对系统的开销及稳定性有一定的影响，因此如果需要进行问题定位时，建议避免将 backtrace_min_messages 的值设置为 fatal 及 panic 以外的级别。

取值范围：枚举类型

有效值有 debug、debug5、debug4、debug3、debug2、debug1、info、log、notice、warning、error、fatal、panic。参数的详细信息请参见下表表 1：信息严重程度分类。

默认值：panic

表 1 解释 PanWeiDB 中使用的消息安全级别。当日志输出到 syslog 或者 eventlog（仅 windows 环境下，PanWeiDB 版本不涉及该参数）时，PanWeiDB 进行如表中的转换。

表 1 信息严重程度分类

信息严重程度类型	详细说明	系统日志	事件日志
debug[1-5]	报告详细调试信息。	DEBUG	INFORMATION
log	报告对数据库管理员有用的信息， 比如检查点操作统计信息。	INFO	INFORMATION
info	报告用户可能需求的信息， 比如在 VACUUM VERBOSE 过程中的信息。	INFO	INFORMATION
notice	报告可能对用户有帮助的信息， 比如， 长标识符的截断， 作为主键一部分创建的索引等。	NOTICE	INFORMATION
warning	报告警告信息， 比如在事务块范围之外的 COMMIT。	NOTICE	WARNING
error	报告导致当前命令退出的错误。	WARNING	ERROR
fatal	报告导致当前会话终止的原因。	ERR	ERROR
panic	报告导致整个数据库被关闭的原因。	CRIT	ERROR

plog_merge_age

参数说明：该参数用于控制性能日志数据输出的周期。

该参数属于 USERSET 类型参数，请参考：配置运行参数->重设参数章节中表 1 对应设置方法进行设置。

须知

该参数以毫秒为单位，建议在使用过程中设置值为 1000 的整数倍，即设置值以秒为最小单位。该参数所控制的性能日志文件以 prf 为扩展名，文件放置在 $GAUSSLOG/gs_profile/<node_name>目录下面，其中 node_name 是由 postgres.conf 文件中的 pgxc_node_name 的值，不建议外部使用该参数。

取值范围：0~2147483647，单位为毫秒（ms）。

当设置为 0 时，当前会话不再输出性能日志数据。当设置为非 0 时，当前会话按照指定的时间周期进行输出性能日志数据。

该参数设置得越小，输出的日志数据越多，对性能的负面影响越大。

默认值：0

12.9.3.记录日志的内容

debug_print_parse

参数说明：用于控制打印解析树结果。

该参数属于 SIGHUP 类型参数，请参考：配置运行参数->重设参数章节中表 1 对应设置方法进行设置。

取值范围：布尔型

- on 表示开启打印结果的功能。
- off 表示关闭打印结果的功能。

默认值：off

debug_print_rewritten

参数说明：用于控制打印查询重写结果。

该参数属于 SIGHUP 类型参数，请参考：配置运行参数->重设参数章节中表 1 对应设置方法进行设置。

取值范围：布尔型

- on 表示开启打印结果的功能。
- off 表示关闭打印结果的功能。

默认值：off

debug_print_plan

参数说明：用于设置是否将查询的执行计划打印到日志中。

该参数属于 SIGHUP 类型参数，请参考：配置运行参数->重设参数章节中表 1 对应设置方法进行设置。

取值范围：布尔型

- on 表示开启打印结果的功能。
- off 表示关闭打印结果的功能。

默认值：off

【须知】

- 只有当日志的级别为 log 及以上时，debug_print_parse、debug_print_rewritten 和 debug_print_plan 的调试信息才会输出。当这些选项打开时，调试信息只会记录在服务器的日志中，而不会输出到客户端的日志中。通过设置 GUC 参数说明->错误报告和日志->记录日志的时间章节的 client_min_messages 和 log_min_messages 参数可以改变日志级别。
- 在打开 debug_print_plan 开关的情况下需尽量避免调用 gs_encrypt_aes128 及 gs_decrypt_aes128 函数，避免敏感参数信息在日志中泄露的风险。同时建议用户在打开 debug_print_plan 开关生成的日志中对 gs_encrypt_aes128 及 gs_decrypt_aes128 函数的参数信息进行过滤后再提供给外部维护人员定位，日志使用完成后请及时删除。

debug_pretty_print

参数说明：设置此选项对 debug_print_parse、debug_print_rewritten 和 debug_print_plan 产生的日志进行缩进，会生成易读但比设置为 off 时更长的输出格式。

该参数属于 USERSET 类型参数，请参考：配置运行参数->重设参数章节中表 1 对应设置方法进行设置。

取值范围：布尔型

- on 表示进行缩进。
- off 表示不进行缩进。

默认值：on

log_checkpoints

参数说明：控制在服务器日志中记录检查点和重启点的信息。打开此参数时，服务器日志消息包含涉及检查点和重启点的统计量，其中包含需要写的缓存区的数量及写入所花费的时间等。

该参数属于 SIGHUP 类型参数，请参考：配置运行参数->重设参数章节中表 1 对应设置方法进行设置。

取值范围：布尔型

- on 表示打开此参数时，服务器日志消息包含涉及检查点和重启点的统计量。
- off 表示关闭此参数时，服务器日志消息包含不涉及检查点和重启点的统计量。

默认值：off

log_connections

参数说明：控制记录客户端的连接请求信息。

该参数属于 BACKEND 类型参数，请参考：配置运行参数->重设参数章节中表 1 对应设置方法进行设置。

【须知】
有些客户端程序（例如 gsql），在判断是否需要口令的时候会尝试连接两次，因此日志消息中重复的“connection receive”（收到连接请求）并不意味着一定是问题。

取值范围：布尔型

- on 表示记录信息。
- off 表示不记录信息。

默认值：off

log_disconnections

参数说明：控制记录客户端结束连接信息。

该参数属于 BACKEND 类型参数，请参考：配置运行参数->重设参数章节中表 1 对应设置方法进行设置。

取值范围：布尔型

- on 表示记录信息。
- off 表示不记录信息。

默认值：off

log_duration

参数说明：控制记录每个已完成 SQL 语句的执行时间。对使用扩展查询协议的客户端、会记录语法分析、绑定和执行每一步所花费的时间。

该参数属于 SUSET 类型参数，请参考：配置运行参数->重设参数章节中表 1 对应设置方法进行设置。

取值范围：布尔型

- 设置为 off，该选项与 GUC 参数说明->错误报告和日志->记录日志的时间中 log_min_duration_statement 参数的不同之处在于 log_min_duration_statement 强制记录查询文本。
- 设置为 on 并且 log_min_duration_statement 大于零，记录所有持续时间，但是仅记录超过阈值的语句。这可用于在高负载情况下搜集统计信息。

默认值：on

log_error_verbosity

参数说明：控制服务器日志中每条记录的消息写入的详细度。

该参数属于 SUSET 类型参数，请参考：配置运行参数->重设参数章节中表 1 对应设置方法进行设置。

取值范围：枚举类型

- terse 代表输出不包括 DETAIL、HINT、QUERY 及 CONTEXT 错误信息的记录。
- verbose 代表输出包括 SQLSTATE 错误代码、源代码文件名、函数名及产生错误所在的行号。
- default 代表输出包括 DETAIL、HINT、QUERY 及 CONTEXT 错误信息的记录，不包括 SQLSTATE 错误代码 、源代码文件名、函数名及产生错误所在的行号。

默认值：default

log_hostname

参数说明：选项关闭状态下，连接消息日志只显示正在连接主机的 IP 地址。打开此选项同时可以记录主机名。由于解析主机名可能需要一定的时间，可能影响数据库的性能。

该参数属于 SIGHUP 类型参数，请参考：配置运行参数->重设参数章节中表 1 对应设置方法进行设置。

取值范围：布尔型

- on 表示可以同时记录主机名。
- off 表示不可以同时记录主机名。

默认值：on

log_line_prefix

参数说明：控制每条日志信息的前缀格式。日志前缀类似于 printf 风格的字符串，在日志的每行开头输出。用以%为开头的“转义字符”代替配置运行参数->重设参数章节中表 1 中的状态信息。

该参数属于 SIGHUP 类型参数，请参考：配置运行参数->重设参数章节中表 1 对应设置方法进行设置。

表 1 转义字符表

转义字符	效果
%a	应用程序名称。
%u	用户名。
%d	数据库名。
%r	远端主机名或者 IP 地址以及远端端口，在不启动 log_hostname 时显示 IP 地址及远端端口。
%h	远端主机名或者 IP 地址，在不启动 log_hostname 时只显示 IP 地址。
%p	线程 ID。
%t	时间戳（没有毫秒）。
%m	带毫秒的时间戳。
%n	表示指定错误从哪个节点上报的。
%i	命令标签：会话当前执行的命令类型。
%e	SQLSTATE 错误码。
%c	会话 ID，详见说明。
%l	每个会话或线程的日志编号，从 1 开始。

转义字符	效果
%s	进程启动时间。
%v	虚拟事务 ID（backendID/ localXID）。
%x	事务 ID（0 表示没有分配事务 ID）。
%q	不产生任何输出。如果当前线程是后端线程，忽略这个转义序列，继续处理后面的转义序列；如果当前线程不是后端线程，忽略这个转义序列和它后面的所有转义序列。
%S	会话 ID。
%%	字符%。

【说明】

转义字符%c 打印一个会话 ID，由两个 4 字节的十六进制数组成，通过字符“.”分开。这两个十六进制数分别表示进程的启动时间及进程编号，所以%c 也可以看作是保存打印这些名目的途径的空间。比如，从 pg_stat_activity 中产生会话 ID，可以用下面的查询：

```
SELECT to_hex(EXTRACT(EPOCH FROM backend_start)::integer) || '.' ||
to_hex(pid)  FROM pg_stat_activity;
```

当 log_line_prefix 设置为非空值时，请将其最后一个字符作为一个独立的段，以此来直观地与后续的日志进行区分，也可以使用一个标点符号。

Syslog 生成自己的时间戳及进程 ID 信息，所以当登录日志时，不需要包含这些转义字符。

取值范围：字符串

默认值：空

【说明】

%m %c %d %p %a %x %n %e 表示在日志开头附加会话开始时间戳、会话 ID、数据库名、线程 ID、应用程序名、事务 ID、报错节点、SQLSTATE 错误码。

log_lock_waits

参数说明：当一个会话的等待获得一个锁的时间超过 GUC 参数说明->锁管理中 deadlock_timeout 参数值时，此选项控制在数据库日志中记录此消息。这对于决定锁等待是否会产生一个坏的行为是非常有用的。

该参数属于 SUSET 类型参数，请参考：配置运行参数->重设参数章节中表 1 对应设置方法进行设置。

取值范围：布尔型

- on 表示记录此信息。
- off 表示不记录此信息。

默认值：off

log_statement

参数说明：控制记录 SQL 语句。对于使用扩展查询协议的客户端，记录接收到执行消息的事件和绑定参数的值（内置单引号要双写）。

该参数属于 SUSET 类型参数，请参考：配置运行参数->重设参数章节中表 1 对应设置方法进行设置。

【须知】
即使 log_statement 设置为 all，包含简单语法错误的语句也不会被记录，因为仅在完成基本的语法分析并确定了语句类型之后才记录日志。在使用扩展查询协议的情况下，在执行阶段之前（语法分析或规划阶段）同样不会记录。将 log_min_error_statement 设为 ERROR 或更低才能记录这些语句。

取值范围：枚举类型

- none 表示不记录语句。
- ddl 表示记录所有的数据定义语句，比如 CREATE、ALTER 和 DROP 语句。
- mod 表示记录所有 DDL 语句，还包括数据修改语句 INSERT、UPDATE、DELETE、TRUNCATE 和 COPY FROM 。

- all 表示记录所有语句，PREPARE、EXECUTE 和 EXPLAIN ANALYZE 语句也同样被记录。

默认值：none

log_temp_files

参数说明：控制记录临时文件的删除信息。临时文件可以用来排序、哈希及临时查询结果。当一个临时文件被删除时，将会产生一条日志消息。

该参数属于 SUSET 类型参数，请参考：配置运行参数->重设参数章节中表 1 对应设置方法进行设置。

取值范围：整型，最小值为-1，最大值为 2147483647，单位 KB。

- 正整数表示只记录比 log_temp_files 设定值大的临时文件的删除信息。
- 值 0 表示记录所有的临时文件的删除信息。
- 值-1 表示不记录任何临时文件的删除信息。

默认值：-1

log_timezone

参数说明：设置服务器写日志文件时使用的时区。与 GUC 参数说明->客户端连接缺省设置->区域和格式化中 TimeZone 参数不同，这个值是数据库范围的，针对所有连接到本数据库的会话生效。

该参数属于 SIGHUP 类型参数，请参考：配置运行参数->重设参数章节中表 1 对应设置方法进行设置。

取值范围：字符串，可查询视图 PG_TIMEZONE_NAMES 获得。

默认值：GMT

【说明】

pw_initdb 进行相应系统环境设置时会对默认值进行修改。

logging_module

参数说明：用于设置或者显示模块日志在服务端的可输出性。该参数属于会话级参数，不建议通过 pw_guc 工具来设置。

该参数属于 USERSET 类型参数，设置请参考：配置运行参数->重设参数章节中表 1 中对应设置的方法进行设置。

取值范围：字符串

默认值：所有模块日志在服务端是不输出的（即 off(ALL)），可由 SHOW logging_module;查看。

依赖关系：该参数依赖于 log_min_messages 参数的设置。

设置方法：首先，可以通过 SHOW logging_module 来查看哪些模块是支持可控制的。例如：

```
show logging_module;
```

查询输出结果为：

```
                                logging_module
-------------------------------------------------------------------------------------------------------------------
 ALL,on(),off(COMMAND,DFS,GUC,GSCLEAN,HDFS,ORC,SLRU,MEM_CTL,AUTOVAC,CACHE,ADIO,SSL,GDS,TBLSPC,WLM,OBS,INDEX,EXECUTOR,OPFUSION,GPC,GSC,VEC_EXECUTOR,S
TREAM,LLVM,OPT,OPT_REWRITE,OPT_JOIN,OPT_AGG,OPT_CHOICE,OPT_SUBPLAN,OPT_SETOP,OPT_SKEW,OPT_PLANNER,UDF,COOP_ANALYZE,WLMCP,ACCELERATE,MOT,PLANHINT,PAR
QUET,PGSTAT,CARBONDATA,SNAPSHOT,XACT,HANDLE,CLOG,EC,REMOTE,CN_RETRY,PLSQL,TEXTSEARCH,SEQ,REDO,FUNCTION,PARSER,INSTR,WDR_SNAPSHOT,INCRE_CKPT,INCRE_BG
_WRITER,DBL_WRT,RTO_RPO,HEARTBEAT,COMM_IPC,COMM_PARAM,TIMESERIES,SCHEMA,SEGMENT_PAGE,LIGHTPROXY,HOTKEY,THREAD_POOL,OPT_AI,WALRECEIVER,USTORE,UNDO,TI
```

```
MECAPSULE,GEN_COL,DCF,DB4AI,PLDEBUGGER,ADVISOR,SEC,SEC_FE,SEC_LEGER,SE
C_POLICY,SEC_SDD,SEC_TDE,COMM_FRAMEWORK,COMM_PROXY,COMM_POOLER,VA
CUUM,JOB,SPI,
NEST_COMPILE,RESOWNER,GSSTACK,LOGICAL_DECODE,GPRC,DISASTER_READ,STAN
DBY_READ,REPSYNC,SQLPATCH,DMS,DSS_API,GPI,PARTITION,SRF,SS_TXNSTATUS,BA
CKEND)
(1 row)
```

支持可控制的模块使用大写来标识，特殊标识 ALL 用于对所有模块日志进行设置。可以使用 on/off 来控制模块日志的输出。

❖ 设置 SSL 模块日志为可输出，使用如下命令：

```
set logging_module='on(SSL)';
show logging_module;
```

查询输出结果如下所示，可以看到模块 SSL 的日志输出被打开：

```
                                                                                                    logging_module
-------------------------------------------------------------------------------------------------------------------------------------------------------
-------------------------ALL,on(SSL),off(COMMAND,DFS,GUC,GSCLEAN,HDFS,ORC,SL
RU,MEM_CTL,AUTOVAC,CACHE,ADIO,GDS,TBLSPC,WLM,OBS,INDEX,EXECUTOR,OPFUS
ION,GPC,GSC,VEC_EX
ECUTOR,STREAM,LLVM,OPT,OPT_REWRITE,OPT_JOIN,OPT_AGG,OPT_CHOICE,OPT_SU
BPLAN,OPT_SETOP,OPT_SKEW,OPT_PLANNER,UDF,COOP_ANALYZE,WLMCP,ACCELER
ATE,MOT,PLANHINT,PARQUET,PGSTAT,CARBONDATA,SNAPSHOT,XACT,HANDLE,CLO
G,EC,REMOTE,CN_RETRY,PLSQL,TEXTSEARCH,SEQ,REDO,FUNCTION,PARSER,INSTR,W
DR_SNAPSHOT,INCRE_CKPT,INCRE_BG_WRITER,DBL_WRT,RTO_RPO,HEARTBEAT,COM
M_IPC,COMM_PARAM,TIMESERIES,SCHEMA,SEGMENT_PAGE,LIGHTPROXY,HOTKEY,TH
READ_POOL,OPT_AI,WALRECEIVER,USTORE,UNDO,GEN_COL,DCF,DB4AI,PLDEBUGGE
R,ADVISOR,SEC,SEC_FE,SEC_LEGER,SEC_POLICY,SEC_SDD,SEC_TDE,COMM_PROXY,CO
MM_POOLER,VACUUM,JOB,SPI,NEST_COMPILE,RESOWNER,LOGICAL_DECODE,GPRC)
(1 row)
```

ALL 标识是相当于一个快捷操作，即对所有模块的日志可输出进行开启或关闭。

设置所有模块日志为可输出，使用如下命令：

```
set logging_module='off(ALL)';
show logging_module;
```

查询输出结果如下所示：

```
                                   logging_module
--------------------------------------------------------------------------------------
------------------------ ALL,on(COMMAND,DFS,GUC,GSCLEAN,HDFS,ORC,SLRU,MEM
_CTL,AUTOVAC,CACHE,ADIO,SSL,GDS,TBLSPC,WLM,OBS,INDEX,EXECUTOR,OPFUSION,
GPC,GSC,VEC_EXECUTOR,STREAM,LLVM,OPT,OPT_REWRITE,OPT_JOIN,OPT_AGG,OPT_
CHOICE,OPT_SUBPLAN,OPT_SETOP,OPT_SKEW,OPT_PLANNER,UDF,COOP_ANALYZE,
WLMCP,ACCELERATE,MOT,PLANHINT,PARQUET,PGSTAT,CARBONDATA,SNAPSHOT,XA
CT,HANDLE,CLOG,EC,REMOTE,CN_RETRY,PLSQL,TEXTSEARCH,SEQ,REDO,FUNCTION,P
ARSER,INSTR,WDR_SNAPSHOT,INCRE_CKPT,INCRE_BG_WRITER,DBL_WRT,RTO_RPO,H
EARTBEAT,COMM_IPC,COMM_PARAM,TIMESERIES,SCHEMA,SEGMENT_PAGE,LIGHTPR
OXY,HOTKEY,THREAD_POOL,OPT_AI,WALRECEIVER,USTORE,UNDO,GEN_COL,DCF,DB
4AI,PLDEBUGGER,ADVISOR,SEC,SEC_FE,SEC_LEGER,SEC_POLICY,SEC_SDD,SEC_TDE,C
OMM_PROXY,COMM_POOLER,VACUUM,JOB,SPI,NEST_COMPILE,RESOWNER,LOGICAL_
DECODE,GPRC),off()
(1 row)
```

opfusion_debug_mode

参数说明：用于调试简单查询是否进行查询优化。设置成 log 级别可以在数据库节点的执行计划中看到没有查询优化的具体原因。

该参数属于 USERSET 类型参数，请参考：配置运行参数->重设参数章节中表 1 对应设置方法进行设置。

取值范围：枚举类型

- off 表示不打开该功能。
- log 表示打开该功能，可以在数据库节点的执行计划中看到没有查询优化的具体原因。

【须知】

提供在 log 中显示语句没有查询优化的具体原因，需要将参数设置成 log 级别，log_min_messages 设置成 debug4 级别，logging_module 设置'on(OPFUSION)'，注意 log 内容可能会比较多，尽可能在调优期间执行少量作业使用。

默认值：off

enable_debug_vacuum

参数说明：允许输出一些与 VACUUM 相关的日志，便于定位 VACUUM 相关问题。开发人员专用，不建议普通用户使用。

该参数属于 SIGHUP 类型参数，请参考：配置运行参数->重设参数章节中表 1 对应设置方法进行设置。

取值范围：布尔型

- on/true 表示开启此日志开关。
- off/false 表示关闭此日志开关。

默认值：off

12.9.4.使用 CSV 格式写日志

前提条件

- GUC 参数说明->错误报告和日志->记录日志的位置中 log_destination 的值设置为 csvlog。
- GUC 参数说明->错误报告和日志->记录日志的位置中 logging_collector 的值设置为 on。

csvlog 定义

以“逗号分隔值”即 CSV（Comma Separated Value）的形式发出日志。

以下是简单的用来存储 CSV 形式日志输出的表定义：

```
CREATE TABLE gaussdb_log
(
log_time timestamp(3) with time zone,
node_name text,
user_name text,
```

```
database_name text,
process_id bigint,
connection_from text,
"session_id" text,
session_line_num bigint,
command_tag text,
session_start_time timestamp with time zone,
virtual_transaction_id text,
transaction_id bigint,
query_id bigint,
module text,
error_severity text,
sql_state_code text,
message text,
detail text,
hint text,
internal_query text,
internal_query_pos integer,
context text,
query text,
query_pos integer,
location text,
application_name text
);
```

详细说明请参见表 1。

表 1 csvlog 字段含义表

字段名	字段含义	字段名	字段含义
log_time	毫秒级的时间戳	module	日志所属模块
node_name	节点名称	error_severity	ERRORSTATE 代码
user_name	用户名	sql_state_code	SQLSTATE 代码
database_name	数据库名	message	错误消息
process_id	进程 ID	detail	详细错误消息

字段名	字段含义	字段名	字段含义
connection_from	客户主机:端口号	hint	提示
session_id	会话 ID	internal_query	内部查询（查询那些导致错误的信息，如果有的话）
session_line_num	每个会话的行数	internal_query_pos	内部查询指针
command_tag	命令标签	context	环境
session_start_time	会话开始时间	query	错误发生位置的字符统计
virtual_transaction_id	常规事务	query_pos	错误发生位置指针
transaction_id	事务 ID	location	在 openGauss 源代码中错误的位置（如果 GUC 参数说明->错误报告和日志->记录日志的内容中 log_error_verbosity 参数的值设为 verbose）
query_id	查询 ID	application_name	应用名称

使用 COPY FROM 命令将日志文件导入这个表：

```
COPY gaussdb_log FROM '/home/panweidb/data/panweidb/pg_log/logfile.csv' WITH csv;
```

【说明】

此处的日志名“logfile.csv”要换成实际生成的日志的名称。

简化输入

简化输入到 CSV 日志文件，可以通过如下操作：

- 设置 GUC 参数说明->错误报告和日志->记录日志的位置中 log_filename 和 log_rotation_age，为日志文件提供一个一致的、可预测的命名方案。通过日志文件名，预测一个独立的日志文件完成并进入准备导入状态的时间。

- 将 GUC 参数说明->错误报告和日志->记录日志的位置中 log_rotation_size 设为 0 来终止基于尺寸的日志回滚，因为基于尺寸的日志回滚让预测日志文件名变得非常的困难。
- 将 GUC 参数说明->错误报告和日志->记录日志的位置中 log_truncate_on_rotation 设为 on 以便区分在同一日志文件中旧的日志数据和新的日志数据。

12.10.告警检测

在 PanWeiDB 运行的过程中，会对数据库中的错误场景进行检测，便于用户及早感知到 PanWeiDB 的错误。

enable_alarm

参数说明：允许打开告警检测线程，检测数据库中可能的错误场景。

该参数属于 POSTMASTER 类型参数，请参考：配置运行参数->重设参数章节中表 1 对应设置方法进行设置。

alarm 告警会按照环境变量 GAUSS_WARNING_TYPE 的配置进行输出：

1、输出到 alarm_component 指定的告警组件，由组件进行进一步处理和输出。

2、输出到操作系统 syslog。

取值范围：布尔型

- on 表示允许打开告警检测线程。
- off 表示不允许打开告警检测线程。

默认值：off

connection_alarm_rate

参数说明：允许和数据库连接的最大并发连接数的比率限制。数据库连接的最大并发连接数为 max_connections（参考：GUC 参数说明->连接和认证->连接设置）* connection_alarm_rate。

该参数属于 SIGHUP 类型参数，请参考：配置运行参数->重设参数章节中表 1 对应设置方法进行设置。

取值范围：浮点型，0.0~1.0

默认值：0.9

alarm_report_interval

参数说明：指定告警上报的时间间隔。

该参数属于 SIGHUP 类型参数，请参考：配置运行参数->重设参数章节中表 1 对应设置方法进行设置。

取值范围：整型，单位为秒。

默认值：10

alarm_component

参数说明：在对告警做上报时，会进行告警抑制，即同一个实例的同一个告警项在 alarm_report_interval（默认值为 10s）内不做重复上报。在这种情况下设置用于处理告警内容的告警组件的位置。

该参数属于 POSTMASTER 类型参数，请参考：配置运行参数->重设参数章节中表 1 对应设置方法进行设置。

取值范围：字符串

默认值：/opt/snas/bin/snas_cm_cmd

table_skewness_warning_threshold

参数说明：设置用于表倾斜告警的阈值。

该参数属于 USERSET 类型参数，请参考：配置运行参数->重设参数章节中表 1 对应设置方法进行设置。

取值范围：浮点型，0～1

默认值：1

table_skewness_warning_rows

参数说明：设置用于表倾斜告警的行数。

该参数属于 USERSET 类型参数，请参考：配置运行参数->重设参数章节中表 1 对应设置方法进行设置。

取值范围：整型，0～INT_MAX

默认值：100000

12.11.运行时统计

12.11.1.查询和索引统计收集器

查询和索引统计收集器负责收集数据库系统运行中的统计数据，如在一个表和索引上进行了多少次插入与更新操作、磁盘块的数量和元组的数量、每个表上最近一次执行清理和分析操作的时间等。可以通过查询系统视图 pg_stats 和 pg_statistic 查看统计数据。下面的参数设置服务器范围内的统计收集特性。

enable_functional_dependency

参数说明： 控制 ANALYZE 生成的多列统计信息是否包含函数依赖统计信息，是否应用函数依赖统计信息计算选择率。如果关闭该参数，每组多列统计信息最多支持 32 列；开启该参数则每组多列统计信息最多支持 4 列。

该参数属于 USERSET 类型参数，请参考重设参数表 1 中对应设置方法进行设置。

取值范围： 布尔型

- on 包含两个功能
  - 执行 ANALYZE 生成的多列统计信息包含函数依赖统计信息。
  - 计算选择率会使用函数依赖统计信息。

- off 包含两个功能：
  - 执行 ANALYZE 生成的多列统计信息不包含函数依赖统计信息。
  - 计算选择率不会使用函数依赖统计信息。

默认值： off

track_activities

参数说明：控制收集每个会话中当前正在执行命令的统计数据。

该参数属于 SUSET 类型参数，请参考：配置运行参数->重设参数章节中表 1 对应设置方法进行设置。

取值范围：布尔型

- on 表示开启收集功能。
- off 表示关闭收集功能。

默认值：on

track_counts

参数说明：控制收集数据库活动的统计数据。

该参数属于 SUSET 类型参数，请参考：配置运行参数->重设参数章节中表 1 对应设置方法进行设置。

取值范围：布尔型

- on 表示开启收集功能。
- off 表示关闭收集功能。

默认值：on

【说明】

在 autoVacuum 自动清理进程中选择清理的数据库时，需要数据库的统计数据，故默认值设为 on。

track_io_timing

参数说明：控制收集数据库 I/O 调用时序的统计数据。I/O 时序统计数据可以在 pg_stat_database 中查询。

该参数属于 SUSET 类型参数，请参考：配置运行参数->重设参数章节中表 1 对应设置方法进行设置。

取值范围：布尔型

- on 表示开启收集功能，开启时，收集器会在重复地去查询当前时间的操作系统，这可能会引起某些平台的重大开销，故默认值设置为 off。
- off 表示关闭收集功能。

默认值：off

track_functions

参数说明：控制收集函数的调用次数和调用耗时的统计数据。

该参数属于 SUSET 类型参数，请参考：配置运行参数->重设参数章节中表 1 对应设置方法进行设置。

【须知】
当 SQL 语言函数设置为调用查询的“内联”函数时，不管是否设置此选项，这些 SQL 语言函数无法被追踪到。

取值范围：枚举类型

- pl 表示只追踪过程语言函数。
- all 表示追踪 SQL 语言函数。
- none 表示关闭函数追踪功能。

默认值：none

track_activity_query_size

参数说明：设置用于跟踪每一个活动会话的当前正在执行命令的字节数。

该参数属于 POSTMASTER 类型参数，请参考：配置运行参数->重设参数章节中表 1 对应设置方法进行设置。

取值范围：整型，100～102400

默认值：1024

stats_temp_directory

参数说明：设置存储临时统计数据的目录。

该参数属于 SIGHUP 类型参数，请参考：配置运行参数->重设参数章节中表 1 对应设置方法进行设置。

【须知】
将其设置为一个基于 RAM 的文件系统目录会减少实际的 I/O 开销并可以提升其性能。

取值范围：字符串

默认值：pg_stat_tmp

track_thread_wait_status_interval

参数说明：用来定期收集 thread 状态信息的时间间隔。

该参数属于 SUSET 类型参数，请参考：配置运行参数->重设参数章节中表 1 对应设置方法进行设置。

取值范围：0～1 天，单位为 min。

默认值：30min

enable_save_datachanged_timestamp

参数说明：确定是否收集 insert/update/delete, exchange/truncate/drop partition 操作对表数据改动的时间。

该参数属于 USERSET 类型参数，请参考：配置运行参数->重设参数章节中表 1 对应设置方法进行设置。

取值范围：布尔型

- on 表示允许收集相关操作对表数据改动的时间。
- off 表示禁止收集相关操作对表数据改动的时间。

默认值：on

track_sql_count

参数说明：控制对每个会话中当前正在执行的 SELECT、INSERT、UPDATE、DELETE、MERGE INTO 语句进行计数的统计数据。

在 x86 平台集中式部署下，硬件配置规格为 32 核 CPU/256GB 内存，使用 Benchmark SQL 5.0 工具测试性能，开关此参数性能影响约 0.8%。

该参数属于 SUSET 类型参数，请参考：配置运行参数->重设参数章节中表 1 对应设置方法进行设置。

取值范围：布尔型

- on 表示开启计数功能。
- off 表示关闭计数功能。

默认值：on

【说明】

- track_sql_count 参数受 track_activities 约束：
  - track_activities 开启而 track_sql_count 关闭时，如果查询了 gs_sql_count 视图，日志中将会有 WARNING 提示 track_sql_count 是关闭的；
  - track_activities 和 track_sql_count 同时关闭，那么此时日志中将会有两条 WARNING，分别提示 track_activities 是关闭的和 track_sql_count 是关闭的；

- track_activities 关闭而 track_sql_count 开启，此时日志中将仅有 WARNING 提示 track_activities 是关闭。
- 当参数关闭时，查询视图的结果为 0 行。

12.11.2.性能统计

在数据库运行过程中，会涉及到锁的访问、磁盘 IO 操作、无效消息的处理，这些操作都可能是数据库的性能瓶颈，通过 PanWeiDB 提供的性能统计方法，可以方便定位性能问题。

输出性能统计日志

参数说明：对每条查询，以下 4 个选项控制在服务器日志里记录相应模块的性能统计数据，具体含义如下：

- log_parser_stats 控制在服务器日志里记录解析器的性能统计数据。
- log_planner_stats 控制在服务器日志里记录查询优化器的性能统计数据。
- log_executor_stats 控制在服务器日志里记录执行器的性能统计数据。
- log_statement_stats 控制在服务器日志里记录整个语句的性能统计数据。

这些参数只能辅助管理员进行粗略分析，类似 Linux 中的操作系统工具 getrusage() 。

这些参数属于 SUSET 类型参数，请参考：配置运行参数->重设参数章节中表 1 对应设置方法进行设置。

【须知】

- log_statement_stats 记录总的语句统计数据，而其他的只记录针对每个模块的统计数据。
- log_statement_stats 不能和其他任何针对每个模块统计的选项一起打开。

取值范围：布尔型

- on 表示开启记录性能统计数据的功能。

❖ off 表示关闭记录性能统计数据的功能。

默认值：off

12.12.负载管理

未对数据库资源做控制时，容易出现并发任务抢占资源导致操作系统过载甚至最终崩溃。操作系统过载时，其响应用户任务的速度会变慢甚至无响应；操作系统崩溃时，整个系统将无法对用户提供任何服务。PanWeiDB 的负载管理功能能够基于可用资源的多少均衡数据库的负载，以避免数据库系统过载。

use_workload_manager

参数说明：是否开启资源管理功能。

该参数属于 SIGHUP 类型参数，请参考：配置运行参数->重设参数章节中表 2 对应设置方法进行设置。

取值范围：布尔型

❖ on 表示打开资源管理。
❖ off 表示关闭资源管理。

【说明】

❖ 当使用：配置运行参数->重设参数章节中表 2 中的方式二来修改参数值时，新参数值只能对更改操作执行后启动的线程生效。此外，对于后台线程以及线程复用执行的新作业，该参数值的改动不会生效。如果希望这类线程即时识别参数变化，可以使用 kill session 或重启节点的方式来实现。
❖ use_workload_manager 参数由 off 变为 on 状态后，不会统计 off 时的存储资源。如果需要统计 off 时用户使用的存储资源，请在数据库中执行以下命令：select gs_wlm_readjust_user_space(0);

默认值：on

cgroup_name

参数说明：设置当前使用的 Cgroups 的名称或者调整当前 group 下排队的优先级。

即如果先设置 cgroup_name，再设置 session_respool，那么 session_respool 关联的控制组起作用，如果再切换 cgroup_name，那么新切换的 cgroup_name 起作用。

切换 cgroup_name 的过程中如果指定到 Workload 控制组级别，数据库不对级别进行验证。级别的范围只要在 1-10 范围内都可以。

该参数属于 USERSET 类型参数，请参考：配置运行参数->重设参数章节中表 2 中方式三的方法进行设置。

建议尽量不要混合使用 cgroup_name 和 session_respool。

取值范围：字符串

默认值：InvalidGroup

cpu_collect_timer

参数说明：设置语句执行时在数据库节点上收集 CPU 时间的周期。

数据库管理员需根据系统资源（如 CPU 资源、IO 资源和内存资源）情况，调整此数值大小，使得系统支持较合适的收集周期，太小会影响执行效率，太大会影响异常处理的精确度。

该参数属于 SIGHUP 类型参数，请参考：配置运行参数->重设参数章节中表 2 对应设置方法进行设置。

取值范围：整型，1～INT_MAX， 单位为秒。

默认值：30

memory_tracking_mode

参数说明：设置记录内存信息的模式。

该参数属于 USERSET 类型参数，请参考：配置运行参数->重设参数章节中表 2 对应设置方法进行设置。

取值范围：

- none：表示不启动内存统计功能。
- peak：表示统计 query 级内存 peak 值，此数值计入数据库日志，也可以由 explain analyze 输出。
- normal：表示仅做内存实时统计，不生成文件。
- executor：表示生成统计文件，包含执行层使用过的所有已分配内存的上下文信息。
- fullexec：表示生成文件包含执行层申请过的所有内存上下文信息。

默认值：none

memory_detail_tracking

参数说明：设置需要的线程内分配内存上下文的顺序号以及当前线程所在 query 的 plannodeid。

该参数属于 USERSET 类型参数，请参考：配置运行参数->重设参数章节中表 2 对应设置方法进行设置。

取值范围：字符型

默认值：空

【须知】
该参数不允许用户进行设置，建议保持默认值。

enable_resource_track

参数说明：是否开启资源实时监控功能。

该参数属于 SIGHUP 类型参数，请参考：配置运行参数->重设参数章节中表 2 对应设置方法进行设置。

取值范围：布尔型

- on 表示打开资源监控。
- off 表示关闭资源监控。

默认值：on

enable_resource_record

参数说明：是否开启资源监控记录归档功能。

该参数属于 SIGHUP 类型参数，请参考：配置运行参数->重设参数章节中表 2 对应设置方法进行设置。

取值范围：布尔型

- on 表示开启资源监控记录归档功能。
- off 表示关闭资源监控记录归档功能。

默认值：off

enable_logical_io_statistics

参数说明：设置是否开启资源监控逻辑 IO 统计功能。开启时，对于 PG_TOTAL_USER_RESOURCE_INFO 视图中的 read_kbytes、write_kbytes、read_counts、write_counts、read_speed 和 write_speed 字段，会统计对应用户的逻辑读写字节数、次数以及速率。

该参数属于 SIGHUP 类型参数，请参考：配置运行参数->重设参数章节中表 2 对应设置方法进行设置。

取值范围：布尔型

- on 表示开启资源监控逻辑 IO 统计功能。
- off 表示关闭资源监控逻辑 IO 统计功能。

默认值：on

enable_user_metric_persistent

参数说明：设置是否开启用户历史资源监控转存功能。开启时，对于 PG_TOTAL_USER_RESOURCE_INFO 视图中数据，会定期采样保存到系统表和系统视图->系统表->GS_WLM_EC_OPERATOR_INFO 表中。

该参数属于 SIGHUP 类型参数，请参考：配置运行参数->重设参数章节中表 2 对应设置方法进行设置。

取值范围：布尔型

- on 表示开启用户历史资源监控转存功能。
- off 表示关闭用户历史资源监控转存功能。

默认值：on

user_metric_retention_time

参数说明：设置用户历史资源监控数据的保存天数。该参数仅在 enable_user_metric_persistent 为 on 时有效。

该参数属于 SIGHUP 类型参数，请参考：配置运行参数->重设参数章节中表 2 中的方法一和方法二进行设置。

取值范围：整型，0～3650，单位为天。

- 值等于 0 时，用户历史资源监控数据将永久保存。
- 值大于 0 时，用户历史资源监控数据将保存对应天数。

默认值：7

enable_instance_metric_persistent

参数说明：设置是否开启实例资源监控转存功能。开启时，对实例的监控数据会保存到系统表和系统视图->系统表->GS_WLM_INSTANCE_HISTORY 表中。

该参数属于 SIGHUP 类型参数，请参考：配置运行参数->重设参数章节中表 2 对应设置方法进行设置。

取值范围：布尔型

- on 表示开启实例资源监控转存功能。
- off 表示关闭实例资源监控转存功能。

默认值：on

instance_metric_retention_time

参数说明：设置实例历史资源监控数据的保存天数。该参数仅在 enable_instance_metric_persistent 为 on 时有效。

该参数属于 USERSET 类型参数，请参考：配置运行参数->重设参数章节中表 2 中的方法一和方法二进行设置。

取值范围：整型，0～3650，单位为天。

- 值等于 0 时，实例历史资源监控数据将永久保存。
- 值大于 0 时，实例历史资源监控数据将保存对应设置天数。

默认值：7

resource_track_level

参数说明：设置当前会话的资源监控的等级。该参数只有当参数 enable_resource_track 为 on 时才有效。

该参数属于 USERSET 类型参数，请参考：配置运行参数->重设参数章节中表 2 对应设置方法进行设置。

取值范围：枚举型

- none：表示不开启资源监控功能。
- query：表示开启 query 级别资源监控功能。
- operator：表示开启 query 级别和算子级别资源监控功能。

