

Mpp数据库参考指南





MPP数据库参考手册

MPP数据库系统的参考信息，包括SQL命令，系统catalog，环境变量，服务器配置参数，字符集支持，数据类型和MPP数据库扩展。

- [SQL命令参考](#)
 - [SQL 2008可选特性兼容性](#)
 - [MPP环境变量](#)
 - [保留标识符和SQL关键字](#)
 - [系统目录参考](#)
 - [gp_toolkit管理模式](#)
 - [gpperfmon 数据库](#)
 - [MPP 数据库数据类型](#)
 - [字符集支持](#)
 - [服务器配置参数](#)
 - [内置函数摘要](#)
 - [MPP MapReduce规范](#)
 - [MPP PL/pgSQL过程语言](#)
 - [MPP PL/R 语言扩展](#)
 - [MPP PL/Python语言扩展](#)
 - [MPP PL/Container语言扩展](#)
 - [MPP的PL/Java语言扩展](#)
 - [MPP的PL/Perl语言扩展](#)
 - [用于分析的MPP MADlib扩展](#)
 - [附加提供的模块](#)
-

- [MPP的Postgis扩展](#)
- [MPP的Job_schedule扩展](#)
- [MPP Partner Connector API](#)
- [后台工作进程](#)
- [MPP 特点概要](#)

SQL命令参考

MPP数据库中提供以下SQL命令：

- [SQL 语法概述](#)
 - [ABORT](#)
 - [ALTER AGGREGATE](#)
 - [ALTER COLLATION](#)
 - [ALTER CONVERSION](#)
 - [ALTER DATABASE](#)
 - [ALTER DEFAULT PRIVILEGES](#)
 - [ALTER DOMAIN](#)
 - [ALTER EXTENSION](#)
 - [ALTER EXTERNAL TABLE](#)
 - [ALTER FOREIGN DATA WRAPPER](#)
 - [ALTER FOREIGN TABLE](#)
 - [ALTER FUNCTION](#)
-

- ALTER GROUP
 - ALTER INDEX
 - ALTER LANGUAGE
 - ALTER OPERATOR
 - ALTER OPERATOR CLASS
 - ALTER OPERATOR FAMILY
 - ALTER PROTOCOL
 - ALTER RESOURCE GROUP
 - ALTER RESOURCE QUEUE
 - ALTER ROLE
 - ALTER SCHEMA
 - ALTER SEQUENCE
 - ALTER SERVER
 - ALTER TABLE
 - ALTER TABLESPACE
 - ALTER TEXT SEARCH CONFIGURATION
 - ALTER TEXT SEARCH DICTIONARY
 - ALTER TEXT SEARCH PARSER
 - ALTER TEXT SEARCH TEMPLATE
 - ALTER TYPE
 - ALTER USER
 - ALTER USER MAPPING
 - ALTER VIEW
 - ANALYZE
-

- **BEGIN**
.....
 - **CHECKPOINT**
.....
 - **CLOSE**
.....
 - **CLUSTER**
.....
 - **COMMENT**
.....
 - **COMMIT**
.....
 - **COPY**
.....
 - **CREATE AGGREGATE**
.....
 - **CREATE CAST**
.....
 - **CREATE COLLATION**
.....
 - **CREATE CONVERSION**
.....
 - **CREATE DATABASE**
.....
 - **CREATE DOMAIN**
.....
 - **CREATE EXTENSION**
.....
 - **CREATE EXTERNAL TABLE**
.....
 - **CREATE FOREIGN DATA WRAPPER**
.....
 - **CREATE FOREIGN TABLE**
.....
 - **CREATE FUNCTION**
.....
 - **CREATE GROUP**
.....
 - **CREATE INDEX**
.....
 - **CREATE LANGUAGE**
.....
 - **CREATE OPERATOR**
.....
 - **CREATE OPERATOR CLASS**
.....
 - **CREATE OPERATOR FAMILY**
.....
-

- **CREATE PROTOCOL**
.....
 - **CREATE RESOURCE GROUP**
.....
 - **CREATE RESOURCE QUEUE**
.....
 - **CREATE ROLE**
.....
 - **CREATE RULE**
.....
 - **CREATE SCHEMA**
.....
 - **CREATE SEQUENCE**
.....
 - **CREATE SERVER**
.....
 - **CREATE TABLE**
.....
 - **CREATE TABLE AS**
.....
 - **CREATE TABLESPACE**
.....
 - **CREATE TEXT SEARCH CONFIGURATION**
.....
 - **CREATE TEXT SEARCH DICTIONARY**
.....
 - **CREATE TEXT SEARCH PARSER**
.....
 - **CREATE TEXT SEARCH TEMPLATE**
.....
 - **CREATE TYPE**
.....
 - **CREATE USER**
.....
 - **CREATE USER MAPPING**
.....
 - **CREATE VIEW**
.....
 - **DEALLOCATE**
.....
 - **DECLARE**
.....
 - **DELETE**
.....
 - **DISCARD**
.....
 - **DO**
.....
-

- **DROP AGGREGATE**
.....
 - **DROP CAST**
.....
 - **DROP COLLATION**
.....
 - **DROP CONVERSION**
.....
 - **DROP DATABASE**
.....
 - **DROP DOMAIN**
.....
 - **DROP EXTENSION**
.....
 - **DROP EXTERNAL TABLE**
.....
 - **DROP FOREIGN DATA WRAPPER**
.....
 - **DROP FOREIGN TABLE**
.....
 - **DROP FUNCTION**
.....
 - **DROP GROUP**
.....
 - **DROP INDEX**
.....
 - **DROP LANGUAGE**
.....
 - **DROP OPERATOR**
.....
 - **DROP OPERATOR CLASS**
.....
 - **DROP OPERATOR FAMILY**
.....
 - **DROP OWNED**
.....
 - **DROP PROTOCOL**
.....
 - **DROP RESOURCE GROUP**
.....
 - **DROP RESOURCE QUEUE**
.....
 - **DROP ROLE**
.....
 - **DROP RULE**
.....
 - **DROP SCHEMA**
.....
-

- DROP SEQUENCE
 - DROP SERVER
 - DROP TABLE
 - DROP TABLESPACE
 - DROP TEXT SEARCH CONFIGURATION
 - DROP TEXT SEARCH DICTIONARY
 - DROP TEXT SEARCH PARSER
 - DROP TEXT SEARCH TEMPLATE
 - DROP TYPE
 - DROP USER
 - DROP USER MAPPING
 - DROP VIEW
 - ENABLE USER
 - END
 - EXECUTE
 - EXPLAIN
 - FETCH
 - GRANT
 - INSERT
 - LOAD
 - LOCK
 - MOVE
 - PREPARE
 - REASSIGN OWNED
-

- [REINDEX](#)
- [RELEASE SAVEPOINT](#)
- [RESET](#)
- [REVOKE](#)
- [ROLLBACK](#)
- [ROLLBACK TO SAVEPOINT](#)
- [SAVEPOINT](#)
- [SELECT](#)
- [SELECT INTO](#)
- [SET](#)
- [SET CONSTRAINTS](#)
- [SET ROLE](#)
- [SET SESSION AUTHORIZATION](#)
- [SET TRANSACTION](#)
- [SHOW](#)
- [START TRANSACTION](#)
- [TRUNCATE](#)
- [UPDATE](#)
- [VACUUM](#)
- [VALUES](#)

Parent topic: [MPP数据库参考手册](#)

SQL 语法概述

ABORT

放弃当前事务.

```
ABORT [WORK | TRANSACTION]
```

See [ABORT](#) for more information.

ALTER AGGREGATE

修改聚集函数的定义。

```
ALTER AGGREGATE name ( type [ , ... ] ) RENAME TO new_name  
  
ALTER AGGREGATE name ( type [ , ... ] ) OWNER TO new_owner  
  
ALTER AGGREGATE name ( type [ , ... ] ) SET SCHEMA new_schema
```

See [ALTER AGGREGATE](#) for more information.

ALTER COLLATION

修改字符集的定义。

```
ALTER COLLATION name RENAME TO new_name
```

```
ALTER COLLATION name OWNER TO new_owner
```

```
ALTER COLLATION name SET SCHEMA new_schema
```

See [ALTER COLLATION](#) for more information.

ALTER CONVERSION

更改一个转换的定义。

```
ALTER CONVERSION name RENAME TO newname
```

```
ALTER CONVERSION name OWNER TO newowner
```

```
ALTER CONVERSION name SET SCHEMA new_schema
```

See [ALTER CONVERSION](#) for more information.

ALTER DATABASE

修改数据库的属性

```

ALTER DATABASE name [ WITH CONNECTION LIMIT conlimit ]

ALTER DATABASE name RENAME TO newname

ALTER DATABASE name OWNER TO new_owner

ALTER DATABASE name SET TABLESPACE new_tablespace

ALTER DATABASE name SET parameter { TO | = } { value | DEFAULT }
ALTER DATABASE name SET parameter FROM CURRENT
ALTER DATABASE name RESET parameter
ALTER DATABASE name RESET ALL

```

See [ALTER DATABASE](#) for more information.

ALTER DEFAULT PRIVILEGES

修改默认访问权限。

```

ALTER DEFAULT PRIVILEGES
  [ FOR { ROLE | USER } target_role [, ...] ]
  [ IN SCHEMA schema_name [, ...] ]
  abbreviated_grant_or_revoke

```

where *abbreviated_grant_or_revoke* is one of:

```

GRANT { { SELECT | INSERT | UPDATE | DELETE | TRUNCATE | REFERENCES | TRIGGER
  [, ...] | ALL [ PRIVILEGES ] }
  ON TABLES
  TO { [ GROUP ] role_name | PUBLIC } [, ...] [ WITH GRANT OPTION ]

GRANT { { USAGE | SELECT | UPDATE }
  [, ...] | ALL [ PRIVILEGES ] }
  ON SEQUENCES
  TO { [ GROUP ] role_name | PUBLIC } [, ...] [ WITH GRANT OPTION ]

GRANT { EXECUTE | ALL [ PRIVILEGES ] }

```

```
ON FUNCTIONS
TO { [ GROUP ] role_name | PUBLIC } [, ...] [ WITH GRANT OPTION ]

GRANT { USAGE | ALL [ PRIVILEGES ] }
ON TYPES
TO { [ GROUP ] role_name | PUBLIC } [, ...] [ WITH GRANT OPTION ]

REVOKE [ GRANT OPTION FOR ]
{ { SELECT | INSERT | UPDATE | DELETE | TRUNCATE | REFERENCES | TRIGGER }
[, ...] | ALL [ PRIVILEGES ] }
ON TABLES
FROM { [ GROUP ] role_name | PUBLIC } [, ...]
[ CASCADE | RESTRICT ]

REVOKE [ GRANT OPTION FOR ]
{ { USAGE | SELECT | UPDATE }
[, ...] | ALL [ PRIVILEGES ] }
ON SEQUENCES
FROM { [ GROUP ] role_name | PUBLIC } [, ...]
[ CASCADE | RESTRICT ]

REVOKE [ GRANT OPTION FOR ]
{ EXECUTE | ALL [ PRIVILEGES ] }
ON FUNCTIONS
FROM { [ GROUP ] role_name | PUBLIC } [, ...]
[ CASCADE | RESTRICT ]

REVOKE [ GRANT OPTION FOR ]
{ USAGE | ALL [ PRIVILEGES ] }
ON TYPES
FROM { [ GROUP ] role_name | PUBLIC } [, ...]
[ CASCADE | RESTRICT ]
```

See [ALTER DEFAULT PRIVILEGES](#) for more information.

ALTER DOMAIN

更改现有域的定义。

```
ALTER DOMAIN name { SET DEFAULT expression | DROP DEFAULT }
```

```
ALTER DOMAIN name { SET | DROP } NOT NULL
```

```
ALTER DOMAIN name ADD domain_constraint [ NOT VALID ]
```

```
ALTER DOMAIN name DROP CONSTRAINT [ IF EXISTS ] constraint_name [RESTRICT | CASCADE]
```

```
ALTER DOMAIN name RENAME CONSTRAINT constraint_name TO new_constraint_name
```

```
ALTER DOMAIN name VALIDATE CONSTRAINT constraint_name
```

```
ALTER DOMAIN name OWNER TO new_owner
```

```
ALTER DOMAIN name RENAME TO new_name
```

```
ALTER DOMAIN name SET SCHEMA new_schema
```

See [ALTER DOMAIN](#) for more information.

ALTER EXTENSION

更改在MPP数据库中注册的扩展的定义。

```
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:

```
ACCESS METHOD object_name |
AGGREGATE aggregate_name ( aggregate_signature ) |
CAST ( source_type AS target_type ) |
COLLATION object_name |
CONVERSION object_name |
DOMAIN object_name |
EVENT TRIGGER object_name |
FOREIGN DATA WRAPPER object_name |
FOREIGN TABLE object_name |
FUNCTION function_name ( [ [ argmode ] [ argname ] argtype [ , ... ] ] ) |
MATERIALIZED VIEW object_name |
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 |
SCHEMA object_name |
SEQUENCE object_name |
SERVER object_name |
TABLE object_name |
TEXT SEARCH CONFIGURATION object_name |
TEXT SEARCH DICTIONARY object_name |
TEXT SEARCH PARSER object_name |
TEXT SEARCH TEMPLATE object_name |
TRANSFORM FOR type_name LANGUAGE lang_name |
TYPE object_name |
VIEW object_name
```

and *aggregate_signature* is:

```
* | [ argmode ] [ argname ] argtype [ , ... ] |
[ [ argmode ] [ argname ] argtype [ , ... ] ]
ORDER BY [ argmode ] [ argname ] argtype [ , ... ]
```

See [ALTER EXTENSION](#) for more information.

ALTER EXTERNAL TABLE

更改一个外部表的定义。

```
ALTER EXTERNAL TABLE name action [, ... ]
```

where *action* is one of:

```
ADD [COLUMN] new_column type  
DROP [COLUMN] column [RESTRICT|CASCADE]  
ALTER [COLUMN] column TYPE type  
OWNER TO new_owner
```

See [ALTER EXTERNAL TABLE](#) for more information.

ALTER FOREIGN DATA WRAPPER

修改一个外部数据包装的定义。

```
ALTER FOREIGN DATA WRAPPER name  
  [ HANDLER handler_function | NO HANDLER ]  
  [ VALIDATOR validator_function | NO VALIDATOR ]  
  [ OPTIONS ( [ ADD | SET | DROP ] option ['value'] [, ... ] ) ]  
  
ALTER FOREIGN DATA WRAPPER name OWNER TO new_owner  
ALTER FOREIGN DATA WRAPPER name RENAME TO new_name
```

See [ALTER FOREIGN DATA WRAPPER](#) for more information.

ALTER FOREIGN TABLE

修改外表的定义。

```
ALTER FOREIGN TABLE [ IF EXISTS ] name
    action [, ... ]
ALTER FOREIGN TABLE [ IF EXISTS ] name
    RENAME [ COLUMN ] column_name TO new_column_name
ALTER FOREIGN TABLE [ IF EXISTS ] name
    RENAME TO new_name
ALTER FOREIGN TABLE [ IF EXISTS ] name
    SET SCHEMA new_schema
```

See [ALTER FOREIGN TABLE](#) for more information.

ALTER FUNCTION

修改函数的定义。

```
ALTER FUNCTION name ( [ [argmode] [argname] argtype [, ...] ] )
    action [, ... ] [RESTRICT]

ALTER FUNCTION name ( [ [argmode] [argname] argtype [, ...] ] )
    RENAME TO new_name

ALTER FUNCTION name ( [ [argmode] [argname] argtype [, ...] ] )
    OWNER TO new_owner

ALTER FUNCTION name ( [ [argmode] [argname] argtype [, ...] ] )
    SET SCHEMA new_schema
```

See [ALTER FUNCTION](#) for more information.

ALTER GROUP

更改角色名称或成员关系。

```
ALTER GROUP groupname ADD USER username [, ... ]  
  
ALTER GROUP groupname DROP USER username [, ... ]  
  
ALTER GROUP groupname RENAME TO newname
```

See [ALTER GROUP](#) for more information.

ALTER INDEX

更改一个索引的定义。

```
ALTER INDEX [ IF EXISTS ] name RENAME TO new_name  
  
ALTER INDEX [ IF EXISTS ] name SET TABLESPACE tablespace_name  
  
ALTER INDEX [ IF EXISTS ] name SET ( storage_parameter = value [, ...] )  
  
ALTER INDEX [ IF EXISTS ] name RESET ( storage_parameter [, ...] )  
  
ALTER INDEX ALL IN TABLESPACE name [ OWNED BY role_name [, ... ] ]  
SET TABLESPACE new_tablespace [ NOWAIT ]
```

See [ALTER INDEX](#) for more information.

ALTER LANGUAGE

更改过程语言的名称。

```
ALTER LANGUAGE name RENAME TO newname  
ALTER LANGUAGE name OWNER TO new_owner
```

See [ALTER LANGUAGE](#) for more information.

ALTER OPERATOR

更改操作符的定义。

```
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
```

See [ALTER OPERATOR](#) for more information.

ALTER OPERATOR CLASS

更改一个操作符类的定义。

```
ALTER OPERATOR CLASS name USING index_method RENAME TO new_name  
  
ALTER OPERATOR CLASS name USING index_method OWNER TO new_owner  
  
ALTER OPERATOR CLASS name USING index_method SET SCHEMA new_schema
```

See [ALTER OPERATOR CLASS](#) for more information.

ALTER OPERATOR FAMILY

更改操作符族的定义。

```
ALTER OPERATOR FAMILY name USING index_method ADD
{ OPERATOR strategy_number operator_name ( op_type, op_type ) [ FOR SEARCH | FOR C
  | FUNCTION support_number [ ( op_type [ , op_type ] ) ] funcname ( argument_type [, ...] )
} [, ... ]
```

```
ALTER OPERATOR FAMILY name USING index_method DROP
{ OPERATOR strategy_number ( op_type, op_type )
  | FUNCTION support_number [ ( op_type [ , op_type ] )
} [, ... ]
```

```
ALTER OPERATOR FAMILY name USING index_method RENAME TO new_name
```

```
ALTER OPERATOR FAMILY name USING index_method OWNER TO new_owner
```

```
ALTER OPERATOR FAMILY name USING index_method SET SCHEMA new_schema
```

See [ALTER OPERATOR FAMILY](#) for more information.

ALTER PROTOCOL

更改一个协议的定义。

```
ALTER PROTOCOL name RENAME TO newname
```

```
ALTER PROTOCOL name OWNER TO newowner
```

See [ALTER PROTOCOL](#) for more information.

ALTER RESOURCE GROUP

ALTER RESOURCE GROUP

更改一个资源组的限制项。

```
ALTER RESOURCE GROUP name SET group_attribute value
```

See [ALTER RESOURCE GROUP](#) for more information.

ALTER RESOURCE QUEUE

更改资源队列的限制。

```
ALTER RESOURCE QUEUE name WITH ( queue_attribute=value [, ... ] )
```

See [ALTER RESOURCE QUEUE](#) for more information.

ALTER ROLE

更改一个数据库角色（用户或组）。

```
ALTER ROLE name [ [ WITH ] option [ ... ] ]
```

其中 *option* 可以是：

```
SUPERUSER | NOSUPERUSER
| CREATEDB | NOCREATEDB
| CREATEROLE | NOCREATEROLE
| CREATEEXTTABLE | NOCREATEEXTTABLE [ ( attribute='value' [, ...] )
  where attributes and values are:
    type='readable'|'writable'
    protocol='gpfdist'|'http'
| INHERIT | NOINHERIT
| LOGIN | NOLOGIN
| REPLICATION | NOREPLICATION
| CONNECTION LIMIT conlimit
| [ ENCRYPTED | UNENCRYPTED ] PASSWORD 'password'
| VALID UNTIL 'timestamp'
```

```
ALTER ROLE name RENAME TO new_name
```

```
ALTER ROLE { name | ALL } [ IN DATABASE database_name ] SET configuration_parameter
ALTER ROLE { name | ALL } [ IN DATABASE database_name ] SET configuration_parameter
ALTER ROLE { name | ALL } [ IN DATABASE database_name ] RESET configuration_parameter
ALTER ROLE { name | ALL } [ IN DATABASE database_name ] RESET ALL
ALTER ROLE name RESOURCE QUEUE {queue_name | NONE}
ALTER ROLE name RESOURCE GROUP {group_name | NONE}
```

See [ALTER ROLE](#) for more information.

ALTER RULE

修改一个规则的定义。

```
ALTER RULE name ON table_name RENAME TO new_name
```

See [ALTER RULE](#) for more information.

ALTER SCHEMA

更改一个模式定义。

```
ALTER SCHEMA name RENAME TO newname
```

```
ALTER SCHEMA name OWNER TO newowner
```

See [ALTER SCHEMA](#) for more information.

ALTER SEQUENCE

更改一个序列发生器的定义。

```
ALTER SEQUENCE [ IF EXISTS ] name [INCREMENT [ BY ] increment]  
  [MINVALUE minvalue | NO MINVALUE]  
  [MAXVALUE maxvalue | NO MAXVALUE]  
  [START [ WITH ] start ]  
  [RESTART [ [ WITH ] restart ] ]  
  [CACHE cache] [[ NO ] CYCLE]  
  [OWNED BY {table.column | NONE}]
```

```
ALTER SEQUENCE [ IF EXISTS ] name OWNER TO new_owner
```

```
ALTER SEQUENCE [ IF EXISTS ] name RENAME TO new_name
```

```
ALTER SEQUENCE [ IF EXISTS ] name SET SCHEMA new_schema
```

See [ALTER SEQUENCE](#) for more information.

ALTER SERVER

修改一个外部服务器的定义。

```
ALTER SERVER server_name [ VERSION 'new_version' ]  
    [ OPTIONS ( [ ADD | SET | DROP ] option ['value'] [, ... ] ) ]
```

```
ALTER SERVER server_name OWNER TO new_owner
```

```
ALTER SERVER server_name RENAME TO new_name
```

See [ALTER SERVER](#) for more information.

ALTER TABLE

更改一个表的定义。

```
ALTER TABLE [IF EXISTS] [ONLY] name  
    action [, ... ]
```

```
ALTER TABLE [IF EXISTS] [ONLY] name  
    RENAME [COLUMN] column_name TO new_column_name
```

```
ALTER TABLE [ IF EXISTS ] [ ONLY ] name  
    RENAME CONSTRAINT constraint_name TO new_constraint_name
```

```
ALTER TABLE [IF EXISTS] name  
    RENAME TO new_name
```

```
ALTER TABLE [IF EXISTS] name  
    SET SCHEMA new_schema
```

```
ALTER TABLE ALL IN TABLESPACE name [ OWNED BY role_name [, ... ] ]  
    SET TABLESPACE new_tablespace [ NOWAIT ]
```

```
ALTER TABLE [IF EXISTS] [ONLY] name SET
```

```

WITH (REORGANIZE=true|false)
| DISTRIBUTED BY ({column_name [opclass]} [, ... ] )
| DISTRIBUTED RANDOMLY
| DISTRIBUTED REPLICATED

```

ALTER TABLE *name*

```

[ ALTER PARTITION { partition_name | FOR (RANK(number))
| FOR (value) } partition_action [...] ]
partition_action

```

其中 *action* 是下列之一：

```

ADD [COLUMN] column_name data_type [ DEFAULT default_expr ]
    [column_constraint [ ... ] ]
    [ COLLATE collation ]
    [ ENCODING ( storage_directive [,...] ) ]
DROP [COLUMN] [IF EXISTS] column_name [RESTRICT | CASCADE]
ALTER [COLUMN] column_name [ SET DATA ] TYPE type [COLLATE collation] [USING expr]
ALTER [COLUMN] column_name SET DEFAULT expression
ALTER [COLUMN] column_name DROP DEFAULT
ALTER [COLUMN] column_name { SET | DROP } NOT NULL
ALTER [COLUMN] column_name SET STATISTICS integer
ALTER [COLUMN] column SET ( attribute_option = value [, ... ] )
ALTER [COLUMN] column RESET ( attribute_option [, ... ] )
ADD table_constraint [NOT VALID]
ADD table_constraint_using_index
VALIDATE CONSTRAINT constraint_name
DROP CONSTRAINT [IF EXISTS] constraint_name [RESTRICT | CASCADE]
DISABLE TRIGGER [trigger_name | ALL | USER]
ENABLE TRIGGER [trigger_name | ALL | USER]
CLUSTER ON index_name
SET WITHOUT CLUSTER
SET WITH OIDS
SET WITHOUT OIDS
SET (storage_parameter = value)
RESET (storage_parameter [, ... ])
INHERIT parent_table
NO INHERIT parent_table
OF type_name
NOT OF
OWNER TO new_owner
SET TABLESPACE new_tablespace

```

See [ALTER TABLE](#) for more information.

ALTER TABLESPACE

更改表空间的定义。

```
ALTER TABLESPACE name RENAME TO new_name

ALTER TABLESPACE name OWNER TO new_owner

ALTER TABLESPACE name SET ( tablespace_option = value [, ... ] )

ALTER TABLESPACE name RESET ( tablespace_option [, ... ] )
```

See [ALTER TABLESPACE](#) for more information.

ALTER TEXT SEARCH CONFIGURATION

更改文本搜索配置的定义。

```
ALTER TEXT SEARCH CONFIGURATION name
  ALTER MAPPING FOR token_type [, ... ] WITH dictionary_name [, ... ]
ALTER TEXT SEARCH CONFIGURATION name
  ALTER MAPPING REPLACE old_dictionary WITH new_dictionary
ALTER TEXT SEARCH CONFIGURATION name
  ALTER MAPPING FOR token_type [, ... ] REPLACE old_dictionary WITH new_dictionary
ALTER TEXT SEARCH CONFIGURATION name
  DROP MAPPING [ IF EXISTS ] FOR token_type [, ... ]
ALTER TEXT SEARCH CONFIGURATION name RENAME TO new_name
ALTER TEXT SEARCH CONFIGURATION name OWNER TO new_owner
ALTER TEXT SEARCH CONFIGURATION name SET SCHEMA new_schema
```

See [ALTER TEXT SEARCH CONFIGURATION](#) for more information.

ALTER TEXT SEARCH DICTIONARY

更改文本搜索词典的定义。

```
ALTER TEXT SEARCH DICTIONARY name (
  option [ = value ] [, ... ]
)
ALTER TEXT SEARCH DICTIONARY name RENAME TO new_name
ALTER TEXT SEARCH DICTIONARY name OWNER TO new_owner
ALTER TEXT SEARCH DICTIONARY name SET SCHEMA new_schema
```

See [ALTER TEXT SEARCH DICTIONARY](#) for more information.

ALTER TEXT SEARCH PARSER

更改文本搜索解析器的定义。

```
ALTER TEXT SEARCH PARSER name RENAME TO new_name  
ALTER TEXT SEARCH PARSER name SET SCHEMA new_schema
```

See [ALTER TEXT SEARCH PARSER](#) for more information.

ALTER TEXT SEARCH TEMPLATE

更改文本搜索模板的定义。

```
ALTER TEXT SEARCH TEMPLATE name RENAME TO new_name  
ALTER TEXT SEARCH TEMPLATE name SET SCHEMA new_schema
```

See [ALTER TEXT SEARCH TEMPLATE](#) for more information.

ALTER TYPE

更改一个数据类型的定义。

```
ALTER TYPE name action [, ... ]  
ALTER TYPE name OWNER TO new_owner  
ALTER TYPE name RENAME ATTRIBUTE attribute_name TO new_attribute_name [ CASCADE ]  
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 } ]  
ALTER TYPE name SET DEFAULT ENCODING ( storage_directive )
```

其中 *action* 是下列选项之一：

```
ADD ATTRIBUTE attribute_name data_type [ COLLATE collation ] [ CASCADE | RESTRICT ]  
DROP ATTRIBUTE [ IF EXISTS ] attribute_name [ CASCADE | RESTRICT ]  
ALTER ATTRIBUTE attribute_name [ SET DATA ] TYPE data_type [ COLLATE collation ] [ ( ... ) ]
```

See [ALTER TYPE](#) for more information.

ALTER USER

更改数据库用户（角色）的定义。

```
ALTER USER name RENAME TO newname

ALTER USER name SET config_parameter {TO | =} {value | DEFAULT}

ALTER USER name RESET config_parameter

ALTER USER name RESOURCE QUEUE {queue_name | NONE}

ALTER USER name RESOURCE GROUP {group_name | NONE}

ALTER USER name [ [WITH] option [ ... ] ]
```

See [ALTER USER](#) for more information.

ALTER USER MAPPING

更改外部服务器的用户映射的定义。

```
ALTER USER MAPPING FOR { username | USER | CURRENT_USER | PUBLIC }
    SERVER servername
    OPTIONS ( [ ADD | SET | DROP ] option ['value'] [, ... ] )
```

See [ALTER USER MAPPING](#) for more information.

ALTER VIEW

更改一个视图的定义。

```
ALTER VIEW [ IF EXISTS ] name ALTER [ COLUMN ] column_name SET DEFAULT expression

ALTER VIEW [ IF EXISTS ] name ALTER [ COLUMN ] column_name DROP DEFAULT

ALTER VIEW [ IF EXISTS ] name OWNER TO new_owner

ALTER VIEW [ IF EXISTS ] name RENAME TO new_name

ALTER VIEW [ IF EXISTS ] name SET SCHEMA new_schema

ALTER VIEW [ IF EXISTS ] name SET ( view_option_name [= view_option_value] [, ... ] )

ALTER VIEW [ IF EXISTS ] name RESET ( view_option_name [, ... ] )
```

See [ALTER VIEW](#) for more information.

ANALYZE

收集有关一个数据库的统计信息。

```
ANALYZE [VERBOSE] [ table [ ( column [, ...] ) ]]

ANALYZE [VERBOSE] { root_partition | leaf_partition } [ ( column [, ...] ) ]

ANALYZE [VERBOSE] ROOTPARTITION { ALL | root_partition [ ( column [, ...] ) ] }
```

See [ANALYZE](#) for more information.

BEGIN

BEGIN

启动事务块。

```
BEGIN [WORK | TRANSACTION] [transaction_mode]
```

See [BEGIN](#) for more information.

CHECKPOINT

强制执行事务日志检查点。

```
CHECKPOINT
```

See [CHECKPOINT](#) for more information.

CLOSE

关闭游标。

```
CLOSE cursor_name
```

See [CLOSE](#) for more information.

CLUSTER

根据索引对磁盘上的堆存储表进行物理重新排序。 在MPP数据库中不建议执行此操作。

```
CLUSTER indexname ON tablename  
CLUSTER [VERBOSE] tablename  
CLUSTER [VERBOSE]
```

See [CLUSTER](#) for more information.

COMMENT

定义或更改对象的注释。

```

COMMENT ON
{ TABLE object_name |
  COLUMN relation_name.column_name |
  AGGREGATE agg_name (agg_type [, ...]) |
  CAST (source_type AS target_type) |
  COLLATION object_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 func_name ([[argmode] [argname] argtype [, ...]]) |
  INDEX object_name |
  LARGE OBJECT large_object_oid |
  OPERATOR operator_name (left_type, right_type) |
  OPERATOR CLASS object_name USING index_method |
  [PROCEDURAL] LANGUAGE object_name |
  RESOURCE GROUP object_name |
  RESOURCE QUEUE object_name |
  ROLE object_name |
  RULE rule_name ON table_name |
  SCHEMA object_name |
  SEQUENCE object_name |
  SERVER object_name |
  TABLESPACE object_name |
  TRIGGER trigger_name ON table_name |
  TYPE object_name |
  VIEW object_name }
IS 'text'

```

See [COMMENT](#) for more information.

COMMIT

提交当前事务。

COMMIT [WORK | TRANSACTION]

See [COMMIT](#) for more information.

COPY

在文件和表之间复制数据。

```
COPY table_name [(column_name [, ...])]
FROM {'filename' | PROGRAM 'command' | STDIN}
[ [ WITH ] ( option [, ...] ) ]
[ ON SEGMENT ]

COPY { table_name [(column_name [, ...])] | (query) }
TO {'filename' | PROGRAM 'command' | STDOUT}
[ [ WITH ] ( option [, ...] ) ]
[ ON SEGMENT ]
```

See [COPY](#) for more information.

CREATE AGGREGATE

定义一个新的聚集函数

```
CREATE AGGREGATE name ( [ argmode ] [ ] arg_data_type [ , ... ] ) (
    SFUNC = sfunc,
    STYPE = state_data_type
    [ , SSPACE = state_data_size ]
    [ , FINALFUNC = ffunc ]
    [ , FINALFUNC_EXTRA ]
    [ , COMBINEFUNC = combinefunc ]
    [ , SERIALFUNC = serialfunc ]
    [ , DESERIALFUNC = deserialfunc ] )
```

```
[ , DESERIALFUNC = deserialfunc ]
[ , INITCOND = initial_condition ]
[ , MSFUNC = msfunc ]
[ , MINVFUNC = minvfunc ]
[ , MSTYPE = mstate_data_type ]
[ , MSSPACE = mstate_data_size ]
[ , MFINALFUNC = mffunc ]
[ , MFINALFUNC_EXTRA ]
[ , MINITCOND = minitial_condition ]
[ , SORTOP = sort_operator ]
)
```

```
CREATE AGGREGATE name ( [ [ argmode ] [ argname ] arg_data_type [ , ... ] ]
    ORDER BY [ argmode ] [ argname ] arg_data_type [ , ... ] ) (
    SFUNC = sfunc,
    STYPE = state_data_type
    [ , SSPACE = state_data_size ]
    [ , FINALFUNC = ffunc ]
    [ , FINALFUNC_EXTRA ]
    [ , COMBINEFUNC = combinefunc ]
    [ , SERIALFUNC = serialfunc ]
    [ , DESERIALFUNC = deserialfunc ]
    [ , INITCOND = initial_condition ]
    [ , HYPOTHETICAL ]
)
```

or the old syntax

```
CREATE AGGREGATE name (
    BASETYPE = base_type,
    SFUNC = sfunc,
    STYPE = state_data_type
    [ , SSPACE = state_data_size ]
    [ , FINALFUNC = ffunc ]
    [ , FINALFUNC_EXTRA ]
    [ , COMBINEFUNC = combinefunc ]
    [ , SERIALFUNC = serialfunc ]
    [ , DESERIALFUNC = deserialfunc ]
    [ , INITCOND = initial_condition ]
    [ , MSFUNC = msfunc ]
    [ , MINVFUNC = minvfunc ]
    [ , MSTYPE = mstate_data_type ]
    [ , MSSPACE = mstate_data_size ]
```

```
[ , MFINALFUNC = mffunc ]  
[ , MFINALFUNC_EXTRA ]  
[ , MINITCOND = initial_condition ]  
[ , SORTOP = sort_operator ]  
)
```

See [CREATE AGGREGATE](#) for more information.

CREATE CAST

定义一种新的造型。

```
CREATE CAST (sourcetype AS targettype)  
  WITH FUNCTION funcname (argtype [, ...])  
  [AS ASSIGNMENT | AS IMPLICIT]
```

```
CREATE CAST (sourcetype AS targettype)  
  WITHOUT FUNCTION  
  [AS ASSIGNMENT | AS IMPLICIT]
```

```
CREATE CAST (sourcetype AS targettype)  
  WITH INOUT  
  [AS ASSIGNMENT | AS IMPLICIT]
```

See [CREATE CAST](#) for more information.

CREATE COLLATION

使用指定的操作系统语言环境或通过复制现有的排序规则来定义新的排序规则。

```
CREATE COLLATION name (  
  [ LOCALE = locale, ]  
  [ LC_COLLATE = lc_collate, ]  
  [ LC_CTYPE = lc_ctype ] )  
  
CREATE COLLATION name FROM existing_collation
```

See [CREATE COLLATION](#) for more information.

CREATE CONVERSION

定义新的编码转换。

```
CREATE [DEFAULT] CONVERSION name FOR source_encoding TO  
  dest_encoding FROM funcname
```

See [CREATE CONVERSION](#) for more information.

CREATE DATABASE

创建一个新的数据库。

```
CREATE DATABASE name [ [WITH] [OWNER [=] user_name]  
  [TEMPLATE [=] template]  
  [ENCODING [=] encoding]  
  [LC_COLLATE [=] lc_collate]  
  [LC_CTYPE [=] lc_ctype]  
  [TABLESPACE [=] tablespace]  
  [CONNECTION LIMIT [=] conlimit ] ]
```

See [CREATE DATABASE](#) for more information.

CREATE DOMAIN

定义一个新域。

```
CREATE DOMAIN name [AS] data_type [DEFAULT expression]  
  [ COLLATE collation ]  
  [ CONSTRAINT constraint_name  
  | NOT NULL | NULL  
  | CHECK (expression) [...]]
```

See [CREATE DOMAIN](#) for more information.

CREATE EXTENSION

Registers an extension in a MPP database.

```
CREATE EXTENSION [ IF NOT EXISTS ] extension_name  
  [ WITH ] [ SCHEMA schema_name ]  
  [ VERSION version ]  
  [ FROM old_version ]  
  [ CASCADE ]
```

See [CREATE EXTENSION](#) for more information.

CREATE EXTERNAL TABLE

定义一个新的外部表。

```
CREATE [READABLE] EXTERNAL [TEMPORARY | TEMP] TABLE table_name  
  ( column_name data_type [, ...] | LIKE other_table )  
  LOCATION ('file://seghost[:port]/path/file' [, ...])
```

```

| ('gpfdist://filehost[:port]/file_pattern[#transform=trans_name]'
  [, ...]
| ('gpfdists://filehost[:port]/file_pattern[#transform=trans_name]'
  [, ...])
| ('pxf://path-to-data?PROFILE=profile_name[&SERVER=server_name][&custom-option=v
| ('s3://S3_endpoint[:port]/bucket_name/[S3_prefix] [region=S3-region] [config=config_file]'
[ON MASTER]
FORMAT 'TEXT'
  [( [HEADER]
    [DELIMITER [AS] 'delimiter' | 'OFF']
    [NULL [AS] 'null string']
    [ESCAPE [AS] 'escape' | 'OFF']
    [NEWLINE [ AS ] 'LF' | 'CR' | 'CRLF']
    [FILL MISSING FIELDS] )]
| 'CSV'
  [( [HEADER]
    [QUOTE [AS] 'quote']
    [DELIMITER [AS] 'delimiter']
    [NULL [AS] 'null string']
    [FORCE NOT NULL column [, ...]]
    [ESCAPE [AS] 'escape']
    [NEWLINE [ AS ] 'LF' | 'CR' | 'CRLF']
    [FILL MISSING FIELDS] )]
| 'CUSTOM' (Formatter=<formatter_specifications>)
[ ENCODING 'encoding' ]
[ [LOG ERRORS] SEGMENT REJECT LIMIT count
[ROWS | PERCENT] ]

```

```

CREATE [READABLE] EXTERNAL WEB [TEMPORARY | TEMP] TABLE table_name
( column_name data_type [, ...] | LIKE other_table )
  LOCATION ('http://webhost[:port]/path/file' [, ...])
| EXECUTE 'command' [ON ALL
  | MASTER
  | number_of_segments
  | HOST ['segment_hostname']
  | SEGMENT segment_id ]
FORMAT 'TEXT'
  [( [HEADER]
    [DELIMITER [AS] 'delimiter' | 'OFF']
    [NULL [AS] 'null string']
    [ESCAPE [AS] 'escape' | 'OFF']
    [NEWLINE [ AS ] 'LF' | 'CR' | 'CRLF']
    [FILL MISSING FIELDS] )]

```



```

| 'CSV'
[( [HEADER]
  [QUOTE [AS] 'quote']
  [DELIMITER [AS] 'delimiter']
  [NULL [AS] 'null string']
  [FORCE NOT NULL column [, ...]]
  [ESCAPE [AS] 'escape']
  [NEWLINE [ AS ] 'LF' | 'CR' | 'CRLF']
  [FILL MISSING FIELDS] )]
| 'CUSTOM' (Formatter=<formatter specifications>)
[ ENCODING 'encoding' ]
[ [LOG ERRORS] SEGMENT REJECT LIMIT count
[ROWS | PERCENT] ]

```

```

CREATE WRITABLE EXTERNAL [TEMPORARY | TEMP] TABLE table_name
( column_name data_type [, ...] | LIKE other_table )
LOCATION('gpfdist://outputhost[:port]/filename[#transform=trans_name]'
[, ...])
| ('gpfdists://outputhost[:port]/file_pattern[#transform=trans_name]'
[, ...])
FORMAT 'TEXT'
  [( [DELIMITER [AS] 'delimiter']
  [NULL [AS] 'null string']
  [ESCAPE [AS] 'escape' | 'OFF'] )]
| 'CSV'
  [( [QUOTE [AS] 'quote']
  [DELIMITER [AS] 'delimiter']
  [NULL [AS] 'null string']
  [FORCE QUOTE column [, ...]] | * ]
  [ESCAPE [AS] 'escape'] )]

| 'CUSTOM' (Formatter=<formatter specifications>)
[ ENCODING 'write_encoding' ]
[ DISTRIBUTED BY ({column [opclass]}, [ ... ] ) | DISTRIBUTED RANDOMLY ]

```

```

CREATE WRITABLE EXTERNAL [TEMPORARY | TEMP] TABLE table_name
( column_name data_type [, ...] | LIKE other_table )
LOCATION('s3://S3_endpoint[:port]/bucket_name/[S3_prefix] [region=S3-region] [config=cc
[ON MASTER]
FORMAT 'TEXT'
  [( [DELIMITER [AS] 'delimiter']
  [NULL [AS] 'null string']
  [ESCAPE [AS] 'escape' | 'OFF'] )]

```

```
| 'CSV'
  [( [QUOTE [AS] 'quote']
    [DELIMITER [AS] 'delimiter']
    [NULL [AS] 'null string']
    [FORCE QUOTE column [, ...]] | * ]
    [ESCAPE [AS] 'escape'] )]
```

```
CREATE WRITABLE EXTERNAL WEB [TEMPORARY | TEMP] TABLE table_name
  ( column_name data_type [, ...] | LIKE other_table )
EXECUTE 'command' [ON ALL]
FORMAT 'TEXT'
  [( [DELIMITER [AS] 'delimiter']
    [NULL [AS] 'null string']
    [ESCAPE [AS] 'escape' | 'OFF'] )]
| 'CSV'
  [( [QUOTE [AS] 'quote']
    [DELIMITER [AS] 'delimiter']
    [NULL [AS] 'null string']
    [FORCE QUOTE column [, ...]] | * ]
    [ESCAPE [AS] 'escape'] )]
| 'CUSTOM' (Formatter=<formatter specifications>)
[ ENCODING 'write_encoding' ]
[ DISTRIBUTED BY ({column [opclass]}, [ ... ] ) | DISTRIBUTED RANDOMLY ]
```

See [CREATE EXTERNAL TABLE](#) for more information.

CREATE FOREIGN DATA WRAPPER

定义一个新的外部数据包装器。

```
CREATE FOREIGN DATA WRAPPER name
  [ HANDLER handler_function | NO HANDLER ]
  [ VALIDATOR validator_function | NO VALIDATOR ]
  [ OPTIONS ( [ mpp_execute { 'master' | 'any' | 'all segments' } [, ] ] option 'value' [, ... ] ) ]
```

See [CREATE FOREIGN DATA WRAPPER](#) for more information.

CREATE FOREIGN TABLE

定义一个新的外部表。

```
CREATE FOREIGN TABLE [ IF NOT EXISTS ] table_name ( [
    column_name data_type [ OPTIONS ( option 'value' [, ... ] ) ] [ COLLATE collation ] [ column
    [, ... ]
] )
SERVER server_name
[ OPTIONS ( [ mpp_execute { 'master' | 'any' | 'all segments' } [, ] ] option 'value' [, ... ] ) ]
```

See [CREATE FOREIGN TABLE](#) for more information.

CREATE FUNCTION

定义一个新函数。

```
CREATE [OR REPLACE] FUNCTION name
( [ [ argmode ] argname ] argtype [ { DEFAULT | = } default_expr ] [, ... ] )
[ RETURNS rettype
| RETURNS TABLE ( column_name column_type [, ...] ) ]
{ LANGUAGE langname
| WINDOW
| IMMUTABLE | STABLE | VOLATILE | [NOT] LEAKPROOF
| CALLED ON NULL INPUT | RETURNS NULL ON NULL INPUT | STRICT
| [EXTERNAL] SECURITY INVOKER | [EXTERNAL] SECURITY DEFINER
| EXECUTE ON { ANY | MASTER | ALL SEGMENTS }
| COST execution_cost
| SET configuration_parameter { TO value | = value | FROM CURRENT }
| AS 'definition'
| AS 'obj_file', 'link_symbol' } ...
[ WITH ( { DESCRIBE = describe_function
    [, ...] ) ]
```

See [CREATE FUNCTION](#) for more information.

CREATE GROUP

定义一个新的数据库角色。

```
CREATE GROUP name [[WITH] option [ ... ]]
```

See [CREATE GROUP](#) for more information.

CREATE INDEX

定义一个新索引。

```
CREATE [UNIQUE] INDEX [name] ON table_name [USING method]  
  ( {column_name | (expression)} [COLLATE parameter] [opclass] [ ASC | DESC ] [ NULLS  
  [ WITH ( storage_parameter = value [, ... ] ) ] ]  
  [ TABLESPACE tablespace ]  
  [ WHERE predicate ]
```

See [CREATE INDEX](#) for more information.

CREATE LANGUAGE

定义一种新的过程语言。

```
CREATE [ OR REPLACE ] [ PROCEDURAL ] LANGUAGE name
```

```
CREATE [ OR REPLACE ] [ TRUSTED ] [ PROCEDURAL ] LANGUAGE name  
  HANDLER call_handler [ INLINE inline_handler ]  
  [ VALIDATOR valfunction ]
```

See [CREATE LANGUAGE](#) for more information.

CREATE OPERATOR

定义一个新的运算符。

```
CREATE OPERATOR name (  
  PROCEDURE = funcname  
  [, LEFTARG = lefttype] [, RIGHTARG = righttype]  
  [, COMMUTATOR = com_op] [, NEGATOR = neg_op]  
  [, RESTRICT = res_proc] [, JOIN = join_proc]  
  [, HASHES] [, MERGES] )
```

See [CREATE OPERATOR](#) for more information.

CREATE OPERATOR CLASS

定义一个新的运算符类。

```
CREATE OPERATOR CLASS name [DEFAULT] FOR TYPE data_type
  USING index_method [ FAMILY family_name ] AS
  { OPERATOR strategy_number operator_name [ ( op_type, op_type ) ] [ FOR SEARCH | FOR
  | FUNCTION support_number funcname (argument_type [, ...] )
  | STORAGE storage_type
  } [, ... ]
```

See [CREATE OPERATOR CLASS](#) for more information.

CREATE OPERATOR FAMILY

定义一个新的运算符族。

```
CREATE OPERATOR FAMILY name USING index_method
```

See [CREATE OPERATOR FAMILY](#) for more information.

CREATE PROTOCOL

注册自定义数据访问协议，该协议可以在定义MPP数据库外部表时指定。

```
CREATE [TRUSTED] PROTOCOL name (
  [readfunc='read_call_handler' ] [, writefunc='write_call_handler' ]
  [, validatorfunc='validate_handler' ] )
```

See [CREATE PROTOCOL](#) for more information.

CREATE RESOURCE GROUP

定义一个新的资源组。

```
CREATE RESOURCE GROUP name WITH (group_attribute=value [, ... ])
```

See [CREATE RESOURCE GROUP](#) for more information.

CREATE RESOURCE QUEUE

定义一个新的资源队列。

```
CREATE RESOURCE QUEUE name WITH (queue_attribute=value [, ... ])
```

See [CREATE RESOURCE QUEUE](#) for more information.

CREATE ROLE

定义一个新的数据库角色（用户或组）。

```
CREATE ROLE name [[WITH] option [ ... ]]
```

See [CREATE ROLE](#) for more information.

CREATE RULE

定义新的重写规则。

```
CREATE [OR REPLACE] RULE name AS ON event
  TO table_name [WHERE condition]
  DO [ALSO | INSTEAD] { NOTHING | command | (command; command
  ... ) }
```

See [CREATE RULE](#) for more information.

CREATE SCHEMA

定义一个新的模式。

```
CREATE SCHEMA schema_name [AUTHORIZATION username]
  [schema_element [ ... ]]

CREATE SCHEMA AUTHORIZATION rolename [schema_element [ ... ]]

CREATE SCHEMA IF NOT EXISTS schema_name [ AUTHORIZATION user_name ]

CREATE SCHEMA IF NOT EXISTS AUTHORIZATION user_name
```

See [CREATE SCHEMA](#) for more information.

CREATE SEQUENCE

定义一个新的序列生成器。

```
CREATE [TEMPORARY | TEMP] SEQUENCE name
  [INCREMENT [BY] value]
  [MINVALUE minvalue | NO MINVALUE]
  [MAXVALUE maxvalue | NO MAXVALUE]
  [START [ WITH ] start]
  [CACHE cache]
  [[NO] CYCLE]
  [OWNED BY { table.column | NONE }]
```

See [CREATE SEQUENCE](#) for more information.

CREATE SERVER

定义一个新的外部服务器。

```
CREATE SERVER server_name [ TYPE 'server_type' ] [ VERSION 'server_version' ]
  FOREIGN DATA WRAPPER fdw_name
  [ OPTIONS ( [ mpp_execute { 'master' | 'any' | 'all segments' } [, ] ] option 'value' [, ... ] ) ]
```

See [CREATE SERVER](#) for more information.

CREATE TABLE

定义一个新表。

```

CREATE [ [GLOBAL | LOCAL] {TEMPORARY | TEMP } | UNLOGGED] TABLE [IF NOT EXISTS]
    table_name (
        [ { column_name data_type [ COLLATE collation ] [column_constraint [ ... ] ]
    [ ENCODING ( storage_directive [, ...] ) ]
        | table_constraint
        | LIKE source_table [ like_option ... ] }
        | [ column_reference_storage_directive [, ...]
        [, ... ]
    ) )
[ INHERITS ( parent_table [, ... ] ) ]
[ WITH ( storage_parameter [=value] [, ... ] )
    | WITH OIDS | WITHOUT OIDS ]
[ ON COMMIT { PRESERVE ROWS | DELETE ROWS | DROP } ]
[ TABLESPACE tablespace_name ]
[ DISTRIBUTED BY (column [opclass], [ ... ] )
    | DISTRIBUTED RANDOMLY | DISTRIBUTED REPLICATED ]
[ PARTITION BY partition_type (column)
    [ SUBPARTITION BY partition_type (column) ]
    [ SUBPARTITION TEMPLATE ( template_spec ) ]
    [...]
    ( partition_spec )
    | [ SUBPARTITION BY partition_type (column) ]
    [...]
    ( partition_spec
    [ ( subpartition_spec
        [(...)]
    ) ]
    ) ]
    ) ]

```

```

CREATE [ [GLOBAL | LOCAL] {TEMPORARY | TEMP} | UNLOGGED ] TABLE [IF NOT EXISTS]
    table_name
    OF type_name [ (
        { column_name WITH OPTIONS [ column_constraint [ ... ] ]
        | table_constraint }
        [, ... ]
    ) ]
[ WITH ( storage_parameter [=value] [, ... ] ) | WITH OIDS | WITHOUT OIDS ]
[ ON COMMIT { PRESERVE ROWS | DELETE ROWS | DROP } ]
[ TABLESPACE tablespace_name ]

```

See [CREATE TABLE](#) for more information.

CREATE TABLE AS

根据查询结果定义一个新表。

```
CREATE [ [ GLOBAL | LOCAL ] { TEMPORARY | TEMP } | UNLOGGED ] TABLE table_name
  [ ( column_name [, ...] ) ]
  [ WITH ( storage_parameter [= value] [, ...] ) | WITH OIDS | WITHOUT OIDS ]
  [ ON COMMIT { PRESERVE ROWS | DELETE ROWS | DROP } ]
  [ TABLESPACE tablespace_name ]
  AS query
  [ WITH [ NO ] DATA ]
  [ DISTRIBUTED BY ( column [, ...] ) | DISTRIBUTED RANDOMLY | DISTRIBUTED REP
```

See [CREATE TABLE AS](#) for more information.

CREATE TABLESPACE

定义一个新的表空间。

```
CREATE TABLESPACE tablespace_name [OWNER username] LOCATION '/path/to/dir'
  [WITH (contentID_1='/path/to/dir1', contentID_2='/path/to/dir2' ... )]]
```

See [CREATE TABLESPACE](#) for more information.

CREATE TEXT SEARCH CONFIGURATION

定义新的文本搜索配置。

```
CREATE TEXT SEARCH CONFIGURATION name (  
  PARSER = parser_name |  
  COPY = source_config  
)
```

See [CREATE TEXT SEARCH CONFIGURATION](#) for more information.

CREATE TEXT SEARCH DICTIONARY

定义一个新的文本搜索字典。

```
CREATE TEXT SEARCH DICTIONARY name (  
  TEMPLATE = template  
  [, option = value [, ... ]]  
)
```

See [CREATE TEXT SEARCH DICTIONARY](#) for more information.

CREATE TEXT SEARCH PARSER

定义一个新的文本搜索解析器。

```
CREATE TEXT SEARCH PARSER name (  
  START = start_function ,  
  GETTOKEN = gettoken_function ,  
  END = end_function ,  
  LEXTYPES = lextypes_function  
  [, HEADLINE = headline_function ]  
)
```

See [CREATE TEXT SEARCH PARSER](#) for more information.

CREATE TEXT SEARCH TEMPLATE

定义一个新的文本搜索模板。

```
CREATE TEXT SEARCH TEMPLATE name (  
  [ INIT = init_function , ]  
  LEXIZE = lexize_function  
)
```

See [CREATE TEXT SEARCH TEMPLATE](#) for more information.

CREATE TYPE

定义新的数据类型。

```
CREATE TYPE name AS  
( attribute_name data_type [ COLLATE collation ] [, ... ] )
```

```
CREATE TYPE name AS ENUM  
( [ 'label' [, ... ] ] )
```

```
CREATE TYPE name AS RANGE (  
    SUBTYPE = subtype  
    [ , SUBTYPE_OPCLASS = subtype_operator_class ]  
    [ , COLLATION = collation ]  
    [ , CANONICAL = canonical_function ]  
    [ , SUBTYPE_DIFF = subtype_diff_function ]  
)
```

```
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 ]  
    [ , INTERNALLENGTH = {internallength | VARIABLE} ]  
    [ , PASSEDBYVALUE ]  
    [ , ALIGNMENT = alignment ]  
    [ , STORAGE = storage ]  
    [ , LIKE = like_type ]  
    [ , CATEGORY = category ]  
    [ , PREFERRED = preferred ]  
    [ , DEFAULT = default ]  
    [ , ELEMENT = element ]  
    [ , DELIMITER = delimiter ]  
    [ , COLLATABLE = collatable ]  
    [ , COMPRESSTYPE = compression_type ]  
    [ , COMPRESSLEVEL = compression_level ]  
    [ , BLOCKSIZE = blocksize ] )
```

```
CREATE TYPE name
```

See [CREATE TYPE](#) for more information.

CREATE USER

默认情况下，使用LOGIN特权定义一个新的数据库角色。

```
CREATE USER name [[WITH] option [ ... ]]
```

See [CREATE USER](#) for more information.

CREATE USER MAPPING

定义用户到外部服务器的新映射。

```
CREATE USER MAPPING FOR { username | USER | CURRENT_USER | PUBLIC }  
  SERVER servername  
  [ OPTIONS ( option 'value' [, ... ] ) ]
```

See [CREATE USER MAPPING](#) for more information.

CREATE VIEW

定义一个新的视图。

```
CREATE [OR REPLACE] [TEMP | TEMPORARY] [RECURSIVE] VIEW name  
  [ ( column_name [, ...] ) ]  
  [ WITH ( view_option_name [= view_option_value] [, ... ] ) ]  
  AS query  
  [ WITH [ CASCADED | LOCAL ] CHECK OPTION ]
```

See [CREATE VIEW](#) for more information.

DEALLOCATE

取消分配预编译语句。

```
DEALLOCATE [PREPARE] name
```

See [DEALLOCATE](#) for more information.

DECLARE

定义一个游标。

```
DECLARE name [BINARY] [INSENSITIVE] [NO SCROLL] CURSOR  
  [{WITH | WITHOUT} HOLD]  
  FOR query [FOR READ ONLY]
```

See [DECLARE](#) for more information.

DELETE

从表中删除行。

```
[ WITH [ RECURSIVE ] with_query [, ...] ]  
DELETE FROM [ONLY] table [[AS] alias]  
  [USING usinglist]  
  [WHERE condition | WHERE CURRENT OF cursor_name]  
  [RETURNING * | output_expression [[AS] output_name] [, ...]]
```

See [DELETE](#) for more information.

DISCARD

放弃会话状态。

```
DISCARD { ALL | PLANS | TEMPORARY | TEMP }
```

See [DISCARD](#) for more information.

DROP AGGREGATE

删除聚合函数。

```
DROP AGGREGATE [IF EXISTS] name ( type [, ...] ) [CASCADE | RESTRICT]
```

See [DROP AGGREGATE](#) for more information.

DO

执行匿名代码块作为临时匿名函数。

```
DO [ LANGUAGE lang_name ] code
```

See [DO](#) for more information.

DROP CAST

删除一个造型。

```
DROP CAST [IF EXISTS] (sourcetype AS targettype) [CASCADE | RESTRICT]
```

See [DROP CAST](#) for more information.

DROP COLLATION

删除以前定义的排序规则。

```
DROP COLLATION [ IF EXISTS ] name [ CASCADE | RESTRICT ]
```

See [DROP COLLATION](#) for more information.

DROP CONVERSION

删除转换。

```
DROP CONVERSION [IF EXISTS] name [CASCADE | RESTRICT]
```

See [DROP CONVERSION](#) for more information.

DROP DATABASE

删除数据库。

```
DROP DATABASE [IF EXISTS] name
```

See [DROP DATABASE](#) for more information.

DROP DOMAIN

删除域。

```
DROP DOMAIN [IF EXISTS] name [, ...] [CASCADE | RESTRICT]
```

See [DROP DOMAIN](#) for more information.

DROP EXTENSION

从MPP数据库中删除扩展。

```
DROP EXTENSION [ IF EXISTS ] name [, ...] [ CASCADE | RESTRICT ]
```

See [DROP EXTENSION](#) for more information.

DROP EXTERNAL TABLE

删除外部表定义。

```
DROP EXTERNAL [WEB] TABLE [IF EXISTS] name [CASCADE | RESTRICT]
```

See [DROP EXTERNAL TABLE](#) for more information.

DROP FOREIGN DATA WRAPPER

删除外部数据包装器。

```
DROP FOREIGN DATA WRAPPER [ IF EXISTS ] name [ CASCADE | RESTRICT ]
```

See [DROP FOREIGN DATA WRAPPER](#) for more information.

DROP FOREIGN TABLE

删除外部表。

```
DROP FOREIGN TABLE [ IF EXISTS ] name [, ...] [ CASCADE | RESTRICT ]
```

See [DROP FOREIGN TABLE](#) for more information.

DROP FUNCTION

删除函数。

```
DROP FUNCTION [IF EXISTS] name ( [ [argmode] [argname] argtype  
[, ...] ] ) [CASCADE | RESTRICT]
```

See [DROP FUNCTION](#) for more information.

DROP GROUP

删除数据库角色。

```
DROP GROUP [IF EXISTS] name [, ...]
```

See [DROP GROUP](#) for more information.

DROP INDEX

删除索引。

```
DROP INDEX [ CONCURRENTLY ] [ IF EXISTS ] name [, ...] [ CASCADE | RESTRICT ]
```

See [DROP INDEX](#) for more information.

DROP LANGUAGE

删除过程语言。

```
DROP [PROCEDURAL] LANGUAGE [IF EXISTS] name [CASCADE | RESTRICT]
```

See [DROP LANGUAGE](#) for more information.

DROP OPERATOR

删除运算符。

```
DROP OPERATOR [IF EXISTS] name ( {lefttype | NONE} ,  
  {righttype | NONE} ) [CASCADE | RESTRICT]
```

See [DROP OPERATOR](#) for more information.

DROP OPERATOR CLASS

删除运算符类。

```
DROP OPERATOR CLASS [IF EXISTS] name USING index_method [CASCADE | RESTRICT]
```

See [DROP OPERATOR CLASS](#) for more information.

DROP OPERATOR FAMILY

删除一个运算符族

```
DROP OPERATOR FAMILY [IF EXISTS] name USING index_method [CASCADE | RESTRICT]
```

See [DROP OPERATOR FAMILY](#) for more information.

DROP OWNED

删除数据库角色拥有的数据库对象。

```
DROP OWNED BY name [, ...] [CASCADE | RESTRICT]
```

See [DROP OWNED](#) for more information.

DROP PROTOCOL

从数据库中删除外部表数据访问协议。

```
DROP PROTOCOL [IF EXISTS] name
```

See [DROP PROTOCOL](#) for more information.

DROP RESOURCE GROUP

删除资源组。

```
DROP RESOURCE GROUP group_name
```

See [DROP RESOURCE GROUP](#) for more information.

DROP RESOURCE QUEUE

删除资源队列。

```
DROP RESOURCE QUEUE queue_name
```

See [DROP RESOURCE QUEUE](#) for more information.

DROP ROLE

删除数据库角色。

```
DROP ROLE [IF EXISTS] name [, ...]
```

See [DROP ROLE](#) for more information.

DROP RULE

删除重写规则。

```
DROP RULE [IF EXISTS] name ON table_name [CASCADE | RESTRICT]
```

See [DROP RULE](#) for more information.

DROP SCHEMA

删除模式。

```
DROP SCHEMA [IF EXISTS] name [, ...] [CASCADE | RESTRICT]
```

See [DROP SCHEMA](#) for more information.

DROP SEQUENCE

删除序列。

```
DROP SEQUENCE [IF EXISTS] name [, ...] [CASCADE | RESTRICT]
```

See [DROP SEQUENCE](#) for more information.

DROP SERVER

删除外部服务器描述符。

```
DROP SERVER [ IF EXISTS ] servername [ CASCADE | RESTRICT ]
```

See [DROP SERVER](#) for more information.

DROP TABLE

删除表。

```
DROP TABLE [IF EXISTS] name [, ...] [CASCADE | RESTRICT]
```

See [DROP TABLE](#) for more information.

DROP TABLESPACE

删除表空间。

```
DROP TABLESPACE [IF EXISTS] tablespacename
```

See [DROP TABLESPACE](#) for more information.

DROP TEXT SEARCH CONFIGURATION

删除文本搜索配置。

```
DROP TEXT SEARCH CONFIGURATION [ IF EXISTS ] name [ CASCADE | RESTRICT ]
```

See [DROP TEXT SEARCH CONFIGURATION](#) for more information.

DROP TEXT SEARCH DICTIONARY

删除文本搜索字典。

```
DROP TEXT SEARCH DICTIONARY [ IF EXISTS ] name [ CASCADE | RESTRICT ]
```

See [DROP TEXT SEARCH DICTIONARY](#) for more information.

DROP TEXT SEARCH PARSER

删除文本搜索解析器。

```
DROP TEXT SEARCH PARSER [ IF EXISTS ] name [ CASCADE | RESTRICT ]
```

See [DROP TEXT SEARCH PARSER](#) for more information.

DROP TEXT SEARCH TEMPLATE

删除文本搜索模板。

```
DROP TEXT SEARCH TEMPLATE [ IF EXISTS ] name [ CASCADE | RESTRICT ]
```

See [DROP TEXT SEARCH TEMPLATE](#) for more information.

DROP TYPE

删除数据类型。

```
DROP TYPE [IF EXISTS] name [, ...] [CASCADE | RESTRICT]
```

See [DROP TYPE](#) for more information.

DROP USER

删除数据库角色。

```
DROP USER [IF EXISTS] name [, ...]
```

See [DROP USER](#) for more information.

DROP USER MAPPING

删除外部服务器的用户映射。

```
DROP USER MAPPING [ IF EXISTS ] { username | USER | CURRENT_USER | PUBLIC }  
SERVER servername
```

See [DROP USER MAPPING](#) for more information.

DROP VIEW

删除视图。

```
DROP VIEW [IF EXISTS] name [, ...] [CASCADE | RESTRICT]
```

See [DROP VIEW](#) for more information.

END

提交当前事务。

```
END [WORK | TRANSACTION]
```

See [END](#) for more information.

EXECUTE

执行准备好的SQL语句。

```
EXECUTE name [ (parameter [, ...] ) ]
```

See [EXECUTE](#) for more information.

EXPLAIN

显示语句的查询计划。

```
EXPLAIN [ ( option [, ...] ) ] statement  
EXPLAIN [ANALYZE] [VERBOSE] statement
```

See [EXPLAIN](#) for more information.

FETCH

使用游标从查询中检索行。

```
FETCH [ forward_direction { FROM | IN } ]  
      cursor_name
```

See [FETCH](#) for more information.

GRANT

定义访问权限。

```

GRANT { {SELECT | INSERT | UPDATE | DELETE | REFERENCES |
TRIGGER | TRUNCATE } [, ...] | ALL [PRIVILEGES] }
    ON { [TABLE] table_name [, ...]
        | ALL TABLES IN SCHEMA schema_name [, ...] }
    TO { [ GROUP ] role_name | PUBLIC } [, ...] [ WITH GRANT OPTION ]

GRANT { { SELECT | INSERT | UPDATE | REFERENCES } ( column_name [, ...] )
    [, ...] | ALL [ PRIVILEGES ] ( column_name [, ...] ) }
    ON [ TABLE ] table_name [, ...]
    TO { role_name | PUBLIC } [, ...] [ WITH GRANT OPTION ]

GRANT { {USAGE | SELECT | UPDATE} [, ...] | ALL [PRIVILEGES] }
    ON { SEQUENCE sequence_name [, ...]
        | ALL SEQUENCES IN SCHEMA schema_name [, ...] }
    TO { [ GROUP ] role_name | PUBLIC } [, ...] [ WITH GRANT OPTION ]

GRANT { {CREATE | CONNECT | TEMPORARY | TEMP} [, ...] | ALL
[PRIVILEGES] }
    ON DATABASE database_name [, ...]
    TO { [ GROUP ] role_name | PUBLIC } [, ...] [ WITH GRANT OPTION ]

GRANT { USAGE | ALL [ PRIVILEGES ] }
    ON DOMAIN domain_name [, ...]
    TO { [ GROUP ] role_name | PUBLIC } [, ...] [ WITH GRANT OPTION ]

GRANT { USAGE | ALL [ PRIVILEGES ] }
    ON FOREIGN DATA WRAPPER fdw_name [, ...]
    TO { [ GROUP ] role_name | PUBLIC } [, ...] [ WITH GRANT OPTION ]

GRANT { USAGE | ALL [ PRIVILEGES ] }
    ON FOREIGN SERVER server_name [, ...]
    TO { [ GROUP ] role_name | PUBLIC } [, ...] [ WITH GRANT OPTION ]

GRANT { EXECUTE | ALL [PRIVILEGES] }
    ON { FUNCTION function_name ( ( [ argmode ] [ argname ] argtype [, ...]
) ) [, ...]
        | ALL FUNCTIONS IN SCHEMA schema_name [, ...] }
    TO { [ GROUP ] role_name | PUBLIC } [, ...] [ WITH GRANT OPTION ]

GRANT { USAGE | ALL [PRIVILEGES] }

```

```

GRANT { { CREATE | USAGE } [, ...] | ALL [PRIVILEGES] }
  ON LANGUAGE lang_name [, ...]
  TO { [ GROUP ] role_name | PUBLIC } [, ...] [ WITH GRANT OPTION ]

GRANT { { CREATE | USAGE } [, ...] | ALL [PRIVILEGES] }
  ON SCHEMA schema_name [, ...]
  TO { [ GROUP ] role_name | PUBLIC } [, ...] [ WITH GRANT OPTION ]

GRANT { CREATE | ALL [PRIVILEGES] }
  ON TABLESPACE tablespace_name [, ...]
  TO { [ GROUP ] role_name | PUBLIC } [, ...] [ WITH GRANT OPTION ]

GRANT { USAGE | ALL [ PRIVILEGES ] }
  ON TYPE type_name [, ...]
  TO { [ GROUP ] role_name | PUBLIC } [, ...] [ WITH GRANT OPTION ]

GRANT parent_role [, ...]
  TO member_role [, ...] [WITH ADMIN OPTION]

GRANT { SELECT | INSERT | ALL [PRIVILEGES] }
  ON PROTOCOL protocolname
  TO username

```

See [GRANT](#) for more information.

INSERT

在表中创建新行。

```

[ WITH [ RECURSIVE ] with_query [, ...] ]
INSERT INTO table [( column [, ...] )]
  {DEFAULT VALUES | VALUES ( {expression | DEFAULT} [, ...] ) [, ...] | query}
  [RETURNING * | output_expression [[AS] output_name] [, ...]]

```

See [INSERT](#) for more information.

LOAD

加载或重新加载共享库文件。

```
LOAD 'filename'
```

See [LOAD](#) for more information.

LOCK

锁表。

```
LOCK [TABLE] [ONLY] name [ * ] [, ...] [IN lockmode MODE] [NOWAIT]
```

See [LOCK](#) for more information.

MOVE

定位游标。

```
MOVE [ forward_direction [ FROM | IN ] ]  
      cursor_name
```

See [MOVE](#) for more information.

PREPARE

准备要执行的语句。

```
PREPARE name [ (datatype [, ...] ) ] AS statement
```

See [PREPARE](#) for more information.

REASSIGN OWNED

更改数据库角色拥有的数据库对象的所有权。

```
REASSIGN OWNED BY old_role [, ...] TO new_role
```

See [REASSIGN OWNED](#) for more information.

REINDEX

重建索引。

```
REINDEX {INDEX | TABLE | DATABASE | SYSTEM} name
```

See [REINDEX](#) for more information.

RELEASE SAVEPOINT

销毁先前定义的保存点。

```
RELEASE [SAVEPOINT] savepoint_name
```

See [RELEASE SAVEPOINT](#) for more information.

RESET

将系统配置参数的值恢复为默认值。

```
RESET configuration_parameter
```

```
RESET ALL
```

See [RESET](#) for more information.

REVOKE

删除访问权限。

```
REVOKE [GRANT OPTION FOR] { {SELECT | INSERT | UPDATE | DELETE  
| REFERENCES | TRIGGER | TRUNCATE } [, ...] | ALL [PRIVILEGES] }
```

```
ON { [TABLE] table_name [, ...]  
| ALL TABLES IN SCHEMA schema_name [, ...] }  
FROM { [ GROUP ] role_name | PUBLIC } [, ...]  
[CASCADE | RESTRICT]
```

```
REVOKE [ GRANT OPTION FOR ] { { SELECT | INSERT | UPDATE  
| REFERENCES } ( column_name [, ...] )  
[, ...] | ALL [ PRIVILEGES ] ( column_name [, ...] ) }  
ON [ TABLE ] table_name [, ...]  
FROM { [ GROUP ] role_name | PUBLIC } [, ...]  
[ CASCADE | RESTRICT ]
```

```
REVOKE [GRANT OPTION FOR] { {USAGE | SELECT | UPDATE} [,...]  
  | ALL [PRIVILEGES] }  
ON { SEQUENCE sequence_name [, ...]  
  | ALL SEQUENCES IN SCHEMA schema_name [, ...] }  
FROM { [ GROUP ] role_name | PUBLIC } [, ...]  
[CASCADE | RESTRICT]
```

```
REVOKE [GRANT OPTION FOR] { {CREATE | CONNECT  
  | TEMPORARY | TEMP} [, ...] | ALL [PRIVILEGES] }  
ON DATABASE database_name [, ...]  
FROM { [ GROUP ] role_name | PUBLIC } [, ...]  
[CASCADE | RESTRICT]
```

```
REVOKE [ GRANT OPTION FOR ]  
  { USAGE | ALL [ PRIVILEGES ] }  
ON DOMAIN domain_name [, ...]  
FROM { [ GROUP ] role_name | PUBLIC } [, ...]  
[ CASCADE | RESTRICT ]
```

```
REVOKE [ GRANT OPTION FOR ]  
  { USAGE | ALL [ PRIVILEGES ] }  
ON FOREIGN DATA WRAPPER fdw_name [, ...]  
FROM { [ GROUP ] role_name | PUBLIC } [, ...]  
[ CASCADE | RESTRICT ]
```

```
REVOKE [ GRANT OPTION FOR ]  
  { USAGE | ALL [ PRIVILEGES ] }  
ON FOREIGN SERVER server_name [, ...]  
FROM { [ GROUP ] role_name | PUBLIC } [, ...]  
[ CASCADE | RESTRICT ]
```

```
REVOKE [GRANT OPTION FOR] {EXECUTE | ALL [PRIVILEGES]}  
ON { FUNCTION funcname ( [[argmode] [argname] argtype  
  [, ...]] ) [, ...]  
  | ALL FUNCTIONS IN SCHEMA schema_name [, ...] }  
FROM { [ GROUP ] role_name | PUBLIC } [, ...]  
[CASCADE | RESTRICT]
```

```
REVOKE [GRANT OPTION FOR] {USAGE | ALL [PRIVILEGES]}  
ON LANGUAGE langname [, ...]  
FROM { [ GROUP ] role_name | PUBLIC } [, ...]  
[ CASCADE | RESTRICT ]
```

```
REVOKE [GRANT OPTION FOR] { {CREATE | USAGE} [, ...]  
  | ALL [PRIVILEGES] }  
ON SCHEMA schema_name [, ...]  
FROM { [ GROUP ] role_name | PUBLIC } [, ...]  
[CASCADE | RESTRICT]
```

```
REVOKE [GRANT OPTION FOR] { CREATE | ALL [PRIVILEGES] }  
ON TABLESPACE tablespacename [, ...]  
FROM { [ GROUP ] role_name | PUBLIC } [, ...]  
[CASCADE | RESTRICT]
```

```
REVOKE [ GRANT OPTION FOR ]  
  { USAGE | ALL [ PRIVILEGES ] }  
ON TYPE type_name [, ...]  
FROM { [ GROUP ] role_name | PUBLIC } [, ...]  
[ CASCADE | RESTRICT ]
```

```
REVOKE [ADMIN OPTION FOR] parent_role [, ...]  
FROM [ GROUP ] member_role [, ...]  
[CASCADE | RESTRICT]
```

See [REVOKE](#) for more information.

ROLLBACK

中止当前事务。

```
ROLLBACK [WORK | TRANSACTION]
```

See [ROLLBACK](#) for more information.

ROLLBACK TO SAVEPOINT

将当前事务回滚到保存点。

```
ROLLBACK [WORK | TRANSACTION] TO [SAVEPOINT]
      savepoint_name
```

See [ROLLBACK TO SAVEPOINT](#) for more information.

SAVEPOINT

在当前事务中定义一个新的保存点。

```
SAVEPOINT savepoint_name
```

See [SAVEPOINT](#) for more information.

SELECT

从表或视图中检索行。

```
[ WITH [ RECURSIVE1 ] with_query [, ...] ]
SELECT [ALL | DISTINCT [ON (expression [, ...])]]
    * | expression[[AS] output_name] [, ...]
[FROM from_item [, ...]]
[WHERE condition]
[GROUP BY grouping_element [, ...]]
[HAVING condition [, ...]]
[WINDOW window_name AS (window_definition) [, ...] ]
[{UNION | INTERSECT | EXCEPT} [ALL | DISTINCT] select]
[ORDER BY expression [ASC | DESC | USING operator] [NULLS {FIRST | LAST}] [, ...]]
[LIMIT {count | ALL}]
[OFFSET start [ ROW | ROWS ] ]
[FETCH { FIRST | NEXT } [ count ] { ROW | ROWS } ONLY]
[FOR {UPDATE | NO KEY UPDATE | SHARE | KEY SHARE} [OF table_name [, ...]] [NOV

TABLE { [ ONLY ] table_name [ * ] | with_query_name }
```

See [SELECT](#) for more information.

SELECT INTO

根据查询结果定义一个新表。

```
[ WITH [ RECURSIVE ] with_query [, ...] ]  
SELECT [ALL | DISTINCT [ON ( expression [, ...] )]]  
    * | expression [AS output_name] [, ...]  
    INTO [TEMPORARY | TEMP | UNLOGGED ] [TABLE] new_table  
    [FROM from_item [, ...]]  
    [WHERE condition]  
    [GROUP BY expression [, ...]]  
    [HAVING condition [, ...]]  
    [{UNION | INTERSECT | EXCEPT} [ALL | DISTINCT ] select]  
    [ORDER BY expression [ASC | DESC | USING operator] [NULLS {FIRST | LAST}] [, ...]]  
    [LIMIT {count | ALL}]  
    [OFFSET start [ ROW | ROWS ] ]  
    [FETCH { FIRST | NEXT } [ count ] { ROW | ROWS } ONLY ]  
    [FOR {UPDATE | SHARE} [OF table_name [, ...]] [NOWAIT]  
    [...]]
```

See [SELECT INTO](#) for more information.

SET

更改MPP数据库配置参数的值。

```
SET [SESSION | LOCAL] configuration_parameter {TO | =} value | 'value' | DEFAULT}  
SET [SESSION | LOCAL] TIME ZONE {timezone | LOCAL | DEFAULT}
```

See [SET](#) for more information.

SET CONSTRAINTS

设置当前事务的约束检查时间。

```
SET CONSTRAINTS { ALL | name [, ...] } { DEFERRED | IMMEDIATE }
```

See [SET CONSTRAINTS](#) for more information.

SET ROLE

设置当前会话的当前角色标识符。

```
SET [SESSION | LOCAL] ROLE rolename
```

```
SET [SESSION | LOCAL] ROLE NONE
```

```
RESET ROLE
```

See [SET ROLE](#) for more information.

SET SESSION AUTHORIZATION

设置会话角色标识符和当前会话的当前角色标识符。

```
SET [SESSION | LOCAL] SESSION AUTHORIZATION rolename
```

```
SET [SESSION | LOCAL] SESSION AUTHORIZATION DEFAULT
```

```
RESET SESSION AUTHORIZATION
```

See [SET SESSION AUTHORIZATION](#) for more information.

SET TRANSACTION

SET TRANSACTION

设置当前事务的特性。

```
SET TRANSACTION [transaction_mode] [READ ONLY | READ WRITE]

SET TRANSACTION SNAPSHOT snapshot_id

SET SESSION CHARACTERISTICS AS TRANSACTION transaction_mode
[READ ONLY | READ WRITE]
[NOT] DEFERRABLE
```

See [SET TRANSACTION](#) for more information.

SHOW

显示系统配置参数的值。

```
SHOW configuration_parameter

SHOW ALL
```

See [SHOW](#) for more information.

START TRANSACTION

启动事务块。

```
START TRANSACTION [transaction_mode] [READ WRITE | READ ONLY]
```

See [START TRANSACTION](#) for more information.

TRUNCATE

清空表的所有行。

```
TRUNCATE [TABLE] [ONLY] name [ * ] [, ...]  
[ RESTART IDENTITY | CONTINUE IDENTITY ] [CASCADE | RESTRICT]
```

See [TRUNCATE](#) for more information.

UPDATE

更新表的行。

```
[ WITH [ RECURSIVE ] with_query [, ...] ]  
UPDATE [ONLY] table [[AS] alias]  
SET {column = {expression | DEFAULT} |  
(column [, ...]) = ({expression | DEFAULT} [, ...])} [, ...]  
[FROM fromlist]  
[WHERE condition] WHERE CURRENT OF cursor_name ]
```

See [UPDATE](#) for more information.

VACUUM

垃圾收集并可选地分析数据库。

```
VACUUM [( { FULL | FREEZE | VERBOSE | ANALYZE } [, ...] )] [table [(column [, ...] )]]
```

```
VACUUM [FULL] [FREEZE] [VERBOSE] [table]
```

```
VACUUM [FULL] [FREEZE] [VERBOSE] ANALYZE  
  [table [(column [, ...] )]]
```

See [VACUUM](#) for more information.

VALUES

计算一组行。

```
VALUES ( expression [, ...] ) [, ...]  
  [ORDER BY sort_expression [ ASC | DESC | USING operator ] [, ...]  
  ]  
  [LIMIT { count | ALL } ]  
  [OFFSET start [ ROW | ROWS ] ]  
  [FETCH { FIRST | NEXT } [count] { ROW | ROWS } ONLY ]
```

See [VALUES](#) for more information.

Parent topic: [SQL Command Reference](#)

ABORT

放弃当前事务.

概要

```
ABORT [WORK | TRANSACTION]
```

Description

ABORT 回滚当前事务并使这个事务触发的所有更新操作都被丢弃掉。这个命令与SQL标准命令ROLLBACK表现相同，[ROLLBACK](#)它是由于历史原因而出现的。

Parameters

WORK
TRANSACTION
可选关键词，对Abort操作没有影响

注解

使用 COMMIT 来成功终止一个事务。

在一个事务之外发送 ABORT 命令会没有效果，但是会产生一个警告消息。

兼容性

由于历史原因，这是MPP数据库的扩展命令，与标准SQL命令 ROLLBACK等效。

另见

[BEGIN, COMMIT, ROLLBACK](#)

Parent topic: [SQL Command Reference](#)

ALTER AGGREGATE

修改聚集函数的定义。

概要

```
ALTER AGGREGATE name ( type [ , ... ] ) RENAME TO new_name
```

```
ALTER AGGREGATE name ( type [ , ... ] ) OWNER TO new_owner
```

```
ALTER AGGREGATE name ( type [ , ... ] ) SET SCHEMA new_schema
```

Description

ALTER AGGREGATE 用来修改一个聚集函数的定义

你必须是该聚集函数的所有者才能使用ALTER AGGREGATE. 要更改聚集函数的模式, 你还必须拥有新模式的 CREATE 权限。要更改聚集函数的所有者, 你还必须是新角色的直接或者成员, 而且新角色必须在聚集函数的模式上面拥有 CREATE 权限。（这些限制强制要求拥有者不能通过丢弃并重建该聚集函数来做任何不能做的事情。然而, 超级用户可以改变任何聚合函数的所有权。）

参数

name

现有聚集函数的名称（可以是限定模式）

type

聚集函数能运行的输入数据类型。要引用零参数聚合函数, 写入*代替输入数据类型的列表。

new_name

聚集函数的新名称。

new_owner

聚集函数的新所有者。

new_schema

聚集函数的新模式名

示例

将 integer 类型的聚集函数myavg 重命名为 my_average:

```
ALTER AGGREGATE myavg(integer) RENAME TO my_average;
```

将 integer 类型的聚集函数 myavg 的所有者改为 joe:

```
ALTER AGGREGATE myavg(integer) OWNER TO joe;
```

将integer 类型的聚集函数myavg 模式修改为 myschema:

```
ALTER AGGREGATE myavg(integer) SET SCHEMA myschema;
```

兼容性

SQL标准中没有ALTER AGGREGATE 语句。

另见

[CREATE AGGREGATE, DROP AGGREGATE](#)

Parent topic: [SQL Command Reference](#)

ALTER COLLATION

修改字符集的定义。

概要

```
ALTER COLLATION name RENAME TO new_name
```

```
ALTER COLLATION name OWNER TO new_owner
```

```
ALTER COLLATION name SET SCHEMA new_schema
```

参数

name

现有字符集名称。（可以是限定模式）

new_name

新的字符集名称

new_owner

新的字符集所有者。

new_schema

新的字符集模式。

描述

你必须是字符集的所有者才能使用 ALTER COLLATION。要更改字符集的所有者，你还必须是新角色的直接或者成员，而且新角色必须在字符集的模式上面拥有 CREATE 的权限。（这些限制强制要求拥有者不能通过丢弃并重建该字符集来做任何不能做的事情。然而，超级用户可以改变任何字符集的所有权。）

示例

将字符集名从 *de_DE* 修改为 *german*:

```
ALTER COLLATION "de_DE" RENAME TO german;
```

将字符集 *en_US* 的所有者修改为 *joe*:

```
ALTER COLLATION "en_US" OWNER TO joe;
```

兼容性

SQL标准中没有 ALTER COLLATION 语法。

另见

[CREATE COLLATION](#), [DROP COLLATION](#)

Parent topic: [SQL Command Reference](#)

ALTER CONVERSION

更改一个转换的定义。

概要

```
ALTER CONVERSION name RENAME TO newname
```

```
ALTER CONVERSION name OWNER TO newowner
```

```
ALTER CONVERSION name SET SCHEMA new_schema
```

描述

ALTER CONVERSION 修改一个转换的定义

你必须是转换的所有者才能使用ALTER CONVERSION。要更改转换的所有者，你还必须是新角色的直接或者成员，而且新角色必须在转换的模式上面拥有CREATE 的权限。（这些限制强制要求拥有者不能通过丢弃并重建该转换来做任何不能做的事情。然而，超级用户可以改变任何转换的所有权。）

参数

name

现有转换的名称。（可以是限定模式）

newname

新的转换名称。

newowner

转换的新所有者。

new_schema

转换的新模式

示例

将转换iso_8859_1_to_utf8重命名为 latin1_to_unicode:

```
ALTER CONVERSION iso_8859_1_to_utf8 RENAME TO
latin1_to_unicode;
```

将转换iso_8859_1_to_utf8的所有者修改为 joe:

```
ALTER CONVERSION iso_8859_1_to_utf8 OWNER TO joe;
```

兼容性

SQL标准中没有 ALTER CONVERSION 语句。

另见

[CREATE CONVERSION](#), [DROP CONVERSION](#)

Parent topic: [SQL Command Reference](#)

ALTER DATABASE

修改数据库的属性

概要

```
ALTER DATABASE name [ WITH CONNECTION LIMIT connlimit ]  
  
ALTER DATABASE name RENAME TO newname  
  
ALTER DATABASE name OWNER TO new_owner  
  
ALTER DATABASE name SET TABLESPACE new_tablespace  
  
ALTER DATABASE name SET parameter { TO | = } { value | DEFAULT }  
ALTER DATABASE name SET parameter FROM CURRENT  
ALTER DATABASE name RESET parameter  
ALTER DATABASE name RESET ALL
```

描述

ALTER DATABASE 修改一个数据库的属性。

第一个语句修改数据库的连接数。只有数据的所有者或者超级用户可以更改此设置。

第二个语句修改数据库的名字。只有数据库的所有者或者超级用户可以重命名数据库;非超级用户必须拥有 CREATEDB 权限。不能修改当前数据库的名字，连接到其他的数据库先。

第三个聚集修改数据库的所有者。要修改所有者，你必须是新角色的直接或者间接的成员, 而且必须有 CREATEDB 的权限。（超级用户可以自动拥有所有的权限。）

第四个语句修改数据库的默认表空间。只有数据库的所有者或者超级用户可以修改默认表空，你还需要在新的表空间上有创建的权限。这个命令会将这个数据库所有在默认表空间的表和索引都移到新的表空间。注意，不在默认表空间的表和索引不受影响。

剩下的语句是用来修改MPP数据库会话级别的参数默认值。当一个会话在数据库开启后，这个会话提供的参数值将成为默认参数值。数据库在配置文件（`postgresql.conf`）配置的默认值将被覆盖。只有数据库所有者或者超级用户才能修改会话默认值。某些参数不能用这种方式设置，或者只能被超级用户设置。

Parameters

name

将要被修改属性的数据库名称。

conlimit

最大并发连接数。缺省值-1表示没有限制

parameter value

将指定配置参数的数据库的会话默认值设置为给定值。如果这个值是 DEFAULT，或者是等效的 RESET，数据库的特定设置会被删除，因此系统范围的默认设置将在新会话中继承。用RESET ALL来清除所有的数据库设置，参见 [服务器配置参数](#) 来获取更多关于用户级别的参数设置。

newname

新的数据库名。

new_owner

新的数据库所有者。

new_tablespace

新的数据库默认表空间。

注意

还可以为特定角色（用户）而不是数据库设置配置参数会话默认值。如果存在冲突，角色特定的设置将覆盖数据库特定的设置。 参见ALTER ROLE.

示例

为数据库mydatabase 设置默认的模式搜索路径:

```
ALTER DATABASE mydatabase SET search_path TO myschema,  
public, pg_catalog;
```

兼容性

ALTER DATABASE 语句是MPP的扩展语句。

另见

[CREATE DATABASE](#), [DROP DATABASE](#), [SET](#), [CREATE TABLESPACE](#)

Parent topic: [SQL Command Reference](#)

ALTER DEFAULT PRIVILEGES

修改默认访问权限。

概要

```
ALTER DEFAULT PRIVILEGES
```

```
  [ FOR { ROLE | USER } target_role [, ...] ]
```

```
  [ IN SCHEMA schema_name [, ...] ]
```

```
  abbreviated_grant_or_revoke
```

where *abbreviated_grant_or_revoke* is one of:

```
GRANT { { SELECT | INSERT | UPDATE | DELETE | TRUNCATE | REFERENCES | TRIGGER
```

```
  [, ...] | ALL [ PRIVILEGES ] }
```

```
ON TABLES
```

```
TO { [ GROUP ] role_name | PUBLIC } [, ...] [ WITH GRANT OPTION ]
```

```
GRANT { { USAGE | SELECT | UPDATE }
```

```
  [, ...] | ALL [ PRIVILEGES ] }
```

```
ON SEQUENCES
```

```
TO { [ GROUP ] role_name | PUBLIC } [, ...] [ WITH GRANT OPTION ]
```

```
GRANT { EXECUTE | ALL [ PRIVILEGES ] }
```

```
ON FUNCTIONS
```

```
TO { [ GROUP ] role_name | PUBLIC } [, ...] [ WITH GRANT OPTION ]
```

```
GRANT { USAGE | ALL [ PRIVILEGES ] }
```

```
ON TYPES
```

```
TO { [ GROUP ] role_name | PUBLIC } [, ...] [ WITH GRANT OPTION ]
```

```
REVOKE [ GRANT OPTION FOR ]
```

```
  { { SELECT | INSERT | UPDATE | DELETE | TRUNCATE | REFERENCES | TRIGGER }
```

```
  [, ...] | ALL [ PRIVILEGES ] }
```

```
ON TABLES
```

```
FROM { [ GROUP ] role_name | PUBLIC } [, ...]
```

```
[ CASCADE | RESTRICT ]
```

```
REVOKE [ GRANT OPTION FOR ]
```

```
  { { USAGE | SELECT | UPDATE }
```

```
  [, ...] | ALL [ PRIVILEGES ] }
```

```
ON SEQUENCES
FROM { [ GROUP ] role_name | PUBLIC } [, ...]
[ CASCADE | RESTRICT ]

REVOKE [ GRANT OPTION FOR ]
{ EXECUTE | ALL [ PRIVILEGES ] }
ON FUNCTIONS
FROM { [ GROUP ] role_name | PUBLIC } [, ...]
[ CASCADE | RESTRICT ]

REVOKE [ GRANT OPTION FOR ]
{ USAGE | ALL [ PRIVILEGES ] }
ON TYPES
FROM { [ GROUP ] role_name | PUBLIC } [, ...]
[ CASCADE | RESTRICT ]
```

描述

ALTER DEFAULT PRIVILEGES允许你为将来会创建的对象赋予 权限。（不会影响那些已经赋予存在对象的权限。）目前只有表（包括视图和 外键表）、序列、函数、类型的权限可以被修改。

你只可以修改那些你自己或者你所在角色会创建的对象权限。这些权限既可以在全局范围内设置（例如：在当前数据库创建的所有对象。），也可以为指定模式下的对象设置。每个模式指定的默认权限将添加到特定对象类型的全局默认权限。

如[GRANT](#)中介绍的，任何对象的默认权限通常都会被授予所有可授予的权限，也可能被授予一些PUBLIC 权限。不管怎样，这些都可以通过ALTER DEFAULT PRIVILEGES 来修改全局默认权限来设置。

参数

target_role

当前的角色是一个成员时，那么target_role为要修改的角色名，如果FOR ROLE 被省略，那么会将当前的角色当为目标角色。

schema_name

模式名，如果指定了模式名，那么之后在这个模式下面创建的所有对象默认的权限都会被修改。如果 IN SCHEMA被省略，那么全局权限会被修改。

role_name

授予或撤消权限的现有角色的名称。这个参数和所有在 *abbreviated_grant_or_revoke* 中的参数，按照[GRANT](#) 或 [REVOKE](#)描述中 进行操作，除非这个是为整个对象类而不是特定的命名对象设置权限。

注意

用 [psql](#)的 \ddp 命令来获取有关默认权限的现有分配信息，权限值的含义与在 [GRANT](#) 中\dp命令的解释相同。

如果你想删除一个被修改了默认权限的角色，必须撤消其默认权限的更改，或使用DROP OWNED BY 删除该角色的默认权限条目。

示例

为随后在myschema中创建的所有表（和视图）的每个人授予SELECT权限，并允许角色 webuser 也插入到它们中：

```
ALTER DEFAULT PRIVILEGES IN SCHEMA myschema GRANT SELECT ON TABLES TO F
ALTER DEFAULT PRIVILEGES IN SCHEMA myschema GRANT INSERT ON TABLES TO w
```

撤销上面的操作，那样随后创建的表不会拥有比正常权限更多的权限：

```
ALTER DEFAULT PRIVILEGES IN SCHEMA myschema REVOKE SELECT ON TABLES FROM  
ALTER DEFAULT PRIVILEGES IN SCHEMA myschema REVOKE INSERT ON TABLES FROM
```

将所有在admin角色上随后创建的函数上删除正常在函数上赋予的公有的执行权限：

```
ALTER DEFAULT PRIVILEGES FOR ROLE admin REVOKE EXECUTE ON FUNCTIONS FROM
```

兼容性

在标准SQL语句中没有ALTER DEFAULT PRIVILEGES语句。

另见

[GRANT](#), [REVOKE](#)

Parent topic: [SQL Command Reference](#)

ALTER DOMAIN

更改现有域的定义。

概要

```
ALTER DOMAIN name { SET DEFAULT expression | DROP DEFAULT }
```

```
ALTER DOMAIN name { SET | DROP } NOT NULL
```

```
ALTER DOMAIN name ADD domain_constraint [ NOT VALID ]
```

```
ALTER DOMAIN name DROP CONSTRAINT [ IF EXISTS ] constraint_name [RESTRICT | CASCADE]
```

```
ALTER DOMAIN name RENAME CONSTRAINT constraint_name TO new_constraint_name
```

```
ALTER DOMAIN name VALIDATE CONSTRAINT constraint_name
```

```
ALTER DOMAIN name OWNER TO new_owner
```

```
ALTER DOMAIN name RENAME TO new_name
```

```
ALTER DOMAIN name SET SCHEMA new_schema
```

描述

ALTER DOMAIN更改一个现有域的定义。 有几种形式：

- **SET/DROP DEFAULT** — 这些形式设置或删除域的默认值。默认值仅适用于后续的INSERT 命令。它们不影响使用域的表中已经存在的行。
- **SET/DROP NOT NULL** — 这些形式会改变域是否被标记为允许NULL 值或者拒绝 NULL 值。用户只能 SET NOT NULL 当使用域的列不包含空值时。
- **ADD *domain_constraint* [NOT VALID]** — 这种形式使用和 CREATE DOMAIN相同的语法为域增加一个新的约束。如果一个新的约束被添加到域中，所有在这个域中的列都会根据新添加的约束重新检查，这些检查可以用NOT VALID 选项来限制; 这些约束可以被随后的 ALTER DOMAIN ... VALIDATE CONSTRAINT来使之可用，新插入或这更新的行会永远根据所有

的约束最检查，即使那些被标记为NOT VALID的约束，NOT VALID 只能定义在 CHECK 的约束上。

- **DROP CONSTRAINT [IF EXISTS]** — 此形式删除域的约束。如果提供了 IF EXISTS 选项而要删除的约束不存在，语句只会抛出一个提醒而不会抛出错误。
- **RENAME CONSTRAINT** — 此形式改稿一个现有域的约束的名字
- **VALIDATE CONSTRAINT** — 此形式会验证之前被标记为 NOT VALID的约束，这样会验证在这个约束中的列的所有数据
- **OWNER** — This form changes the owner of the domain to the specified user.
- **RENAME** — 此形式将域的所有者更改为指定的用户。
- **SET SCHEMA** — 此形式更改域的模式。与域相关联的任何约束也被移动到新的模式中。

用户必须拥有域才能ALTER DOMAIN. 要更改域的模式，用户还必须对新模式具有 CREATE 特权 要更改所有者，用户还必须是新拥有角色的直接或间接成员，并且该角色必须对该域的模式具有 CREATE特权。这些限制强制修改拥有者不能做一些通过删除和重建域做不到的事情。不过，一个超级用户怎么都能更改任何域的所有权。）。。

参数

name

要更改的现有域的名称（可选方案限定。

domain_constraint

域的新域约束。

constraint_name

要修改护着删除的约束名

NOT VALID

不检查在约束中定义的已存在的数据。

CASCADE

自动删除依赖于此约束的所有对象。

RESTRICT

如果有任何依赖对象，拒绝删除约束。这是默认行为。

new_name

新的域名。

new_constraint_name

新的约束名。

new_owner

域的新所有者的用户名。

new_schema

域的新模式。

示例

添加NOT NULL 约束到一个域：

```
ALTER DOMAIN zipcode SET NOT NULL;
```

从一个域中移除NOT NULL约束：

```
ALTER DOMAIN zipcode DROP NOT NULL;
```

向域添加检查约束：

```
ALTER DOMAIN zipcode ADD CONSTRAINT zipchk CHECK (char_length(VALUE) = 5);
```

从域中删除检查约束：

```
ALTER DOMAIN zipcode DROP CONSTRAINT zipchk;
```

在一个域中从命名一个约束

```
ALTER DOMAIN zipcode RENAME CONSTRAINT zipchk TO zip_check;
```

将域移动到不同的模式：

```
ALTER DOMAIN zipcode SET SCHEMA customers;
```

兼容性

ALTER DOMAIN 符合SQL标准，除了 OWNER, RENAME, SET SCHEMA, and VALIDATE CONSTRAINT 变形，这些是MPP Database的扩展功能。ADD CONSTRAINT中的NOT VALID变形也是MPP Database的扩展功能。

另见

[CREATE DOMAIN](#), [DROP DOMAIN](#)

Parent topic: [SQL Command Reference](#)

ALTER EXTENSION

更改在MPP数据库中注册的扩展的定义。

概要

```

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:

```

ACCESS METHOD object_name |
AGGREGATE aggregate_name ( aggregate_signature ) |
CAST ( source_type AS target_type ) |
COLLATION object_name |
CONVERSION object_name |
DOMAIN object_name |
EVENT TRIGGER object_name |
FOREIGN DATA WRAPPER object_name |
FOREIGN TABLE object_name |
FUNCTION function_name ( [ [ argmode ] [ argname ] argtype [ , ... ] ] ) |
MATERIALIZED VIEW object_name |
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 |
SCHEMA object_name |
SEQUENCE object_name |
SERVER object_name |
TABLE object_name |
TEXT SEARCH CONFIGURATION object_name |
TEXT SEARCH DICTIONARY object_name |
TEXT SEARCH PARSER object_name |
TEXT SEARCH TEMPLATE object_name |
TRANSFORM FOR type_name LANGUAGE lang_name |
TYPE object_name |
VIEW object_name

```

and *aggregate_signature* is:

```

* | [ argmode ] [ argname ] argtype [ , ... ] |
[ [ argmode ] [ argname ] argtype [ , ... ] ]
ORDER BY [ argmode ] [ argname ] argtype [ , ... ]

```


描述

ALTER EXTENSION 更改已安装扩展的定义。 有几种子形式:

UPDATE

此形式将扩展更新到一个较新版本。该扩展必须提供一个合适的更新脚本（或一系列脚本），可以将当前安装的版本修改为所要求的版本。

SET SCHEMA

这种形式将扩展对象移动到另一个模式中。 扩展名必须是 *可重定位*。

ADD *member_object*

这种形式将一个现有对象添加到该扩展中。这在扩展更新脚本中很有用。该对象后续将被当作该扩展的一个成员。尤其是该对象只有通过删除扩展才能删除。

DROP *member_object*

这种形式从扩展中删除一个成员对象。这这主要对扩展更新脚本有用。只有 撤销该对象与其扩展之间的关联后才能删除该对象。

参阅 [将相关对象打包到扩展中](#) 获取更多关于这些操作的信息。

用户必须对扩展有所有权才可以使用ALTER EXTENSION。 ADD 和 DROP 形式也要求对需要增加或者删除的对象有所有权。

参数

name

需要安装的扩展名

new_version

新版本的扩展。*new_version*以是标识符或字符串文字。如果未指定，该命令将尝试更新到扩展控制文件中的默认版本。

new_schema

扩展的新模式。

object_name

aggregate_name

function_name

operator_name

要从该扩展增加或者移除的对象的名称。表、聚集、域、外部表、函数、操作符、操作符类、操作符族、序列、文本搜索对象、类型和视图的名称可以被方案限定。

source_type

需要转换的源数据类型的名称。

target_type

需要转换的目标数据类型的名称。

argmode

函数或聚集参数的模型：IN, OUT, INOUT, 或者 VARIADIC。默认值是IN.