默认值：query

resource_track_cost

参数说明：设置对当前会话的语句进行资源监控的最小执行代价。该参数只有当参数 enable_resource_track 为 on 时才有效。

该参数属于 USERSET 类型参数，请参考：配置运行参数->重设参数章节中表 2 对应设置方法进行设置。

取值范围：整型，-1～INT_MAX

- 值为-1 时，不进行资源监控。
- 值大于或等于 0 且小于等于 9 时，对执行代价大于等于 10 的语句进行资源监控。
- 值大于或等于 10 时，对执行代价超过该参数值的语句进行资源监控。

默认值：100000

resource_track_duration

参数说明：设置资源监控实时视图中记录的语句执行结束后进行历史信息转存的最小执行时间。当执行完成的作业，其执行时间不小于此参数值时，作业信息会从实时视图（以 statistics 为后缀的视图）转存到相应的历史视图（以 history 为后缀的视图）中。该参数只有当 enable_resource_track 为 on 时才有效。

该参数属于 USERSET 类型参数，请参考：配置运行参数->重设参数章节中表 2 对应设置方法进行设置。

取值范围：整型，0～INT_MAX，单位为秒。

- 值为 0 时，资源监控实时视图中记录的所有语句都进行历史信息归档。
- 值大于 0 时，资源监控实时视图中记录的语句的执行时间超过这个值就会进行历史信息归档。

默认值：1min

disable_memory_protect

参数说明：设置是否禁止内存保护功能。当系统内存不足时如果需要查询系统视图，可以先将此参数置为 on，禁止内存保护功能，保证视图可以正常查询。

该参数只适用于在系统内存不足时进行系统诊断和调试，正常运行时请保持该参数配置为 off。

该参数属于 USERSET 类型参数，且只对当前会话有效。请参考：配置运行参数->重设参数章节中表 2 对应设置方法进行设置。

取值范围：布尔型

- on 表示禁止内存保护功能。
- off 表示启动内存保护功能。

默认值：off

query_band

参数说明：用于标示当前会话的作业类型，由用户自定义。

该参数属于 USERSET 类型参数，请参考：配置运行参数->重设参数章节中表 2 对应设置方法进行设置。

取值范围：字符型

默认值：空

memory_fault_percent

参数说明：内存故障测试时内存申请失败的比例，仅用在 DEBUG 版本。

该参数属于 SIGHUP 类型参数，请参考：配置运行参数->重设参数章节中表 2 对应设置方法进行设置。

取值范围：整型，0～2147483647

默认值：0

enable_bbox_dump

参数说明：是否开启黑匣子功能，在系统不配置 core 机制的时候仍可产生 core 文件。

该参数属于 SIGHUP 类型参数，请参考：配置运行参数->重设参数章节中表 2 对应设置方法进行设置。

取值范围：布尔型

- on 表示打开黑匣子功能。
- off 表示关闭黑匣子功能。

默认值：on

【须知】
黑匣子功能生成 core 文件依赖操作系统开放 ptrace 接口。若发生权限不足(errno = 1)，请确保/proc/sys/kernel/yama/ptrace_scope 配置合理。

bbox_dump_count

参数说明：在 bbox_dump_path 定义的路径下，允许存储的 PanWeiDB 所产生 core 文件最大数。超过此数量，旧的 core 文件会被删除。此参数只有当 enable_bbox_dump 为 on 时才生效。

该参数属于 USERSET 类型参数，请参考：配置运行参数->重设参数章节中表 2 对应设置方法进行设置。

取值范围：整型，1～20

默认值：8

【说明】
在并发产生 core 文件时，core 文件的产生个数可能大于 bbox_dump_count。

bbox_dump_path

参数说明：黑匣子 core 文件的生成路径。此参数只有当 enable_bbox_dump 为 on 时才生效。

该参数属于 SIGHUP 类型参数，请参考：配置运行参数->重设参数章节中表 2 对应设置方法进行设置。

取值范围：字符型

默认值：空。默认生成黑匣子 core 文件的路径为读取 /proc/sys/kernel/core_pattern 下的路径，如果这个路径不是一个目录，或者用户对此目录没有写权限，黑匣子 core 文件将生成在数据库的 data 目录下。或者以安装时指定的目录为准。

enable_ffic_log

参数说明：是否开启 FFIC（First Failure Info Capture）功能。

该参数属于 POSTMASTER 类型参数，请参考：配置运行参数->重设参数章节中表 2 对应设置方法进行设置。

取值范围：布尔型

- on 表示打开 FFIC 功能。
- off 表示关闭 FFIC 功能。

默认值：on

io_limits

参数说明：每秒触发 IO 的上限。

该参数属于 USERSET 类型参数，请参考：配置运行参数->重设参数章节中表 2 对应设置方法进行设置。

取值范围：整型，0～1073741823

默认值：0

io_priority

参数说明：IO 利用率高达 50%时，重消耗 IO 作业进行 IO 资源管控时关联的优先级等级。

该参数属于 USERSET 类型参数，请参考：配置运行参数->重设参数章节中表 2 对应设置方法进行设置。

取值范围：枚举型

- None：表示不受控。
- Low：表示限制 iops 为该作业原始触发数值的 10%。
- Medium：表示限制 iops 为该作业原始触发数值的 20%。
- High：表示限制 iops 为该作业原始触发数值的 50%。

默认值：None

io_control_unit

参数说明：行存场景下，io 管控时用来对 io 次数进行计数的单位。

该参数属于 SIGHUP 类型参数，请参考：配置运行参数->重设参数章节中表 2 中对应类型的设置方法进行设置。

记多少次 io 触发为一计数单位，通过此计数单位所记录的次数进行 io 管控。

取值范围：整型，1000～1000000

默认值：6000

session_respool

参数说明：当前的 session 关联的 resource pool。

该参数属于 USERSET 类型参数，请参考：配置运行参数->重设参数章节中表 2 中对应类型的设置方法进行设置。

即如果先设置 cgroup_name，再设置 session_respool，那么 session_respool 关联的控制组起作用，如果再切换 cgroup_name，那么新切换的 cgroup_name 起作用。

切换 cgroup_name 的过程中如果指定到 Workload 控制组级别，数据库不对级别进行验证。级别的范围只要在 1-10 范围内都可以。

建议尽量不要混合使用 cgroup_name 和 session_respool。

取值范围：string 类型，通过 create resource pool 所设置的资源池。

默认值：invalid_pool

session_statistics_memory

参数说明：设置实时查询视图的内存大小。

该参数属于 SIGHUP 类型参数，请参考：配置运行参数->重设参数章节中表 2 对应设置方法进行设置。

取值范围：整型，5 * 1024 ~ max_process_memory 的 50%，单位 KB。

默认值：5MB

topsql_retention_time

参数说明：设置历史 TopSQL 中 gs_wlm_operator_info 表中数据的保存时间。

该参数属于 SIGHUP 类型参数，请参考：配置运行参数->重设参数章节中表 2 对应设置方法进行设置。

取值范围：整型，0～3650，单位为天。

- 值为 0 时，表示数据永久保存。
- 值大于 0 时，表示数据能够保存的对应天数。

默认值：0

session_history_memory

参数说明：设置历史查询视图的内存大小。

该参数属于 SIGHUP 类型参数，请参考：配置运行参数->重设参数章节中表 2 对应设置方法进行设置。

取值范围：整型，10 * 1024 ~ max_process_memory 的 50%，单位 KB。

默认值：10MB

transaction_pending_time

参数说明：事务块语句和存储过程语句排队的最大时间。

该参数属于 USERSET 类型参数，请参考：配置运行参数->重设参数章节中表 1 对应设置方法进行设置。

取值范围：整型，-1 ~ INT_MAX/2，单位为秒。

- 值为-1 或 0 时，事务块语句和存储过程语句无超时判断，排队至资源满足可执行条件。
- 值大于 0 时，事务块语句和存储过程语句排队超过所设数值的时间后，无视当前资源情况强制执行。

默认值：0

12.13.自动清理

系统自动清理线程（autovacuum）自动执行 VACUUM 和 ANALYZE 命令，回收被标识为删除状态的记录空间，并更新表的统计数据。

autovacuum

参数说明：控制数据库自动清理线程（autovacuum）的启动。自动清理线程运行的前提是将 GUC 参数说明->运行时统计->查询和索引统计收集器中 track_counts 参数设置为 on。

该参数属于 SIGHUP 类型参数，请参考：配置运行参数->重设参数章节中表 2 对应设置方法进行设置。

【说明】

- 如果希望系统在故障恢复后，具备自动清理两阶段事务的功能，请将 autovacuum 设置为 on;

- 当设置 autovacuum 为 on，autovacuum_max_workers 为 0 时，表示系统不会自动进行 autovacuum，只会在故障恢复后，自动清理两阶段事务；
- 当设置 autovacuum 为 on，autovacuum_max_workers 大于 0 时，表示系统不仅在故障恢复后，自动清理两阶段事务，并且还可以自动清理线程。

取值范围：布尔型

- on 表示开启数据库自动清理线程。
- off 表示关闭数据库自动清理线程。

默认值：on

autovacuum_mode

参数说明：该参数仅在 autovacuum 设置为 on 的场景下生效，它控制 autoanalyze 或 autovacuum 的打开情况。

该参数属于 SIGHUP 类型参数，请参考：配置运行参数->重设参数章节中表 2 对应设置方法进行设置。

取值范围：枚举类型

- analyze 表示只做 autoanalyze。
- vacuum 表示只做 autovacuum。
- mix 表示 autoanalyze 和 autovacuum 都做。
- none 表示二者都不做。

默认值：mix

autoanalyze_timeout

参数说明：设置 autoanalyze 的超时时间。在对某张表做 autoanalyze 时，如果该表的 analyze 时长超过了 autoanalyze_timeout，则自动取消该表此次 analyze。

该参数属于 SIGHUP 类型参数，请参考：配置运行参数->重设参数章节中表 2 对应设置方法进行设置。

取值范围：整型，0~2147483，单位是秒。

默认值：5min（即 300s）

autovacuum_io_limits

参数说明：控制 autovacuum 线程每秒触发 IO 的上限。

该参数属于 SIGHUP 类型参数，请参考：配置运行参数->重设参数章节中表 2 对应设置方法进行设置。

取值范围：整型，0～1073741823 和-1。其中-1 表示不控制，而是使用系统默认控制组。

默认值：-1

log_autovacuum_min_duration

参数说明：当自动清理的执行时间大于或者等于某个特定的值时，向服务器日志中记录本次自动清理执行的概要信息。设置此选项有助于追踪自动清理的行为。

该参数属于 SIGHUP 类型参数，请参考：配置运行参数->重设参数章节中表 2 对应设置方法进行设置。

举例如下：

将 log_autovacuum_min_duration 设置为 250ms，记录所有运行大于或者等于 250ms 的自动清理命令的相关信息。

其相关日志表现如下：

```
automatic vacuum of table x: index scans: x
pages: x removed, x remain
tuples: x removed, x remain
buffer usage: x hits, x misses, x dirtied
avg read rate: x MiB/s, avg write rate: x MiB/s
system usage: x
```

取值范围：整型，最小值为-1，最大值为 2147483647，单位为毫秒。

- 当参数设置为 0 时，表示所有的自动清理的概要信息都记录到日志中。
- 当参数设置为-1 时，表示所有的自动清理的概要信息都不记录到日志中。
- 当参数设置为非-1、非 0 时，当由于锁冲突的存在导致一个自动清理操作被跳过，记录一条消息。

默认值：-1

autovacuum_max_workers

参数说明：设置能同时运行的自动清理线程的最大数量，该参数的取值上限与 GUC 参数 max_connections 和 job_queue_processes 大小有关。

该参数属于 POSTMASTER 类型参数，请参考：配置运行参数->重设参数章节中表 2 对应设置方法进行设置。

取值范围：整型，最小值为 0（表示不会自动进行 autovacuum），理论最大值为 262143，实际最大值为动态值，计算公式为“262143 - max_connections 的值 - job_queue_processes 的值 - 辅助线程数 - autovacuum 的 lancher 线程数 - 1”，其中辅助线程数和 autovacuum 的 lancher 线程数由两个宏来指定，当前版本的默认值分别为 20 和 2。

默认值：3

autovacuum_naptime

参数说明：设置两次自动清理操作的时间间隔。

该参数属于 SIGHUP 类型参数，请参考：配置运行参数->重设参数章节中表 2 对应设置方法进行设置。

取值范围：整型，单位为 s，最小值为 1，最大值为 2147483。

默认值：10min（即 600s）

autovacuum_vacuum_threshold

参数说明：设置触发 VACUUM 的阈值。当表上被删除或更新的记录数超过设定的阈值时才会对这个表执行 VACUUM 操作。

该参数属于 SIGHUP 类型参数，请参考：配置运行参数->重设参数章节中表 2 对应设置方法进行设置。

取值范围：整型，最小值为 0，最大值为 2147483647。

默认值：50

autovacuum_analyze_threshold

参数说明：设置触发 ANALYZE 操作的阈值。当表上被删除、插入或更新的记录数超过设定的阈值时才会对这个表执行 ANALYZE 操作。

该参数属于 SIGHUP 类型参数，请参考：配置运行参数->重设参数章节中表 2 对应设置方法进行设置。

取值范围：整型，最小值为 0，最大值为 2147483647。

默认值：50

autovacuum_vacuum_scale_factor

参数说明：设置触发一个 VACUUM 时增加到 autovacuum_vacuum_threshold 的表大小的缩放系数。

该参数属于 SIGHUP 类型参数，请参考：配置运行参数->重设参数章节中表 2 对应设置方法进行设置。

取值范围：浮点型，0.0 ~ 100.0

默认值：0.2

autovacuum_analyze_scale_factor

参数说明：设置触发一个 ANALYZE 时增加到 autovacuum_analyze_threshold 的表大小的缩放系数。

该参数属于 SIGHUP 类型参数，请参考：配置运行参数->重设参数章节中表 2 对应设置方法进行设置。

取值范围：浮点型，0.0 ~ 100.0

默认值：0.1

autovacuum_freeze_max_age

参数说明：设置事务内的最大时间，使得表的 pg_class.relfrozenxid 字段在 VACUUM 操作执行之前被写入。

- VACUUM 也可以删除 pg_clog/子目录中的旧文件。
- 即使自动清理线程被禁止，系统也会调用自动清理线程来防止循环重复。

该参数属于 POSTMASTER 类型参数，请参考：配置运行参数->重设参数章节中表 2 对应设置方法进行设置。

取值范围：长整型，100 000 ~ 576 460 752 303 423 487

默认值：4000000000

autovacuum_vacuum_cost_delay

参数说明：设置在自动 VACUUM 操作里使用的开销延迟数值。

该参数属于 SIGHUP 类型参数，请参考：配置运行参数->重设参数章节中表 2 对应设置方法进行设置。

取值范围：整型，-1～100，单位为毫秒（ms）。其中-1 表示使用常规的 vacuum_cost_delay。

默认值：20ms

autovacuum_vacuum_cost_limit

参数说明：设置在自动 VACUUM 操作里使用的开销限制数值。

该参数属于 SIGHUP 类型参数，请参考：配置运行参数->重设参数章节中表 2 对应设置方法进行设置。

取值范围：整型，-1～10000。其中-1 表示使用常规的 vacuum_cost_limit。

默认值：-1

defer_csn_cleanup_time

参数说明：用来指定本地回收时间间隔。

该参数属于 SIGHUP 类型参数，请参考：配置运行参数->重设参数章节中表 2 对应设置方法进行设置。

取值范围：整型，0~INT_MAX，单位为毫秒（ms）。

默认值：5s（即 5000ms）

12.14.客户端连接缺省设置

12.14.1.语句行为

介绍 SQL 语句执行过程的相关默认参数。

search_path

参数说明：当一个被引用对象没有指定模式时，此参数设置模式搜索顺序。它的值由一个或多个模式名构成，不同的模式名用逗号隔开。

该参数属于 USERSET 类型参数，请参考：配置运行参数->重设参数章节中表 2 对应设置方法进行设置。

- 当前会话如果存放临时表的模式时，可以使用别名 pg_temp 将它列在搜索路径中，如'pg_temp，public'。存放临时表的模式始终会作为第一个被搜索的对象，排在 pg_catalog 和 search_path 中所有模式的前面，即具有第一搜索优先级。建议用户不要在 search_path 中显示设置 pg_temp。如果在 search_path 中指定了 pg_temp，但不是在最前面，系统会提示设置无效，pg_temp 仍被优先搜索。通过使用别名 pg_temp，系统只会在存放临时表的模式中搜索表、视图和数据类型这样的数据库对象，不会在里面搜索函数或运算符这样的数据库对象。
- 系统表所在的模式 pg_catalog，总是排在 search_path 中指定的所有模式前面被搜索，即具有第二搜索优先级（pg_temp 具有第一搜索优先级）。建议用户不要在 search_path 中显式设置 pg_catalog。如果在 search_path 中指定了 pg_catalog，但不是在最前面，系统会提示设置无效，pg_catalog 仍被第二优先搜索。
- 当没有指定一个特定模式而创建一个对象时，它们被放置到以 search_path 为命名的第一个有效模式中。当搜索路径为空时，会报错误。
- 通过 SQL 函数 current_schema 可以检测当前搜索路径的有效值。这和检测 search_path 的值不尽相同，因为 current_schema 显示 search_path 中首位有效的模式名称。

取值范围：字符串

【说明】

- 设置为“$user”，public 时，支持共享数据库（没有用户具有私有模式和所有共享使用 public）、用户私有模式和这些功能的组合使用。可以通过改变默认搜索路径来获得其他效果，无论是全局化的还是私有化的。
- 设置为空串（''）的时候，系统会自动转换成一对双引号。

- 设置的内容中包含双引号，系统会认为是不安全字符，会将每个双引号转换成一对双引号。

默认值： “$user” ,public

【说明】
$user 表示与当前会话用户名同名的模式名，如果这样的模式不存在，$user 将被忽略。

current_schema

参数说明：此参数设置当前的模式。

该参数属于 USERSET 类型参数，请参考：配置运行参数->重设参数章节中表 2 对应设置方法进行设置。

取值范围：字符串

默认值："$user",public

【说明】
$user 表示与当前会话用户名同名的模式名，如果这样的模式不存在，$user 将被忽略。

default_tablespace

参数说明：当 CREATE 命令没有明确声明表空间时，所创建对象（表和索引等）的缺省表空间。

- 值是一个表空间的名称或者一个表示使用当前数据库缺省表空间的空字符串。若指定的是一个非默认表空间，用户必须具有它的 CREATE 权限，否则尝试创建会失败。
- 临时表不使用此参数，可以用 temp_tablespaces 代替。
- 创建数据库时不使用此参数。默认情况下，一个新的数据库从模板数据库继承表空间配置。
- 该参数属于 USERSET 类型参数，请参考：配置运行参数->重设参数章节中表 2 对应设置方法进行设置。

取值范围：字符串，其中空表示使用默认表空间。

默认值：空

temp_tablespaces

参数说明：当一个 CREATE 命令没有明确指定一个表空间时，temp_tablespaces 指定了创建临时对象（临时表和临时表的索引）所在的表空间。在这些表空间中创建临时文件用来做大型数据的排序工作。

其值是一系列表空间名的列表。如果列表中有多个表空间时，每次临时对象的创建，PanWeiDB 会在列表中随机选择一个表空间；如果在事务中，连续创建的临时对象被放置在列表里连续的表空间中。如果选择的列表中的元素是一个空串，PanWeiDB 将自动将当前的数据库设为默认的表空间。

该参数属于 USERSET 类型参数，请参考：配置运行参数->重设参数章节中表 2 对应设置方法进行设置。

取值范围：字符串。空字符串表示所有的临时对象仅在当前数据库默认的表空间中创建，请参见 default_tablespace。

默认值：空

check_function_bodies

参数说明：设置是否在 CREATE FUNCTION 执行过程中进行函数体字符串的合法性验证。为了避免产生问题（比如避免从转储中恢复函数定义时向前引用的问题），偶尔会禁用验证。

该参数属于 USERSET 类型参数，请参考：配置运行参数->重设参数章节中表 2 对应设置方法进行设置。

取值范围：布尔型

- on 表示在 CREATE FUNCTION 执行过程中进行函数体字符串的合法性验证。

- off 表示在 CREATE FUNCTION 执行过程中不进行函数体字符串的合法性验证。

默认值：on

default_transaction_isolation

参数说明：设置默认的事务隔离级别。

该参数属于 USERSET 类型参数，请参考：配置运行参数->重设参数章节中表 2 对应设置方法进行设置。

取值范围：枚举类型

- read committed 表示事务读已提交。
- repeatable read 表示事务可重复读。
- serializable，PanWeiDB 目前功能上不支持此隔离级别，等价于 repeatable read。

默认值：read committed

default_transaction_read_only

参数说明：设置每个新创建事务是否是只读状态。

该参数属于 USERSET 类型参数，请参考：配置运行参数->重设参数章节中表 2 对应设置方法进行设置。

取值范围：布尔型

- on 表示只读状态。
- off 表示非只读状态。

默认值：off

default_transaction_deferrable

参数说明：控制每个新事务的默认延迟状态。只读事务或者那些比序列化更加低的隔离级别的事务除外。

PanWeiDB 不支持可串行化的隔离级别，因此，该参数无实际意义。

该参数属于 USERSET 类型参数，请参考：配置运行参数->重设参数章节中表 2 对应设置方法进行设置。

取值范围：布尔型

- on 表示默认延迟。
- off 表示默认不延迟。

默认值：off

session_replication_role

参数说明：控制当前会话与复制相关的触发器和规则的行为。

该参数属于 SUSET 类型参数，请参考：配置运行参数->重设参数章节中表 2 对应设置方法进行设置。

【说明】
设置此参数会丢弃之前任何缓存的执行计划。

取值范围：枚举类型

- origin 表示从当前会话中复制插入、删除、更新等操作。
- replica 表示从其他地方复制插入、删除、更新等操作到当前会话。
- local 表示函数执行复制时会检测当前登录数据库的角色并采取相应的操作。

默认值：origin

statement_timeout

参数说明：当语句执行时间超过该参数设置的时间（从服务器收到命令时开始计时）时，该语句将会报错并退出执行。

该参数属于 USERSET 类型参数，请参考：配置运行参数->重设参数章节中表 2 对应设置方法进行设置。

取值范围：整型，最小值为 0，最大值为 2147483647，单位为毫秒。

默认值：0

vacuum_freeze_min_age

参数说明：指定 VACUUM 在扫描一个表时用于判断是否用 FrozenXID 替换事务 ID 的中断寿命（在同一个事务中）。

该参数属于 USERSET 类型参数，请参考：配置运行参数->重设参数章节中表 2 对应设置方法进行设置。

取值范围：整型，0～576 460 752 303 423 487

【说明】
尽管随时可以将此参数设为 0 到 10 亿之间的任意值，但是，VACUUM 将默认其有效值范围限制在 GUC 参数说明->自动清理中 autovacuum_freeze_max_age 参数的 50%以内。

默认值：2000000000

vacuum_freeze_table_age

参数说明：指定 VACUUM 对全表的扫描冻结元组的时间。如果表的 pg_class.relfrozenxid 字段的值已经达到了参数指定的时间，VACUUM 对全表进行扫描。

该参数属于 USERSET 类型参数，请参考：配置运行参数->重设参数章节中表 2 对应设置方法进行设置。

取值范围：整型，0～576 460 752 303 423 487

【说明】

尽管随时可以将此参数设为零到 20 亿之间的值，但是，VACUUM 将默认其有效值范围限制在 GUC 参数说明->自动清理中 autovacuum_freeze_max_age 参数的 95%以内。定期的手动 VACUUM 可以在对此表的反重叠自动清理启动之前运行。

默认值：4000000000

bytea_output

参数说明：设置 bytea 类型值的输出格式。

该参数属于 USERSET 类型参数，请参考：配置运行参数->重设参数章节中表 2 对应设置方法进行设置。

取值范围：枚举类型

- hex：将二进制数据编码为每字节 2 位十六进制数字。
- escape：传统化的 PostgreSQL 格式。采用以 ASCII 字符序列表示二进制串的方法，同时将那些无法表示成 ASCII 字符的二进制串转换成特殊的转义序列。

默认值：hex

xmlbinary

参数说明：设置二进制值是如何在 XML 中进行编码的。

该参数属于 USERSET 类型参数，请参考：配置运行参数->重设参数章节中表 2 对应设置方法进行设置。

取值范围：枚举类型

- base64
- hex

默认值：base64

xmloption

参数说明：当 XML 和字符串值之间进行转换时，设置 document 或 content 是否是隐含的。

该参数属于 USERSET 类型参数，请参考：配置运行参数->重设参数章节中表 2 对应设置方法进行设置。

取值范围：枚举类型

- document：表示 HTML 格式的文档。
- content：普通的字符串。

默认值：content

max_compile_functions

参数说明：设置服务器存储的函数编译结果的最大数量。存储过多的函数和存储过程的编译结果可能占用很大内存。将此参数设置为一个合理的值，有助于减少内存占用，提升系统性能。

该参数属于 POSTMASTER 类型参数，请参考：配置运行参数->重设参数章节中表 2 对应设置方法进行设置。

取值范围：整型，1~2147483647。

默认值：1000

gin_pending_list_limit

参数说明：设置当 GIN 索引启用 fastupdate 时，pending list 容量的最大值。当 pending list 的容量大于设置值时，会把 pending list 中数据批量移动到 GIN 索引数据结构中以进行清理。单个 GIN 索引可通过更改索引存储参数覆盖此设置值。

该参数属于 USERSET 类型参数，请参考：配置运行参数->重设参数章节中表 2 对应设置方法进行设置。

取值范围：整型，最小值为 64，最大值为 INT_MAX，单位为 KB。

默认值：4MB

pw_exclude_reserved_words

参数说明：设置此参数可以屏蔽部分 PanWeiDB 的关键字，使关键字可以作为字段名或别名。避免因表名（字段名）是 PanWeiDB 的关键字导致建表失败的问题。

该参数属于 POSTMASTER 类型参数，请参考：配置运行参数->重设参数章节中表 2 对应设置方法进行设置。

【说明】

可以设置多个关键字，用“,”隔开。

关键字被屏蔽后，相关关键字语法将同步失效。

某些关键字对于系统的正常运行非常重要，例如“select”。因此，及时参数指定了这些关键词，也不会真正生效。详见：《PanWeiDB V2.0-S3.0.1_B01 管理员指南》-> 数据库使用->其他操作->设置保留字（关键字）可以作为字段名或别名。

取值范围：字符串。

默认值：” ”

12.14.2.区域和格式化

介绍时间格式设置的相关参数。

DateStyle

参数说明：设置日期和时间值的显示格式，以及有歧义的输入值的解析规则。

这个变量包含两个独立的加载部分：输出格式声明（ISO、Postgres、SQL、German）和输入输出的年/月/日顺序（DMY、MDY、YMD）。这两个可以独立设置或者一起设置。关键字 Euro 和 European 等价于 DMY；关键字 US、NonEuro、NonEuropean 等价于 MDY。

该参数属于 USERSET 类型参数，请参考：配置运行参数->重设参数章节中表 2 对应设置方法进行设置。

取值范围：字符串

默认值：ISO,MDY

【说明】
pw_initdb 会将这个参数初始化成与 lc_time 一致的值。

设置建议：优先推荐使用 ISO 格式。Postgres、SQL 和 German 均采用字母缩写的形式来表示时区，例如“EST、WST、CST”等。这些缩写可同时指代不同的时区，比如 CST 可同时代表美国中部时间（Central Standard Time (USA) UT-6:00）、澳大利亚中部时间（Central Standard Time (Australia) UT+9:30）、中国标准时间（China Standard Time UT+8:00）、古巴标准时间（Cuba Standard Time UT-4:00）。这种情况下在时区转化时可能会得不到正确的结果，从而引发其他问题。

IntervalStyle

参数说明：设置区间值的显示格式。

该参数属于 USERSET 类型参数，请参考：配置运行参数->重设参数章节中表 2 对应设置方法进行设置。

取值范围：枚举类型

- sql_standard 表示产生与 SQL 标准规定匹配的输出。
- postgres 表示产生与 PostgreSQL 8.4 版本相匹配的输出，当 DateStyle 参数被设为 ISO 时。
- postgres_verbose 表示产生与 PostgreSQL 8.4 版本相匹配的输出，当 DateStyle 参数被设为 non_ISO 时。

- iso_8601 表示产生与在 ISO 8601 中定义的“格式与代号”相匹配的输出。
- a 表示与 numtodsinterval 函数相匹配的输出结果，详细请参考：SQL 语法参考->SQL 基本要素->函数和操作符->时间和日期处理函数和操作符中 numtodsinterval。

【须知】
IntervalStyle 参数也会影响不明确的间隔输入的说明。

默认值：postgres

TimeZone

参数说明：设置显示和解释时间类型数值时使用的时区。

该参数属于 USERSET 类型参数，请参考：配置运行参数->重设参数章节中表 2 对应设置方法进行设置。

取值范围：字符串，可查询视图系统表和系统视图->系统视图->PG_TIMEZONE_NAMES 获得。

默认值：GMT

【说明】
pw_initdb 将设置一个与其系统环境一致的时区值。

timezone_abbreviations

参数说明：设置服务器接受的时区缩写值。

该参数属于 USERSET 类型参数，请参考：配置运行参数->重设参数章节中表 2 对应设置方法进行设置。

取值范围：字符串，可查询视图 pg_timezone_names 获得。

默认值：Default

【说明】

Default 表示通用时区的缩写，适合绝大部分情况。但也可设置其他诸如 'Australia' 和 'India' 等用来定义特定的安装。而设置除此之外的时区缩写，需要在建数据库之前通过相应的配置文件进行设置。

extra_float_digits

参数说明：这个参数为浮点数值调整显示的数据位数，浮点类型包括 float4、float8 以及几何数据类型。参数值加在标准的数据位数上（FLT_DIG 或 DBL_DIG 中合适的）。

该参数属于 USERSET 类型参数，请参考：配置运行参数->重设参数章节中表 2 对应设置方法进行设置。

取值范围：整型，-15～3

📖 说明

- 设置为 3，表示包括部分关键的数据位。这个功能对转储那些需要精确恢复的浮点数据特别有用。
- 设置为负数，表示消除不需要的数据位。

默认值：0

client_encoding

参数说明：设置客户端的字符编码类型。

请根据前端业务的情况确定。尽量客户端编码和服务器端编码一致，提高效率。

该参数属于 USERSET 类型参数，请参考：配置运行参数->重设参数章节中表 2 对应设置方法进行设置。

取值范围：兼容 PostgreSQL 所有的字符编码类型。其中 UTF8 表示使用数据库的字符编码类型。

说明

- 使用命令 locale -a 查看当前系统支持的区域和相应的编码格式，并可以选择进行设置。
- 默认情况下，pw_initdb 会根据当前的系统环境初始化此参数，通过 locale 命令可以查看当前的配置环境。
- 参数建议保持默认值，不建议通过 pw_guc 工具或其他方式直接在 postgresql.conf 文件中设置 client_encoding 参数，即使设置也不会生效，以保证 PanWeiDB 内部通信编码格式一致。

默认值：UTF8

推荐值：SQL_ASCII/UTF8

lc_messages

参数说明：设置信息显示的语言。

- 可接受的值是与系统相关的。
- 在一些系统上，这个区域范畴并不存在，不过仍然允许设置这个变量，只是不会有任何效果。同样，也有可能是所期望的语言的翻译信息不存在。在这种情况下，用户仍然能看到英文信息。
- 该参数属于 SUSET 类型参数，请参考：配置运行参数->重设参数章节中表 2 对应设置方法进行设置。

取值范围：字符串

说明

- 使用命令 locale -a 查看当前系统支持的区域和相应的编码格式，并可以选择进行设置。

- 默认情况下，pw_initdb 会根据当前的系统环境初始化此参数，通过 locale 命令可以查看当前的配置环境。

默认值：en_US.utf8

lc_monetary

参数说明：设置货币值的显示格式，影响 to_char 之类的函数的输出。可接受的值是系统相关的。

该参数属于 USERSET 类型参数，请参考：配置运行参数->重设参数章节中表 2 对应设置方法进行设置。

取值范围：字符串

说明

- 用命令 locale -a 查看当前系统支持的区域和相应的编码格式，并可以选择进行设置。
- 默认情况下，pw_initdb 会根据当前的系统环境初始化此参数，通过 locale 命令可以查看当前的配置环境。

默认值：en_US.utf8

lc_numeric

参数说明：设置数值的显示格式，影响 to_char 之类的函数的输出。可接受的值是系统相关的。

该参数属于 USERSET 类型参数，请参考：配置运行参数->重设参数章节中表 2 对应设置方法进行设置。

取值范围：字符串

说明

- 使用命令 locale -a 查看当前系统支持的区域和相应的编码格式，并可以选择进行设置。
- 默认情况下，pw_initdb 会根据当前的系统环境初始化此参数，通过 locale 命令可以查看当前的配置环境。

默认值：en_US.utf8

lc_time

参数说明：设置时间和区域的显示格式，影响 to_char 之类的函数的输出。可接受的值是系统相关的。

该参数属于 USERSET 类型参数，请参考：配置运行参数->重设参数章节中表 2 对应设置方法进行设置。

取值范围：字符串

说明

- 使用命令 locale -a 查看当前系统支持的区域和相应的编码格式，并可以选择进行设置。
- 默认情况下，pw_initdb 会根据当前的系统环境初始化此参数，通过 locale 命令可以查看当前的配置环境。

默认值：en_US.utf8

default_text_search_config

参数说明：设置全文检索的配置信息。

如果设置为不存在的文本搜索配置时将会报错。如果 default_text_search_config 对应的文本搜索配置被删除，需要重新设置 default_text_search_config，否则会报设置错误。

- 其被文本搜索函数使用，这些函数并没有一个明确指定的配置。
- 当与环境相匹配的配置文件确定时，pw_initdb 会选择一个与环境相对应的设置来初始化配置文件。
- 该参数属于 USERSET 类型参数，请参考：配置运行参数->重设参数章节中表 2 对应设置方法进行设置。

取值范围：字符串

📖 说明

PanWeiDB 支持 pg_catalog.english、pg_catalog.simple 两种配置。

默认值：pg_catalog.english

12.14.3.其他缺省

主要介绍数据库系统默认的库加载参数。

dynamic_library_path

参数说明：设置数据查找动态加载的共享库文件的路径。当需要打开一个可以动态装载的模块并且在 CREATE FUNCTION 或 LOAD 命令里面声明的名称没有目录部分时，系统将搜索这个目录以查找声明的文件。

用于 dynamic_library_path 的数值必须是一个冒号分隔的绝对路径列表。当一个路径名称以特殊变量$libdir 为开头时，会替换为 PanWeiDB 发布提供的模块安装路径。例如：

```
dynamic_library_path = '/usr/local/lib/gaussdb:/opt/testgs/lib:$libdir'
```

该参数属于 SUSET 类型参数，请参考：配置运行参数->重设参数章节中表 2 对应设置方法进行设置。

取值范围：字符串

说明：

设置为空字符串，表示关闭自动路径搜索。

默认值：$libdir

gin_fuzzy_search_limit

参数说明：设置 GIN 索引返回的集合大小的上限。

该参数属于 USERSET 类型参数，请参考：配置运行参数->重设参数章节中表 2 对应设置方法进行设置。

取值范围：整型，0~2147483647

默认值：0

local_preload_libraries

参数说明：指定一个或多个共享库，它们在开始连接前预先加载。多个加载库之间用逗号分隔，除了双引号，所有的库名都转换为小写。

- 并非只有系统管理员才能更改此选项，因此只能加载安装的标准库目录下 plugins 子目录中的库文件，数据库管理员有责任确保该目录中的库都是安全的。local_preload_libraries 中指定的项可以明确含有该目录，例如 $libdir/plugins/mylib；也可以仅指定库的名称，例如 mylib（等价于 $libdir/plugins/mylib）。
- 与 shared_preload_libraries 不同，在会话开始之前加载模块与在会话中使用到该模块的时候临时加载相比并不具有性能优势。相反，这个特性的目的是为了调试或者测量在特定会话中不明确使用 LOAD 加载的库。例如针对某个用户将该参数设为 ALTER USER SET 来进行调试。
- 当指定的库未找到时，连接会失败。
- 每一个支持 PanWeiDB 的库都有一个“magic block”用于确保兼容性，因此不支持 PanWeiDB 的库不能通过这个方法加载。

该参数属于 BACKEND 类型参数，请参考：配置运行参数->重设参数章节中表 2 对应设置方法进行设置。

取值范围：字符串

默认值：空

12.15.锁管理

在 PanWeiDB 中，并发执行的事务由于竞争资源会导致死锁。本节介绍的参数主要管理事务锁的机制。

deadlock_timeout

参数说明：设置死锁超时检测时间，以毫秒为单位。当申请的锁超过设定值时，系统会检查是否产生了死锁。该参数仅针对常规锁生效。

- 死锁的检查代价是比较高的，服务器不会在每次等待锁的时候都运行这个过程。在系统运行过程中死锁是不经常出现的，因此在检查死锁前只需等待一个相对较短的时间。增加这个值就减少了无用的死锁检查浪费的时间，但是会减慢真正的死锁错误报告的速度。在一个负载过重的服务器上，用户可能需要增大它。这个值的设置应该超过事务持续时间，这样就可以减少在锁释放之前就开始死锁检查的问题。
- 当设置 GUC 参数说明->错误报告和日志->记录日志的内容中 log_lock_waits 参数为 on 时，deadlock_timeout 决定一个等待时间来将查询执行过程中的锁等待耗时信息写入日志。如果要研究锁延时情况，可以设置 deadlock_timeout 的值比正常情况小。

该参数属于 SUSET 类型参数，请参考：配置运行参数->重设参数章节中表 2 对应设置方法进行设置。

取值范围：整型，1~2147483647，单位为毫秒（ms）。

默认值：1s

lockwait_timeout

参数说明：控制单个锁的最长等待时间。当申请的锁等待时间超过设定值时，系统会报错。该参数仅针对常规锁生效。

该参数属于 SUSET 类型参数，请参考：配置运行参数->重设参数章节中表 2 对应设置方法进行设置。

取值范围：整型，0～INT_MAX，单位为毫秒（ms）。

默认值：20min

update_lockwait_timeout

参数说明：允许并发更新参数开启情况下，该参数控制并发更新同一行时单个锁的最长等待时间。当申请的锁等待时间超过设定值时，系统会报错。该参数仅针对常规锁生效。

该参数属于 SUSET 类型参数，请参考：配置运行参数->重设参数章节中表 2 对应设置方法进行设置。

取值范围：整型，0～INT_MAX，单位为毫秒（ms）。

默认值：20min

max_locks_per_transaction

参数说明：控制每个事务能够得到的平均的对象锁的数量。

- 共享的锁表的大小是以假设任意时刻最多只有 max_locks_per_transaction*(max_connections+max_prepared_transactions) 个独立的对象需要被锁住为基础进行计算的。不超过设定数量的多个对象可以在任一时刻同时被锁定。当在一个事务里面修改很多不同的表时，可能需要提高这个默认数值。只能在数据库启动的时候设置。
- 增大这个参数可能导致 PanWeiDB 请求更多的 System V 共享内存，有可能超过操作系统的缺省配置。
- 当运行备机时，请将此参数设置不小于主机上的值，否则，在备机上查询操作不会被允许。

该参数属于 POSTMASTER 类型参数，请参考：配置运行参数->重设参数章节中表 2 对应设置方法进行设置。

取值范围：整型，10 ~ INT_MAX

默认值：256

max_pred_locks_per_transaction

参数说明：控制每个事务允许断定锁的最大数量，是一个平均值。

- 共享的断定锁表的大小是以假设任意时刻最多只有 max_pred_locks_per_transaction*(max_connections+max_prepared_transactions) 个独立的对象需要被锁住为基础进行计算的。不超过设定数量的多个对象可以在任一时刻同时被锁定。当在一个事务里面修改很多不同的表时，可能需要提高这个默认数值。只能在服务器启动的时候设置。
- 增大这个参数可能导致 PanWeiDB 请求更多的 System V 共享内存，有可能超过操作系统的缺省配置。

该参数属于 POSTMASTER 类型参数，请参考：配置运行参数->重设参数章节中表 2 对应设置方法进行设置。

取值范围：整型，10 ~ INT_MAX

默认值：64

gs_clean_timeout

参数说明：控制主节点周期性清理临时表的时间，是一个平均值。

- 数据库连接异常终止时，通常会有临时表残留，此时需要对数据库中的临时表进行清理。
- 增大这个参数可能导致 PanWeiDB 临时表清理时间延长。
- 当设置为 0 时，会关闭自动清理临时表的功能，不建议设置为 0。

该参数属于 SIGHUP 类型参数，请参考：配置运行参数->重设参数章节中表 2 对应设置方法进行设置。

取值范围：整型，0～2147483，单位为秒（s）。

默认值：1min

partition_lock_upgrade_timeout

参数说明：在执行某些查询语句的过程中，会需要将分区表上的锁级别由允许读的 ExclusiveLock 级别升级到读写阻塞的 AccessExclusiveLock 级别。如果此时已经存在并发的读事务，那么该锁升级操作将阻塞等待。
partition_lock_upgrade_timeout 为尝试锁升级的等待超时时间。

- 在分区表上进行 CLUSTER PARTITION 操作时，利用了临时表进行数据重排和文件交换，为了最大程度提高分区上的操作并发度，在数据重排阶段给相关分区加锁 ExclusiveLock，在文件交换阶段加锁 AccessExclusiveLock。
- 常规加锁方式是等待加锁，直到加锁成功，或者等待时间超过 lockwait_timeout 发生超时失败。
- 在分区表上进行 CLUSTER PARTITION 操作时，进入文件交换阶段需要申请加锁 AccessExclusiveLock，加锁方式是尝试性加锁，加锁成功了则立即返回，不成功则等待 50ms 后继续下次尝试，加锁超时时间使用会话级设置参数 partition_lock_upgrade_timeout。
- 特殊值：若 partition_lock_upgrade_timeout 取值-1，表示无限等待，即不停的尝试锁升级，直到加锁成功。

该参数属于 USERSET 类型参数，请参考：配置运行参数->重设参数章节中表 1 对应设置方法进行设置。

取值范围：整型，最小值-1，最大值 3000，单位为秒（s）。

默认值：1800

fault_mon_timeout

参数说明：轻量级死锁检测周期。

该参数属于 SIGHUP 类型参数，请参考：配置运行参数->重设参数章节中表 2 对应设置方法进行设置。

取值范围：整型，最小值 0，最大值 1440，单位为分钟（min）

默认值：5min

enable_online_ddl_waitlock

参数说明：控制 DDL 是否会阻塞等待 pg_advisory_lock/pgxc_lock_for_backup 等 PanWeiDB 锁。主要用于 OM 在线操作场景，不建议用户设置。

该参数属于 SIGHUP 类型参数，参考：配置运行参数->重设参数章节中表 2 对应设置方法进行设置。

取值范围：布尔型

- on 表示开启。
- off 表示关闭。

默认值：off

xloginsert_locks

参数说明：控制用于并发写预写式日志锁的个数。主要用于提高写预写式日志的效率。

该参数属于 POSTMASTER 类型参数，参考：配置运行参数->重设参数章节中表 2 对应设置方法进行设置。

该参数需要设置为 NUMA 节点的整数倍。

取值范围：整型，最小值 1，最大值 1000

默认值：8

num_internal_lock_partitions

参数说明：控制内部轻量级锁分区的个数。主要用于各类场景的性能调优。内容以关键字和数字的 KV 方式组织，各个不同类型锁之间以逗号隔开。先后顺序对设置结果不影响，例如“CLOG_PART=256,CSNLOG_PART=512”等同于“CSNLOG_PART=512,CLOG_PART=256”。重复设置同一关键字时，以最后一次设置为准，例如“CLOG_PART=256,CLOG_PART=2”，设置的结果为 CLOG_PART=2。当没有设置关键字时，则为默认值，各类锁的使用描述和最大、最小、默认值如下。

- CLOG_PART：CLOG 文件控制器的个数。增大该值可以提高 CLOG 日志写入效率，提升事务提交性能，但是会增大内存使用；减小该值会减少相应内存使用，但可能使得 CLOG 日志写入冲突变大，影响性能。最小值为 1，最大值为 256。
- CSNLOG_PART：CSNLOG 文件控制器的个数。增大该值可以提高 CSNLOG 日志写入效率，提升事务提交性能，但是会增大内存使用；减小该值会减少相应内存使用，但可能使得 CSNLOG 日志写入冲突变大，影响性能。最小值为 1，最大值为 512。
- LOG2_LOCKTABLE_PART：常规锁表锁分区个数的 2 对数。增大该值可以提升正常流程常规锁获取锁的并行度，但是可能增加锁转移和锁消除时的耗时，对于等待事件在 LockMgrLock 时，可以调大该锁增加性能。最小值为 4，即锁分区数为 16；最大值为 16，即锁分区数为 65536。
- TWOPCME_PART：两阶段事务锁的分区数。调大该值可以提高两阶段事务提交的并发数。最小值为 1，最大值为 64。
- FASTPATH_PART：每个线程可以不通过主锁表拿锁的最大锁个数，对于分区表读取、更新、插入、删除操作且等待事件在 LockMgrLock 时，可以通过调大该值避免获取 LockMgrLock 提升性能，建议调整数量大于等于分区数*（1+本地索引数量）+全局索引数量+10，调大该值会额外增加内存。最小值为 20，最大值为 10000。

该参数属于 POSTMASTER 类型参数，参考：配置运行参数->重设参数章节中表 2 对应设置方法进行设置。

取值范围：字符串

默认值：

- CLOG_PART：256
- CSNLOG_PART：512
- LOG2_LOCKTABLE_PART：4
- TWOPCME_PART：1

12.16.版本和平台兼容性

12.16.1.历史版本兼容性

PanWeiDB 介绍数据库的向下兼容性和对外兼容性特性的参数控制。数据库系统的向后兼容性能够为旧版本的数据库应用提供支持。本节介绍的参数主要控制数据库的向后兼容性。

array_nulls

参数说明：控制数组输入解析器是否将未用引号的 NULL 识别为数组的一个 NULL 元素。

该参数属于 USERSET 类型参数，请参考：配置运行参数->重设参数章节中表 2 对应设置方法进行设置。

取值范围：布尔型

- on 表示允许向数组中输入不带引号的 NULL，并将其识别为空值。
- off 表示向前兼容旧版本模式。将不带引号的 NULL 当作字符串“NULL”的正常数组元素。即仍然能够创建包含 NULL 值的数组。

默认值：on

backslash_quote

参数说明：控制字符串文本中的单引号是否能够用'表示。

该参数属于 USERSET 类型参数，请参考：配置运行参数->重设参数章节中表 2 对应设置方法进行设置。

【须知】
在字符串文本符合 SQL 标准的情况下，没有任何其他含义。这个参数影响的是如何处理不符合标准的字符串文本，包括明确的字符串转义语法是（E'...'）。

取值范围：枚举类型

- on 表示一直允许使用'表示。
- off 表示拒绝使用'表示。
- safe_encoding 表示仅在客户端字符集编码不会在多字节字符末尾包含的 ASCII 值时允许。

默认值：safe_encoding

escape_string_warning

参数说明：警告在普通字符串中直接使用反斜杠转义。

- 如果需要使用反斜杠作为转义，可以调整为使用转义字符串语法（E'...'）来做转义，因为在每个 SQL 标准中，普通字符串的默认行为现在将反斜杠作为一个普通字符。
- 这个变量可以帮助定位需要改变的代码。

该参数属于 USERSET 类型参数，请参考：配置运行参数->重设参数章节中表 2 对应设置方法进行设置。

取值范围：布尔型

默认值：on

lo_compat_privileges

参数说明：控制是否启动对大对象权限检查的向后兼容模式。

该参数属于 SUSET 类型参数，请参考：配置运行参数->重设参数章节中表 2 对应设置方法进行设置。

取值范围：布尔型

on 表示当读取或修改大对象时禁用权限检查，与 PostgreSQL 9.0 以前的版本兼容。

默认值：off

quote_all_identifiers

参数说明：当数据库生成 SQL 时，此选项强制引用所有的标识符（包括非关键字）。这将影响到 EXPLAIN 的输出及函数的结果，例如 pg_get_viewdef。详细说明请参见《管理员指南》->pw_dump 章节的--quote-all-identifiers 选项。

该参数属于 USERSET 类型参数，请参考：配置运行参数->重设参数章节中表 2 对应设置方法进行设置。

取值范围：布尔型

- on 表示打开强制引用。
- off 表示关闭强制引用。

默认值：off

sql_inheritance

参数说明：控制继承语义。用来控制继承表的访问策略，off 表示各种命令不能访问子表，即默认使用 ONLY 关键字。这是为了兼容旧版本而设置的。

该参数属于 USERSET 类型参数，请参考：配置运行参数->重设参数章节中表 2 对应设置方法进行设置。

取值范围：布尔型

- on 表示可以访问子表。

❖ off 表示不访问子表。

默认值：on

standard_conforming_strings

参数说明：控制普通字符串文本（'...'）中是否按照 SQL 标准把反斜扛当普通文本。

❖ 应用程序通过检查这个参数可以判断字符串文本的处理方式。
❖ 建议明确使用转义字符串语法（E'...'）来转义字符。

该参数属于 USERSET 类型参数，请参考：配置运行参数->重设参数章节中表 2 对应设置方法进行设置。

取值范围：布尔型

❖ on 表示打开控制功能。
❖ off 表示关闭控制功能。

默认值：on

synchronize_seqscans

参数说明：控制启动同步的顺序扫描。在大约相同的时间内并行扫描读取相同的数据块，共享 I/O 负载。

该参数属于 USERSET 类型参数，请参考：配置运行参数->重设参数章节中表 2 对应设置方法进行设置。

取值范围：布尔型

❖ on 表示扫描可能从表的中间开始，然后选择“环绕”方式来覆盖所有的行，为了与已经在进行中的扫描活动同步。这可能会造成没有用 ORDER BY 子句的查询得到行排序造成不可预测的后果。
❖ off 表示确保顺序扫描是从表头开始的。

默认值：on

enable_beta_features

参数说明：控制开启某些非正式发布的特性，仅用于 POC 验证。这些特性属于延伸特性，建议客户谨慎开启，在某些功能场景下可能存在问题。

该参数属于 USERSET 类型参数，请参考：配置运行参数->重设参数章节中表 2 对应设置方法进行设置。

取值范围：布尔型

- on 表示开启这些功能受限的特性，保持前向兼容。但某些场景可能存在功能上的问题。
- off 表示禁止使用这些特性。

默认值：off

default_with_oids

参数说明： 在没有声明 WITH OIDS 和 WITHOUT OIDS 的情况下，这个选项控制在新创建的表中 CREATE TABLE 和 CREATE TABLE AS 是否包含一个 OID 字段。它还决定 SELECT INTO 创建的表里面是否包含 OID 。该参数暂未使用。

不推荐在用户表中使用 OID，故默认设置为 off。需要带有 OID 字段的表应该在创建时声明 WITH OIDS 。

该参数属于 USERSET 类型参数，请参考表 1 中对应设置方法进行设置。

取值范围： 布尔型

- on 表示在新创建的表中 CREATE TABLE 和 CREATE TABLE AS 可以包含一个 OID 字段。
- off 表示在新创建的表中 CREATE TABLE 和 CREATE TABLE AS 不可以包含一个 OID 字段。

默认值： off

default_with_oids

参数说明： 在没有声明`WITH OIDS`和`WITHOUT OIDS`的情况下，这个选项控制在新创建的表中`CREATE TABLE`和`CREATE TABLE AS`是否包含一个 OID 字段。它还决定`SELECT INTO`创建的表里面是否包含 OID 。该参数暂未使用。

不推荐在用户表中使用 OID，故默认设置为 off。需要带有 OID 字段的表应该在创建时声明 WITH OIDS 。

该参数属于 USERSET 类型参数，请参考重设参数表 1 中对应设置方法进行设置。

取值范围： 布尔型

- on：表示在新创建的表中`CREATE TABLE`和`CREATE TABLE AS`可以包含一个 OID 字段。
- off：表示在新创建的表中`CREATE TABLE`和`CREATE TABLE AS`不可以包含一个 OID 字段。

默认值： off

enable_custom_parser

该参数在当前版本不支持。

12.16.2.平台和客户端兼容性

很多平台都使用数据库系统，数据库系统的对外兼容性给平台提供了很大的方便。

convert_string_to_digit

参数说明：设置隐式转换优先级，是否优先将字符串转为数字。

该参数属于 USERSET 类型参数，请参考：配置运行参数->重设参数章节中表 1 对应设置方法进行设置。

取值范围：布尔型

- on 表示优先将字符串转为数字。
- off 表示不优先将字符串转为数字。

默认值：on

【须知】
该参数调整会修改内部数据类型转换规则，导致不可预期的行为，请谨慎操作。

nls_timestamp_format

参数说明：设置时间戳默认格式。

该参数属于 USERSET 类型参数，请参考：配置运行参数->重设参数章节中表 1 对应设置方法进行设置。

取值范围：字符串

默认值：DD-Mon-YYYY HH:MI:SS.FF AM

max_function_args

参数说明：函数参数最大个数。

该参数属于 INTERNAL 类型参数，为固定参数，用户无法修改此参数，只能查看。

取值范围：整型

默认值：8192

transform_null_equals

参数说明：控制表达式 expr = NULL（或 NULL = expr）当做 expr IS NULL 处理。如果 expr 得出 NULL 值则返回真，否则返回假。

- 正确的 SQL 标准兼容的 expr = NULL 总是返回 NULL（未知）。
- Microsoft Access 里的过滤表单生成的查询使用 expr = NULL 来测试空值。打开这个选项，可以使用该接口来访问数据库。

该参数属于 USERSET 类型参数，请参考：配置运行参数->重设参数章节中表 1 对应设置方法进行设置。

取值范围：布尔型

- on 表示控制表达式 expr = NULL（或 NULL = expr）当做 expr IS NULL 处理。
- off 表示不控制，即 expr = NULL 总是返回 NULL（未知）。

默认值：off

【说明】
新用户经常在涉及 NULL 的表达式上语义混淆，故默认值设为 off。

support_extended_features

参数说明：控制是否支持数据库的扩展特性。

该参数属于 POSTMASTER 类型参数，请参考：配置运行参数->重设参数章节中表 1 对应设置方法进行设置。

取值范围：布尔型

- on 表示支持数据库的扩展特性。
- off 表示不支持数据库的扩展特性。

默认值：off

sql_compatibility

参数说明：控制数据库的 SQL 语法和语句行为同哪一个主流数据库兼容。

该参数属于 INTERNAL 类型参数，用户无法修改，只能查看。

取值范围：枚举型

- A 表示同 Oracle 数据库兼容。
- B 表示同 MySQL 数据库兼容。
- C 表示同 Teradata 数据库兼容。
- PG 表示同 PostgreSQL 数据库兼容。
- MSSQL 表示同 SQL Server 数据库兼容。

默认值：B

【须知】
该参数需要在执行数据库初始化时通过 dbcompatibility 设置。

behavior_compat_options

参数说明：数据库兼容性行为配置项，该参数的值由若干个配置项用逗号隔开构成。

该参数属于 USERSET 类型参数，请参考：配置运行参数->重设参数章节中表 1 对应设置方法进行设置。

取值范围：字符串

默认值：reduce_tailing_zero

【须知】
- 当前只支持“表 1 兼容性配置项”中的内容
- 配置多个兼容性配置项时，相邻配置项用逗号隔开，例如：set behavior_compat_options='end_month_calculate,display_leading_zero';

表 1 兼容性配置项

兼容性配置项	兼容性行为控制
display_leading_zero	浮点数显示配置项。浮点数显示配置项。控制数值类型中浮点类型、任意精度类型的小数点前零显示。并且 length 计算数字长度同步显示。 ❖ 不设置此配置项时，对于-1~0 和 0~1 之间的小数，不显示小数点前的 0。比如： panweidb=# select 0.1231243 as a, 0.1231243::numeric as b,0.1231243::integer(10,3) as c, length(0.1242343) as d; a | b | c | d ------+------+----+--- .1231243 | .1231243 | .123 | 8 (1 row) ❖ 设置此配置项时，对于-1~0 和 0~1 之间的小数，显示小数点前的 0。比如： panweidb=# select 0.1231243 as a, 0.1231243::numeric as b,0.1231243::integer(10,3) as c, length(0.1242343) as d; a | b | c | d ---------+---------+----+--- 0.1231243 |0 .1231243 |0 .123 | 9 (1 row)
end_month_calculate	add_months 函数计算逻辑配置项。 假定函数 add_months 的两个参数分别为 param1 和 param2，param1 的月份和 param2 的和为月份 result。 ❖ 不设置此配置项时，如果 param1 的日期（Day 字段）为月末，并且 param1 的日期（Day 字段）比 result 月份的月末日期小，计算结果中的日期字段（Day 字段）和 param1 的日期字段保持一致。比如： openGauss=# select add_months('2018-02-28',3) from sys_dummy; add_months \---------------------\ 2018-05-28 00:00:00

兼容性配置项	兼容性行为控制
	(1 row) ❖ 设置此配置项时，如果 param1 的日期（Day 字段）为月末，并且 param1 的日期（Day 字段）比 result 月份的月末日期比小，计算结果中的日期字段（Day 字段）和 result 的月末日期保持一致。比如： openGauss=# select add_months('2018-02-28',3) from sys_dummy; add_months \---------------------\ 2018-05-31 00:00:00 (1 row)
compat_analyze_sample	analyze 采样行为配置项。 设置此配置项时，会优化 analyze 的采样行为，主要体现在 analyze 时全局采样会更精确的控制在 3 万条左右，更好的控制 analyze 时 DBnode 端的内存消耗，保证 analyze 性能的稳定性。
bind_schema_tablespace	绑定模式与同名表空间配置项。 如果存在与模式名 sche_name 相同的表空间名，那么如果设置 search_path 为 sche_name， default_tablespace 也会同步切换到 sche_name。
bind_procedure_searchpath	未指定模式名的数据库对象的搜索路径配置项。 在存储过程中如果不显示指定模式名，会优先在存储过程所属的模式下搜索。 如果找不到，则有两种情况： ❖ 若不设置此参数，报错退出。 ❖ 若设置此参数，按照 search_path 中指定的顺序继续搜索。如果还是找不到，报错退出。
correct_to_number	控制 to_number()结果兼容性的配置项。 若设置此配置项，则 to_number()函数结果与 pg11 保持一致，否则默认与 O db 保持一致。

兼容性配置项	兼容性行为控制
unbind_divide_bound	控制对整数除法的结果进行范围校验。 若设置此配置项，则不需要对除法结果做范围校验，例如，INT_MIN/(-1)可以得到输出结果为 INT_MAX+1，反之，则会因为超过结果大于 INT_MAX 而报越界错误。
merge_update_multi	控制 merge into 匹配多行时是否进行 update 操作。 若设置此配置项，匹配多行时 update 不报错，否则默认与 a db 保持一致，报错。
return_null_string	控制函数 lpad()和 rpad()结果为空字符串”的显示配置项。 不设置此配置项时，空字符串显示为 NULL。 postgres=# select length(lpad('123',0,'*')) from sys_dummy; length ---- (1 row) 设置此配置项时，空字符串显示为”。 postgres=# select length(lpad('123',0,'*')) from sys_dummy; length ---- 0 (1 row)
compat_concat_variadic	控制函数 concat()和 concat_ws()对 variadic 类型结果兼容性的配置项。 若设置此配置项，当 concat 函数参数为 variadic 类型时，保留 a db 和 Teradata 兼容模式下不同的结果形式；否则默认 a db 和 Teradata 兼容模式下结果相同，且与 a db 保持一致。由于 MY 无 variadic 类型，所以该选项对 MY 无影响。
merge_update_multi	控制在使用 MERGE INTO … WHEN MATCHED THEN UPDATE（参考 MERGE INTO）和 INSERT … ON DUPLICATE KEY UPDATE（参考 INSERT）时，当目标表中一条目标数据与多条源数据冲突时 UPDATE 行为。 若设置此配置项，当存在上述场景时，该冲突行将会多次执行 UPDATE；否则（默认）报错，即 MERGE 或 INSERT 操作失败。

兼容性配置项	兼容性行为控制
hide_tailing_zero	numeric 显示配置项。不设置此项时，numeric 按照指定精度显示。设置此项时，隐藏小数点后的末尾 0。 set behavior_compat_options='hide_tailing_zero'; select cast(123.123 as numeric(15,10)); numeric ----- 123.123
rownum_type_compat	控制 ROWNUM 的类型，ROWNUM 默认类型为 INT8，设置此参数后，ROWNUM 类型变更为 NUMERIC 类型。
aformat_null_test	控制 rowtype 类型判空逻辑,设置此项时，对于 rowtype is not null 判断，当一行数据有一列不为空的时候返回 ture。 否则，对于 rowtype is not null 判断，当一行数据所有列不为空的时候返回 ture.
aformat_regexp_match	控制正则表达式函数的匹配行为。 设置此项，且 sql_compatibility 参数的值为 A 或 B 时，正则表达式的 flags 参数支持的选项含义有变更： ❖ 默认不能匹配 'n' 字符。 ❖ flags 中包含 n 选项时， 能够匹配 'n' 字符。 ❖ regexp_replace(source, pattern replacement) 函数替换所有匹配的子串。 ❖ regexp_replace(source, pattern, replacement, flags) 在 flags 值为” 或者 null 时，返回值为 null。 否则，正则表达式的 flags 参数支持的选项含义： ❖ 默认能匹配 'n' 字符。 ❖ flags 中的 n 选项表示按照多行模式匹配。 ❖ regexp_replace(source, pattern replacement) 函数仅替换第一

兼容性配置项	兼容性行为控制
	个匹配到的子串。 ❖ regexp_replace(source, pattern, replacement, flags) 在 flags 值为”或者 null 时，返回值为替换后的字符串。
compat_cursor	控制隐式游标状态兼容行为。设置此项，且兼容 O，隐式游标状态（SQL%FOUND、SQL%NOTFOUND、SQL%ISOPNE、SQL%ROWCOUNT）由原先的仅在当前执行的函数有效，拓展到包括本函数调用的子函数有效。
proc_outparam_override	控制存储过程出参的重载行为，打开该参数后，对于存储过程只有 out 出参部分不同的情况下，也可以正常调用。
proc_implicit_for_loop_variable	控制存储过程中 FOR_LOOP 查询语句行为设置此项时，在 FOR rec IN query LOOP 语句中，若 rec 已经定义，不会复用已经定义的 rec 变量，而且重新建立一个新的变量。否则，会复用已经定义的 rec 变量，不会建立新的变量。
allow_procedure_compile_check	控制存储过程中 select 语句和 open cursor 语句的编译检查设置此项时，在存储过程中执行 select 语句、open cursor for 语句、cursor%rowtype 语句、for rec in 语句时，若查询的表不存在，则无法创建创建存储过程，不支持 trigger 函数的编译检查，若查询的表存在，则成功创建存储过程。
char_coerce_compat	控制 char(n)类型向其他变长字符串类型转换时的行为，以及 char(n)和 varchar 操作时的行为。默认情况下 char(n)类型转换其他变长字符串类型时会省略尾部的空格，开启该参数后，转换时不再省略尾部的空格，并且在转换时如果 char(n)类型的长度超过其他变长字符串类型时将会报错；开启该参数后，char(n)和 varchar 操作时，例如比较操作符，首先各自转换为 text 类型再进行操作，具体表现为在操作时不会忽略尾部空格，如' '::char(5)和' '::varchar(5)不相等。该参数仅在 sql_compatibility 参数的值为 A 时生效。
pgformat_substr	控制 substr(str, from, for)在不同场景下的表现。默认情况下，当 from 小于 0 时，substr 将从字符串尾部开始计数；当 for 小于 1 时，substr 将返回 NULL。开启该参数后，当 from 小于 0 时，将从字符串的第一位的前(-from + 1)位开始计数；当 for 小于 0 时，substr 将报错。该参数仅在 sql_compatibility 参数的值为 PG 时生效。

兼容性配置项	兼容性行为控制
plpgsql_dependency	开启此参数后，创建函数，存储过程，包支持未定义的对象。可以新建成功。可以在 GS_DEPENDENCIES 和 GS_DEPENDENCIES_OBJ 查询对应的依赖关系。 开启此参数后，创建 PLSQL 对象时，会主动维护依赖于该 PLSQL 对象的 OID，不再需要用户手动更新。 注意: 在并发创建 PLSQL 对象时，如果需要维护的对象间存在竞争关系，可能会造成死锁。
block_return_multi_results	此选项表示开启使用存储过程或者匿名块支持返回一个或者多个结果集的功能。开启后，存储过程或者匿名块中的一个或者多个查询语句结果，会随着查询语句的执行输出到客户端，且每条查询结果是相互独立的。该参数仅在 sql_compatibility 参数的值为 B 和 MSSQL 时生效。

accept_empty_str	在 Oracle 兼容模式下，开启该参数，PanWeiDB 数据库会正常接收空字符串，关闭时则空字符串会当被做 NULL 处理。
	
	`set behavior_compat_options='accept_empty_str';`
	`select '' is null;`
	` ?column?`
	`----------`
	` f`
	`(1 row)`
	`set behavior_compat_options='';`
	`select '' is null;`
	` ?column?`
	`----------`
	` t`
	`(1 row)`

skip_insert_gs_source	开启此参数后，创建 PL/pgSQL 对象时，不再插入 DBE_PLDEVELOPER.gs_source 表中。 【须知】 该配置项默认开启，如果关闭，会导致 MySQL 兼容模式与 SQL SERVER 兼容模式（即初始化数据库时指定 DBCOMPATIBILITY = 'B' 或 'MSS

兼容性配置项	兼容性行为控制
	QL'）中创建存储过程报错。

plpgsql.variable_conflict

参数说明：设置同名的存储过程变量和表的列的使用优先级。

该参数属于 USERSET 类型参数，为固定参数，用户无法修改此参数，只能查看。

取值范围：字符串

- error 表示遇到存储过程变量和表的列名同名则编译报错。
- use_variable 表示存储过程变量和表的列名同名则优先使用变量。
- use_column 表示存储过程变量和表的列名同名则优先使用列名。

默认值：空

td_compatible_truncation

参数说明：控制是否开启与 Teradata 数据库相应兼容的特征。该参数在用户连接上与 TD 兼容的数据库时，可以将参数设置成为 on（即超长字符串自动截断功能启用），该功能启用后，在后续的 insert 语句中，对目标表中 char 和 varchar 类型的列插入超长字符串时，会按照目标表中相应列定义的最大长度对超长字符串进行自动截断。保证数据都能插入目标表中，而不是报错。

【说明】

超长字符串自动截断功能不适用于 insert 语句包含外表的场景。如果向字符集为字节类型编码（SQL_ASCII、LATIN1 等）的数据库中插入多字节字符数据（如汉字等），且字符数据跨越截断位置，这种情况下，按照字节长度自动截断，自动截断后会在尾部产生非预期结果。如果用户有对于截断结果正确性的要求，建议用户采用 UTF8 等能够按照字符截断的输入字符集作为数据库的编码集。

该参数属于 USERSET 类型参数，请参考：配置运行参数->重设参数章节中表 1 对应设置方法进行设置。

取值范围：布尔型

- on 表示启动超长字符串自动截断功能。
- off 表示停止超长字符串自动截断功能。

默认值：off

lastval_supported

参数说明：在分布式模式中，该参数控制是否可以使用 lastval 函数；而在单机模式下，该参数控制 nextval 函数在并行查询中是否可以下推到并行算子。

事实上，无论是分布式中 lastval 的使用，还是单机模式下 nextval 是否下推到并行算子，都不是由该参数单独控制的。enable_beta_features 在这两个行为的控制上也发挥着作用，因此在讨论该参数的作用时，前提都是 enable_beta_features 为 off。

该参数属于 POSTMASTER 类型参数，请参考：配置运行参数->重设参数章节中表 1 对应设置方法进行设置。

取值范围：布尔型

- on：在分布式模式中，表示允许 lastval 函数的使用；在单机模式中，表示 nextval 函数不可以下推到并行算子。
- off：在分布式模式中，表示不允许 lastval 函数的使用；在单机模式中，表示 nextval 函数可以下推到并行算子。

默认值：off

b_format_behavior_compat_options

参数说明：数据库 B 模式兼容性行为配置项，该参数的值由若干个配置项用逗号隔开构成。

该参数属于 USERSET 类型参数，请参考重设参数表 1 中对应设置方法进行设置。

取值范围： 字符串

默认值： ""

兼容性配置项	兼容性行为控制
enable_set_variables	set 语法增强控制开关。 ❖ 不设置此配置时，不支持 set 自定义变量、set [global | session]语法。 ❖ 设置此配置时，支持 B 兼容模式下使用上述语法，比如 set @v1 = 1;。
set_session_transaction	set_session_transaction 控制开关。 ❖ 不设置此配置时， set session transaction 等效于 set local transaction。 ❖ 设置此配置时，支持 B 兼容模式下使用上述语法，修改当前会话事务特性。
enable_modify_column	ALTER TABLE MODIFY 语义控制开关。 ❖ 不设置此配置时， ALTER TABLE table_name MODIFY column_name data_type;只修改列的数据类型。 ❖ 设置此配置时， ALTER TABLE table_name MODIFY column_name data_type;修改整个列定义。
default_collatio	默认字符序前向兼容开关。 ❖ 不设置此配置时，在未显式指定字符类型字段的字符集或字符序且表级字符序也为空时，字段为 default 字符序。 ❖ 设置此配置时，字符类型字段的字符

	序当表级字符序不为空时继承表级字符序，为空时设置为数据库编码对应的默认字符序。
fetch	FETCH 语句结尾报错兼容开关。 ❖ 不设置此配置时，FETCH 抓取完了所有可用行，它就停在最后一行后面，或者在反向抓取的情况下是停在第一行前面。 ❖ 设置此配置时，FETCH 抓取完了所有可用行后报错。
enable_multi_charset	多字符集功能的控制开关。 ❖ 不设置此配置时，不允许不同于数据库字符集的数据，不处理不同于数据库字符集的表达式。 ❖ 设置此配置时，允许数据库中含有不同于数据库字符集的数据，合并不同字符集的表达式。不同字符集的表达式运算规则详见：字符集与字符序。 【说明】 目前 PanWeiDB 暂不支持字符集的合并功能，不表示多字符集功能的控制开关，目前该参数开启只是为了可以设置 collation_connection。
b_format_escape	V2.0-S3.0.1_B01 及以上版本支持选项 b_format_escape，用来控制逃逸字符在 MySQL 兼容模式中的表现。 ❖ 开启该选项时，逃逸字符为反斜线时需要双写，与 MySQL 表现保持一致。 ❖ 关闭该选项时，表示使用 standard_conforming_strings 控制逃逸字符，与历史版本一致，参数设置为 on 时，逃逸字符为反斜线不用双写；参数设置为 off 时，在文串常量中写的

	任何反斜线都需要被双写。

collation_connection

参数说明：该参数用于指定常量字符串的排序规则。只有当参数 b_format_behavior_compat_options 包含选项 enable_multi_charset 的场景下才允许设置该参数。

该参数属于 USERSET 类型参数，请参考重设参数表 1 中对应设置方法进行设置。

【说明】

- 该参数仅在数据库兼容模式为 MySQL 时生效（即数据库实例初始化时指定 DBCOMPATIBILITY='B'）。
- 该参数属于会话级别参数，必须在配置文件设置或者 session 级别设置才会生效。

取值范围：字符串，合法的字符序的字符串，例如 utf8_general_ci。

默认值： 当前数据库字符集的默认字符序。

panwei_prioritize_dblink_call

参数说明： 语法中同时包含@操作符与 dblink 远程调用时，当参数为 on 时优先 dblink 远程调用，反之优先@操作符。

该参数属于 USERSET 类型参数，请参考重设参数表 1 中对应设置方法进行设置。

取值范围： 布尔型

- on：优先 dblink 远程调用。
- off：优先@操作符。

默认值： off

12.17.容错性

当数据库系统发生错误时，以下参数控制服务器处理错误的方式。

exit_on_error

参数说明：打开该开关，ERROR 级别报错会升级为 PANIC 报错，从而可以产生 core 堆栈。主要用于问题定位和业务测试。

该参数属于 USERSET 类型参数，请参考：配置运行参数->重设参数章节中表 1 对应设置方法进行设置。

取值范围：布尔型

- on 表示 ERROR 级别报错会升级为 PANIC 报错。
- off 表示不会对 ERROR 级别报错进行升级。

默认值：off

restart_after_crash

参数说明：设置为 on，后端进程崩溃时，PanWeiDB 将自动重新初始化此后端进程。

该参数属于 SIGHUP 类型参数，请参考：配置运行参数->重设参数章节中表 2 对应设置方法进行设置。

取值范围：布尔型

- on 表示能够最大限度地提高数据库的可用性。
  在某些情况（比如当采用管理工具（例如 xCAT）管理 PanWeiDB 时），能够最大限度地提高数据库的可用性。
- off 表示能够使得管理工具在后端进程崩溃时获取控制权并采取适当的措施进行处理。

默认值：on

omit_encoding_error

参数说明：设置为 on，数据库的客户端字符集编码为 UTF-8 时，出现的字符编码转换错误将打印在日志中，有转换错误的被转换字符会被忽略，以“?”代替。

该参数属于 USERSET 类型参数，请参考：配置运行参数->重设参数章节中表 2 对应设置方法进行设置。

取值范围：布尔型

- on 表示有转换错误的字符将被忽略，以“?”代替，打印错误信息到日志中。
- off 表示有转换错误的字符不能被转换，打印错误信息到终端。

默认值：off

max_query_retry_times

参数说明：指定 SQL 语句出错自动重试功能的最大重跑次数（目前支持重跑的错误类型为“Connection reset by peer”、“Lock wait timeout”和“Connection timed out”等），设定为 0 时关闭重跑功能。

该参数属于 USERSET 类型参数，请参考：配置运行参数->重设参数章节中表 2 对应设置方法进行设置。

取值范围：整型，0~20。

默认值：0

cn_send_buffer_size

参数说明：指定数据库主节点发送数据缓存区的大小。

该参数属于 POSTMASTER 类型参数，请参考：配置运行参数->重设参数章节中表 2 对应设置方法进行设置。

取值范围：整型，8~128，单位为 KB。

默认值：8KB

max_cn_temp_file_size

参数说明：指定 SQL 语句出错自动重试功能中数据库主节点端使用临时文件的最大值，设定为 0 表示不使用临时文件。

该参数属于 SIGHUP 类型参数，为固定参数，用户无法修改此参数，只能查看。

取值范围：整型，0~10485760， 单位为 KB。

默认值：空

retry_ecode_list

参数说明：指定 SQL 语句出错自动重试功能支持的错误类型列表。

该参数属于 USERSET 类型参数，请参考：配置运行参数->重设参数章节中表 2 对应设置方法进行设置。

取值范围：字符串。

默认值：YY001 YY002 YY003 YY004 YY005 YY006 YY007 YY008 YY009 YY010 YY011 YY012 YY013 YY014 YY015 53200 08006 08000 57P01 XX003 XX009 YY016

data_sync_retry

参数说明：控制当 fsync 到磁盘失败后是否继续运行数据库。由于在某些操作系统的场景下，fsync 失败后重试阶段即使再次 fsync 失败也不会报错，从而导致数据丢失。

该参数属于 POSTMASTER 类型参数，请参考：配置运行参数->重设参数章节中表 2 对应设置方法进行设置。

取值范围：布尔型

- on 表示当 fsync 同步到磁盘失败后采取重试机制，数据库继续运行。

❖ off 表示当 fsync 同步到磁盘失败后直接报 panic，停止数据库。

默认值：off

remote_read_mode

参数说明：远程读功能开关。读取主机上的页面失败时可以从备机上读取对应的页面。

该参数属于 POSTMASTER 类型参数，请参考：配置运行参数->重设参数章节中表 1 对应设置方法进行设置。

取值范围：枚举类型

❖ off 表示关闭远程读功能。
❖ non_authentication 表示开启远程读功能，但不进行证书认证。
❖ authentication 表示开启远程读功能，但要进行证书认证。

默认值：authentication

12.18.连接池参数

当使用连接池访问数据库时，在系统运行过程中，数据库连接是被当作对象存储在内存中的，当用户需要访问数据库时，并非建立一个新的连接，而是从连接池中取出一个已建立的空闲连接来使用。用户使用完毕后，数据库并非将连接关闭，而是将连接放回连接池中，以供下一个请求访问使用。

pooler_maximum_idle_time

参数说明：Pooler 链接自动清理功能使用，当链接池中链接空闲时间超过所设置值时，会触发自动清理机制，清理各节点的空闲链接数到 minimum_pool_size。

【说明】
此参数在该版本不生效。

该参数属于 USERSET 类型参数，请参考：配置运行参数->重设参数章节中表 2 中对应设置方法进行设置。

取值范围：整型，最小值为 0，最大值为 INT_MAX，最小单位为分钟

默认值：1h（即 60min）

minimum_pool_size

参数说明：Pooler 链接自动清理功能使用，自动清理后各 pooler 链接池对应节点的链接数最小剩余量，当参数设置为 0 时，可以关闭 pooler 链接自动清理功能。