这个命令会忽略掉OUT 参数。只需要输入参数才决定标识一个函数。这足够列出 IN, INOUT和 VARIADIC 参数。

argname

函数或聚集参数的名字。

这个命令会忽略掉参数的名字，因为参数的数据类型已经决定标识一个函数了。

argtype

函数或者聚集参数的数据类型。

left_type

right_type

操作符参数的数据类型（可以是方案限定）。指定 NONE 对于一个前缀或后缀操作符的缺失的参数。

PROCEDURAL

Z这是一个噪声词。

type_name

该转换的数据类型的名称。

lang_name

该转换的语言的名称。

示例

将hstore更新到2.0版本：

```
ALTER EXTENSION hstore UPDATE TO '2.0';
```

将hstore扩展的模式更改为 utils:

```
ALTER EXTENSION hstore SET SCHEMA utils;
```

将一个已存在的函数添加到hstore 扩展：

```
ALTER EXTENSION hstore ADD FUNCTION populate_record(anyelement, hstore);
```

兼容性

ALTER EXTENSION是一个MPP数据库的扩展

另见

[CREATE EXTENSION](#), [DROP EXTENSION](#)

Parent topic: [SQL Command Reference](#)

ALTER EXTERNAL TABLE

更改一个外部表的定义。

概要

```
ALTER EXTERNAL TABLE name action [, ... ]
```

where *action* is one of:

```
ADD [COLUMN] new_column type  
DROP [COLUMN] column [RESTRICT|CASCADE]  
ALTER [COLUMN] column TYPE type  
OWNER TO new_owner
```

描述

ALTER EXTERNAL TABLE改变了一个已存在的外部表的定义。 下面这些是ALTER EXTERNAL TABLE支持的动作。

- **ADD COLUMN** — 添加一个新列到外部表的定义。
- **DROP COLUMN** — 从一个外部表定义中删除一列。如果你删掉只读外部表的列，这只会MPP的表定义中修改。CASCADE 关键字是用来当一些其他表依赖于这些列时使用，比如说一个视图依赖于这列。
- **ALTER COLUMN TYPE** — 改变一个表列的数据类型。
- **OWNER** — 将一个外部表的所有者改为指定用户。

用[ALTER TABLE](#) 来在一个外部表中实现这些操作。

- 设置（修改）表模式
- 重命名表
- 重命名表中的列

你必须时外部表的所有者才能使用ALTER EXTERNAL TABLE或者 ALTER TABLE。要改变一个外部表的模式，你还必须在新模式上有CREATE的权限。要更改所有者，用户还必须是新拥有角色的直接或间接成员，该角色必须对外部表的模式具有 CREATE特权。超级用户自动拥有这些权限。

用ALTER EXTERNAL TABLE或者 ALTER TABLE 修改外部表的定义不会影响外部的数据。

ALTER EXTERNAL TABLE 和 ALTER TABLE命令 不会修改外部表的类型（读、写、网页），表的FORMAT信息，或者 外部数据的位置。要修改这些信息，你必须删除然后重建外部表的定义。

参数

name

要修改的现有外部表定义的名称（可以是方案限定）。

column

现有列的名称。

new_column

新列的名称

type

新列的数据类型，或者现有列的新数据类型。

new_owner

外部表的新所有者角色名。

CASCADE

自动删除那些依赖于要删除列的对象，比如说依赖于要删除列的视图。

RESTRICT

拒绝删除那些有依赖对象的列，这是默认的设置。

示例

往外部表的定义中添加新列：

```
ALTER EXTERNAL TABLE ext_expenses ADD COLUMN manager text;
```

改变外部表的所有者：

```
ALTER EXTERNAL TABLE ext_data OWNER TO jojo;
```

改变外部表的数据类型：

```
ALTER EXTERNAL TABLE ext_leads ALTER COLUMN acct_code TYPE integer
```

兼容性

ALTER EXTERNAL TABLE是MPP数据的扩展。标准SQL语句或者PostgreSQL中没有ALTER EXTERNAL TABLE语句。

另见

[CREATE EXTERNAL TABLE](#), [DROP EXTERNAL TABLE](#), [ALTER TABLE](#)

Parent topic: [SQL Command Reference](#)

ALTER FOREIGN DATA WRAPPER

修改一个外部数据包装的定义。

概要

```
ALTER FOREIGN DATA WRAPPER name
  [ HANDLER handler_function | NO HANDLER ]
  [ VALIDATOR validator_function | NO VALIDATOR ]
  [ OPTIONS ( [ ADD | SET | DROP ] option ['value'] [, ... ] ) ]

ALTER FOREIGN DATA WRAPPER name OWNER TO new_owner
ALTER FOREIGN DATA WRAPPER name RENAME TO new_name
```

描述

ALTER FOREIGN DATA WRAPPER修改一个外部数据包装的定义。第一个语句修改外部数据包装的支持函数或者一个一般性的选项。MPP 数据库要求只要有一个选项。第二和第三的语句修改外部数据包装的所有者或者名字。

只有超级用户才可以修改外部数据包装。另外，只有超级用户才可以拥有外部数据包装。

参数

name

外部数据包装的名字。

HANDLER *handler_function*

为外部数据包装提供一个新的处理函数。

NO HANDLER

指定外部数据包装不再拥有处理函数。

Note: 你不能访问没有处理的外部数据包装的外部表。

VALIDATOR *validator_function*

为外部数据包装提供一个新的验证函数。

当你修改验证函数时，外部数据包装、服务器、用户映射 可能会变成不可用。在修改外部数据包装前你必须确保这些选项是正确的。注意使用新验证者时，MPP数据库会检查在ALTER FOREIGN DATA WRAPPER 命令中用到的所有选项。

NO VALIDATOR

指定外部数据包装不再拥有验证函数。

OPTIONS ([ADD | SET | DROP] *option* ['*value*'] [, ...])

修改外部数据包装的选项。ADD, SET和DROP 指定了操作的动作。如果没有明确指出操作，默认的操作是ADD。选项的名字必须是唯一的。如果使用了验证函数，MPP会验证提供的选项名字和值。

OWNER TO *new_owner*

指定外部数据包装的新所有者，只有超级用户可以拥有外部数据包装。

RENAME TO *new_name*

指定外部数据包装的新名字。

示例

修改外部数据包装名为dbi的定义，增加一个新选项foo，然后移除名为bar 的选项：

```
ALTER FOREIGN DATA WRAPPER dbi OPTIONS (ADD foo '1', DROP 'bar');
```

修改外部数据包装名为dbi的验证函数为 bob.myvalidator :

```
ALTER FOREIGN DATA WRAPPER dbi VALIDATOR bob.myvalidator;
```

兼容性

ALTER FOREIGN DATA WRAPPER 适配ISO/IEC 9075-9 (SQL/MED) , 但是HANDLER、 VALIDATOR、 OWNER TO和 RENAME TO 选项是MPP数据库的扩展。

另见

[CREATE FOREIGN DATA WRAPPER](#), [DROP FOREIGN DATA WRAPPER](#)

Parent topic: [SQL Command Reference](#)

ALTER FOREIGN TABLE

修改外表的定义。

概要

```
ALTER FOREIGN TABLE [ IF EXISTS ] name
    action [, ... ]
ALTER FOREIGN TABLE [ IF EXISTS ] name
    RENAME [ COLUMN ] column_name TO new_column_name
ALTER FOREIGN TABLE [ IF EXISTS ] name
    RENAME TO new_name
ALTER FOREIGN TABLE [ IF EXISTS ] name
    SET SCHEMA new_schema
```

*action*是下列中的一个：

```
ADD [ COLUMN ] column_name column_type [ COLLATE collation ] [ column_constraint [ .
DROP [ COLUMN ] [ IF EXISTS ] column_name [ RESTRICT | CASCADE ]
ALTER [ COLUMN ] column_name [ SET DATA ] TYPE data_type
ALTER [ COLUMN ] column_name SET DEFAULT expression
ALTER [ COLUMN ] column_name DROP DEFAULT
ALTER [ COLUMN ] column_name { SET | DROP } NOT NULL
ALTER [ COLUMN ] column_name SET STATISTICS integer
ALTER [ COLUMN ] column_name SET ( attribute_option = value [, ... ] )
ALTER [ COLUMN ] column_name RESET ( attribute_option [, ... ] )
ALTER [ COLUMN ] column_name OPTIONS ( [ ADD | SET | DROP ] option ['value'] [, ... ] )
DISABLE TRIGGER [ trigger_name | ALL | USER ]
ENABLE TRIGGER [ trigger_name | ALL | USER ]
ENABLE REPLICA TRIGGER trigger_name
ENABLE ALWAYS TRIGGER trigger_name
OWNER TO new_owner
OPTIONS ( [ ADD | SET | DROP ] option ['value'] [, ... ] )
```

描述

ALTER FOREIGN TABLE 修改一个已存在的外表的定义，命令有以下的几种形式：

ADD COLUMN

这种方式往外表中新增一列，和[CREATE FOREIGN TABLE](#)使用相同的语法。这个不像往通常的表中新加一列，它其实在存储层面 不做任何操作，仅仅只是往现在访问的外表中定义了一些新列。

DROP COLUMN [IF EXISTS]

这种形式从外表中删除一列，如果有表之外的其他对象，比如说视图依赖于此列，必须指定 **CASCADE**关键字。如果指定了**IF EXISTS**关键字而这个列不存在，MPP数据库只会产生一个提醒而不是抛出错误。

IF EXISTS

如果指定了**IF EXISTS**关键字而外表不存在，MPP数据库只会产生一个提醒而不是抛出错误。

SET DATA TYPE

这种形式修改外表中一列的类型。

SET/DROP DEFAULT

这种形式设置或者删除了列的默认值，默认值只会应用在随后的**INSERT**或者 **UPDATE**命令；不会触发那些表中已经存在的行做修改。

SET/DROP NOT NULL

标志一个列允许或者不允许空值。

SET STATISTICS

这种形式为随后的**ANALYZE**操作设置每列的统计信息收集目标。更多细节参考[ALTER TABLE](#)中类似的形式。

SET (*attribute_option* = *value* [, ...])

RESET (*attribute_option* [, ...])

这种形式设置或者重置了每个属性的选项。更多细节参考[ALTER TABLE](#)中类似的形式。

DISABLE/ENABLE [REPLICA | ALWAYS] TRIGGER

这些形式配置了触发那些属于外表的触发器。更多细节参考[ALTER TABLE](#)中类似的形式。

OWNER

这种形式修改外表的所有者为指定的用户。

RENAME

RENAME修改一个外表的名字或者外表中个别列的名字。

SET SCHEMA

这种形式将外表移到其他的模式下。

OPTIONS ([ADD | SET | DROP] *option* ['*value*'] [, ...])

为外表修改选项。ADD, SET和DROP 指定了要执行的操作。如果没有明确指出操作，默认的操作是ADD。选项名 必须是唯一的。MPP数据库用外部数据包装器来验证名字和值。

你可以将除You can combine all of the actions except RENAME 和 SET SCHEMA之外的所有操作写在一个列表里让Greemplum数据库来并行的应用这些修改。比如，可以在一个命令中增加多列或者修改多列的属性。

你必须是拥有这个表才能使用ALTER FOREIGN TABLE操作。为了修改外表的模式，你还必须在新的模式下拥有CREATE的权限。为了修改所有者，你必须是新的角色的直接或者间接成员，而且那个角色必须在表的模式下拥有CREATE权限。（这些限制强制要求通过删除和重新创建表来更改所有者不

会执行任何操作。但是，超级用户无论如何都可以更改任何表的所有权。) 为了新增或者修改一个列的类型，你必须在数据类型上拥有USAGE权限。

参数

name

要更改的现有外部表的名称（可能是模式限定的）。

column_name

新的或者已存在的列的名称。

new_column_name

已存在的列的新名称

new_name

外表的新的名称。

data_type

新的列的数据类型，或者已存在的列的新数据类型。

CASCADE

自动删除依赖于要删除列的对象（比如说，关联到列的视图）。

RESTRICT

拒绝删除有任何对象依赖的列。这是默认的表现。

trigger_name

要启用或者禁用的单个触发器名称。

ALL

禁用或启用属于外表的所有触发器。（如果任何触发器是内部生成的触发器，则需要超级用户权限。核心系统不会将此类触发器添加到外表，但附加代码可以这样做。）

USER

除内部生成的触发器外，禁用或启用属于外表的所有触发器。

new_owner

外表的新所有者的用户名。

new_schema

外表将移动到的模式的名称

Notes

关键字COLUMN是一个噪声词可以被忽略。

使用Consistency with the foreign server is not checked when a column is added or removed with ADD COLUMN或DROP COLUMN添加或删除列，添加NOT NULL 约束，或者使用SET DATA TYPE修改列类型时，不会检查与外部服务器的一致性.您有责任确保表定义与远程端匹配。

参考[CREATE FOREIGN TABLE](#)来获取未来更多的验证参数的描述。

示例

标志一个列为not-null:

```
ALTER FOREIGN TABLE distributors ALTER COLUMN street SET NOT NULL;
```

修改一个外表的选项：

```
ALTER FOREIGN TABLE myschema.distributors  
  OPTIONS (ADD opt1 'value', SET opt2 'value2', DROP opt3 'value3');
```

兼容性

这些ADD, DROP和 SET DATA TYPE的形式于SQL标准相符。其他形式为MPP数据库在SQL标准的扩展。在单个ALTER FOREIGN TABLE命令中指定多个操作的功能也是MPP数据库的扩展。

你可以使用ALTER FOREIGN TABLE ... DROP COLUMN 来删除外表中唯一的列，留下一个零列的表。这也是SQL的扩展，允许存在零列的外表。

另见

[ALTER TABLE](#), [CREATE FOREIGN TABLE](#), [DROP FOREIGN TABLE](#)

Parent topic: [SQL Command Reference](#)

ALTER FUNCTION

修改函数的定义。

概要

```
ALTER FUNCTION name ( [ [argmode] [argname] argtype [, ...] ] )  
    action [, ... ] [RESTRICT]
```

```
ALTER FUNCTION name ( [ [argmode] [argname] argtype [, ...] ] )  
    RENAME TO new_name
```

```
ALTER FUNCTION name ( [ [argmode] [argname] argtype [, ...] ] )  
    OWNER TO new_owner
```

```
ALTER FUNCTION name ( [ [argmode] [argname] argtype [, ...] ] )  
    SET SCHEMA new_schema
```

where *action* is one of:

```
{CALLED ON NULL INPUT | RETURNS NULL ON NULL INPUT | STRICT}  
{IMMUTABLE | STABLE | VOLATILE | [ NOT ] LEAKPROOF}  
{[EXTERNAL] SECURITY INVOKER | [EXTERNAL] SECURITY DEFINER}  
EXECUTE ON { ANY | MASTER | ALL SEGMENTS }  
COST execution_cost  
SET configuration_parameter { TO | = } { value | DEFAULT }  
SET configuration_parameter FROM CURRENT  
RESET configuration_parameter  
RESET ALL
```

描述

ALTER FUNCTION修改函数的定义。

你必须是函数的所有者才能使用ALTER FUNCTION。为了修改函数的模式，你还必须 在新模式下拥有CREATE权限。为了修改所有者，你还必须是新角色的直接或间接成员， 而且那个角色必须在函数模式上有用CREATE的权限。（这些限制强制执行 改变所有者不会通过删除和重新创建函数做任何你不能做的事情，但是，超级用户无论如何都可以改变任何函数的所有权。）

参数

name

一个存在的函数的名称（可能是模式限定的）。

argmode

参数的模式：IN, OUT, INOUT或者VARIADIC。如果省略，默认值为 IN。请注意ALTER FUNCTION实际上并不关注OUT参数， 因为只需要输入参数来确定函数的身份。因此已经足够了对于只列出IN, INOUT 和 VARIADIC参数

argname

参数的名称。请注意，ALTER FUNCTION实际上并不关心参数名称，因为只需要参数数据类型来确定函数的身份。

argtype

函数参数（如果有）的数据类型（可以是方案限定）。

new_name

函数的新名称。

new_owner

函数的新拥有者。请注意，如果函数被标记为 SECURITY DEFINER, 随后它将作为新的所有者执行。

new_schema

该函数的新模式。

CALLED ON NULL INPUT

RETURNS NULL ON NULL INPUT

STRICT

CALLED ON NULL INPUT将该函数改为在某些 或者全部参数为空值时可以被调用。 RETURNS NULL ON NULL INPUT 或者STRICT 更改函数，以便如果其任何参数为空，则不会调用该函数;而是自动假设一个空的结果。参阅[CREATE FUNCTION](#) 获取更多信息。

IMMUTABLE

STABLE

VOLATILE

将函数的波动性改为指定的设置。参阅[CREATE FUNCTION](#) 获取更多信息

[EXTERNAL] SECURITY INVOKER

[EXTERNAL] SECURITY DEFINER

更改该函数是否为一个安全性定义者。 关键词 EXTERNAL为了SQL的一致性而被忽略。参阅CREATE FUNCTION获取更多有关此功能的信息。

LEAKPROOF

更改函数是否被视为防漏的。参阅 [CREATE FUNCTION](#) 关于此功能的更多信息。

EXECUTE ON ANY

EXECUTE ON MASTER

EXECUTE ON ALL SEGMENTS

EXECUTE ON属性指定函数在查询执行过程中调用时执行的位置（主实例或段实例）。

EXECUTE ON ANY（默认值）表示该函数可以在主服务器或任何段实例上执行，并且无论执行的位置如何，它都会返回相同的结果。 MPP数据库确定函数执行的位置。

EXECUTE ON MASTER表示该函数必须只在主实例上执行。

EXECUTE ON ALL SEGMENTS表示对于每次调用，该函数必须在所有主段实例上执行，而不是在 主节点上执行。 该函数的总体结果是来自所有段实例的 UNION ALL结果。

EXECUTE ON属性的更多信息，请参阅[CREATE FUNCTION](#).

COST *execution_cost*

更改该函数的估计执行代价。参阅[CREATE FUNCTION](#) 获取更多信息。

configuration_parameter
value

当该函数被调用时，要为一个配置参数做出增加或者更改的赋值。如果 *value* 是 DEFAULT 或者, 等价的 RESET 被使用, 该函数本地的设置将会被 移除，这样该函数会使用其环境中存在的值执行。使用RESET ALL可以清除所有函数本地的设置。SET FROM CURRENT应用会话的当前值当函数被输入时。

RESTRICT

忽略SQL标准。

注意

MPP数据库对某些定义的函数有 STABLE 或者 VOLATILE 这样的限制。参阅[CREATE FUNCTION](#)来获取更多信息。

示例

将integer类型的函数sqrt重命名为square_root：

```
ALTER FUNCTION sqrt(integer) RENAME TO square_root;
```

更改integer 类型的 sqrt 函数的所有者为 joe

```
ALTER FUNCTION sqrt(integer) OWNER TO joe;
```

更改integer 类型的函数 sqrt 的模式为math:

```
ALTER FUNCTION sqrt(integer) SET SCHEMA math;
```

要调整一个函数的自动搜索路径:

```
ALTER FUNCTION check_password(text) RESET search_path;
```

兼容性

这个语句部分兼容 SQL 标准中的 ALTER FUNCTION 语句。该标准允许修改一个函数的更多属性，但是不提供重命名一个函数、标记一个函数为安全性定义者、为一个函数附加配置参数值或者更改一个函数的拥有者、模式或者稳定性等功能。该标准还需要 RESTRICT 关键字, 它在MPP数据库中是可选的。

另见

[CREATE FUNCTION](#), [DROP FUNCTION](#)

Parent topic: [SQL Command Reference](#)

ALTER GROUP

更改角色名称或成员关系。

概要

```
ALTER GROUP groupname ADD USER username [, ... ]
```

```
ALTER GROUP groupname DROP USER username [, ... ]
```

```
ALTER GROUP groupname RENAME TO newname
```

描述

ALTER GROUP 更改了用户组的属性，这是一个被废弃的命令，不过为了向后兼容 还是被接受的。组和用户被更一般的概念角色所替代，参阅 [ALTER ROLE](#) 获取更多信息

前面两个语句往组里面添加或者删除一个用户，任何角色都可以当作是用户名或者组名。完成这种 任务的首选方式是使用[GRANT](#) 和 [REVOKE](#).

参数

groupname

要修改的组（角色）的名称

username

要添加到组或从组中删除的用户（角色）。用户（角色）必须已经存在。

newname

组（角色）的新名称。

示例

要将用户添加到组中：

```
ALTER GROUP staff ADD USER karl, john;
```

从组中删除用户:

```
ALTER GROUP workers DROP USER beth;
```

兼容性

在SQL标准中没有ALTER GROUP语句

See Also

[ALTER ROLE](#), [GRANT](#), [REVOKE](#)

Parent topic: [SQL Command Reference](#)

ALTER INDEX

更改一个索引的定义。

概要

```
ALTER INDEX [ IF EXISTS ] name RENAME TO new_name
```

```
ALTER INDEX [ IF EXISTS ] name SET TABLESPACE tablespace_name
```

```
ALTER INDEX [ IF EXISTS ] name SET ( storage_parameter = value [, ...] )
```

```
ALTER INDEX [ IF EXISTS ] name RESET ( storage_parameter [, ...] )
```

```
ALTER INDEX ALL IN TABLESPACE name [ OWNED BY role_name [, ...] ]  
SET TABLESPACE new_tablespace [ NOWAIT ]
```

描述

ALTER INDEX更改存在的一个索引的定义。 有下列的一些子形式：

- **RENAME** — 修改索引的名字，对存储的数据无影响。
- **SET TABLESPACE** — 修改索引的表空间到指定的表空间，然后将分配给这个索引的数据文件也移到新的表空间。要修改索引的表空间，你必须在新的表空间有CREATE 的权限。在当前数据库中一个表空间下所有的索引可以用ALL IN TABLESPACE形式移动, 这会锁住所有的要移动的索引，然后移动每一个索引。这种方式也在OWNED BY形式中支持，这只会移动指定角色所拥有的索引。指定了NOWAIT选项的话，如果不能立即获取到需要的锁，命令将会失败。注意系统目录不会被这个命令移动，如果需要，请使用ALTER DATABASE 或者显式调用调用ALTER INDEX。另见CREATE TABLESPACE.
- **IF EXISTS** — 如果索引不存在不会抛出错误，而会触发一个提醒。
- **SET** — 更改索引的特定于索引方法的存储参数。 内置索引方法都接受一个参数：*fillfactor*。索引的fillfactor是一个百分比，用于确定索引方法尝试打包索引页的程度。此命令不会立即修改索引内容。使用REINDEX重建索引以获得所需的效果。
- **RESET** — 将索引的存储参数重置为其默认值。 内置索引方法都接受一个参

数：*fillfactor*。与SET一样，可能需要 REINDEX 来完全更新索引。

参数

name

要修改的现有索引的名称（可选方案限定）。

new_name

索引的新名称。

tablespace_name

要移动索引的表空间。

storage_parameter

index-method-specific指定的存储参数名称。

value

index-method-specific指定的存储参数值，这可能是基于参数的数字或者词

注意

这些操作也可能用ALTER TABLE实现。

不允许更改系统目录索引的任何部分。

示例

修改一个索引的名字：

```
ALTER INDEX distributors RENAME TO suppliers;
```

移动索引到另外一个表空间：

```
ALTER INDEX distributors SET TABLESPACE fasttablespace;
```

要更改索引的填充因子（假设索引方法支持它）：

```
ALTER INDEX distributors SET (fillfactor = 75);  
REINDEX INDEX distributors;
```

兼容性

ALTER INDEX 是MPP数据库的扩展。

另见

[CREATE INDEX](#), [REINDEX](#), [ALTER TABLE](#)

Parent topic: [SQL Command Reference](#)

ALTER LANGUAGE

更改过程语言的名称。

概要

```
ALTER LANGUAGE name RENAME TO newname  
ALTER LANGUAGE name OWNER TO new_owner
```

描述

ALTER LANGUAGE修改指定数据库的过程语言定义。定义的修改支持包括重命名语言或者指定一个新的所有者。你必须是超级用户或者是这个语言的所有者才能使用ALTER LANGUAGE.

参数

name

语言的名称。

newname

语言的新名称

new_owner

语言的新的所有者。

兼容性

在SQL标准中没有ALTER LANGUAGE语句。

另见

CREATE LANGUAGE, DROP LANGUAGE

Parent topic: [SQL Command Reference](#)

ALTER OPERATOR

更改操作符的定义。

概要

```
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
```

描述

ALTER OPERATOR 更改一个操作符的定义。目前唯一可用的功能是修改操作符的所有者。

用户必须是操作符的所有者才能使用ALTER OPERATOR。要更改所有者，用户必须是新角色的直接或间接成员，而且该角色必须在操作符的模式上有CREATE 权限。（这种限制强制要求即使更改所有者也不能做那些通过删除或重建操作符所不能做到的事情。然而，超级用户可以任意修改操作符的所有

权。)

参数

name

现有操作符的名称（可选方案限定）

left_type

操作符左操作数的数据类型；记为NONE 如果没有左操作数。

right_type

NONE操作符右操作数的数据类型；记为 NONE 如果操作符没有右操作数。

new_owner

操作符新的所有者。

new_schema

操作符新的模式。

示例

更改一个text类型的自定义操作符a @@ b：

```
ALTER OPERATOR @@ (text, text) OWNER TO joe;
```

兼容性

在SQL标准中没有 ALTER OPERATOR语句。 in the SQL standard.

另见

[CREATE OPERATOR, DROP OPERATOR](#)

Parent topic: [SQL Command Reference](#)

ALTER OPERATOR CLASS

更改一个操作符类的定义。

概要

```
ALTER OPERATOR CLASS name USING index_method RENAME TO new_name
```

```
ALTER OPERATOR CLASS name USING index_method OWNER TO new_owner
```

```
ALTER OPERATOR CLASS name USING index_method SET SCHEMA new_schema
```

描述

ALTER OPERATOR CLASS 更改操作符类的定义。

用户必须是操作符类的所有者才能使用ALTER OPERATOR CLASS. 要更改所有者，用户必须是新角色的直接或间接成员，而且该角色必须在操作符 的模式

上有CREATE 权限。（这种限制强制要求即使更改所有者也不能做那些通过删除或重建操作符所不能做到的事情。然而，超级用户可以任意修改操作符类的所有权。）

参数

- name*
现有操作符类的名称（可选方案限定）
- index_method*
操作符类索引方法的名称。
- new_name*
操作符类的新名称。
- new_owner*
操作符类的新的拥有者。
- new_schema*
操作符类的新模式。

兼容性

在SQL标准中没有ALTER OPERATOR CLASS语句

另见

[CREATE OPERATOR CLASS](#), [DROP OPERATOR CLASS](#)

Parent topic: [SQL Command Reference](#)

ALTER OPERATOR FAMILY

更改操作符族的定义。

概要

```
ALTER OPERATOR FAMILY name USING index_method ADD
{ OPERATOR strategy_number operator_name ( op_type, op_type ) [ FOR SEARCH | FOR C
  | FUNCTION support_number [ ( op_type [ , op_type ] ) ] funcname ( argument_type [ , ... ] )
} [ , ... ]
```

```
ALTER OPERATOR FAMILY name USING index_method DROP
{ OPERATOR strategy_number ( op_type, op_type )
  | FUNCTION support_number [ ( op_type [ , op_type ] )
} [ , ... ]
```

```
ALTER OPERATOR FAMILY name USING index_method RENAME TO new_name
```

```
ALTER OPERATOR FAMILY name USING index_method OWNER TO new_owner
```

```
ALTER OPERATOR FAMILY name USING index_method SET SCHEMA new_schema
```

描述

ALTER OPERATOR FAMILY更改操作符族的定义。用户能够向族中添加操作符并提供函数支持、从族中删除它们，或者更改族的名称或所有者。

当通过ALTER OPERATOR FAMILY将操作符和函数支持添加到族中，则在族

中它们不是任何指定操作符类的一部分，而仅是“松散”的存在于族内。这表示这些操作符和函数与该族的语义兼容，但是没有被任何特定索引的正确功能所要求。（要求操作符和函数作为操作符类的部分声明；参见 [CREATE OPERATOR CLASS](#)。）用户可以在任何时候将一个族中松散的成员从族中删除，但是除非将整个类和所有依赖于这个类的索引都删除，否则一个操作符类的成员不能被删除。具有代表性的是，单一数据类型的操作符和函数是操作类的一部分，因为在指定数据类型上的索引需要他们支持，而多数据类型操作符和函数则被作为该族的松散成员。

用户必须是超级用户才能使用ALTER OPERATOR FAMILY。（做这样的限制是因为一个错误的操作符族定义可能会迷惑服务器甚至让它崩溃）

ALTER OPERATOR FAMILY 目前不检测操作符族定义是否包括该索引方法所要求的所有操作符和函数，也不检查操作符和函数是否形成了一个有理的集合。定义合法的操作符族是用户的责任。

OPERATOR 和 FUNCTION子句可以以任何顺序出现。

参数

name

现有操作符族的名称（可选限定模式）。

index_method

操作符族所应用的索引方法的名称。

strategy_number

与操作符族相关联的一个操作符的索引方法策略号。

operator_name

与操作符族相关联的一个操作符的名称（可选限定模式）。

op_type

在一个OPERATOR子句中，操作符的操作数的数据类型，或者用NONE来表示一个左一元或者右一元操作符。不同于CREATE OPERATOR CLASS中类似的语法，操作数数据类型总是必须被指定。

在一个ADD FUNCTION子句中, 指定该函数意图支持的操作数 数据类型 (如果不同于该函数的输入数据类型)。对于 B-树比较函数和哈希 函数, 有必要指定`op_type`, 因为该函数的输入数据类型 总是正确的。对于 B-树排序支持函数和GIN和GiST 操作符类中 的所有函数, 有必要指定该函数要使用的操作数数据类型。在一个 DROP FUNCTION 子句中, 必须指定该函数要支持的操 作数数据类型。

sort_family_name

在排序操作符中关联的描述排序顺序的现有btree 操作符名称。

如果即未指定FOR SEARCH也未指定FOR ORDER BY, 则默认为FOR SEARCH。

support_number

索引方法与操作员族相关联的函数的支持程序编号。

funcname

作为该操作符族的一种索引方法支持过程的函数的名称 (可以是方案限定的)。

argument_types

该函数的参数数据类型。

new_name

该操作符族的新名称。

new_owner

该操作符族的新拥有者。

new_schema

该操作族中的新模式。

兼容性

在SQL标准中没有ALTER OPERATOR FAMILY语句。

注意

注意DROP语法只通过策略或者支持号以及输入数据类型指定该操作符族中的"slot"。占用这个槽的操作符或函数的名称不会被提及。还有，对于DROP FUNCTION要指定的类型是该函数意图支持的输入数据类型；对于SP_GiST和GIN索引可能无需对该函数的实际输入参数类型做任何事情。

因为索引机制在使用函数之前不会检查其上的访问权限，包括一个操作符族中的函数或操作符都等同于授予了其上的公共执行权限。这对于操作符族中很有用的这类函数来说，这通常不成问题。

操作符不应该由 SQL 函数定义。一个 SQL 函数很可能被内联到调用查询中，这将阻止优化器识别出该查询匹配一个索引。

在MPP Database 6.0之前，OPERATOR 子句可以包括一个RECHECK 选项。这个选项不再被支持了。MPP数据库现在运行中决定一个索引操作符是否是动态有损，这将允许更多有效处理那些操作符的是否有损的情况。

示例

下列示例命令为一个操作符族增加跨数据类型的操作符和支持函数，该操作符族已经包含用于数据类型int4以及int2的 B-树 操作符类：

```
ALTER OPERATOR FAMILY integer_ops USING btree ADD
```

```
-- int4 vs int2
OPERATOR 1 < (int4, int2) ,
OPERATOR 2 <= (int4, int2) ,
OPERATOR 3 = (int4, int2) ,
OPERATOR 4 >= (int4, int2) ,
OPERATOR 5 > (int4, int2) ,
FUNCTION 1 btint42cmp(int4, int2) ,
```

```
-- int2 vs int4
OPERATOR 1 < (int2, int4) ,
OPERATOR 2 <= (int2, int4) ,
OPERATOR 3 = (int2, int4) ,
OPERATOR 4 >= (int2, int4) ,
OPERATOR 5 > (int2, int4) ,
FUNCTION 1 btint24cmp(int2, int4) ;
```

再次移除这些项：

```
ALTER OPERATOR FAMILY integer_ops USING btree DROP
```

```
-- int4 vs int2
OPERATOR 1 (int4, int2) ,
OPERATOR 2 (int4, int2) ,
OPERATOR 3 (int4, int2) ,
OPERATOR 4 (int4, int2) ,
OPERATOR 5 (int4, int2) ,
FUNCTION 1 (int4, int2) ,
```

```
-- int2 vs int4
OPERATOR 1 (int2, int4) ,
OPERATOR 2 (int2, int4) ,
OPERATOR 3 (int2, int4) ,
OPERATOR 4 (int2, int4) ,
OPERATOR 5 (int2, int4) ,
FUNCTION 1 (int2, int4) ;
```

另见

[CREATE OPERATOR FAMILY](#), [DROP OPERATOR FAMILY](#), [ALTER OPERATOR CLASS](#), [CREATE OPERATOR CLASS](#), [DROP OPERATOR CLASS](#)

Parent topic: [SQL Command Reference](#)

ALTER PROTOCOL

更改一个协议的定义。

概要

ALTER PROTOCOL *name* RENAME TO *newname*

ALTER PROTOCOL *name* OWNER TO *newowner*

描述

ALTER PROTOCOL 更改一个协议的定义，只有协议的名字和所有者可以被更改。

用户必须拥有协议才可以使用ALTER PROTOCOL。要更改所有者，用户还必须

须是新角色的 直接或者间接成员，而且新角色必须在该转换模式下拥有CREATE权限。

这些限制适当的确保修改所有者只能通过删除或重建协议。注意一个超级用户可以修改任何协议的所属关系。

参数

name

现有协议的名称（可选方案限定）。

newname

协议的新名称

newowner

协议的新所有者。

示例

重命名转换GPDBauth为 GPDB_authentication：

```
ALTER PROTOCOL GPDBauth RENAME TO GPDB_authentication;
```

更改转换GPDB_authentication 的所有者为 joe：

```
ALTER PROTOCOL GPDB_authentication OWNER TO joe;
```

兼容性

SQL标准中没有 ALTER PROTOCOL 语句。

另见

[CREATE EXTERNAL TABLE](#), [CREATE PROTOCOL](#)

Parent topic: [SQL Command Reference](#)

ALTER RESOURCE GROUP

更改一个资源组的限制项。

概要

```
ALTER RESOURCE GROUP name SET group_attribute value
```

where *group_attribute* is one of:

- CONCURRENCY *integer*
- CPU_RATE_LIMIT *integer*
- CPUSET *tuple*
- MEMORY_LIMIT *integer*
- MEMORY_SHARED_QUOTA *integer*
- MEMORY_SPILL_RATIO *integer*

描述

ALTER RESOURCE GROUP更改一个资源组的限制项。只有超级用户才能修改资源组。

用户可以为控制角色最大活动并行度的资源组设置或者重置并行度的限制。用户还可以通过 为资源组重置内存或者CPU资源来控制所有通过资源组提交的查询在每个Segment主机上可以使用 的内存或CPU总量。

当用户更改资源组CPU资源的模式或者限制时，新的模式或限制会立即生效。

当用户更改为角色创建的资源组的内存限制时，如果当前资源使用率小于或等于新值且资源组中没有正在运行的事务，则会立即应用新的资源限制。如果当前资源使用量超过新内存限制值，或者其他资源组中存在运行事务在持有某些资源，则MPP数据库会推迟分配新限制，直到资源使用率落在新值的范围内。

当用户提升为外部组件创建的资源组的内存限制时，新的资源限制会随着资源可用而逐步生效。如果用户降低为外部组件创建的资源组的内存限制时，则表现是基于特定组件的。比如说，如果用户降低为PL / Container运行时创建的资源组的内存限制，则正在运行的容器中的查询可能会因内存不足的错误而失败。

用户可以在单个 ALTER RESOURCE GROUP 调用中更改限制的类型。

参数

name

要更改的资源组的名称。

CONCURRENCY *integer*

用户为角色分配的资源组的允许的最大的事务并行数，包括活动和空闲的事务。任何在 CONCURRENCY 的限制值达到后要提交的事务都会进入队列， 当一个运行的事务结束后，最早排队的事务会被执行。

CONCURRENCY 值必须是在[0 .. max_connections]范围内的整数。用户

为角色创建的资源组默认的 CONCURRENCY 值为 20。

Note: 用户不可以将 admin_group 的 CONCURRENCY 值设为零 (0)。

CPU_RATE_LIMIT *integer*

为当前资源组分配的CPU资源的百分比。资源组的CPU资源最小百分比为 1，最大为 100。为MPP数据库集群设置的所有资源租的 CPU_RATE_LIMIT值总共不能超过 100。

如果用户为之前设置过CPUSET的资源组修

改CPU_RATE_LIMIT，CPUSET 会被禁用，保留的 CPU 核返回给 MPP 数据库，CPUSET 被设置成 -1。

CPUSET *tuple*

为当前资源组保留的CPU核。用户在 *tuple* 中指定的CPU核必须在系统上是可用的，而且用户不能将其他资源组中已指定的CPU核交叠分配给当前资源组

tuple 是以逗号分隔的单核数或核心间隔列表。您必须用单引号括起*tuple*，例如'1,3-4'。

如果用户为之前设置过CPU_RATE_LIMIT的资源组修改CPUSET，CPU_RATE_LIMIT 会被禁用，保留的CPU资源会返还给Greeplum数据库，然后 CPU_RATE_LIMIT 会被设置为 -1。

用户只能在启用了基于组管理资源的MPP数据库集群中为资源组修改CPUSET。

MEMORY_LIMIT *integer*

为当前资源组分配的内存资源的百分比。资源组的内存资源最小百分比为 1，最大为 100，为MPP数据库集群中所有资源组定义的MEMORY_LIMIT总值不能超过100。

MEMORY_SHARED_QUOTA *integer*

为当前资源组中分配事务间共享的内存资源的百分比。资源组中最小的共享内存分配比为0，最大为100，默认的MEMORY_SHARED_QUOTA值为20。

MEMORY_SPILL_RATIO *integer*

资源组中发出的事务中内存密集型运算符的内存使用量阈值。资源组的最小内存溢出率百分比为0.最大值为100。MEMORY_SPILL_RATIO 的默认值是 20。

注意

用 `CREATE ROLE` 或者 `ALTER ROLE` 来将一个指定资源组分配给角色（用户）。

用户不能在显式事务或者子事务中提交 `ALTER RESOURCE GROUP` 命令。

示例

更改资源组的活动事务限制：

```
ALTER RESOURCE GROUP rgroup1 SET CONCURRENCY 13;
```

更新资源组的CPU限制：

```
ALTER RESOURCE GROUP rgroup2 SET CPU_RATE_LIMIT 45;
```

更新资源组的内存限制：

```
ALTER RESOURCE GROUP rgroup3 SET MEMORY_LIMIT 30;
```

从默认值中提升资源组的内存溢出率：

```
ALTER RESOURCE GROUP rgroup4 SET MEMORY_SPILL_RATIO 25;
```

为资源组保留CPU核 1：

```
ALTER RESOURCE GROUP rgroup5 SET CPUSSET '1';
```

兼容性

ALTER RESOURCE GROUP 语句是MPP数据库的扩展，在PostgreSQL标准中没有该命令。

另见

[CREATE RESOURCE GROUP](#), [DROP RESOURCE GROUP](#), [CREATE ROLE](#), [ALTER ROLE](#)

Parent topic: [SQL Command Reference](#)

ALTER RESOURCE QUEUE

更改资源队列的限制。

概要

```
ALTER RESOURCE QUEUE name WITH ( queue_attribute=value [, ... ] )
```

其中 *queue_attribute* 是：

```
ACTIVE_STATEMENTS=integer  
MEMORY_LIMIT='memory_units'  
MAX_COST=float  
COST_OVERCOMMIT={TRUE|FALSE}  
MIN_COST=float  
PRIORITY={MIN|LOW|MEDIUM|HIGH|MAX}
```

```
ALTER RESOURCE QUEUE name WITHOUT ( queue_attribute [, ... ] )
```

其中 *queue_attribute* 是：

```
ACTIVE_STATEMENTS  
MEMORY_LIMIT  
MAX_COST  
COST_OVERCOMMIT  
MIN_COST
```

Note: 一个资源队列必须拥有 ACTIVE_STATEMENTS 或 MAX_COST 其中的一个值。不能将两者都从资源队列的 queue_attributes 中删除。

描述

ALTER RESOURCE QUEUE 更改一个资源队列的限制，只有超级用户才能更改资源队列。一个资源队列必须拥有 ACTIVE_STATEMENTS 或者 MAX_COST 值（或者两者都有）。用户可以设置或者重置一个资源队列的优先级来控制与资源队列相关的查询所使用的CPU可用资源，或者设置资源队列的内存限制来控制资源队列中所有提交的查询在一个segment主机上可以使用的内存总量。

ALTER RESOURCE QUEUE WITHOUT 删除之前在资源上指定的限制。一个资源队列必须拥有 ACTIVE_STATEMENTS 或者 MAX_COST 其中一个值。不能两者都从资源队列的 queue_attributes 中删除。

参数

name

要更改其限制的资源队列的名称。

ACTIVE_STATEMENTS integer

任何时刻系统中允许在该资源队列中的用户提交的活动语句的数量。

ACTIVE_STATEMENTS 的值应该是一个大于0的整数。要把ACTIVE_STATEMENTS 重置为没有限制，则指定值为 -1。

MEMORY_LIMIT 'memory_units'

设置从此资源队列中的用户提交的所有语句的总内存配额。内存单位可以指定为kB，MB或GB。资源队列的最小内存配额为10MB。没有最大值；然而，查询执行时间的上边界受到segment主机的物理内存的限制。默认值为无限制(-1)。

MAX_COST float

任何时刻系统中允许在该资源队列中的用户提交的语句的查询优化器总代价。MAX_COST的 值被指定为一个浮点数（例如100.00）或者还可以被指定为一个指数（例如1e+2）。要把MAX_COST重置为没有限制，输入一个值-1.0。

COST_OVERCOMMIT boolean

如果资源队列受到基于查询代价的限制，那么管理员可以允许代价过量使用（默认COST_OVERCOMMIT=TRUE）。这意味着一个超过允许的代价阈值的查询将被允许运行，但只能在系统空闲时运行。如果指定COST_OVERCOMMIT=FALSE，超过代价限制的查询将总是被拒绝并且绝不会被允许运行。

MIN_COST float

代价低于此限制的查询将不会排队而是立即运行。代价是以取得的磁盘页为单位来衡量的。1.0等于一次顺序磁盘页面读取。MIN_COST的值被指定为浮点数（例如100.00）或者还能被指定为一个指数（例如1e+2）。要把MIN_COST重置为没有限制，输入一个值 -1.0。

PRIORITY={MIN|LOW|MEDIUM|HIGH|MAX}

设置与资源队列关联的查询的优先级。具有较高优先级的队列中的查询或语句将在竞争中获得更多的可用CPU资源份额。低优先级队列中的查询可能会被延迟，同时执行更高优先级的查询

注意

GPORCA和MPP数据库原生优化器利用不同的查询耗费模型，所以同样的查询可能会计算出不同的消耗值。MPP数据库资源队列资源管理模型既不区分也不调整GPORCA和Postgres优化器的成本，它会直接使用优化器返回的消耗值来限制查询。

当基于资源队列的资源管理处于活动状态时，请使用资源队列的MEMORY_LIMIT和ACTIVE_STATEMENTS限制，而不是配置基于成本的限制。即使在使用GPORCA时，MPP数据库可能会回到使用postgres规划器进行某些查询，因此使用基于成本的限制可能会导致意外的结果。

示例

更改资源队列的活动查询限制：

```
ALTER RESOURCE QUEUE myqueue WITH (ACTIVE_STATEMENTS=20);
```

更改资源队列的内存限制：

```
ALTER RESOURCE QUEUE myqueue WITH (MEMORY_LIMIT='2GB');
```

将资源队列的最大和最小查询代价限制重置为无限制：

```
ALTER RESOURCE QUEUE myqueue WITH (MAX_COST=-1.0,  
MIN_COST= -1.0);
```

将资源队列的查询代价限制重置为 3^{10} (或者是 30000000000.0) 不允许过量使用：

```
ALTER RESOURCE QUEUE myqueue WITH (MAX_COST=3e+10,  
COST_OVERCOMMIT=FALSE);
```

将与资源队列关联的查询的优先级重置为最小级别：

```
ALTER RESOURCE QUEUE myqueue WITH (PRIORITY=MIN);
```

从资源队列中去除 MAX_COST 和 MEMORY_LIMIT 限制：

```
ALTER RESOURCE QUEUE myqueue WITHOUT (MAX_COST, MEMORY_LIMIT);
```

兼容性

ALTER RESOURCE QUEUE 语句是 MPP数据库的扩展，在标准PostgreSQL中不存在这个命令。

另见

[CREATE RESOURCE QUEUE](#), [DROP RESOURCE QUEUE](#), [CREATE ROLE](#), [ALTER ROLE](#)

Parent topic: [SQL Command Reference](#)

ALTER ROLE

更改一个数据库角色（用户或组）。

概要

```
ALTER ROLE name [ [ WITH ] option [ ... ] ]
```

其中 *option* 可以是：

```
SUPERUSER | NOSUPERUSER  
| CREATEDB | NOCREATEDB  
| CREATEROLE | NOCREATEROLE  
| CREATEEXTTABLE | NOCREATEEXTTABLE [ ( attribute='value' [, ...] )  
  where attributes and values are:  
    type='readable'|'writable'  
    protocol='gpfdist'|'http'  
| INHERIT | NOINHERIT  
| LOGIN | NOLOGIN  
| REPLICATION | NOREPLICATION  
| CONNECTION LIMIT connlimit  
| [ ENCRYPTED | UNENCRYPTED ] PASSWORD 'password'  
| VALID UNTIL 'timestamp'
```

```
ALTER ROLE name RENAME TO new_name
```

```
ALTER ROLE { name | ALL } [ IN DATABASE database_name ] SET configuration_parameter  
ALTER ROLE { name | ALL } [ IN DATABASE database_name ] SET configuration_parameter  
ALTER ROLE { name | ALL } [ IN DATABASE database_name ] RESET configuration_parameter  
ALTER ROLE { name | ALL } [ IN DATABASE database_name ] RESET ALL  
ALTER ROLE name RESOURCE QUEUE {queue_name | NONE}  
ALTER ROLE name RESOURCE GROUP {group_name | NONE}
```

描述

ALTER ROLE更改MPP数据库角色的属性。此命令有几种变体

WITH *option*

修改可以在CREATE ROLE中指定的大多数角色属性。(包含了所有可能的属性，但不包括添加或删除成员身份的选项；为那些选项使用 GRANT 和 REVOKE。)在这个命令中没提到的属性将保留她们原来的值。超级用户可以为任何角色修改任何设置。拥有 CREATEROLE 权限的角色可以修改任意的这些设置，但是仅限于非超级用户 和非replication角色。普通用户只可以修改自己的密码。

RENAME

更改角色的名称。数据库超级用户可以重命名任何角色。角色有 CREATEROLE 特权 可以重命名非超级用户角色。无法重命名当前会话用户（以其他用户身份连接重命名角色）。因为MD5加密的密码使用角色名称作为密钥，如果密码为MD5加密，则重命名角色将清除其密码。

SET | RESET

为指定的配置参数更改角色的会话默认值，对于所有数据库，或者在IN DATABASE子句指定数据库时，仅对命名数据库中的会话 进行更改。如果指定了all而不是角色名，则会更改所有角色的设置。在IN DATABASE使用ALL实际上与使用命令ALTER DATABASE...SET...一样。

每当角色随后启动新会话时，指定的值将成为会话默认值，覆盖服务器配置文件（postgresql.conf）中存在的,或从postgres命令行接收到的任何设置。这只会发生在登录时，执行SET ROLE 或者 SET SESSION AUTHORIZATION 不会触发设置新的值

附加到角色的数据库特定设置将覆盖所有数据库的设置。特定数据库或特定角色的设置将覆盖所有角色的设置。

对于没有LOGIN权限的角色，会话默认值无效。普通角色可以更改自己的会话默认值。超级用户可以更改任何人的会话默认值。具有CREATEROLE权限的角色可以更改非超级用户角色的默认值。普通角色只能为自己设置默认值。某些配置变量不能这样设置，或者只能在超级用户发出命令时设置。有关所有用户可设置配置参数的信息，请参阅MPP数据库参考手册。只有超级用户才能更改所有数据库中所有角色的设置。

RESOURCE QUEUE

将角色分配给工作负载管理资源队列。在发出查询时，角色将受到分配资源队列的限制。指定NONE将角色分配给默认资源队列。一个角色只能属于一个资源队列。对于没有LOGIN特权的角色，会话默认值没有任何作用。参考[CREATE RESOURCE QUEUE](#) 获取更多信息。

RESOURCE GROUP

为角色分配资源组。然后，角色将受制于为资源组配置的并发事务、内存和CPU限制。可以将单个资源组分配给一个或多个角色。不能将为外部组件创建的资源组分配给角色。参考[CREATE RESOURCE GROUP](#)获取更多信息。

参数

name

将被修改属性的角色名。

new_name

该角色的新名称。

database_name

将要设置配置参数的数据库名。

config_parameter=value

将指定配置参数的此角色会话默认值设置为给定值。如果 *value* 是 DEFAULT 指或者指定 RESET，则角色的指定参数设置会被删除，角色将在新会话里面继承系统层面的默认值，使用 RESET ALL 可以清除所有的角色特殊配置。SET FROM CURRENT 保存会话的当前参数值作为角色指定的值。如果指定了 IN DATABASE，则只会为指定的角色数据库。当随后角色 开启新的会话时，指定的参数值成为会话的默认值，覆盖服务器配置文件（postgresql.conf）中存在的,或从 postgres 命令行接收到的任何设置。

角色指定的变量设置只会在登录时生效; SET ROLE 和 [SET SESSION AUTHORIZATION](#) 不会处理指定角色的变量设置。

参阅 [服务器配置参数](#) 来获取更多关于用户可设置参数的信息。

group_name

分配给该角色的资源组名称。指定`group_name`为 `NONE` 删除角色当前分配的资源组，基于角色的能力分配一个默认资源组。 `SUPERUSER` 角色分配 `admin_group` 资源组，而`default_group` 资源组则分配给非`admin` 角色。

不能将为外部组件创建的资源组分配给角色。

queue_name

要分配用户级角色的资源队列的名称。 只有`LOGIN`特权的角色 可以分配给资源队列。要从资源队列中取消分配角色并将其置于默认资源队列中，请指定`NONE`。 角色只能属于一个资源队列。

`SUPERUSER` | `NOSUPERUSER`

`CREATEDB` | `NOCREATEDB`

`CREATEROLE` | `NOCREATEROLE`

`CREATEUSER` | `NOCREATEUSER`

`CREATEUSER` 和 `NOCREATEUSER` 已经过期，但是 仍被接受为`SUPERUSER`和 `NOSUPERUSER`。注意，它们 不等同于`CREATEROLE` and和`NOCREATEROLE`子句。

`CREATEEXTTABLE` | `NOCREATEEXTTABLE` [(attribute='value')]

如果`CREATEEXTTABLE` 被指定， 允许定义的角色创建外部表。如果没被指定，默认类型是`readable`，并且默认协议 是`gpfdist`。

`NOCREATEEXTTABLE`（默认）拒绝角色有创建外部表的能力。 注意使用的外部表`file`或`execute`协议只能由超级用户创建。

`INHERIT` | `NOINHERIT`

`LOGIN` | `NOLOGIN`

`REPLICATION`

`NOREPLICATION`

`CONNECTION LIMIT` `conlimit`

`PASSWORD` `password`

`ENCRYPTED` | `UNENCRYPTED`

`VALID UNTIL` 'timestamp'

这些子句通过[CREATE ROLE](#)改变了原来设置的角色属性。

`DENY` `deny_point`

`DENY BETWEEN` `deny_point` `AND` `deny_point`

DENY和DENY BETWEEN关键字设置了在登录时强制执行的基于时间的约束。DENY设置一天或一天的时间来拒绝访问。DENY BETWEEN设置访问被拒绝的间隔。两者都使用以下格式的参数`deny_point`：

```
DAY day [ TIME 'time' ]
```

`deny_point`两部分参数使用以下格式：

对于 day：

```
{ 'Sunday' | 'Monday' | 'Tuesday' | 'Wednesday' | 'Thursday' | 'Friday' |  
'Saturday' | 0-6 }
```

对于time：

```
{ 00-23 : 00-59 | 01-12 : 00-59 { AM | PM } }
```

DENY BETWEEN子句使用两种`deny_point`参数。

```
DENY BETWEEN deny_point AND deny_point
```

有关基于时间的约束和示例的更多信息，参阅MPP数据库管理员手册中的“管理角色和特权”。

DROP DENY FOR `deny_point`

该DROP DENY FOR子句从角色中删除基于时间的约束。它使用上述的`deny_point`参数。

有关基于时间的约束和示例的更多信息，参阅MPP数据库管理员手册中的“管理角色和特权”。

注意

使用 **CREATE ROLE** 新增角色，使用**DROP ROLE** 删除角色。

使用 [GRANT](#) 和 [REVOKE](#) 来增加和删除角色成员。

使用此命令指定未加密的密码时，必须小心。密码将以明文形式发送到服务器，也可能会记录在客户端的命令历史记录或服务器日志中。该 `psql` 命令行客户端包含一个元命令 `\password` 可用于安全地更改角色的密码。

还可以将会话默认值与特定数据库而不是角色绑定。如果存在冲突，则特定于角色的设置将覆盖数据库特定的设置。参阅 [ALTER DATABASE](#)。

示例

更改角色的密码：

```
ALTER ROLE daria WITH PASSWORD 'passwd123';
```

删除角色的密码：

```
ALTER ROLE daria WITH PASSWORD NULL;
```

更改密码失效日期：

```
ALTER ROLE scott VALID UNTIL 'May 4 12:00:00 2015 +1';
```

使密码永久有效：

```
ALTER ROLE luke VALID UNTIL 'infinity';
```

赋予角色创建其他角色和新数据库的能力：

```
ALTER ROLE joelle CREATEROLE CREATEDB;
```

给角色一个非默认设置maintenance_work_mem参数：

```
ALTER ROLE admin SET maintenance_work_mem = 100000;
```

给角色一个非默认，指定数据库的参数client_min_messages 值：

```
ALTER ROLE fred IN DATABASE devel SET client_min_messages = DEBUG;
```

将角色分配给资源队列：

```
ALTER ROLE sammy RESOURCE QUEUE poweruser;
```

授予创建可写外部表的角色权限：

```
ALTER ROLE load CREATEEXTTABLE (type='writable');
```

更改角色在星期日不允许登录访问：

```
ALTER ROLE user3 DENY DAY 'Sunday';
```

改变角色以消除星期日不允许登录访问的约束：

```
ALTER ROLE user3 DROP DENY FOR DAY 'Sunday';
```

指定一个新的资源组给角色：

```
ALTER ROLE parttime_user RESOURCE GROUP rg_light;
```

兼容性

ALTER ROLE 语句是 MPP 数据库的扩展

另见

[CREATE ROLE](#), [DROP ROLE](#), [ALTER DATABASE](#), [SET](#), [CREATE RESOURCE GROUP](#), [CREATE RESOURCE QUEUE](#), [GRANT](#), [REVOKE](#)

Parent topic: [SQL Command Reference](#)

ALTER RULE

修改一个规则的定义。

概要

```
ALTER RULE name ON table_name RENAME TO new_name
```

描述

ALTER RULE 更改一个存在的规则的属性，当前可用的操作时修改规则的名

称。

要使用ALTER RULE用户必须是这个规则应用的表或视图拥有者。

参数

name
要更改的存在的规则的名称。

table_name
规则应用的表或者视图名称（可选的限定模式）。

new_name
规则的新名称。

兼容性

ALTER RULE 是MPP数据库语言的扩展，是一个完整的查询重写系统。

另见

[CREATE RULE](#), [DROP RULE](#)

ALTER SCHEMA

更改一个模式定义。

概要

```
ALTER SCHEMA name RENAME TO newname
```

```
ALTER SCHEMA name OWNER TO newowner
```

描述

ALTER SCHEMA 更改一个模式定义。

用户必须拥有该模式才能去使用 ALTER SCHEMA. 要重命名一个模式，用户还必须拥有该数据库的 CREATE 特权。要更改拥有者，用户还必须 是新拥有角色的一个直接或者间接成员，并且该角色必须具有该数据库上的 CREATE 特权。注意超级用户自动拥有所有这些特权。.

参数

name

现有模式的名称。

newname

该模式的新名称。新名称不能以pg_开头,因为这些名称被保留用于系统模式。

newowner

该模式的新的拥有者。

兼容性

SQL标准中没有ALTER SCHEMA 语句。

另见

[CREATE SCHEMA](#), [DROP SCHEMA](#)

Parent topic: [SQL Command Reference](#)

ALTER SEQUENCE

更改一个序列发生器的定义。

概要

```
ALTER SEQUENCE [ IF EXISTS ] name [INCREMENT [ BY ] increment]  
  [MINVALUE minvalue | NO MINVALUE]  
  [MAXVALUE maxvalue | NO MAXVALUE]  
  [START [ WITH ] start ]  
  [RESTART [ [ WITH ] restart] ]  
  [CACHE cache] [[ NO ] CYCLE]  
  [OWNED BY {table.column | NONE}]
```

```
ALTER SEQUENCE [ IF EXISTS ] name OWNER TO new_owner
```

```
ALTER SEQUENCE [ IF EXISTS ] name RENAME TO new_name
```

```
ALTER SEQUENCE [ IF EXISTS ] name SET SCHEMA new_schema
```

描述

ALTER SEQUENCE 更改一个现有序列发生器的参数。任何没有被明确设置的参数在 ALTER SEQUENCE 命中，都要维持他们之前的设置。

用户必须拥有该序列才能使用ALTER SEQUENCE。要更改一个序列的模式，用户还必须拥有新模式上的CREATE特权。注意超级用户自动拥有所有的特权。

要改变所有者，用户必须是新角色的直接或者间接成员，而且新角色必须在序列的模式上拥有CREATE权限 (这些限制迫使更改所有者不能执行删除和重新创建序列所无法做的任何事情。但是，超级用户仍然可以更改任何序列的所有权。)

参数

name

要修改的序列的名称（可选方案限定）。

IF EXISTS

如果序列不存在不会抛出错误，而只会触发一次提醒。

increment

子句 INCREMENT BY *increment* 是可选的。一个正值将产生一个上升序列，一个负值会产生一个下降序列。如果未被指定，则旧的增量值将被保持。

minvalue

NO MINVALUE

可选的子句 MINVALUE *minvalue* 决定一个序列能产生的最小值。如果 NO MINVALUE 被指定, 上升序列和下降序列的默认值分别是 1 和 -263-1。如果这些选项都没有被指定，将保持当前的 最小值。

maxvalue

NO MAXVALUE

可选子句 MAXVALUE *maxvalue* 决定一个序列能产生的最大值。如果 NO MAXVALUE 被指定, 上升序列和下降序列的默认值分别是 263-1 和 -1。如果这些选项都没有被指定，将保持当前的最大值。

start

可选子句 START WITH *start* 更改记录的序列起始值。这对 *current* 序列值没有影响。它只是设置将来的 ALTER SEQUENCE RESTART 命令将使用的值。

restart

可选子句 RESTART [WITH *restart*] 更改序列的当前值。这等效于使用 *is_called* = false 调用 setval(*sequence*, *start_val*, *is_called*) 函数。指定的值将由 nextval(*sequence*) 函数的下一次调用返回。在没有 *restart* 值的情况下写入 RESTART 等同于提供由 CREATE SEQUENCE 记录或由 ALTER SEQUENCE START WITH 最后设置的开始值。

new_owner

序列新所有者的用户名。

cache

CACHE *cache* 子句中使序列号可以预先分配并存储在内存中，以加快访问速度。最小值为1（一次只能生成一个值，即没有高速缓存）。如果未指定，则将保留旧的缓存值

CYCLE

可选的 CYCLE 关键字可用于在序列由升序或降序达

到`maxvalue`或`minvalue`时使序列回绕。如果达到限制，则生成的下一个数字将是各自的`minvalue`或`maxvalue`。

NO CYCLE

如果指定了可选的NO CYCLE关键字，则在序列达到最大值后，对`nextval()`的任何调用都将返回错误。如果未指定CYCLE或NO CYCLE，则将保留旧的循环行为。

OWNED BY *table.column*

OWNED BY NONE

OWNED BY选项使序列与特定的表列相关联，这样，如果该列（或其整个表）被删除，该序列也将被自动删除。如果指定，则此关联替换该序列的任何先前指定的关联。指定的表必须具有相同的所有者，并且与序列具有相同的架构。指定OWNED BY NONE会删除任何现有的表列关联。

new_name

序列的新名称。

new_schema

序列的新模式。

注意

为了避免阻塞从同一序列中获取数字的并发事务，永远不会回滚ALTER SEQUENCE对序列生成参数的影响。这些更改将立即生效，并且不可逆。但是，OWNED BY，OWNER TO，RENAME TO 和SET SCHEMA子句是普通的目录更新，可以回滚。

ALTER SEQUENCE 不会立即影响除当前会话以外的具有预分配（缓存）序列值的会话中的`nextval()`结果。在注意到更改的序列生成参数之前，它们将用尽所有缓存的值。当前会话将立即受到影响。

由于历史原因，ALTER TABLE 也可以与序列一起使用。但是序列允许的ALTER TABLE的唯一变体与上述形式等效。

示例

重启一个被称为serial的序列在105：

```
ALTER SEQUENCE serial RESTART WITH 105;
```

兼容性

ALTER SEQUENCE符合SQL标准，START WITH, OWNED BY, OWNER TO, RENAME TO和 SET SCHEMA子句除外，它们是MPP数据库的扩展。

另见

[CREATE SEQUENCE](#), [DROP SEQUENCE](#), [ALTER TABLE](#)

Parent topic: [SQL Command Reference](#)

ALTER SERVER

修改一个外部服务器的定义。

g概要

```
ALTER SERVER server_name [ VERSION 'new_version' ]  
    [ OPTIONS ( [ ADD | SET | DROP ] option ['value] [, ... ] ) ]
```

```
ALTER SERVER server_name OWNER TO new_owner
```

```
ALTER SERVER server_name RENAME TO new_name
```

Description

ALTER SERVER更改外部服务器的定义。命令的第一种形式更改服务器的版本字符串或通用选项。MPP数据库需要至少一个子句。该命令的第二和第三种形式更改服务器的所有者或名称。

要更改服务器，您必须是服务器的所有者。要更改所有者，用户必须：

- 拥有服务器。
- 成为新拥有角色的直接或间接成员。
- 在服务器的外部数据包装程序上具有USAGE特权。

超级用户自动满足所有这些条件。

参数

server_name

现有服务器的名称。

new_version

新的服务器版本。

OPTIONS ([ADD | SET | DROP] *option* ['*value*'] [, ...])

更改服务器的选项。ADD, SET和DROP指定要执行的操作。如果未明确指定任何操作，则默认操作为ADD。选项名称必须唯一。MPP数据库使用服务器的外部数据包装器库来验证名称和值。

OWNER TO *new_owner*

指定外部服务器的新所有者。

RENAME TO *new_name*

指定外部服务器的新名称。

Examples

通过添加连接选项来更改名为foo的服务器的定义：

```
ALTER SERVER foo OPTIONS (host 'foo', dbname 'foodb');
```

为名为foo的服务器更改名为host的选项，并设置服务器版本：

```
ALTER SERVER foo VERSION '9.1' OPTIONS (SET host 'baz');
```

兼容性

ALTER SERVER 符合 ISO/IEC 9075-9 (SQL/MED)。OWNER TO 和 RENAME 选项是MPP数据库的扩展。

另见

[CREATE SERVER, DROP SERVER](#)

Parent topic: [SQL Command Reference](#)

ALTER TABLE

更改一个表的定义。

概要

```
ALTER TABLE [IF EXISTS] [ONLY] name  
    action [, ... ]
```

```
ALTER TABLE [IF EXISTS] [ONLY] name  
    RENAME [COLUMN] column_name TO new_column_name
```

```
ALTER TABLE [ IF EXISTS ] [ ONLY ] name  
    RENAME CONSTRAINT constraint_name TO new_constraint_name
```

```
ALTER TABLE [IF EXISTS] name  
    RENAME TO new_name
```

```
ALTER TABLE [IF EXISTS] name  
    SET SCHEMA new_schema
```

```
ALTER TABLE ALL IN TABLESPACE name [ OWNED BY role_name [, ... ] ]  
    SET TABLESPACE new_tablespace [ NOWAIT ]
```

```
ALTER TABLE [IF EXISTS] [ONLY] name SET
  WITH (REORGANIZE=true|false)
  | DISTRIBUTED BY ({column_name [opclass]} [, ... ] )
  | DISTRIBUTED RANDOMLY
  | DISTRIBUTED REPLICATED
```

```
ALTER TABLE name
  [ ALTER PARTITION { partition_name | FOR (RANK(number))
  | FOR (value) } partition_action [...] ]
  partition_action
```

其中 *action* 是下列之一：

```
ADD [COLUMN] column_name data_type [ DEFAULT default_expr ]
  [column_constraint [ ... ]]
  [ COLLATE collation ]
  [ ENCODING ( storage_directive [,...] ) ]
DROP [COLUMN] [IF EXISTS] column_name [RESTRICT | CASCADE]
ALTER [COLUMN] column_name [ SET DATA ] TYPE type [COLLATE collation] [USING expr]
ALTER [COLUMN] column_name SET DEFAULT expression
ALTER [COLUMN] column_name DROP DEFAULT
ALTER [COLUMN] column_name { SET | DROP } NOT NULL
ALTER [COLUMN] column_name SET STATISTICS integer
ALTER [COLUMN] column SET ( attribute_option = value [, ... ] )
ALTER [COLUMN] column RESET ( attribute_option [, ... ] )
ADD table_constraint [NOT VALID]
ADD table_constraint_using_index
VALIDATE CONSTRAINT constraint_name
DROP CONSTRAINT [IF EXISTS] constraint_name [RESTRICT | CASCADE]
DISABLE TRIGGER [trigger_name | ALL | USER]
ENABLE TRIGGER [trigger_name | ALL | USER]
CLUSTER ON index_name
SET WITHOUT CLUSTER
SET WITH OIDS
SET WITHOUT OIDS
SET (storage_parameter = value)
RESET (storage_parameter [, ... ])
INHERIT parent_table
NO INHERIT parent_table
OF type_name
NOT OF
OWNER TO new_owner
SET TABLESPACE new_tablespace
```

```
SET TABLESPACE new_tablespace
```

where *partition_action* is one of:

```
ALTER DEFAULT PARTITION
DROP DEFAULT PARTITION [IF EXISTS]
DROP PARTITION [IF EXISTS] { partition_name |
    FOR (RANK(number)) | FOR (value) } [CASCADE]
TRUNCATE DEFAULT PARTITION
TRUNCATE PARTITION { partition_name | FOR (RANK(number)) |
    FOR (value) }
RENAME DEFAULT PARTITION TO new_partition_name
RENAME PARTITION { partition_name | FOR (RANK(number)) |
    FOR (value) } TO new_partition_name
ADD DEFAULT PARTITION name [ ( subpartition_spec ) ]
ADD PARTITION [partition_name] partition_element
    [ ( subpartition_spec ) ]
EXCHANGE PARTITION { partition_name | FOR (RANK(number)) |
    FOR (value) } WITH TABLE table_name
    [ WITH | WITHOUT VALIDATION ]
EXCHANGE DEFAULT PARTITION WITH TABLE table_name
    [ WITH | WITHOUT VALIDATION ]
SET SUBPARTITION TEMPLATE (subpartition_spec)
SPLIT DEFAULT PARTITION
    { AT (list_value)
    | START([datatype] range_value) [INCLUSIVE | EXCLUSIVE]
      END([datatype] range_value) [INCLUSIVE | EXCLUSIVE] }
    [ INTO ( PARTITION new_partition_name,
      PARTITION default_partition_name ) ]
SPLIT PARTITION { partition_name | FOR (RANK(number)) |
    FOR (value) } AT (value)
    [ INTO (PARTITION partition_name, PARTITION partition_name)]
```

where *partition_element* is:

```
VALUES (list_value [,...])  
| START ([datatype] 'start_value') [INCLUSIVE | EXCLUSIVE]  
  [ END ([datatype] 'end_value') [INCLUSIVE | EXCLUSIVE] ]  
| END ([datatype] 'end_value') [INCLUSIVE | EXCLUSIVE]  
[ WITH ( partition_storage_parameter=value [, ... ] ) ]  
[ TABLESPACE tablespace ]
```

where *subpartition_spec* is:

```
subpartition_element [, ...]
```

and *subpartition_element* is:

```
DEFAULT SUBPARTITION subpartition_name  
| [SUBPARTITION subpartition_name] VALUES (list_value [,...])  
| [SUBPARTITION subpartition_name]  
  START ([datatype] 'start_value') [INCLUSIVE | EXCLUSIVE]  
  [ END ([datatype] 'end_value') [INCLUSIVE | EXCLUSIVE] ]  
  [ EVERY ( [number | datatype] 'interval_value') ]  
| [SUBPARTITION subpartition_name]  
  END ([datatype] 'end_value') [INCLUSIVE | EXCLUSIVE]  
  [ EVERY ( [number | datatype] 'interval_value') ]  
[ WITH ( partition_storage_parameter=value [, ... ] ) ]  
[ TABLESPACE tablespace ]
```

where *storage_parameter* is:

```
appendoptimized={TRUE|FALSE}  
blocksize={8192-2097152}  
orientation={COLUMN|ROW}  
compresstype={ZLIB|ZSTD|QUICKLZ|RLE_TYPE|NONE}  
compresslevel={0-9}  
fillfactor={10-100}  
oids[=TRUE|FALSE]
```

描述

ALTER TABLE 更改一个表的定义。下文描述了几种形式：

- **ADD COLUMN** — 向表中增加一个新列，使用和**CREATE TABLE**相同的语义。ENCODING 子句只有在追加和列存储表中有效
- **DROP COLUMN [IF EXISTS]** — 从表中删除列。请注意，如果删除用作MPP数据库分配键的表列，则表的分配策略将更改为DISTRIBUTED RANDOMLY.。涉及该列的索引和表约束也会自动删除。如果表外的任何内容都取决于列（例如视图），则需要指定CASCADE。如果指定了IF EXISTS且该列不存在，则不会引发任何错误；而是发出通知。
- **IF EXISTS** — 如果表不存在，请不要抛出错误。在这种情况下发出通知。
- **SET DATA TYPE** — 此表单更改表的列的数据类型。请注意，您不能更改用作分发键或分区键的列数据类型。通过重新解析最初提供的表达式，涉及该列的索引和简单表约束将自动转换为使用新的列类型。可选的COLLATE子句为新列指定排序规则，如果省略，则排序规则是新列类型的默认排序规则。可选的USING子句指定如何从旧的值计算新的列值。如果省略，则默认转换与从旧数据类型到新数据类型的转换相同。如果没有从旧类型转换为新类型的隐式或赋值，则必须提供USING子句。
Note: 仅当查询中的所有列使用相同的排序规则时，GPORCA才支持排序规则。如果查询中的列使用不同的排序规则，则MPP使用Postgres Planner。
- **SET/DROP DEFAULT** — 设置或删除列的默认值。默认值仅适用于后续的INSERT或UPDATE命令。它们不会导致表中已有的行发生更改。
- **SET/DROP NOT NULL** — 更改是将列标记为允许空值还是拒绝空值。当列不包含空值时，只能使用SET NOT NULL。
- **SET STATISTICS** — 为后续的ANALYZE操作设置每个列的统计信息收集目标。可以在100到10000的范围内设置目标，也可以设置为-1以使用系统默认统计信息目标（default_statistics_target）恢复为目标。
- **SET (attribute_option = value [, ...])**
RESET (attribute_option [, ...]) — 设置或重置每个属性选项。当前，唯一定义的按属性的选项是 n_distinct 和 n_distinct_inherited，它们覆盖了后续

ANALYZE 操作所做的不同值估计数。 `n_distinct` 影响表本身的统计信息，而 `n_distinct_inherited` 影响表及其继承子级收集的统计信息。当设置为正值时，**ANALYZE** 将假定该列恰好包含指定数量的不同非空值。当设置为负值（必须大于或等于 -1）时，**ANALYZE** 将假定列中不同的非空值的数量在表的大小中是线性的；确切的计数应通过将估计的表大小乘以给定数字的绝对值来计算。例如，值 -1 表示该列中的所有值都是不同的，而值 -0.5 表示每个值平均出现两次。当表的大小随时间变化时，这很有用，因为直到查询计划时间才执行表中行数的乘法。将值指定为 0 可恢复为通常估计不同值的数量

- **ADD table_constraint [NOT VALID]** — 使用与 **CREATE TABLE** 相同的语法向表（不仅仅是分区）添加新约束。当前仅将 **NOT VALID** 选项用于外键和 **CHECK** 约束。如果约束标记为 **NOT VALID**，则 MPP 数据库将跳过可能冗长的初始检查，以验证表中的所有行均满足约束。约束将仍然针对后续的插入或更新（即，对于外键而言，除非在引用表中有匹配的行，否则它们将失败；对于外键，除非新行与指定的检查匹配，否则它们将失败）约束。但是，除非使用 **VALIDATE CONSTRAINT** 选项对其进行验证，否则数据库将不假定该约束对表中的所有行均有效。创建表时将跳过约束检查，因此 **CREATE TABLE** 语法不包括此选项。
- **VALIDATE CONSTRAINT** — 该形式通过扫描表以确保没有不满足该约束的行，从而验证了以前创建为 **NOT VALID** 的外键约束。如果约束已被标记为有效，则什么也不会发生。将验证与约束的初始创建分开的好处是，与约束创建相比，验证对表的锁定更少。
- **ADD table_constraint_using_index** — 根据现有的唯一索引将新的 **PRIMARY KEY** 或 **UNIQUE** 约束添加到表中。索引的所有列都将包含在约束中。索引不能具有表达式列，也不能是部分索引。另外，它必须是具有默认排序顺序的 b 树索引。这些限制确保索引等于由常规 **ADD PRIMARY KEY** 或 **ADD UNIQUE** 命令建立的索引。
如果指定了 **PRIMARY KEY**，并且索引的列尚未标记为 **NOT NULL**，则此命令将尝试对每个此类列执行 **ALTER COLUMN SET NOT NULL**。这需要全表扫描，以验证列不包含空值。在所有其他情况下，这是一个快速的操作。如果提供了约束名称，那么索引将被重命名以匹配约束名称。否则，约束将被命名为与索引相同。
执行此命令后，索引将由约束“拥有”，就像使用常规 **ADD PRIMARY**

KEY或ADD UNIQUE命令构建索引一样。特别是，删除约束将使索引也消失。

Note: 在需要添加新约束而不长时间阻止表更新的情况下，使用现有索引添加约束可能会有所帮助。为此，请使用 CREATE INDEX CONCURRENTLY 创建索引，然后使用此语法将其安装为正式约束。请参见下面的示例。

- **DROP CONSTRAINT [IF EXISTS]** — 将指定的约束放在表上。如果指定了IF EXISTS且该约束不存在，则不会引发任何错误。在这种情况下，将发出通知。
- **DISABLE/ENABLE TRIGGER** — 禁用或启用属于该表的触发器。禁用的触发器对于系统仍然是已知的，但是在其触发事件发生时不会执行。对于延迟的触发器，将在事件发生时而不是在实际执行触发器功能时检查启用状态。可以禁用或启用由名称指定的单个触发器，或表上的所有触发器，或仅由用户创建的触发器。禁用或启用约束触发器需要超级用户特权。

Note: MPP数据库中不支持触发器。由于MPP数据库的并行性，触发器通常具有非常有限的功能。

- **CLUSTER ON/SET WITHOUT CLUSTER** — 选择或删除默认索引以用于将来的CLUSTER操作。它实际上并没有重新群集表。请注意，建议不要使用CLUSTER对MPP数据库中的表进行物理重新排序，因为它会花费很长时间。最好使用CREATE TABLE AS重新创建表并按索引列对其进行排序。

Note: 追加优化表不支持CLUSTER ON。

- **SET WITH OIDS** — 往表中添加一个系统列 oid。如果表中已有OID列则不会做任何操作，这与ADD COLUMN oid oid的操作并不相等，ADD COLUMN oid oid会增加一个正常的名为oid的正常列，而不是增加一个系统列。OIDs不允许增加在分区表或者追加优化列组织表。

Warning: MPP强烈建议您在创建表时不要启用OIDs。在大型表（例如典型的MPP数据库系统中的表）上，对表行使用OID可能会导致32位OID计数器的折回。一旦计数器回绕，就不能再认为OID是唯一的，这不仅使OID对用户应用程序无用，而且还会在MPP数据库系统目录表中引起问题。此外，从表中排除OID会使每行将表存储在磁盘上所需的空间减少了每行4个字节，从而略微提高了性能。

- **SET WITHOUT OIDS** — 从表中删除OID系统列。
-

- **SET (FILLFACTOR = *value*) / RESET (FILLFACTOR)** — 更改表的填充因子。表格的填充系数是10到100之间的百分比。默认值为100（完全打包）。当指定较小的填充因子时，INSERT操作仅将表页面打包到指定的百分比；每个页面上的剩余空间都保留用于更新该页面上的行。这样，UPDATE就有机会将行的更新副本与原始副本放在同一页面上，这比将其放在另一页面上更为有效。对于一个条目从不更新的表，完全打包是最佳选择，但在更新频繁的表中，较小的填充因子是合适的。请注意，此命令不会立即修改表内容。您将需要重写表以获得所需的效果。可以使用VACUUM或强制表重写的ALTER TABLE形式之一来完成。
- **SET DISTRIBUTED** — 更改表的分配策略。更改哈希分发策略，或更改为复制策略或从复制策略更改将导致表数据在磁盘上进行物理重新分发，这可能会占用大量资源。
- **INHERIT *parent_table* / NO INHERIT *parent_table*** — 添加或删除目标表作为指定父表的子表。对父级的查询将包括其子表的记录。要作为子项添加，目标表必须已经包含与父项相同的所有列（它也可以具有其他列）。这些列必须具有匹配的数据类型，并且如果它们在父级中具有NOT NULL约束，那么它们在子级中也必须具有NOT NULL约束。对于父级的所有CHECK约束，还必须有匹配的子表约束，但在父级中标记为不可继承的（即用ALTER TABLE ... ADD CONSTRAINT ... NO INHERIT创建）的约束除外。匹配的所有子表约束均不得标记为不可继承。当前不考虑UNIQUE, PRIMARY KEY和 FOREIGN KEY约束，但是将来可能会改变。
- **OF *type_name*** — 这种形式将表链接到复合类型，就像CREATE TABLE OF已经形成了它一样。该表的列名和类型列表必须与组合类型的列表完全匹配；oid系统列的存在可以不同。该表不得从任何其他表继承。这些限制确保 CREATE TABLE OF将允许等效的表定义。
- **NOT OF** — 这种形式将类型化表与其类型分离。
- **OWNER** — 将表，序列或视图的所有者更改为指定的用户。
- **SET TABLESPACE** — 将表的表空间更改为指定的表空间，并将与表关联的数据文件移动到新表空间。表中的索引（如果有）不会移动；但是可以使用其他SET TABLESPACE命令分别移动它们。可以使用ALL IN TABLESPACE表单移动表空间中当前数据库中的所有表，该表单将锁定所有

要先移动的表，然后再移动每个表。此表单还支持OWNED BY，该操作仅移动指定角色所拥有的表。如果指定了NOWAIT选项，则如果该命令无法立即获取所有必需的锁，则该命令将失败。请注意，此命令不会移动系统目录，请根据需要使用ALTER DATABASE或显式ALTER TABLE调用。

information_schema关系不视为系统目录的一部分，将被移动。另请参见CREATE TABLESPACE。如果更改分区表的表空间，则所有子表分区也将移至新表空间。

- **RENAME** — 更改表（或索引，序列或视图）的名称，表中单个列的名称或表的约束的名称。对存储的数据没有影响。请注意，MPP数据库分发密钥列不能重命名。
- **SET SCHEMA** — 将表移到另一个架构。表列拥有的关联索引，约束和序列也将移动。
- **ALTER PARTITION | DROP PARTITION | RENAME PARTITION | TRUNCATE PARTITION | ADD PARTITION | SPLIT PARTITION | EXCHANGE PARTITION | SET SUBPARTITION TEMPLATE** — 更改分区表的结构。在大多数情况下，您必须遍历父表才能更改其子表分区之一。

Note: 如果将分区添加到具有子分区编码的表中，则新分区将继承该子分区的存储指令。有关压缩设置优先级的更多信息，请参阅*MPP Database Administrator Guide*中的“使用压缩”。

除 RENAME和SET SCHEMA之外，所有作用于单个表的ALTER TABLE形式都可以组合成多个更改列表以一起应用。例如，可以在单个命令中添加几列和/或更改几列的类型。这对于大型表尤其有用，因为只需要对表进行一次遍历。

您必须拥有该表才能使用ALTER TABLE。要更改表的架构或表空间，您还必须对新架构或表空间具有CREATE特权。要将表添加为父表的新子级，您还必须拥有父表。要更改所有者，您还必须是新拥有角色的直接或间接成员，并且该角色必须对表的架构具有CREATE特权。要添加列或更改列类型或使用OF子句，您还必须对数据类型具有USAGE特权。超级用户自动具有这些特权。

Note: 如果表具有多个分区，表具有压缩功能或表的块大小很大，则内存使用量会显着增加。如果与该表关联的关系的数量很大，则这种情况可能会迫使对该

表进行的操作使用更多的内存。例如，如果该表是一个CO表并具有大量列，则每个列都是一个关系。诸如ALTER TABLE ALTER COLUMN之类的操作将打开表中的所有列，以分配关联的缓冲区。如果CO表具有40列和100个分区，并且这些列被压缩并且块大小为2 MB（系统系数为3），则系统尝试分配24 GB，即 $(40 \times 100) \times (2 \times 3)$ MB或24 GB。

参数

ONLY

仅对指定的表名执行操作。如果不使用 ONLY关键字，则将在命名表以及与该表关联的任何子表分区上执行该操作。

Note: 只允许在父表或子表中添加或删除列，或更改列的类型。父表及其后代必须始终具有相同的列和类型。

name

要更改的现有表的名称（可能是模式限定的）。如果ONLY指定，则仅更改该表。如果未指定ONLY，则更新表及其所有后代表（如果有）。

Note: 约束只能添加到整个表，不能添加到分区。由于该限制，*name*参数只能包含表名，而不能包含分区名。

column_name

新列或现有列的名称。请注意，MPP数据库分发键列必须格外小心。更改或删除这些列可以更改表的分发策略。

new_column_name

现有列的新名称。

new_name

表的新名称。

type

新列的数据类型，或现有列的新数据类型。如果更改MPP分布键列的数据类型，则只能将其更改为兼容类型（例如，text到varchar可以，但text到int则不能）。

table_constraint

表的新表约束。请注意，MPP数据库当前不支持外键约束。此外，表仅

允许一个唯一约束，并且唯一性必须在MPP数据库分发密钥内。

constraint_name

要删除的现有约束的名称。

CASCADE

自动删除依赖于已删除列或约束的对象（例如，引用该列的视图）。

RESTRICT

如果有任何相关对象，则拒绝删除列或约束。这是默认行为。

trigger_name

要禁用或启用的单个触发器的名称。 请注意，MPP数据库不支持触发器。

ALL

禁用或启用属于该表的所有触发器，包括与约束相关的触发器。 如果任何触发器是内部生成的约束触发器（例如用于实现外键约束或可延迟的唯一性和排除约束的触发器），则这需要超级用户特权。

USER

禁用或启用属于该表的所有触发器，但内部生成的约束触发器（例如用于实现外键约束或可延迟的唯一性和排除约束的触发器除外）除外。

index_name

表应标记为集群的索引名称。 请注意，建议不要使用CLUSTER对MPP数据库中的表进行物理重新排序，因为它会花费很长时间。 最好使用[CREATE TABLE AS](#)重新创建表并按索引列对其进行排序。

FILLFACTOR

设置表格的填充系数百分比。

value

FILLFACTOR参数的新值，介于10到100之间的百分比。默认值为100。

DISTRIBUTED BY ({*column_name* [*opclass*]}) | DISTRIBUTED RANDOMLY |
DISTRIBUTED REPLICATED

指定表的分发策略。 更改哈希分发策略会导致物理上重新分配表数据，这可能会占用大量资源。 如果声明相同的哈希分配策略或从哈希更改为随机分配，则除非声明SET WITH (REORGANIZE=true)，否则不会重新分配数据。

更改为复制的分发策略或从复制的分发策略更改将导致表数据被重新分发。

REORGANIZE=true|false

当哈希分配策略未更改或从哈希更改为随机分配，并且无论如何都希望重新分配数据时，请使用REORGANIZE=true。

parent_table

父表要与此表关联或取消关联。

new_owner

表的新所有者的角色名称。

new_tablespace

该表将被移动到的表空间的名称。

new_schema

表将被移动到的模式的名称。

parent_table_name

更改分区表时，顶级父表的名称。

ALTER [DEFAULT] PARTITION

如果要更改的分区比第一级分区深，请使用ALTER PARTITION子句指定要更改层次结构中的哪个子分区

DROP [DEFAULT] PARTITION

删除指定的分区。如果分区具有子分区，则子分区也会自动删除。

TRUNCATE [DEFAULT] PARTITION

截断指定的分区。如果分区具有子分区，则子分区也会自动被截断。

RENAME [DEFAULT] PARTITION

更改分区的分区名称（而不是关系名称）。分区表是使用以下命名约定创建的：<parentname>_<level>_prt_<partition_name>.

ADD DEFAULT PARTITION

将默认分区添加到现有分区设计中。当数据与现有分区不匹配时，会将其插入默认分区。没有默认分区的分区设计将拒绝与现有分区不匹配的传入行。必须为默认分区指定名称。

ADD PARTITION

partition_element - 使用表（范围或列表）的现有分区类型，定义要添加的新分区的边界。

name - 此新分区的名称。

VALUES - 对于列表分区，定义分区将包含的值。

START - 对于范围分区，定义分区的起始范围值。默认情况下，起始值为INCLUSIVE。例如，如果您声明开始日期为“2016-01-01”，则分区将

包含所有大于或等于“2016-01-01”的日期。通常，START表达式的数据类型与分区键列的类型相同。如果不是这种情况，则必须显式转换为预期的数据类型。

END - 对于范围分区，定义分区的结束范围值。默认情况下，最终值为EXCLUSIVE。例如，如果您声明结束日期为“2016-02-01”，则分区将包含所有小于但不等于“2016-02-01”的日期。通常，END表达式的数据类型与分区键列的类型相同。如果不是这种情况，则必须显式转换为预期的数据类型。

WITH - 设置分区的表存储选项。例如，您可能希望将较旧的分区作为追加优化表，而将较新的分区作为常规堆表。有关存储选项的说明，请参见[CREATE TABLE](#)。

TABLESPACE - 要在其中创建分区的表空间的名称。

subpartition_spec - 仅允许在没有子分区模板的情况下创建的分区设计。声明要添加的新分区的子分区规范。如果分区表最初是使用子分区模板定义的，则该模板将用于自动生成子分区。

EXCHANGE [DEFAULT] PARTITION

将另一个表交换到分区层次结构中，替换为现有分区的位置。在多级分区设计中，您只能交换最低级别的分区（包含数据的分区）。

MPP数据库服务器配置参数gp_enable_exchange_default_partition控制EXCHANGE DEFAULT PARTITION子句的可用性。该参数的默认值是off。该子句不可用，如果在ALTER TABLE命令中指定了该子句，则MPP数据库将返回错误。

有关该参数的信息，请参阅[服务器配置参数](#)。

Warning: 交换默认分区之前，必须确保要交换的表中的数据（新的默认分区）对默认分区有效。例如，新的默认分区中的数据不得包含在分区表的其他叶子分区中有效的数据。否则，由GPORCA执行的具有交换的默认分区的分区表查询可能会返回错误的结果。

WITH TABLE *table_name* - 您要交换到分区设计中的表的名称。您可以交换一个表，其中表数据存储在数据库中。例如，该表是使用CREATE TABLE命令创建的。该表必须具有与父表相同的列数，列顺序，列名，列类型和分发策略。

使用EXCHANGE PARTITION子句，您还可以将可读的外部表（使用CREATE EXTERNAL TABLE命令创建）交换到分区层次结构中，而不

是现有的叶子子分区。如果您指定了可读的外部表，则还必须指定WITHOUT VALIDATION子句，以针对要交换的分区的CHECK约束跳过表验证。

如果分区表包含具有检查约束或NOT NULL约束的列，则不支持与外部表交换叶子分区。

您无法与复制表交换分区。不支持将分区与分区表或分区表的子分区交换。

WITH | WITHOUT VALIDATION - 验证表中的数据是否与您要交换的分区的CHECK约束相匹配。默认设置是根据CHECK约束验证数据。

Warning: 如果指定WITHOUT VALIDATION子句，则必须确保针对现有子叶分区交换的表中的数据对于分区上的CHECK约束是有效的。

SET SUBPARTITION TEMPLATE

修改现有分区的子分区模板。设置新的子分区模板后，所有添加的新分区将具有新的子分区设计（现有分区不会被修改）。

SPLIT DEFAULT PARTITION

分割默认分区。在多级分区中，只能拆分范围分区，而不能拆分列表分区，并且只能拆分最低级别的默认分区（包含数据的分区）。拆分默认分区将创建一个包含指定值的新分区，并保留默认分区，其中包含与现有分区不匹配的任何值。

AT - 对于列表分区表，指定一个列表值，该值应用作拆分条件。

START - 对于范围分区表，指定新分区的起始值。

END - 对于范围分区表，指定新分区的结束值。

INTO - 允许您为新分区指定名称。使用INTO子句拆分默认分区时，指定的第二个分区名称应始终为现有默认分区的名称。如果您不知道默认分区的名称，则可以使用`pg_partitions`视图进行查找。

SPLIT PARTITION

将现有分区分分为两个分区。在多级分区中，只能拆分范围分区，而不能拆分列表分区，并且只能拆分最低级别的分区（包含数据的分区）。

AT - 指定一个单一值，该值应用作分割条件。该分区将分为两个新分区，指定的分割值是后一个分区的起始范围。

INTO - 允许用户为分裂创建两个新分区指定名字。

partition_name

给定的分区名称。

FOR (RANK(number))

对于范围分区，分区在范围中的排名。

FOR ('value')

通过声明一个落在分区边界说明中的值来指定一个分区。如果用FOR声明的值匹配一个分区和它的一个子分区（例如，如果值是一个日期并且表先按月分区然后按日分区），那么FOR将在第一个找到匹配的层次上操作（例如，每月的分区）。如果用户的目的是在子分区上操作，则必须按如下方式声明：`ALTER TABLE name ALTER PARTITION FOR ('2016-10-01') DROP PARTITION FOR ('2016-10-01');`

注解

ALTER TABLE 命令中指定的表名不能是一个表中的分区名。

在修改或者删除作为MPP数据库分布键一部分的列时要特别小心，因为这可能会改变表的分布策略。

MPP数据库当前不支持外键约束。对于要在MPP数据库中实施的唯一约束，表必须被哈希分布（不能用DISTRIBUTED RANDOMLY），并且所有的分布键列必须和唯一约束列中前部的列相同。

增加CHECK或者NOT NULL约束要求扫描表以验证现有的行是否符合约束。

当使用ADD COLUMN添加列时，表中的所有现有行都使用该列的默认值初始化，如果未指定DEFAULT子句，则初始化为NULL。添加具有非空默认值的列或更改现有列的类型将需要重写整个表和索引。作为例外，如果USING子句不更改列的内容，并且旧类型可以强制转换为新类型或新类型不受限制的域，则不需要重写表，但是受影响列上的任何索引都必须仍在重建中。添加或删除系统oid列还需要重写整个表。对于大型表，表和/或索引的重建可能会花费大量时间；并且暂时需要多达两倍的磁盘空间。

您可以在单个ALTER TABLE命令中指定多个更改，这些更改将在表上一次传

递。

DROP COLUMN 表单不会物理删除列，而只是使它对SQL操作不可见。表中随后的插入和更新操作将为该列存储一个空值。因此，删除列很快，但是不会立即减小表的磁盘大小，因为删除的列所占用的空间不会被回收。随着现有行的更新，空间将随着时间的推移而回收。但是，如果删除系统oid列，则表将立即被重写。

要强制立即回收被删除的列占用的空间，您可以执行ALTER TABLE的一种形式来重写整个表。这将导致重建的每一行，并将删除的列替换为空值。

ALTER TABLE的重写形式不是MVCC安全的。在表重写之后，如果并发事务使用的是在重写发生之前拍摄的快照，则该表将对并发事务显示为空。有关更多详细信息，请参见[MVCC Caveats](#)。

SET DATA TYPE的USING选项实际上可以指定涉及该行的旧值的任何表达式。也就是说，它可以引用其他列以及要转换的列。这允许使用SET DATA TYPE语法完成非常通用的转换。由于具有这种灵活性，因此USING表达式不会应用于列的默认值（如果有）；结果可能不是默认值所需的常量表达式。这意味着当没有从旧类型到新类型的隐式或赋值转换时，即使提供了USING子句，SET DATA TYPE也可能无法转换默认值。在这种情况下，请使用DROP DEFAULT删除默认值，执行ALTER TYPE，然后使用SET DEFAULT添加合适的新默认值。类似的考虑适用于涉及该列的索引和约束。

如果表已分区或具有任何后代表，则不允许在父表中添加，重命名或更改列的类型或重命名继承的约束，而无需对后代进行相同的操作。这样可以确保后代始终具有与父代匹配的列。

要查看分区表的结构，可以使用视图[pg_partitions](#)。该视图可以帮助您识别您可能要更改的特定分区。

仅当后代不从任何其他父级继承该列并且从未对该列进行独立定义时，递归DROP COLUMN操作才会删除后代表的列。非递归DROP COLUMN（仅ALTER TABLE ONLY ... DROP COLUMN）从不删除任何后代

列，而是将它们标记为独立定义而不是继承。

TRIGGER, CLUSTER, OWNER和TABLESPACE操作永远不会递归到后代表。也就是说，它们始终像指定ONLY那样起作用。仅对未标记为NO INHERIT的CHECK约束重复添加约束。

如果包含已被交换以使用外部表的叶子分区的分区表上的数据没有更改，则支持这些ALTER PARTITION操作。否则，将返回错误。

- Adding or dropping a column.
- Changing the data type of column.

分区表不支持这些ALTER PARTITION操作，该分区表包含已被交换以使用外部表的叶子分区：

- Setting a subpartition template.
- Altering the partition properties.
- Creating a default partition.
- Setting a distribution policy.
- Setting or dropping a NOT NULL constraint of column.
- Adding or dropping constraints.
- Splitting an external partition.

不允许更改系统目录表的任何部分。

Examples

向列中添加列：

```
ALTER TABLE distributors ADD COLUMN address varchar(30);
```

重命名现有列：

```
ALTER TABLE distributors RENAME COLUMN address TO city;
```

重命名现有表：

```
ALTER TABLE distributors RENAME TO suppliers;
```

向列添加非空约束：

```
ALTER TABLE distributors ALTER COLUMN street SET NOT NULL;
```

重命名现有约束：

```
ALTER TABLE distributors RENAME CONSTRAINT zipchk TO zip_check;
```

向表及其所有子级添加检查约束：

```
ALTER TABLE distributors ADD CONSTRAINT zipchk CHECK  
(char_length(zipcode) = 5);
```

要将检查约束仅添加到表而不是其子表，请执行以下操作：

```
ALTER TABLE distributors ADD CONSTRAINT zipchk CHECK (char_length(zipcode) = 5) NO
```

(检查约束也不会被将来的子代继承。)

从表及其所有子级中删除检查约束：

```
ALTER TABLE distributors DROP CONSTRAINT zipchk;
```

仅从一个表中删除检查约束：

```
ALTER TABLE ONLY distributors DROP CONSTRAINT zipchk;
```

(对于任何继承distributors的子表，检查约束仍然存在。)

将表移动到不同的模式：

```
ALTER TABLE myschema.distributors SET SCHEMA yourschema;
```

将表的分发策略更改为已复制：

```
ALTER TABLE myschema.distributors SET DISTRIBUTED REPLICATED;
```

将新分区添加到分区表：

```
ALTER TABLE sales ADD PARTITION  
    START (date '2017-02-01') INCLUSIVE  
    END (date '2017-03-01') EXCLUSIVE;
```

向现有分区设计添加默认分区：

```
ALTER TABLE sales ADD DEFAULT PARTITION other;
```

重命名分区：

```
ALTER TABLE sales RENAME PARTITION FOR ('2016-01-01') TO  
jan08;
```

删除范围序列中的第一个（最旧的）分区：

```
ALTER TABLE sales DROP PARTITION FOR (RANK(1));
```

将表交换到用户的分区设计中：

```
ALTER TABLE sales EXCHANGE PARTITION FOR ('2016-01-01') WITH  
TABLE jan08;
```

拆分默认分区(现有的默认分区名称other)为2017年1月添加新的每月分区：

```
ALTER TABLE sales SPLIT DEFAULT PARTITION  
START ('2017-01-01') INCLUSIVE  
END ('2017-02-01') EXCLUSIVE  
INTO (PARTITION jan09, PARTITION other);
```

将每月分区分成两个分区，第一个分区包含日期1月1日至15日，第二个分区包含日期1月16日至31日：

```
ALTER TABLE sales SPLIT PARTITION FOR ('2016-01-01')  
AT ('2016-01-16')  
INTO (PARTITION jan081to15, PARTITION jan0816to31);
```

要重新创建主键约束，而在重建索引时不阻止更新：

```
CREATE UNIQUE INDEX CONCURRENTLY dist_id_temp_idx ON distributors (dist_id);  
ALTER TABLE distributors DROP CONSTRAINT distributors_pkey,  
ADD CONSTRAINT distributors_pkey PRIMARY KEY USING INDEX dist_id_temp_idx;
```

兼容性

ADD (不包含 USING INDEX), DROP, SET DEFAULT和 SET DATA TYPE (不包含 USING) 符合SQL标准。 其他形式是SQL标准的MPP数据库扩展。 同样，在单个ALTER TABLE命令中指定多个操纵的功能也是一种扩展。

ALTER TABLE DROP COLUMN可用于删除表的唯一列，而保留零列表。 这是SQL的扩展，不允许使用零列表。

另见

[CREATE TABLE](#), [DROP TABLE](#)

Parent topic: [SQL Command Reference](#)

ALTER TABLESPACE

更改表空间的定义。

概要

```
ALTER TABLESPACE name RENAME TO new_name
```

```
ALTER TABLESPACE name OWNER TO new_owner
```

```
ALTER TABLESPACE name SET ( tablespace_option = value [, ... ] )
```

```
ALTER TABLESPACE name RESET ( tablespace_option [, ... ] )
```

描述

ALTER TABLESPACE更改表空间的定义。

您必须拥有表空间才能使用ALTER TABLESPACE。要更改所有者，您还必须
是新拥有角色的直接或间接成员。（请注意，超级用户会自动拥有这些特权。
）

参数

name

一个现有表空间的名称。

new_name

新的表空间的名称。新的表空间名称不能以`pg_`或`gp_`开头。(这类名称保留用于系统表空间)。

new_owner

该表空间的新所有者。

tablespace_parameter

要设置或重置的表空间参数。当前，唯一可用的参数是`seq_page_cost`和`random_page_cost`。为特定表空间设置这两个值将覆盖计划者通常对从该表空间中的表读取页面的开销的估计，这是由同名的配置参数确定的（请参见[seq-page-cost](#), [random-page-cost](#)）。如果一

个表空间位于比I / O子系统的其余部分更快或更慢的磁盘上，则这可能很有用。

示例

重命名表空间 index_space 为 fast_raid:

```
ALTER TABLESPACE index_space RENAME TO fast_raid;
```

更改表空间index_space的所有者:

```
ALTER TABLESPACE index_space OWNER TO mary;
```

兼容性

在SQL标准中没有ALTER TABLESPACE语句。

另见

[CREATE TABLESPACE, DROP TABLESPACE](#)

Parent topic: [SQL Command Reference](#)

ALTER TEXT SEARCH CONFIGURATION

更改文本搜索配置的定义。

概要

```
ALTER TEXT SEARCH CONFIGURATION name
    ALTER MAPPING FOR token_type [, ... ] WITH dictionary_name [, ... ]
ALTER TEXT SEARCH CONFIGURATION name
    ALTER MAPPING REPLACE old_dictionary WITH new_dictionary
ALTER TEXT SEARCH CONFIGURATION name
    ALTER MAPPING FOR token_type [, ... ] REPLACE old_dictionary WITH new_dictionary
ALTER TEXT SEARCH CONFIGURATION name
    DROP MAPPING [ IF EXISTS ] FOR token_type [, ... ]
ALTER TEXT SEARCH CONFIGURATION name RENAME TO new_name
ALTER TEXT SEARCH CONFIGURATION name OWNER TO new_owner
ALTER TEXT SEARCH CONFIGURATION name SET SCHEMA new_schema
```

Description

ALTER TEXT SEARCH CONFIGURATION更改文本搜索配置的定义。您可以修改其从令牌类型到字典的映射，或者更改配置的名称或所有者。

您必须是配置的所有者才能使用ALTER TEXT SEARCH CONFIGURATION.

参数

name

现有文本搜索配置的名称（可选，模式限定）。

token_type

配置的解析器发出的令牌类型的名称。

dictionary_name

用于指定令牌类型的文本搜索字典的名称。如果列出了多个词典，则将以指定顺序对其进行查询。

old_dictionary

映射中要替换的文本搜索字典的名称。

new_dictionary

用来替换old_dictionary的文本搜索字典的名称

new_name

文本搜索配置的新名称。

new_owner

文本搜索配置的新所有者。

new_schema

文本搜索配置的新架构。

ADD MAPPING FOR表单会安装一个字典列表，供您查询指定的令牌类型；如果已经有任何令牌类型的映射，则为错误。ADD MAPPING FOR表单的功能相同，但是首先删除那些令牌类型的任何现有映射。ADD MAPPING FOR表单用*new_dictionary*替换*old_dictionary*的任何地方。当出现FOR时，仅对指定的令牌类型执行此操作，否则不进行配置的所有映射。DROP MAPPING表单删除指定令牌类型的所有字典，从而使这些类型的令牌被文本搜索配置忽略。如果没有令牌类型的映射，这是一个错误，除非出现IF EXISTS。

示例

以下示例在my_config中使用english的任何地方，用swedish词典替换english词

典。

```
ALTER TEXT SEARCH CONFIGURATION my_config
ALTER MAPPING REPLACE english WITH swedish;
```

兼容性

在SQL标准中没有ALTER TEXT SEARCH CONFIGURATION语句。

另见

[CREATE TEXT SEARCH CONFIGURATION](#), [DROP TEXT SEARCH CONFIGURATION](#)

Parent topic: [SQL Command Reference](#)

ALTER TEXT SEARCH DICTIONARY

更改文本搜索词典的定义。

概要

```
ALTER TEXT SEARCH DICTIONARY name (  
    option [ = value ] [, ... ]  
)  
ALTER TEXT SEARCH DICTIONARY name RENAME TO new_name  
ALTER TEXT SEARCH DICTIONARY name OWNER TO new_owner  
ALTER TEXT SEARCH DICTIONARY name SET SCHEMA new_schema
```

Description

ALTER TEXT SEARCH DICTIONARY更改文本搜索词典的定义。 您可以更改词典的特定于模板的选项，或更改词典的名称或所有者。

您必须是字典的所有者，才能使用ALTER TEXT SEARCH DICTIONARY。

参数

name

现有文本搜索字典的名称（可选，用模式限定）。

option

要为此字典设置的模板特定选项的名称。

value

用于模板特定选项的新值。 如果省略了等号和值，则将从字典中删除该选项的所有先前设置，从而允许使用默认值。

new_name

文本搜索字典的新名称。

new_owner

文本搜索字典的新所有者。

new_schema

文本搜索字典的新模式。

模板特定的选项可以按任何顺序出现。

示例

以下示例命令更改基于Snowball的字典的停用词列表。 其他参数保持不变。

```
ALTER TEXT SEARCH DICTIONARY my_dict ( StopWords = newrussian );
```

以下示例命令将language选项更改为dutch，并完全删除了停用词选项。

```
ALTER TEXT SEARCH DICTIONARY my_dict ( language = dutch, StopWords  
);
```

下面的示例命令“更新”字典的定义，而无需实际更改任何内容。

```
ALTER TEXT SEARCH DICTIONARY my_dict ( dummy );
```

（这样做的原因是，如果没有这样的选项，则选项删除代码不会抱怨。）此技巧在更改字典的配置文件时很有用：ALTER将强制现有的数据库会话重新读取配置文件， 否则，如果他们早些阅读它们，他们将永远做不到。

兼容性

SQL标准中没有ALTER TEXT SEARCH DICTIONARY语句。

另见

[CREATE TEXT SEARCH DICTIONARY](#), [DROP TEXT SEARCH DICTIONARY](#)

Parent topic: [SQL Command Reference](#)

ALTER TEXT SEARCH PARSER

描述

更改文本搜索解析器的定义。

概要

```
ALTER TEXT SEARCH PARSER name RENAME TO new_name  
ALTER TEXT SEARCH PARSER name SET SCHEMA new_schema
```

描述

ALTER TEXT SEARCH PARSER更改文本搜索解析器的定义。 当前，唯一支持的功能是更改解析器的名称。

您必须是超级用户才能使用ALTER TEXT SEARCH PARSER.