【说明】
此参数在该版本不生效。

该参数属于 USERSET 类型参数，请参考：配置运行参数->重设参数章节中表 2 中对应设置方法进行设置。

取值范围：整型，最小值为 1，最大值为 65535

默认值：200

cache_connection

参数说明：是否回收连接池的连接。

该参数属于 SIGHUP 类型参数，请参考：配置运行参数->重设参数章节中表 2 对应设置方法进行设置。

取值范围：布尔型

- on 表示回收连接池的连接。
- off 表示不回收连接池的连接。

默认值：on

12.19.事务

介绍 PanWeiDB 事务隔离、事务只读、最大 prepared 事务数、维护模式目的参数设置及取值范围等内容。

transaction_isolation

参数说明：设置当前事务的隔离级别。

该参数属于 USERSET 类型参数，为固定参数，用户无法修改此参数，只能查看。

取值范围：字符串，只识别以下字符串，大小写空格敏感：

- serializable：PanWeiDB 中等价于 REPEATABLE READ。
- read committed：只能读取已提交的事务的数据（缺省），不能读取到未提交的数据。
- repeatable read：仅能读取事务开始之前提交的数据，不能读取未提交的数据以及在事务执行期间由其他并发事务提交的修改。
- default：设置为 default_transaction_isolation 所设隔离级别。

默认值：read committed

transaction_read_only

参数说明：设置当前事务是只读事务。

该参数在数据库恢复过程中或者在备机里，固定为 on；否则，固定为 default_transaction_read_only 的值。

该参数属于 USERSET 类型参数，为固定参数，用户无法修改此参数，只能查看。

取值范围：布尔型

- on 表示设置当前事务为只读事务。
- off 表示该事务可以是非只读事务。

默认值：off

xc_maintenance_mode

参数说明：设置系统进入维护模式。

该参数属于 SUSET 类型参数，仅支持：配置运行参数->重设参数章节中表 1 中的方式三进行设置。

取值范围：布尔型

- on 表示该功能启用。
- off 表示该功能被禁用。

【注意】
请谨慎打开这个开关，避免引起 PanWeiDB 数据不一致。

默认值：off

allow_concurrent_tuple_update

参数说明：设置是否允许并发更新。

该参数属于 USERSET 类型参数，请参考：配置运行参数->重设参数章节中表 1 对应设置方法进行设置。

取值范围：布尔型

- on 表示该功能启用。
- off 表示该功能被禁用。

默认值：on

transaction_deferrable

参数说明：指定是否允许一个只读串行事务延迟执行，使其不会执行失败。该参数设置为 on 时，当一个只读事务发现读取的元组正在被其他事务修改，则延迟该只读事务直到其他事务修改完成。该参数为预留参数，该版本不生效。与该参数类似的还有一个请参考：GUC 参数说明->客户端连接缺省设置->语句行为中 default_transaction_deferrable 参数，设置它来指定一个事务是否允许延迟。

该参数属于 USERSET 类型参数，为固定参数，用户无法修改此参数，只能查看。

取值范围：布尔型

- on 表示允许执行。
- off 表示不允许执行。

默认值：off

enable_show_any_tuples

参数说明：该参数只有在只读事务中可用，用于分析。

该参数属于 USERSET 类型参数，请参考：配置运行参数->重设参数章节中表 1 对应设置方法进行设置。

取值范围：布尔型

- on/true 表示表中元组的所有版本都会可见。
- off/false 表示表中元组的所有版本都不可见。

默认值：off

replication_type

参数说明：标记当前 HA 模式是单主机模式、主备从模式还是一主多备模式。

该参数属于 POSTMASTER 类型参数，请参考：配置运行参数->重设参数章节中表 1 对应设置方法进行设置。

该参数用户不能自己去设置参数值。

取值范围：仅限 1，不可修改。

- 2 表示单主机模式，此模式无法扩展备机，已不支持。
- 1 表示使用一主多备模式，全场景覆盖，推荐使用。
- 0 表示主备从模式，目前此模式暂不支持。

默认值：1

pgxc_node_name

参数说明：指定节点名称。

该参数属于 POSTMASTER 类型参数，请参考：配置运行参数->重设参数章节中表 1 对应设置方法进行设置。

在备机请求主机进行日志复制时，如果 application_name 参数没有设置，那么该参数会被用来作为备机在主机上的流复制槽名字。该流复制槽的命名方式为 "该参数值*备机 ip* 备机 port"。其中，备机 ip 和备机 port 取自 replconninfo 参数中指定的备机 ip 和端口号。该流复制槽最大长度为 61 个字符，如果拼接后的字符串超过该长度，则会使用截断后的 pgxc_node_name 进行拼接，以保证流复制槽名字长度小于等于 61 个字符。

【注意】
此参数修改后会导致连接数据库实例失败，不建议进行修改。

取值范围：字符串

默认值：当前节点名称

enable_defer_calculate_snapshot

参数说明：延迟计算快照的 xmin 和 oldestxmin，执行 1000 个事务或者间隔 1s 才触发计算，设置为 on 时可以在高负载场景下减少计算快照的开销，

但是会导致 oldestxmin 推进较慢，影响垃圾元组回收，设置为 off 时 xmin 和 oldestxmin 可以实时推进，但是会增加计算快照时的开销。

该参数属于 SIGHUP 类型参数，改请参考：配置运行参数->重设参数章节中表 2 进行设置

取值范围：布尔型。

- on 表示延迟计算快照 xmin 和 oldestxmin。
- off 表示实时计算快照 xmin 和 oldestxmin。

默认值：on

enable_on_error_rollback

功能描述： 控制是否在出现错误时自动启用事务回滚功能，开启此参数后，在事务执行期间，当有 SQL 执行失败之后，后续的 SQL 可以正常执行，并且 commit 正常提交。

该参数属于 SIGHUP 类型参数，请参考重设参数表 1 中对应设置方法进行设置。

取值范围： 布尔型

- on：表示开启自动事务回滚开关。
- off：表示不开启自动事务回滚开关。

默认值： off

enable_on_error_rollback_debug

功能描述： 控制是否开启打印 enable_on_error_rollback 特性的相关调试信息。

该参数属于 USERSET 类型参数，请参考重设参数表 1 中对应设置方法进行设置。

取值范围： 布尔型

- on：表示开启。
- off：表示不开启。

默认值： off

12.20.双数据库实例复制参数

RepOriginId

参数说明：该参数是一个会话级别的 GUC 参数，在双向逻辑复制的场景下，为避免数据循环复制，需要设置为一个非 0 的值。

该参数属于 USERSET 类型参数，仅支持：配置运行参数->重设参数章节中表 1 中的方式三进行设置。

取值范围：整型，0~2147483647

默认值：0

stream_cluster_run_mode

参数说明：流式容灾双实例容灾场景，标识 DN 节点属于主实例还是备实例。单实例使用默认值主实例。

该参数属于 POSTMASTER 类型参数，请参考重设参数表 1 中对应设置方法进行设置。

取值范围： 枚举类型

- cluster_primary：表示节点是主实例的节点。
- cluster_standby：表示节点是备实例的节点。

默认值： cluster_primary

12.21.开发人员选项

allow_create_sysobject

参数说明： 设置是否允许在系统模式下创建或修改函数、存储过程、同义词等对象。此处的系统模式指数据库初始后自带的模式，但不包含 public 模式。该参数属于 POSTMASTER 类型参数，请参考重设参数表 1 中对应设置方法进行设置。

取值范围： 布尔型

- on：表示允许初始用户和系统管理员在系统模式下创建或修改函数、存储过程、同义词等对象。其他用户是否允许创建这些对象请参考对应模式的权限要求。
- off：表示禁止所有用户在系统模式下创建或修改函数、存储过程、同义词等对象。

默认值： on

block_encryption_mode

参数说明： 控制基于块算法(如 AES)的块加密模式。

该参数属于 USERSET 类型参数，请参考重设参数表 1 中对应设置方法进行设置。

取值范围： 枚举类型，有效值如下：

- aes-128-cbc
- aes-192-cbc
- aes-256-cbc
- aes-128-cfb1
- aes-192-cfb1
- aes-256-cfb1
- aes-128-cfb8
- aes-192-cfb8

- aes-256-cfb8
- aes-128-cfb128
- aes-192-cfb128
- aes-256-cfb128
- aes-128-ofb
- aes-192-ofb
- aes-256-ofb

其中 aes 表示加/解密算法，128/192/256 表示密钥长度（单位：bit），cbc/cfb1/cfb8/cfb128/ofb 表示块加/解密模式。

默认值： aes-128-cbc

allow_system_table_mods

参数说明：设置是否允许普通用户修改系统表，无法限制超级用户对系统表的修改。因修改系统表可能导致数据库运行异常或无法启动等故障问题，请谨慎操作。

该参数属于 POSTMASTER 类型参数，请参考：配置运行参数->重设参数章节中表 1 对应设置方法进行设置。

取值范围：布尔型

- on 表示允许修改系统表的结构。
- off 表示不允许修改系统表的结构。

默认值：off

【注意】
不建议修改该参数默认值，若设置为 on，可能导致系统表损坏，甚至数据库无法启动。

debug_assertions

参数说明：控制打开各种断言检查。能够协助调试，当遇到奇怪的问题或者崩溃，请把此参数打开，因为它能暴露编程的错误。要使用这个参数，必须在编

译 PanWeiDB 的时候定义宏 USE_ASSERT_CHECKING（通过 configure 选项 --enable-cassert 完成）。

该参数属于 USERSET 类型参数，为固定参数，用户无法修改此参数，只能查看。

取值范围：布尔型

- on 表示打开断言检查。
- off 表示不打开断言检查。

【说明】

当启用断言选项编译 PanWeiDB 时，debug_assertions 缺省值为 on 。

默认值：off

ignore_checksum_failure

参数说明：设置读取数据时是否忽略校验信息检查失败（但仍然会告警），继续执行可能导致崩溃，传播或隐藏损坏数据，无法从远程节点恢复数据及其他严重问题。不建议用户修改设置。

该参数属于 SUSET 类型参数，请参考：配置运行参数->重设参数章节中表 1 对应设置方法进行设置。

取值范围：布尔型

- on 表示忽略数据校验错误。
- off 表示数据校验错误正常报错。

默认值：off

ignore_system_indexes

参数说明：读取系统表时忽略系统索引（但是修改系统表时依然同时修改索引）。

该参数属于 BACKEND 类型参数，请参考：配置运行参数->重设参数章节中表 1 对应设置方法进行设置。

【须知】
这个参数在从系统索引被破坏的表中恢复数据的时候非常有用。

取值范围：布尔型

- on 表示忽略系统索引。
- off 表示不忽略系统索引。

默认值：off

post_auth_delay

参数说明：在认证成功后，延迟指定时间，启动服务器连接。允许调试器附加到启动进程上。

该参数属于 BACKEND 类型参数，请参考：配置运行参数->重设参数章节中表 1 对应设置方法进行设置。

取值范围：整型，最小值为 0，最大值为 2147，单位为秒。

默认值：0

【说明】
此参数只用于调试和问题定位，为避免影响正常业务运行，生产环境下请确保参数值为默认值 0。参数设置为非 0 时可能会因认证延迟时间过长导致数据库实例状态异常。

pre_auth_delay

参数说明：启动服务器连接后，延迟指定时间，进行认证。允许调试器附加到认证过程上。

该参数属于 SIGHUP 类型参数，请参考：配置运行参数->重设参数章节中表 1 对应设置方法进行设置。

取值范围：整型，最小值为 0～60，单位为秒。

默认值：0

【说明】

此参数只用于调试和问题定位，为避免影响正常业务运行，生产环境下请确保参数值为默认值 0。参数设置为非 0 时可能会因认证延迟时间过长导致数据库实例状态异常。

trace_notify

参数说明：为 LISTEN 和 NOTIFY 命令生成大量调试输出。GUC 参数说明->错误报告和日志->记录日志的时间中 client_min_messages、log_min_messages 参数级别必须是 DEBUG1 或者更低时，才能把这些输出分别发送到客户端或者服务器日志。

该参数属于 USERSET 类型参数，请参考：配置运行参数->重设参数章节中表 1 对应设置方法进行设置。

取值范围：布尔型

- on 表示打开输出功能。
- off 表示关闭输出功能。

默认值：off

trace_recovery_messages

参数说明：启用恢复相关调试输出的日志录，否则将不会被记录。该参数允许覆盖正常设置 GUC 参数说明->错误报告和日志->记录日志的时间中的 log_min_messages 参数，但是仅限于特定的消息，这是为了在调试备机中使用。

该参数属于 SIGHUP 类型参数，请参考：配置运行参数->重设参数章节中表 1 对应设置方法进行设置。

取值范围：枚举类型，有效值有 debug5、debug4、debug3、debug2、debug1、log，取值的详细信息请参见：GUC 参数说明->错误报告和日志->记录日志的时间中 log_min_messages 参数。

默认值：log

【说明】

- 默认值 log 表示不影响记录决策。
- 除默认值外，其他值会导致优先级更高的恢复相关调试信息被记录，因为它们有 log 优先权。对于常见的 log_min_messages 设置，这会导致无条件地将它们记录到服务器日志上。

trace_sort

参数说明：控制是否在日志中打印排序操作中的资源使用相关信息。这个选项只有在编译 PanWeiDB 的时候定义了 TRACE_SORT 宏的时候才可用，不过目前 TRACE_SORT 是由缺省定义的。

该参数属于 USERSET 类型参数，请参考：配置运行参数->重设参数章节中表 1 对应设置方法进行设置。

取值范围：布尔型

- on 表示打开控制功能。
- off 表示关闭控制功能。

默认值：off

repl_uuid

参数说明： 设置用于主备 UUID 验证的 UUID 码。

该参数属于 SIGHUP 类型参数，请参考重设参数表 1 中对应设置方法进行设置。

【说明】

- 如果主机上开启了 UUID 验证功能、且配置了非空字符串的 repl_uuid 验证码，那么备机也需要开启 UUID 验证功能、且配置相同的 repl_uuid 验证码，否则主备日志复制和备机重建请求将被主机拒绝。
- 该参数支持 SIGHUP 动态加载新值。修改之后，不影响已建连的主备连接，对后续主备复制请求和主备重建请求生效。
- 支持 Quorum、DCF 协议下的备机重建验证；支持 Quorum 协议下的主备复制验证；不支持 DCF 协议下的主备复制验证。
- UUID 验证功能主要为了防止主、备误连导致的数据串扰和污染，不是用于安全目的。
- 该参数不支持主、备间自动同步。

取值范围： 字符串类型。长度 0－63 个字符，字母和数字的组合，大小写不敏感，内部统一转换为小写存储。空字符串表示不启用 UUID 验证功能。

默认值： 空字符串

zero_damaged_pages

参数说明：控制检测导致 PanWeiDB 报告错误的损坏的页头，终止当前事务。

该参数属于 SUSET 类型参数，请参考：配置运行参数->重设参数章节中表 1 对应设置方法进行设置。

取值范围：布尔型

设置为 on 时，会导致系统报告一个警告，把损坏的页面填充为零然后继续处理。这种行为会破坏数据，也就是所有在已经损坏页面上的行记录。但是它允许绕开坏页面然后从表中尚存的未损坏页面上继续检索数据行。因此它在因为硬件或者软件错误导致的崩溃中进行恢复是很有用的。通常不应该把它设置为 on，除非不需要从崩溃的页面中恢复数据。

默认值：off

remotetype

参数说明：设置远程连接类型。

该参数不支持修改。

取值范围：枚举类型，有效值有 application、datanode、internaltool。

默认值：application

max_user_defined_exception

参数说明：异常最大个数，默认值不可更改。

该参数属于 USERSET 类型参数，请参考：配置运行参数->重设参数章节中表 1 对应设置方法进行设置。

取值范围：整型，当前只能取固定值 1000

默认值：1000

enable_fast_numeric

参数说明：标识是否开启 Numeric 类型数据运算优化。Numeric 数据运算是较为耗时的操作之一，通过将 Numeric 转化为 int64/int128 类型，提高 Numeric 运算的性能。

该参数属于 SUSET 类型参数，请参考：配置运行参数->重设参数章节中表 1 对应设置方法进行设置。

取值范围：布尔型

- on/true 表示开启 Numeric 优化。
- off/false 表示关闭 Numeric 优化。

默认值：on

enable_compress_spill

参数说明：标识是否开启下盘压缩功能。

该参数属于 USERSET 类型参数，请参考：配置运行参数->重设参数章节中表 1 对应设置方法进行设置。

取值范围：布尔型

- on/true 表示开启下盘优化。
- off/false 表示关闭下盘优化。

默认值：on

resource_track_log

参数说明：控制自诊断的日志级别。目前仅对多列统计信息进行控制。

该参数属于 USERSET 类型参数，请参考：配置运行参数->重设参数章节中表 1 对应设置方法进行设置。

取值范围：字符串

- summary：显示简略的诊断信息。
- detail：显示详细的诊断信息。

目前这两个参数值只在显示多列统计信息未收集的告警的情况下有差别，summary 不显示未收集多列统计信息的告警，detail 会显示这类告警。

默认值：summary

show_acce_estimate_detail

参数说明：评估信息一般用于运维人员在维护工作中使用，因此该参数默认关闭，此外为了避免这些信息干扰正常的 explain 信息显示，只有在 explain 命令的 verbose 选项打开的情况下才显示评估信息

该参数属于 USERSET 类型参数，请参考：配置运行参数->重设参数章节中表 1 对应设置方法进行设置。

取值范围：布尔型

- on 表示可以在 explain 命令的输出中显示评估信息。
- off 表示不在 explain 命令的输出中显示评估信息。

默认值：off

【说明】
当前版本不支持加速数据库实例，因此该参数设置后不生效。

support_batch_bind

参数说明：控制是否允许通过 JDBC、ODBC、Libpq 等接口批量绑定和执行 PBE 形式的语句。

该参数属于 SIGHUP 类型参数，请参考：配置运行参数->重设参数章节中表 1 对应设置方法进行设置。

取值范围：布尔型

- on 表示使用批量绑定和执行。
- off 表示不使用批量绑定和执行。

默认值：on

numa_distribute_mode

参数说明：用于控制部分共享数据和线程在 NUMA 节点间分布的属性。用于大型多 NUMA 节点的 ARM 服务器性能调优，一般不用设置。

该参数属于 POSTMASTER 类型参数，请参考：配置运行参数->重设参数章节中表 1 对应设置方法进行设置。

取值范围：字符串，当前有效取值为'none'，'all'。

- 'none'：表示不启用本特性。
- 'all'：表示将部分共享数据和线程分布到不同的 NUMA 节点下，减少远端访存次数，提高性能。目前仅适用于拥有多个 NUMA 节点的 ARM 服务

器，并且要求全部 NUMA 节点都可用于数据库进程，不支持仅选择一部分 NUMA 节点。

【说明】
当前版本 x86 平台下不支持 numa_distribute_mode 设置为 all。

默认值：'none'

log_pagewriter

参数说明：设置用于增量检查点打开后，显示线程的刷页信息以及增量检查点的详细信息，信息比较多，不建议设置为 true。

该参数属于 SIGHUP 类型参数，请参考：配置运行参数->重设参数章节中表 1 对应设置方法进行设置。

取值范围：布尔型

默认值：off

advance_xlog_file_num

参数说明：用于控制备机在后台周期性地提前初始化 xlog 文件的数目。该参数是为了避免事务提交时执行 xlog 文件初始化影响性能，但仅在备机超重负载时才可能出现，因此一般不用配置。

该参数属于 POSTMASTER 类型参数，请参考：配置运行参数->重设参数章节中表 1 对应设置方法进行设置。

取值范围：整型，0~1000000（0 表示不提前初始化）。例如，取值 10，表示后台线程会周期性地根据当前 xlog 写入位置提前初始化 10 个 xlog 文件。

默认值：0

enable_beta_opfusion

参数说明：在 enable_opfusion 参数打开的状态下，如果开启该参数，可以支持 TPCC 中出现的聚集函数，排序两类 SQL 语句的加速执行，提升 SQL 执行性能。

该参数属于 USERSET 类型参数，请参考：配置运行参数->重设参数章节中表 1 对应设置方法进行设置。

取值范围：布尔型

- on 表示开启。
- off 表示不开启。

默认值：off

var_eq_const_selectivity

参数说明： 整型常量选择率是否使用新型选择率模型进行估算。

该参数属于 USERSET 类型参数，请参考重设参数表 1 中对应设置方法进行设置。

取值范围： 布尔型

- on：表示使用新型选择率模型计算整型常量的选择率：
  - 若整型不落入 MCV 且不为 NULL 值，但落入直方图，则利用直方图左右边界情况进行估算。
  - 如果也不落入直方图，则使用表的行数进行估算。
  - 若整型为 NULL 值或者 MCV 值，使用原逻辑计算选择率。
- off：表示使用原有的选择率计算模型。

默认值： off

string_hash_compatible

参数说明：该参数用来说明 char 类型和 varchar/text 类型的 hash 值计算方式是否相同，以此来判断进行分布列从 char 类型到相同值的 varchar/text 类型转换，数据分布变化时，是否需要进行重分布。

该参数属于 POSTMASTER 类型参数，请参考：配置运行参数->重设参数章节中表 1 对应设置方法进行设置。

取值范围：布尔型

- on 表示计算方式相同，不需要进行重分布。
- off 表示计算方式不同，需要进行重分布。

【说明】

计算方式的不同主要体现在字符串计算 hash 值时传入的字节长度上。（如果为 char，则会忽略字符串后面空格的长度，如果为 text 或 varchar，则会保留字符串后面空格的长度。）hash 值的计算会影响到查询的计算结果，因此此参数一旦设置后，在整个数据库使用过程中不能再对其进行修改，以避免查询错误。

默认值：off

pldebugger_timeout

参数说明：该参数用来控制 pldebugger server 端等待 debug 端响应的超时时间。

该参数属于 USERSET 类型参数，请参考：配置运行参数->重设参数章节中表 1 对应设置方法进行设置。

取值范围：整型，1 ~ 86400，单位为秒。

默认值：15min

plsql_show_all_error

参数说明：该参数用来控制编译 PLPGSQL 对象时是否支持跳过报错继续编译，具体影响请参考：schema->Information-Schema->DBE_PLDEVELOPER 内的说明。

该参数属于 USERSET 类型参数，请参考：配置运行参数->重设参数章节中表 1 对应设置方法进行设置。

取值范围：布尔型

默认值：off

【注意】
将该参数设置为 ON 会导致 MySQL 兼容模式与 SQL SERVER 兼容模式（即初始化数据库时指定 DBCOMPATIBILITY = 'B' 或 'MSSQL'）中创建存储过程报错。

varstr_optimize_prop_card

参数说明： 该参数用于控制变长字符串比较优化。

- 如果缩略键基数/原始键基数小于此值，会退出优化。
- 此值设置为 0，可以强制使用比较优化，但性能不一定提高。

该参数属于 USERSET 类型参数，请参考重设参数表 1 中对应设置方法进行设置。

取值范围：浮点型，0~1

默认值： 0.2

enable_runtime_prune

参数说明： 该参数用于控制是否启用动态分区剪枝功能。

该参数属于 USERSET 类型参数，请参考重设参数表 1 中对应设置方法进行设置。

取值范围： 布尔型

- on：启用动态分区剪枝功能。
- off：不启用动态分区剪枝功能。

默认值： on

12.22.审计

12.22.1.审计开关

audit_enabled

参数说明：控制审计进程的开启和关闭。审计进程开启后，将从管道读取后台进程写入的审计信息，并写入审计文件。

该参数属于 SIGHUP 类型参数，请参考：配置运行参数->重设参数章节中表 1 对应设置方法进行设置。

取值范围：布尔型

- on 表示启动审计功能。
- off 表示关闭审计功能。

默认值：off

audit_directory

参数说明：审计文件的存储目录。一个相对于数据目录 data 的路径，可自行指定。

该参数属于 POSTMASTER 类型参数，请参考：配置运行参数->重设参数章节中表 1 对应设置方法进行设置。

取值范围：字符串

默认值：pg_audit。如果使用 om 工具部署 PanWeiDB，则审计日志路径为“$GAUSSLOG/pg_audit/实例名称”。

【注意】

- ❖ 不同的 DN 实例需要设置不同的审计文件存储目录，否则会导致审计日志异常。
- ❖ 当配置文件中 audit_directory 的值为非法路径时，会导致审计功能无法使用。路径说明：
- ➢ 合法路径：用户对此路径有读写权限。
- ➢ 非法路径：用户对此路径无读写权限，使用非法路径后，会导致审计日志线程会启动失败，然后又会被自动拉起，每次拉起都会分配内存但没释放就会导致内存不断增高。

audit_dump_directory

参数说明： 设置审计日志自动转储目录，详细内容请参考审计日志自动转储。

该参数属于 POSTMASTER 类型参数，请参考重设参数表 1 中对应设置方法进行设置。

取值范围： 字符串（绝对路径）

默认值： 空字符串

audit_backup_directory

功能描述： 审计文件的备份存储路径。可以是一个相对于数据目录（即$PGDATA）的路径或绝对路径。

该参数属于 POSTMASTER 类型参数，请参考重设参数表 1 中对应设置方法进行设置。

取值范围： 字符串（相对于数据目录（即$PGDATA）的路径或绝对路径）

默认值： 空

audit_data_format

参数说明：审计日志文件的格式。当前仅支持二进制格式。

该参数属于 POSTMASTER 类型参数，请参考：配置运行参数->重设参数章节中表 1 对应设置方法进行设置。

取值范围：字符串

默认值：binary

audit_rotation_interval

参数说明：指定创建一个新审计日志文件的时间间隔。当现在的时间减去上次创建一个审计日志的时间超过了此参数值时，服务器将生成一个新的审计日志文件。

该参数属于 SIGHUP 类型参数，请参考：配置运行参数->重设参数章节中表 1 对应设置方法进行设置。

取值范围：整型，1~INT_MAX/60，单位为 min。

默认值：1d

【须知】
请不要随意调整此参数，否则可能会导致 audit_resource_policy 无法生效，如果需要控制审计日志的存储空间和时间，请使用 audit_resource_policy、audit_space_limit 和 audit_file_remain_time 参数进行控制。

audit_rotation_size

参数说明：指定审计日志文件的最大容量。当审计日志消息的总量超过此参数值时，服务器将生成一个新的审计日志文件。

该参数属于 SIGHUP 类型参数，请参考：配置运行参数->重设参数章节中表 1 对应设置方法进行设置。

取值范围：整型，1024~1048576，单位为 KB。

默认值：10MB

【须知】
请不要随意调整此参数，否则可能会导致 audit_resource_policy 无法生效，如果需要控制审计日志的存储空间和时间，请使用 audit_resource_policy、audit_space_limit 和 audit_file_remain_time 参数进行控制。

audit_resource_policy

参数说明：控制审计日志的保存策略，以空间还是时间限制为优先策略。

该参数属于 SIGHUP 类型参数，请参考：配置运行参数->重设参数章节中表 1 对应设置方法进行设置。

取值范围：布尔型

- on 表示采用空间优先策略，最多存储 audit_space_limit 大小的日志。
- off 表示采用时间优先策略，最少存储 audit_file_remain_time 长度时间的日志。

默认值：on

audit_file_remain_time

参数说明：表示需记录审计日志的最短时间要求，该参数在 audit_resource_policy 为 off 时生效。

该参数属于 SIGHUP 类型参数，请参考：配置运行参数->重设参数章节中表 1 对应设置方法进行设置。

取值范围：整型，0~730，单位为 day，0 表示无时间限制。

默认值：90

audit_space_limit

参数说明：审计文件占用的磁盘空间总量。

该参数属于 SIGHUP 类型参数，请参考：配置运行参数->重设参数章节中表 1 对应设置方法进行设置。

取值范围：整型，1024KB~1024GB，单位为 KB。

默认值：1GB

audit_file_remain_threshold

参数说明：审计目录下审计文件个数的最大值。

该参数属于 SIGHUP 类型参数，请参考：配置运行参数->重设参数章节中表 1 对应设置方法进行设置。

取值范围：整型，100~1048576

默认值：1048576

【须知】
请尽量保证此参数为 1048576，并不要随意调整此参数，否则可能会导致 audit_resource_policy 无法生效，如果需要控制审计日志的存储空间和时间，请使用 audit_resource_policy、audit_space_limit 和 audit_file_remain_time 参数进行控制。

audit_thread_num

参数说明：审计线程的个数。

该参数属于 POSTMASTER 类型参数，请参考：配置运行参数->重设参数章节中表 1 对应设置方法进行设置。

取值范围：整型，1~48

默认值：1

【须知】
当 audit_dml_state 开关打开且在高性能场景下，建议增大此参数保证审计消息可以被及时处理和记录。

audit_buffer_size

参数说明： 该参数表示审计日志写缓存大小。

该参数属于 POSTMASTER 类型参数，请参考重设参数表 1 中对应设置方法进行设置。

取值范围： 整型，0-51200，单位 kb，取值为 0 表示不开启审计日志写缓存。

默认值： 0

audit_buffer_fflush_interval

参数说明： 该参数表示自动检测需要刷审计日志写缓存的时间间隔。

【须知】
该参数只有在 audit_buffer_size 开启时才有效果。

该参数属于 SIGHUP 类型参数，请参考重设参数表 1 中对应设置方法进行设置。

取值范围： 整型，1-1800，单位 s

默认值： 120

12.22.2.用户和权限审计

audit_login_logout

参数说明：这个参数决定是否审计 PanWeiDB 用户的登录（包括登录成功和登录失败）、注销。

该参数属于 SIGHUP 类型参数，请参考：配置运行参数->重设参数章节中表 1 对应设置方法进行设置。

取值范围：整型，0~7。

- 0 表示关闭用户登录、注销审计功能。
- 1 表示只审计用户登录成功。
- 2 表示只审计用户登录失败。
- 3 表示只审计用户登录成功和失败。
- 4 表示只审计用户注销。
- 5 表示只审计用户注销和登录成功。
- 6 表示只审计用户注销和登录失败。
- 7 表示审计用户登录成功、失败和注销。

默认值：7

audit_database_process

参数说明：该参数决定是否对 PanWeiDB 的启动、停止、切换和恢复进行审计。

该参数属于 SIGHUP 类型参数，请参考：配置运行参数->重设参数章节中表 1 对应设置方法进行设置。

取值范围：整型，0、1。

- 0 表示关闭 PanWeiDB 启动、停止、恢复和切换审计功能。
- 1 表示开启 PanWeiDB 启动、停止、恢复和切换审计功能。

默认值：1

【说明】

PanWeiDB 启动时 DN 执行备升主流程，因此 DN 启动时审计日志中类型为 system_switch。

audit_user_locked

参数说明：该参数决定是否审计 PanWeiDB 用户的锁定和解锁。

该参数属于 SIGHUP 类型参数，请参考：配置运行参数->重设参数章节中表 1 对应设置方法进行设置。

取值范围：整型，0、1。

- 0 表示关闭用户锁定和解锁审计功能。
- 1 表示开启审计用户锁定和解锁功能。

默认值：1

audit_user_violation

参数说明：该参数决定是否审计用户的越权访问操作。

该参数属于 SIGHUP 类型参数，请参考：配置运行参数->重设参数章节中表 1 对应设置方法进行设置。

取值范围：整型，0、1。

- 0 表示关闭用户越权操作审计功能。
- 1 表示开启用户越权操作审计功能。

默认值：0

audit_grant_revoke

参数说明：该参数决定是否审计 PanWeiDB 用户权限授予和回收的操作。

该参数属于 SIGHUP 类型参数，请参考：配置运行参数->重设参数章节中表 1 对应设置方法进行设置。

取值范围：整型，0、1。

- 0 表示关闭审计用户权限授予和回收功能。
- 1 表示开启审计用户权限授予和回收功能。

默认值：1

12.22.3.操作审计

audit_system_object

参数说明：该参数决定是否对 PanWeiDB 数据库对象的 CREATE、DROP、ALTER 操作进行审计。PanWeiDB 数据库对象包括 DATABASE、USER、schema、TABLE 等。通过修改该配置参数的值，可以只审计需要的数据库对象的操作。

该参数属于 SIGHUP 类型参数，请参考：配置运行参数->重设参数章节中表 1 对应设置方法进行设置。

取值范围：整型，0～134217727

- 0 代表关闭 PanWeiDB 数据库对象的 CREATE、DROP、ALTER 操作审计功能。
- 非 0 代表只审计 PanWeiDB 的某类或者某些数据库对象的 CREATE、DROP、ALTER 操作。

取值说明：该参数的值由 29 个二进制位（第 0 位~第 28 位）的组合求出，这 29 个二进制位分别代表 PanWeiDB 的 27 类数据库对象。

- 如果对应的二进制位取值为 0，表示不审计对应的数据库对象的 CREATE、DROP、ALTER 操作；
- 取值为 1，表示审计对应的数据库对象的 CREATE、DROP、ALTER 操作。各个二进制位代表的具体审计内容请参见表 1。

默认值：67121159

表 1 audit_system_object 取值含义说明

二进制位	含义	取值说明
第 0 位	是否审计 DATABASE 对象的 CREATE、DROP、ALTER 操作。	❖ 0 表示不审计该对象的 CREATE、DROP、ALTER 操作； ❖ 1 表示审计该对象的 CREATE、DROP、ALTER 操作。
第 1 位	是否审计 SCHEMA 对象的 CREATE、DROP、ALTER 操作。	❖ 0 表示不审计该对象的 CREATE、DROP、ALTER 操作； ❖ 1 表示审计该对象的 CREATE、DROP、ALTER 操作。
第 2 位	是否审计 USER 对象的 CREATE、DROP、ALTER 操作。	❖ 0 表示不审计该对象的 CREATE、DROP、ALTER 操作； ❖ 1 表示审计该对象的 CREATE、DROP、ALTER 操作。
第 3 位	是否审计 TABLE 对象的 CREATE、DROP、ALTER、TRUNCATE 操作。	❖ 0 表示不审计该对象的 CREATE、DROP、ALTER、TRUNCATE 操作； ❖ 1 表示审计该对象的 CREATE、DROP、ALTER、TRUNCATE 操作。
第 4 位	是否审计 INDEX 对象的 CREATE、DROP、ALTER 操作。	❖ 0 表示不审计该对象的 CREATE、DROP、ALTER 操作； ❖ 1 表示审计该对象的 CREATE、DROP、ALTER 操作。
第 5 位	是否审计 VIEW/MATVIEW 对象的 CREATE、DROP 操作。	❖ 0 表示不审计该对象的 CREATE、DROP 操作； ❖ 1 表示审计该对象的 CREATE、DROP 操作。

二进制位	含义	取值说明
第 6 位	是否审计 TRIGGER 对象的 CREATE、DROP、ALTER 操作。	❖ 0 表示不审计该对象的 CREATE、DROP、ALTER 操作； ❖ 1 表示审计该对象的 CREATE、DROP、ALTER 操作。
第 7 位	是否审计 PROCEDURE/FUNCTION 对象的 CREATE、DROP、ALTER 操作。	❖ 0 表示不审计该对象的 CREATE、DROP、ALTER 操作； ❖ 1 表示审计该对象的 CREATE、DROP、ALTER 操作。
第 8 位	是否审计 TABLESPACE 对象的 CREATE、DROP、ALTER 操作。	❖ 0 表示不审计该对象的 CREATE、DROP、ALTER 操作； ❖ 1 表示审计该对象的 CREATE、DROP、ALTER 操作。
第 9 位	是否审计 RESOURCE POOL 对象的 CREATE、DROP、ALTER 操作。	❖ 0 表示不审计该对象的 CREATE、DROP、ALTER 操作； ❖ 1 表示审计该对象的 CREATE、DROP、ALTER 操作
第 10 位	是否审计 WORKLOAD 对象的 CREATE、DROP、ALTER 操作。	❖ 0 表示不审计该对象的 CREATE、DROP、ALTER 操作； ❖ 1 表示审计该对象的 CREATE、DROP、ALTER 操作
第 11 位	保留	-
第 12 位	是否审计 DATA SOURCE 对象的 CRAETE、DROP、ALTER 操作。	❖ 0 表示不审计该对象的 CREATE、DROP、ALTER 操作； ❖ 1 表示审计该对象的 CREATE、DROP、ALTER 操作。
第 13 位	保留	-
第 14 位	是否审计 ROW LEVEL SECURITY 对象的	❖ 0 表示不审计该对象的 CREATE、DROP、ALTER 操作；

二进制位	含义	取值说明
	CREATE、DROP、ALTER 操作。	❖ 1 表示审计该对象的 CREATE、DROP、ALTER 操作。
第 15 位	是否审计 TYPE 对象的 CREATE、DROP、ALTER 操作。	❖ 0 表示不审计 TYPE 对象的 CREATE、DROP、ALTER 操作； ❖ 1 表示审计 TYPE 对象的 CREATE、DROP、ALTER 操作。
第 16 位	是否审计 TEXT SEARCH 对象（CONFIGURATION 和 DICTIONARY）的 CREATE、DROP、ALTER 操作。	❖ 0 表示不审计 TEXT SEARCH 对象的 CREATE、DROP、ALTER 操作； ❖ 1 表示审计 TEXT SEARCH 对象的 CREATE、DROP、ALTER 操作。
第 17 位	是否审计 DIRECTORY 对象的 CREATE、DROP、ALTER 操作。	❖ 0 表示不审计 DIRECTORY 对象的 CREATE、DROP、ALTER 操作； ❖ 1 表示审计 DIRECTORY 对象的 CREATE、DROP、ALTER 操作。
第 18 位	是否审计 SYNONYM 对象的 CREATE、DROP、ALTER 操作。	❖ 0 表示不审计 SYNONYM 对象的 CREATE、DROP、ALTER 操作； ❖ 1 表示审计 SYNONYM 对象的 CREATE、DROP、ALTER 操作。
第 19 位	是否审计 SEQUENCE 对象的 CREATE、DROP、ALTER 操作。	❖ 0 表示不审计 SEQUENCE 对象的 CREATE、DROP、ALTER 操作； ❖ 1 表示审计 SEQUENCE 对象的 CREATE、DROP、ALTER 操作。
第 20 位	是否审计 CMK、CEK 对象的 CREATE、DROP 操作。	❖ 0 表示不审计 CMK、CEK 对象的 CREATE、DROP 操作；

二进制位	含义	取值说明
		❖ 1 表示审计 CMK、CEK 对象的 CREATE、DROP 操作。
第 21 位	是否审计 PACKAGE 对象的 CREATE、DROP、ALTER 操作。	❖ 0 表示不审计 PACKAGE 对象的 CREATE、DROP、ALTER 操作； ❖ 1 表示审计 PACKAGE 对象的 CREATE、DROP、ALTER 操作。
第 22 位	是否审计 MODEL 对象的 CREATE、DROP 操作。	❖ 0 表示不审计 MODEL 对象的 CREATE、ALTER 操作； ❖ 1 表示审计 MODEL 对象的 CREATE、DROP 操作。
第 23 位	是否审计 PUBLICATION 和 SUBSCRIPTION 对象的 CREATE、DROP、ALTER 操作。	❖ 0 表示不审计 PUBLICATION 和 SUBSCRIPTION 对象的 CREATE、DROP、ALTER 操作； ❖ 1 表示审计 PUBLICATION 和 SUBSCRIPTION 对象的 CREATE、DROP、ALTER 操作。
第 24 位	是否审计对 gs_global_config 全局对象的 ALTER、DROP 操作。	❖ 0 表示不审计对系统表 gs_global_config 全局对象的 ALTER、DROP 操作； ❖ 1 表示审计对系统表 gs_global_config 全局对象的 ALTER、DROP 操作。
第 25 位	是否审计 FOREIGN DATA WRAPPER 对象的 CREATE、DROP、ALTER 操作。	❖ 0 表示不审计 FOREIGN DATA WRAPPER 对象的 CREATE、DROP、ALTER 操作； ❖ 1 表示审计 FOREIGN DATA WRAPPER 对象的 CREATE、DROP、ALTER 操作。
第 26 位	是否审计安全标签的	❖ 0 表示不审计安全标签的