参数

- name*
现有文本搜索解析器的名称（可选，由模式限定）。
- new_name*
文本搜索解析器的新名称。
- new_schema*
文本搜索解析器的新架构。

兼容性

SQL标准中没有ALTER TEXT SEARCH PARSE语句。

另见

[CREATE TEXT SEARCH PARSE, DROP TEXT SEARCH PARSE](#)

Parent topic: [SQL Command Reference](#)

ALTER TEXT SEARCH TEMPLATE

描述

更改文本搜索模板的定义。

概要

```
ALTER TEXT SEARCH TEMPLATE name RENAME TO new_name  
ALTER TEXT SEARCH TEMPLATE name SET SCHEMA new_schema
```

描述

ALTER TEXT SEARCH TEMPLATE更改文本搜索解析器的定义。 当前，唯一受支持的功能是更改解析器的名称。

您必须是超级用户才能使用ALTER TEXT SEARCH TEMPLATE.

参数

- name*
现有文本搜索模板的名称（可选，由模式限定）。
- new_name*
文本搜索模板的新名称。
- new_schema*
文本搜索模板的新架构。

兼容性

SQL标准中没有ALTER TEXT SEARCH TEMPLATE语句。

另见

[CREATE TEXT SEARCH TEMPLATE](#), [DROP TEXT SEARCH TEMPLATE](#)

Parent topic: [SQL Command Reference](#)

ALTER TYPE

更改一个数据类型的定义。

概要

```
ALTER TYPE name action [, ... ]
ALTER TYPE name OWNER TO new_owner
ALTER TYPE name RENAME ATTRIBUTE attribute_name TO new_attribute_name [ CASCADE ]
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 } ]
ALTER TYPE name SET DEFAULT ENCODING ( storage_directive )
```

其中 *action*是下列选项之一：

```
ADD ATTRIBUTE attribute_name data_type [ COLLATE collation ] [ CASCADE | RESTRICT ]
DROP ATTRIBUTE [ IF EXISTS ] attribute_name [ CASCADE | RESTRICT ]
ALTER ATTRIBUTE attribute_name [ SET DATA ] TYPE data_type [ COLLATE collation ] [ ( ... ) ]
```

其中 *storage_directive*是：

```
COMPRESSTYPE={ZLIB | ZSTD | QUICKLZ | RLE_TYPE | NONE}  
COMPRESSLEVEL={0-19}  
BLOCKSIZE={8192-2097152}
```

描述

ALTER TYPE更改现有类型的定义。有几个子形式：

- **ADD ATTRIBUTE** — 使用与CREATE TYPE相同的语法向复合类型添加新属性。
- **DROP ATTRIBUTE [IF EXISTS]** — 从复合类型中删除属性。如果指定了IF EXISTS且该属性不存在，则不会引发任何错误。在这种情况下，将发出通知。
- **SET DATA TYPE** — 更改复合类型的属性的类型。
- **OWNER** — 更改类型的所有者。
- **RENAME** — 更改类型的名称或复合类型的单个属性的名称。
- **SET SCHEMA** — 将类型移动到另一个架构。
- **ADD VALUE [IF NOT EXISTS] [BEFORE | AFTER]** — 将新值添加到枚举类型。可以将新值在枚举顺序中的位置指定为现有值之一—BEFORE或AFTER。否则，新项目将添加到值列表的末尾。如果指定了IF NOT EXISTS，则该类型已经包含新值就不是错误；发出通知，但不采取其他措施。否则，如果新值已经存在，将发生错误。
- **CASCADE** — 自动将操作传播到要更改的类型的类型表及其后代。
- **RESTRICT** — 如果要更改的类型是类型表的类型，则拒绝该操作。这是默认值。

可以将ADD ATTRIBUTE, DROP ATTRIBUTE和ALTER ATTRIBUTE操作组合为多个更改列表，以并行应用。例如，可以在单个命令中添加多个属性和/或更改多个属性的类型。

您可以更改名称，所有者和类型的架构。您还可以添加或更新标量类型的存储选项。

Note: MPP数据库不支持为行或复合类型添加存储选项。

您必须拥有该类型才能使用ALTER TYPE。要更改类型的架构，您还必须对新架构具有CREATE特权。要更改所有者，您还必须是新拥有角色的直接或间接成员，并且该角色必须对类型的架构具有CREATE特权。（这些限制迫使更改所有者不能执行删除和重新创建类型的任何操作。但是，超级用户可以更改任何类型的所有权。）要添加属性或更改属性类型，还必须具有USAGE数据类型的特权。

ALTER TYPE ... ADD VALUE（向枚举类型添加新值的形式）不能在事务块内执行。

涉及增加的枚举值的比较有时会比仅涉及枚举类型的原始成员的比较慢。通常只有在使用BEFORE或AFTER设置新值的排序位置（而不是列表的末尾）时，才会发生这种情况。但是，有时即使将新值添加到末尾也会发生（如果自从最初创建枚举类型以来，OID计数器“环绕”，就会发生这种情况）。增长速度通常很小。但是，如果重要的话，可以通过删除并重新创建枚举类型，或者通过转储并重新加载数据库来获得最佳性能。

参数

name

要更改的现有类型的名称（可选的模式限定）。

new_name

类型的新名称。

new_owner

该类型的新所有者的用户名。

new_schema

类型的新架构。

attribute_name

要添加，更改或删除的属性的名称。

new_attribute_name

要重命名的属性的新名称。

data_type

要添加的属性的数据类型，或要更改的属性的新类型。

new_enum_value

要添加的属性的数据类型，或要更改的属性的新类型。

existing_enum_value

应该在枚举类型的排序顺序之前或之后立即添加新值的现有枚举值。像所有枚举文字一样，也需要用引号引起来。

storage_directive

在表列定义中指定时，标识该类型的默认存储选项。选项包括COMPRESSTYPE，COMPRESSLEVEL和BLOCKSIZE。

COMPRESSTYPE — 设置为ZLIB（默认设置），ZSTD，RLE_TYPE或QUICKLZ¹以指定使用的压缩类型。

COMPRESSLEVEL — 对于Zstd压缩，将其设置为1（最快压缩）到19（最高压缩率）之间的整数值。对于zlib压缩，有效范围是1到9。QuickLZ压缩级别只能设置为1。对于RLE_TYPE，压缩级别可以设置为从1（最快压缩）到4（最高压缩率）的整数值。缺省压缩级别为1。

BLOCKSIZE — 设置为列中每个块的大小（以字节为单位）。

BLOCKSIZE必须介于8192和2097152字节之间，并且是8192的倍数。默认块大小为32768。

Note: 在表或列级别定义的*storage_directives* 会覆盖为类型定义的默认存储选项。

示例

要重命名名为electronic_mail的数据类型：

```
ALTER TYPE electronic_mail RENAME TO email;
```

更改用户自定义类型email的所有者为joe：

```
ALTER TYPE email OWNER TO joe;
```

更改用户自定义类型email的模式为customers：

```
ALTER TYPE email SET SCHEMA customers;
```

设置或更改名为int33的用户定义类型的压缩类型和压缩级别：

```
ALTER TYPE int33 SET DEFAULT ENCODING (compresstype=zlib, compresslevel=7);
```

要将新属性添加到类型：

```
ALTER TYPE compfoo ADD ATTRIBUTE f3 int;
```

要将新值添加到特定排序位置的枚举类型：

```
ALTER TYPE colors ADD VALUE 'orange' AFTER 'red';
```

兼容性

添加和删除属性的变体是SQL标准的一部分。其他变体是MPP数据库扩展。

另见

[CREATE TYPE](#), [DROP TYPE](#)

Parent topic: [SQL Command Reference](#)

ALTER USER

更改数据库用户（角色）的定义。

概要

```
ALTER USER name RENAME TO newname
```

```
ALTER USER name SET config_parameter {TO | =} {value | DEFAULT}
```

```
ALTER USER name RESET config_parameter
```

```
ALTER USER name RESOURCE QUEUE {queue_name | NONE}
```

```
ALTER USER name RESOURCE GROUP {group_name | NONE}
```

```
ALTER USER name [ [WITH] option [ ... ] ]
```

其中`option`可以是：

```
SUPERUSER | NOSUPERUSER
| CREATEDB | NOCREATEDB
| CREATEROLE | NOCREATEROLE
| CREATEUSER | NOCREATEUSER
| CREATEEXTTABLE | NOCREATEEXTTABLE
[ ( attribute='value'[ , ... ] ) ]
    where attributes and value are:
        type='readable'|'writable'
        protocol='gpfdist'|'http'
| INHERIT | NOINHERIT
| LOGIN | NOLOGIN
| REPLICATION | NOREPLICATION
| CONNECTION LIMIT connlimit
| [ENCRYPTED | UNENCRYPTED] PASSWORD 'password'
| VALID UNTIL 'timestamp'
| [ DENY deny_point ]
| [ DENY BETWEEN deny_point AND deny_point ]
| [ DROP DENY FOR deny_point ]
```

描述

ALTER USER是ALTER ROLE的别名。 参阅 [ALTER ROLE](#)获取更多信息。

兼容性

ALTER USER语句是一个MPP数据库扩展。 SQL标准中使用用户的定义来实现。

See Also

[ALTER ROLE](#)

Parent topic: [SQL Command Reference](#)

ALTER USER MAPPING

更改外部服务器的用户映射的定义。

概要

```
ALTER USER MAPPING FOR { username | USER | CURRENT_USER | PUBLIC }  
    SERVER servername  
    OPTIONS ( [ ADD | SET | DROP ] option ['value'] [, ... ] )
```

Description

ALTER USER MAPPING更改外部服务器的用户映射的定义。

外部服务器的所有者可以为任何用户更改该服务器的用户映射。 同样，在服务器上被授予USAGE特权的用户可以更改其自己的用户名的用户映射。

参数

username

映射的用户名。 CURRENT_USER和USER与当前用户的名称匹配。
PUBLIC 用于匹配系统中所有当前和将来的用户名。

servername

用户映射的服务器名称。

OPTIONS ([ADD | SET | DROP] *option* ['*value*'] [, ...])

更改用户映射的选项。 新选项将覆盖所有先前指定的选项。 ADD, SET和DROP指定要执行的操作。 如果未明确指定任何操作，则默认操作为ADD。 选项名称必须唯一。 MPP数据库使用服务器的外部数据包装器来验证名称和值。

示例

更改用户映射bob，服务器foo的密码：

```
ALTER USER MAPPING FOR bob SERVER foo OPTIONS (SET password 'public');
```

兼容性

ALTER USER MAPPING符合ISO / IEC 9075-9 (SQL / MED)。 存在一个微妙的语法问题：标准省略了FOR 关键字。 由于CREATE USER MAPPING和DROP USER MAPPING都在相似的位置使用FOR，因此MPP Database在此处出于一致性和互操作性而与标准有所不同。

另见

CREATE USER MAPPING, DROP USER MAPPING

Parent topic: [SQL Command Reference](#)

ALTER VIEW

更改一个视图的定义。

概要

```
ALTER VIEW [ IF EXISTS ] name ALTER [ COLUMN ] column_name SET DEFAULT expression

ALTER VIEW [ IF EXISTS ] name ALTER [ COLUMN ] column_name DROP DEFAULT

ALTER VIEW [ IF EXISTS ] name OWNER TO new_owner

ALTER VIEW [ IF EXISTS ] name RENAME TO new_name

ALTER VIEW [ IF EXISTS ] name SET SCHEMA new_schema

ALTER VIEW [ IF EXISTS ] name SET ( view_option_name [= view_option_value] [, ... ] )

ALTER VIEW [ IF EXISTS ] name RESET ( view_option_name [, ... ] )
```

描述

ALTER VIEW更改视图的各种辅助属性。（如果要修改视图的定义查询，请使用CREATE OR REPLACE VIEW。）

要执行此命令，您必须是视图的所有者。要更改视图的架构，您还必须对新架构具有CREATE特权。要更改所有者，您还必须是新拥有角色的直接或间接成员，并且该角色必须在视图的架构上具有CREATE特权。这些限制迫使更改所有者不能执行删除和重新创建视图所无法做的任何事情。但是，超级用户可以更改任何视图的所有权。

参数

name

现有视图的名称（可选，由模式限定）。

IF EXISTS

如果该视图不存在，不会抛出错误，而是发出通知。

SET/DROP DEFAULT

这些表格设置或删除列的默认值。在为视图应用任何规则或触发器之前，将视图列的默认值替换为目标为视图的任何INSERT 或UPDATE命令。因此，视图的默认值将优先于基础关系中的任何默认值。

new_owner

视图的新所有者。

new_name

视图的新名称。

new_schema

视图的新架构。

SET (*view_option_name* [= *view_option_value*] [, ...])

RESET (*view_option_name* [, ...])

设置或重置视图选项。当前支持的选项是：

check_option (string)

更改视图的检查选项。该值必须是 local或cascaded。

security_barrier (boolean)

更改视图的安全屏障属性。该值必须是布尔值，例如 true或false。

注意

由于历史原因，ALTER TABLE 也可以与视图一起使用；但是，视图允许的ALTER TABLE 的唯一变体与上面显示的语句等效。

将视图myview重命名为newview：Rename the view myview to newview:

```
ALTER VIEW myview RENAME TO newview;
```

示例

要将视图foo重命名为bar：

```
ALTER VIEW foo RENAME TO bar;
```

要将默认列值附加到可更新视图：

```
CREATE TABLE base_table (id int, ts timestamptz);  
CREATE VIEW a_view AS SELECT * FROM base_table;  
ALTER VIEW a_view ALTER COLUMN ts SET DEFAULT now();  
INSERT INTO base_table(id) VALUES(1); -- ts will receive a NULL  
INSERT INTO a_view(id) VALUES(2); -- ts will receive the current time
```

兼容性

ALTER VIEW是SQL标准的MPP数据库扩展。

另见

[CREATE VIEW](#), [DROP VIEW](#) in the *MPP Database Utility Guide*

Parent topic: [SQL Command Reference](#)

ANALYZE

收集有关一个数据库的统计信息。

概要

```
ANALYZE [VERBOSE] [table [ (column [, ...] ) ] ]  
  
ANALYZE [VERBOSE] {root_partition|leaf_partition} [ (column [, ...] )]  
  
ANALYZE [VERBOSE] ROOTPARTITION {ALL | root_partition [ (column [, ...] )]}
```

描述

ANALYZE收集有关数据库中表内容的统计信息，并将结果存储在系统

表 `pg_statistic` 中。随后，MPP数据库使用这些统计信息来帮助确定最有效的查询执行计划。有关收集的表统计信息的信息，请参阅[Notes](#)。

如果不使用任何参数，则ANALYZE会收集当前数据库中每个表的统计信息。您可以指定表名称以收集单个表的统计信息。您可以指定一组列名，在这种情况下，仅收集这些列的统计信息。

ANALYZE不会收集外部表的统计信息。

对于分区表，ANALYZE在叶子分区上收集其他统计信息，即HyperLogLog (HLL) 统计信息。HLL统计信息用于得出针对分区表的查询的不同值 (NDV) 数量。

- 当汇总多个叶子分区的NDV估计值时，HLL统计信息比标准表统计信息生成更准确的NDV估计值。
- 更新HLL统计信息时，仅在已更改的叶子分区上才需要ANALYZE操作。例如，如果叶子分区数据已更改，或者叶子分区已与另一个表交换，则ANALYZE是必需的。有关更新分区表统计信息的更多信息，请参见[Notes](#)。

Important: 如果要在启用GPORCA的分区表上执行查询（默认设置），则必须使用ANALYZE或ANALYZE ROOTPARTITION命令在分区表的根分区上收集统计信息。有关收集分区表上的统计信息以及何时需要ROOTPARTITION关键字的信息，请参阅[Notes](#)。有关GPORCA的信息，请参阅中的[GPORCA概述](#)。

Note: 您还可以使用MPP数据库实用程序analyzedb更新表统计信息。

analyzedb 可以同时更新多个表的统计信息。该实用程序还可以检查表统计信息并仅在统计信息不是当前统计信息或不存在时更新统计信息。有关该实用程序的信息，请参阅*MPP Database Utility Guide*。

参数

`{ root_partition | leaf_partition } [ (column [, ...]) ]`

收集分区表的统计信息，包括HLL统计信息。HLL统计信息仅在叶子分区上收集。

ANALYZE *root_partition*, 收集所有叶子分区和根分区的统计信息。

ANALYZE *leaf_partition*, 收集有关叶子分区的统计信息。

默认情况下，如果指定叶子分区，并且所有其他叶子分区都具有统计信息，则ANALYZE更新根分区统计信息。如果不是所有叶子子分区都具有统计信息，则ANALYZE记录有关没有统计信息的叶子子分区的信息。有关何时收集根分区统计信息的信息，请参阅[Notes](#)。

ROOTPARTITION [ALL]

仅基于分区表中的数据收集分区表的根分区上的统计信息。如果可能，ANALYZE使用叶子分区统计信息生成根分区统计信息。否则，ANALYZE通过对叶子分区数据进行采样来收集统计信息。未在叶子分区上收集统计信息，仅对数据进行采样。不会收集HLL统计信息。

有关何时需要ROOTPARTITION关键字的信息，请参阅[Notes](#)。

指定ROOTPARTITION时，必须指定ALL或分区表的名称。

如果将ROOTPARTITION指定为ALL，则MPP Database会收集数据库中所有分区表的根分区的统计信息。如果数据库中没有分区表，则会返回一条消息，指出没有分区表。对于不是分区表的表，不会收集统计信息。如果使用ROOTPARTITION指定表名，并且该表不是分区表，则不会为该表收集任何统计信息，并且会返回警告消息。

ROOTPARTITION子句不适用于VACUUM ANALYZE。VACUUM ANALYZE ROOTPARTITION命令返回错误。

运行ANALYZE ROOTPARTITION的时间类似于分析具有相同数据的非分区表的时间，因为ANALYZE ROOTPARTITION仅采样叶子分区数据。

对于分区表 *sales_curr_yr*，此示例命令仅在分区表的根分区上收集统计信息。ANALYZE ROOTPARTITION *sales_curr_yr*;

此示例ANALYZE命令收集有关数据库中所有分区表的根分区的统计信息。

```
ANALYZE ROOTPARTITION ALL;
```

VERBOSE

启用显示进度消息。 启用显示进度消息。 指定时，ANALYZE 发出此信息

- 正在处理的表。
- 执行该查询以生成示例表。

- 要为其计算统计信息的列。
- 发出以收集单个列的不同统计信息的查询。
- 收集的统计信息。

table

要分析的特定表的名称（可能是模式限定的）。如果省略，则分析当前数据库中的所有常规表（而不是外部表）。

column

要分析的特定列的名称。默认为所有列。

注意

仅在明确选择外表时才进行分析。并非所有外部数据包装器都支持ANALYZE。如果表的包装器不支持ANALYZE，则该命令将显示警告并且不执行任何操作。

最好定期或在对表内容进行重大更改之后立即运行ANALYZE。准确的统计信息有助于MPP数据库选择最合适的查询计划，从而提高查询处理的速度。只读数据库的常见策略是在一天的低使用时间内每天运行一次 [VACUUM](#) 和ANALYZE。（如果有大量更新活动，这是不够的。）您可以使用gp_toolkit模式中的gp_stats_missing视图来检查缺少统计信息的表：

```
SELECT * from gp_toolkit.gp_stats_missing;
```

ANALYZE要求对目标表进行SHARE UPDATE EXCLUSIVE锁定。此锁与以下锁冲突：SHARE UPDATE EXCLUSIVE, SHARE, SHARE ROW EXCLUSIVE, EXCLUSIVE, ACCESS EXCLUSIVE。

如果您在不包含数据的表上运行ANALYZE，则不会为该表收集统计信息。例如，如果您对具有统计信息的表执行TRUNCATE操作，然后对该表运行ANALYZE，则统计信息不会更改。

对于分区表，如果分区表具有大量已分析的分区，而只有几个叶子分区具有分区，则指定要分析的表部分，根分区或子分区（叶子分区表）可能会有用 改变了。

Note: 当您使用CREATE TABLE命令创建分区表时，MPP数据库将创建您指定的表（根分区或父表），并根据您指定的分区层次结构（子表）创建表层次结构。

- 在根分区表上运行ANALYZE时，将收集所有叶子分区的统计信息。叶子子分区是MPP数据库创建供子表使用的子表层次结构中的最低级表。
- 在叶子分区上运行ANALYZE时，仅收集该叶子分区和根分区的统计信息。如果叶子分区中的数据已更改（例如，您对叶子子分区数据进行了重大更新或交换了叶子子分区），则可以在叶子子分区上运行ANALYZE来收集表统计信息。默认情况下，如果所有其他叶子分区都具有统计信息，则该命令将更新根分区统计信息。

例如，如果您在具有大量分区的分区表上收集统计信息，然后仅在几个叶子分区中更新数据，则可以仅在那些分区上运行ANALYZE来更新分区的统计信息和根分区的统计信息。

- 在不是叶子分区的子表上运行ANALYZE时，不会收集统计信息。例如，您可以创建一个分区表，其中包含2006年至2016年的分区以及每年每个月的子分区。如果您在2013年的子表上运行ANALYZE，则不会收集任何统计信息。如果您在2013年3月在叶子分区上运行ANALYZE，则仅收集该叶子分区的统计信息。

对于包含已被交换以使用外部表的叶子分区的分区表，ANALYZE不会收集外部表分区的统计信息：

- 如果在外部表分区上运行ANALYZE，则不会分析该分区。
- 如果在根分区上运行ANALYZE或ANALYZE ROOTPARTITION，则不对外部表分区进行采样，并且根表统计信息不包括外部表分区。
- 如果指定了VERBOSE子句，则会显示一条参考消息：skipping external table。

MPP数据库服务器配置参数[optimizer_analyze_root_partition](#)影响何时在分区表

的根分区上收集统计信息。如果该参数为on（默认值），则在运行ANALYZE时，不需要ROOTPARTITION关键字来收集根分区上的统计信息。在根分区上运行ANALYZE或在分区表的子叶分区上运行ANALYZE且其他子叶分区具有统计信息时，将收集根分区统计信息。如果该参数是off，则必须运行ANALYZE ROOTPARTITION来收集根分区统计信息。

ANALYZE收集的统计信息通常包括每列中一些最常用值的列表以及显示每列中近似数据分布的直方图。如果ANALYZE认为它们不重要（例如，在唯一键列中没有公共值），或者列数据类型不支持适当的运算符，则可以忽略其中一个或两个。

对于大型表，ANALYZE会从表内容中随机抽取一个样本，而不是检查每一行。这样就可以在很短的时间内分析非常大的表。但是请注意，统计信息仅是近似的，并且每次运行ANALYZE都会略有变化，即使实际的表内容没有变化。这可能会导致EXPLAIN所显示的计划者估算成本发生细微变化。在极少数情况下，这种不确定性将导致查询优化器在ANALYZE运行之间选择不同的查询计划。为了避免这种情况，请通过调整`default_statistics_target`配置参数来提高ANALYZE收集的统计信息的数量，或者通过使用ALTER TABLE ... ALTER COLUMN ... SET (n_distinct ...)（请参阅ALTER TABLE）。目标值设置最常用值列表中的最大条目数和直方图中的最大bin数。默认目标值是100，但是可以向上或向下调整该值以权衡规划器估计的准确性与ANALYZE所花费的时间以及pg_statistic中占用的空间量。特别是，将统计目标设置为零会禁用该列的统计收集。对于从未用作查询的WHERE, GROUP BY或ORDER BY子句一部分的列，执行此操作可能很有用，因为计划器将不会使用此类列的统计信息。

要分析的列中最大的统计信息目标确定为准备统计信息而采样的表行数。增加目标会导致进行ANALYZE所需的时间和空间成比例增加。

ANALYZE估计的值之一是出现在每列中的不同值的数量。因为仅检查了行的子集，所以即使使用最大可能的统计目标，此估计有时也可能非常不准确。如果此错误导致查询计划不正确，则可以手动确定更准确的值，然后与ALTER TABLE ... ALTER COLUMN ... SET STATISTICS DISTINCT一起安装（请参阅ALTER TABLE）。

当MPP数据库执行ANALYZE操作以收集表的统计信息并检测到所有采样的表数据页均为空（不包含有效数据）时，MPP数据库将显示一条消息，指出应该执行VACUUM FULL操作。如果采样页为空，则表统计信息将不准确。对表进行大量更改（例如删除大量行）后，页面将变为空。VACUUM FULL操作可删除空白页，并允许ANALYZE操作收集准确的统计信息。

如果该表没有统计信息，则服务器配置参数[gp_enable_relsizes_collection](#)将控制Postgres查询优化器使用默认统计信息文件还是使用pg_relation_size函数估计表的大小。默认情况下，如果统计信息不可用，Postgres优化器将使用默认的统计信息文件来估计行数。

示例

收集表mytable的统计信息：

```
ANALYZE mytable;
```

兼容性

SQL标准中没有ANALYZE语句。

另见

[ALTER TABLE](#), [EXPLAIN](#), [VACUUM](#), analyzedb utility in the *MPP Database Utility Guide*.

Parent topic: [SQL Command Reference](#)

BEGIN

启动事务块。

概要

```
BEGIN [WORK | TRANSACTION] [transaction_mode]
```

其中*transaction_mode*是：

```
ISOLATION LEVEL {READ UNCOMMITTED | READ COMMITTED | REPEATABLE READ |  
READ WRITE | READ ONLY  
[ NOT ] DEFERRABLE
```

描述

BEGIN启动事务块，即BEGIN命令之后的所有语句将在单个事务中执行，直到给出显式的COMMIT或ROLLBACK。默认情况下（没有BEGIN），MPP数据库以自动提交模式执行事务，即，每个语句在其自己的事务中执行，并且在语句末尾隐式执行提交（如果执行成功，否则回滚）。

在事务块中，语句的执行速度更快，因为事务启动/提交需要大量的CPU和磁盘活动。在事务中执行多个语句对于确保进行一些相关更改时的一致性也很有用：其他会话将无法看到中间状态，其中不是所有相关更新都完成。

如果指定了隔离级别，读/写模式或可延迟模式，则新事务具有那些特征，就像执行[SET TRANSACTION](#)一样。

参数

WORK

TRANSACTION

可选关键字。它们没有作用。

SERIALIZABLE

READ COMMITTED

READ UNCOMMITTED

SQL标准定义了四个事务隔离级别：READ UNCOMMITTED，READ COMMITTED，REPEATABLE READ和SERIALIZABLE。

READ UNCOMMITTED允许事务查看未提交的并发事务所做的更改。在MPP数据库中这是不可能的，因此READ UNCOMMITTED与READ COMMITTED相同。

READ COMMITTED是MPP数据库中的默认隔离级别，可确保语句只能看到在开始之前提交的行。如果在第一次执行该语句后又提交了另一个并发事务，则在一个事务中执行两次的同一条语句可能会产生不同的结果。

REPEATABLE READ隔离级别确保事务只能看到在事务开始之前提交的行。REPEATABLE READ是MPP数据库支持的最严格的事务隔离级别。由于序列化失败，必须准备使用REPEATABLE READ隔离级别的应用程序以重试事务。

SERIALIZABLE事务隔离级别确保执行多个并发事务与串行执行相同的事务产生的效果相同。如果指定SERIALIZABLE，则MPP数据库将退回到REPEATABLE READ。

指定DEFERRABLE在MPP数据库中无效，但是支持语法以与PostgreSQL兼容。仅当事务为READ ONLY且SERIALIZABLE，并且MPP数据库不

支持SERIALIZABLE事务时，才可以推迟该事务。

注解

START TRANSACTION具有与BEGIN相同的功能。

使用**COMMIT**或**ROLLBACK**终止事务块。

如果已经在事务块中，则发出BEGIN会引起警告消息。事务状态不受影响。要将事务嵌套在事务块中，请使用保存点（请参见SAVEPOINT）。

示例

要开始事务块：

```
BEGIN;
```

要以可重复读隔离级别开始事务块：

```
BEGIN TRANSACTION ISOLATION LEVEL REPEATABLE READ;
```

兼容性

BEGIN是MPP数据库语言扩展。它等效于SQL标准命令**START TRANSACTION**。

DEFERRABLE *transaction_mode*是MPP数据库语言的扩展。

附带地，BEGIN关键字在嵌入式SQL中用于不同的目的。 建议您在移植数据库应用程序时注意事务语义。

另见

[COMMIT](#), [ROLLBACK](#), [START TRANSACTION](#), [SAVEPOINT](#)

Parent topic: [SQL Command Reference](#)

CHECKPOINT

强制执行事务日志检查点。

概要

CHECKPOINT

描述

检查点是事务日志序列中的一点，所有数据文件都已在该点更新，以反映日志中的信息。 所有数据文件将刷新到磁盘。

发出该命令时，CHECKPOINT命令将强制立即执行检查点，而无需等待系统安排的常规检查点。CHECKPOINT不适合在常规操作中使用。

如果在恢复期间执行，则CHECKPOINT命令将强制重新启动点，而不是写入新的检查点。

只有超级用户可以调用CHECKPOINT。

兼容性

CHECKPOINT命令是MPP数据库扩展。

Parent topic: [SQL Command Reference](#)

CLOSE

关闭游标。

概要

```
CLOSE cursor_name
```

描述

CLOSE释放与打开的游标关联的资源。关闭游标后，不允许对其进行后续操作。不再需要游标时，应将其关闭。

当事务通过COMMIT或ROLLBACK终止时，每个非持有的打开游标都会隐式关闭。如果创建游标的事务通过ROLLBACK中止，则该游标将隐式关闭。如果创建事务成功提交，则持有的游标将保持打开状态，直到执行了显式CLOSE或客户端断开连接为止。

参数

cursor_name

要关闭的打开的游标的名称。

注解

MPP数据库没有显式的OPEN游标语句。当游标被声明时，它被认为是打开的。使用DECLARE语句声明（并打开）游标。

您可以通过查询pg_cursors系统视图来查看所有可用的游标。

如果在保存点之后关闭游标，该保存点随后会回滚，则CLOSE不会回滚；即光标保持关闭状态。

示例

关闭游标portala：

```
CLOSE portala;
```

兼容性

CLOSE完全符合SQL标准。

另见

[DECLARE](#) , [FETCH](#) , [MOVE](#)

Parent topic: [SQL Command Reference](#)

CLUSTER

根据索引对磁盘上的堆存储表进行物理重新排序。 在MPP数据库中不建议执行此操作。

概要

```
CLUSTER indexname ON tablename  
CLUSTER [VERBOSE] tablename  
CLUSTER [VERBOSE]
```

描述

CLUSTER根据索引对堆存储表进行排序。追加优化存储表不支持CLUSTER。对索引进行聚簇意味着根据索引信息在磁盘上对记录进行物理排序。如果您需要的记录随机分布在磁盘上，则数据库必须跨磁盘搜索以获取请求的记录。如果将这些记录更紧密地存储在一起，则从磁盘进行的获取将更加有序。聚集索引的一个很好的例子是在date列上，该列按日期顺序对数据进行排序。对特定日期范围的查询将导致从磁盘中有顺序地进行访问，从而利用了更快的顺序访问。

聚簇是一项一次性操作：表在随后进行更新时，更改不会被聚簇。也就是说，没有尝试根据新的或更新的行的索引顺序来存储它们。如果您愿意，可以通过再次发出命令来定期重新聚簇。将表的FILLFACTOR存储参数设置为小于100 %可以有助于在更新期间保留聚簇顺序，因为如果那里有足够的空间，则更新的行将保留在同一页上。

当使用此命令将表聚簇时，MPP数据库会记住该表在哪个索引上聚簇。形式CLUSTER *tablename*将表重新聚集在与聚簇相同的索引上。您可以使用[ALTER TABLE](#)的CLUSTER或SET WITHOUT CLUSTER形式来设置索引以用于将来的聚簇操作，或清除任何先前的设置。不带任何参数的CLUSTER会重新聚簇调用用户所拥有的当前数据库中所有先前聚簇的表，或者重新聚簇所有表（如果由超级用户调用）。这种形式的CLUSTER不能在事务块内执行。

对表进行聚簇时，将在其上获取ACCESS EXCLUSIVE锁。这样可以防止在表CLUSTER完成之前对表执行任何其他数据库操作（读取和写入）。

参数

indexname

索引名称。

VERBOSE

当每个表聚簇时，打印进度报告。

tablename

表的名称（可以由模式指定）。

注解

如果您要随机访问表中的单个行，则表中数据的实际顺序并不重要。但是，如果您倾向于访问比其他数据更多的数据，并且有一个将它们分组在一起的索引，则可以从使用CLUSTER中受益。如果要从表中请求某个范围的索引值，或者是具有多个匹配行的单个索引值，则CLUSTER会有所帮助，因为一旦索引为匹配的第一行标识了表页面，则所有其他匹配的行都可能已经在同一表页面上，因此您可以节省磁盘访问并加快查询速度。

CLUSTER可以使用对指定索引的索引扫描或顺序扫描后再排序来对表进行重新排序（如果索引是b树）。它将尝试根据优化器成本参数和可用的统计信息来选择更快的方法。

使用索引扫描时，将创建表的临时副本，该副本包含按索引顺序排列的表数据。还将创建表上每个索引的临时副本。因此，磁盘上需要的可用空间至少等于表大小和索引大小的总和。

当使用顺序扫描和排序时，还会创建一个临时排序文件，因此临时空间的峰值需求是表大小加索引大小的两倍。此方法通常比索引扫描方法快，但是如果磁盘空间需求无法忍受，则可以通过将[enable_sort](#)配置参数临时设置为off来禁用此选择。

建议在聚簇之前将[maintenance_work_mem](#)配置参数设置为一个相当大的值

(但不要大于您可以用于CLUSTER操作的RAM数量)。

因为查询优化器会记录有关表顺序的统计信息，所以建议在新的聚簇表上运行ANALYZE。否则，优化器可能会选择错误的查询计划。

由于CLUSTER会记住对哪些索引进行了聚簇，因此您可以在第一次手动对要聚簇的表进行聚簇，然后设置一个定期维护脚本，该脚本无需任何参数即可执行CLUSTER，以便定期重新聚簇所需的表。

Note: 追加优化表不支持CLUSTER。

示例

根据索引emp_ind对表employees进行聚簇：

```
CLUSTER emp_ind ON emp;
```

通过重新创建大型表并将其以正确的索引顺序加载来对大型表进行聚簇：

```
CREATE TABLE newtable AS SELECT * FROM table ORDER BY column;  
DROP table;  
ALTER TABLE newtable RENAME TO table;  
CREATE INDEX column_ix ON table (column);  
VACUUM ANALYZE table;
```

兼容性

SQL标准中没有CLUSTER语句。

另见

[CREATE TABLE AS](#) , [CREATE INDEX](#)

Parent topic: [SQL Command Reference](#)

COMMENT

定义或更改对象的注释。

概要

```
COMMENT ON
{ TABLE object_name |
  COLUMN relation_name.column_name |
  AGGREGATE agg_name (agg_type [, ...]) |
  CAST (source_type AS target_type) |
  COLLATION object_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 func_name ([[argmode] [argname] argtype [, ...]]) |
  INDEX object_name |
  LARGE OBJECT large_object_oid |
  OPERATOR operator_name (left_type, right_type) |
  OPERATOR CLASS object_name USING index_method |
  [PROCEDURAL] LANGUAGE object_name |
  RESOURCE GROUP object_name |
  RESOURCE QUEUE object_name |
  ROLE object_name |
  RULE rule_name ON table_name |
  SCHEMA object_name |
  SEQUENCE object_name |
  SERVER object_name |
  TABLESPACE object_name |
  TRIGGER trigger_name ON table_name |
  TYPE object_name |
  VIEW object_name }
IS 'text'
```

描述

COMMENT存储有关数据库对象的注释。每个对象仅存储一个注释字符串。要删除注释，请在文本字符串的位置写入NULL。删除对象时，注释将自动删除。

对于大多数对象，只有对象的所有者才能设置注释。角色没有所有者，因此COMMENT ON ROLE的规则是您必须是超级用户才能对超级用户角色进行评论，或者具有CREATEROLE特权才能对非超级用户角色添加注释。当然，超级用户可以对任何对象添加注释。

使用psql元命令\dd，\d+和\l+可以轻松提取注释。可以在psql使用的相同内置函数（即obj_description，col_description和shobj_description）之上构建其他用于提取注释的用户界面。

参数

object_name

relation_name.column_name

agg_name

constraint_name

func_name

operator_name

rule_name

trigger_name

要注释的对象的名称。表，聚合，排序规则，转换，域，外部表，函数，索引，运算符，运算符类，运算符族，序列，文本搜索对象，类型和视图的名称可以通过模式指定。在对列进行注释时，*relation_name*必须引用表，视图，组合类型或外部表。

Note: MPP数据库不支持触发器。

aggregate_type

聚合函数所基于的输入数据类型。要引用零参数聚合函数，请写*代替输入数据类型列表。

source_type

强制转换的源数据类型的名称。

target_type

强制转换的目标数据类型的名称。

argmode

函数参数的模式：IN，OUT，INOUT或VARIADIC。如果省略，则默认值为IN。注意，由于仅需要输入参数来确定函数的标识，因此COMMENT ON FUNCTION实际上并不对OUT参数给予任何关注。因此，列出IN，INOUT和VARIADIC参数就足够了。

argname

函数参数的名称。请注意，COMMENT ON FUNCTION实际上并不关注参数名称，因为仅需要参数数据类型来确定函数的身份。

argtype

函数参数的数据类型（可以由模式指定）（如果有）。

large_object_oid

大对象的OID。

Note: MPP数据库不支持PostgreSQL [大对象工具](#)来流存储在大对象结构中的用户数据。

left_type

right_type

运算符参数的数据类型（可以由模式指定）。为前缀或后缀运算符的缺少参数写NONE。

PROCEDURAL

这是一个干扰词。

text

新注释，以字符串文字形式表示；或NULL以删除注释。

注解

当前没有用于查看注释的安全机制：连接到数据库的任何用户都可以查看该数据库中对象的所有注释。对于共享对象，例如数据库，角色和表空间，注释是全局存储的，因此连接到集群中任何数据库的任何用户都可以查看共享对象的所有注释。因此，请勿在评论中添加对安全性至关重要的信息。

示例

在表mytable上添加注释：

```
COMMENT ON TABLE mytable IS 'This is my table.';
```

再次删除：

```
COMMENT ON TABLE mytable IS NULL;
```

其他示例：

```
COMMENT ON AGGREGATE my_aggregate (double precision) IS 'Computes sample variance';
COMMENT ON CAST (text AS int4) IS 'Allow casts from text to int4';
COMMENT ON COLLATION "fr_CA" IS 'Canadian French';
COMMENT ON COLUMN my_table.my_column IS 'Employee ID number';
COMMENT ON CONVERSION my_conv IS 'Conversion to UTF8';
COMMENT ON CONSTRAINT bar_col_cons ON bar IS 'Constrains column col';
COMMENT ON DATABASE my_database IS 'Development Database';
COMMENT ON DOMAIN my_domain IS 'Email Address Domain';
COMMENT ON EXTENSION hstore IS 'implements the hstore data type';
COMMENT ON FOREIGN DATA WRAPPER mywrapper IS 'my foreign data wrapper';
COMMENT ON FOREIGN TABLE my_foreign_table IS 'Employee Information in other database';
COMMENT ON FUNCTION my_function (timestamp) IS 'Returns Roman Numeral';
COMMENT ON INDEX my_index IS 'Enforces uniqueness on employee ID';
COMMENT ON LANGUAGE plpython IS 'Python support for stored procedures';
COMMENT ON LARGE OBJECT 346344 IS 'Planning document';
COMMENT ON OPERATOR ^ (text, text) IS 'Performs intersection of two texts';
COMMENT ON OPERATOR - (NONE, integer) IS 'Unary minus';
COMMENT ON OPERATOR CLASS int4ops USING btree IS '4 byte integer operators for btree';
COMMENT ON OPERATOR FAMILY integer_ops USING btree IS 'all integer operators for btree';
COMMENT ON ROLE my_role IS 'Administration group for finance tables';
COMMENT ON RULE my_rule ON my_table IS 'Logs updates of employee records';
COMMENT ON SCHEMA my_schema IS 'Departmental data';
COMMENT ON SEQUENCE my_sequence IS 'Used to generate primary keys';
COMMENT ON SERVER myserver IS 'my foreign server';
COMMENT ON TABLE my_schema.my_table IS 'Employee Information';
COMMENT ON TABLESPACE my_tablespace IS 'Tablespace for indexes';
COMMENT ON TEXT SEARCH CONFIGURATION my_config IS 'Special word filtering';
COMMENT ON TEXT SEARCH DICTIONARY swedish IS 'Snowball stemmer for Swedish language';
COMMENT ON TEXT SEARCH PARSER my_parser IS 'Splits text into words';
COMMENT ON TEXT SEARCH TEMPLATE snowball IS 'Snowball stemmer';
COMMENT ON TRIGGER my_trigger ON my_table IS 'Used for RI';
COMMENT ON TYPE complex IS 'Complex number data type';
COMMENT ON VIEW my_view IS 'View of departmental costs';
```

兼容性

SQL标准中没有COMMENT语句。

Parent topic: [SQL Command Reference](#)

COMMIT

提交当前事务。

概要

COMMIT [WORK | TRANSACTION]

描述

COMMIT提交当前事务。事务所做的所有更改对其他人都可见，并且如果发生崩溃，则保证是持久的。

参数

WORK
TRANSACTION

可选关键字。它们没有作用。

注解

使用[ROLLBACK](#)中止事务。

如果不在事务内部，则发出COMMIT不会造成任何危害，但是会引发警告消息。

示例

提交当前事务并使所有更改永久生效：

```
COMMIT;
```

兼容性

SQL标准仅指定两种形式：COMMIT和COMMIT WORK。 否则，此命令完全符合要求。

另见

[BEGIN](#) , [END](#) , [START TRANSACTION](#) , [ROLLBACK](#)

Parent topic: [SQL Command Reference](#)

COPY

在文件和表之间复制数据。

概要

```
COPY table_name [(column_name [, ...])]
FROM {'filename' | PROGRAM 'command' | STDIN}
[ [ WITH ] ( option [, ...] ) ]
[ ON SEGMENT ]
```

```
COPY { table_name [(column_name [, ...])] | (query) }
TO {'filename' | PROGRAM 'command' | STDOUT}
[ [ WITH ] ( option [, ...] ) ]
[ ON SEGMENT ]
```

其中`option`可以是以下之一：

```
FORMAT format_name
OIDS [ boolean ]
FREEZE [ boolean ]
DELIMITER 'delimiter_character'
NULL 'null_string'
HEADER [ boolean ]
QUOTE 'quote_character'
ESCAPE 'escape_character'
FORCE_QUOTE { ( column_name [, ...] ) | * }
FORCE_NOT_NULL ( column_name [, ...] )
ENCODING 'encoding_name'
FILL MISSING FIELDS
LOG ERRORS [ SEGMENT REJECT LIMIT count [ ROWS | PERCENT ] ]
IGNORE EXTERNAL PARTITIONS
```

描述

COPY在MPP数据库表和标准文件系统文件之间移动数据。COPY TO将表的内容复制到一个文件（如果在ON SEGMENT上复制，则将基于segment ID复制到多个文件），而COPY FROM将数据从文件复制到表（将数据追加到表中已有的任何内容）。COPY TO还可以复制SELECT查询的结果。

如果指定了列列表，则COPY仅将指定列中的数据复制到文件中或从文件复制。如果表中的任何列不在列列表中，则COPY FROM将为这些列插入默认值。

带有文件名的COPY指示MPP数据库master主机直接从文件读取或写入文件。该文件必须可供master主机访问，并且必须从master主机的角度指定名称。

当COPY与ON SEGMENT子句一起使用时，ON SEGMENT导致segment创建单独的面向segment的文件，这些文件保留在segment主机上。ON SEGMENT的*filename*参数采用字符串文字<SEGID>（必需），并使用绝对路径或<SEG_DATA_DIR>字符串文字。运行COPY操作时，segment ID和segment数据目录的路径将替换为字符串文字值。

使用复制表（DISTRIBUTED REPLICATED）作为源的COPY TO将创建一个文件，其中包含来自单个segment的行，以便目标文件不包含重复的行。将COPY TO与ON SEGMENT子句一起使用，并将复制表作为源，则在包含所有表行的segment主机上创建目标文件。

ON SEGMENT子句允许您将表数据复制到segment主机上的文件中，以用于诸如在集群之间迁移数据或执行备份之类的操作。可以使用诸如gpfdist之类的工具来恢复由ON SEGMENT子句创建的细分数据，这对于高速数据加载非常有用。

Warning: 建议仅对专业用户使用ON SEGMENT子句。

指定PROGRAM时，服务器将执行给定命令并从程序的标准输出中读取或写入程序的标准输入。该命令必须从服务器的角度指定，并且可由gpadmin用户执行。

当指定STDIN或STDOUT时，数据将通过客户端和master之间的连接进行传输。STDIN和STDOUT不能与ON SEGMENT子句一起使用。

如果使用SEGMENT REJECT LIMIT，则COPY FROM操作将在单行错误隔离模式下运行。在此版本中，单行错误隔离模式仅适用于输入文件中格式错误的行 - 例如，多余或缺失的属性，错误的数据类型的属性或无效的客户端编码序列。约束错误（例如违反NOT NULL，CHECK或UNIQUE约束）仍将在“全有或全无”输入模式下处理。用户可以指定可接受的错误行数（基于每个segment），之后将终止整个COPY FROM操作，并且不会加载任何行。错误行的计数是按segment而不是整个加载操作计数的。如果未达到每个segment拒绝的限制，那么将加载所有不包含错误的行，并丢弃所有错误行。要保留错误行以供进一步检查，请指定LOG ERRORS子句以捕获错误日志信息。错误信息和该行存储在MPP内部数据库中。

输出

成功完成后，COPY命令将返回以下形式的命令标签，其中*count*是复制的行数：

```
COPY count
```

如果以单行错误隔离模式运行COPY FROM命令，如果由于格式错误而未加载任何行，则将返回以下通知消息，其中*count*是拒绝的行数：

```
NOTICE: Rejected count badly formatted rows.
```

参数

table_name

现有表的名称（可以由模式指定）。

column_name

要复制的列的可选列表。如果未指定列列表，则将复制表的所有列。
当以文本格式复制时，默认情况下，bytea类型的列中的一行数据最多可为256MB。

query

SELECT或VALUES命令，其结果将被复制。 请注意，查询周围需要括号。

filename

输入或输出文件的路径名。 输入文件名可以是绝对路径或相对路径，但输出文件名必须是绝对路径。 Windows用户可能需要使用E"字符串并将路径名中使用的所有反斜杠加倍。

PROGRAM '*command*'

指定要执行的命令。 在COPY FROM中，从命令的标准输出中读取输入，而在COPY TO中，将输出写入命令的标准输入中。 必须从MPP数据库master主机系统的角度指定该*command*，并且该命令必须由MPP数据库管理员用户（gpadmin）执行。

该*command*由shell程序调用。 将参数传递给shell时，请删除或转义对shell具有特殊含义的任何特殊字符。 出于安全原因，最好使用固定的命令字符串，或者至少避免在字符串中传递任何用户输入。

当指定ON SEGMENT时，该命令必须由MPP数据库管理员用户（gpadmin）在所有MPP数据库primary segment主机上可执行。 该命令由每个MPP segment实例执行。 <SEGID>在*command*中是必需的。 有关命令语法要求和指定该子句时要复制的数据的信息，请参见ON SEGMENT子句。

STDIN

指定输入来自客户端应用程序。 STDIN不支持ON SEGMENT子句。

STDOUT

指定将输出发送到客户端应用程序。 STDOUT不支持ON SEGMENT子句。

boolean

指定是打开还是关闭所选选项。 您可以写入TRUE，ON或1以启用该选项，而可以写入FALSE，OFF或0以禁用该选项。 布尔值也可以省略，在这种情况下，假定为TRUE。

FORMAT

选择要读取或写入的数据格式：text，csv（逗号分隔值）或binary。默认为text。

oids

指定为每行复制OID。（如果为没有OID的表指定了oids，或者在复制查询的情况下，将引发错误。）

freeze

请求复制已冻结的行的数据，就像在运行VACUUM FREEZE命令之后一样。这旨在用作初始数据加载的性能选项。仅当在当前子事务中已创建或截断了要加载的表，没有打开游标并且该事务没有任何较旧的快照时，行才会被冻结。

请注意，一旦成功加载数据，所有其他会话将立即能够看到数据。这违反了MVCC可见性的常规规则，指定此选项的用户应注意可能引起的潜在问题。

delimiter

指定用于分隔文件每一行（行）中各列的字符。默认为text格式的制表符，CSV格式的逗号。这必须是一个单字节字符。使用binary格式时，不允许使用此选项。

NULL

指定表示空值的字符串。默认值为文本格式\N（反斜杠-N），以及CSV格式的空引号的空字符串。对于不想将空值与空字符串区分开的情况，甚至可能以text格式使用空字符串。使用binary格式时，不允许使用此选项。

Note: 使用COPY FROM时，与该字符串匹配的任何数据项都将存储为空值，因此您应确保使用与COPY TO相同的字符串。

header

指定文件包含标题行，其中包含文件中每一列的名称。输出时，第一行包含表中的列名，输入时，第一行被忽略。仅当使用CSV格式时才允许使用此选项。

quote

指定在引用数据值时要使用的引用字符。默认值为双引号。这必须是一个单字节字符。仅当使用CSV格式时才允许使用此选项。

escape

指定应该出现在与QUOTE值匹配的数据字符之前的字符。缺省值与QUOTE值相同（因此，如果引号字符出现在数据中，则将引号字符加

倍)。这必须是一个单字节字符。仅当使用CSV格式时才允许使用此选项。

FORCE_QUOTE

强制将引号用于每个指定列中的所有非NULL值。NULL输出从不引用。如果指定*，则在所有列中都引用非NULL值。仅在COPY TO中和使用CSV格式时才允许使用此选项。

FORCE_NOT_NULL

不要将指定列的值与空字符串匹配。在默认情况下，空字符串为空时，这意味着空值将被读取为零长度的字符串，而不是空值，即使没有引号也是如此。仅在COPY FROM中和使用CSV格式时才允许使用此选项。

ENCODING

指定文件以`encoding_name`编码。如果省略此选项，则使用当前的客户端编码。有关更多详细信息，请参见下面的注释。

ON SEGMENT

在segment主机上指定各个segment数据文件。每个文件都包含由primary segment实例管理的表数据。例如，当使用COPY TO...ON SEGMENT命令从表中将数据复制到文件时，该命令会在segment主机上为主机上的每个segment实例创建一个文件。每个文件都包含由segment实例管理的表数据。

COPY命令不会从mirror实例和segment数据文件之间复制数据。

ON SEGMENT不支持关键字STDIN和STDOUT。

<SEG_DATA_DIR>和<SEGID>字符串文字用于使用以下语法指定绝对路径和文件名：

```
COPY table [TO|FROM] '<SEG_DATA_DIR>/gpdumpname<SEGID>_suffix' ON SEGM
```

<SEG_DATA_DIR>

字符串文字，表示用于ON SEGMENT复制的segment实例数据目录的绝对路径。尖括号（<和>）是用于指定路径的字符串文字的一部分。运行COPY时，COPY用segment路径替换字符串文字。可以使用绝对路径代替<SEG_DATA_DIR>字符串文字。

<SEGID>

字符串文字，表示在复制ON SEGMENT时要复制的segment实例的

content ID号。当指定ON SEGMENT时，<SEGID>是文件名的必需部分。尖括号是用于指定文件名的字符串文字的一部分。

使用COPY TO，当运行COPY命令时，字符串文字将被segment实例的content ID替换。

使用COPY FROM，在文件名中指定segment实例content ID，然后将该文件放在segment实例主机上。每个主机上的每个primary segment实例必须有一个文件。运行COPY FROM命令时，数据从文件复制到segment实例。

当指定了PROGRAM *command*子句时，*command*中需要<SEGID>字符串文字，而<SEG_DATA_DIR>字符串文字是可选的。请参阅[示例](#)。

对于COPY FROM...ON SEGMENT命令，将数据复制到表中时将检查表分配策略。默认情况下，如果数据行违反表分发策略，则返回错误。您可以使用服务器配置参数gp_enable_segment_copy_checking禁用分发策略检查。请参阅[注释](#)。

NEWLINE

指定数据文件中使用的换行符 - LF（换行，0x0A），CR（回车，0x0D），或CRLF（回车加换行，0x0D 0x0A）。如果未指定，MPP数据库segment将通过查看接收到的第一行数据并使用遇到的第一个换行符类型来检测换行符类型。

CSV

选择逗号分隔值（CSV）模式。请参阅[CSV格式](#)。

FILL MISSING FIELDS

在TEXT和CSV的COPY FROM more中，指定FILL MISSING FIELDS会将行尾缺少数据字段的数据行设置为NULL（而不是报告错误）。空白行，具有NOT NULL约束的字段以及行尾的定界符仍然会报告错误。

LOG ERRORS

这是一个可选的子句，可以在SEGMENT REJECT LIMIT子句之前，以捕获有关具有格式错误的行的错误日志信息。

错误日志信息存储在内部，并可以通过MPP数据库内置的SQL函数gp_read_error_log()进行访问。

请参阅[注解](#)以获取有关错误日志的信息以及用于查看和管理错误日志信息的内置函数。

SEGMENT REJECT LIMIT count [ROWS | PERCENT]

在单行错误隔离模式下运行COPY FROM操作。如果输入行存在格式错误，只要在加载操作期间未在任何MPP数据库segment实例上达到拒绝限制计数，它们将被丢弃。拒绝限制计数可以指定为行数（默认）或总行数的百分比（1-100）。如果使用PERCENT，则只有在处理了参数gp_reject_percent_threshold指定的行数之后，每个segment才开始计算错误行百分比。gp_reject_percent_threshold的默认值为300行。约束错误（例如违反NOT NULL，CHECK或UNIQUE约束）仍将在“全有或全无”输入模式下处理。如果未达到限制，则将加载所有正确的行，并丢弃所有错误行。

Note: 如果未先触发或未指定SEGMENT REJECT LIMIT，则MPP数据库会限制可能包含格式错误的初始行数。如果前1000行被拒绝，则COPY操作将停止并回滚。

可以使用MPP数据库服务器配置参数gp_initial_bad_row_limit更改初始拒绝行数的限制。有关参数的信息，请参阅[服务器配置参数](#)。

IGNORE EXTERNAL PARTITIONS

从分区表复制数据时，不会从外部表的叶子分区复制数据。不复制数据时，将在日志文件中添加一条消息。

如果未指定此子句，并且MPP数据库尝试从作为外部表的叶子分区中复制数据，则返回错误。

有关指定SQL查询以从作为外部表的叶子分区中复制数据的信息，请参见下一节“注释”。

注解

COPY只能与表一起使用，而不能与外部表或视图一起使用。但是，您可以执行COPY (SELECT * FROM viewname) TO ...