二进制位	含义	取值说明
	CREATE、DROP、ALTER 操作。	CREATE、DROP、ALTER 操作； ❖ 1 表示审计安全标签的 CREATE、DROP、ALTER 操作。
第 27 位	是否审计 SQL PATCH 对象的 CREATE、ENABLE、DISABLE、DROP 操作。	❖ 0 表示不审计 SQL PATCH 对象的 CREATE、ENABLE、DISABLE、DROP 操作； ❖ 1 表示审计 SQL PATCH 对象的 CREATE、ENABLE、DISABLE、DROP 操作。
第 28 位	是否审计 EVENT 对象的 CREATE、DROP、ALTER 操作。	❖ 0 表示不审计 EVENT 对象的 CREATE、DROP、ALTER 操作； ❖ 1 表示审计 EVENT 对象的 CREATE、DROP、ALTER 操作。

audit_dml_state

参数说明：这个参数决定是否对具体表的 INSERT、UPDATE、DELETE 操作进行审计。

该参数属于 SIGHUP 类型参数，请参考：配置运行参数->重设参数章节中表 1 对应设置方法进行设置。

取值范围：整型，0、1。

❖ 0 表示关闭具体表的 DML 操作（SELECT 除外）审计功能。
❖ 1 表示开启具体表的 DML 操作（SELECT 除外）审计功能。

默认值：0

audit_dml_state_select

参数说明：这个参数决定是否对 SELECT 操作进行审计。

该参数属于 SIGHUP 类型参数，请参考：配置运行参数->重设参数章节中表 1 对应设置方法进行设置。

取值范围：整型，0、1。

- 0 表示关闭 SELECT 操作审计功能。
- 1 表示开启 SELECT 审计操作功能。

默认值：0

audit_function_exec

参数说明：这个参数决定在执行存储过程、匿名块或自定义函数（不包括系统自带函数）时是否记录审计信息。

该参数属于 SIGHUP 类型参数，请参考：配置运行参数->重设参数章节中表 1 对应设置方法进行设置。

取值范围：整型，0、1。

- 0 表示关闭过程或函数执行的审计功能。
- 1 表示开启过程或函数执行的审计功能。

默认值：0

audit_system_function_exec

参数说明： 该参数决定在执行白名单内的系统函数时是否记录审计日志。

该参数属于 SIGHUP 类型参数，请参考重设参数表 1 中对应设置方法进行设置。

取值范围： 整型，0、1。

- 0 表示关闭对系统函数执行的审计功能。

❖ 1 表示开启对系统函数执行的审计功能。

默认值： 0

支持记录审计的系统函数白名单如下表所示：

pg_stat_activity	pg_terminate_backend	set_config	pg_cancel_backend	pg_cancel_session	pg_cancel_invalid_query
pg_reload_conf	pg_rotate_logfile	pg_terminate_session	pg_create_restore_point	pg_start_backup	pg_stop_backup
pg_switch_xlog	pg_cbm_get_merged_file	pg_cbm_recycle_file	pg_enable_delay_ddl_recycle	pg_disable_delay_ddl_recycle	pg_cbm_rotate_file
gs_roach_stop_backup	gs_roach_enable_delay_ddl_recycle	gs_roach_disable_delay_ddl_recycle	gs_roach_switch_xlog	pg_last_xlog_receive_location	pg_xlog_replay_pause
pg_create_physical_replication_slot_extern	gs_set_obs_delete_location	gs_hadr_do_switchover	gs_set_obs_delete_location_with_slotname	gs_streaming_dr_in_switchover	pg_advisory_lock
pg_advisory_lock_shared	pg_advisory_unlock	pg_advisory_unlock_shared	pg_advisory_unlock_all	pg_advisory_xact_lock	pg_advisory_xact_lock_shared
pg_try_advisory_lock	pg_try_advisory_lock_shared	pg_try_advisory_xact_lock	pg_try_advisory_xact_lock_shared	lock_cluster_ddl	unlock_cluster_ddl
pg_create_lo	pg_drop_r	pg_logical	pg_logical_s	pg_logical	pg_repli

gical_replication_slot	eplication_slot	_slot_peek_changes	lot_get_changes	_slot_get_binary_changes	cation_origin_create
pg_replication_origin_drop	pg_replication_origin_session_setup	pg_replication_origin_session_reset	pg_replication_origin_advance	local_space_shrink	gs_space_shrink
global_space_shrink	pg_free_remain_segment	gs_fault_inject	gs_repair_file	local_clear_bad_block_info	remote_clear_bad_block_info
gs_repair_page					

audit_copy_exec

参数说明：这个参数决定是否对 COPY 操作进行审计。

该参数属于 SIGHUP 类型参数，请参考：配置运行参数->重设参数章节中表 1 对应设置方法进行设置。

取值范围：整型，0、1。

- 0 表示关闭 COPY 审计功能。
- 1 表示开启 COPY 审计功能。

默认值：1

audit_set_parameter

参数说明：这个参数决定是否对 SET 操作进行审计。

该参数属于 SIGHUP 类型参数，请参考：配置运行参数->重设参数章节中表 1 对应设置方法进行设置。

取值范围：整型，0、1。

- 0 表示关闭 SET 审计功能。
- 1 表示开启 SET 审计功能。

默认值：0

audit_xid_info

参数说明：这个参数决定是否在审计日志字段 detail_info 中记录 SQL 语句的事务 ID。

该参数属于 SIGHUP 类型参数，请参考：配置运行参数->重设参数章节中表 1 对应设置方法进行设置。

取值范围：整型，0、1。

- 0 表示关闭审计日志记录事务 ID 功能。
- 1 表示开启审计日志记录事务 ID 功能。

默认值：0

【须知】
如果开启此开关，审计日志中 detail_info 信息则以 xid 开始，例如：
detail_info: xid=14619 , create table t1(id int);
对于不存在事务 ID 的审计行为，记录 xid=NA。

enable_Separation_Of_Duty

参数说明：是否开启三权分立选项。

该参数属于 POSTMASTER 类型参数，请参考：配置运行参数->重设参数章节中表 1 对应设置方法进行设置。

取值范围：布尔型

- on 表示开启三权分立。
- off 表示不开启三权分立。

默认值：off

enable_nonsysadmin_execute_direct

参数说明：是否允许非系统管理员和非监控管理员执行 EXECUTE DIRECT ON 语句。

该参数属于 POSTMASTER 类型参数，请参考：配置运行参数->重设参数章节中表 1 对应设置方法进行设置。

取值范围：布尔型

- on 表示允许任意用户执行 EXECUTE DIRECT ON 语句。
- off 表示只允许系统管理员和监控管理员执行 EXECUTE DIRECT ON 语句。

默认值：off

enable_access_server_directory

参数说明：是否开启非初始用户创建、修改和删除 DIRECTORY 的权限。

该参数属于 SIGHUP 类型参数，请参考：配置运行参数->重设参数章节中表 1 对应设置方法进行设置。

取值范围：布尔型

- on 表示开启非初始用户创建、修改和删除 DIRECTORY 的权限。
- off 表示不开启非初始用户创建、修改和删除 DIRECTORY 的权限。

默认值：off

【须知】

- 出于安全考虑，默认情况下，只有初始用户才能够创建、修改和删除 DIRECTORY 对象。
- 如果开启了 enable_access_server_directory，具有 SYSADMIN 权限的用户和继承了内置角色 gs_role_directory_create 权限的用户可以创建 directory 对象；具有 SYSADMIN 权限的用户、directory 对象的属主、被授予了该 directory 的 DROP 权限的用户或者继承了内置角色 gs_role_directory_drop 权限的用户可以删除 directory 对象；具有 SYSADMIN 权限的用户和 directory 对象的属主可以修改 directory 对象的所有者，且要求该用户是新属主的成员。

ogmac_enable_mac

参数说明： 控制强制访问控制功能是否开启。

该参数属于 SIGHUP 类型参数，请参考重设参数表 1 中对应设置方法进行设置。

取值范围： 布尔型

- on：开启强制访问控制功能。
- off：关闭强制访问控制功能。

默认值： off

【说明】

一旦对某个表授予了标签，即使关闭了访问控制开关，后面再对此表进行增删改查，依然会进行强制访问控制。此设计是为了避免同一个表里同时出现带安全属性和行和不带安全属性的行。

audit_transaction

参数说明： 该参数用于控制是否启用 transaction 的审计。

该参数属于 SIGHUP 类型参数，请参考重设参数表 1 中对应设置方法进行设置。

取值范围： 整型，0 或 1

- 0：关闭 transaction 的审计。
- 1：启用 transaction 的审计。

默认值： 0

12.23.升级参数

IsInplaceUpgrade

参数说明：标示是否在升级的过程中。该参数用户无法修改。

该参数属于 SUSET 类型参数，请参考：配置运行参数->重设参数章节中表 1 对应设置方法进行设置。

取值范围：布尔型

- on 表示在升级过程中。
- off 表示不在升级过程中。

默认值：off

inplace_upgrade_next_system_object_oids

参数说明：标示就地升级过程中，新增系统对象的 OID。该参数用户无法修改。

该参数属于 SUSET 类型参数，请参考：配置运行参数->重设参数章节中表 1 对应设置方法进行设置。

取值范围：字符串

默认值：空

upgrade_mode

参数说明：标示升级模式。

该参数属于 INTERNAL 类型参数，为固定参数，用户无法修改此参数，只能查看。

取值范围：整数，0~INT_MAX

- 0 表示不在升级过程中。
- 1 表示在就地升级过程中。
- 2 表示在灰度升级过程中。

默认值：0

【说明】
特殊情况：在使用灰度升级的情况下，若选择策略为大版本升级，即需要执行升级脚本和替换二进制包，会将 upgrade_mode 设置为 2，选择策略为小版本升级，只替换二进制包，则不会设置 upgrade_mode 设置为 2。

12.24.其他选项

enable_default_ustore_table

参数说明：指定是否开启默认支持 Ustore 存储引擎。该参数为 on 时，创建的表类型都为 Ustore 表。

该参数属于 USERSET 类型，为固定参数，用户无法修改此参数，只能查看。

取值范围：[off,on]

默认值：off

enable_obj_reuse

功能描述： 控制是否开启客体（主要指数据库对象、数据文件、缓存区）重用功能。包含如下内容：

- 内存重用：PanWeiDB 从系统分配内存及释放内存时均对内存内容进行清零，以保证不利用内存中前一线程所残留内容，且不泄漏 PanWeiDB 的内容给其他线程。
- 文件重用：PanWeiDB 在系统生成、扩展及删除文件时，对文件内容也进行了清零。

该参数属于 SIGHUP 类型参数，请参考重设参数表 1 中对应设置方法进行设置。

取值范围： 布尔型

- on：表示开启客体重用功能。
- off：表示不开启客体重用功能。

默认值： off

reserve_space_for_nullable_atts

参数说明：指定是否为 Ustore 表的可空属性预留空间。该参数为 on 时默认为 Ustore 表的可空属性预留空间。

该参数属于 USERSET 类型，请参考：配置运行参数->重设参数章节中表 1 对应设置方法进行设置。

取值范围：[off,on]

默认值：on

ustore_attr

参数说明：Ustore 测试参数。

该参数属于 USERSET 类型，可以设置包括 enable_ustore_partial_seqscan（仅在 ustore 表中顺序扫描时复制选择性列）、enable_candidate_buf_usage_count（是否脏页淘汰加入使用次数权重）、ustats_tracker_naptime（重新加载统计文件所用的时间）、

umax_search_length_for_prune（扩展表前要修剪的块数）、ustore_unit_test（开启 Ustore 白盒测试）。设置方法为 ustore_attr='需要设置的参数'，例如需要设置 enable_ustore_partial_seqscan 时，ustore_attr='enable_ustore_partial_seqscan=on'。

取值范围：空

server_version

参数说明：报告服务器版本号（字符串形式）。

该参数属于 INTERNAL 类型参数，为固定参数，用户无法修改此参数，只能查看。该参数继承自 PostgreSQL 内核，表示当前数据库内核兼容 PostgreSQL 对应的 server_version 版本，无实际含义，为保持北向对外工具接口的生态兼容性（工具连接时查询），保留该参数。该参数不推荐使用，如想查询服务器版本号，可通过函数 PanWeiDB_version()获取。

取值范围：字符串

默认值：9.2.4

server_version_num

参数说明：报告服务器版本号（整数形式）。

该参数属于 INTERNAL 类型参数，为固定参数，用户无法修改此参数，只能查看。该参数为 PanWeiDB 版本的整数形式。例如 PanWeiDB 的版本为 9.2.4，那么这个参数的值为 90204.

取值范围：整数

默认值：90204

block_size

参数说明：报告当前数据库所使用的块大小。

该参数属于 INTERNAL 类型参数，为固定参数，用户无法修改此参数，只能查看。

取值范围：8192

默认值：8192

segment_size

参数说明：报告当前数据库所使用的段文件大小。

该参数属于 INTERNAL 类型参数，为固定参数，用户无法修改此参数，只能查看。

单位: 8KB

默认值：131072，即 1GB

max_index_keys

参数说明：报告当前数据库能够支持的索引键值的最大数目。

该参数属于 INTERNAL 类型参数，为固定参数，用户无法修改此参数，只能查看。

默认值：32

integer_datetimes

参数说明：报告是否支持 64 位整数形式的日期和时间格式。

该参数属于 INTERNAL 类型参数，为固定参数，用户无法修改此参数，只能查看。

取值范围：布尔型

- on 表示支持。
- off 表示不支持。

默认值：on

lc_collate

参数说明：报告当前数据库的字符串排序区域设置。

该参数属于 INTERNAL 类型参数，为固定参数，用户无法修改此参数，只能查看。

默认值：依赖于 PanWeiDB 安装部署时的配置。

lc_ctype

参数说明：报告当前数据库的字母类别区域设置。如：哪些字符属于字母，它对应的大写形式是什么。

该参数属于 INTERNAL 类型参数，为固定参数，用户无法修改此参数，只能查看。

默认值：依赖于 PanWeiDB 安装部署时的配置。

max_identifier_length

参数说明：报告当前系统允许的标识符最大长度。

该参数属于 INTERNAL 类型参数，为固定参数，用户无法修改此参数，只能查看。

取值范围：整型

默认值：127

server_encoding

参数说明：报告当前数据库的服务端编码字符集。

默认情况下，pw_initdb 会根据当前的系统环境初始化此参数，通过 locale 命令可以查看当前的配置环境。

该参数属于 INTERNAL 类型参数，为固定参数，用户无法修改此参数，只能查看。

默认值：在创建数据库的时候由当前系统环境决定的。

enable_upgrade_merge_lock_mode

参数说明：当该参数设置为 on 时，通过提升 deltamerge 内部实现的锁级别，避免和 update/delete 并发操作时的报错。

该参数属于 USERSET 类型参数，请参考：配置运行参数->重设参数章节中表 1 对应设置方法进行设置。

取值范围： 布尔型

- on：表示提升 deltamerge 内部实现的锁级别，并发执行 deltamerge 和 update/delete 操作时，一个操作先执行，另一个操作被阻塞，在前一个操作完成后，后一个操作再执行。
- off：表示在对表的 delta table 的同一行并发执行 deltamerge 和 update/delete 操作时，后一个对同一行数据更新的操作会报错退出。

默认值：off

transparent_encrypted_string

参数说明：它存储的是透明加密的一个样本串，使用数据库加密密钥加密固定串“TRANS_ENCRYPT_SAMPLE_STRING”后的密文，用来校验二次启动时获取的 DEK 是否正确。如果校验失败，那么数据库节点将拒绝启动。

该参数属于 POSTMASTER 类型参数，请参考：配置运行参数->重设参数章节中表 1 对应设置方法进行设置。该参数当前版本只适用于 DWS 场景。

取值范围：字符串，设置为空表示 PanWeiDB 非加密。

默认值：空

【说明】
请勿手动设置该参数，设置不当将导致 PanWeiDB 不可用。

transparent_encrypt_kms_url

参数说明：它存储的是透明加密的数据库密钥获取地址，内容要求不可出现 RFC3986 标准外的字符，最大长度 2047 字节。格式为“kms://协议@KMS 主机名 1;KMS 主机名 2:KMS 端口号/kms”，例如 kms://https@linux175:29800/。该参数当前版本只适用于 DWS 场景。

该参数属于 POSTMASTER 类型参数，请参考：配置运行参数->重设参数章节中表 1 对应设置方法进行设置。

取值范围：字符串

默认值：空

transparent_encrypt_kms_region

参数说明：它存储的是 PanWeiDB 的部署区域，内容要求不可出现 RFC3986 标准外的字符，最大长度 2047 字节。该参数当前版本只适用于 DWS 场景。

该参数属于 POSTMASTER 类型参数，请参考：配置运行参数->重设参数章节中表 1 对应设置方法进行设置。

取值范围：字符串

默认值：空

basebackup_timeout

参数说明：备份传输完成后连接无读写的超时时间。

通过 pw_basebackup 工具作传输时，如果指定较高压缩率时，可能在传输表空间完成后超时（客户端需要压缩传输数据）。

取值范围：整型，0～INT_MAX，单位为秒。其中 0 表示禁用该功能。

默认值：600s

datanode_heartbeat_interval

参数说明：设置心跳线程间心跳消息发送时间间隔，建议值不超过 wal_receiver_timeout / 2。

该参数属于 SIGHUP 类型参数，请参考：配置运行参数->重设参数章节中表 1 对应设置方法进行设置。

取值范围：整型，1000～60000，单位为毫秒。

默认值：1s

lower_case_column_names

参数说明：该参数用于控制数据库返回字段名的大小写敏感性。详细内容请参考设置字段名大小写敏感。

该参数属于 POSTMASTER 类型参数，请参考：配置运行参数->重设参数章节中表 1 对应设置方法进行设置。

【说明】
该参数仅在数据库兼容模式为 MySQL 时能够使用（即数据库实例初始化时指定 DBCOMPATIBILITY='B'），在其他数据库兼容模式下不能使用该特性。

取值范围：0、1

- 0：开启大小写敏感。

❖ 1：关闭大小写敏感。

默认值：0

enable_double_type

参数说明： 该参数用于控制数据库是否支持 double 类型，默认不支持 double 类型，需开启 enable_double_type 参数才可支持。

该参数属于 SIGHUP 类型参数，请参考重设参数表 1 中对应设置方法进行设置。

取值范围： 布尔型

❖ on：支持 double 类型。
❖ off：不支持 double 类型。

默认值： off

acceleration_with_compute_pool

参数说明： 在查询包含 OBS 时，通过该参数决定查询是否通过计算资源池进行加速。

该参数属于 USERSET 类型参数，请参考重设参数表 1 中对应设置方法进行设置。

取值范围： 布尔型

❖ on：表示包含有 OBS 或的查询在计算资源池可用时，会根据代价评估决定是否通过计算资源池对查询加速。
❖ off：表示任何查询都不会通过计算资源池进行加速。

默认值： off

enable_connectby_dfs

参数说明：控制在`connect by`语句里是否启用 dfs。

该参数属于 USERSET 类型参数，请参考重设参数表 1 中对应设置方法进行设置。

取值范围：布尔型

- on：`connect by`语句里启用 dfs。
- off：`connect by`语句里不启用 dfs。

默认值：on

dfs_partition_directory_length

参数说明： 在 HDFS 文件系统上，构造 HDFS VALUE 分区表的分区目录时，目录名长度的上限值。

该参数属于 USERSET 类型参数，请参考重设参数表 1 中对应设置方法进行设置。

取值范围： 92-7999

默认值： 512

max_resource_package

参数说明： 加速数据库实例每个 DN 可同时运行任务的线程数的上限。

该参数属于 POSTMASTER 类型参数，请参考重设参数表 1 中对应设置方法进行设置。

取值范围： 0~2147483647

默认值： 0

acceleration_with_compute_pool

参数说明： 在查询包含 OBS 时，通过该参数决定查询是否通过计算资源池进行加速。（由于规格变更，当前版本已经不再支持本特性，请不要使用）

该参数属于 USERSET 类型参数，请参考重设参数表 1 中对应设置方法进行设置。

取值范围： 布尔型

- on 表示包含有 OBS 或的查询在计算资源池可用时，会根据代价评估决定是否通过计算资源池对查询加速。
- off 表示任何查询都不会通过计算资源池进行加速。

enable_segment

参数说明： 指定是否默认使用段页式存储。需要注意的是，在资源池化下安装过程中，会自动设置为 on，提升易用性。

该参数属于 SIGHUP 类型参数，请参考重设参数表 1 中对应设置方法进行设置。

【须知】
资源池化模式下，此参数默认值为 on。表示使用段页式存储。

取值范围： 布尔型

- on：默认使用段页式存储。
- off：不使用段页式存储。

默认值： off

bonjour_name

参数说明： 设置 Bonjour 服务的名称。

该参数属于 POSTMASTER 类型参数，请参考重设参数表 1 中对应设置方法进行设置。该参数当前版本只适用于 DWS 场景。

取值范围： 字符串。

默认值： “”

role

参数说明： 设置当前角色。该参数无法在配置文件中配置。

该参数属于 USERSET 类型参数，请参考重设参数表 1 中对应设置方法进行设置。

取值范围： 字符串。

默认值： role

12.25.等待事件

enable_instr_track_wait

参数说明：是否开启等待事件信息实时收集功能。

在 x86 平台集中式部署下，硬件配置规格为 32 核 CPU/256GB 内存，使用 Benchmark SQL 5.0 工具测试性能，开关此参数性能影响约 1.4%。

该参数属于 SIGHUP 类型参数，请参考：配置运行参数->重设参数章节中表 1 对应设置方法进行设置。

取值范围：布尔型

- on：表示打开等待事件信息收集功能。
- off：表示关闭等待事件信息收集功能。

默认值：on

acceleration_with_compute_pool

参数说明： 在查询包含 OBS 时，通过该参数决定查询是否通过计算资源池进行加速。

该参数属于 USERSET 类型参数，请参考重设参数表 1 中对应设置方法进行设置。

取值范围： 布尔型

- on：表示包含有 OBS 或的查询在计算资源池可用时，会根据代价评估决定是否通过计算资源池对查询加速。
- off：表示任何查询都不会通过计算资源池进行加速。

默认值： off

12.26.Query

instr_unique_sql_count

参数说明：控制系统中 unique sql 信息实时收集功能。配置为 0 表示不启用 unique sql 信息收集功能。

该值由大变小将会清空系统中原有的数据重新统计（备机不支持此能力）；从小变大不受影响。

当系统中产生的 unique sql 条目数量（dbe_perf.statement/dbe_perf.summary_statement 统计）大于 instr_unique_sql_count 时，若开启了 unique sql 自动淘汰，则系统会按 unique sql 的更新时间由远到近自动淘汰一定比例的条目，使得新产生的 unique sql 信息可以继续被统计。若没有开启自动淘汰，则系统产生的新的 unique sql 信息将不再被统计。

在 x86 平台集中式部署下，硬件配置规格为 32 核 CPU/256GB 内存，使用 Benchmark SQL 5.0 工具测试性能，开关此参数性能影响约 3%。

该参数属于 SIGHUP 类型参数，请参考：配置运行参数->重设参数章节中表 1 对应设置方法进行设置。

取值范围：整型，0~2147483647

默认值：100

【须知】

- 在开启自动淘汰的情况下，如果该值设置的较小，可能会导致系统频繁的进行自动淘汰，有可能会影响数据库系统性能，所以实际场景中建议不要将该值设置的过小，建议值为 200000。
- 在开启自动淘汰的情况下，如果该值设置的较大（例如 38347922），清理过程中可能会引发大内存问题而无法清理。

instr_unique_sql_track_type

参数说明：unique sql 记录 SQL 方式。

该参数属于 INTERNAL 类型参数，请参考：配置运行参数->重设参数章节中表 1 对应设置方法进行设置。

取值范围：枚举类型

top：代表只记录顶层 SQL。

默认值：top

enable_instr_rt_percentile

参数说明：是否开启计算系统中 80%和 95%的 SQL 响应时间的功能。

该参数属于 SIGHUP 类型参数，为固定参数，用户无法修改此参数，只能查看。

取值范围：布尔型

- on：表示打开 SQL 响应时间信息计算功能。

❖ off：表示关闭 SQL 响应时间信息计算功能。

默认值：on

percentile

参数说明：SQL 响应时间百分比信息，后台计算线程根据设置的值计算相应的百分比信息。

该参数属于 INTERNAL 类型参数，请参考：配置运行参数->重设参数章节中表 1 对应设置方法进行设置。

取值范围：字符串。

默认值：80,95

instr_rt_percentile_interval

参数说明：SQL 响应时间信息计算间隔，SQL 响应时间信息计算功能打开后，后台计算线程每隔设置的时间进行一次计算。

该参数属于 SIGHUP 类型参数，请参考：配置运行参数->重设参数章节中表 1 对应设置方法进行设置。

取值范围：整型，0～3600，单位为秒。

默认值：10s

enable_instr_cpu_timer

参数说明：是否捕获 SQL 执行的 cpu 时间消耗 。

在 x86 平台集中式部署下，硬件配置规格为 32 核 CPU/256GB 内存，使用 Benchmark SQL 5.0 工具测试性能，开关此参数性能影响约 3.5%。

该参数属于 SIGHUP 类型参数，请参考：配置运行参数->重设参数章节中表 1 对应设置方法进行设置。

取值范围：布尔型

- on：表示捕获 SQL 执行的 CPU 时间消耗。
- off：表示不捕获 SQL 执行的 CPU 时间消耗。

默认值：on

enable_stmt_track

参数说明：控制是否启用 Full /Slow SQL 特性。

在 x86 平台集中式部署下，硬件配置规格为 32 核 CPU/256GB 内存，使用 Benchmark SQL 5.0 工具测试性能，开关此参数性能影响约 1.2%。

该参数属于 SIGHUP 类型参数，请参考：配置运行参数->重设参数章节中表 1 对应设置方法进行设置。

取值范围：布尔型

- on：表示开启 Full /Slow SQL 捕获。
- off：表示关闭 Full /Slow SQL 捕获。

默认值：on

track_stmt_session_slot

参数说明：设置一个 session 缓存的最大的全量/慢 SQL 的数量，超过这个数量，新的语句执行将不会被跟踪，直到落盘线程将缓存语句落盘，留出空闲的空间。

该参数属于 SIGHUP 类型参数，请参考：配置运行参数->重设参数章节中表 1 对应设置方法进行设置。

取值范围：整型，0 ~ 2147483647

默认值：1000

track_stmt_details_size

参数说明：设置单语句可以收集的最大的执行事件的大小（byte）。

该参数属于 USERSET 类型参数，请参考：配置运行参数->重设参数章节中表 1 对应设置方法进行设置。

取值范围：整型，0 ~ 100000000

默认值：4096

track_stmt_retention_time

参数说明：组合参数，控制全量/慢 SQL 记录的保留时间。以 60 秒为周期读取该参数，并执行清理超过保留时间的记录，仅 sysadmin 用户可以访问。

该参数属于 SIGHUP 类型参数，请参考：配置运行参数->重设参数章节中表 1 对应设置方法进行设置。

取值范围：字符型

该参数分为两部分，形式为'full sql retention time, slow sql retention time'

full sql retention time 为全量 SQL 的保留时间，取值范围为 0 ~ 86400

slow sql retention time 为慢 SQL 的保留时间，取值范围为 0 ~ 604800

默认值：3600,604800

track_stmt_stat_level

参数说明：控制语句执行跟踪的级别。

该参数属于 USERSET 类型参数，请参考：配置运行参数->重设参数章节中表 1 对应设置方法进行设置，不区分英文字母大小写。

取值范围：字符型

该参数分为两部分，形式为'full sql stat level, slow sql stat level'

full sql stat level 为全量 SQL 跟踪级别，取值范围为 OFF、L0、L1、L2

slow sql stat level 为慢 SQL 的跟踪级别，取值范围为 OFF、L0、L1、L2

【说明】

若全量 SQL 跟踪级别值为非 OFF 时，当前 SQL 跟踪级别值为全量 SQL 和慢 SQL 的较高级别（L2 > L1 > L0），级别说明请参见：系统表和系统视图->系统表->STATEMENT_HISTORY 中表 1。

默认值：OFF,L0

enable_auto_clean_unique_sql

参数说明：当系统中产生的 unique sql 条目数量大于等于 instr_unique_sql_count 时，是否启用 unique sql 自动淘汰功能。

该参数属于 POSTMASTER 类型参数，请参考：配置运行参数->重设参数章节中表 1 对应设置方法进行设置。

取值范围：布尔型

默认值：off

【注意】

由于快照有部分信息是来源于 unique sql，所以开启自动淘汰的情况下，在生成 wdr 报告时，如果选择的起始快照和终止快照跨过了淘汰发生的时间，会导致无法生成 wdr 报告。

enable_gtt_concurrent_truncate

参数说明： 是否支持全局临时表 truncate table 和 DML 的并发执行，以及全局临时表 truncate table 和普通表的 truncate table 的并发执行。

该参数属于 SIGHUP 类型参数，请参考重设参数表 1 中对应设置方法进行设置。

取值范围： 布尔型

- on：表示支持上述操作并发。
- off：表示不支持上述操作并发。

默认值： on

unique_sql_clean_ratio

参数说明： 当系统中产生的 unique sql 条目数量大于等于 instr_unique_sql_count 时，每次自动淘汰的 unique sql 条目数量占总条目数量预设上限 instr_unique_sql_count 的比例。

该参数属于 POSTMASTER 类型参数，请参考重设参数表 1 中对应设置方法进行设置。

取值范围： double 类型，0~0.2

默认值： 0.1

【说明】

- 该值设置过小每次清理的条目较少，可能会导致频繁进行清理；设置过大时每次清理的条目较多，可能会导致频繁插入。建议值 0.1。
- unique_sql_clean_ratio 设置为 0 不代表关闭自动淘汰功能，请通过 enable_auto_clean_unique_sql 来控制是否开启自动淘汰。当开启自动淘汰，且将 unique_sql_clean_ratio 设置为 0 时，将自动把 unique_sql_clean_ratio 重置为默认值。

enable_slow_query_log（废弃）

参数说明： 是否将慢查询信息写到日志文件中，在该版本中已废弃。

该参数属于 SIGHUP 类型参数，请参考重设参数表 1 中对应设置方法进行设置。

取值范围： 布尔型

- on：表示需要将慢查询信息写到日志文件中。

❖ off：表示不需要将慢查询信息写到日志文件中。

默认值： on

query_log_file（废弃）

参数说明： GUC 参数 enable_slow_query_log 设置为 ON，表示需要将慢查询记录写进日志文件中，query_log_file 决定服务器慢查询日志文件的名称，仅 sysadmin 用户可以访问。通常日志文件名是按照 strftime 模式生成，因此可以用系统时间定义日志文件名，用%转义字符实现，在该版本中已废弃。

该参数属于 SIGHUP 类型参数，请参考重设参数表 1 中对应设置方法进行设置。

【说明】
建议使用%转义字符定义日志文件名称，否则难以对日志文件进行有效的管理。

取值范围： 字符串

默认值： slow_query_log-%Y-%m-%d_%H%M%S.log

enable_csqual_pushdown

参数说明： 进行查询时，是否要将过滤条件下推，进行 Rough Check。

该参数属于 SUSET 类型参数，请参考重设参数表 1 中对应设置方法进行设置。

取值范围： 布尔型

❖ on：表示进行查询时，要将过滤条件下推，进行 Rough Check。

❖ off：表示进行查询时，不要将过滤条件下推，进行 Rough Check。

默认值： on

asp_log_directory

参数说明：asp_flush_mode 设置为 all 或者 file 时，asp_log_directory 决定存放服务器 asp 日志文件的目录。它可以是绝对路径，或者是相对路径（相对于数据目录的路径），仅 sysadmin 用户可以访问。

该参数属于 POSTMASTER 类型参数，请参考重设参数表 1 中对应设置方法进行设置。

【须知】
当配置文件中 asp_log_directory 的值为非法路径时，会导致数据库实例无法重新启动。

【说明】
- 合法路径：用户对此路径有读写权限。
- 非法路径：用户对此路径无读写权限。

取值范围：字符串。

默认值：安装时指定。

query_log_directory（废弃）

参数说明：enable_slow_query_log 设置为 on 时，query_log_directory 决定存放服务器慢查询日志文件的目录，仅 sysadmin 用户可以访问。它可以是绝对路径，或者是相对路径（相对于数据目录的路径），在该版本中已废弃。

该参数属于 POSTMASTER 类型参数，请参考重设参数表 1 中对应设置方法进行设置。

【须知】
当配置文件中 query_log_directory 的值为非法路径时，会导致数据库实例无

法重新启动。

【说明】

- 合法路径：用户对此路径有读写权限。
- 非法路径：用户对此路径无读写权限。

取值范围：字符串

默认值：安装时指定

unique_sql_retention_time

参数说明：清理 unique sql 哈希表的间隔。

该参数属于 SIGHUP 类型参数，请参考重设参数表 1 中对应设置方法进行设置。

取值范围：整型，1~3650，单位 min。

默认值：30min。

perf_directory

参数说明：perf_directory 决定性能视图的输出文件的目录，仅 sysadmin 用户可以访问。它可以是绝对路径，或者相对路径（相对于数据目录的路径）。

该参数属于 POSTMASTER 类型参数。请参考重设参数表 1 中对应设置方法进行设置。

- 合法路径：用户对此路径有读写权限。
- 非法路径：用户对此路径无读写权限。

取值范围：字符串

12.27.系统性能快照

enable_wdr_snapshot

参数说明：是否开启数据库监控快照功能。

该参数属于 SIGHUP 类型参数，请参考：配置运行参数->重设参数章节中表 1 对应设置方法进行设置。

取值范围：布尔型

- on：打开数据库监控快照功能。
- off：关闭数据库监控快照功能。

默认值：off

wdr_snapshot_retention_days

参数说明：系统中数据库监控快照数据的保留天数。当数据库运行过程期间所生成的快照量数超过保留天数内允许生成的快照数量的最大值（保留天数的分钟值 / 自动生成时间间隔的分钟值）时，系统将每隔 wdr_snapshot_interval 时间间隔，清理 snapshot_id 最小的快照数据。

该参数属于 SIGHUP 类型参数，请参考：配置运行参数->重设参数章节中表 1 对应设置方法进行设置。

取值范围：整型，1～8。

默认值：8

wdr_snapshot_query_timeout

参数说明：系统执行数据库监控快照操作时，设置快照操作相关的 sql 语句的执行超时时间。如果语句超过设置的时间没有执行完并返回结果，则本次快照操作失败。

该参数属于 SIGHUP 类型参数，请参考：配置运行参数->重设参数章节中表 1 对应设置方法进行设置。

取值范围：整型，100～INT_MAX（秒）。

默认值：100s

wdr_snapshot_interval

参数说明：后台线程 Snapshot 自动对数据库监控数据执行快照操作的时间间隔。

该参数属于 SIGHUP 类型参数，请参考：配置运行参数->重设参数章节中表 1 对应设置方法进行设置。

取值范围：整型，10～720（分钟）。

默认值：1h

asp_flush_mode

参数说明：ASP 刷新到磁盘上的方式分为写文件和写系统表，当为‘file’时，默认写文件，为‘table’时写系统表，为‘all’时，即写文件也写系统表，仅 sysadmin 用户可以访问。

该参数属于 SIGHUP 类型参数，请参考：配置运行参数->重设参数章节中表 1 对应设置方法进行设置。

取值范围：字符串，‘table’、‘file’、‘all’。

默认值：‘table’

asp_flush_rate

参数说明：当内存中样本个数达到 asp_sample_num 时，会按一定比例把内存中样本刷新到磁盘上，asp_flush_rate 为刷新比例。该参数为 10 时表示按 10：1 进行刷新。

该参数属于 SIGHUP 类型参数，请参考：配置运行参数->重设参数章节中表 1 对应设置方法进行设置。

取值范围：整型，1~10。

默认值：10

asp_log_filename

参数说明：当 ASP 写文件时，该参数设置文件名的格式，仅 sysadmin 用户可以访问。

该参数属于 SIGHUP 类型参数，请参考：配置运行参数->重设参数章节中表 1 对应设置方法进行设置。

取值范围：字符串。

默认值：“asp-%Y-%m-%d_%H%M%S.log”

asp_retention_days

参数说明：当 ASP 样本写到系统表时，该参数表示保留的最大天数。

该参数属于 SIGHUP 类型参数，请参考：配置运行参数->重设参数章节中表 1 对应设置方法进行设置。

取值范围：整型，1~7。

默认值：2

asp_sample_interval

参数说明：每次采样的间隔。

该参数属于 SIGHUP 类型参数，请参考：配置运行参数->重设参数章节中表 1 对应设置方法进行设置。

取值范围：整型，1~10，单位为秒。

默认值：1s

asp_sample_num

参数说明：LOCAL_ACTIVE_SESSION 视图最大的样本个数，仅 sysadmin 用户可以访问。

该参数属于 POSTMASTER 类型参数，请参考：配置运行参数->重设参数章节中表 1 对应设置方法进行设置。

取值范围：整型，10～100000。

默认值：100000

enable_asp

参数说明：是否开启活跃会话信息 active session profile。

该参数属于 SIGHUP 类型参数，请参考：配置运行参数->重设参数章节中表 1 对应设置方法进行设置。

取值范围：布尔型

- on：打开 active session profile 功能。
- off：关闭 active session profile 功能。

默认值：on

wdr_snapshot_auto_vacuum_full

参数说明： 用于开启在清理快照后对相关表进行 vacuum full 操作的功能。开启本功能后，wdr 相关统计表中的死元组会在原有每次的清理操作之后被清除。

该参数属于 SIGHUP 类型参数，请参考重设参数表 1 中对应设置方法进行设置。

取值范围： 布尔型。

on：开启清理快照后对相关表进行 vacuum full 操作的功能。

off：关闭清理快照后对相关表进行 vacuum full 操作的功能。

默认值： off

12.28.安全配置

elastic_search_ip_addr

参数说明：Elastic Search 系统 IP 地址。

该参数属于 POSTMASTER 类型参数，请参考：配置运行参数->重设参数章节中表 1 对应设置方法进行设置。

取值范围：字符串。

默认值：'https://127.0.0.1'

enable_security_policy

参数说明：安全策略开关，控制统一审计和数据动态脱敏策略是否生效。

该参数属于 SIGHUP 类型参数，请参考：配置运行参数->重设参数章节中表 1 对应设置方法进行设置。

取值范围：布尔型。

- on：安全策略开关打开。

❖ off：安全策略开关关闭。

默认值：off

use_elastic_search

参数说明：使能统一审计发送日志至 Elastic Search 系统，enable_security_policy 打开且本参数打开后，统一审计日志会通过 http（https）传递至 Elastic Search 系统（默认使用 https 安全协议）。此参数打开后需要保证 elastic_search_ip_addr 对应的 es 服务可正常连通，否则进程启动失败。