当指定ON SEGMENT子句时，COPY命令不支持在COPY TO命令中指定SELECT语句。例如，不支持此命令。

```
COPY (SELECT * FROM testtbl) TO '/tmp/mytst<SEGID>' ON SEGMENT
```

COPY仅处理特定名称的表，它不会在子表之间复制数据。因此，例如COPY *table* TO显示的数据与SELECT * FROM ONLY *table*相同。但是COPY (SELECT * FROM *table*) TO ...可用于转储继承层次结构中的所有数据。

同样，要从具有作为外部表的叶子分区的分区表中复制数据，请使用SQL查询来复制数据。例如，如果表my_sales包含带有作为外部表的叶子分区，则此命令COPY my_sales TO stdout返回错误。此命令将数据发送到stdout：

BINARY关键字使所有数据以二进制格式而不是文本形式存储/读取。它比普通的文本模式要快一些，但是二进制格式的文件在计算机体系结构和MPP数据库版本之间的移植性较差。另外，如果数据为二进制格式，则不能以单行错误隔离模式运行COPY FROM。

您必须对通过COPY TO读取的表具有SELECT特权，并且对于通过COPY FROM将值插入其中的表具有插入特权。在命令中列出的列上具有列特权就足够了。

COPY命令中命名的文件由数据库服务器而不是客户端应用程序直接读取或写入。因此，它们必须位于MPP数据库master主机上（而不是客户端）或可被其访问。MPP数据库系统用户（服务器运行时使用的用户ID），而不是客户端，必须可以访问它们并对其进行读写。COPY命名文件只允许数据库超级用户使用，因为它允许读取或写入服务器有权访问的任何文件。

COPY FROM将调用任何触发器并检查目标表上的约束。但是，它不会调用重写规则。请注意，在此版本中，不针对单行错误隔离模式评估违反约束的情况。

COPY输入和输出受DateStyle影响。为了确保可移植到可能使用非默认DateStyle设置的其他MPP数据库安装，在使用COPY TO之前，应将DateStyle设置为ISO。避免转储IntervalStyle设置为sql_standard的数据也是一个好主意，因为对于IntervalStyle不同设置的服务器可能会误解负间隔值。

输入数据将根据ENCODING选项或当前客户端编码进行解释，而输出数据将以ENCODING或当前客户端编码进行编码，即使数据没有通过客户端，而是由服务器直接从文件读取或写入文件。

在文本模式下从文件复制XML数据时，服务器配置参数`xmloption`会影响对复制的XML数据的验证。如果该值为content（默认值），则将XML数据验证为XML内容片段。如果参数值为document，则将XML数据验证为XML文档。如果XML数据无效，则COPY返回错误。

默认情况下，COPY在第一个错误时停止操作。如果是COPY TO，这应该不会导致问题，但是目标表已经在COPY FROM中接收到了较早的行。这些行将不可见或不可访问，但仍会占用磁盘空间。如果故障在大型COPY FROM操作中发生，则可能会浪费大量磁盘空间。您可能希望调用VACUUM来恢复浪费的空间。另一种选择是使用单行错误隔离模式来过滤掉错误行，同时仍然加载正确的行。

运行COPY FROM...ON SEGMENT命令时，服务器配置参数`gp_enable_segment_copy_checking`控制在将数据复制到表中时是否检查表分发策略（来自表DISTRIBUTED子句）。默认设置为检查分发策略。如果数据行违反了segment实例的分配策略，则返回错误。有关该参数的信息，请参阅[服务器配置参数](#)。

COPY TO...ON SEGMENT命令生成的表数据可用于通过COPY FROM...ON SEGMENT恢复表数据。但是，使用COPY TO命令生成文件时，将根据表分发策略来分发恢复到segment的数据。如果尝试还原表数据并且在运行COPY FROM...ON SEGMENT之后更改了表分发策略，则COPY命令可能会返回表分发策略错误。

Note: 如果运行COPY FROM...ON SEGMENT，并且服务器配置参数`gp_enable_segment_copy_checking`为false，则可能需要手动重新分配表数据。请参阅ALTER TABLE子句WITH REORGANIZE。

当您指定LOG ERRORS子句时，MPP数据库将捕获读取外部表数据时发生的错误。您可以查看和管理捕获的错误日志数据。

- 使用内置的SQL函数`gp_read_error_log('table_name')`。它要求对`table_name`具有SELECT特权。本示例使用COPY命令显示加载数据到表`ext_expenses`的错误日志信息：

```
SELECT * from gp_read_error_log('ext_expenses');
```

有关错误日志格式的信息，请参阅*MPP数据库管理员手册*中的[查看错误日志中的坏行](#)。

如果`table_name`不存在，则该函数返回FALSE。

- 如果指定表存在错误日志数据，则新的错误日志数据将附加到现有错误日志数据中。错误日志信息不会复制到mirror。
- 使用内置的SQL函数`gp_truncate_error_log('table_name')`删除`table_name`的错误日志数据。它需要表所有者特权。本示例删除将数据移入表`ext_expenses`时捕获的错误日志信息：

```
SELECT gp_truncate_error_log('ext_expenses');
```

如果`table_name`不存在，则该函数返回FALSE。

指定*通配符可删除当前数据库中所有表的错误日志信息。指定字符串*.*以删除所有数据库错误日志信息，包括由于先前的数据库问题而未被删除的错误日志信息。如果指定*，则需要数据库所有者特权。如果指定*.*，则需要操作系统超级用户特权。

当不是超级用户的MPP数据库用户运行COPY命令时，该命令可以由资源队列控制。资源队列必须配置有ACTIVE_STATEMENTS参数，该参数指定分配给该队列的角色可以执行的查询数量的最大限制。MPP数据库不会将成本值或内存值应用于COPY命令，仅具有成本或内存限制的资源队列不会影响COPY命令的运行。

非超级用户只能运行以下类型的COPY命令：

- 源为stdin的COPY FROM命令
- 目标为stdout的COPY TO命令

有关资源队列的信息，请参阅*MPP数据库管理员手册*中的“使用资源队列进行资源管理”。

文件格式

COPY支持的文件格式。

文本格式

使用文本格式时，读取或写入的数据是一个文本文件，每行一行。行中的列由`delimiter_character`分隔（默认为制表符）。列值本身是由每个属性的数据类型的输出函数生成的字符串，或输入函数可接受的字符串。使用空字符串代替空列。如果输入文件的任何行包含的列比预期的多或少，则COPY FROM将引发错误。如果指定了OIDs，则将OID读取或写入为用户数据列之前的第一列。

数据文件具有两个保留字符，它们对于COPY具有特殊含义：

- 指定的分隔符（默认为制表符），用于分隔数据文件中的字段。
- UNIX样式的换行符（`\n`或`0x0a`），用于在数据文件中指定新行。强烈建议生成COPY数据的应用程序将数据换行转换为UNIX样式的换行，而不是Microsoft Windows样式的回车换行（`\r\n`或`0x0a 0x0d`）。

如果数据包含这些字符中的任何一个，则必须转义该字符，以便COPY将其视为数据而不是字段分隔符或新行。

默认情况下，对于文本格式的文件，转义字符是`\`（反斜杠），对于csv格式的文件，转义字符是`"`（双引号）。如果要使用其他转义字符，则可以使用ESCAPE AS子句来实现。确保选择一个在数据文件中任何地方都没有使用的转义字符作为实际数据值，也可以通过使用ESCAPE 'OFF'来禁用文本格式文件中的转义。

例如，假设您有一个包含三列的表，并且想使用COPY加载以下三个字段。

- 百分号 = %
- 竖线 = |
- 反斜杠 = \

您指定的`delimiter_character`为|（竖线字符），而您指定的转义字符为*（星号）。数据文件中的格式化行如下所示：

```
percentage sign = % | vertical bar = *| | backslash = \
```

请注意，如何使用星号字符（*）对作为数据一部分的管道字符进行转义。另请注意，由于我们使用的是替代转义字符，因此我们不需要转义反斜杠。

如果以下字符作为列值的一部分出现，则必须在其后加上转义字符：转义字符本身，换行符，回车符和当前定界符。您可以使用ESCAPE AS子句指定其他转义字符。

CSV格式

此格式选项用于导入和导出许多其他程序（例如电子表格）使用的逗号分隔值（CSV）文件格式。它生成并识别常见的CSV转义机制，而不是MPP数据库标准文本格式使用的转义规则。

每个记录中的值由DELIMITER字符分隔。如果值包含定界符，QUOTE字符，ESCAPE字符（默认情况下为双引号），NULL字符串，回车符或换行符，则整个值将以QUOTE字符作为前缀和后缀。在特定列中输出非NULL值时，也可以使用FORCE_QUOTE强制使用引号。

CSV格式没有区分NULL值和空字符串的标准方法。MPP数据库COPY通过引用进行处理。输出NULL作为NULL参数字符串，并且不加引号，而与NULL字符串匹配的非NULL值被加引号。例如，使用默认设置，将NULL写入未加引号的空字符串，而将空字符串数据值写入双引号（""）。读取值遵循类似的规则。您可以使用FORCE_NOT_NULL来防止对特定列进行NULL输入比较。

由于反斜杠不是CSV格式的特殊字符，因此\（数据结尾标记）也可能会显示为数据值。为避免任何误解，使用\在行上显示为单独条目的数据值会在输出上自动加引号，并且在输入上（如果加引号）也不会被解释为数据结束标记。如果要加载的文件是由另一个应用程序创建的，该文件具有未加引号的单列并且可能具有\值，则可能需要在输入文件中使用该值。

Note: 在CSV格式中，所有字符均有效。用引号括起来的空格或除DELIMITER以外的任何字符都将包含这些字符。如果从空白行填充到固定宽度的CSV行的系统中导入数据，则可能会导致错误。如果出现这种情况，在将数据导入MPP数据库之前，可能需要预处理CSV文件以删除尾随空白。

CSV格式将识别并生成带有引用值的CSV文件，这些值包含嵌入式回车符和换行符。因此，与文本格式的文件相比，文件并非严格限于每行一行

Note: 许多程序会生成奇怪的（有时是错误的）CSV文件，因此文件格式更像是约定俗成的标准文件。因此，您可能会遇到一些无法使用此机制导入的文件，并且COPY可能会生成其他程序无法处理的文件。

二进制格式

二进制格式选项使所有数据以二进制格式而不是文本形式存储/读取。它比文本和CSV格式要快一些，但是二进制格式的文件在计算机体系结构和MPP数据库版本之间的可移植性较差。同样，二进制格式是非常特定于数据类型的。例如，即使在文本格式下也可以正常工作，但从smallint列输出二进制数据并将其读入整数列将不起作用。

二进制文件格式由文件头，包含行数据的零个或多个元组和文件尾部组成。文件头和数据按网络字节序排列。

- **文件头** — 文件头由15个字节的固定字段组成，后跟可变长度的头扩展区。固定字段是：
 - **签名** — 11字节序列PGCOPY\n\377\r\n\0 - 请注意，零字节是签名的必需部分。（签名旨在方便识别非8位纯净传输所破坏的文件。此签名将通过行尾翻译过滤器，丢失的零字节，丢失的高位或奇偶性更改。）
-

- **标识字段** — 32位整数位掩码，表示文件格式的重要方面。位的编号从0 (LSB) 到31 (MSB)。请注意，该字段以网络字节序存储（最高有效字节在前），文件格式中使用的所有整数字段也是如此。保留位16-31标志关键的文件格式问题；如果读取器发现在此范围内设置了意外的位，则应该中止。保留位0-15标志向后兼容的格式问题；读者只需忽略此范围内设置的任何意外位。当前仅定义一个标志，其余标志必须为零（如果数据具有OID，则位16为1，否则为0）。
- **文件头扩展区长度** — 32位整数，文件头其余部分的字节长度，不包括其自身。当前，它为零，并且第一个元组紧随其后。将来对格式进行的更改可能会允许其他数据出现在文件头中。读取器应静默跳过任何不知道该怎么处理的文件头扩展数据。文件头扩展区被设想为包含一系列自识别块。标志字段不是要告诉读取器扩展区中的内容。文件头扩展内容的特定设计留给以后的版本。
- **元组** — 每个元组都以元组中字段数的16位整数计数开头。（当前，表中的所有元组将具有相同的计数，但可能并不总是正确的。）然后，对于元组中的每个字段重复一次，存在一个32位长的字，后跟那么多字节的字段数据。（长度字不包括自身，并且可以为零。）在特殊情况下，-1表示NULL字段值。在NULL情况下，没有值字节。
字段之间没有对齐填充或任何其他额外数据。
当前，二进制格式文件中的所有数据值都假定为二进制格式（格式代码1）。可以预期，将来的扩展可能会添加一个文件头字段，以允许指定每列格式代码。
如果文件中包含OID，则OID字段紧跟在字段计数字之后。这是一个普通字段，只是它不包括在字段计数中。特别是它有一个长度字 - 这将允许处理4字节和8字节的OID，而不会带来太多麻烦，并且如果确实需要的话，也可以将OID显示为null。
- **文件尾** — 文件尾部由一个包含-1的16位整数字组成。这很容易与元组的字段计数字区分开。如果字段计数字既不是-1也不是预期的列数，则读取器应报告错误。这提供了额外的检查，以防止以某种方式与数据不同步。

使用竖线 (|) 作为字段定界符，将表复制到客户端：

```
COPY country TO STDOUT (DELIMITER '|');
```

将数据从文件复制到country表：

```
COPY country FROM '/home/usr1/sql/country_data';
```

仅将名称以'A'开头的国家复制到文件中：

```
COPY (SELECT * FROM country WHERE country_name LIKE 'A%') TO  
'/home/usr1/sql/a_list_countries.copy';
```

使用单行错误隔离模式将数据从文件复制到sales表中并记录错误：

```
COPY sales FROM '/home/usr1/sql/sales_data' LOG ERRORS  
SEGMENT REJECT LIMIT 10 ROWS;
```

若要复制segment数据供以后使用，请使用ON SEGMENT子句。COPY TO ON SEGMENT命令的使用形式如下：

```
COPY table TO '<SEG_DATA_DIR>/gpdumpname<SEGID>_suffix' ON SEGMENT;
```

<SEGID>是必需的。但是，您可以用绝对路径替换路径中的<SEG_DATA_DIR>字符串文字。

将字符串文字<SEG_DATA_DIR>和<SEGID>传递给COPY时，运行操作时COPY将填充适当的值。

例如，如果您的mytable包含以下segment和mirror：

```
contentid | dbid | file segment location
0 | 1 | /home/usr1/data1/gpsegdir0
0 | 3 | /home/usr1/data_mirror1/gpsegdir0
1 | 4 | /home/usr1/data2/gpsegdir1
1 | 2 | /home/usr1/data_mirror2/gpsegdir1
```

运行命令：

```
COPY mytable TO '<SEG_DATA_DIR>/gpbackup<SEGID>.txt' ON SEGMENT;
```

将会产生如下行：

```
/home/usr1/data1/gpsegdir0/gpbackup0.txt
/home/usr1/data2/gpsegdir1/gpbackup1.txt
```

第一列中的内容ID是插入文件路径中的标识符（例如，上面的gpsegdir0/gpbackup0.txt）。文件是在segment上而不是在master上创建的，就像在标准COPY操作中那样。使用ON SEGMENT复制时，不会为mirror创建任何数据文件。

如果指定了绝对路径，而不是<SEG_DATA_DIR>，例如在语句中

```
COPY mytable TO '/tmp/gpdir/gpbackup_<SEGID>.txt' ON SEGMENT;
```

文件将放置在每个segment的/tmp/gpdir中。如果需要重新分发，gpfdist工具还可用于使用ON SEGMENT选项还原由COPY TO生成的数据文件。

Note: 可以使用诸如gpfdist之类的工具来还原数据。备份/还原工具不适用于使用COPY TO ON SEGMENT手动生成的文件。

本示例从lineitem表复制数据，并使用PROGRAM子句使用cat实用程序将数据添加到/tmp/lineitem_program.csv文件。该文件放置在MPP数据库master上。

```
COPY LINEITEM TO PROGRAM 'cat > /tmp/lineitem.csv' CSV;
```

本示例使用PROGRAM和ON SEGMENT子句将数据复制到segment主机上的文件中。在segment主机上，COPY命令将<SEGID>替换为segment content ID，以为segment主机上的每个segment实例创建一个文件。

```
COPY LINEITEM TO PROGRAM 'cat > /tmp/lineitem_program<SEGID>.csv' ON SEGMENT (
```

本示例使用PROGRAM和ON SEGMENT子句从segment主机上的文件复制数据。从文件复制数据时，COPY命令用segment content ID替换<SEGID>。在segment主机上，每个segment实例必须有一个文件，其中文件名包含segment主机上的segment content ID。

```
COPY LINEITEM_4 FROM PROGRAM 'cat /tmp/lineitem_program<SEGID>.csv' ON SEGMEI
```

兼容性

SQL标准中没有COPY语句。

在较早版本的MPP数据库中使用了以下语法，仍然被支持：

```

COPY table_name [(column_name [, ...])] FROM
    {'filename' | PROGRAM 'command' | STDIN}
[ [WITH]
  [ON SEGMENT]
  [BINARY]
  [OIDS]
  [HEADER]
  [DELIMITER [ AS ] 'delimiter_character']
  [NULL [ AS ] 'null string']
  [ESCAPE [ AS ] 'escape' | 'OFF']
  [NEWLINE [ AS ] 'LF' | 'CR' | 'CRLF']
  [CSV [QUOTE [ AS ] 'quote']
    [FORCE NOT NULL column_name [, ...]]
  [FILL MISSING FIELDS]
  [[LOG ERRORS]
  SEGMENT REJECT LIMIT count [ROWS | PERCENT] ]

```

```

COPY { table_name [(column_name [, ...])] | (query) } TO {'filename' | PROGRAM 'command' | S
[ [WITH]
  [ON SEGMENT]
  [BINARY]
  [OIDS]
  [HEADER]
  [DELIMITER [ AS ] 'delimiter_character']
  [NULL [ AS ] 'null string']
  [ESCAPE [ AS ] 'escape' | 'OFF']
  [CSV [QUOTE [ AS ] 'quote']
    [FORCE QUOTE column_name [, ...]] | * ]
  [IGNORE EXTERNAL PARTITIONS ]

```

请注意，在此语法中，BINARY和CSV被视为独立的关键字，而不是FORMAT选项的参数。

另见

CREATE EXTERNAL TABLE

Parent topic: [SQL Command Reference](#)

CREATE AGGREGATE

定义一个新的聚集函数

概要

argname

```
CREATE AGGREGATE name ( [ argmode ] [ ] arg_data_type [ , ... ] ) (
    SFUNC = sfunc,
    STYPE = state_data_type
    [ , SSPACE = state_data_size ]
    [ , FINALFUNC = ffunc ]
    [ , FINALFUNC_EXTRA ]
    [ , COMBINEFUNC = combinefunc ]
    [ , SERIALFUNC = serialfunc ]
    [ , DESERIALFUNC = deserialfunc ]
    [ , INITCOND = initial_condition ]
    [ , MSFUNC = msfunc ]
    [ , MINVFUNC = minvfunc ]
    [ , MSTYPE = mstate_data_type ]
    [ , MSSPACE = mstate_data_size ]
    [ , MFINALFUNC = mffunc ]
    [ , MFINALFUNC_EXTRA ]
    [ , MINITCOND = minitial_condition ]
    [ , SORTOP = sort_operator ]
)
```

CREATE AGGREGATE (SFUNC) (STYPE) (SSPACE) (FINALFUNC) (FINALFUNC_EXTRA) (COMBINEFUNC) (SERIALFUNC) (DESERIALFUNC) (INITCOND) (MSFUNC) (MINVFUNC) (MSTYPE) (MSSPACE) (MFINALFUNC) (MFINALFUNC_EXTRA) (MINITCOND) (SORTOP)


```
CREATE AGGREGATE name ( [ [ argmode ] [ argname ] arg_data_type [ , ... ] ]  
    ORDER BY [ argmode ] [ argname ] arg_data_type [ , ... ] ) (  
    SFUNC = sfunc,  
    STYPE = state_data_type  
    [ , SSSPACE = state_data_size ]  
    [ , FINALFUNC = ffunc ]  
    [ , FINALFUNC_EXTRA ]  
    [ , COMBINEFUNC = combinefunc ]  
    [ , SERIALFUNC = serialfunc ]  
    [ , DESERIALFUNC = deserialfunc ]  
    [ , INITCOND = initial_condition ]  
    [ , HYPOTHETICAL ]  
)
```

or the old syntax

```
CREATE AGGREGATE name (  
    BASETYPE = base_type,  
    SFUNC = sfunc,  
    STYPE = state_data_type  
    [ , SSSPACE = state_data_size ]  
    [ , FINALFUNC = ffunc ]  
    [ , FINALFUNC_EXTRA ]  
    [ , COMBINEFUNC = combinefunc ]  
    [ , SERIALFUNC = serialfunc ]  
    [ , DESERIALFUNC = deserialfunc ]  
    [ , INITCOND = initial_condition ]  
    [ , MSFUNC = msfunc ]  
    [ , MINVFUNC = minvfunc ]  
    [ , MSTYPE = mstate_data_type ]  
    [ , MSSPACE = mstate_data_size ]  
    [ , MFINALFUNC = mffunc ]  
    [ , MFINALFUNC_EXTRA ]  
    [ , MINITCOND = minitial_condition ]  
    [ , SORTOP = sort_operator ]  
)
```

描述

CREATE AGGREGATE定义一个新的聚合函数。MPP数据库中已经提供了一些基本且常用的聚合函数，例如count，min，max，sum，avg等。如果定义了新类型或需要尚未提供的聚合函数，则可以使用CREATE AGGREGATE提供所需的功能。

如果指定了模式名称（例如CREATE AGGREGATE myschema.myagg ...），则将在指定的模式中创建聚合函数。否则，它将在当前模式中创建。

聚合函数由其名称和输入数据类型标识。如果同一模式中的两个聚合函数对不同的输入类型进行操作，则它们可以具有相同的名称。聚合函数的名称和输入数据类型也必须与同一模式中每个普通函数的名称和输入数据类型不同。此行为与普通函数名称的重载相同。请参阅[CREATE FUNCTION](#)。

一个简单的聚合函数由一个，两个或三个普通函数（必须是IMMUTABLE函数）组成：

- 状态转换函数 *sfunc*
- 可选的最终计算函数 *ffunc*
- 可选的合并函数 *combinefunc*

这些函数的用法如下：

```
sfunc( internal-state, next-data-values ) ---> next-internal-state  
ffunc( internal-state ) ---> aggregate-value  
combinefunc( internal-state, internal-state ) ---> next-internal-state
```

MPP数据库创建一个数据类型为*stype*的临时变量，以保存聚合函数的当前内部状态。在每个输入行，将计算聚合参数值，并使用当前状态值和新参数值调用状态转换函数以计算新的内部状态值。处理完所有行后，将一次调用最终函数以计算合计返回值。如果没有最终函数，则按原样返回最终状态值。

您可以将*combinefunc*指定为优化聚合执行的方法。通过指定*combinefunc*，可以首先在segment上并行执行聚合，然后在master上并行执行。当执行两级执

行时，在segment上执行*sfunc*以生成部分聚合结果，而在master上执行*combinefunc*来聚合segment中的部分结果。如果执行单级聚合，则将所有行发送到master，并将*sfunc*应用于行。

单级聚合和二级聚合是等效的执行策略。两种类型的聚合都可以在查询计划中实现。在实现函数*combinefunc*和*sfunc*时，必须确保在segment实例上调用*sfunc*，然后在master上调用*combinefunc*产生与将所有行发送到master然后仅应用*sfunc*到行的单级聚合相同的结果。

聚合函数可以提供可选的初始条件，即内部状态值的初始值。它是作为text类型的值指定并存储在数据库中的，但是它必须是状态值数据类型的常量的有效外部表示形式。如果未提供，则状态值开始为NULL。

如果状态转换函数声明为STRICT，则不能使用NULL输入调用它。使用这种转换函数，聚合执行的行为如下。输入值为空的行将被忽略（不调用该函数，并且保留先前的状态值）。如果初始状态值为NULL，则在具有所有非空输入值的第一行，第一个参数值将替换状态值，并且在具有所有非空输入值的后续行中调用转换函数。这对于实现像max这样的聚合很有用。请注意，只有当*state_data_type*与第一个*input_data_type*相同时，此行为才可用。当这些类型不同时，必须提供一个非空的初始条件或使用一个非严格的转换函数。

如果未将状态转换函数声明为STRICT，则将在每个输入行无条件调用该状态转换函数，并且必须自己处理NULL输入和NULL转换值。这使聚合作者可以完全控制聚合对NULL值的处理。

如果将最终函数声明为STRICT，则当结束状态值为NULL时将不调用该函数；而是将自动返回NULL结果。（这是STRICT函数的正常行为。）在任何情况下，最终函数都可以选择返回NULL值。例如，当avg的最终函数看到输入行为零时，将返回NULL。

有时，将最终函数声明为不仅接受状态值，而且接受与聚合的输入值相对应的额外参数是有用的。这样做的主要原因是，如果最终函数是多态的，并且状态值的数据类型不足以固定结果类型。这些额外的参数始终以NULL的形式传递（因此，在使用FINALFUNC_EXTRA选项时，最后的函数一定不能严格），但是它

们仍然是有效的参数。最终函数可以例如使用`get_fn_expr_argtype`来标识当前调用中的实际参数类型。

聚合可以选择支持移动聚合模式，如PostgreSQL文档中的[移动聚合模式](#)中所述。这需要指定`msfunc`，`minfunc`和`mstype`函数，以及可选的`mSPACE`，`mfinalfunc`，`mfinalfunc_extra`和`minitcond`函数。除了`minvfunc`以外，这些函数的工作方式与不带`m`的相应简单聚合函数相同；它们定义了包含逆转换函数的聚合的单独实现。

参数列表中带有ORDER BY的语法会创建一种特殊的聚合类型，称为有序集聚合。或者，如果指定了HYPOTHETICAL，则创建一个假设集合。这些聚合以顺序相关的方式对排序值的组进行操作，因此指定输入排序顺序是调用的重要组成部分。而且，它们可以具有直接参数，这些参数是每个聚合仅评估一次的参数，而不是每个输入行评估一次的参数。假设集合聚合是有序集合聚合的子类，其中一些直接参数需要在数量和数据类型上与聚合参数列匹配。这允许将那些直接自变量的值作为附加的“假设”行添加到集合输入行的集合中。

有时可以通过查看索引而不是扫描每个输入行来优化单参数聚合函数（例如min或max）。如果可以如此优化此聚合，请通过指定排序运算符进行指示。基本要求是，集合必须在运算符引起的排序顺序中产生第一个元素；换一种说法：

```
SELECT agg(col) FROM tab;
```

必须等于：

```
SELECT col FROM tab ORDER BY col USING sortop LIMIT 1;
```

进一步假设聚合函数将忽略NULL输入，并且当且仅当不存在非null输入时，它才会传递NULL结果。通常，数据类型的<运算符是MIN的适当排序运算符，>是MAX的适当的排序运算符。请注意，除非指定的运算符是B树索引运算符类的“小于”或“大于”策略成员，否则优化实际上不会生效。

为了能够创建聚合函数，您必须对参数类型，状态类型和返回类型具

有USAGE特权，并且对过渡和最终函数具有EXECUTE特权。

参数

name

要创建的聚合函数的名称（可以由模式指定）。

argmode

参数的模式：IN或VARIADIC。（聚合函数不支持OUT参数。）如果省略，则默认值为IN。只有最后一个参数可以标记为VARIADIC。

argname

参数的名称。当前这仅用于文档目的。如果省略，则参数没有名称。

arg_data_type

此聚合函数在其上操作的输入数据类型。要创建零参数聚合函数，请写*代替参数说明列表。（此类汇总的一个示例是count(*)。）

base_type

在CREATE AGGREGATE的旧语法中，输入数据类型由basetype参数指定，而不是写在聚合名称旁边。请注意，此语法仅允许一个输入参数。要使用此语法定义零参数聚合函数，请将basetype指定为"ANY"（不是*）。不能使用旧语法定义有序集的聚合。

sfunc

每个输入行要调用的状态转换函数的名称。对于普通的N参数聚合函数，*sfunc*必须采用N + 1个参数，第一个参数为*state_data_type*类型，其余参数与聚合的声明输入数据类型匹配。该函数必须返回*state_data_type*类型的值。该函数获取当前状态值和当前输入数据值，并返回下一个状态值。

对于有序集（包括假设集）聚合，状态转换函数仅接收当前状态值和聚合参数，而不接收直接参数。否则它是相同的。

state_data_type

聚合状态值的数据类型。

state_data_size

聚合状态值的近似平均大小（以字节为单位）。如果省略此参数或将其设

置为零，则基于`state_data_type`使用默认估计。优化器使用此值来估计分组聚合查询所需的内存。此参数的较大值不鼓励使用哈希聚合。

ffunc

遍历所有输入行后，用于计算合计结果的最终函数的名称。该函数必须采用`state_data_type`类型的单个参数。聚集的返回数据类型定义为该函数的返回类型。如果未指定`ffunc`，则将结束状态值用作汇总结果，返回类型为`state_data_type`。

对于有序聚集（包括假设集合），最终函数不仅接收最终状态值，而且还接收所有直接参数的值。

如果指定了`FINALFUNC_EXTRA`，则除了最终状态值和任何直接参数之外，最终函数还将接收与聚合的常规（聚合）参数相对应的额外`NULL`值。当定义多态聚合时，这主要用于允许正确解析聚合结果类型。

serialfunc

仅当`state_data_type`为`internal`的聚合函数具有`serialfunc`函数时，该函数才能参与并行聚合，该函数必须将聚合状态序列化为`bytea`值以传输到另一个进程。该函数必须接受一个类型为`internal`的单个参数，并返回`bytea`类型。还需要相应的`deserialfunc`。

deserialfunc

将先前序列化的聚合状态反序列化回`state_data_type`。此函数必须接受`bytea`和`internal`类型的两个参数，并产生一个`internal`类型的结果。（注意：第二个`internal`参数未使用，但出于类型安全的原因是必需的。）

initial_condition

状态值的初始设置。这必须是数据类型`state_data_type`接受的字符串常量。如果未指定，则状态值开始为`null`。

msfunc

在移动聚合模式下要为每个输入行调用的前向状态转换函数的名称。这与常规转换函数完全相同，除了它的第一个参数和结果的类型为`mstate_data_type`，这可能与`state_data_type`不同。

minvfunc

在移动聚集模式下使用的逆状态转换函数的名称。该函数的参数和结果类型与`msfunc`相同，但是用于从当前聚合状态中删除一个值，而不是向其添加值。逆转换函数必须具有与前向状态转换函数相同的严格性属性。

mstate_data_type

使用移动聚合模式时，聚合状态值的数据类型。

mstate_data_size

使用移动聚合模式时，聚合状态值的近似平均大小（以字节为单位）。这与*state_data_size*相同。

mffunc

使用移动聚合模式时，在遍历所有输入行之后将用来计算聚合结果的最终函数的名称。它与*ffunc*相同，不同之处在于其第一个参数的类型为*mstate_data_type*，并且通过编写MFINALFUNC_EXTRA指定了额外的伪参数。由*mffunc*或*mstate_data_type*确定的聚合结果类型必须与由聚合的常规实现确定的结果类型匹配。

minitial_condition

使用移动聚合模式时，状态值的初始设置。这与*initial_condition*相同。

sort_operator

类似于MIN或MAX的聚合的关联排序运算符。这只是一个运算符名称（可以由模式指定）。假定运算符具有与聚合相同的输入数据类型（必须是单参数常规聚合）。

HYPOTHETICAL

仅对于有序集合聚合，此标志指定将根据假设集合聚合的要求处理聚合参数：也就是说，最后几个直接参数必须与聚合（WITHIN GROUP）参数的数据类型匹配。HYPOTHETICAL标志对运行时行为没有影响，仅对数据类型的解析时解析和聚合参数的排序规则有影响。

combinefunc

组合函数的名称。这是两个参数的函数，两个参数的类型均为*state_data_type*。它必须返回*state_data_type*类型的值。组合函数采用两个过渡状态值，并返回代表组合聚合的新过渡状态值。在MPP数据库中，如果以分段方式计算聚合函数的结果，则对各个内部状态调用合并函数，以将它们组合为结束内部状态。

请注意，该函数在segment内的哈希聚合模式下也被调用。因此，如果您在没有合并函数的情况下调用此聚合函数，则永远不会选择哈希聚合。由于哈希聚合非常有效，因此请考虑尽可能定义一个合并函数。

注解

必须先定义用于定义新聚合函数的普通函数。 请注意，在此版本的MPP数据库中，要求将用于创建聚合的*sfunc*，*ffunc*和*Combinefunc*函数定义为IMMUTABLE。

如果MPP数据库服务器配置参数gp_enable_multiphase_agg的值关闭，则仅执行单级聚合。

用于自定义函数的所有已编译代码（共享库文件）必须放在MPP数据库阵列（master和所有segment）中每个主机的相同位置。此位置也必须位于LD_LIBRARY_PATH中，以便服务器可以找到文件。

在以前的MPP数据库版本中，有一个有序聚合的概念。从版本6开始，可以使用以下语法将任何聚合称为有序聚合：

```
name ( arg [ , ... ] [ORDER BY sortspec [ , ...]] )
```

接受ORDERED关键字是为了实现向后兼容，但会被忽略。

在以前的MPP数据库版本中，COMBINEFUNC选项称为PREFUNC。为了向后兼容，它仍然被接受，作为COMBINEFUNC的同义词。

Example

以下简单示例创建一个聚合函数，该函数计算两列的总和。

在创建聚合函数之前，创建两个用作聚合函数的*sfunc*和*combinefunc*函数的函数。

该函数在聚合函数中指定为*sfunc*函数。

```
CREATE FUNCTION myfunc_accum(numeric, numeric, numeric)
RETURNS numeric
AS 'select $1 + $2 + $3'
LANGUAGE SQL
IMMUTABLE
RETURNS NULL ON NULL INPUT;
```

该函数在聚合函数中被指定为combinefunc函数。

```
CREATE FUNCTION mycombine_accum(numeric, numeric )
RETURNS numeric
AS 'select $1 + $2'
LANGUAGE SQL
IMMUTABLE
RETURNS NULL ON NULL INPUT;
```

此CREATE AGGREGATE命令创建添加两列的聚合函数。

```
CREATE AGGREGATE agg_prefunc(numeric, numeric) (
SFUNC = myfunc_accum,
STYPE = numeric,
COMBINEFUNC = mycombine_accum,
INITCOND = 0 );
```

以下命令创建一个表，添加一些行，并运行聚合函数。

```
create table t1 (a int, b int) DISTRIBUTED BY (a);
insert into t1 values
(10, 1),
(20, 2),
(30, 3);
select agg_prefunc(a, b) from t1;
```

此EXPLAIN命令显示两阶段聚合。

```
explain select agg_prefunc(a, b) from t1;
```

QUERY PLAN

```
-----  
Aggregate (cost=1.10..1.11 rows=1 width=32)  
-> Gather Motion 2:1 (slice1; segments: 2) (cost=1.04..1.08 rows=1  
    width=32)  
    -> Aggregate (cost=1.04..1.05 rows=1 width=32)  
        -> Seq Scan on t1 (cost=0.00..1.03 rows=2 width=8)  
(4 rows)
```

兼容性

CREATE AGGREGATE是MPP数据库语言的扩展。 SQL标准不提供用户定义的聚合函数。

另见

[ALTER AGGREGATE](#) , [DROP AGGREGATE](#) , [CREATE FUNCTION](#)

Parent topic: [SQL Command Reference](#)

CREATE CAST

定义一种新的造型。

概要

```
CREATE CAST (sourcetype AS targettype)  
  WITH FUNCTION funcname (argtype [, ...])  
  [AS ASSIGNMENT | AS IMPLICIT]
```

```
CREATE CAST (sourcetype AS targettype)  
  WITHOUT FUNCTION  
  [AS ASSIGNMENT | AS IMPLICIT]
```

```
CREATE CAST (sourcetype AS targettype)  
  WITH INOUT  
  [AS ASSIGNMENT | AS IMPLICIT]
```

描述

CREATE CAST定义一个新的造型。造型指定如何在两种数据类型之间执行转换。例如，

```
SELECT CAST(42 AS float8);
```

通过调用先前指定的函数（在本例中为float8(int4)）将整数常量42转换为float8类型。如果未定义合适的造型，则转换将失败。

两种类型可以是二进制强制的，这意味着可以在不调用任何功能的情况下将这些类型相互转换。这要求相应的值使用相同的内部表示形式。例如，类型text和varchar在两个方向上都是二进制可强制的。二进制强制性不一定是对称关系。例如，在本实现中，可以无代价执行从xml到text的转换，但是相反的方向需要至少执行语法检查的函数。（两种都可以二进制强制的类型也称为二进制兼容。）

您可以使用WITH INOUT语法将转换定义为I/O转换造型。通过调用源数据类型的输出函数并将结果字符串传递给目标数据类型的输入函数来执行I/O转换造

型。在许多常见情况下，此功能无需编写单独的造型函数即可进行转换。I/O转换造型与基于常规函数的造型相同；只是实现不同。

默认情况下，只能由显式造型请求调用造型，该请求是显式`CAST(x AS typename)`或`x:: typename`构造。

如果造型标记为AS ASSIGNMENT，则在为目标数据类型的列分配值时可以隐式调用它。例如，假设foo.f1是text类型的列，则：

```
INSERT INTO foo (f1) VALUES (42);
```

如果从integer类型到text类型的造型标记为AS ASSIGNMENT，则将被允许，否则被禁止。术语分配转换通常用于描述这种转换。

如果造型标记为AS IMPLICIT，则可以在任何上下文中隐式调用它，无论是赋值还是在表达式内部。术语隐式造型通常用于描述这种造型。例如，考虑以下查询：

```
SELECT 2 + 4.0;
```

解析器最初将常量标记为integer和numeric。系统catalog中没有integer + numeric运算符，但是有numeric + numeric运算符。如果存在从integer到numeric的造型（确实存在）并且标记为AS IMPLICIT（实际上是），则此查询成功。解析器仅应用隐式造型，并按以下方式解析查询：

```
SELECT CAST ( 2 AS numeric ) + 4.0;
```

catalog还提供了从numeric到integer的转换。如果该类型转换被标记为AS IMPLICIT（实际不是），则解析器将面临在上述解释与将numeric常量转换为integer并应用integer + integer运算符的选择之间进行选择的选择。缺乏对选择哪种选择的了解，解析器将放弃并声明模糊查询。这两个造型中只有一个是隐式的，这是我们教导解析器将混合的numeric和integer表达式解析

为numeric的方式。 解析器对此没有内置的知识。

保守地将造型标记为隐式是明智的。 过多的隐式转换路径可能会使MPP数据库选择令人惊讶的命令解释， 或者因为存在多种可能的解释而根本无法解析命令。 一个好的经验法则是， 仅对于同一通用类型类别中的类型之间的信息保留转换， 隐式调用造型。 例如， 从int2到int4的造型可以合理地是隐式的， 但是从float8到int4的造型应该仅是赋值的。 跨类型类别的造型（例如， text到int4）最好设为显式。

Note: 有时， 出于可用性或符合标准的原因， 有必要在一组类型之间提供多个隐式造型， 从而导致如上所述的无法避免的歧义。 解析器使用基于类型类别和首选类型的后备启发式方法， 有助于在这种情况下提供所需的行为。 有关更多信息， 请参见[CREATE TYPE](#)。

为了能够创建造型， 您必须拥有源或目标数据类型， 并对其他类型具有USAGE特权。 要创建二进制造型， 您必须是超级用户。（之所以做出此限制， 是因为错误的二进制可强制造型转换很容易使服务器崩溃。）

参数

sourcetype

造型的源数据类型的名称。

targettype

造型的目标数据类型的名称。

funcname(*argtype* [, ...])