该参数属于 POSTMASTER 类型参数，请参考：配置运行参数->重设参数章节中表 1 对应设置方法进行设置。

取值范围：布尔型。

❖ on：使能统一审计日志发送至 Elastic Search。
❖ off：关闭统一审计日志发送至 Elastic Search。

默认值：off

is_sysadmin

参数说明：表示当前用户是否是初始用户。

该参数属于 INTERNAL 类型参数，为固定参数，用户无法修改此参数，只能查看。

取值范围：布尔型。

❖ on：是初始用户。
❖ off：不是初始用户。

默认值：on

enable_tde

参数说明：透明数据加密功能开关。创建加密表前需要将此参数置为 on。当前参数值为 off 时，禁止创建新的加密表，对于已经创建的加密表只在读取数据时解密，写入数据时不再加密。

该参数属于 POSTMASTER 类型参数，为固定参数，用户无法修改此参数，只能查看。

取值范围：布尔型。

- on：开启透明数据加密功能。
- off：关闭透明数据加密功能。

默认值：off

tde_cmk_id

参数说明：透明数据加密功能使用的数据库实例主密钥 CMK 的 ID 编号，由使用的密钥管理服务 KMS 生成。数据库实例主密钥 CMK 用于对数据加密密钥 DEK 进行加密保护，当前需要对 DEK 进行解密时，需要给 KMS 发起请求报文，将 DEK 密文和对应 CMK 的 ID 编号一起发送给 KMS。

该参数属于 SIGHUP 类型参数，请参考：配置运行参数->重设参数章节中表 1 对应设置方法进行设置。

取值范围：字符串。

默认值：空

default_with_secids

参数说明： 使用默认的安全标签创建新表。

该参数属于 USERSET 类型参数，请参考重设参数表 1 中对应设置方法进行设置。

取值范围：布尔型

- on：开启该参数，创建新表时则会使用默认的安全标签。
- off：关闭该参数，创建表不会使用默认的安全标签。

默认值：off

12.29.全局临时表

max_active_global_temporary_table

参数说明：全局临时表功能开关，控制是否可以创建全局临时表，暂不支持限制最大活跃临时表数。

该参数属于 POSTMASTER 类型参数，请参考：配置运行参数->重设参数章节中表 1 对应设置方法进行设置。

取值范围：整型，0 ~ 1000000

- 0：全局临时表功能关闭。
- 大于 0：全局临时表功能打开，数值不代表具体最大活跃全局临时表数。

默认值：1000

vacuum_gtt_defer_check_age

参数说明：vacuum 执行后检查全局临时表 relfrozenxid 与普通表的差异。如果全局临时表 relfrozenxid 落后超过指定参数值，就产生 WARNING。一般不用修改。

该参数 USERSET 类型参数，请参考：配置运行参数->重设参数章节中表 1 对应设置方法进行设置。

取值范围：整型，0 ~ 1000000

默认值：10000

12.30.HyperLogLog

hll_default_log2m

参数说明：该参数可以指定 hll 数据结构桶的个数。桶的个数会影响 hll 计算 distinct 值的精度，桶的个数越多，误差越小。误差范围为：[-1.04/2log2m*1/2,+1.04/2log2m*1/2]。

该参数属于 USERSET 类型参数，请参考：配置运行参数->重设参数章节中表 1 对应设置方法进行设置。

取值范围：整型，10~16。

默认值：14

hll_default_log2explicit

参数说明：该参数可以用来设置从 Explicit 模式到 Sparse 模式的默认阈值大小。

该参数属于 USERSET 类型参数，请参考：配置运行参数->重设参数章节中表 1 对应设置方法进行设置。

取值范围：整型，0~12。0 表示跳过 Explicit 模式，取 1-12 表示在基数到达 2hll_default_log2explicit 时切换模式。

默认值：10

hll_default_log2sparse

参数说明：该参数可以用来设置从 Sparse 模式到 Full 模式的默认阈值大小。

该参数属于 USERSET 类型参数，请参考：配置运行参数->重设参数章节中表 1 对应设置方法进行设置。

取值范围：整型，0~14。0 表示跳过 Explicit 模式，取 1-14 表示在基数到达 2hll_default_log2sparse 时切换模式。

默认值：12

hll_duplicate_check

参数说明：该参数可以用来指定是否默认开启 duplicatecheck。

该参数属于 USERSET 类型参数，请参考：配置运行参数->重设参数章节中表 1 对应设置方法进行设置。

取值范围：0，1。

- 0 表示默认关闭。
- 1 表示默认开启。

默认值：0

hll_default_regwidth（废弃）

参数说明：该参数可以指定 hll 数据结构每个桶的位数，该值越大，hll 所占内存越高。hll_default_regwidth 和 hll_default_log2m 可以决定当前 hll 能够计算的最大 distinct value。当前 regwidth 设为固定值，该参数不再使用。

该参数属于 USERSET 类型参数，请参考：配置运行参数->重设参数章节中表 1 对应设置方法进行设置。

取值范围：整型，1~5。

默认值：5

hll_default_expthresh（废弃）

参数说明：该参数可以用来设置从 Explicit 模式到 Sparse 模式的默认阈值大小。当前已经使用参数 hll_default_log2explicit 替代类似功能。

该参数属于 USERSET 类型参数，请参考：配置运行参数->重设参数章节中表 1 对应设置方法进行设置。

取值范围：整型，-1~7。-1 表示自动模式，0 表示跳过 Explicit 模式，取 1-7 表示在基数到达 2hll_default_expthresh 时切换模式。

默认值：-1

hll_default_sparseon（废弃）

参数说明：该参数可用来指定是否默认开启 Sparse 模式。当前已经使用参数 hll_default_log2sparse 替代类似功能，hll_default_log2sparse 设置为 0 时关闭 Sparse 模式。

该参数属于 USERSET 类型参数，请参考：配置运行参数->重设参数章节中表 1 对应设置方法进行设置。

取值范围：0，1。

- 0 表示默认关闭。
- 1 表示默认开启。

默认值：1

hll_max_sparse（废弃）

参数说明：该参数可以用来指定 max_sparse 的大小。当前已经使用参数 hll_default_log2sparse 替代类似功能。

该参数属于 USERSET 类型参数，请参考：配置运行参数->重设参数章节中表 1 对应设置方法进行设置。

取值范围：整型，-1~2147483647

默认值：-1

enable_compress_hll（废弃）

参数说明：该参数可以用来指定是否对 hll 开启内存优化模式。目前 hll 内存已经进行了优化设计，该参数不再使用。

该参数属于 USERSET 类型参数，请参考：配置运行参数->重设参数章节中表 1 对应设置方法进行设置。

取值范围：布尔型

- on/true 表示对 hll 开启内存优化模式。
- off/false 表示不开启内存优化模式。

默认值：off

12.31.用户自定义函数

udf_memory_limit

参数说明：控制每个数据库节点执行 UDF 时可用的最大物理内存量。本参数当前版本不生效，请使用 FencedUDFMemoryLimit 和 UDFWorkerMemHardLimit 参数控制 fenced udf worker 虚存。

该参数属于 POSTMASTER 类型参数，请参考：配置运行参数->重设参数章节中表 1 对应设置方法进行设置。

取值范围：整型，200*1024～max_process_memory，单位为 KB。

默认值：200MB

FencedUDFMemoryLimit

参数说明：控制每个 fenced udf worker 进程使用的虚拟内存。

该参数属于 USERSET 类型参数，请参考：配置运行参数->重设参数章节中表 1 对应设置方法进行设置。

取值范围：整数，0～2147483647，单位为 KB，设置可带单位（KB，MB，GB）。其中 0 表示不做内存控制。

默认值：0

UDFWorkerMemHardLimit

参数说明：控制 fencedUDFMemoryLimit 的最大值。

该参数属于 POSTMASTER 类型参数，请参考：配置运行参数->重设参数章节中表 1 对应设置方法进行设置。

取值范围：整数，0～2147483647，单位为 KB，设置时可带单位（KB，MB，GB）。

默认值：1GB

pljava_vmoptions

参数说明：用户自定义设置 PL/Java 函数所使用的 JVM 虚拟机的启动参数，仅 sysadmin 用户可以访问。

该参数属于 SUSET 类型参数，请参考：配置运行参数->重设参数章节中表 1 对应设置方法进行设置。

取值范围：字符串，支持：

- JDK8 JVM 启动参数（可参见 JDK 官方说明）
- JDK8 JVM 系统属性参数（以-D 开头如-Djava.ext.dirs，可参见 JDK 官方说明）
- 用户自定义参数（以-D 开头，如-Duser.defined.option）

【须知】

如果用户在 pljava_vmoptions 中设置参数不满足上述取值范围，会在使用 PL/Java 语言函数时报错。

默认值：空

12.32.定时任务

job_queue_processes

参数说明：表示系统可以并发执行的 job 数目。该参数为 postmaster 级别，通过 pw_guc 设置，需要重启才能生效。

该参数属于 POSTMASTER 类型参数，请参考：配置运行参数->重设参数章节中表 1 对应设置方法进行设置。

取值范围：0～1000

功能：

- 当 job_queue_processes 设置为 0 时，表示不启用定时任务功能，任何 job 都不会被执行（因为开启定时任务的功能会对系统的性能有影响，有些局点可能不需要定时任务的功能，可以通过设置为 0 不启用定时任务功能）。
- 当 job_queue_processes 设置为大于 0 时，表示启用定时任务功能且系统能够并发处理的最大任务数。

启用定时任务功能后，job_scheduler 线程会在定时时间间隔轮询 pg_job 系统表，系统设置定时任务检查周期默认为 1s。

由于并行运行的任务数太多会消耗更多的系统资源，因此需要设置系统并发处理的任务数，当前并发的任务数达到 job_queue_processes 时，且此时又有任务到期，那么这些任务本次得不到执行而延期到下一轮询周期。因此，建议用户需要根据每个任务的执行时长合理的设置任务的时间间隔（即 submit 接口中的 interval 参数），来避免由于任务执行时间太长而导致下个轮询周期无法正常执行。

注：如果同一时间内并行的 job 数很多，过小的参数值会导致 job 等待。而过大的参数值则消耗更多的系统资源，建议设置此参数为 100，用户可以根据系统资源情况合理调整。

默认值：10

enable_prevent_job_task_startup

参数说明：控制是否启动 job 线程。

该参数属于 SIGHUP 类型参数，请参考：配置运行参数->重设参数章节中表 1 对应设置方法进行设置。

取值范围：布尔型

- on 表示不能启动 job 线程。
- off 表示可以启动 job 线程。

默认值：off

12.33.线程池

enable_thread_pool

参数说明：控制是否使用线程池功能。该参数属于 POSTMASTER 类型参数，请参考：配置运行参数->重设参数章节中表 1 对应设置方法进行设置。

取值范围：布尔型

- on 表示开启线程池功能。
- off 表示不开启线程池功能。

默认值：off

thread_pool_attr

参数说明：用于控制线程池功能的详细属性，该参数仅在 enable_thread_pool 打开后生效。该参数属于 POSTMASTER 类型参数，请参考：配置运行参数->重设参数章节中表 1 对应设置方法进行设置。

取值范围：字符串，长度大于 0

该参数分为 3 个部分，'thread_num、 group_num、 cpubind_info'，这 3 个部分的具体含义如下：

- thread_num：线程池中的线程总数，取值范围是 0~4096。其中 0 的含义是数据库根据系统 CPU core 的数量来自动配置线程池的线程数，如果参数值大于 0，线程池中的线程数等于 thread_num。线程池大小推荐根据硬件配置设置，计算公式为 thread_num=CPU 核数*3~5，thread_num 最大值 4096。
- group_num：线程池中的线程分组个数，取值范围是 0~64。其中 0 的含义是数据库根据系统 NUMA 组的个数来自动配置线程池的线程分组个数，如果参数值大于 0，线程池中的线程组个数等于 group_num。
- cpubind_info：线程池是否绑核的配置参数。可选择的配置方式有：
  - '(nobind)' ，线程不做绑核；
  - '(allbind)'，利用当前系统所有能查询到的 CPU core 做线程绑核；
  - '(nodebind: 1, 2)'，利用 NUMA 组 1，2 中的 CPU core 进行绑核；
  - '(cpubind: 0-30)'，利用 0-30 号 CPU core 进行绑核。
  - '(numabind: 0-30)'，在 NUMA 组内利用 0-30 号 CPU core 进行绑核。利用 0-30 号 CPU core 进行绑核。该参数不区分大小写。

默认值：'16, 2, (nobind)'

resilience_threadpool_reject_cond

参数说明： 用于控制线程池过载逃生的堆积会话数占比。该参数仅在 GUC 参数 use_workload_manager 和 enable_thread_pool 打开时生效。

该参数属于 SIGHUP 类型参数，请参考重设参数表 1 中对应设置方法进行设置。

取值范围： 字符串，长度大于 0

该参数分为 recover_threadpool_percent、overload_threadpool_percent 两部分，这两个部分的具体含义如下：

- recover_threadpool_percent：线程池恢复正常状态的接入会话占线程池初始设置线程数的百分比，当已经接入的会话数小于线程池初始设置数乘以该值对应的百分比后，停止过载逃生并放开新连接接入，取值为0~INT_MAX，设置为多少表示百分之多少。
- overload_threadpool_percent：线程池过载时的接入会话占线程池初始设置线程数的百分比，当已经接入的会话数大于线程池初始设置数乘以该值对应的百分比后，表示当前线程池已经过载，触发过载逃生 kill 会话并禁止新连接接入，取值为 0~INT_MAX，设置为多少表示百分之多少。

默认值： '0,0'，表示关闭线程池逃生功能。

示例：

```
resilience_threadpool_reject_cond = '100,200'
```

上述设置表示已经堆积的会话数超过线程池初始设置的线程数的 200%后禁止新连接接入并 kill 堆积的会话，kill 会话过程中会话数恢复到线程池初始设置的线程数的 100%以下时停止 kill 会话并允许新连接接入。

【说明】

- 已经堆积的会话数可以通过查询 pg_stat_activity 视图有多少条数据获得，需要过滤少量后台线程。线程池设置的初始线程池线程数目可以通过查询 thread_pool_attr 参数获得。
- 该参数如果设置的百分比过小，则会频繁触发线程池过载逃生流程，会使正在执行的会话被强制退出，新连接短时间接入失败，因此需要根据实际线程池使用情况慎重设置。

thread_pool_stream_attr

参数说明： 用于控制 stream 线程池功能的详细属性，stream 线程只在 DN 生效，该参数仅在 enable_thread_pool 打开后生效，仅 sysadmin 用户可以访问。

该参数属于 POSTMASTER 类型参数，请参考重设参数表 1 中对应设置方法进行设置。

取值范围：字符串，长度大于 0

该参数分为 4 个部分，`stream_thread_num, stream_proc_ratio ,group_num ,cpubind_info`，这 4 个部分的具体含义如下：

- stream_thread_num：stream 线程池中的线程总数，取值范围是 0~4096。其中 0 的含义是数据库根据系统 CPU core 的数量来自动配置线程池的线程数，如果参数值大于 0，线程池中的线程数等于 stream_thread_num。线程池大小推荐根据硬件配置设置，计算公式如下：`stream_thread_num = CPU 核数*3~5`，stream_thread_num 最大值为 4096。
- stream_proc_ratio：预留给 stream 线程的 proc 数量比例，浮点类型，默认为 0.2，预留 proc 计算方式为：`stream_proc_ratio * stream_thread_num`。
- group_num：线程池中的线程分组个数，取值范围是 0~64。其中 0 的含义是数据库根据系统 NUMA 组的个数来自动配置线程池的线程分组个数，如果参数值大于 0，线程池中的线程组个数等于 group_num。thread_pool_stream_attr 的 group_num 需与 thread_pool_attr 的 group_num 配置和使用保持一致，若设置为不同值，以 thread_pool_attr 的 group_num 为准。
- cpubind_info：线程池是否绑核的配置参数。可选择的配置方式有：

  - 1. '(nobind)'，线程不做绑核。

  - 2. '(allbind)'，利用当前系统所有能查询到的 CPU core 做线程绑核。

  - 3. '(nodebind: 1, 2)'，利用 NUMA 组 1,2 中的 CPU core 进行绑核。

  - 4. '(cpubind: 0-30)'，利用 0-30 号 CPU core 进行绑核。

  - 5. '(numabind: 0-30)'，在 NUMA 组内利用 0-30 号 CPU core 进行绑核。

该参数不区分大小写。thread_pool_stream_attr 的 cpubind_info 需与 thread_pool_attr 的 cpubind_info 配置和使用保持一致，若设置为不同值，以 thread_pool_attr 的 cpubind_info 为准。

默认值： 16, 0.2, 2,(nobind)'

12.34.备份恢复

operation_mode

参数说明：标示系统进入备份恢复模式。

该参数属于 SIGHUP 类型参数，请参考：配置运行参数->重设参数章节中表 1 对应设置方法进行设置。

取值范围：布尔型

- on 表示在备份恢复过程中。
- off 表示不在备份恢复过程中。

默认值：off

enable_cbm_tracking

参数说明：当使用 roach 执行数据库实例的全量和增量备份时需要开启此参数，如果关闭会导致备份失败。

该参数属于 SIGHUP 类型参数，请参考：配置运行参数->重设参数章节中表 1 对应设置方法进行设置。

取值范围：布尔型

- on 表示追踪功能开启。
- off 表示追踪功能关闭。

默认值：off

hadr_max_size_for_xlog_receiver

参数说明：该参数为异地容灾参数，表示灾备数据库实例中实例获取 obs 端日志和本地回放日志的最大允许差距，若差距大于此值时停止获取 obs 端日志。

该参数属于 SIGHUP 类型参数，请参考：配置运行参数->重设参数章节中表 1 中方式对应设置方法进行设置。

修改建议：该参数的取值应和本地磁盘大小相关，建议设置为磁盘大小的 50%。

取值范围：整型，0~2147483647

默认值：256GB

12.35.Undo

undo_space_limit_size

参数说明：用于控制 undo 强制回收阈值，达到阈值的 80%启动强制回收，用户需要根据自己的业务情况，设置该值，可以通过先设置一个较大值，然后观察实际业务运行占用 undo 空间，再将该值调整为合理值。

该参数属于 SIGHUP 类型参数，请参考：配置运行参数->重设参数章节中表 1 对应设置方法进行设置。

取值范围：整型，800MB~16TB

默认值：256GB

undo_limit_size_per_transaction

参数说明：用于控制单事务 undo 分配空间阈值，达到阈值时事务报错回滚。

该参数属于 SIGHUP 类型参数，请参考：配置运行参数->重设参数章节中表 1 对应设置方法进行设置。

取值范围：整型，2MB~16TB

默认值：32GB

12.36.DCF 参数设置

enable_dcf

参数说明：是否开启 DCF 模式，该参数不允许修改。（单机部署不支持该参数配置）

该参数属于 POSTMASTER 类型参数，请参考：配置运行参数->重设参数章节中表 1 对应设置方法进行设置。

取值范围：布尔型，on/off。on 表示当前安装部署方式为 DCF 模式，off 表示当前安装部署方式为非 DCF 模式。

默认值：off

dcf_majority_groups

参数说明： DCF 策略化多数派功能设置。对于需要配置此参数的 group，该 group 内至少有一台备机收到日志。即该 group 内存在一台同步备机。若对 DCF 实例内做了增删节点或者对实例内节点 group 值进行了调整修改，需同步修改此配置。

该参数属于 SIGHUP 类型参数，请参考重设参数表 1 中对应设置方法进行设置。

取值范围： 字符串

- 关闭：”“，空字符串表示策略化多数派功能关闭。
- 开启： 配置有效的 group 值，使用逗号分隔，group 值需在 dcf_config 中存在。例如将 group 值分别为 1 和 2，加入 DCF 的策略化多数派配置时，可以设置为"1,2"；若配置了 dcf_config 中不存在的 group 值或者其他字符，DCF 将认为该配置的 group 无效。

默认值： **空字符串**

【注意】

若配置了参数后某一 group 内所有节点均故障，在对其中某个节点做涉及节点 build 相关操作（节点修复、不换 ip 的节点替换）时，需要将该 group 从此参数列表中移除，待节点恢复正常后可将该 group 再次配置到此参数。

dcf_ssl

参数说明：是否开启 SSL，重启生效。

该参数属于 POSTMASTER 类型参数，请参考：配置运行参数->重设参数章节中表 1 对应设置方法进行设置。

取值范围：布尔型，on/off。on 表示使用 SSL，off 表示不使用 SSL。

默认值：on

dcf_config

参数说明：用户安装时自定义配置信息，该参数不允许修改。

该参数属于 POSTMASTER 类型参数，请参考：配置运行参数->重设参数章节中表 1 对应设置方法进行设置。

默认值：字符串，安装时用户自定义配置

dcf_data_path

参数说明：DCF 数据路径，该参数不允许修改。

该参数属于 POSTMASTER 类型参数，请参考：配置运行参数->重设参数章节中表 1 对应设置方法进行设置。

默认值：字符串，DN 数据目录下的 dcf_data 目录

dcf_log_path

参数说明：DCF 日志路径，该参数不允许修改。

该参数属于 POSTMASTER 类型参数，请参考：配置运行参数->重设参数章节中表 1 对应设置方法进行设置。

默认值：字符串，DN 数据目录下的 dcf_log 目录

dcf_node_id

参数说明：DCF 所在 DN 节点 ID，用户安装时自定义，该参数不允许修改。

该参数属于 POSTMASTER 类型参数，请参考：配置运行参数->重设参数章节中表 1 对应设置方法进行设置。

默认值：整型，安装时用户自定义配置

dcf_max_workers

参数说明：DCF 回调函数线程个数。如果节点数量超过 7 个，需要增加这个参数的数值（比如增加到 40），否则可能会出现主节点一直处于 promoting 状态，主备节点日志不推进的状态。

该参数属于 POSTMASTER 类型参数，请参考：配置运行参数->重设参数章节中表 1 对应设置方法进行设置。

取值范围：整型，10~262143

默认值：10

dcf_truncate_threshold

参数说明：DN 对 DCF 日志进行 truncate 的门限阈值。

该参数属于 SIGHUP 类型参数，请参考：配置运行参数->重设参数章节中表 1 对应设置方法进行设置。

取值范围：整型，1~2147483647

默认值：100000

dcf_election_timeout

参数说明：DCF leader 和 follower 选举超时时间。选举超时时间数值依赖于当前 DN 之间的网络状况，在超时时间较小且网络极差的情形下，会有超时选举发生，待网络恢复选举恢复正常。建议根据当前网络状态合理设置超时时间。对 DCF 节点时钟的约束：DCF 节点间最大时钟差异小于选举超时时间的一半。

该参数属于 SIGHUP 类型参数，请参考：配置运行参数->重设参数章节中表 1 对应设置方法进行设置。

取值范围：整型，单位 s，1~600

默认值：3

dcf_enable_auto_election_priority

参数说明：DCF 优先级选主是否允许内部自动调整优先级值。0 表示不允许，1 表示允许内部自动调整。

该参数属于 SIGHUP 类型参数，请参考：配置运行参数->重设参数章节中表 1 对应设置方法进行设置。

取值范围：整型，0~1

默认值：1

dcf_election_switch_threshold

参数说明：DCF 防频繁切主门限。推荐根据用户业务可接受的最大故障时间配置。

该参数属于 SIGHUP 类型参数，请参考：配置运行参数->重设参数章节中表 1 对应设置方法进行设置。

取值范围：整型，单位 s，0~2147483647

默认值：0

dcf_run_mode

参数说明：DCF 选举模式，0 表示自动选举模式，2 表示去使能选举模式。目前去使能选举模式只限定少数派恢复场景使用，修改会导致数据库实例不可用。

该参数属于 SIGHUP 类型参数，请参考：配置运行参数->重设参数章节中表 1 对应设置方法进行设置。

取值范围：枚举类型，0、2

默认值：0

dcf_log_level

参数说明：DCF 日志级别。

该参数属于 SIGHUP 类型参数，请参考：配置运行参数->重设参数章节中表 1 对应设置方法进行设置。

取值范围：字符串

- 关闭日志：“NONE”，NONE 表示关闭日志打印，不能与以下日志级别混合使用。
- 开启日志：
  “RUN_ERR|RUN_WAR|RUN_INF|DEBUG_ERR|DEBUG_WAR|DEBUG_INF|TRACE|PROFILE|OPER”

日志级别可以从上述字符串中选取字符串并使用竖线组合使用，不能配置空串。

默认值：“RUN_ERR|RUN_WAR|DEBUG_ERR|OPER|RUN_INF|PROFILE”

dcf_log_backup_file_count

参数说明：DCF 运行日志备份保留个数。

该参数属于 SIGHUP 类型参数，请参考：配置运行参数->重设参数章节中表 1 对应设置方法进行设置。

取值范围：整型，1~100

默认值：10

dcf_max_log_file_size

参数说明：DCF 运行日志单个文件最大大小。

该参数属于 SIGHUP 类型参数，请参考：配置运行参数->重设参数章节中表 1 对应设置方法进行设置。

取值范围：整型，单位 MB，1~1000

默认值：10

dcf_socket_timeout

参数说明：DCF 通信模块连接 socket 超时时间，参数重启生效。对于网络环境比较差的环境，若配置很小的超时时间，可能会导致建链不成功，此时需要适当增大此值。

该参数属于 POSTMASTER 类型参数，请参考：配置运行参数->重设参数章节中表 1 对应设置方法进行设置。

取值范围：整型，单位 ms，10~600000

默认值：5000

dcf_connect_timeout

参数说明：DCF 通信模块建立连接超时时间，参数重启生效。对于网络环境比较差的环境，若配置很小的超时时间，可能会导致建链不成功，此时需要适当增大此值。

该参数属于 POSTMASTER 类型参数，请参考：配置运行参数->重设参数章节中表 1 对应设置方法进行设置。

取值范围：整型，单位 ms，10~600000

默认值：60000

dcf_mec_fragment_size

参数说明：DCF 通信模块 fragment 大小，参数重启生效。

该参数属于 POSTMASTER 类型参数，请参考：配置运行参数->重设参数章节中表 1 对应设置方法进行设置。

取值范围：整型，单位 KB，32~10240

默认值：64

dcf_stg_pool_max_size

参数说明：DCF 存储模内存池最大值，参数重启生效。

该参数属于 POSTMASTER 类型参数，请参考：配置运行参数->重设参数章节中表 1 对应设置方法进行设置。

取值范围：整型，单位 MB，32~2147483647

默认值：2048

dcf_stg_pool_init_size

参数说明：DCF 存储模块内存池最小值，参数重启生效。

该参数属于 POSTMASTER 类型参数，请参考：配置运行参数->重设参数章节中表 1 对应设置方法进行设置。

取值范围：整型，单位 MB，32~2147483647

默认值：32

dcf_mec_pool_max_size

参数说明：DCF 通信模块内存池最大值，参数重启生效。

该参数属于 POSTMASTER 类型参数，请参考：配置运行参数->重设参数章节中表 1 对应设置方法进行设置。

取值范围：整型，单位 MB，32~2147483647

默认值：200

dcf_flow_control_disk_rawait_threshold

参数说明：DCF 流控功能的磁盘等待阈值。

该参数属于 SIGHUP 类型参数，请参考：配置运行参数->重设参数章节中表 1 对应设置方法进行设置。

取值范围：整型，单位 us，0~2147483647

默认值：100000

dcf_flow_control_net_queue_message_num_threshold

参数说明：DCF 流控功能的网络队列消息数阈值。

该参数属于 SIGHUP 类型参数，请参考：配置运行参数->重设参数章节中表 1 对应设置方法进行设置。

取值范围：整型，0~2147483647

默认值：1024

dcf_flow_control_cpu_threshold

参数说明：DCF CPU 流控阈值。

该参数属于 SIGHUP 类型参数，请参考：配置运行参数->重设参数章节中表 1 对应设置方法进行设置。

取值范围：整型，单位百分比，0~2147483647

默认值：100

dcf_mec_batch_size

参数说明：DCF 通信批量消息数，数值为 0 时，DCF 会根据网络以及写入数据量自适应调整，参数重启生效。

该参数属于 POSTMASTER 类型参数，请参考：配置运行参数->重设参数章节中表 1 对应设置方法进行设置。

取值范围：整型，0~1024

默认值：0

dcf_mem_pool_max_size

参数说明：DCF 内存最大值，参数重启生效。

该参数属于 POSTMASTER 类型参数，请参考：配置运行参数->重设参数章节中表 1 对应设置方法进行设置。

取值范围：整型，单位 MB，32~2147483647

默认值：2048

dcf_mem_pool_init_size

参数说明：DCF 内存初始化大小，参数重启生效。

该参数属于 POSTMASTER 类型参数，请参考：配置运行参数->重设参数章节中表 1 对应设置方法进行设置。

取值范围：整型，单位 MB，32~2147483647

默认值：32

dcf_compress_algorithm

参数说明：DCF 运行日志传输压缩算法，参数重启生效。

该参数属于 POSTMASTER 类型参数，请参考：配置运行参数->重设参数章节中表 1 对应设置方法进行设置。

取值范围：整型

- 0 表示不压缩
- 1 表示 ZSTD 压缩算法
- 2 表示 LZ4 压缩算法

默认值：0

dcf_compress_level

参数说明：DCF 日志传输压缩级别，参数重启生效，此参数生效前提必须配置有效的压缩算法，即设置合法的 dcf_compress_algorithm 参数。

该参数属于 POSTMASTER 类型参数，请参考：配置运行参数->重设参数章节中表 1 对应设置方法进行设置。

取值范围：整型，1~22

若不开启压缩，配置的压缩级别将不生效。

默认值：1

dcf_mec_channel_num

参数说明：DCF 通信通道数量，参数重启生效。

该参数属于 POSTMASTER 类型参数，请参考：配置运行参数->重设参数章节中表 1 对应设置方法进行设置。

取值范围：整型，1~64

默认值：1

dcf_rep_append_thread_num

参数说明：DCF 日志复制线程数量，参数重启生效。

该参数属于 POSTMASTER 类型参数，请参考：配置运行参数->重设参数章节中表 1 对应设置方法进行设置。

取值范围：整型，1~1000

默认值：2

dcf_mec_agent_thread_num

参数说明：DCF 通信工作线程数量，参数重启生效。
dcf_mec_agent_thread_num 值建议不少于 2 *节点数* dcf_mec_channel_num。

该参数属于 POSTMASTER 类型参数，请参考：配置运行参数->重设参数章节中表 1 对应设置方法进行设置。

取值范围：整型，1~1000

默认值：10

dcf_mec_reactor_thread_num

参数说明：DCF 使用 reactor 线程数量，参数重启生效。dcf_mec_reactor_thread_num 与 dcf_mec_agent_thread_num 比例建议 1：40。

该参数属于 POSTMASTER 类型参数，请参考：配置运行参数->重设参数章节中表 1 对应设置方法进行设置。

取值范围：整型，1~100

默认值：1

dcf_log_file_permission

参数说明：DCF 运行日志文件属性，参数重启生效，参数安装阶段配置，后续不支持修改。若用户需要支持同组的其他用户访问日志，首先需要所有的父目录都支持同组的其他用户也能访问。即若参数 dcf_log_path_permission 配置为 750，dcf_log_file_permission 只能为 600 或者 640。若参数 dcf_log_path_permission 配置为 700，dcf_log_file_permission 只能为 600。

该参数属于 POSTMASTER 类型参数，请参考：配置运行参数->重设参数章节中表 1 对应设置方法进行设置。

取值范围：枚举型，600、640

默认值：600

dcf_log_path_permission

参数说明：DCF 运行日志目录属性，参数重启生效，参数安装阶段配置，后续不支持修改。若用户需要支持同组的其他用户访问日志路径，需选择参数 750，否则选择 700。

该参数属于 POSTMASTER 类型参数，请参考：配置运行参数->重设参数章节中表 1 对应设置方法进行设置。

取值范围：枚举型，700、750

默认值：700

12.37.闪回相关参数

本章节介绍闪回功能相关参数。

enable_recyclebin

参数说明：用来控制回收站的实时打开和关闭。

该参数属于 SIGHUP 类型参数，请参考：配置运行参数->重设参数章节中表 1 对应设置方法进行设置。

取值范围：布尔型

默认值：off

【注意】
recyclebin 支持 Astore/Ustore 表。

max_flashback_time

参数说明：用来控制最大闪回时长。超出时长的旧版本数据将无法使用闪回功能查看。

该参数属于 SIGHUP 类型参数，请参考重设参数表 1 中对应设置方法进行设置。

取值范围：整型，0～576460752303423487，值为 0 表示不开启闪回。单位为 s。

默认值： 0

【须知】
集群模式下，设置 max_flashback_time 参数的同时应打开最大可用模式，即设置 most_available_sync=on。

recyclebin_retention_time

参数说明：设置回收站对象保留时间，超过该时间的回收站对象将被自动清理。

该参数属于 SIGHUP 类型参数，请参考：配置运行参数->重设参数章节中表 1 对应设置方法进行设置。

取值范围：整型，单位为 s，最小值为 1，最大值为 2147483647。

默认值：15min（即 900s）

version_retention_age

参数说明：设置旧版本保留的事务数，超过该事务数的旧版本将被回收清理。

该参数属于 SIGHUP 类型参数，请参考：配置运行参数->重设参数章节中表 1 对应设置方法进行设置。

取值范围：整型，0～576460752303423487，值为 0 表示不延迟。

默认值：0

【注意】
该参数已弃用。

vacuum_defer_cleanup_age

参数说明：指定 VACUUM 使用的事务数，VACUUM 会延迟清除无效的行存表记录，延迟的事务个数通过 vacuum_defer_cleanup_age 进行设置。即 VACUUM 和 VACUUM FULL 操作不会立即清理刚刚被删除元组。也可以通过设置该参数，配置闪回功能旧版本保留期限。

该参数属于 SIGHUP 类型参数，请参考：配置运行参数->重设参数章节中表 1 对应设置方法进行设置。

取值范围：整型，0～1000000，值为 0 表示不延迟。

默认值：0

【注意】
在进行 Ustore 闪回时，无需关注该参数。其服务于之前版本的 astore 闪回功能，同时具有其他用途。本版本闪回功能已不使用。

undo_retention_time

参数说明：设置 undo 旧版本保留时间。

该参数属于 SIGHUP 类型参数，请参考：配置运行参数->重设参数章节中表 1 对应设置方法进行设置。

取值范围：整型，单位为 s，最小值为 0，最大值为 259200。

默认值：0

12.38.回滚相关参数

max_undo_workers

参数说明：异步回滚调用的 undoworker 线程数量，参数重启生效。

该参数属于 SIGHUP 类型参数，请参考：配置运行参数->重设参数章节中表 1 对应设置方法进行设置。

取值范围：整型，1~100

默认值：5

12.39.预留参数

【说明】

下列参数为预留参数，该版本不生效。

- acce_min_datasize_per_thread
- cstore_insert_mode
- enable_constraint_optimization
- enable_hadoop_env
- enable_hdfs_predicate_pushdown
- enable_orc_cache
- schedule_splits_threshold
- backend_version
- undo_zone_count
- version_retention_age

12.40.AI 特性

enable_hypo_index

参数说明：该参数控制数据库的优化器进行 EXPLAIN 时是否考虑创建虚拟索引。通过对特定的查询语句执行 explain，用户可根据优化器给出的执行计划评估该索引是否能够提升该查询语句的执行效率。

该参数属于 USERSET 类型参数，请参考：配置运行参数->重设参数章节中表 1 对应设置方法进行设置。

取值范围：布尔型

- on 表示在进行 EXPLAIN 时创建虚拟索引。
- off 表示在进行 EXPLAIN 时不创建虚拟索引。

默认值：off

db4ai_snapshot_mode

参数说明：snapshot 有 2 种模式：MSS（物化模式，存储数据实体）和 CSS（计算模式，存储增量信息）。

该参数属于 USERSET 类型参数，请参考：配置运行参数->重设参数章节中表 1 对应设置方法进行设置。

取值范围：字符串，MSS/CSS

- MSS 表示物化模式，db4ai 在创建快照的时候存储数据实体。
- CSS 表示计算模式，db4ai 在创建快照的时候存储增量信息。

默认值：MSS

db4ai_snapshot_version_delimiter

参数说明：该参数为数据表快照版本分隔符。

该参数属于 USERSET 类型参数，请参考：配置运行参数->重设参数章节中表 1 对应设置方法进行设置。

取值范围：字符串，长度大于 0

默认值：@

db4ai_snapshot_version_separator

参数说明：该参数用于指定数据表快照子版本分隔符。

该参数属于 USERSET 类型参数，请参考：配置运行参数->重设参数章节中表 1 对应设置方法进行设置。

取值范围：字符串，长度等于 1

默认值：.

unix_socket_directory

参数说明：用于指定 unix_socket 通信方式中，文件存放的路径。此参数只能在配置文件 postgresql.conf 中指定。再启动 fenced 模式前需要设定该 GUC 参数。

该参数属于 POSTMASTER 类型参数，请参考：配置运行参数->重设参数章节中表 1 对应设置方法进行设置。

取值范围：字符串，长度等于 1

默认值：‘‘

12.41.Global SysCache 参数

enable_global_syscache

参数说明：控制是否使用全局系统缓存功能。

该参数属于 POSTMASTER 类型参数，请参考：配置运行参数->重设参数章节中表 1 对应设置方法进行设置。

取值范围：布尔型

- on 表示开启全局系统缓存功能。
- off 表示不开启全局系统缓存功能。

默认值：on

推荐结合线程池参数使用。打开该参数后，如果需要访问备机，建议设置备机 wal_level 级别为 hot_standby 以上。

global_syscache_threshold

参数说明：全局系统缓存内存最大占用大小。如果设置的值过小，会导致内存频繁淘汰，内存存在大量碎片无法回收，导致内存控制失效。需要打开 enable_global_syscache 参数，该参数才生效。

该参数属于 PGC_SIGHUP 类型参数，请参考：配置运行参数->重设参数章节中表 1 对应设置方法进行设置。

需要打开 enable_global_syscache 参数。

取值范围：整型，16384~1073741824，单位为 kB。

默认值：163840

推荐计算公式：热点 DB 个数和线程个数的最小值乘以每个 DB 分配的内存大小，即 global_syscache_threshold = min(count(hot dbs), count(threads)) * memofdb。

【说明】

- hot dbs：热点 DB 数，即访问较为频繁的数据库。
- threads：线程数，在线程池模式下取线程池线程个数和后台线程个数之和，非线程池模式不需要计算这个值，直接使用热点 DB 数。
- memofdb：平均每个 db 应该分配的内存，每个 DB 的底噪内存是 2M，平均每增加一个表或者索引，增加 11k 内存