用于执行造型的函数。 函数名称可以是模式限定的。 如果不是， MPP数据库将在模式搜索路径中查找该函数。 函数的结果数据类型必须与造型的目标类型匹配。

造型实现函数可能具有一到三个参数。 第一个参数类型必须与造型的源类型相同或可以从二进制强制转换。 第二个参数（如果存在）必须为integer类型； 它接收与目标类型关联的类型修饰符； 如果没有， 则返回-1。 第三个参数（如果存在）必须为boolean类型。 如果造型是显式造

型，则为true；否则为false。在某些情况下，SQL规范要求对显式和隐式造型使用不同的行为。为必须实现此类造型的函数提供此参数。不建议您以此方式设计自己的数据类型。

造型函数的返回类型必须与造型的目标类型相同或二进制强制。

通常，造型必须具有不同的源和目标数据类型。但是，如果它具有一个使用多个参数的造型实现，则可以声明具有相同源类型和目标类型的造型。它用于表示系统catalog中特定于类型的长度强制函数。命名函数用于将类型的值强制为其第二个参数给出的类型修饰符值。

当造型具有不同的源类型和目标类型以及一个函数使用多个参数时，造型将从一种类型转换为另一种类型，并在一个步骤中施加长度强制。如果没有此类条目，则强制使用类型修饰符的类型涉及两个步骤，一个步骤在数据类型之间进行转换，第二个步骤应用该修饰符。

域类型的造型当前无效。向域或从域进行强制转换使用与其基础类型关联的造型。

WITHOUT FUNCTION

指示源类型对目标类型是二进制强制的，因此不需要任何函数即可执行造型。

WITH INOUT

指示造型是I/O转换造型，通过调用源数据类型的输出函数并将结果字符串传递给目标数据类型的输入函数来执行。

AS ASSIGNMENT

指示造型可以在分配上下文中隐式调用。

AS IMPLICIT

指示在任何上下文中都可以隐式调用造型。

注解

请注意，在此版本的MPP数据库中，必须将用户定义的造型中使用的用户定义的函数定义为IMMUTABLE。用于自定义函数的所有已编译代码（共享库文件）必须放在MPP数据库阵列（master和所有segment）中每个主机的相同位置。此位置也必须位于LD_LIBRARY_PATH中，以便服务器可以找到文件。

请记住，如果您希望能够以两种方式转换类型，则需要显式地声明两种类型的造型。

通常不必在用户定义的类型和标准字符串类型（text，varchar和char(*n*)以及定义为字符串类别的用户定义的类型）之间创建造型。MPP数据库为此提供了自动的I/O转换造型。字符串类型的自动造型被视为赋值强制造型，而字符串类型的自动造型仅是显式的。您可以通过声明自己的造型来替换自动造型来覆盖此行为，但是通常这样做的唯一原因是，如果您希望转换比标准分配或仅显式设置更容易调用。另一个可能的原因是，您希望转换的行为不同于类型的I/O函数 - 在执行此操作之前请三思。（少数内置类型的确实具有不同的转换行为，主要是由于SQL标准的要求。）

建议您在目标数据类型之后遵循命名造型实现函数的约定，因为已命名内置造型实现函数。许多用户习惯于使用函数样式表示法（即`typename(x)`）来转换数据类型。

在两种情况下，函数调用构造被视为转换请求，而没有将其与实际函数匹配。如果函数调用`name(x)`与现有函数不完全匹配，但是`name`是数据类型的名称，并且`pg_cast`提供了从`x`类型到该类型的二进制强制转换，则该调用将被解释为二进制强制转换。MPP数据库例外，因此即使缺少任何函数，也可以使用函数语法调用二进制强制转换。同样，如果没有`pg_cast`条目，但强制类型转换为字符串类型或从字符串类型强制类型转换，则该调用将被解释为I/O转换造型。此异常允许使用函数语法调用I/O转换造型。

上述异常是一个例外：不能使用函数语法调用从复合类型到字符串类型的I/O转换造型，而必须使用显式的转换语法（`CAST`或`::`表示法）编写。之所以存在该异常，是因为在引入自动提供的I/O转换造型后，当您打算使用函数或列引用时，发现很容易意外调用此类造型。

示例

要使用函数`int4(bigint)`创建从类型`bigint`到类型`int4`的赋值造型（此造型已在系统

中预先定义)：

```
CREATE CAST (bigint AS int4) WITH FUNCTION int4(bigint) AS ASSIGNMENT;
```

参数

CREATE CAST命令符合SQL标准，但SQL不为二进制可强制类型或实现函数的额外参数提供准备。 AS IMPLICIT也是MPP数据库扩展。

另见

[CREATE FUNCTION](#) , [CREATE TYPE](#) , [DROP CAST](#)

Parent topic: [SQL Command Reference](#)

CREATE COLLATION

使用指定的操作系统语言环境或通过复制现有的排序规则来定义新的排序规则。

概要

```
CREATE COLLATION name (  
  [ LOCALE = locale, ]  
  [ LC_COLLATE = lc_collate, ]  
  [ LC_CTYPE = lc_ctype ])  
  
CREATE COLLATION name FROM existing_collation
```

描述

为了能够创建排序规则，您必须在目标模式上具有CREATE特权。

参数

name

排序规则的名称。排序规则名称可以由模式指定。如果不是，则在当前模式中定义排序规则。排序规则名称在该模式中必须唯一。（系统catalog可以包含与其他编码相同名称的排序规则，但是如果数据库编码不匹配，则将忽略这些排序规则。）

locale

这是一次设置LC_COLLATE和LC_CTYPE的快捷方式。如果指定此选项，则不能指定这些参数中的任何一个。

lc_collate

将指定的操作系统语言环境用于LC_COLLATE语言环境类别。语言环境必须适用于当前数据库编码。（有关详细规则，请参见[CREATE DATABASE](#)。）

lc_ctype

将指定的操作系统语言环境用于LC_CTYPE语言环境类别。语言环境必须适用于当前数据库编码。（有关详细规则，请参见[CREATE DATABASE](#)。）

existing_collation

要复制的现有排序规则的名称。新的归类将具有与现有归类相同的属性，但是它将是一个独立的对象。

注解

为了能够创建排序规则，您必须在目标模式上具有CREATE特权。

使用DROP COLLATION删除用户定义的排序规则。

有关MPP数据库中归类支持的更多信息，请参见PostgreSQL文档中的[归类支持](#)。

示例

要从操作系统语言环境fr_FR.utf8创建排序规则（假设当前数据库编码为UTF8）：UTF8):

```
CREATE COLLATION french (LOCALE = 'fr_FR.utf8');
```

要从现有排序规则创建排序规则：

```
CREATE COLLATION german FROM "de_DE";
```

能够在应用程序中使用独立于操作系统的排序规则名称可能很方便。

兼容性

SQL标准中有一个CREATE COLLATION语句，但是它仅限于复制现有的排序规则。创建新排序规则的语法是MPP数据库扩展。

另见

[ALTER COLLATION](#), [DROP COLLATION](#)

Parent topic: [SQL Command Reference](#)

CREATE CONVERSION

定义新的编码转换。

概要

```
CREATE [DEFAULT] CONVERSION name FOR source_encoding TO  
  dest_encoding FROM funcname
```

描述

CREATE CONVERSION定义了字符集编码之间的新转换。转换名称可以在转换函数中使用，以指定特定的编码转换。另外，标记为DEFAULT的转换可用于客户端和服务端之间的自动编码转换。为此，必须定义从编码A到B和从编码B到A的两次转换。

要创建转换，您必须在函数上具有EXECUTE特权，并在目标模式上具有CREATE特权。

参数

DEFAULT

指示此转换是此特定源到目标编码的默认转换。模式中的编码对应该只有一种默认编码。

name

转换的名称。转换名称可以由模式指定。如果不是，则在当前模式中定义转换。转换名称在模式中必须唯一。

source_encoding

源编码名称。

dest_encoding

目标编码名称。

funcname

用于执行转换的函数。函数名称可以由模式指定。如果不是，将在路径中查找该函数。该函数必须具有以下签名：

```
conv_proc(  
    integer, -- source encoding ID  
    integer, -- destination encoding ID  
    cstring, -- source string (null terminated C string)  
    internal, -- destination (fill with a null terminated C string)  
    integer -- source string length  
) RETURNS void;
```

注解

请注意，在此版本的MPP数据库中，用户定义转换中使用的用户定义函数必须定义为IMMUTABLE。用于自定义函数的所有已编译代码（共享库文件）必须放在MPP数据库阵列（master和所有segment）中每个主机的相同位置。此位置也必须位于LD_LIBRARY_PATH中，以便服务器可以找到文件。

示例

要使用myfunc创建从UTF8编码到LATIN1的转换，请执行以下操作：

```
CREATE CONVERSION myconv FOR 'UTF8' TO 'LATIN1' FROM myfunc;
```

兼容性

SQL标准中没有CREATE CONVERSION语句，但是在目的和语法上有一个非常相似的CREATE TRANSLATION语句。

另见

[ALTER CONVERSION](#) , [CREATE FUNCTION](#) , [DROP CONVERSION](#)

Parent topic: [SQL Command Reference](#)

CREATE DATABASE

创建一个新的数据库。

概要

```
CREATE DATABASE name [ [WITH] [OWNER [=] user_name]  
    [TEMPLATE [=] template]  
    [ENCODING [=] encoding]  
    [LC_COLLATE [=] lc_collate]  
    [LC_CTYPE [=] lc_ctype]  
    [TABLESPACE [=] tablespace]  
    [CONNECTION LIMIT [=] conlimit ] ]
```

描述

CREATE DATABASE 创建一个新的数据库。要创建数据库，您必须是超级用户或具有特殊的CREATEDB特权。

默认情况下，创建者成为新数据库的所有者。超级用户可以使用OWNER子句创建其他用户拥有的数据库。他们甚至可以创建没有特殊特权的用户所拥有的数据库。具有CREATEDB特权的非超级用户只能创建自己拥有的数据库。

默认情况下，将通过克隆标准系统数据库template1来创建新数据库。可以通过写入TEMPLATE *name*来指定其他模板。特别是，通过编写TEMPLATE template0，您可以创建一个仅包含MPP数据库预定义的标准对象的干净数据库。如果您希望避免复制任何可能已添加到template1的本地安装对象，则此功能很有用。

参数

name

要创建的数据库的名称。

user_name

拥有新数据库的用户的名称，或者使用默认所有者（执行命令的用户） DEFAULT。

template

创建新数据库的模板的名称，或者DEFAULT使用默认模板的模板（*template1*）。

encoding

在新数据库中使用的字符集编码。指定字符串常量（例如'SQL_ASCII'），整数编码数字或DEFAULT以使用默认编码。有关更多信息，请参见[字符集支持](#)。

lc_collate

在新数据库中使用的排序规则（LC_COLLATE）。这会影响应用于字符串的排序顺序，例如在使用ORDER BY的查询中，以及在文本列的索引中使用的顺序。默认设置是使用模板数据库的排序规则。有关其他限制，请参见“注释”部分。

lc_ctype

在新数据库中使用的字符分类（LC_CTYPE）。这会影响字符的分类，例如下，上和数字。默认值是使用模板数据库的字符分类。有关其他限制，请参见下文。

tablespace

将与新数据库关联的表空间的名称，或者为DEFAULT以使用模板数据库的表空间。该表空间将是用于在此数据库中创建的对象默认表空间。

connlimit

可能的最大并发连接数。默认值-1表示没有限制。

注解

CREATE DATABASE不能在事务块内执行。

通过将数据库名称指定为模板来复制数据库时，在复制数据库过程中，不能将其其他会话连接到模板数据库。模板数据库的新连接被锁定，直到CREATE DATABASE完成。

对超级用户没有强制CONNECTION LIMIT。

为新数据库指定的字符集编码必须与所选的语言环境设置

(LC_COLLATE和LC_CTYPE) 兼容。如果语言环境为C（或等价的POSIX），则允许所有编码，但是对于其他语言环境设置，只有一种编码可以正常工作。CREATE DATABASE将允许超级用户指定SQL_ASCII编码，而与语言环境设置无关，但是不建议使用此选项，并且如果与语言环境不兼容的数据存储在数据库中，则可能会导致字符串函数出现异常。

编码和语言环境设置必须与模板数据库的设置匹配，除非将template0用作模板。这是因为COLLATE和CTYPE影响索引的顺序，因此从模板数据库复制的任何索引在具有不同设置的新数据库中都将无效。但是，已知template0不包含任何会受到影响的数据或索引。

示例

要创建一个新的数据库：

```
CREATE DATABASE gpdb;
```

要创建用户salesapp拥有的数据库sales并使用salesspace的默认表空间：

```
CREATE DATABASE sales OWNER salesapp TABLESPACE salesspace;
```

要创建支持ISO-8859-1字符集的数据库music：

```
CREATE DATABASE music ENCODING 'LATIN1' TEMPLATE template0;
```

在此示例中，仅当template1的编码不是ISO-8859-1时才需要TEMPLATE template0子句。请注意，更改编码可能还需要选择新的LC_COLLATE和LC_CTYPE设置。

兼容性

SQL标准中没有CREATE DATABASE语句。 数据库等效于目录，目录的创建是实现定义的。

另见

[ALTER DATABASE](#) , [DROP DATABASE](#)

Parent topic: [SQL Command Reference](#)

CREATE DOMAIN

定义一个新域。

概要

```
CREATE DOMAIN name [AS] data_type [DEFAULT expression]  
    [ COLLATE collation ]  
    [ CONSTRAINT constraint_name  
    | NOT NULL | NULL  
    | CHECK (expression) [...]]
```

描述

CREATE DOMAIN创建一个新域。域本质上是具有可选约束（对允许的值集的约束）的数据类型。定义域的用户将成为其所有者。域名在其模式中存在的数据类型和域之间必须唯一。

如果提供了模式名称（例如CREATE DOMAIN myschema.mydomain ...），则将在指定的模式中创建域。否则，它将在当前模式中创建。

域对于将字段上的公共约束抽象到单个位置进行维护很有用。例如，几个表可能包含电子邮件地址列，所有这些列都需要相同的CHECK约束来验证地址语法。定义域比为每个具有电子邮件列的表设置列约束要容易得多。

为了能够创建域，您必须对基础类型具有USAGE特权。

参数

name

要创建的域的名称（可以由模式指定）。

data_type

域的基础数据类型。这可能包括数组说明符。

DEFAULT *expression*

为域数据类型的列指定默认值。该值是无变量的表达式（但不允许子查询）。默认表达式的数据类型必须与域的数据类型匹配。如果未指定默认值，则默认值为空值。默认表达式将在未为列指定值的任何插入操作中使用。如果为特定列定义了默认值，它将覆盖与该域关联的任何默认值。反过来，域默认值将覆盖与基础数据类型关联的任何默认值。

COLLATE *collation*

域的可选归类。如果未指定排序规则，则使用基础数据类型的默认排序规则。如果指定了COLLATE，则基础类型必须可整理。

CONSTRAINT *constraint_name*

约束的可选名称。如果未指定，系统将生成一个名称。

NOT NULL

通常可以防止该域的值为null。但是，如果为该域分配了已变为空的匹配域类型仍然是可行的，例如，通过左连接或诸如INSERT INTO tab (domcol) VALUES ((SELECT domcol FROM tab WHERE false))之类的命令。

NULL

该域的值允许为空。这是默认值。此子句仅用于与非标准SQL数据库兼容。不建议在新应用中使用它。

CHECK(*expression*)

CHECK子句指定完整性约束或测试域必须满足的值。每个约束必须是一个产生布尔结果的表达式。它应该使用关键字VALUE来引用要测试的值。当前，CHECK表达式不能包含子查询，也不能引用VALUE以外的变量。

示例

创建us_zip_code数据类型。使用正则表达式测试来验证该值看起来像是有效的美国邮政编码。

```
CREATE DOMAIN us_zip_code AS TEXT CHECK  
( VALUE ~ '^d{5}$' OR VALUE ~ '^d{5}-d{4}$' );
```

兼容性

CREATE DOMAIN符合SQL标准。

另见

ALTER DOMAIN , DROP DOMAIN

Parent topic: [SQL Command Reference](#)

CREATE EXTENSION

Registers an extension in a MPP database.

概要

```
CREATE EXTENSION [ IF NOT EXISTS ] extension_name
[ WITH ] [ SCHEMA schema_name ]
    [ VERSION version ]
    [ FROM old_version ]
    [ CASCADE ]
```

描述

CREATE EXTENSION将新的扩展名加载到当前数据库中。不可以已加载具有相同名称的扩展名。

加载扩展实质上等于运行扩展脚本文件。该脚本通常创建新的SQL对象，例如函数，数据类型，运算符和索引支持方法。CREATE EXTENSION命令还记录所有已创建对象的标识，因此，如果执行DROP EXTENSION，则可以再次删除它们。

加载扩展需要与创建组件扩展对象相同的特权。对于大多数扩展，这意味着需要超级用户或数据库所有者特权。运行CREATE EXTENSION的用户将成为扩展的所有者，以进行以后的权限检查，以及该扩展脚本创建的任何对象的所有者。

参数

IF NOT EXISTS

如果已经存在具有相同名称的扩展名，请不要报错。在这种情况下发出通知。无法保证现有扩展与安装的扩展类似。

extension_name

要安装的扩展名。该名称在数据库中必须唯一。根据扩展控制文件*SHAREDIR/extension/extension_name.control*中的详细信息创建扩展。

*SHAREDIR*是安装共享数据目录，例如*/usr/local/mpp-db/share/postgresql*。命令*pg_config --sharedir*显示目录。

SCHEMA *schema_name*

在其中安装扩展对象的模式名称。假定扩展名允许其内容重定位。命名模式必须已经存在。如果未指定，并且扩展控制文件未指定模式，则使用当前的默认模式创建对象。

如果扩展名在其控制文件中指定了模式参数，则无法使用SCHEMA子句覆盖该模式。通常，如果给定了SCHEMA子句，并且与扩展模式参数冲突，则会引发错误。但是，如果还给出了CASCADE子句，则*schema_name*在冲突时将被忽略。给定的*schema_name*用于安装所有需要的扩展，这些扩展在其控制文件中未指定架构。

扩展本身不在任何模式内。扩展名具有非限定名称，这些名称在数据库中必须唯一。但是属于扩展名的对象可以在模式内。

VERSION *version*

要安装的扩展程序的版本。可以将其写为标识符或字符串文字。默认版本是扩展控制文件中指定的值。

FROM *old_version*

仅在尝试安装扩展来替换旧模块（该扩展是未打包到扩展中的对象集合）的扩展时，才指定FROM *old_version*。如果指定，则CREATE EXTENSION运行替代安装脚本，该脚本将现有对象吸收到扩展中，而不是创建新对象。确保SCHEMA子句指定了包含这些预先存在的对象的模式。

用于*old_version*的值由扩展作者确定，如果可以将多个旧版本的模块升级为扩展，则该值可能会有所不同。对于9.1之前的PostgreSQL提供的标准附加模块，在将模块更新为扩展样式时，请为*old_version*指定unpackaged。

CASCADE

尚未安装自动安装相关扩展。递归检查相关的扩展，并且这些从属扩展也将自动安装。如果指定了SCHEMA子句，则该模式适用于扩展和已安装的所有从属扩展。指定的其他选项不适用于自动安装的从属扩展。特别是，在安装从属扩展时始终选择默认版本。

注解

当前可用于加载的扩展名可以从[*pg_available_extensions*](#)或[*pg_available_extension_versions*](#)系统视图中识别。

在使用CREATE EXTENSION将扩展加载到数据库之前，必须安装支持的扩展文件，包括扩展控制文件和至少一个SQL脚本文件。支持文件必须安装在所有MPP数据库主机上的相同位置。有关创建新扩展的信息，请参见PostgreSQL有关[将相关对象打包到扩展中](#)的信息。

兼容性

CREATE EXTENSION是MPP数据库扩展。

另见

[ALTER EXTENSION](#), [DROP EXTENSION](#)

Parent topic: [SQL Command Reference](#)

CREATE EXTERNAL TABLE

定义一个新的外部表。

概要

```
CREATE [READABLE] EXTERNAL [TEMPORARY | TEMP] TABLE table_name
    ( column_name data_type [, ...] | LIKE other_table )
```

```

LOCATION ('file://seghost[:port]/path/file' [, ...])
| ('gpfdist://filehost[:port]/file_pattern[#transform=trans_name]'
  [, ...])
| ('gpfdists://filehost[:port]/file_pattern[#transform=trans_name]'
  [, ...])
| ('pxf://path-to-data?PROFILE=profile_name[&SERVER=server_name][&custom-option=v
| ('s3://S3_endpoint[:port]/bucket_name/[S3_prefix] [region=S3-region] [config=config_file]'
[ON MASTER]
FORMAT 'TEXT'
  [( [HEADER]
    [DELIMITER [AS] 'delimiter' | 'OFF']
    [NULL [AS] 'null string']
    [ESCAPE [AS] 'escape' | 'OFF']
    [NEWLINE [ AS ] 'LF' | 'CR' | 'CRLF']
    [FILL MISSING FIELDS] )]
  | 'CSV'
  [( [HEADER]
    [QUOTE [AS] 'quote']
    [DELIMITER [AS] 'delimiter']
    [NULL [AS] 'null string']
    [FORCE NOT NULL column [, ...]]
    [ESCAPE [AS] 'escape']
    [NEWLINE [ AS ] 'LF' | 'CR' | 'CRLF']
    [FILL MISSING FIELDS] )]
  | 'CUSTOM' (Formatter=<formatter_specifications>)
[ ENCODING 'encoding' ]
[ [LOG ERRORS] SEGMENT REJECT LIMIT count
[ROWS | PERCENT] ]

```

```

CREATE [READABLE] EXTERNAL WEB [TEMPORARY | TEMP] TABLE table_name
( column_name data_type [, ...] | LIKE other_table )
  LOCATION ('http://webhost[:port]/path/file' [, ...])
| EXECUTE 'command' [ON ALL
  | MASTER
  | number_of_segments
  | HOST ['segment_hostname']
  | SEGMENT segment_id ]
FORMAT 'TEXT'
  [( [HEADER]
    [DELIMITER [AS] 'delimiter' | 'OFF']
    [NULL [AS] 'null string']
    [ESCAPE [AS] 'escape' | 'OFF']
    [NEWLINE [ AS ] 'LF' | 'CR' | 'CRLF']

```



```

    [FILL MISSING FIELDS] ))
  | 'CSV'
  [( [HEADER]
    [QUOTE [AS] 'quote']
    [DELIMITER [AS] 'delimiter']
    [NULL [AS] 'null string']
    [FORCE NOT NULL column [, ...]]
    [ESCAPE [AS] 'escape']
    [NEWLINE [ AS ] 'LF' | 'CR' | 'CRLF']
    [FILL MISSING FIELDS] ))
  | 'CUSTOM' (Formatter=<formatter specifications>)
[ ENCODING 'encoding' ]
[ [LOG ERRORS] SEGMENT REJECT LIMIT count
[ROWS | PERCENT] ]

```

```

CREATE WRITABLE EXTERNAL [TEMPORARY | TEMP] TABLE table_name
( column_name data_type [, ...] | LIKE other_table )
LOCATION('gpfdist://outputhost[:port]/filename[#transform=trans_name]'
[, ...])
| ('gpfdists://outputhost[:port]/file_pattern[#transform=trans_name]'
[, ...])
FORMAT 'TEXT'
  (( [DELIMITER [AS] 'delimiter']
    [NULL [AS] 'null string']
    [ESCAPE [AS] 'escape' | 'OFF'] ))
  | 'CSV'
  ([([QUOTE [AS] 'quote']
    [DELIMITER [AS] 'delimiter']
    [NULL [AS] 'null string']
    [FORCE QUOTE column [, ...]] | * ]
    [ESCAPE [AS] 'escape'] ))

  | 'CUSTOM' (Formatter=<formatter specifications>)
[ ENCODING 'write_encoding' ]
[ DISTRIBUTED BY ({column [opclass]}, [ ... ] ) | DISTRIBUTED RANDOMLY ]

```

```

CREATE WRITABLE EXTERNAL [TEMPORARY | TEMP] TABLE table_name
( column_name data_type [, ...] | LIKE other_table )
LOCATION('s3://S3_endpoint[:port]/bucket_name/[S3_prefix] [region=S3-region] [config=cc]
[ON MASTER]
FORMAT 'TEXT'
  (( [DELIMITER [AS] 'delimiter']
    [NULL [AS] 'null string']

```

```

    [ESCAPE [AS] 'escape' | 'OFF' ]]
| 'CSV'
    [( [QUOTE [AS] 'quote'
    [DELIMITER [AS] 'delimiter'
    [NULL [AS] 'null string'
    [FORCE QUOTE column [, ...]] | * ]
    [ESCAPE [AS] 'escape' ] ]
| 'CUSTOM' (Formatter=<formatter specifications>)
[ ENCODING 'write_encoding' ]
[ DISTRIBUTED BY ({column [opclass]}, [ ... ] ) | DISTRIBUTED RANDOMLY ]

```

描述

有关外部表的详细信息，请参阅***MPP数据库管理员手册***中的“使用外部表”。

CREATE EXTERNAL TABLE或CREATE EXTERNAL WEB TABLE在MPP数据库中创建一个新的可读外部表定义。可读的外部表通常用于快速并行数据加载。定义外部表后，您可以使用SQL命令直接（或并行）查询其数据。例如，您可以选择，联接或排序外部表数据。您也可以为外部表创建视图。可读外部表上不允许DML操作（UPDATE，INSERT，DELETE或TRUNCATE），并且您不能在可读外部表上创建索引。

CREATE WRITABLE EXTERNAL TABLE或**CREATE WRITABLE EXTERNAL WEB TABLE**在MPP数据库中创建新的可写外部表定义。可写的外部表通常用于将数据库中的数据卸载到一组文件或命名管道中。可写的外部Web表也可以用于将数据输出到可执行程序。可写的外部表也可以用作MPP并行MapReduce计算的输出目标。一旦定义了可写外部表，就可以从数据库表中选择数据并将其插入可写外部表中。可写的外部表仅允许INSERT操作 – 不允许执行SELECT，UPDATE，DELETE或TRUNCATE。

常规外部表和外部Web表之间的主要区别在于它们的数据源。常规可读的外部表访问静态平面文件，而外部Web表访问动态数据源 - 在Web服务器上或通过执行OS命令或脚本。

参数

READABLE | WRITABLE

指定外部表的类型，默认为可读。可读的外部表用于将数据加载到MPP数据库中。可写的外部表用于卸载数据。

WEB

在MPP数据库中创建一个可读或可写的外部Web表定义。可读的外部Web表有两种形式 - 通过http://协议访问文件的表或通过执行OS命令访问数据的表。可写的外部Web表将数据输出到可执行程序，该程序可以接受输入的数据流。外部Web表在查询执行期间不可重新扫描。

s3协议不支持外部Web表。但是，您可以创建一个外部Web表，该表执行第三方工具以从S3读取数据或直接向S3写入数据。

TEMPORARY | TEMP

如果指定，则在MPP数据库中创建一个临时的可读或可写外部表定义。临时外部表存在于特殊模式中。创建表时，无法指定模式名称。在会话结束时，将自动删除临时外部表。

临时表存在时，当前会话将看不到具有相同名称的已知永久表，除非您引用具有其模式限定名称的永久表。

table_name

新外部表的名称。

column_name

要在外部表定义中创建的列的名称。与常规表不同，外部表没有列约束或默认值，因此请不要指定它们。

LIKE other_table

LIKE子句指定一个表，新的外部表将从该表中自动复制所有列名称，数据类型和MPP分配策略。如果原始表指定了任何列约束或默认列值，则不会将这些约束复制到新的外部表定义中。

data_type

列的数据类型。

LOCATION ('protocol://[host[:port]]/path/file' [, ...])

如果使用pxf协议访问外部数据源，请参阅[PXF创建外部表文档](#)，以获取有关pxf协议LOCATION子句语法的详细信息。

如果使用s3协议读取或写入S3，请参阅[关于S3协议URL](#)，以获取有关s3协议LOCATION子句语法的其他信息。

对于可读的外部表，指定用于填充外部表或Web表的外部数据源的URI。常规的可读外部表允许使用gpfdist或file协议。外部Web表允许使用http协议。如果省略port，则将http和gpfdist协议的端口假定为8080。如果使用gpfdist协议，则path是相对于gpfdist从中提供文件的目录（启动gpfdist程序时指定的目录）。另外，gpfdist可以使用通配符或其他C样式模式匹配（例如，空格字符为[:space:]）来表示目录中的多个文件。例如：

```
'gpfdist://filehost:8081/*'  
'gpfdist://masterhost/my_load_file'  
'file://seghost1/dbfast1/external/myfile.txt'  
'http://intranet.example.com/finance/expenses.csv'
```

对于可写的外部表，指定gpfdist进程或S3协议的URI位置，该进程将收集MPP segment中的数据输出并将其写入一个或多个命名文件。对于gpfdist，该路径是相对于gpfdist从中提供文件的目录（启动gpfdist程序时指定的目录）的。如果列出了多个gpfdist位置，则发送数据的segment将在可用的输出位置之间平均分配。例如：

```
'gpfdist://outputhost:8081/data1.out',  
'gpfdist://outputhost:8081/data2.out'
```

在上面的示例中列出了两个gpfdist位置的情况下，一半的segment会将其输出数据发送到data1.out文件，另一半发送到data2.out文件。

使用`#transform=trans_name`选项，可以指定在加载或提取数据时要应用的转换。`trans_name`是您在运行gpfdist实用程序时指定的YAML配置文件中转换的名称。有关指定转换的信息，请参阅MPP实用程序手册中的[gpfdist](#)。

ON MASTER

将所有与表相关的操作限制为MPP master。仅允许使用s3或自定义协议创建的可读写外部表。gpfdist，gpfdists，pxf和file协议不支持ON MASTER。

Note: 在使用ON MASTER子句读取或写入创建的外部表时，请注意潜在的资源影响。当您仅将表操作限制为MPP master时，可能会遇到性能问题。

EXECUTE 'command' [ON ...]

只允许可读的外部Web表或可写的外部表。对于可读的外部Web表，指定要由segment实例执行的OS命令。该`command`可以是单个OS命令或脚本。ON子句用于指定哪些segment实例将执行给定命令。

- 默认为ON ALL。该命令将由MPP数据库系统中所有segment主机上的每个活动primary实例执行。如果命令执行脚本，则该脚本必须位于所有segment主机上的相同位置，并且可由MPP超级用户（gpadmin）执行。
- ON MASTER仅在master主机上运行命令。
Note: 指定ON MASTER子句时，外部Web表不支持日志记录。
- ON *number*表示将按指定的segment数执行命令。特定的segment是在运行时由MPP数据库系统随机选择的。如果命令执行脚本，则该脚本必须位于所有segment主机上的相同位置，并且可由MPP超级用户（gpadmin）执行。
- HOST表示该命令将由每个segment主机上的一个segment执行（每个segment主机一次），而不管每个主机的活动segment实例数量如何。

- `HOST segment_hostname`表示命令将由指定的segment主机上的所有活动primary实例执行。
- `SEGMENT segment_id`表示命令只能由指定的segment执行一次。您可以通过查看系统catalog表[gp_segment_configuration](#)中的`content`编号来确定segment实例的ID。MPP数据库主数据库的content ID始终为-1。

对于可写外部表，必须准备在EXECUTE子句中指定的`command`以将数据传递到其中。由于所有要发送数据的segment都会将其输出写入指定的命令或程序，因此ON子句的唯一可用选项是ON ALL。

FORMAT 'TEXT | CSV' (*options*)

当FORMAT子句标识定界文本（TEXT）或逗号分隔值（CSV）格式时，格式设置选项与PostgreSQL [COPY](#)命令可用的格式设置选项相似。如果文件中的数据不使用默认的列定界符，转义符，空字符串等，则必须指定其他格式选项，以便MPP数据库可以正确读取外部文件中的数据。有关使用自定义格式的信息，请参阅[MPP数据库管理员手册](#)中的“加载和卸载数据”。

如果使用pxf协议访问外部数据源，请参阅PXF[创建外部表](#)文档，以获取有关pxf协议FORMAT子句语法的详细信息。

FORMAT 'CUSTOM' (*formatter=formatter_specification*)

指定自定义数据格式。`formatter_specification`指定用于格式化数据的函数，后跟格式化函数的逗号分隔参数。格式化程序规范的长度（包括Formatter=的字符串）最多可达到约50K字节。

如果使用pxf协议访问外部数据源，请参阅PXF[创建外部表](#)文档，以获取有关pxf协议FORMAT子句语法的详细信息。

有关使用自定义格式的一般信息，请参阅[MPP数据库管理员手册](#)中的“加载和卸载数据”。

DELIMITER

指定一个ASCII字符，用于分隔数据的每一行（行）中的列。默认为TEXT模式下的制表符，而CSV模式下为逗号。在用于可读外部表的TEXT模式下，对于将非结构化数据加载到单列表的特殊用例，可以将定界符设置为OFF。

对于s3协议，定界符不能为换行符（\n）或回车符（\r）。

NULL

指定表示NULL值的字符串。在TEXT模式下，默认值为N（反斜杠-N），在CSV模式下，默认值为无引号的空值。对于不希望将NULL值与空字符串区分开的情况，即使在TEXT模式下，您也可能更喜欢空字符串。使用外部表和Web表时，与该字符串匹配的任何数据项都将被视为NULL值。

作为text格式的示例，此FORMAT子句可用于指定两个单引号（'）的字符串为NULL值。

```
FORMAT 'text' (delimiter ';' null '\\\')
```

ESCAPE

指定用于C转义序列的单个字符（例如\n，\t，\100等）和转义可能用作行或列定界符的数据字符。确保选择一个在实际列数据中未使用的转义字符。对于文本格式的文件，默认转义字符是\（反斜杠），对于csv格式的文件，默认转义字符是"（双引号），但是可以指定另一个字符来表示转义。也可以通过将值'OFF'指定为转义值在文本格式化文件中禁用文本转义。这对于诸如文本格式的Web日志数据之类的数据非常有用，该数据具有许多嵌入式反斜杠，而这些反斜杠并非旨在转义。

NEWLINE

指定数据文件中使用的换行符 – LF（换行，0x0A），CR（回车，0x0D）或CRLF（回车加换行，0x0D 0x0A）。如果未指定，MPP数据库segment将通过查看接收到的第一行数据并使用遇到的第一个换行符类型来检测换行符类型。

HEADER

对于可读的外部表，指定数据文件中的第一行是标题行（包含表列的名称），并且不应包括表的数据在内。如果使用多个数据源文件，则所有文件都必须具有标题行。

对于s3协议，标题行中的列名称不能包含换行符（\n）或回车符（\r）。

pxf协议不支持HEADER格式化选项。

QUOTE

指定CSV模式的引号字符。默认值为双引号（"）。

FORCE NOT NULL

在CSV模式下，处理每个指定的列就好像被引用了一样，因此不是NULL值。对于CSV模式下的默认空字符串（两个定界符之间都不存在），这将导致缺失值被评估为零长度字符串。

FORCE QUOTE

在可写外部表的CSV模式下，强制将引号用于每个指定列中的所有非NULL值。如果指定*，则在所有列中都引用非NULL值。NULL输出从不引用。

FILL MISSING FIELDS

对于可读外部表，在TEXT和CSV模式下，指定FILL MISSING FIELDS时，如果一行数据的行或行末尾缺少数据字段，则将缺少的尾随字段值设置为NULL（而不是报告错误）。空白行，具有NOT NULL约束的字段以及行尾的定界符仍然会报告错误。

ENCODING '*encoding*'

用于外部表的字符集编码。指定字符串常量（例如'SQL_ASCII'），整数编码数字或DEFAULT以使用默认的客户端编码。请参见[字符集支持](#)。

LOG ERRORS

这是一个可选的子句，可以在SEGMENT REJECT LIMIT子句之前记录有关具有格式错误的行的信息。错误日志信息存储在内部，并可以通过MPP数据库内置的SQL函数gp_read_error_log()进行访问。

请参阅[注解](#)以获取有关错误日志信息的信息以及用于查看和管理错误日志信息的内置函数。

SEGMENT REJECT LIMIT *count* [ROWS | PERCENT]

在单行错误隔离模式下运行COPY FROM操作。如果输入行存在格式错误，只要在加载操作期间未在任何MPP segment实例上达到拒绝限制计数，它们将被丢弃。拒绝限制计数可以指定为行数（默认）或总行数的百分比（1-100）。如果使用PERCENT，则只有在处理了参数gp_reject_percent_threshold指定的行数之后，每个segment才开始计算错误行百分比。gp_reject_percent_threshold的默认值为300行。约束错误（例如违反NOT NULL，CHECK或UNIQUE约束）仍将在“全有或全无”输入模式下处理。如果未达到限制，则将加载所有正确的行，并丢弃所有错误行。

Note: 读取外部表时，如果未首先触发SEGMENT REJECT LIMIT或未指

定SEGMENT REJECT LIMIT，则MPP数据库会限制可能包含格式错误的初始行数。如果前1000行被拒绝，则COPY操作将停止并回滚。

可以使用MPP数据库服务器配置参数gp_initial_bad_row_limit更改初始拒绝行数的限制。有关参数的信息，请参阅[服务器配置参数](#)。

DISTRIBUTED BY ({column [opclass]}, [...])

DISTRIBUTED RANDOMLY

用于声明可写外部表的MPP数据库分发策略。默认情况下，可写外部表是随机分布的。如果要从中导出数据的源表具有哈希分发策略，并且为可写外部表定义了相同的分发键列和运算符类opclass，则将消除在interconnect上移动行的需要，从而提高了卸载性能。当您发出诸如INSERT INTO wex_table SELECT * FROM source_table之类的卸载命令时，如果两个表具有相同的哈希分配策略，则可以将这些已卸载的行直接从segment发送到输出位置。

示例

在端口8081的后台启动gpfdist文件服务器程序，以服务目录/var/data/staging中的文件：

```
gpfdist -p 8081 -d /var/data/staging -l /home/gpadmin/log &
```

使用gpfdist协议和gpfdist目录中找到的所有文本格式文件 (*.txt) 创建一个名为ext_customer的可读外部表。使用竖线 (|) 作为列定界符，并使用空白作为NULL来格式化文件。还要以单行错误隔离模式访问外部表：

```
CREATE EXTERNAL TABLE ext_customer
(id int, name text, sponsor text)
LOCATION ( 'gpfdist://filehost:8081/*.txt' )
FORMAT 'TEXT' ( DELIMITER '|' NULL ' ')
LOG ERRORS SEGMENT REJECT LIMIT 5;
```

创建与上述相同的可读外部表定义，但使用CSV格式的文件：

```
CREATE EXTERNAL TABLE ext_customer
(id int, name text, sponsor text)
LOCATION ( 'gpfdist://filehost:8081/*.csv' )
FORMAT 'CSV' ( DELIMITER ',' );
```

使用file协议和一些带有标题行的CSV格式的文件，创建一个名为ext_expenses的可读外部表：

```
CREATE EXTERNAL TABLE ext_expenses (name text, date date,
amount float4, category text, description text)
LOCATION (
'file://seghost1/dbfast/external/expenses1.csv',
'file://seghost1/dbfast/external/expenses2.csv',
'file://seghost2/dbfast/external/expenses3.csv',
'file://seghost2/dbfast/external/expenses4.csv',
'file://seghost3/dbfast/external/expenses5.csv',
'file://seghost3/dbfast/external/expenses6.csv'
)
FORMAT 'CSV' ( HEADER );
```

创建一个可读的外部Web表，每个表主机执行一次脚本：

```
CREATE EXTERNAL WEB TABLE log_output (linenum int, message
text) EXECUTE '/var/load_scripts/get_log_data.sh' ON HOST
FORMAT 'TEXT' (DELIMITER '|');
```

创建一个名为sales_out的可写外部表，该表使用gpfdist将输出数据写入名

为sales.out的文件。使用竖线 (|) 作为列定界符，并使用空白作为NULL来格式化文件。

```
CREATE WRITABLE EXTERNAL TABLE sales_out (LIKE sales)
  LOCATION ('gpfdist://etl1:8081/sales.out')
  FORMAT 'TEXT' ( DELIMITER '|' NULL ' ')
  DISTRIBUTED BY (txn_id);
```

创建一个可写的外部Web表，将segment接收的输出数据通过管道传输到名为to_adreport_etl.sh的可执行脚本：

```
CREATE WRITABLE EXTERNAL WEB TABLE campaign_out
(LIKE campaign)
EXECUTE '/var/unload_scripts/to_adreport_etl.sh'
FORMAT 'TEXT' (DELIMITER "|");
```

使用上面定义的可写外部表来卸载所选数据：

```
INSERT INTO campaign_out SELECT * FROM campaign WHERE
customer_id=123;
```

注解

当您指定LOG ERRORS子句时，MPP数据库将捕获读取外部表数据时发生的错误。您可以查看和管理捕获的错误日志数据。

- 使用内置的SQL函数gp_read_error_log('table_name')。它要求对table_name具有SELECT特权。本示例使用COPY命令显示加载到表ext_expenses中的数据的错误日志信息：

```
SELECT * from gp_read_error_log('ext_expenses');
```

有关错误日志格式的信息，请参阅*MPP数据库管理员手册*中的[查看错误日志中的坏行](#)。

如果`table_name`不存在，则该函数返回FALSE。

- 如果指定表存在错误日志数据，则新的错误日志数据将附加到现有错误日志数据中。 错误日志信息不会复制到mirror。
- 使用内置的SQL函数`gp_truncate_error_log('table_name')`删除`table_name`的错误日志数据。 它需要表所有者特权。本示例删除将数据移入表`ext_expenses`时捕获的错误日志信息：

```
SELECT gp_truncate_error_log('ext_expenses');
```

如果`table_name`不存在，则该函数返回FALSE。

指定`*`通配符可删除当前数据库中现有表的错误日志信息。 指定字符串`.*`以删除所有数据库错误日志信息，包括由于先前的数据库问题而未被删除的错误日志信息。 如果指定`*`，则需要数据库所有者特权。 如果指定`.*`，则需要操作系统超级用户特权。

当使用`gpfdist`，`gpfdists`或`file`协议定义了多个MPP数据库外部表并访问Linux系统中的同一命名管道时，MPP数据库会将命名管道的访问限制为单个读取器。 如果第二个读取器尝试访问命名管道，则返回错误。

兼容性

`CREATE EXTERNAL TABLE`是MPP数据库扩展。 SQL标准没有为外部表做任何准备。

另见

[CREATE TABLE AS](#), [CREATE TABLE](#), [COPY](#), [SELECT INTO](#), [INSERT](#)

Parent topic: [SQL Command Reference](#)

CREATE FOREIGN DATA WRAPPER

定义一个新的外部数据包装器。

概要

```
CREATE FOREIGN DATA WRAPPER name
  [ HANDLER handler_function | NO HANDLER ]
  [ VALIDATOR validator_function | NO VALIDATOR ]
  [ OPTIONS ( [ mpp_execute { 'master' | 'any' | 'all segments' } [, ] ] option 'value' [, ... ] ) ]
```

描述

CREATE FOREIGN DATA WRAPPER在当前数据库中创建一个新的外部数据包装器。定义外部数据包装器的用户将成为其所有者。

只有超级用户才能创建外部数据包装器。

参数

name

要创建的外部数据包装器的名称。 该名称在数据库中必须唯一。

HANDLER *handler_function*

MPP数据库调用以检索外部表的执行函数的先前注册函数的名称。

*handler_function*必须不带任何参数，其返回类型必须是fdw_handler。

可以创建没有处理程序函数的外部数据包装器，但是只能使用此类包装器声明而不访问外部表。

VALIDATOR *validator_function*

MPP数据库调用以检查提供给外部数据包装程序的选项的先前注册函数的名称。 此函数还检查使用外部数据包装器的外部服务器，用户映射和外部表的选项。 如果未指定验证器函数或指定NO VALIDATOR，则MPP数据库在创建时不会检查选项。（根据实现的不同，外部数据包装程序可能会在运行时忽略或拒绝无效的选项。）

*validator_function*必须带有两个参数：一个是text[]类型，它包含存储在系统catalog中的选项数组，另一个是oid类型，它标识包含选项的系统catalog的OID。

返回类型将被忽略；*validator_function*应该使用ereport(ERROR)函数报告无效的选项。

OPTIONS (*option 'value' [, ...]*)

新的外部数据包装程序的选项。 选项名称必须唯一。 选项名称和值是特定于外部数据包装器的，并使用外部数据包装器的*validator_function*进行了验证。

mpp_execute { 'master' | 'any' | 'all segments' }

一个选项，用于标识外部数据包装器从其请求数据的主机：

- master（默认）— 从master主机请求数据。
- any — 向master主机或任一segment请求数据，具体取决于哪条路径的成本更低。
- all segments — 从所有segment请求数据。要支持此选项值，外部数据包装器必须具有将segment与数据匹配的策略。

可以在多个命令中指定mpp_execute选项：CREATE FOREIGN TABLE，CREATE SERVER和CREATE FOREIGN DATA WRAPPER。外部表设置优先于外部服务器设置，然后是外部数据包装器设置。

注解

外部数据包装器功能仍在开发中。查询的优化是原始的（大部分留给包装器）。

示例

创建一个无用的外部数据包装器，命名为dummy：

```
CREATE FOREIGN DATA WRAPPER dummy;
```

使用名为file_fdw_handler的处理函数创建一个名为file的外部数据包装器：

```
CREATE FOREIGN DATA WRAPPER file HANDLER file_fdw_handler;
```

创建一个名为mywrapper的外部数据包装器，其中包括一个选项：

```
CREATE FOREIGN DATA WRAPPER mywrapper OPTIONS (debug 'true');
```

兼容性

CREATE FOREIGN DATA WRAPPER符合ISO/IEC 9075-9（SQL/MED），但LIBRARY和VALIDATOR子句是扩展是例外，而MPP数据库中未实现标准子句LIBRARY和LANGUAGE。

但是请注意，整个SQL/MED函数尚未符合要求。

另见

[ALTER FOREIGN DATA WRAPPER](#), [DROP FOREIGN DATA WRAPPER](#), [CREATE SERVER](#), [CREATE USER MAPPING](#)

Parent topic: [SQL Command Reference](#)

CREATE FOREIGN TABLE

定义一个新的外部表。

概要

```
CREATE FOREIGN TABLE [ IF NOT EXISTS ] table_name ( [  
    column_name data_type [ OPTIONS ( option 'value' [, ... ] ) ] [ COLLATE collation ] [ column  
    [, ... ]  
    ] )  
    SERVER server_name  
    [ OPTIONS ( [ mpp_execute { 'master' | 'any' | 'all segments' } [, ] ] option 'value' [, ... ] ) ]
```

其中`column_constraint`为：

```
[ CONSTRAINT constraint_name ]  
{ NOT NULL |  
  NULL |  
  DEFAULT default_expr }
```

描述

CREATE FOREIGN TABLE在当前数据库中创建一个新的外部表。创建外部表的用户将成为其所有者。

如果您对表名称进行模式限定（例如，CREATE FOREIGN TABLE myschema.mytable ...），则MPP数据库将在指定模式中创建表。否则，将在当前模式中创建外部表。外部表的名称必须与同一模式中任何其他外部表，表，序列，索引或视图的名称不同。

因为CREATE FOREIGN TABLE自动创建一种数据类型，该数据类型表示与外部表的一行相对应的复合类型，所以外部表不能与同一模式中的任何现有数据类型具有相同的名称。

要创建外部表，您必须在外服务器上具有USAGE特权，并且对表中使用的所有列类型都具有USAGE特权。

参数

IF NOT EXISTS

如果已经存在同名关系，则不要抛出错误。在这种情况下，MPP数据库会发出通知。 请注意，不能保证现有关系类似于将要创建的关系。

table_name

要创建的外部表的名称（可以由模式指定）。

column_name

在新的外部表中创建的列的名称。

data_type

列的数据类型，包括数组说明符。

NOT NULL

该列不允许包含空值。

NULL

该列允许包含空值。这是默认值。

提供此子句仅是为了与非标准SQL数据库兼容。不建议在新应用中使用它。

DEFAULT *default_expr*

DEFAULT子句为其定义所在的列分配默认值。该值是无变量表达式；MPP数据库不允许子查询和对当前表中其他列的交叉引用。默认表达式的数据类型必须与列的数据类型匹配。

MPP数据库在未为列指定值的任何插入操作中使用默认表达式。如果列没有默认值，则默认值为null。

server_name

用于外部表的现有服务器的名称。有关定义服务器的详细信息，请参见[CREATE SERVER](#)。

OPTIONS (*option* 'value' [, ...])

新外部表或其列之一的选项。虽然选项名称必须唯一，但表选项和列选项可能具有相同的名称。选项名称和值是特定于外部数据包装程序的。MPP数据库使用外部数据包装程序的*validator_function*验证选项和值。

mpp_execute { 'master' | 'any' | 'all segments' }

一个选项，用于标识外部数据包装器从其请求数据的主机：

- master（默认设置）- 从master主机请求数据。
- any — 向master主机或任一segment请求数据，具体取决于哪条路径的成本更低。
- all segments — 从所有segment中请求数据。要支持此选项值，外部数据包装器必须具有将segment与数据匹配的策略。

外部表mpp_execute选项和受支持的特定模式的使用是特定于外部数据包装器的。

可以在多个命令中指定mpp_execute选项：CREATE FOREIGN TABLE，CREATE SERVER和CREATE FOREIGN DATA WRAPPER。外部表设置优先于外部服务器设置，然后是外部数据包装器设置。

示例

使用名为film_server的服务器创建一个名为films的外部表：

```
CREATE FOREIGN TABLE films (  
    code      char(5) NOT NULL,  
    title     varchar(40) NOT NULL,  
    did       integer NOT NULL,  
    date_prod date,  
    kind      varchar(10),  
    len       interval hour to minute  
)  
SERVER film_server;
```

兼容性

CREATE FOREIGN TABLE基本上符合SQL标准；但是，与[CREATE TABLE](#)一样，MPP数据库允许NULL约束和零列外部表。指定默认值的功能是MPP数据库扩展，以及mpp_execute选项。

另见

[ALTER FOREIGN TABLE](#), [DROP FOREIGN TABLE](#), [CREATE TABLE](#), [CREATE SERVER](#)

Parent topic: [SQL Command Reference](#)

CREATE FUNCTION

定义一个新函数。

概要

```
CREATE [OR REPLACE] FUNCTION name
  ( [ [argmode] [argname] argtype [ { DEFAULT | = } default_expr] [, ...] ] )
  [ RETURNS rettype
    | RETURNS TABLE ( column_name column_type [, ...] ) ]
{ LANGUAGE langname
| WINDOW
| IMMUTABLE | STABLE | VOLATILE | [NOT] LEAKPROOF
| CALLED ON NULL INPUT | RETURNS NULL ON NULL INPUT | STRICT
| [EXTERNAL] SECURITY INVOKER | [EXTERNAL] SECURITY DEFINER
| EXECUTE ON { ANY | MASTER | ALL SEGMENTS }
| COST execution_cost
| SET configuration_parameter { TO value | = value | FROM CURRENT }
| AS 'definition'
| AS 'obj_file', 'link_symbol' } ...
[ WITH ( { DESCRIBE = describe_function
        } [, ...] ) ]
```

描述

CREATE FUNCTION定义一个新函数。CREATE OR REPLACE FUNCTION要么创建新函数，要么替换现有定义。

新函数的名称不得与任何在同一模式中具有相同输入参数类型的现有函数匹配。但是，不同参数类型的函数可能共享一个名称（重载）。

要更新现有函数的定义，请使用CREATE OR REPLACE FUNCTION。不能以这种方式更改函数的名称或参数类型（这实际上会创建一个新的，不同的函数）。同样，CREATE OR REPLACE FUNCTION将不允许您更改现有函数的返回类型。为此，必须删除并重新创建该函数。使用OUT参数时，这意味着您

不能更改任何OUT参数的类型，除非删除该函数。如果删除然后重新创建函数，则必须删除引用旧函数的现有对象（规则，视图，触发器等）。使用CREATE OR REPLACE FUNCTION来更改函数定义，而不会破坏引用该函数的对象。

创建函数的用户将成为该函数的所有者。

为了能够创建函数，您必须对参数类型和返回类型具有USAGE特权。

有关创建函数的更多信息，请参见PostgreSQL文档的[用户定义函数](#)部分。

限制使用VOLATILE和STABLE函数

为了防止数据在MPP数据库的各个segment之间变得不同步，如果分类为STABLE或VOLATILE的任何函数包含SQL或以任何方式修改了数据库，则不能在segment级别执行该函数。例如，不允许在MPP数据库中的分布式数据上执行诸如random()或timeofday()之类的函数，因为它们可能会导致segment实例之间的数据不一致。

为了确保数据的一致性，可以在master数据库上评估并执行的语句中安全地使用VOLATILE和STABLE函数。例如，以下语句始终在master数据库上执行（没有FROM子句的语句）：

```
SELECT setval('myseq', 201);  
SELECT foo();
```

如果语句的FROM子句包含一个分布式表，并且FROM子句中使用的函数仅返回一组行，则可以在这些segment上执行：

```
SELECT * FROM foo();
```

此规则的一个例外是返回表引用（rangeFuncs）的函数或使用refCursor数据类型的函数。请注意，您无法从MPP数据库中的任何类型的函数返回refcursor。

函数易变性和EXECUTE ON属性

易变性属性（IMMUTABLE，STABLE，VOLATILE）和EXECUTE ON属性指定函数执行的两个不同方面。通常，易变性表示执行该函数的时间，EXECUTE ON表示执行该函数的位置。

例如，可以在查询计划时执行使用IMMUTABLE属性定义的函数，而必须对查询中的每一行执行带有VOLATILE属性的函数。具有EXECUTE ON MASTER属性的函数仅在master上执行，具有EXECUTE ON ALL SEGMENTS属性的函数仅在所有primary实例（而不是master）上执行。

请参阅*MPP数据库管理员手册*中的[使用函数和运算符](#)。

函数和复制表

在复制表上仅执行SELECT命令的用户定义函数可以在segment上运行。使用DISTRIBUTED REPLICATED子句创建的复制表将其所有行存储在每个segment上。函数在segment上读取它们是安全的，但是对复制表的更新必须在master实例上执行。

参数

name

要创建的函数的名称（可以由模式指定）。

argmode

参数的模式：IN，OUT，INOUT或VARIADIC。如果省略，则默认值为IN。只有OUT参数可以跟随声明为VARIADIC的参数。同样，OUT和INOUT参数不能与RETURNS TABLE表示法一起使用。

argname

参数的名称。某些语言（当前仅SQL和PL/pgSQL）允许您在函数主体中使用名称。对于其他语言，就函数本身而言，输入参数的名称只是额外的文档。但是您可以在调用函数时使用输入参数名称来提高可读性。在任何

情况下，输出参数的名称都是有效的，因为它在结果行类型中定义了列名称。（如果省略输出参数的名称，则系统将选择默认列名称。）

argtype

函数参数的数据类型（可以由模式指定）（如果有）。参数类型可以是基本类型，复合类型或域类型，或者可以引用表列的类型。

根据实现语言的不同，还可以允许指定伪类型，例如cstring。伪类型表示实际参数类型未完全指定，或者位于普通SQL数据类型集之外。

通过编写`tablename.columnname%TYPE`来引用列的类型。使用此函数有时可以使函数独立于表定义的更改。

default_expr

如果未指定参数，则用作默认值的表达式。该表达式必须对参数的参数类型具有强制性。只有IN和INOUT参数可以具有默认值。参数列表中紧随默认值的参数之后的每个输入参数也必须具有默认值。

rettype

返回数据类型（可以由模式指定）。返回类型可以是基本类型，复合类型或域类型，或者可以引用表列的类型。根据实现语言的不同，还可以允许指定伪类型，例如cstring。如果该函数不应该返回值，则将void指定为返回类型。

当有OUT或INOUT参数时，可以省略RETURNS子句。如果存在，则必须与输出参数隐含的结果类型相符：RECORD，如果有多个输出参数，或者与单个输出参数具有相同的类型。

SETOF修饰符表示该函数将返回一组项目，而不是单个项目。

通过编写`tablename.columnname%TYPE`来引用列的类型。

column_name

RETURNS TABLE语法中的输出列的名称。这实际上是声明命名OUT参数的另一种方法，除了RETURNS TABLE还暗含RETURNS SETOF。

column_type

RETURNS TABLE语法中输出列的数据类型。

langname

函数所使用的语言的名称。可以是SQL，C，internal或用户定义的过程语言的名称。有关MPP数据库中支持的过程语言，请参见[CREATE LANGUAGE](#)。为了向后兼容，名称可以用单引号引起来。

WINDOW

WINDOW表示该函数是窗口函数，而不是普通函数。当前这仅对用C编写的函数有用。替换现有函数定义时，不能更改WINDOW属性。

IMMUTABLE

STABLE

VOLATILE

LEAKPROOF

这些属性将有关函数的行为通知查询优化器。最多可以指定一种选择。如果这些都不出现，则默认为VOLATILE。由于MPP数据库当前对VOLATILE函数的使用受到限制，因此，如果一个函数确实是IMMUTABLE，则必须将其声明为可以不受限制地使用。

IMMUTABLE指示该函数无法修改数据库，并且在给定相同的参数值时始终返回相同的结果。它不进行数据库查找，或者使用其参数列表中不直接存在的信息。如果指定了此选项，则可以使用函数值立即替换具有全常数参数的任何函数调用。

STABLE表示该函数无法修改数据库，并且在单个表扫描中它将针对相同的参数值一致地返回相同的结果，但其结果可能会在SQL语句之间发生变化。对于结果取决于数据库查找，参数值（例如当前时区）等等的函数，这是适当的选择。还要注意，*current_timestamp*系列函数符合稳定条件，因为它们的值在事务中不会更改。

VOLATILE表示该函数值即使在一次表扫描中也可以更改，因此无法进行优化。从这个意义上说，相对来说很少有数据库函数是易变的。一些示例是random()，timeofday()。但是请注意，任何具有副作用的函数都必须归类为易失性，即使其结果是可以预测的，也可以防止调用被优化掉。一个示例是setval()。