12.42.date 类型控制参数

功能描述

date 类型控制参数 pw_date_type 用来控制 date 类型的底层存储类型。

语法格式

- ORACLE 兼容模式下，支持通过 alter 命令设置 date 类型控制参数 pw_date_type，默认值为 1。

```
alter system set pw_date_type = [1/2];
```

- ORACLE 兼容模式下，通过修改 postgresql.conf 设置 date 类型控制参数

```
vi $PGDATA/postgresql.conf
 //设置参数:
pw_date_type = [1/2];
```

参数说明

- pw_date_type = 1 时 date 底层存储为 oradate。
- pw_date_type = 2 时 date 底层存储为 timestamp。

注意事项

- PG 兼容模式下，不支持 date 类型控制参数。
- MYSQL 兼容模式下，不支持 date 类型控制参数。
- 不指定兼容模式时，数据库默认为 oracel 兼容模式，支持 date 类型控制参数。

示例

示例 1： oracle 兼容下设置 date 类型控制参数。

1、创建数据库，兼容模式为 oracle。

```
CREATE DATABASE date_test_oracle;
```

2、配置 date 类型控制参数 pw_date_type。

```
alter system set pw_date_type = 1;
```

3、重启数据库。

```
pw_ctl restart
```

4、查看 date 兼容效果。

```
select '2022-08-19'::date;
```

查询结果显示为：

```
    oradate
-----------
 2022-08-19 00:00:00
(1 row)
```

示例 2：ORACLE 兼容模式下，通过修改 postgresql.conf 设置 date 类型控制参数。

1、创建数据库。

```
CREATE DATABASE date_test_postgresqlconf;
```

2、配置 date 类型控制参数。

```
vi $PGDATA/postgresql.conf
//设置参数:
pw_date_type = 2;
```

3、重启数据库。

```
pw_ctl restart
```

4、查看 date 兼容效果。

```
\c date_test_postgresqlconf
select '2022-08-19'::date;
```

查询结果显示为：

```
    timestamp
-----------
 2022-08-19 00:00:00
(1 row)
```

12.43.备机支持写语句参数

eenable_remote_excute

参数说明：是否开启允许备机执行写语句，启动后不允许修改。

该参数属于 POSTMASTER 类型参数，请参考：配置运行参数->重设参数章节中表 1 对应设置方法进行设置。

取值范围：布尔型，true、false。true 表示当前安装部署开启备机执行写语句模式，off 表示不开启。

默认值：off

【说明】

- enable_remote_excute 参数在单机模式下默认 false。
- 若开启备机允许执行写语句需要将此参数设置为 true。
- 开关打开后，允许备机执行写语句和 DDL，支持简单查询和扩展查询；此状态下读语句仍然在备机执行，写语句会转发到主机执行。
- 开关打开后，备机启动事务后会将所有 SQL 语句全部无条件转发给主机，包括读语句。
- 在传统主备架构下，开关打开后，备机启动事务后会将所有 SQL 语句全部无条件转发给主机,包括读语句。
- 在资源池化架构下，开关打开后，备机启动事务后会将事务中的涉及修改的写 SQL 语句转发给主机, 事务中的读语句仍然在备机本地执行。
- 在资源池化架构下，开关打开后，备机不支持事务内包含有 DDL 语句和 LOCK 语句，遇到这种情况会报错。
- 在资源池化架构下，开关打开后，如果事务内包含子事务，那么事务内的读也会转发到主。
- 在资源池化架构下，开关打开后，如果执行写业务的备机的并发连接数超过 50，且业务端存在报错`“standbywrite could not connect to the remote server”`时，需要调整连接主机的最大等待超时时间 wal_receiver_connect_timeout，建议配置为 120s，配置方式参考备机参数中该参数的说明。
- 开关打开后，用于在备机上执行写业务操作的用户必须具有 SYSADMIN 权限。

12.44.多级缓存管理参数

enable_nvm

参数说明： 是否开启多级缓存管理功能，启动后不允许修改。

该参数属于 POSTMASTER 类型参数，请参考重设参数表 1 中对应设置方法进行设置。

取值范围： 布尔型

- on：表示当前安装部署开启多级缓存管理功能。
- off：表示当前安装部署不开启多级缓存管理功能。

默认值： off

【说明】

- 开关打开后，将原有缓存池划分为 Dram Buffer Pool 和 Nvm Buffer Pool，通过访问频率控制各缓存层次间的页面迁移，实现热数据在内存中，温数据在 NVM，冷数据在磁盘中。
- 只有当 enable_nvm 为 on 时，参数 nvm_buffers、nvm_file_path、bypass_nvm、bypass_dram 才会生效。

nvm_buffers

参数说明： Nvm Buffer Pool 的大小。

该参数属于 POSTMASTER 类型参数，请参考重设参数表 1 中对应设置方法进行设置。

取值范围： 整型，0~2147483647

默认值： 0

nvm_file_path

参数说明： nvm 文件路径。

该参数属于 POSTMASTER 类型参数，请参考重设参数表 1 中对应设置方法进行设置。

取值范围： 字符串，nvm 文件路径

【说明】

nvm 介质（例如 SCM）以 App Direct 模式通过文件系统接口暴露给应用，PanWeiDB 通过对 nvm 文件进行 mmap 的方式实现对 nvm 介质的字节寻址，达到作为 PanWeiDB 缓存的效果。

bypass_nvm

参数说明： 当页面没有在缓存中命中，绕过 nvm 缓存池直接加载至 dram 缓存池的概率。

该参数属于 POSTMASTER 类型参数，请参考重设参数表 1 中对应设置方法进行设置。

取值范围： 浮点型，0.0~1.0

默认值： 0.5

bypass_dram

参数说明： 当页面在 nvm 缓存池命中，被迁移至 dram 缓存池的概率。

该参数属于 POSTMASTER 类型参数，请参考重设参数表 1 中对应设置方法进行设置。

取值范围： 浮点型，0.0~1.0

默认值： 0.01

【说明】

以默认值 0.01 为例，当页面在 nvm 缓存池命中后，有 1%的概率会迁移至 dram 缓存池。

12.45.大页内存

操作系统页表所需空间会随着运行环境总内存的增加而增加。在数百 GB 级别的内存环境中，使用 4K 的内存页将导致数据库进程页表空间变大，同时 TLB miss 的概率增加，这将拖慢数据库的查询速度，并且浪费主存空间。为避免大内存运行环境下系统内存页过小导致性能下降，PanWeiDB 引入大页内存功能。当配置文件的 enable_huge_pages 设置为 on 时 PanWeiDB 数据库在启动时将从操作系统预分配的大页内存中创建共享内存。

用户可以根据运行环境决定是否开启大页内存。根据实践，数百 GB 级别内存的环境下，若系统内存页大小为 4K，开启大页内存可以提高数据库性能。一般而言，数百 GB 级别的环境下，可以考虑使用大小为 2MB 的大页；若环境内存达到 TB 级别，可以选择更大的大页。

enable_huge_pages

参数说明：设置数据库共享内存是否使用大页内存分配。

该参数属于 POSTMASTER 类型参数，请参考重设参数表 1 中对应设置方法进行设置。在主备场景中，该参数不会被同步到其他节点。

取值范围：布尔型

- on：表示使用大页内存进行共享内存分配。
- off：表示使用普通内存页进行共享内存分配。

默认值：off

【须知】
启用大页内存需要提前在操作系统完成大页配置。具体配置方法详见下方示例。

huge_page_size

参数说明：大页内存使用的页面大小。该选项仅在 enable_huge_pages 设置为 on 时生效。

该参数属于 POSTMASTER 类型参数，请参考重设参数表 1 中对应设置方法进行设置。在主备场景中，该参数不会被同步到其他节点。

取值范围：整型，0 ~ 1073741823，单位为 8KB。也可按照 xxkB、xxMB、xxGB 的方式设置。

默认值： 0，代表使用操作系统默认的大页大小。操作系统默认值可以通过 /proc/meminfo 中的 Hugepagesize 项查看。

【须知】
- 此参数的值必须是当前环境操作系统下可选的大页大小。具体的大页可选值可以通过/sys/kernel/mm/hugepages/目录查看。若 enable_huge_pages 设置为 on 时，填入的大页大小在当前环境下不支持，数据库将无法启动。
- 不同操作系统提供不同大小的大页，具体选择需参考运行环境总内存大小，

以及当前环境下内存负载情况。一般而言，对于数百 GB 级别的内存环境，可以选择 2MB 的大页。对于 TB 级别的内存环境，则可选择更大的大页。

示例

1、查看系统默认大页大小。

```
cat /proc/meminfo | grep Hugepagesize
```

2、查看系统支持大页大小。

```
ls /sys/kernel/mm/hugepages
```

3、设置大页数量。

设置大页数量前，用户需要知道当前配置下数据库的共享内存共需多少个页面。以使用 2MB 大页为例。当配置文件中 enable_huge_pages 设置为 on 时，数据库启动将打印如下日志：

```
LOG:  Allocate shared memory as huge pages. Huge page size: 2048 KB, requi
red pages count: 1670
```

此日志提示用户在当前配置下，数据库的共享内存将需要 1670 个 2MB 大页。此时，更改/sys/kernel/mm/hugepages/hugepages-2048kB/nr_hugepages 文件的值。

【注意】
设置大页时操作系统需寻找指定数量的空载内存。以本示例为例，操作系统将寻找 1670 个大小为 2M 的空载内存进行大页内存的分配。若无法找到指定数量的空载内存，操作系统将设置当前可分配的最大数量。因此，设置大页内存后请检查文件，确保大页数量充足以保证数据库正常启动。

```
echo 1670 > /sys/kernel/mm/hugepages/hugepages-2048kB/nr_hugepages
cat /sys/kernel/mm/hugepages/hugepages-2048kB/nr_hugepages
```

启动数据库后，可以通过/proc/meminfo 查看系统当前大页的使用情况。

4、给与数据库进程用户使用大页内存权限。

查看数据库进程用户所属用户组 id，给与用户使用大页内存权限。

```
id PanWeiDB
sysctl -w vm.hugetlb_shm_group=xxxx
```

5、设置配置文件 postgresql.conf，将 enable_huge_pages 设置为 on，开启大页功能。

6、重启数据库。

12.46.数据导入导出

safe_data_path

参数说明：设置初始用户以外的路径前缀限制，目前包括 copy 和高级包路径限制。

该参数属于 SIGHUP 类型参数，请参考重设参数表 1 中对应设置方法进行设置。

取值范围：字符串（小于 4096 个字符）

默认值：NULL

【注意】

- 如果 safe_data_path 目录下存在软链接文件，则会按软链接实际指向的文件路径进行处理，实际路径如果不在 safe_data_path 下会报错处理。
- 如果 safe_data_path 目录下存在硬链接文件，则可以正常使用。为安全起见，请谨慎使用硬链接文件，切勿在 safe_data_path 目录下创建指向目录以外的硬链接文件，并确保 safe_data_path 目录权限最小化。

12.47.分隔符

delimiter_name

参数说明：保存一个 delimiter 分隔符名称。

gsql 客户端识别到分隔符的时候，会立即将输入的一条或多条 SQL 语句发送到服务端执行。该用法可以用在输入语句较多，并且语句中存在分号时，指定一个特殊的符号作为结束符。 此参数只提供在 gsql 客户端设置，可以通过 DELIMITER 命令配置。

该参数属于 USERSET 类型参数，请参考重设参数表 1 中对应设置方法进行设置。

取值范围：字符串，长度大于 0

默认值： ";"

12.48.兼容性参数

12.48.1.Oracle 兼容性

result_case_mode

参数说明： 用于控制返回字段名的大小写。

该参数属于 USERSET 类型参数，请参考重设参数表 1 中对应设置方法进行设置。

【须知】

- 该功能仅在数据库兼容模式为 Oracle 时能够使用（即数据库实例初始化时指定 DBCOMPATIBILITY='A'），在其他数据库兼容模式下不能使用该特性。
- 该参数只影响返回字段名的大小写形式，不影响 PanWeiDB 原有的大小写匹配逻辑。
- 该参数在 postgresql.conf 文件中配置无效，仅支持在会话中配置。

取值范围： 枚举类型

- lower：参数初始化取值，未使用引号指定的字段名及别名返回纯小写形式，否则返回引号指定形式。

- upper：未使用引号指定的字段名及别名返回纯大写形式，否则返回引号指定形式。

默认值： lower

nls_date_format

参数说明： 该参数用于指定时间格式。

该参数属于 USERSET 类型参数，请参考重设参数表 1 中对应设置方法进行设置。

取值范围： 合法的时间格式字符串

默认值： 空

enable_ignore_ident_case

参数说明： 为了对 Oracle 的大小写解析特性进行兼容，PanWeiDB 提供了参数 enable_ignore_ident_case 用于控制双引号内标识符的大小写解析逻辑。

该参数属于 USERSET 类型参数，请参考重设参数表 1 中对应设置方法进行设置。

取值范围： 布尔型

- on：语法解析阶段，表名、字段名等标识符会被置为小写。
- off：语法解析阶段，不对大小写进行处理，根据原始逻辑进行解析。

默认值： off

【说明】

- 当 enable_ignore_ident_case=on 时，其作用效果与 result_case_mode = upper 的作用效果存在冲突，不建议这样使用。二者的冲突在于 result_case_mode 要求返回值必须与引号内的输入一致，但 enable_ignore_i

dent_case 为 on 时引号内的输入都返回小写。

❖ 下表对比展示了 enable_ignore_ident_case 参数的效果：

数据库类型	双引号	输入	解析
Oracle	有	小写	小写
	有	大写	大写
PanWeiDB（enable_ignore_ident_case=on）	有	小写	小写
	有	大写	小写
PanWeiDB（enable_ignore_ident_case=off）	有	小写	小写
	有	大写	大写

示例 1

❖ enable_ignore_ident_case 为 off 时（默认）：

1、创建测试表并插入数据。表名、字段名使用小写字母，无双引号。

```
create table name(id int, col text);
insert into name values(1001,'panweidb');
```

2、使用大写字母进行查询，对比是否使用双引号的区别。

```
select ID,COL from NAME;      --大写字母，无双引号
select "ID","COL" from "NAME";  --大写字母，有双引号
```

无双引号时，查询成功：

```
id   |  col
------+----------
1001  | panweidb
(1 row)
使用带双引号的大写字母，查询失败。报错信息如下：ERROR:  relation "NAME"
 does not exist on node1
LINE 1: select "ID","COL" from "NAME";
                    ^
```

❖ enable_ignore_ident_case 为 on 时：

```
set enable_ignore_ident_case = on;
```

1、创建测试表并插入数据。表名、字段名使用小写字母，无双引号。

```
create table book(id int, author text);
```

```
insert into book values(10976458,'MoYan');
```

2、使用大写字母进行查询，对比是否使用双引号的区别。

```
select ID,AUTHOR from BOOK;       --大写字母，无双引号
select "ID","AUTHOR" from "BOOK";  --大写字母，有双引号
```

开启参数后，标识符即使被双引号包裹，但仍被解析为小写字母，能够成功查询。上述两条查询语句的返回结果依次为：

```
  id   | author
----------+--------
10976458 | MoYan
(1 row)

  id   | author
----------+--------
10976458 | MoYan
(1 row)
```

【注意】

注意，若在开启参数前，已在数据库中创建了名为“BOOK”的表（解析为大写），那么开启参数后，则无法使用“BOOK”去匹配该表（解析为小写），为了避免此类情况的发生，请注意设置参数的时机。参考示例 2。

示例 2

1、当参数 enable_ignore_ident_case 为 off 时（默认）：

```
set enable_ignore_ident_case=off;
show enable_ignore_ident_case;
```

2、创建测试表并插入数据。表名使用双引号包围的大写字母。

```
create table "BOOK"(id int, name text);
```

3、向测试表插入数据并查看测试表内容，需使用双引号包围的大写字母。

```
insert into "BOOK" values(1001,'TheLittlePrince');
select * from "BOOK";
```

能够正常插入和查询，返回结果如下：

```
 id |     name
----+-----------------
 1001 | TheLittlePrince
(1 row)
```

4、此时设置参数 enable_ignore_ident_case 为 on：

```
set enable_ignore_ident_case=on;
```

5、再次向测试表插入数据时，以下写法均无法匹配之前创建的”BOOK”表：

```
insert into "BOOK" values (1003,'HarryPotter');
insert into BOOK values (1003,'HarryPotter');
insert into book values (1003,'HarryPotter');
```

均得到如下报错信息：

```
ERROR:  relation "book" does not exist on node1
```

func_colname_with_args

参数说明： 该参数用于控制函数返回结果的列名是否带参数列表，即返回结果的字段名称与查询语句中一致。

该参数属于 USERSET 类型参数，请参考重设参数表 1 中对应设置方法进行设置。

取值范围： 布尔型

- on：函数返回结果带参数。
- off：函数返回结果不带参数。

默认值： off

示例：

1、设置 func_colname_with_args 参数。

```
set func_colname_with_args=on;
```

2、执行函数查询。

```
select to_char(123) from dual;
```

返回结果为：

```
to_char(123)
------------
 123
(1 row)
```

oracle_vpd_enabled

参数说明： 该参数用于控制 oracle_vpd 功能。

该参数属于 SIGHUP 类型参数，请参考重设参数表 1 中对应设置方法进行设置。

取值范围： 布尔型

- on：表示打开 oracle_vpd 功能。
- off：表示关闭 oracle_vpd 功能。

默认值： off

pw_date_type

参数说明： Oracle 兼容模式下，date 类型的控制参数，用来控制 date 类型的底层存储类型。

该参数属于 POSTMASTER 类型参数，请参考重设参数表 1 中对应设置方法进行设置。

取值范围： 布尔型

- 1：date 底层存储为 oradate。
- 2：date 底层存储为 timestamp。

默认值： 1

示例：

示例 1： Oracle 兼容下设置 date 类型控制参数。

1、创建数据库，设置兼容模式为 Oracle。

```
CREATE DATABASE date_test_Oracle DBCOMPATIBILITY 'A';
```

2、配置 date 类型控制参数 pw_date_type。

```
alter system set pw_date_type = 1;
```

3、重启数据库。

```
pw_ctl restart
```

4、查看 date 兼容效果。

```
\c date_test_Oracle
select '2022-08-19'::date;
```

查询结果显示为：

```
    oradate
---------------------
 2022-08-19 00:00:00
(1 row)
```

示例 2：Oracle 兼容模式下，通过修改 postgresql.conf 设置 date 类型控制参数。

1、配置 date 类型控制参数。

```
vi $PGDATA/postgresql.conf
//设置参数:
pw_date_type = 2;
```

2、重启数据库。

```
pw_ctl restart
```

3、创建数据库。

```
CREATE DATABASE date_test_postgresqlconf DBCOMPATIBILITY 'A';
\c date_test_postgresqlconf
```

4、查看 date 兼容效果。

```
select '2022-08-19'::date;
```

查询结果显示为：

```
   timestamp
---------------------
 2022-08-19 00:00:00
(1 row)
```

lob_return_lobloc

参数说明： 该参数用于控制查询 lob 字段的返回内容。关闭该参数，查询 lob 字段时直接返回真实数据；开启该参数，则返回 lob 定位器，需要借助 DBMS_LOB.READ 来获取真实数据。

【说明】

开启此参数适用于原始数据特别大，不能通过 select 语句直接返回的情况。

该参数仅在数据库兼容模式为 Oracle 时有效（即数据库实例初始化时指定 DBCOMPATIBILITY='A'）。

该参数属于 USERSET 类型参数，请参考重设参数表 1 中对应设置方法进行设置。

取值范围： 布尔型

- on：查询 lob 字段时返回 lob 定位器。
- off：查询 lob 字段时直接返回真实数据。

默认值： off

enable_oralob_type

参数说明：用于控制当前实例中 SQL 涉及的 LOB 类型的解析方式。

该参数属于 POSTMASTER 类型参数，请参考重设参数表 1 中对应设置方法进行设置。

【须知】

- 此参数仅在数据库兼容模式为 Oracle 时有效（即数据库实例初始化时指定 DBCOMPATIBILITY='A'）。
- 此参数只能在实例初始化完成后，第一次启动前在配置文件中进行修改。
- 如果还有 pw_initdb 初始化实例时指定了参数--enable-oralob-type，则此参数必须设置为 on，否则无需设置或设置为 off。

12.48.2. MySQL 兼容性

sql_ignore_strategy

参数说明： 在 MySQL 兼容模式下，该参数可控制 ignore_error 的 hint 在违反非空约束时的处理策略。其他相关内容可参考 INSERT IGNORE。

【须知】

该功能仅在数据库兼容模式为 MySQL 时能够使用（即数据库实例初始化时指定 DBCOMPATIBILITY='B'），在其他数据库兼容模式下不能使用该特性。

该参数属于 USERSET 类型参数，请参考重设参数表 1 中对应设置方法进行设置。

取值范围： 枚举类型

- ignore_null: 忽略违反非空约束的行的处理，并继续执行剩余数据库操作。
- overwrite_null: 将违反约束的 null 值覆写为目标类型的默认值，并继续执行剩余数据库操作。

默认值： ignore_null

lower_case_table_names

参数说明： 在 MySQL 兼容模式下，控制表名的大小写敏感特性。具体内容请参考设置表名大小写敏感。

该参数属于 INTERNAL 类型参数，请参考重设参数表 1 中对应设置方法进行设置。

【须知】

- 该功能仅在数据库兼容模式为 MySQL 时能够使用（即数据库实例初始化时指定 DBCOMPATIBILITY='B'），在其他数据库兼容模式下不能使用该特性。
- 创建数据库后，用户无法修改此参数。

取值范围： 0，1

- 0：对象名使用创建时的大小写样式保存，在比较判断时使用大小写敏感的方式。
- 1：对象名统一使用小写样式保存，在比较判断时使用大小写不敏感的方式。

默认值： 0

b_compatibility_mode

参数说明： 该参数是当 PanWeiDB 与 MySQL 有相同的语法内容，而功能有冲突时，区分使用 MySQL 的功能还是 PanWeiDB 的功能，开启该参数则使用 MySQL 的功能。

该参数属于 USERSET 类型参数，请参考重设参数表 1 中对应设置方法进行设置。

取值范围： 布尔型

- on：表示使用 MySQL 功能。
- off：表示使用 PanWeiDB 功能。

默认值： off

enable_convert_illegal_chars

参数说明： 该参数用于控制对无法识别的字符的转换功能。

该参数属于 USERSET 类型参数，请参考重设参数表 1 中对应设置方法进行设置。

【说明】

该参数仅在数据库兼容模式为 MySQL 时生效（即数据库实例初始化时指定 DBCOMPATIBILITY='B'）。

取值范围： 布尔型

- on：支持对非法字符进行转换操作，会将非法字符转换为？。

❖ off：不支持转换无法识别的字符。

默认值： off

div_precision_increment

参数说明： 当参数大于 0 时开启功能，将使用 numeric 类型计算时的最后精度按照 MySQL 对 numeric 类型运算后的规则进行一些截断处理。

该参数属于 USERSET 类型参数，请参考重设参数表 1 中对应设置方法进行设置。

取值范围： 0~30

默认值： 0

panweidb_sql_mode

参数说明： PanWeiDB 在 MySQL 兼容模式下，支持使用 GUC 参数 panweidb_sql_mode 控制 B 兼容下的语法校验规则。

【须知】
该功能仅在数据库兼容模式为 MySQL 时支持（即数据库实例初始化时指定 DBCOMPATIBILITY='B'）。
仅在 MySQL 兼容模式下校验此参数。在其他兼容模式下修改此参数值并不会报错，但修改不会生效。
若将此参数取值设为空字符串，则表示不启用任何相关特性。
参数值由若干个配置项用逗号隔开构成，例如：set panweidb_sql_mode='only_full_group_by,sql_mode_strict';。

该参数属于 USERSET 类型参数，请参考重设参数表 1 中对应设置方法进行设置。

取值范围： 字符串，仅允许合法字符串设定。参数值由若干个配置项用逗号隔开构成。

panweidb_sql_mode 取值范围

取值	说明	用法参考
ONLY_FULL_GROUP_BY	应用于下列两种场景： ● 出现在 select 列表中的列，一定要出现在 group by 子句中。 ● 出现在 order by 中的列，一定要出现在 select distinct 中。	SELECT
ANSI_QUOTES	主要是针对出现在各种需要使用双引号表示字符串值的地方。 ● 当 ansi_quotes 打开，就表示此时的双引号中的内容要作为对象引用看待。 ● 当 ansi_quotes 关闭时，表示双引号中的内容要作为字符串的值看待。	《MySQL 兼容性手册》“操作符>双引号解释为标识符”。
SQL_MODE_STRICT	在为字段设置 UNSIGNED 属性后，限制插入的列值正负和长度。涉及的类型有 tinyint、smallint、int、integer、bigint、mediumint。	《MySQL 兼容性手册》“SQL 语法>COLUMN 定义支持 UNSIGNED 属性”。
pipes_as_concat	控制||被当成连接符还是或操作符。 ● 当 pipes_as_concat 打开，表示||被当成连接符。 ● 当 pipes_as_concat 关闭，表示||被当成操作符。	参考下方示例 1。
pad_char_to_full_length	控制 char 类型查询时是否删除尾部空格。 ● 当 pad_char_to_full_length 打开，就表示 char 类型查询时尾部空格不会被删除。 ● 当 pad_char_to_full_length 关闭，就表示 char 类型查询时尾部空格会被删除。	参考下方示例 2。
no_zero_date	控制 '0000-00-00' 是否为合法日期，支持 DATE、DATETIME 类型。取值及表现如下所示： ● no_zero_date, sql_mode_strict：非法日期，报错（使用 update/insert ignore 时告警） ● no_zero_date：非法日期，告警 ● sql_mode_strict：合法日期，无告警 ● –：合法日期，无告警	参考下方示例 3。

默认值：
ONLY_FULL_GROUP_BY,ANSI_QUOTES,pipes_as_concat,pad_char_to_full_length,sql_mode_strict,no_zero_date

示例

示例 1： pipes_as_concat 选项效果展示。

1、创建测试表。

```
create table test1(a int,b int);
```

2、设置 GUC 参数 panweidb_sql_mode= ''。

```
SET panweidb_sql_mode='';
```

3、向表中插入数据。

```
insert into test1 values(1||2,2);
```

4、查询 test1 表数据。

```
select * from test1;
```

返回结果为：

```
a | b
---+---
 1 | 2
(1 row)
```

5、设置 GUC 参数 panweidb_sql_mode= 'pipes_as_concat'，并查询结果。

```
set panweidb_sql_mode= 'pipes_as_concat';
show panweidb_sql_mode;
```

返回结果为：

```
panweidb_sql_mode
------------------
 pipes_as_concat
(1 row)
```

6、向表中插入数据。

```
insert into test1 values(1||2,2);
```

7、查询 test1 表数据。

```
select * from test1;
```

如下所示，由于 panweidb_sql_mode 参数包含 pipes_as_concat 选项，因此|| 的功能从或运算改为字符串拼接：

```
  a  | b
-----+---
  1  | 2
 12  | 2
(2 rows)
```

示例 2： pad_char_to_full_length 选项效果展示。

1、创建测试表。

```
create table t1(x char(5),y varchar(5));
```

2、向表格中插入数据（char 存储 5 个字符，而 varchar 存储 4 个字符）。

```
insert into t1 values('数据库 ','数据库 ');
```

3、设置 GUC 参数 panweidb_sql_mode=''。

```
SET panweidb_sql_mode='';
```

4、查询表中存储数据的长度。

```
select x,char_length(x),y,char_length(y) from t1;
```

如下所示，char 类型数据会把尾部空格删除，而 varchar 不会：

```
   x    | char_length |   y    | char_length
--------+-------------+--------+-------------
 数据库 |           3 | 数据库 |           4
(1 row)
```

5、设置 GUC 参数 panweidb_sql_mode='pad_char_to_full_length'，并查询结果。

```
set panweidb_sql_mode= 'pad_char_to_full_length';
show panweidb_sql_mode;
```

返回结果为：

```
  panweidb_sql_mode
------------------------
 pad_char_to_full_length
(1 row)
```

6、再次查询表中存储数据的长度。

```
select x,char_length(x),y,char_length(y) from t1;
```

如下所示，由于 panweidb_sql_mode 参数包含 pad_char_to_full_length 选项，因此 char 类型数据尾部空格不会被自动删除：

```
  x   | char_length |  y   | char_length
----------+-------------+---------+-------------
数据库  |           5 | 数据库 |           4
(1 row)
```

示例 3： no_zero_date 选项效果展示。

1、设置 GUC 参数 panweidb_sql_mode= 'no_zero_date'，并查询结果。

```
set panweidb_sql_mode= 'no_zero_date';
show panweidb_sql_mode;
```

返回结果为：

```
panweidb_sql_mode
-------------------
 no_zero_date
(1 row)
```

2、创建测试表。

```
CREATE TABLE my_table_1160646 (
id INT AUTO_INCREMENT PRIMARY KEY,
date_column DATE
);
```

3、插入数据。

```
INSERT INTO my_table_1160646 (date_column) VALUES ('2022-01-01');
INSERT INTO my_table_1160646 (date_column) VALUES ('0000-00-00');
```

由于设置了 panweidb_sql_mode= 'no_zero_date'，因此'0000-00-00'为非法日期，插入成功但发出告警，返回结果如下：

```
INSERT 0 1
WARNING:  date/time field value out of range: "0000-00-00"
LINE 1: ...NSERT INTO my_table_1160646 (date_column) VALUES ('0000-00-0...
                                                             ^
CONTEXT:  referenced column: date_column
INSERT 0 1
```

4、设置 GUC 参数 panweidb_sql_mode= 'sql_mode_strict'，并查询结果。

```
set panweidb_sql_mode= 'sql_mode_strict';
show panweidb_sql_mode;
```

返回结果为：

```
panweidb_sql_mode
------------------
 sql_mode_strict
(1 row)
```

5、再次插入日期'0000-00-00'。

```
INSERT INTO my_table_1160646 (date_column) VALUES ('0000-00-00');
```

插入成功，返回结果如下：

```
INSERT 0 1
```

6、查询表内容。

```
select * from my_table_1160646;
```

返回结果为：

```
id | date_column
----+-------------
 1 | 2022-01-01
```

```
 2 | 0000-00-00
 3 | 0000-00-00
(3 rows)
```

b_compatibility_user_host_auth

参数说明： 控制是否允许创建`user@host`、`'user'@'host'`之类的用户并兼容 MySQL 的 user@host 认证鉴权，对兼容 MySQL 的 user@host 进行认证时，需要在配置文件 postgresql.conf 中设为 on。
该参数属于 POSTMASTER 类型参数，请参考重设参数表 1 中对应设置方法进行设置。

取值范围： 布尔型

- on：允许创建`user@host`、`'user'@'host'`之类的用户并兼容 MySQL 的 user@host 认证鉴权。
- off：不允许创建`user@host`、`'user'@'host'`之类的用户。

默认值： off

default_week_format

参数说明： 参数值为整数，该参数影响 B 模式下的 week 函数，该参数的取值范围为[0,7]，分别对应 8 种不同的计算策略，这些策略的详细内容参见《PanWeiDB V2.0-S3.0.1_B01 MySQL 兼容性手册》的 week 函数。当此 GUC 参数设置的值超过对应边界值时，会报 warning，并且将此 GUC 参数的值设置为对应边界值。

该参数属于 SIGHUP 类型参数，请参考重设参数表 1 中对应设置方法进行设置。

取值范围： 整型，[0, 7]

默认值： 0

示例：

1、查看 default_week_format 默认值。

```
show default_week_format;
```

返回结果如下：

```
default_week_format
--------------------
0
(1 row)
```

2、调用 week 函数。

```
select week('2000-1-1');
```

返回结果如下：

```
week
------
   0
(1 row)
```

3、修改 GUC 参数并再次调用 week 函数。

```
alter system set default_week_format = 2;
select week('2000-1-1');
```

返回结果如下：

```
ALTER SYSTEM SET
week
------
52
(1 row)
```

max_error_count

参数说明：控制`show warnings/errors`语句，输出的 error, warning, note 信息的最大数量。

该参数属于 USERSET 类型参数，请参考重设参数表 1 中对应设置方法进行设置。

取值范围： 整数型，0~65535

默认值： 64

enable_convert_illegal_chars

参数说明： 该参数用于控制对无法识别的字符的转换功能。

该参数属于 USERSET 类型参数，请参考重设参数表 1 中对应设置方法进行设置。

【说明】
该参数仅在数据库兼容模式为 MySQL 时生效（即数据库实例初始化时指定 DBCOMPATIBILITY='B'）。

取值范围： 布尔型

- on：支持对非法字符进行转换操作，会将非法字符转换为？。
- off：不支持转换无法识别的字符。

默认值： off

func_colname_with_args

参数说明： 该参数用于控制函数返回结果的列名是否带参数列表，即返回结果的字段名称与查询语句中一致。

该参数属于 USERSET 类型参数，请参考重设参数表 1 中对应设置方法进行设置。

【说明】
该参数仅在数据库兼容模式为 Oracle 或者 MySQL 时支持（即数据库实例初始化时指定 DBCOMPATIBILITY='A'或者'B'）。

取值范围：布尔型

- on：函数返回结果带参数。
- off：函数返回结果不带参数。

默认值：off

b_format_behavior_compat_options

参数说明： 数据库 B 模式兼容性行为配置项，该参数的值由若干个配置项用逗号隔开构成。

该参数属于 USERSET 类型参数，请参考重设参数表 1 中对应设置方法进行设置。

取值范围： 字符串

默认值： “”

【说明】

- 该参数仅在数据库兼容模式为 MySQL 时生效（即数据库实例初始化时指定 DBCOMPATIBILITY='B'）。
- 该参数属于会话级别参数，必须在配置文件设置或者 session 级别设置才会生效。
- 配置多个兼容性配置项时，相邻配置项用逗号隔开，例如：set b_format_behavior_compat_options='enable_set_variables,set_session_transaction';

当前只支持下表所示的兼容性 B 模式配置项。

兼容性配置项	兼容性行为控制
enable_set_variables	set 语法增强控制开关。 ❖ 不设置此配置时，不支持 set 自定义变量、set [global | session]语法。

	❖ 设置此配置时，支持 B 兼容模式下使用上述语法，比如 set @v1 = 1;。
set_session_transaction	set session transaction 控制开关。 ❖ 不设置此配置时，set session transaction 等效于 set local transaction。 ❖ 设置此配置时，支持 B 兼容模式下使用上述语法，修改当前会话事务特性。
enable_modify_column	ALTER TABLE MODIFY 语义控制开关。 ❖ 不设置此配置时，ALTER TABLE table_name MODIFY column_name data_type;只修改列的数据类型。 ❖ 设置此配置时，ALTER TABLE table_name MODIFY column_name data_type;修改整个列定义。
default_collation	默认字符序前向兼容开关。 ❖ 不设置此配置时，在未显式指定字符类型字段的字符集或字符序且表级字符序也为空时，字段为 default 字符序。 ❖ 设置此配置时，字符类型字段的字符序当表级字符序不为空时继承表级字符序，为空时设置为数据库编码对应的默认字符序。
fetch	FETCH 语句结尾报错兼容开关。 ❖ 不设置此配置时，FETCH 抓取完了所有可用行，它就停在最后一行后面，或者在反向抓取的情况下是停在第一行前面。 ❖ 设置此配置时，FETCH 抓取完了所有可用行后报错。
enable_multi_charset	多字符集功能的控制开关。 ❖ 不设置此配置时，不允许不同于数据

	库字符集的数据，不处理不同于数据库字符集的表达式。 ❖ 设置此配置时，允许数据库中含有不同于数据库字符集的数据，合并不同字符集的表达式。不同字符集的表达式运算规则详见：字符集与字符序。 【说明】 目前 PanWeiDB 暂不支持字符集的合并功能，不表示多字符集功能的控制开关，目前该参数开启只是为了可以设置 collation_connection。

character_set_connection

参数说明： 在兼容 B 模式（sql_compatibility = 'B'），并且 GUC 参数 b_format_behavior_compat_options 设置值包含 enable_multi_charset 的场景下，连接到数据库时自动设置、生效。用于设置字符串常量的默认字符集。

该参数属于 USERSET 类型参数，请参考重设参数表 1 中对应设置方法进行设置。仅在设置会话级时生效，设置数据库级和用户级不生效。

修改此参数会同时将 GUC 参数 collation_connection 设置成该字符集的默认字符序。

取值范围： 字符串，合法的字符序的字符串，例如 utf8_general_ci

默认值： 当前数据库的字符集。

collation_connection

参数说明： 该参数用于指定常量字符串的排序规则。只有当参数 b_format_behavior_compat_options 包含选项 enable_multi_charset 的场景下才允许设置该参数。

该参数属于 USERSET 类型参数，请参考重设参数表 1 中对应设置方法进行设置。

【说明】
该参数仅在数据库兼容模式为 MySQL 时生效（即数据库实例初始化时指定 DBCOMPATIBILITY='B'）。
该参数属于会话级别参数，必须在配置文件设置或者 session 级别设置才会生效。

取值范围： 字符串，合法的字符序的字符串，例如 utf8_general_ci

默认值： 当前数据库字符集的默认字符序

b_func_display_mode

参数说明： 控制 B 模式下函数查询时的显示。该参数不支持通过 SET 语句进行设置，需在 postgresql.conf 中配置后重启数据库生效。

该参数属于 POSTMASTER 类型参数，请参考重设参数表 1 中对应设置方法进行设置。

取值范围： 布尔型

- on：开启该参数，在 B 模式下查询函数显示的结果与查询语句相同。
- off：关闭该参数，在 B 模式下查询函数显示的结果与原来一样。

默认值： off

示例：

1、修改配置文件 postgresql.conf 设置 GUC 参数'b_func_display_mode'为'on'，并重启数据库使参数生效。

2、一条语句中多次使用同一函数。

```
select date_format(now(),'yyyymmdd'),date_format(now()+1,'yyyymmdd');
```

结果返回如下：

```
date_format(now(),'yyyymmdd') | date_format(now()+1,'yyyymmdd')
-------------------------------+--------------------------------
 20231107                      | 20231108
(1 row)
```

12.48.3.SQL SERVER 兼容性

enable_set_variable_mssql_format

参数说明： 该参数用于控制在 SQL Server 数据库兼容模式下是否支持自定义用户变量的功能。

该参数属于 USERSET 类型参数，请参考重设参数表 1 中对应设置方法进行设置。

取值范围： 布尔型

- on：表示在 SQLServer 数据库模式下支持自定义用户变量。
- off：表示在 SQLServer 数据库模式下不支持自定义用户变量。

默认值： off

result_case_mode

参数说明： 用于控制返回字段名的大小写。当用户在 CREATE TABLE、ALTER TABLE、SELECT INTO 等语句时，指定的列字段名称显式的被双引号包含，则在 SELECT 语句的输出结果中，列字段名称的大小写与引号内的内容保持一致。

该参数属于 USERSET 类型参数，请参考重设参数表 1 中对应设置方法进行设置。

【须知】

- 该参数只影响返回字段名的大小写形式，不影响 PanWeiDB 原有的大小写匹配逻辑。

- 不支持在 postgresql.conf 文件中配置该参数，可能导致严重问题，仅支持在会话中配置。

取值范围： 枚举类型

- lower：参数初始化取值，未使用引号指定的字段名及别名返回纯小写形式，否则返回引号指定形式。
- upper：未使用引号指定的字段名及别名返回纯大写形式，否则返回引号指定形式。

默认值： lower

示例：

1、数据库初始化和用户与数据库建立连接后配置 result_case_mode 的值为 upper。

```
set result_case_mode="upper";
```

2、查看参数值。

```
show result_case_mode;
```

返回结果为：

```
result_case_mode
--------------------------
 upper
(1 row)
```

3、创建测试表。

```
create table test(aa int,"bb" int,Dd int,"Ee" int);
```

4、查询字段名带引号的结果。

```
select * from test;
```

返回结果为如下，其中被双引号包含的字段保持其大小写格式，未被包含的字段均变为大写：

```
AA | bb | DD | Ee
----+----+----+----
(0 rows)
```

mssql_mbracket

参数说明： 该参数用于控制在 SQL Server 数据库兼容模式下是否支持使用中括号作为分隔标识符，用于标识数据库对象。

该参数属于 USERSET 类型参数，修改完成后需要重启数据库服务才能生效，具体请参考重设参数表 1 中对应设置方法进行设置。

取值范围： 布尔型

- on：表示在 SQL Server 数据库模式下支持分隔标识符`[]`。
- off：表示在 SQL Server 数据库模式下不支持分隔标识符`[]`。

默认值： off

13. 配置运行参数

13.1.重设参数

背景信息

PanWeiDB 提供了多种修改 GUC 参数的方法，用户可以方便的针对数据库、用户、会话进行设置。

- 参数名称不区分大小写。
- 参数取值有整型、浮点型、字符串、布尔型和枚举型五类。
  - 布尔值可以是（on，off）、（true，false）、（yes，no）或者（1，0），且不区分大小写。
  - 枚举类型的取值是在系统表 pg_settings 的 enumvals 字段取值定义的。
- 对于有单位的参数，在设置时请指定单位，否则将使用默认的单位。
  - 参数的默认单位在系统表 pg_settings 的 unit 字段定义的。
  - 内存单位有：KB（千字节）、MB（兆字节）和 GB（吉字节）。
  - 时间单位：ms（毫秒）、s（秒）、min（分钟）、h（小时）和 d（天）。

具体参数说明请参见 GUC 参数说明章节。

GUC 参数设置

PanWeiDB 提供了六类 GUC 参数，具体分类和设置方式请参考表 1：

表 1 GUC 参数分类

参数类型	说明	设置方式
INTERNAL	固定参数，在创建数据库的时候确定，用户无法修改，只能通过 show 语法或者 pg_settings 视图进行查看。	无
POSTMASTER	数据库服务端参数，在数据库启动时确定，可以通过配置文件指定。	支持表 2 中的方式一、方式四。

SIGHUP	数据库全局参数，可在数据库启动时设置或者在数据库启动后，发送指令重新加载。	支持表 2 中的方式一、方式二、方式四。
BACKEND	会话连接参数。在创建会话连接时指定，连接建立后无法修改。连接断掉后参数失效。内部使用参数，不推荐用户设置。	支持表 2 中的方式一、方式二、方式四。 说明： 设置该参数后，下一次建立会话连接时生效
SUSET	数据库管理员参数。可在数据库启动时、数据库启动后或者数据库管理员通过 SQL 进行设置。	支持表 2 中的方式一、方式二或由数据库管理员通过方式三设置。
USERSET	普通用户参数。可被任何用户在任何时刻设置。	支持表 2 中的方式一、方式二或方式三设置。

表 2 GUC 参数设置方式

序号	设置方法

方式一	1、使用如下命令修改参数。
	`pw_guc set -D datadir -c "paraname=value"`
	说明：
	如果参数是一个字符串变量，则使用-c parameter="'value'"或者使用-c "parameter = 'value'"。
	使用以下命令在数据库节点上设置 cm_agent 某个参数。
	`pw_guc set -Z cmagent -c "paraname=value"`
	使用以下命令在数据库节点上设置 cm_server 某个参数。
	`pw_guc set -Z cmserver -c "paraname=value"`
	2、重启数据库使参数生效。
	说明：
	重启 PanWeiDB 操作会导致用户执行操作中断，请在操作之前规划好合适的执行窗口。
	`pw_ctl restart`

序号	设置方法
方式二	pw_guc reload -D datadir -c "paraname=value"
方式三	修改指定数据库、用户、会话级别的参数。 1、设置数据库级别的参数 ALTER DATABASE dbname SET paraname TO value; 在下次会话中生效。 2、设置用户级别的参数 ALTER USER username SET paraname TO value; 在下次会话中生效。 3、设置会话级别的参数 SET paraname TO value; 修改本次会话中的取值。退出会话后，设置将失效。 说明： SET 设置的会话级参数优先级最高，其次是 ALTER 设置的，其中 ALTER DATABASE 设置的参数值优先级高于 ALTER USER 设置，这三种设置方式设置的优先级都高于 pw_guc 设置方式。
方式四	使用 ALTER SYSTEM SET 修改数据库参数。 设置 POSTMASERT 级别的参数 ALTER SYSTEM SET paraname TO value; 重启后生效。 设置 SIGHUP 级别的参数 ALTER SYSTEM SET paraname TO value; 立刻生效(实际等待线程重新加载参数略有延迟)。 设置 BACKEND 级别的参数 ALTER SYSTEM SET paraname TO value; 在下次会话中生效。

【注意】

使用方式一和方式二设置参数时，若所设参数不属于当前环境，数据库会提示参数不在支持范围内的相关信息。

示例

使用方式一设置数据库参数，以在数据库主节点设置 archive_mode 参数为例。

1、以操作系统用户 panweidb 登录数据库主节点。

2、查看 archive_mode 参数，on 表示日志要进行归档操作。

```
cat /home/panweidb/data/panweidb/postgresql.conf | grep archive_mode
archive_mode = on
```

3、设置 archive_mode 参数为 off，关闭日志的归档操作。

```
pw_guc set -D $PGDATA -c "archive_mode=off"
```

4、重启数据库使参数生效。

```
pw_ctl restart
```

5、使用如下命令连接数据库，panweidb 为需要连接的数据库名称，5432 为数据库主节点的端口号。

```
gsql -d panweidb -p 5432
```

6、检查参数设置的正确性。

```
SHOW archive_mode;
 archive_mode
--------
 off
(1 row)
```

使用方式二设置参数，以在数据库主节点设置 authentication_timeout 参数为例。

1、以操作系统用户 panweidb 登录数据库主节点。

2、查看 authentication_timeout 参数。

```
cat /home/panweidb/data/panweidb/postgresql.conf | grep authentication_timeo
ut
```

```
authentication_timeout = 1min
```

3、设置 authentication_timeout 参数为 59s。

```
pw_guc reload -D $PGDATA -c "authentication_timeout = 59s"

Total instances: 2. Failed instances: 0.
Success to perform gs_guc!
```

4、使用如下命令连接数据库，panweidb 为需要连接的数据库名称，5432 为数据库主节点的端口号。

```
gsql -d panweidb -p 5432
```

5、检查参数设置的正确性。

```
SHOW authentication_timeout;
 authentication_timeout
------------
 59s
(1 row)
```

使用方式三设置参数，以设置 explain_perf_mode 参数为例。

1、以操作系统用户 panweidb 登录数据库主节点。

2、使用如下命令连接数据库，panweidb 为需要连接的数据库名称，5432 为数据库主节点的端口号。

```
gsql -d panweidb -p 5432
```

3、查看 explain_perf_mode 参数。

```
SHOW explain_perf_mode;
 explain_perf_mode
-----------
 normal
(1 row)
```

4、设置 explain_perf_mode 参数。

使用以下任意方式进行设置：

❖ 设置数据库级别的参数

```
ALTER DATABASE postgres SET explain_perf_mode TO pretty;
```

当结果显示为如下信息，则表示设置成功。

```
ALTER DATABASE
```

在下次会话中生效。

❖ 设置用户级别的参数

```
ALTER USER panweidb SET explain_perf_mode TO pretty;
```

当结果显示为如下信息，则表示设置成功。

```
ALTER ROLE
```

在下次会话中生效。

❖ 设置会话级别的参数

```
SET explain_perf_mode TO pretty;
```

当结果显示为如下信息，则表示设置成功。

```
SET
```

5、检查参数设置的正确性。

```
SHOW explain_perf_mode;
 explain_perf_mode
--------
 pretty
(1 row)
```

❖ 使用方式四设置参数，以设置 autovacuum 参数为例。

1、以操作系统用户 panweidb 登录数据库主节点。

2、使用如下命令连接数据库，panweidb 为需要连接的数据库名称，5432 为数据库主节点的端口号。

```
gsql -d panweidb -p 5432
```

3、查看 autovacuum 当前值。

```
show autovacuum;
 autovacuum
------
 off
(1 row)
```

4、设置参数 autovacuum 为 on。

```
ALTER SYSTEM SET autovacuum to on;
```

5、检查参数设置的正确性。

```
show autovacuum;
 autovacuum
------
 on
(1 row)
```

13.2.查看参数当前取值

功能描述

PanWeiDB 安装后，有一套默认的运行参数，为了使 PanWeiDB 与业务的配合度更高，用户需要根据业务场景和数据量的大小进行 GUC 参数调整。

操作步骤

1、以操作系统用户 panweidb 登录数据库主节点。

2、使用如下命令连接数据库。

```
gsql -d postgres -p 5432
```

postgres 为需要连接的数据库名称，5432 为数据库主节点的端口号。

连接成功后，系统显示类似如下信息：

```
gsql ((PanWeiDB x.x.x build 290d125f) compiled at 2020-05-08 02:59:43 commit 21
43 last mr 131
Non-SSL connection (SSL connection is recommended when requiring high-securit
y)
Type "help" for help.

panweidb=#
```

3、查看数据库运行参数当前取值。

❖ 方法一：使用 SHOW 命令。

使用如下命令查看单个参数：

```
SHOW server_version;
```

server_version 显示数据库版本信息的参数。

使用如下命令查看所有参数：

```
SHOW ALL;
```

❖ 方法二：使用 pg_settings 视图。

使用如下命令查看单个参数：

```
SELECT * FROM pg_settings WHERE NAME='server_version';
```

使用如下命令查看所有参数：

```
SELECT * FROM pg_settings;
```

示例

查看服务器的版本号。

```
panweidb=# SHOW server_version;
 server_version
----------------
 9.2.4
(1 row)
```

14. 错误日志信息参考

14.1.内核错误信息

ERRMSG: "unsupported syntax: ENCRYPTED WITH in this operation"

SQLSTATE: 42601

CAUSE: "client encryption feature is not supported this operation."

ACTION: "Check client encryption feature whether supported this operation."

ERRMSG: "invalid grant operation"

SQLSTATE: 0LP01

CAUSE: "Grant options cannnot be granted to public."

ACTION: "Grant grant options to roles."

ERRMSG: "unrecognized object kind: %d"

SQLSTATE: XX004

CAUSE: "The object type is not suppor

ted for GRANT/REVOKE."

ACTION: "Check GRANT/REVOKE syntax to obtain the supported object types."

ERRMSG: "unrecognized GrantStmt.targtype: %d"

SQLSTATE: XX004

CAUSE: "The target type is not supported for GRANT/REVOKE."

ACTION: "Check GRANT/REVOKE syntax to obtain the supported target types."

ERRMSG: "invalid grant operation"

SQLSTATE: 0LP01

CAUSE: "Grant to public operation is forbidden in security mode."

ACTION: "Don't grant to public in security mode."

ERRMSG: "unrecognized object type"

SQLSTATE: XX004

CAUSE: "The object type is not supported for GRANT/REVOKE."

ACTION: "Check GRANT/REVOKE syntax to obtain the supported object types."

ERRMSG: "invalid grant/revoke operation"

SQLSTATE: 0LP01

CAUSE: "Column privileges are only valid for relations in GRANT/REVOKE."

ACTION: "Use the column privileges only for relations."

ERRMSG: "invalid AccessPriv node"

SQLSTATE: 0LP01

CAUSE: "System error."

ACTION: "Contact engineer to support."

ERRMSG: "unrecognized GrantStmt.objtype: %d"

SQLSTATE: XX004

CAUSE: "The object type is not supported for GRANT/REVOKE."

ACTION: "Check GRANT/REVOKE syntax to obtain the supported object types."

ERRMSG: "undefined client master key"

SQLSTATE: 42705

CAUSE: "The client master key does not exist."

ACTION: "Check whether the client master key exists."

ERRMSG: "undefined column encryption key"

SQLSTATE: 42705

CAUSE: "The column encryption key does not exist."

ACTION: "Check whether the column encryption key exists."

ERRMSG: "large object %u does not exist"

SQLSTATE: 42704

CAUSE: "The large object does not exist."

ACTION: "Check whether the large object exists."

ERRMSG: "redundant options"

SQLSTATE: 42601

CAUSE: "The syntax 'schemas' is redundant in ALTER DEFAULT PRIVILEGES statement."

ACTION: "Check ALTER DEFAULT PRIVILEGES syntax."

ERRMSG: "redundant options"

SQLSTATE: 42601

CAUSE: "The syntax 'roles' is redundant in ALTER DEFAULT PRIVILEGES statement."

ACTION: "Check ALTER DEFAULT PRIVILEGES syntax."

ERRMSG: "option '%s' not recognized"

SQLSTATE: 42601

CAUSE: "The option in ALTER DEFAULT PRIVILEGES statement is not supported."

ACTION: "Check ALTER DEFAULT PRIVILEGES syntax."

ERRMSG: "unrecognized GrantStmt.objtype: %d"

SQLSTATE: XX004

CAUSE: "The object type is not supported for ALTER DEFAULT PRIVILEGES."

ACTION: "Check ALTER DEFAULT PRIVILEGES syntax to obtain the supported object types."

ERRMSG: "invalid alter default privileges operation"

SQLSTATE: 0LP01

CAUSE: "Default privileges cannot be set for columns."

ACTION: "Check ALTER DEFAULT PRIVILEGES syntax."

ERRMSG: "unrecognized objtype: %d"

SQLSTATE: XX004

CAUSE: "The object type is not supported for default privileges."

ACTION: "Check ALTER DEFAULT PRIVILEGES syntax to obtain the supported object types."

ERRMSG: "could not find tuple for default ACL %u"

SQLSTATE: 29P01

CAUSE: "System error."

ACTION: "Contact engineer to support."

ERRMSG: "unexpected default ACL type: %d"

SQLSTATE: 0LP01

CAUSE: "The object type is not supported for default privilege."

ACTION: "Check ALTER DEFAULT PRIVILEGES syntax to obtain the supported object types."

ERRMSG: "invalid object id"

SQLSTATE: 0LP01

CAUSE: "The object type is not supported for GRANT/REVOKE."

ACTION: "Check GRANT/REVOKE syntax to obtain the supported object types."

ERRMSG: "undefined column"

SQLSTATE: 42703

CAUSE: "The column of the relation does not exist."

ACTION: "Check whether the column exists."

ERRMSG: "column number out of range"

SQLSTATE: 0LP01

CAUSE: "System error."

ACTION: "Contact engineer to support."

ERRMSG: "cache lookup failed for attribute %d of relation %u"

SQLSTATE: 29P01

CAUSE: "System error."

ACTION: "Contact engineer to support."

ERRMSG: "cache lookup failed for relation %u"

SQLSTATE: 29P01

CAUSE: "System error."

ACTION: "Contact engineer to support."

ERRMSG: "unsupported object type"

SQLSTATE: 42809

CAUSE: "Index type is not supported for GRANT/REVOKE."

ACTION: "Check GRANT/REVOKE syntax to obtain the supported object types."

ERRMSG: "unsupported object type"

SQLSTATE: 42809

CAUSE: "Composite type is not supported for GRANT/REVOKE."

ACTION: "Check GRANT/REVOKE syntax to obtain the supported object types."

ERRMSG: "wrong object type"

SQLSTATE: 42809

CAUSE: "GRANT/REVOKE SEQUENCE only support sequence objects."

ACTION: "Check GRANT/REVOKE syntax to obtain the supported object types."

ERRMSG: "invalid privilege type USAGE for table"

SQLSTATE: 0LP01

CAUSE: "GRANT/REVOKE TABLE do not support USAGE privilege."

ACTION: "Check GRANT/REVOKE syntax to obtain the supported privilege types for tables."

ERRMSG: "invalid privilege type %s for column"

SQLSTATE: 0LP01

CAUSE: "The privilege type is not supported for column object."

ACTION: "Check GRANT/REVOKE syntax to obtain the supported privilege types for column object."

ERRMSG: "cache lookup failed for database %u"

SQLSTATE: 29P01

CAUSE: "System error."

ACTION: "Contact engineer to support."

ERRMSG: "cache lookup failed for foreign-data wrapper %u"

SQLSTATE: 29P01

CAUSE: "System error."

ACTION: "Contact engineer to support."

ERRMSG: "cache lookup failed for foreign server %u"

SQLSTATE: 29P01

CAUSE: "System error."

ACTION: "Contact engineer to support."

ERRMSG: "cache lookup failed for function %u"

SQLSTATE: 29P01

CAUSE: "System error."

ACTION: "Contact engineer to support."

ERRMSG: "cache lookup failed for language %u"

SQLSTATE: 29P01

CAUSE: "System error."

ACTION: "Contact engineer to support."

ERRMSG: "Grant/revoke on untrusted languages if forbidden."

SQLSTATE: 0LP01

CAUSE: "Grant/revoke on untrusted languages if forbidden."

ACTION: "Support grant/revoke on trusted C languages"

ERRMSG: "Forbid grant language c to user with grant option."

SQLSTATE: 0A000

CAUSE: "Forbid grant language c to user with grant option."

ACTION: "Only support grant language c to user."

ERRMSG: "Forbid grant language c to public."

SQLSTATE: 0A000

CAUSE: "Forbid grant language c to public."

ACTION: "Grant language c to specified users."

ERRMSG: "cache lookup failed for large object %u"

SQLSTATE: 29P01

CAUSE: "System error."

ACTION: "Contact engineer to support."

ERRMSG: "cache lookup failed for namespace %u"

SQLSTATE: 29P01

CAUSE: "System error."

ACTION: "Contact engineer to support."

ERRMSG: "cache lookup failed for tablespace %u"

SQLSTATE: 29P01

CAUSE: "System error."

ACTION: "Contact engineer to support."

ERRMSG: "cache lookup failed for type %u"

SQLSTATE: 29P01

CAUSE: "System error."

ACTION: "Contact engineer to support."

ERRMSG: "cannot set privileges of array types"

SQLSTATE: 0LP01

CAUSE: "Cannot set privileges of array types."

ACTION: "Set the privileges of the element type instead."

ERRMSG: "wrong object type"

SQLSTATE: 42809

CAUSE: "GRANT/REVOKE DOMAIN only support domain objects."

ACTION: "Check GRANT/REVOKE syntax to obtain the supported object types."

ERRMSG: "cache lookup failed for data source %u"

SQLSTATE: 29P01

CAUSE: "System error."

ACTION: "Contact engineer to support."

ERRMSG: "cache lookup failed for client master key %u"

SQLSTATE: 29P01

CAUSE: "System error."

ACTION: "Contact engineer to support."

ERRMSG: "cache lookup failed for column encryption key %u"

SQLSTATE: 29P01

CAUSE: "System error."

ACTION: "Contact engineer to support."

ERRMSG: "cache lookup failed for directory %u"

SQLSTATE: 29P01

CAUSE: "System error."

ACTION: "Contact engineer to support."

ERRMSG: "unrecognized privilege type '%s'"

SQLSTATE: 42601

CAUSE: "The privilege type is not supported."

ACTION: "Check GRANT/REVOKE syntax to obtain the supported privilege types."

ERRMSG: "unrecognized privilege: %d"

SQLSTATE: XX004

CAUSE: "The privilege type is not supported."

ACTION: "Check GRANT/REVOKE syntax to obtain the supported privilege types."

ERRMSG: "unrecognized AclResult"

SQLSTATE: XX004

CAUSE: "System error."

ACTION: "Contact engineer to support."

ERRMSG: "permission denied for column '%s' of relation '%s'"

SQLSTATE: 42501

CAUSE: "Insufficient privileges for the column."

ACTION: "Select the system tables to get the acl of the column."

ERRMSG: "role with OID %u does not exist"

SQLSTATE: 42704

CAUSE: "System error."

ACTION: "Contact engineer to support."

ERRMSG: "unrecognized objkind: %d"

SQLSTATE: XX004

CAUSE: "The object type is not supported for privilege check."

ACTION: "Check GRANT/REVOKE syntax to obtain the supported object types."

ERRMSG: "attribute %d of relation with OID %u does not exist"

SQLSTATE: 42703

CAUSE: "System error."

ACTION: "Contact engineer to support."

ERRMSG: "the column cm been dropped"

SQLSTATE: 42703

CAUSE: "The column does not exist."

ACTION: "Check whether the column exists."

ERRMSG: "relation with OID %u does not exist"

SQLSTATE: 42P01

CAUSE: "System error."

ACTION: "Contact engineer to support."

ERRMSG: "invalid group"

SQLSTATE: 22000

CAUSE: "System error."

ACTION: "Contact engineer to support."

ERRMSG: "database with OID %u does not exist"

SQLSTATE: 3D000

CAUSE: "System error."

ACTION: "Contact engineer to support."

ERRMSG: "directory with OID %u does not exist"

SQLSTATE: 42704

CAUSE: "System error."

ACTION: "Contact engineer to support."

ERRMSG: "function with OID %u does not exist"

SQLSTATE: 42883

CAUSE: "System error."

ACTION: "Contact engineer to support."

ERRMSG: "client master key with OID %u does not exist"

SQLSTATE: 42705

CAUSE: "System error."

ACTION: "Contact engineer to support."

ERRMSG: "language with OID %u does not exist"

SQLSTATE: 42704

CAUSE: "System error."

ACTION: "Contact engineer to support."

ERRMSG: "large object %u does not exist"

SQLSTATE: 42704

CAUSE: "System error."

ACTION: "Contact engineer to support."

ERRMSG: "schema with OID %u does not exist"

SQLSTATE: 3F001

CAUSE: "System error."

ACTION: "Contact engineer to support."

ERRMSG: "tablespace with OID %u does not exist"

SQLSTATE: 42704

CAUSE: "System error."

ACTION: "Contact engineer to support."

ERRMSG: "foreign-data wrapper with OID %u does not exist"

SQLSTATE: 42704

CAUSE: "System error."

ACTION: "Contact engineer to support."

ERRMSG: "foreign server with OID %u does not exist"

SQLSTATE: 42704

CAUSE: "System error."

ACTION: "Contact engineer to support."

ERRMSG: "data source with OID %u does not exist"

SQLSTATE: 42704

CAUSE: "System error."

ACTION: "Contact engineer to support."

ERRMSG: "type with OID %u does not exist"

SQLSTATE: 42704

CAUSE: "System error."

ACTION: "Contact engineer to support."

ERRMSG: "operator with OID %u does not exist"

SQLSTATE: 42883

CAUSE: "System error."

ACTION: "Contact engineer to support."

ERRMSG: "column encryption key with OID %u does not exist"

SQLSTATE: 42705

CAUSE: "System error."

ACTION: "Contact engineer to support."

ERRMSG: "operator class with OID %u does not exist"

SQLSTATE: 42704

CAUSE: "System error."

ACTION: "Contact engineer to support."

ERRMSG: "operator family with OID %u does not exist"

SQLSTATE: 42704

CAUSE: "System error."

ACTION: "Contact engineer to support."

ERRMSG: "text search dictionary with OID %u does not exist"

SQLSTATE: 42704

CAUSE: "System error."

ACTION: "Contact engineer to support."

ERRMSG: "text search configuration with OID %u does not exist"

SQLSTATE: 42704

CAUSE: "System error."

ACTION: "Contact engineer to support."

ERRMSG: "collation with OID %u does not exist"

SQLSTATE: 42704

CAUSE: "System error."

ACTION: "Contact engineer to support."

ERRMSG: "conversion with OID %u does not exist"

SQLSTATE: 42704

CAUSE: "System error."

ACTION: "Contact engineer to support."

ERRMSG: "extension with OID %u does not exist"

SQLSTATE: 42704

CAUSE: "System error."

ACTION: "Contact engineer to support."

ERRMSG: "synonym with OID %u does not exist"

SQLSTATE: 42704

CAUSE: "System error."

ACTION: "Contact engineer to support."

ERRMSG: "package can not create the same name with schema."

SQLSTATE: 22023

CAUSE: "Package name conflict"

ACTION: "Please rename package name"

ERRMSG: "type is not exists %s."

SQLSTATE: 22023

CAUSE: "System error."

ACTION: "Contact Huawei Engineer."

ERRMSG: "This input type is not supported for tdigest_in()"

SQLSTATE: 0A000

CAUSE: "input type is not supported"

ACTION: "Check tdigest_in syntax to obtain the supported privilege types"

ERRMSG: "Failed to apply for memory"

SQLSTATE: 53200

CAUSE: "palloc failed"

ACTION: "Check memory"

ERRMSG: "Failed to get tde info from relation '%s'."

SQLSTATE: XX005

CAUSE: "System error."

ACTION: "Contact engineer to support."

ERRMSG: "SPI_connect failed: %s"

SQLSTATE: SP001

CAUSE: "System error."

ACTION: "Analyze the error message before the error"

ERRMSG: "permission denied for terminate snapshot thread"

SQLSTATE: 42501

CAUSE: "The user does not have system admin privilege"

ACTION: "Grant system admin to user"

ERRMSG: "terminate snapshot thread failed"

SQLSTATE: OP001

CAUSE: "Execution failed due to: %s"

ACTION: "check if snapshot thread exists"

ERRMSG: "terminate snapshot thread failed"

SQLSTATE: OP001

CAUSE: "restart wdr snapshot thread timeoutor The thread did not respond to the kill signal"

ACTION: "Check the wdr snapshot thread is restarted"

ERRMSG: "set lockwait_timeout failed"

SQLSTATE: XX000

CAUSE: "System error."

ACTION: "Contact engineer to support."

ERRMSG: "permission denied for create WDR Snapshot"

SQLSTATE: 42501

CAUSE: "The user does not have system admin privilege"

ACTION: "Grant system admin to user"

ERRMSG: "WDR snapshot request can not be accepted, please retry later"

SQLSTATE: OP001

CAUSE: "wdr snapshot thread does not exist"

ACTION: "Check if wdr snapshot thread exists"

ERRMSG: "Cannot respond to WDR snapshot request"

SQLSTATE: OP001

CAUSE: "Execution failed due to: %s"

ACTION: "Check if wdr snapshot thread exists"

ERRMSG: "query(%s) can not get datum values"

SQLSTATE: 22000

CAUSE: "System error."

ACTION: "Check whether the query can be executed"

ERRMSG: "create sequence failed"

SQLSTATE: 22000

CAUSE: "System error."

ACTION: "Check if sequence can be created"

ERRMSG: "update snapshot end time stamp filled"

SQLSTATE: 22000

CAUSE: "System error."

ACTION: "Check whether the snapshot retry is successful"

ERRMSG: "query can not get datum values"

SQLSTATE: 22000

CAUSE: "System error."

ACTION: "Check whether the query can be executed"

ERRMSG: "SPI_connect failed: %s"

SQLSTATE: XX000

CAUSE: "System error."

ACTION: "Check whether the snapshot retry is successful"

ERRMSG: "query(%s) execute failed"

SQLSTATE: 22000

CAUSE: "System error."

ACTION: "Check whether the snapshot retry is successful"

ERRMSG: "clean table of snap_%s is failed"

SQLSTATE: 22000

CAUSE: "System error."

ACTION: "Check whether the snapshot retry is successful"

ERRMSG: "analyze table failed"

SQLSTATE: 22000

CAUSE: "System error."

ACTION: "Check whether the snapshot retry is successful"

ERRMSG: "insert into tables_snap_timestamp start time stamp is failed"

SQLSTATE: 22000

CAUSE: "System error."

ACTION: "Check whether the snapshot retry is successful"

ERRMSG: "insert data failed"

SQLSTATE: 22000

CAUSE: "System error."

ACTION: "Check whether the snapshot retry is successful and check whether the query can be executed"

ERRMSG: "update tables_snap_timestamp end time stamp is failed"

SQLSTATE: 22000

CAUSE: "System error."

ACTION: "Check whether the snapshot retry is successful"

ERRMSG: "clean snapshot id %lu is failed in snapshot table"

SQLSTATE: 22000

CAUSE: "System error."

ACTION: "Check whether the snapshot retry is successful and check whether the query can be executed"

ERRMSG: "clean snapshot failed"

SQLSTATE: 22000

CAUSE: "System error."

ACTION: "Check whether the snapshot retry is successful"

ERRMSG: "can not create snapshot stat table"

SQLSTATE: 22000

CAUSE: "System error."

ACTION: "Check whether the query can be executed"

ERRMSG: "create WDR snapshot data table failed"

SQLSTATE: 22000

CAUSE: "System error."

ACTION: "Check whether the query can be executed"

ERRMSG: "insert into tables_snap_timestamp start time stamp failed"

SQLSTATE: 22000

CAUSE: "System error."

ACTION: "Check whether the query can be executed"

ERRMSG: "insert into snap_%s is failed"

SQLSTATE: 22000

CAUSE: "System error."

ACTION: "Check whether the query can be executed"

ERRMSG: "update tables_snap_timestamp end time stamp failed"

SQLSTATE: 22000

CAUSE: "System error."

ACTION: "Check whether the query can be executed"

ERRMSG: "create index failed"

SQLSTATE: 22000

CAUSE: "System error."

ACTION: "Check whether the query can be executed"

ERRMSG: "analyze table, connection failed: %s"

SQLSTATE: XX000

CAUSE: "System error."

ACTION: "Check whether the snapshot retry is successful"

ERRMSG: "snapshot thread SPI_connect failed: %s"

SQLSTATE: XX000

CAUSE: "System error."

ACTION: "Check whether the snapshot retry is successful"

ERRMSG: "Distributed key column can't be transformed"

SQLSTATE: 42P10

CAUSE: "There is a risk of violating uniqueness when transforming distribution columns."

ACTION: "Change transform column."

ERRMSG: "cannot convert %s to %s"

SQLSTATE: 42804

CAUSE: "There is no conversion path in pg_cast."

ACTION: "Rewrite or cast the expression."

ERRMSG: "create matview on TDE table failed"

SQLSTATE: 0A000

CAUSE: "create materialized views is not supported on TDE table"

ACTION: "check CREATE syntax about create the materialized views"

ERRMSG: "schema name can not same as package"

SQLSTATE: 22023

CAUSE: "schema name conflict"

ACTION: "rename schema name"

ERRMSG: "Unrecognized commandType when checking read-only attribute."

SQLSTATE: XX004

CAUSE: "System error."

ACTION: "Contact Huawei Engineer."

ERRMSG: "Fail to generate subquery plan."

SQLSTATE: XX005

CAUSE: "System error."

ACTION: "Contact Huawei Engineer."

ERRMSG: "Unrecognized node type when processing qual condition."

SQLSTATE: XX004

CAUSE: "System error."

ACTION: "Contact Huawei Engineer."

ERRMSG: "Unrecognized node type when processing const parameters."

SQLSTATE: XX004

CAUSE: "System error."

ACTION: "Contact Huawei Engineer."

ERRMSG: "SELECT FOR UPDATE/SHARE is not allowed with UNION/INTERSECT/EXCEPT"

SQLSTATE: 0A000

CAUSE: "SQL uses unsupported feature."

ACTION: "Modify SQL statement according to the manual."

ERRMSG: "GROUP BY cannot be implemented."

SQLSTATE: 0A000

CAUSE: "GROUP BY uses unsupported datatypes."

ACTION: "Modify SQL statement according to the manual."

ERRMSG: "TSDB functions cannot be used if enable_tsdb is off."

SQLSTATE: D0011

CAUSE: "Functions are not loaded."

ACTION: "Turn on enable_tsdb according to manual."

ERRMSG: "Unrecognized node type when extracting index."

SQLSTATE: XX004

CAUSE: "System error."

ACTION: "Contact Huawei Engineer."

ERRMSG: "Ordering operator cannot be identified."

SQLSTATE: 42883

CAUSE: "Grouping set columns must be able to sort their inputs."

ACTION: "Modify SQL statement according to the manual."

ERRMSG: "DISTINCT cannot be implemented."

SQLSTATE: 0A000

CAUSE: "DISTINCT uses unsupported datatypes."

ACTION: "Modify SQL statement according to the manual."

ERRMSG: "Failed to locate grouping columns."

SQLSTATE: 55000

CAUSE: "System error."

ACTION: "Contact Huawei Engineer."

ERRMSG: "Resjunk output columns are not implemented."

SQLSTATE: 20000

CAUSE: "System error."

ACTION: "Contact Huawei Engineer."

ERRMSG: "PARTITION BY cannot be implemented."

SQLSTATE: 0A000

CAUSE: "PARTITION BY uses unsupported datatypes."

ACTION: "Modify SQL statement according to the manual."

ERRMSG: "ORDER BY cannot be implemented."

SQLSTATE: 0A000

CAUSE: "ORDER BY uses unsupported datatypes."

ACTION: "Modify SQL statement according to the manual."

ERRMSG: "Failed to deconstruct sort operators into partitioning/ordering operators."

SQLSTATE: D0011

CAUSE: "System error."

ACTION: "Contact Huawei Engineer."

ERRMSG: "OBS and HDFS foreign table can NOT be in the same plan."

SQLSTATE: XX008

CAUSE: "SQL uses unsupported feature."

ACTION: "Modify SQL statement according to the manual."

ERRMSG: "Pool size should not be zero"

SQLSTATE: 22012

CAUSE: "Compute pool configuration file contains error."

ACTION: "Please check the value of 'pl' in cp_client.conf."

ERRMSG: "Failed to get the runtime info from the compute pool."

SQLSTATE: 22004

CAUSE: "System error."

ACTION: "Contact Huawei Engineer."

ERRMSG: "Version is not compatible between local cluster and the compute pool."

SQLSTATE: XX008

CAUSE: "Compute pool is not installed appropriately."

ACTION: "Configure compute pool according to manual."

ERRMSG: "No optional index path is found."

SQLSTATE: 01000

CAUSE: "System error."

ACTION: "Contact Huawei Engineer."

ERRMSG: "MERGE INTO on replicated table does not yet support using distributed tables."

SQLSTATE: 0A000

CAUSE: "SQL uses unsupported feature."

ACTION: "Modify SQL statement according to the manual."

ERRMSG: "Fail to find ForeignScan node!"

SQLSTATE: P0002

CAUSE: "System error."

ACTION: "Contact Huawei Engineer."

ERRMSG: "sql advisor don't support none table, temp table, system table."

SQLSTATE: 42601

CAUSE: "sql advisor don't support none table, temp table, system table."

ACTION: "check query component"

ERRMSG: "Invalid autonomous transaction return datatypes"

SQLSTATE: P0000

CAUSE: "PL/SQL uses unsupported feature."

ACTION: "Contact Huawei Engineer."

ERRMSG: "new row for relation '%s' violates check constraint '%s'"

SQLSTATE: 23514

CAUSE: "some rows copy failed"

ACTION: "check table defination"

ERRMSG: "new row for relation '%s' violates check constraint '%s'"

SQLSTATE: 23514

CAUSE: "some rows copy failed"

ACTION: "set client_min_messages = info for more details"

ERRMSG: "get gauss home path is NULL"

SQLSTATE: XX005

CAUSE: "gauss home path not set"

ACTION: "check if $GAUSSHOME is exist"

ERRMSG: "unable to open kms_iam_info.json file"

SQLSTATE: 58P03

CAUSE: "file not exist or broken"

ACTION: "check the kms_iam_info.json file"

ERRMSG: "can not get password plaintext"

SQLSTATE: XX005

CAUSE: "file not exist or broken"

ACTION: "check the password cipher rand file"

ERRMSG: "IAM info json key is NULL"

SQLSTATE: XX005

CAUSE: "IAM info value error"

ACTION: "check tde_config kms_iam_info.json file"

ERRMSG: "get internal password is NULL"

SQLSTATE: XX005

CAUSE: "cipher rand file missing"

ACTION: "check password cipher rand file"

ERRMSG: "KMS info json key is NULL"

SQLSTATE: XX005

CAUSE: "KMS info value error"

ACTION: "check tde_config kms_iam_info.json file"

ERRMSG: "unable to get json file"

SQLSTATE: 58P03

CAUSE: "parse json file failed"

ACTION: "check the kms_iam_info.json file format"

ERRMSG: "get JSON tree is NULL"

SQLSTATE: XX005

CAUSE: "get KMS JSON tree failed"

ACTION: "check input prarmeter or config.ini file"

ERRMSG: "failed to get json tree"

SQLSTATE: XX005

CAUSE: "config.ini json tree error"

ACTION: "check input prarmeter or config.ini file"

ERRMSG: "failed to set the value of json tree"

SQLSTATE: XX005

CAUSE: "config.ini json tree error"

ACTION: "check input prarmeter or config.ini file"

ERRMSG: "http request failed"

SQLSTATE: XX005

CAUSE: "http request error"

ACTION: "check KMS or IAM connect or config parameter"

ERRMSG: "get iam token or iam agency token is NULL"

SQLSTATE: XX005

CAUSE: "connect IAM failed"

ACTION: "check if your env can connect with IAM server"

ERRMSG: "KMS dek json key is NULL"

SQLSTATE: XX005

CAUSE: "KMS return value error"

ACTION: "check KMS config paramenter"

ERRMSG: "get kms dek is NULL"

SQLSTATE: XX005

CAUSE: "connect KMS failed"

ACTION: "check if your env can connect with KMS server"

ERRMSG: "get http header is NULL"

SQLSTATE: XX005

CAUSE: "http request failed"

ACTION: "check IAM config parameter"

ERRMSG: "create KMS dek failed"

SQLSTATE: XX005

CAUSE: "KMS error"

ACTION: "check KMS connect or config parameter"

ERRMSG: "get KMS dek failed"

SQLSTATE: XX005

CAUSE: "KMS error"

ACTION: "check KMS connect or config parameter"

ERRMSG: "get KMS DEK is NULL"

SQLSTATE: XX005

CAUSE: "get KMS dek_plaintext failed"

ACTION: "check KMS network or cipher is right"

ERRMSG: "create matview with TDE failed"

SQLSTATE: 0A000

CAUSE: "TDE feature is not supported for Create materialized views"

ACTION: "check CREATE syntax about create the materialized views"

ERRMSG: "failed to add item to the index page"

SQLSTATE: XX002

CAUSE: "System error."

ACTION: "Check WARNINGS for the details."

ERRMSG: "index row size %lu exceeds maximum %lu for index '%s'"

SQLSTATE: 54000

CAUSE: "Values larger than 1/3 of a buffer page cannot be indexed."

ACTION: "Consider a function index of an MD5 hash of the value, or use full text indexing."


中国移动
China Mobile