LEAKPROOF表示该函数没有副作用。除了返回值以外，它不显示有关其参数的任何信息。例如，对于某些参数值而不是其他参数值抛出错误消息的函数，或者在任何错误消息中包含参数值的函数，都不是防漏的。查询优化器可以将防泄漏函数（但不能禁止其他功能）推入使

用security_barrier选项创建的视图中。请参阅[CREATE VIEW](#)和[CREATE RULE](#)。该选项只能由超级用户设置。

CALLED ON NULL INPUT

RETURNS NULL ON NULL INPUT

STRICT

CALLED ON NULL INPUT（默认值）表示该函数的某些参数为null时将正常调用该函数。然后，函数作者有责任检查空值（如有必要）并做出适当响应。RETURNS NULL ON NULL INPUT或STRICT表示该函数在任何参数为null时始终返回null。如果指定了此参数，则在参数为空时不执行该函数；而是自动假定为空结果。

[EXTERNAL] SECURITY INVOKER

[EXTERNAL] SECURITY DEFINER

SECURITY INVOKER（默认值）指示该函数将以调用该函数的用户权限执行。SECURITY DEFINER指定要使用创建该函数的用户的特权来执行该函数。允许使用EXTERNAL关键字来实现SQL一致性，但它是可选的，因为与SQL不同，此功能不仅适用于外部函数，还适用于所有函数。

EXECUTE ON ANY

EXECUTE ON MASTER

EXECUTE ON ALL SEGMENTS

EXECUTE ON属性指定在查询执行过程中调用函数时在何处（master或segment实例）执行。

EXECUTE ON ANY（默认值）表示该函数可以在master或任何segment实例上执行，并且无论在何处执行，它均返回相同的结果。MPP数据库确定函数在哪里执行。

EXECUTE ON MASTER表示该函数只能在master实例上执行。

EXECUTE ON ALL SEGMENTS表明对于每个调用，该函数必须在所有primary实例上执行，但不能在master实例上执行。该函数的总结果是所有segment实例的结果的UNION ALL。

有关使用EXECUTE ON属性的信息，请参阅[注释](#)。

COST *execution_cost*

一个正数，标识函数的估计执行成本，以[cpu_operator_cost](#)为单位。如果函数返回一个集合，则*execution_cost*会标识每个返回行的成本。如果未指定成本，则C语言和内部函数默认为1个单位，而其他语言的函数默认为100个单位。当您指定较大的*execution_cost*值时，优化器将尝试较少地评估函数。

configuration_parameter
value

进入该函数时，SET子句将一个值应用于会话配置参数。函数退出时，配

置参数将恢复为其先前的值。SET FROM CURRENT会将执行CREATE FUNCTION时当前的参数值保存为进入该函数时要应用的值。

definition

定义函数的字符串常量；含义取决于语言。它可以是内部函数名称，目标文件的路径，SQL命令或过程语言的文本。

obj_file, link_symbol

当C语言源代码中的函数名称与SQL函数的名称不同时，这种形式的AS子句用于可动态加载的C语言函数。字符串*obj_file*是包含动态可加载对象的文件的名称，而*link_symbol*是C语言源代码中的函数的名称。如果省略链接符号，则假定它与所定义的SQL函数的名称相同。所有函数的C名称必须不同，因此必须为重载的SQL函数赋予不同的C名称（例如，将参数类型用作C名称的一部分）。建议相对于\$libdir（位于\$GPHOME/lib）或通过动态库路径（由dynamic_library_path服务器配置参数设置）定位共享库。如果新安装位于其他位置，则可以简化版本升级。

describe_function

解析调用此函数的查询时要执行的回调函数的名称。回调函数返回一个表示结果类型的元组描述符。

注解

用于自定义函数的所有已编译代码（共享库文件）必须放在MPP数据库阵列（master和所有segment）中每个主机的相同位置。此位置也必须位于LD_LIBRARY_PATH中，以便服务器可以找到文件。建议在MPP阵列中的所有master实例上，相对于\$libdir（位于\$GPHOME/lib）或通过动态库路径（由dynamic_library_path服务器配置参数设置）定位共享库。

输入参数和返回值允许使用完整的SQL类型语法。但是，类型规范的某些详细信息（例如，类型为*numeric*的精度字段）是基础函数实现的职责，而CREATE FUNCTION命令无法识别或强制执行。

MPP数据库允许函数重载。相同的名称可以用于多个不同的函数，只要它们具有不同的输入参数类型即可。但是，所有函数的C名称都必须不同，因此必须为

重载的C函数赋予不同的C名称（例如，将参数类型用作C名称的一部分）。

如果两个函数具有相同的名称和输入参数类型，而忽略任何OUT参数，则认为它们是相同的。因此，例如，这些声明冲突：

```
CREATE FUNCTION foo(int) ...  
CREATE FUNCTION foo(int, out text) ...
```

具有不同参数类型列表的函数在创建时不会被认为是冲突的，但是如果提供了参数默认值，则它们在使用中可能会发生冲突。例如，考虑：

```
CREATE FUNCTION foo(int) ...  
CREATE FUNCTION foo(int, int default 42) ...
```

由于不清楚应调用哪个函数，因此调用foo(10)将失败。

当重复的CREATE FUNCTION调用引用相同的目标文件时，该文件仅被加载一次。要卸载并重新加载文件，请使用LOAD命令。

您必须对一种语言具有USAGE特权，才能使用该语言定义函数。

使用美元引号而不是常规的单引号语法编写函数定义字符串通常会很有帮助。如果不使用美元引号，则必须通过将它们加倍来转义函数定义中的任何单引号或反斜杠。用美元引用的字符串常量由一个美元符号（\$），一个零个或多个字符的可选标记，另一个美元符号，构成字符串内容的任意字符序列，一个美元符号以及与开始此美元引用相同的标记和一个美元符号组成。在用美元引用的字符串中，可以使用单引号，反斜杠或任何字符而无需转义。字符串内容始终按原义书写。例如，以下是两种使用美元引号指定字符串"Dianne's horse"的方法：

```
$$Dianne's horse$$  
$SomeTag$Dianne's horse$SomeTag$
```

如果将SET子句附加到函数，则在函数内部针对相同变量执行的SET LOCAL命令的作用仅限于该函数；函数退出时，配置参数的先前值仍会恢复。但是，普通的SET命令（不包含LOCAL）会覆盖CREATE FUNCTION SET子句，就像以前的SET LOCAL命令所使用的一样。除非当前事务回滚，否则该命令的效果将在函数退出后继续存在。

如果将带有VARIADIC参数的函数声明为STRICT，则严格性检查将测试可变参数数组整体是否为非空。如果数组具有空元素，PL/pgSQL仍将调用该函数。

用CREATE OR REPLACE FUNCTION替换现有函数时，更改参数名称存在限制。您无法更改已分配给任何输入参数的名称（尽管您可以将名称添加到以前没有输入参数的参数中）。如果有多个输出参数，则不能更改输出参数的名称，因为这将更改描述函数结果的匿名复合类型的列名称。进行这些限制是为了确保函数的现有调用在被替换时不会停止工作。

将函数与查询一起用于分布式数据

在某些情况下，如在FROM子句中指定的表中的数据分布在MPP数据库segment上，MPP数据库不支持在查询中使用函数。例如，此SQL查询包含函数func()：

```
SELECT func(a) FROM table1;
```

如果满足以下所有条件，则该函数不支持在查询中使用：

- 表table1的数据分布在MPP数据库segment上。
- 函数func()从分布式表中读取或修改数据。
- 函数func()返回多个行或采用来自table1的参数（a）。

如果不满足任何条件，则支持该函数。具体来说，如果满足以下任一条件，则支持该函数：

- 函数func()不会访问分布式表中的数据，也不会访问仅在MPP数据库master

上的数据。

- 表table1是仅在master上。
- 函数func()仅返回一行，并且仅接受常量值的输入参数。如果可以将其更改为不需要输入参数，则支持该函数。

使用EXECUTE ON属性

大多数执行查询以访问表的函数只能在master上执行。但是，仅对复制表执行SELECT查询的函数可以在segment上运行。如果函数访问哈希分布表或随机表，则应使用EXECUTE ON MASTER属性定义该函数。否则，在复杂的查询中使用该函数时，该函数可能会返回错误的结果。如果没有该属性，优化器优化可能会确定将函数调用推到segment实例将是有益的。

这些是使用EXECUTE ON MASTER或EXECUTE ON ALL SEGMENTS属性定义的函数的限制：

- 该函数必须是返回集合的函数。
- 该函数不能在查询的FROM子句中。
- 该函数不能在带有FROM子句的查询的SELECT列表中。
- 包含该函数的查询从GPORCA退回到Postgres查询计划程序。

示例

一个非常简单的加法函数：

```
CREATE FUNCTION add(integer, integer) RETURNS integer
AS 'select $1 + $2;'
LANGUAGE SQL
IMMUTABLE
RETURNS NULL ON NULL INPUT;
```

在PL/pgSQL中使用参数名称递增一个整数：

```
CREATE OR REPLACE FUNCTION increment(i integer) RETURNS
integer AS $$
    BEGIN
        RETURN i + 1;
    END;
$$ LANGUAGE plpgsql;
```

为PL/pgSQL函数增加每个查询的默认segment主机内存：

```
CREATE OR REPLACE FUNCTION function_with_query() RETURNS
SETOF text AS $$
    BEGIN
        RETURN QUERY
        EXPLAIN ANALYZE SELECT * FROM large_table;
    END;
$$ LANGUAGE plpgsql
SET statement_mem='256MB';
```

使用多态类型返回ENUM数组：

```
CREATE TYPE rainbow AS ENUM('red','orange','yellow','green','blue','indigo','violet');
CREATE FUNCTION return_enum_as_array( anyenum, anyelement, anyelement )
    RETURNS TABLE (ae anyenum, aa anyarray) AS $$
    SELECT $1, array[$2, $3]
$$ LANGUAGE SQL STABLE;

SELECT * FROM return_enum_as_array('red'::rainbow, 'green'::rainbow, 'blue'::rainbow);
```

返回包含多个输出参数的记录：

```
CREATE FUNCTION dup(in int, out f1 int, out f2 text)
    AS $$ SELECT $1, CAST($1 AS text) || ' is text' $$
    LANGUAGE SQL;

SELECT * FROM dup(42);
```

您可以使用明确命名的复合类型来更详细地执行相同的操作：

```
CREATE TYPE dup_result AS (f1 int, f2 text);
CREATE FUNCTION dup(int) RETURNS dup_result
  AS $$ SELECT $1, CAST($1 AS text) || ' is text' $$
  LANGUAGE SQL;

SELECT * FROM dup(42);
```

返回多列的另一种方法是使用TABLE函数：

```
CREATE FUNCTION dup(int) RETURNS TABLE(f1 int, f2 text)
  AS $$ SELECT $1, CAST($1 AS text) || ' is text' $$
  LANGUAGE SQL;

SELECT * FROM dup(4);
```

该函数在EXECUTE ON ALL SEGMENTS中定义，以在所有primary实例上运行。SELECT命令执行返回在每个segment实例上运行时间的函数。

```
CREATE FUNCTION run_on_segs (text) returns setof text as $$
begin
  return next ($1 || ' - ' || now()::text );
end;
$$ language plpgsql VOLATILE EXECUTE ON ALL SEGMENTS;

SELECT run_on_segs('my test');
```

此功能在parts表中查找part名称。 parts表是复制表，因此该函数可以在master或primary上执行。

```
CREATE OR REPLACE FUNCTION get_part_name(partno int) RETURNS text AS
$$
DECLARE
    result text := '';
BEGIN
    SELECT part_name INTO result FROM parts WHERE part_id = partno;
    RETURN result;
END;
$$ LANGUAGE plpgsql;
```

如果在master上执行SELECT get_part_name(100);，该函数在master上执行。（master实例将查询定向到单个primary。）如果orders是分布式表，并且您执行以下查询，则get_part_name()函数将在primary上执行。

```
SELECT order_id, get_part_name(orders.part_no) FROM orders;
```

兼容性

CREATE FUNCTION在SQL：1999及更高版本中定义。MPP数据库版本相似，但不完全兼容。这些属性不是可移植的，不同的可用语言也不是。

为了与某些其他数据库系统兼容，可以在`argname`之前或之后写入`argmode`。但是只有第一种方法是符合标准的。

对于参数默认值，SQL标准仅使用DEFAULT关键字指定语法。在T-SQL和Firebird中使用=语法。

另见

[ALTER FUNCTION](#) , [DROP FUNCTION](#) , [LOAD](#)

Parent topic: [SQL Command Reference](#)

CREATE GROUP

定义一个新的数据库角色。

概要

```
CREATE GROUP name [[WITH] option [ ... ]]
```

其中`option`可以是：

```
SUPERUSER | NOSUPERUSER
| CREATEDB | NOCREATEDB
| CREATEROLE | NOCREATEROLE
| CREATEUSER | NOCREATEUSER
| CREATEEXTTABLE | NOCREATEEXTTABLE
[ ( attribute='value'[ , ... ] ) ]
    where attributes and value are:
        type='readable'|'writable'
        protocol='gpfdist'|'http'
| INHERIT | NOINHERIT
| LOGIN | NOLOGIN
| CONNECTION LIMIT conlimit
| [ ENCRYPTED | UNENCRYPTED ] PASSWORD 'password'
| VALID UNTIL 'timestamp'
| IN ROLE rolename [ , ... ]
| ROLE rolename [ , ... ]
| ADMIN rolename [ , ... ]
| RESOURCE QUEUE queue_name
| RESOURCE GROUP group_name
| [ DENY deny_point ]
| [ DENY BETWEEN deny_point AND deny_point ]
```

描述

CREATE GROUP是[CREATE ROLE](#)的别名。

兼容性

SQL标准中没有CREATE GROUP语句。

另见

CREATE ROLE

Parent topic: [SQL Command Reference](#)

CREATE INDEX

定义一个新索引。

概要

```
CREATE [UNIQUE] INDEX [name] ON table_name [USING method]  
    ( {column_name | (expression)} [COLLATE parameter] [opclass] [ ASC | DESC ] [ NULLS  
    [ WITH ( storage_parameter = value [, ... ] ) ] ]  
    [ TABLESPACE tablespace ]  
    [ WHERE predicate ]
```

描述

CREATE INDEX在指定表的指定列上构造索引。索引主要用于增强数据库性能（尽管使用不当会导致性能降低）。

索引的键字段指定为列名，或者指定为用括号括起来的表达式。如果index方法支持多列索引，则可以指定多个字段。

索引字段可以是根据表行的一个或多个列的值计算的表达式。该功能可用于基于基本数据的某种转换来快速访问数据。例如，在upper(col)上计算的索引将允许子句WHERE upper(col) = 'JIM'使用索引。

MPP数据库提供了索引方法B树，位图，GiST，SP-GiST和GIN。用户还可以定义自己的索引方法，但这相当复杂。

存在WHERE子句时，将创建部分索引。部分索引是仅包含表一部分的条目的索引，通常索引的部分比表的其余部分更有用。例如，如果您有一个既包含开票订单又包含未开票订单的表，其中未开票订单仅占总表的一小部分，但最常被选择，则可以通过仅在该部分上创建索引来提高性能。

WHERE子句中使用的表达式可能仅引用基础表的列，但它可以使使用所有列，而不仅仅是被索引的列。WHERE中也禁止子查询和聚合表达式。相同的限制适用于作为表达式的索引字段。

索引定义中使用的所有函数和运算符必须是不可变的。它们的结果必须仅取决于其参数，而不得取决于任何外部影响（例如另一个表的内容或参数值）。此限制可确保索引的行为得到良好定义。要在索引表达式或WHERE子句中使用用户定义的函数，请记住在创建函数时将其标记为IMMUTABLE。

参数

UNIQUE

创建索引并每次添加数据时，检查表中是否有重复值。重复的条目将产生错误。唯一索引仅适用于B树索引。在MPP数据库中，仅当索引键的列与MPP分布键相同（或超集）时，才允许唯一索引。在分区表上，唯一索引仅在单个分区中受支持 - 不可以跨所有分区。

name

要创建的索引的名称。索引始终与其父表在相同的模式中创建。如果省略该名称，则MPP数据库将基于父表的名称和索引列的名称选择一个合适的名称。

table_name

要建立索引的表的名称（可以由模式指定）。

method

要使用的索引方法的名称。选择为btree，bitmap，gist，spgist和gin。
默认方法是btree。

当前，仅B树，GiST和GIN索引方法支持多列索引。默认情况下，最多可以指定32个字段。当前只有B树支持唯一索引。

GPORCA仅支持B树，位图和GiST索引。GPORCA会忽略使用不受支持的索引方法创建的索引。

column_name

在其上创建索引的表的列的名称。仅B树，位图，GiST和GIN索引方法支持多列索引。

expression

一种基于表的一个或多个列的表达式。表达式通常必须用括号括起来，如语法所示。但是，如果表达式具有函数调用的形式，则可以省略括号。

collation

索引使用的归类名称。默认情况下，索引使用为要建立索引的列声明的归类或要建立索引的表达式的结果归类。具有非默认归类的索引对于涉及使用非默认归类的表达式的查询很有用。

opclass

运算符类的名称。operator类标识该列的索引要使用的运算符。例如，在四字节整数上的B树索引将使用int4_ops类（此运算符类包含四字节整数的比较函数）。实际上，列数据类型的默认运算符类通常就足够了。拥有运算符类的要点是，对于某些数据类型，可能会有不止一种有意义的排序。例如，复数数据类型可以按绝对值或实数进行排序。为此，我们可以为数据类型定义两个运算符类，然后在创建索引时选择适当的类。

ASC

指定升序排序（默认）。

DESC

指定降序排序。

NULLS FIRST

指定null排在非null之前。当指定DESC时，这是默认设置。

NULLS LAST

指定空值在非空值之后排序。如果未指定DESC，这是默认设置。

storage_parameter

特定于索引方法的存储参数的名称。每个索引方法都有其自己的一组允许的存储参数。

FILLFACTOR-B树，位图，GiST和SP-GiST索引方法都接受此参数。索引的FILLFACTOR是一个百分比，它确定索引方法将尝试打包索引页面的程度。对于B树，在初始索引构建期间以及在向右扩展索引时（添加新的最大键值），叶子页将填充到该百分比。如果页面随后完全填满，它们将被拆分，从而导致索引效率逐渐下降。B树使用默认填充因子90，但可以选择10到100之间的任何整数值。如果表是静态的，则fillfactor 100最好最大程度地减小索引的物理大小，但是对于大量更新的表，较小的fillfactor更好地最小化对页面拆分的需求。其他索引方法以不同但大致相似的方式使用fillfactor；默认的填充因子因方法而异。

BUFFERING - 除FILLFACTOR之外，GiST索引还接受BUFFERING参数。BUFFERING确定MPP数据库是否使用PostgreSQL文档中[GiST缓冲构建](#)中描述的缓冲构建技术构建索引。如果设置为OFF，则禁用它；如果设置为ON，则启用；将其设置为AUTO时，最初将其禁用；但是，一旦索引大小达到[有效缓存大小](#)，它就会即时打开。默认为AUTO。

FASTUPDATE - GIN索引方法接受FASTUPDATE存储参数。

FASTUPDATE是一个布尔参数，用于禁用或启用GIN索引快速更新技术。值ON启用快速更新（默认值），而值OFF禁用它。有关更多信息，请参见PostgreSQL文档中的[GIN快速更新技术](#)。

Note: 通过ALTER INDEX关闭FASTUPDATE可以防止将来的插入进入挂起的索引条目列表，但本身不会刷新以前的条目。您可能需要稍后对表进行VACUUM处理，以确保清空待处理列表。

tablespace_name

在其中创建索引的表空间。如果未指定，则使用默认表空间。

predicate

部分索引的约束表达式。

注解

可以为索引的每一列指定一个运算符类。operator类标识该列的索引要使用的运算符。例如，在四字节整数上的B树索引将使用int4_ops类。此运算符类包含用于四字节整数的比较函数。实际上，列数据类型的默认运算符类通常就足够了。拥有运算符类的要点是，对于某些数据类型，可能会有不止一种有意义的排序。例如，我们可能想按绝对值或实数对复数数据类型进行排序。为此，我们可以为数据类型定义两个运算符类，然后在创建索引时选择适当的类。

对于支持排序扫描的索引方法（当前仅支持B树），可以指定可选的子句ASC，DESC，NULLS FIRST和/或NULLS LAST来修改索引的排序顺序。由于可以向前或向后扫描有序索引，因此创建单列DESC索引通常没有用 - 常规索引已经可以使用排序顺序。这些选项的值在于可以创建与混合排序查询所请求的排序顺序匹配的多列索引，例如SELECT ... ORDER BY x ASC, y DESC。如果您需要在依赖索引以避免排序步骤的查询中支持“nulls sort low”行为，而不是默认的“nulls sort high”行为，则NULLS选项很有用。

对于大多数索引方法，创建索引的速度取决于maintenance_work_mem的设置。较大的值将减少索引创建所需的时间，只要您不使其大于实际可用的内存量即可，这会促使计算机进行交换。

在分区表上创建索引后，该索引将传播到MPP数据库创建的所有子表。不支持在由MPP数据库创建的表上创建索引以供分区表使用。

仅当索引列与MPP分布键列相同（或超集）时，才允许使用UNIQUE索引。

附加优化表上不允许使用UNIQUE索引。

可以在分区表上创建UNIQUE索引。但是，唯一性仅在分区内有效。分区之间不强制唯一性。例如，对于具有基于年份的分区和基于季度的子分区的分区表，仅在每个单独的季度分区上实施唯一性。季度分区之间不强制唯一性。

默认情况下，索引不用于IS NULL子句。在这种情况下，使用索引的最佳方法是使用IS NULL谓词创建部分索引。

对于具有100到100,000个不同值的列，位图索引的性能最佳。对于具有超过

100,000个不同值的列，位图索引的性能和空间效率会下降。位图索引的大小与表中的行数乘以索引列中不同值的数量成正比。

少于100个不同值的列通常不会从任何类型的索引中受益太多。例如，对于男性和女性仅具有两个不同值的性别列将不是索引的理想选择。

MPP数据库的先前版本也具有R树索引方法。该方法已被删除，因为它与GiST方法相比没有明显的优势。如果指定了USING rtree，则CREATE INDEX会将其解释为USING gist。

有关GiST索引类型的更多信息，请参见[PostgreSQL文档](#)。

哈希索引的使用已在MPP数据库中禁用。

示例

要在films表的title列上创建B树索引：

```
CREATE UNIQUE INDEX title_idx ON films (title);
```

要在表employee中的gender列上创建位图索引：

```
CREATE INDEX gender_bmp_idx ON employee USING bitmap  
(gender);
```

要在表达式lower(title)上创建索引，以允许有效的不区分大小写的搜索：

```
CREATE INDEX ON films ((lower(title)));
```

（在此示例中，我们选择省略索引名称，因此系统将选择一个名称，通常为films_lower_idx。）

要使用非默认排序规则创建索引：

```
CREATE INDEX title_idx_german ON films (title COLLATE "de_DE");
```

要创建具有非默认填充因子的索引：

```
CREATE UNIQUE INDEX title_idx ON films (title) WITH  
(fillfactor = 70);
```

要创建禁用了快速更新的GIN索引：

```
CREATE INDEX gin_idx ON documents_table USING gin (locations) WITH (fastupdate = off);
```

要在表films的列code上创建索引并使索引驻留在表空间indexspace中：

```
CREATE INDEX code_idx ON films(code) TABLESPACE indexspace;
```

要在point属性上创建GiST索引，以便我们可以对转换函数的结果有效地使用box运算符：

```
CREATE INDEX pointloc ON points USING gist (box(location,location));  
SELECT * FROM points WHERE box(location,location) && '(0,0),(1,1)::box;
```

兼容性

CREATE INDEX是MPP数据库语言的扩展。SQL标准中没有索引的规定。

MPP数据库不支持并发创建索引（不支持CONCURRENTLY关键字）。

另见

[ALTER INDEX](#), [DROP INDEX](#), [CREATE TABLE](#), [CREATE OPERATOR CLASS](#)

Parent topic: [SQL Command Reference](#)

CREATE LANGUAGE

定义一种新的过程语言。

概要

```
CREATE [ OR REPLACE ] [ PROCEDURAL ] LANGUAGE name

CREATE [ OR REPLACE ] [ TRUSTED ] [ PROCEDURAL ] LANGUAGE name
  HANDLER call_handler [ INLINE inline_handler ]
  [ VALIDATOR valfunction ]
```

描述

CREATE LANGUAGE在MPP数据库中注册了新的过程语言。 随后，可以用这种新语言定义函数和触发过程。

Note: MPP数据库的过程语言已被制成“扩展名”，因此应与[CREATE EXTENSION](#)一起安装，而不是CREATE LANGUAGE。直接使用CREATE LANGUAGE应该仅限于扩展安装脚本。如果数据库中存在“裸”语言（可能是升级的结果），则可以使用CREATE EXTENSION *langname* FROM unpackaged将其转换为扩展名。

超级用户可以在MPP数据库中注册新语言。数据库所有者还可以在该数据库中注册pg_pltemplate catalog中列出的tmpldbcreate字段为true的任何语言。默认配置仅允许数据库所有者注册受信任的语言。语言的创建者成为其所有者，以后可以将其删除，重命名或将所有权分配给新所有者。

CREATE OR REPLACE LANGUAGE将创建新语言或替换现有定义。如果该语言已经存在，则其参数将根据pg_pltemplate中指定的值或从pg_pltemplate中获取的值进行更新，但该语言的所有权和权限设置不会更改，并且假定使用该语言编写的任何现有函数仍然有效。除了创建语言的正常特权要求外，用户还必须是现有语言的超级用户或所有者。REPLACE情况主要用于确保该语言存在。如果该语言具有pg_pltemplate条目，则REPLACE实际上不会更改有关现有定义的任何内容，除非在异常情况下自创建该语言以来已对pg_pltemplate条目进行了修改。

CREATE LANGUAGE有效地将语言名称与处理程序函数关联，该处理函数负责执行以该语言编写的函数。对于以过程语言（C或SQL以外的语言）编写的函数，数据库服务器没有有关如何解释函数源代码的内置知识。该任务将传递给知道该语言详细信息的特殊处理程序。处理程序可以执行解析，语法分析，执行等所有工作，也可以充当MPP数据库与编程语言的现有实现之间的桥梁。处理程序本身是一种C语言函数，已编译为共享对象并按需加载，就像其他任何C函数一样。这些过程语言程序包包含在标准的MPP数据库发行版中：PL/pgSQL，PL/Perl和PL/Python。还为PL/Java和PL/R添加了语言处理程序，但是MPP数据库未预安装这些语言。有关使用这些过程语言开发函数的更多信息，请参见PostgreSQL文档中有关[过程语言](#)的主题。

PL/Perl，PL/Java和PL/R库分别需要安装正确版本的Perl，Java和R。

CREATE LANGUAGE命令有两种形式。在第一种形式中，用户指定所需语言

的名称，MPP数据库服务器使用pg_pltemplate系统catalog来确定正确的参数。在第二种形式中，用户指定语言参数以及语言名称。您可以使用第二种形式来创建pg_pltemplate中未定义的语言。

当服务器在pg_pltemplate catalog中找到给定语言名称的条目时，即使命令包含语言参数，它也会使用catalog数据。此行为简化了旧转储文件的加载，这些文件可能包含有关语言支持函数的过时信息。

参数

TRUSTED

TRUSTED指定该语言不会授予用户原本就不会访问的数据的访问权限。如果在注册语言时省略此关键字，则只有具有MPP数据库超级用户特权的用户才能使用该语言创建新函数。

PROCEDURAL

这是一个干扰词。

name

新过程语言的名称。该名称在数据库中的语言之间必须唯一。包含对plpgsql，plperl和plpythonu的内置支持。默认情况下，在MPP数据库中已默认安装了语言plpgsql（PL/pgSQL）和plpythonu（PL/Python）。

HANDLER *call_handler*

如果服务器在pg_pltemplate中有用于指定语言名称的条目则忽略。先前注册的函数的名称，将调用该函数来执行过程语言函数。程序语言的调用处理程序必须使用编译后的语言编写，例如使用版本1调用约定的C语言，并在MPP数据库中注册为不带任何参数的函数，并返回language_handler类型，该仅是用于标识函数的占位符类型作为呼叫处理程序。

INLINE *inline_handler*

以前注册的函数的名称，该函数被调用以执行用DO命令创建的该语言的匿名代码块。如果未指定inline_handler函数，则该语言不支持匿名代码块。处理函数必须采用类型为internal的一个参数，即DO命令的内部表示形式。该函数通常返回void。处理程序的返回值将被忽略。

VALIDATOR *valfunction*

如果服务器在pg_pltemplate中有用于指定语言名称的条目则忽略。先前注册的函数的名称，将调用该函数来执行过程语言函数。程序语言的调用处理程序必须使用编译式语言编写，例如使用版本1调用约定的C语言，并在MPP数据库中注册为不带任何参数的函数，并返回language_handler类型，该仅是用于标识函数的占位符类型作为呼叫处理程序。

注解

默认情况下，PL/pgSQL语言已在所有数据库中注册。PL/Python语言扩展已安装但未注册。

系统catalog pg_language记录有关当前安装的语言的信息。

要使用过程语言创建函数，用户必须具有该语言的USAGE特权。默认情况下，USAGE被授予PUBLIC（每个人）用于受信任的语言。如果需要，可以将其撤消。

过程语言是各个数据库的本地语言。您为单个数据库创建和删除语言。

如果服务器在pg_pltemplate中没有该语言的条目，则调用处理程序函数和验证器函数（如果有）必须已经存在。但是，当有一个条目时，函数就不必已经存在了。如果数据库中不存在它们，它们将被自动定义。

任何实现一种语言的共享库都必须位于MPP数据库阵列中所有segment主机上的LD_LIBRARY_PATH位置中。

示例

创建任何标准过程语言的首选方法是使用CREATE EXTENSION而不

是CREATE LANGUAGE。 例如：

```
CREATE EXTENSION plperl;
```

对于pg_pltemplate catalog中未知的语言：

```
CREATE FUNCTION plsample_call_handler() RETURNS
language_handler
  AS '$libdir/plsample'
  LANGUAGE C;
CREATE LANGUAGE plsample
  HANDLER plsample_call_handler;
```

兼容性

CREATE LANGUAGE是MPP数据库扩展。

另见

[ALTER LANGUAGE](#), [CREATE EXTENSION](#), [CREATE FUNCTION](#), [DROP EXTENSION](#), [DROP LANGUAGE](#), [GRANT DO](#)

Parent topic: [SQL Command Reference](#)

CREATE OPERATOR

定义一个新的运算符。

概要

```
CREATE OPERATOR name (  
    PROCEDURE = funcname  
    [, LEFTARG = lefttype] [, RIGHTARG = righttype]  
    [, COMMUTATOR = com_op] [, NEGATOR = neg_op]  
    [, RESTRICT = res_proc] [, JOIN = join_proc]  
    [, HASHES] [, MERGES] )
```

描述

CREATE OPERATOR定义一个新的运算符。定义运算符的用户将成为其所有者。

运算符名称是以下列表中最多NAMEDATALEN-1（默认为63）个字符的序列：
+ - * / < > = ~ ! @ # % ^ & | ` ?

名称的选择有一些限制：

- --和/*不能出现在运算符名称的任何位置，因为它们将被用作注释的开头。
- 多字符运算符名称不能以+或-结尾，除非该名称也包含以下至少一个字符：~ ! @ # % ^ & | ` ?

例如，@-是允许的运算符名称，但*-不是。此限制使MPP数据库可以解析与SQL兼容的命令，而无需在标记之间使用空格。

不建议将=>用作运算符。在将来的版本中可能完全不允许使用它。

运算符!=在输入时映射为<>，因此这两个名称始终相等。

必须至少定义LEFTARG和RIGHTARG之一。对于二元运算符，必须同时定义两者。对于右一元运算符，仅应定义LEFTARG，而对于左一元运算符，仅应定义RIGHTARG。

*funcname*过程必须事先使用CREATE FUNCTION定义，必须是IMMUTABLE，并且必须定义为接受所指示类型的正确数量的参数（一个或两个）。

其他子句指定可选的运算符优化子句。适当时应提供这些子句，以加快使用运算符的查询的速度。但是，如果提供它们，则必须确保它们是正确的。错误地使用优化子句可能导致服务器进程崩溃，错误的输出或其他意外结果。如果您不确定优化子句，则可以始终忽略该子句。

为了能够创建运算符，您必须对参数类型和返回类型具有USAGE特权，并且对基础函数具有EXECUTE特权。如果指定了换向或负运算符，则您必须拥有这些运算符。

参数

name

要定义的运算符的名称（可以由模式指定）。如果同一模式中的两个运算符对不同的数据类型进行运算，则它们可以具有相同的名称。

funcname

用于实现此运算符的函数（必须为IMMUTABLE函数）。

lefttype

运算符的左操作数的数据类型（如果有）。左一元运算符将忽略此选项。

righttype

运算符的右操作数的数据类型（如果有）。右一元运算符将忽略此选项。

com_op

可选的COMMUTATOR子句命名一个运算符，该运算符是要定义的运算符

的交换器。我们说如果对于所有可能的输入值 x, y ， $(x A y)$ 等于 $(y B x)$ ，则算符A是算符B的交换子。请注意，B也是A的交换子。例如，特定数据类型的运算符<和>通常是彼此的交换子，而运算符+通常是与自身交换的。但是运算符-通常不与任何东西互换。可交换运算符的左操作数类型与其交换器的右操作数类型相同，反之亦然。因此，只需在COMMUTATOR子句中提供交换器运算符的名称即可。

neg_op

可选的NEGATOR子句命名一个运算符，该运算符是要定义的运算符的否定符。我们说如果两个操作符都返回布尔结果，并且对于所有可能的输入 x, y ， $(x A y)$ 等于 $\text{NOT } (x B y)$ ，则运算符A是运算符B的否定符。请注意，B也是A的否定符。例如，<和>=是大多数数据类型的否定符对。运算符的否定符必须具有与要定义的运算符相同的左右操作数类型，因此，在NEGATOR子句中仅需要指定运算符名称。

res_proc

可选的RESTRICT为运算符命名了一个限制选择性估计函数。请注意，这是一个函数名称，而不是运算符名称。RESTRICT子句仅对返回布尔值的二元运算符有意义。限制选择性估计器背后的想法是猜测表中哪几部分行将满足以下形式的WHERE子句条件：

```
column OP constant
```

用于当前运算符和特定的常量值。这有助于优化器了解具有这种形式的WHERE子句将消除多少行。

You can usually just use one of the following system standard estimator functions for many of your own operators:

eqsel 为 =

neqsel 为 <>

scalartsel 为 < 或 <=

scalargtsel 为 > 或 >=

join_proc

可选的JOIN子句为运算符命名联接选择性估计函数。 请注意，这是一个函数名称，而不是运算符名称。 JOIN子句仅对返回布尔值的二元运算符有意义。 连接选择性估计器背后的想法是猜测一对表中的哪几部分行将满足以下WHERE子句条件

```
table1.column1 OP table2.column2
```

对于当前的运算符。通过让优化器找出几个可能的连接序列中哪一个可能花费最少的工作，可以帮助优化器。

通常，您可以对许多自己的运算符使用以下系统标准联接选择性估计器函数之一：

eqjoinse1 为 =

neqjoinse1 为 <>

scalartjjoinse1 为 < 或 <=

scalartgjoinse1 为 > 或 >=

areajoinse1 为基于2D区域的比较

positionjoinse1 为基于2D位置的比较

contjoinse1 为基于2D包含的比较

HASHES

可选的HASHES子句告诉系统允许使用哈希联接方法进行基于此运算符的联接。 HASHES仅对返回布尔值的二元运算符有意义。 哈希联接运算符只能对哈希到同一哈希代码的一对左右值返回true。 如果将两个值放在不同的哈希存储桶中，则联接将永远不会比较它们，隐式地假设联接运算符的结果必须为false。 因此，为不表示相等性的运算符指定HASHES永远没有意义。

在大多数情况下，仅在两侧采用相同数据类型的运算符支持散列。但是，您可以为两个或多个数据类型设计兼容的哈希函数，即使这些值的表示方式不同，这些函数也会为“相等”值生成相同的哈希码。

要标记为HASHES，联接运算符必须出现在哈希索引运算符类中。如果不存在此类运算符类，则尝试在哈希联接中使用运算符将在运行时失败。系统需要运算符类为运算符的输入数据类型查找特定于数据类型的哈希函数。在创建运算符类之前，还必须提供合适的哈希函数。在准备散列函数时，请格外小心，因为存在与机器相关的方法，散列函数可能无法正确运行。例如，在满足IEEE浮点标准的计算机上，负零和正零是不同的值（不同的位模式），但被定义为相等。如果浮点值可以包含负零，则将其定义为生成与正零相同的哈希值。

可哈希连接的运算符必须具有出现在同一运算符族中的换向符（如果两个操作数数据类型相同，则为自身，或者如果两个运算符数据类型不同，则为相关的相等运算符）。否则，使用运算符时可能会发生优化器错误。为了更好的优化，支持多种数据类型的哈希运算符族应为数据类型的每种组合提供相等运算符。

Note: 哈希可连接运算符的基础函数必须标记为不可变或稳定的；标记为volatile的运算符将不会使用。如果可哈希连接的运算符具有标记为严格的基础函数，则该函数也必须是完整的，对于任何两个非空输入都返回true或false，并且不返回null。

MERGES

MERGES子句（如果存在）告诉系统允许使用merge-join方法基于该运算符进行联接。MERGES仅对返回布尔值的二元运算符有意义，并且在实践中，该运算符必须表示某些数据类型或一对数据类型的相等性。

合并联接基于的想法是将左表和右表排序，然后并行扫描它们。这意味着这两种数据类型都必须能够完全排序，并且联接运算符必须是仅对按排序顺序位于相等位置的值对成功的运算符。实际上，这意味着联接运算符必须表现得像相等运算符。但是，您可以合并联接两种不同的数据类型，只要它们在逻辑上是兼容的。例如，smallint-versus-integer等价运算符是可合并连接的。仅需要将两种数据类型都放入逻辑兼容序列的排序运算符。要标记为MERGES，联接运算符必须显示为btree索引运算符族的相等成员。创建运算符时不会强制执行此操作，因为引用运算符族直到稍后才存在。但是，除非可以找到匹配的运算符族，否则该运算符实际上不会用于

合并联接。因此，MERGE标记可作为对优化器的建议，以寻找匹配的运算符族。

可合并连接的运算符必须具有出现在同一运算符族中的换向器。如果两个操作数数据类型相同，则其为数据本身；如果数据类型不同，则为相关的相等运算符。如果没有合适的换向器，则在使用运算符时会发生优化器错误。同样，尽管不是严格要求，但是支持多种数据类型的btree运算符族应该能够为数据类型的每种组合提供相等运算符；这样可以实现更好的优化。

Note: SORT1，SORT2，LTCMP和GTCMP以前用于指定与可合并连接运算符关联的排序运算符的名称。现在，通过查看B树运算符族可以找到有关关联运算符的信息。指定这些运算符中的任何一个都将被忽略，除非它将隐式地将MERGES设置为true。

注解

用于实现运算符的任何函数都必须定义为IMMUTABLE。

在CREATE OPERATOR中不能指定运算符的词法优先级，因为解析器的优先级行为是固定的。有关优先级的详细信息，请参见PostgreSQL文档中的[运算符优先级](#)。

使用[DROP OPERATOR](#)从数据库中删除用户定义的运算符。使用[ALTER OPERATOR](#)修改数据库中的运算符。

示例

假设我们已经创建了complex类型的定义，这是创建一个用于添加两个复数的运算符的示例。首先定义完成工作的函数，然后定义运算符：

```
CREATE FUNCTION complex_add(complex, complex)
  RETURNS complex
  AS 'filename', 'complex_add'
  LANGUAGE C IMMUTABLE STRICT;
CREATE OPERATOR + (
  leftarg = complex,
  rightarg = complex,
  procedure = complex_add,
  commutator = +
);
```

要在查询中使用此运算符：

```
SELECT (a + b) AS c FROM test_complex;
```

兼容性

CREATE OPERATOR是一个MPP数据语言的扩展。SQL标准不提供用户定义的运算符。

另见

[CREATE FUNCTION](#) , [CREATE TYPE](#) , [ALTER OPERATOR](#) , [DROP OPERATOR](#)

Parent topic: [SQL Command Reference](#)

CREATE OPERATOR CLASS

定义一个新的运算符类。

概要

```
CREATE OPERATOR CLASS name [DEFAULT] FOR TYPE data_type
  USING index_method [ FAMILY family_name ] AS
  { OPERATOR strategy_number operator_name [ ( op_type, op_type ) ] [ FOR SEARCH | FOR
  | FUNCTION support_number funcname (argument_type [, ...] )
  | STORAGE storage_type
  } [, ... ]
```

描述

CREATE OPERATOR CLASS创建一个新的运算符类。运算符类定义如何将特定数据类型与索引一起使用。运算符类指定某些运算符将为此数据类型和此索引方法填充特定角色或策略。当为索引列选择了运算符类时，运算符类还指定了索引方法要使用的支持过程。在创建运算符类之前，必须定义运算符类使用的所有运算符和函数。用于实现运算符类的任何函数都必须定义为IMMUTABLE。

CREATE OPERATOR CLASS目前不检查运算符类定义是否包括所有由索引方法所需的运算符和函数，也不检查运算符和函数是否构成一个一致的集合。定义有效的运算符类是用户的责任。

您必须是超级用户才能创建运算符类。

参数

name

要定义的运算符类的名称（可以由模式指定）。同一模式中的两个运算符类仅在用于不同的索引方法时才可以使用相同的名称。

DEFAULT

使运算符类成为其数据类型的默认运算符类。对于特定的数据类型和索引方法，最多可以使用一个运算符类作为默认值。

data_type

此运算符类的列数据类型。

index_method

此运算符类用于的索引方法的名称。选项为btree，bitmap和gist。

family_name

要添加此运算符类的现有运算符符族的名称。如果未指定，则使用与运算符类相同的名称族（如果尚不存在，则创建它）。

strategy_number

与运算符类关联的运算符由策略编号标识，该策略编号用于标识每个运算符在其运算符类的上下文中的语义。例如，B树对键施加严格的排序，从小到大，因此对于B树来说，类似小于和大于或等于的运算符更有吸引力。可以将这些策略视为广义运算符。每个运算符类指定针对特定数据类型和索引语义解释的每个策略对应的实际运算符。每种索引方法的相应策略编号如下：

Table 1. B树和位图策略

操作	策略编号
小于	1
小于等于	2
等于	3
大于等于	4
大于	5

Table 2. GiST二维策略（R树）

操作	策略编号
严格左边	1
不延伸到右边	2
覆盖	3
不延伸到左边	4
严格右边	5
相同	6
包含	7
被包含	8
不延伸到上边	9
严格下边	10
严格上边	11
不延伸到下边	12

sort_family_name

现有btree运算符族的名称（可以由模式指定），用于描述与排序运算符关联的排序顺序。

如果未指定FOR SEARCH或FOR ORDER BY，则默认为FOR SEARCH。

operator_name

与运算符类关联的运算符的名称（可以由模式指定）。

op_type

在OPERATOR子句中，运算符的操作数数据类型，或NONE表示左一元或右一元运算符。 在正常情况下，与运算符类的数据类型相同时，可以省略操作数数据类型。

在FUNCTION子句中， 如果与函数的输入数据类型（用于B树比较函数和哈希函数）或类的数据类型（用于B树排序支持函数以及GiST，SP-GiST

和GIN运算符类中的所有函数) 不同， 则函数要支持的操作数数据类型。这些默认值是正确的， 因此， 除了B树排序支持函数旨在支持跨数据类型比较的情况外， 无需在FUNCTION子句中指定`op_type`。

support_number

索引方法为了工作需要额外的支持例程。 这些操作是索引方法在内部使用的管理例程。 与策略一样， 运算符类标识对于给定的数据类型和语义解释， 哪些特定函数应扮演这些角色中的每一个。 索引方法定义了所需的函数集， 运算符类通过将它们分配给支持函数编号来标识要使用的正确函数， 如下所示：

Table 3. B树和位图支持函数

函数	支持编号
比较两个键并返回小于零， 零或大于零的整数， 指示第一个键是否小于， 等于或大于第二个键。	1

Table 4. GiST Support Functions

函数	支持编号
consistent - 确定键是否满足查询限定符。	1
union - 计算一组键的并集。	2
compress - 计算要索引的键或值的压缩表示形式。	3
decompress - 计算压缩键的解压缩表示。	4
penalty - 计算使用给定子树的键将新键插入子树的代价。	5
picksplit - 确定一个页面的条目将被移动到新的页面， 并计算联合键结果页面。	6
equal - 比较两个键， 如果相等则返回true。	7

funcname

函数的名称（可以由模式指定）， 该函数是运算符类的索引方法支持过

程。

argument_types

函数的参数数据类型。

storage_type

实际存储在索引中的数据类型。通常，这与列数据类型相同，但是某些索引方法（当前为GiST和GIN）允许其不同。除非索引方法允许使用其他类型，否则必须省略STORAGE子句。

注解

因为索引机制在使用函数之前不会检查对函数的访问权限，所以在运算符类中包含函数或运算符就等同于对其授予公共执行权限。对于在运算符类中有用的各种函数，这通常不是问题。

运算符不应由SQL函数定义。SQL函数很可能内联到调用查询中，这将阻止优化器识别查询与索引匹配。

用于实现运算符类的任何函数都必须定义为IMMUTABLE。

在MPP数据库6.0之前，OPERATOR子句可以包含RECHECK选项。此选项不再支持。MPP数据库现在可以确定索引运算符是否在运行时是“有损的”。这允许在一个运算符可能是或者可能不是有损的情况下有效地处理。

示例

以下示例命令为数据类型_int4（int4的数组）定义了GiST索引运算符类。有关完整的示例，请参见intarray contrib模块。

```
CREATE OPERATOR CLASS gist__int_ops
  DEFAULT FOR TYPE _int4 USING gist AS
  OPERATOR 3 &&,
  OPERATOR 6 = (anyarray, anyarray),
  OPERATOR 7 @>,
  OPERATOR 8 <@,
  OPERATOR 20 @@ (_int4, query_int),
  FUNCTION 1 g_int_consistent (internal, _int4, int, oid, internal),
  FUNCTION 2 g_int_union (internal, internal),
  FUNCTION 3 g_int_compress (internal),
  FUNCTION 4 g_int_decompress (internal),
  FUNCTION 5 g_int_penalty (internal, internal, internal),
  FUNCTION 6 g_int_picksplit (internal, internal),
  FUNCTION 7 g_int_same (_int4, _int4, internal);
```

兼容性

CREATE OPERATOR CLASS是MPP数据库扩展。 SQL标准中没有CREATE OPERATOR CLASS语句。

另见

[ALTER OPERATOR CLASS](#) , [DROP OPERATOR CLASS](#) , [CREATE FUNCTION](#)

Parent topic: [SQL Command Reference](#)

CREATE OPERATOR FAMILY

定义一个新的运算符族。

概要

```
CREATE OPERATOR FAMILY name USING index_method
```

描述

CREATE OPERATOR FAMILY创建一个新的运算符族。运算符族定义了一组相关的运算符类，以及可能与这些运算符类兼容但对于任何单个索引的函数而言并非必不可少的一些其他运算符和支持函数。（对于索引必不可少的运算符和函数应该在相关的运算符类中分组，而不是"松散地"在运算符族中。通常，单数据类型的运算符绑定到运算符类，而跨数据类型的运算符在包含两种数据类型的运算符类的运算符族中可能会比较松散。）

新的运算符族最初是空的。应该通过发出后续的CREATE OPERATOR CLASS命令来添加所包含的运算符类，并可选地通过ALTER OPERATOR FAMILY命令来添加“松散”运算符及其相应的支持函数来填充该表。

如果指定了模式名称，则会在指定的模式中创建运算符族。否则，它将在当前模式中创建。同一模式中的两个运算符族只有用于不同索引方法的名称才能相同。

定义运算符族的用户将成为其所有者。当前，创建用户必须是超级用户。（之所以做出此限制，是因为错误的运算符族定义可能会使服务器混乱甚至崩溃。）

参数

- name*
要定义的运算符族的名称（可以由模式指定）。
- index_method*
该运算符族所针对的索引方法的名称。

兼容性

CREATE OPERATOR FAMILY是MPP数据库扩展。 SQL标准中没有CREATE OPERATOR FAMILY语句。

另见

[ALTER OPERATOR FAMILY](#) , [DROP OPERATOR FAMILY](#) , [CREATE FUNCTION](#) , [ALTER OPERATOR CLASS](#) , [CREATE OPERATOR CLASS](#) , [DROP OPERATOR CLASS](#)

Parent topic: [SQL Command Reference](#)

CREATE PROTOCOL

注册自定义数据访问协议，该协议可以在定义MPP数据库外部表时指定。

概要

```
CREATE [TRUSTED] PROTOCOL name (  
    [readfunc='read_call_handler'] [, writefunc='write_call_handler']  
    [, validatorfunc='validate_handler' ])
```

描述

CREATE PROTOCOL将数据访问协议名称与负责从外部数据源读取数据和将数据写入外部数据源的调用处理程序相关联。您必须是超级用户才能创建协议。

CREATE PROTOCOL命令必须指定读调用处理程序或写调用处理程序。必须在数据库中定义CREATE PROTOCOL命令中指定的调用处理程序。

可以在CREATE EXTERNAL TABLE命令中指定协议名称。

有关创建和启用自定义数据访问协议的信息，请参阅*MPP数据库管理员手册*中的“示例自定义数据访问协议”。

参数

TRUSTED

如果未指定，则只有超级用户和协议所有者才能使用协议创建外部表。如果指定，超级用户和协议所有者可以将协议的许可权限授予其他数据库角色。

name

数据访问协议的名称。协议名称区分大小写。该名称在数据库中的协议之间必须唯一。

readfunc= '*read_call_handler*'

MPP数据库调用以从外部数据源读取数据的先前注册函数的名称。该命令必须指定读调用处理程序或写调用处理程序。

`writefunc= 'write_call_handler'`

MPP数据库调用以将数据写入外部数据源时调用的先前注册函数的名称。该命令必须指定读调用处理程序或写调用处理程序。

`validatorfunc='validate_handler'`

可选的验证器函数，用于验证CREATE EXTERNAL TABLE命令中指定的URL。

注解

MPP数据库在内部处理类型为file，gpfdist和gpfdists的外部表。有关启用S3协议的信息，请参见[配置和使用S3外部表](#)。有关配置PXF扩展和使用pxf协议的信息，请参阅[MPP平台扩展框架（PXF）](#)文档。

任何实现数据访问协议的共享库都必须位于所有MPP数据库segment主机上的同一位置。例如，共享库可以位于所有主机上的操作系统环境变量LD_LIBRARY_PATH指定的位置。定义处理程序函数时，也可以指定位置。例如，当您在CREATE PROTOCOL命令中定义s3协议时，您将\$libdir/gps3ext.so指定为共享库的位置，其中\$libdir位于\$GPHOME/lib。

兼容性

CREATE PROTOCOL是MPP数据库扩展。

另见

[ALTER PROTOCOL](#), [CREATE EXTERNAL TABLE](#), [DROP PROTOCOL](#),

Parent topic: [SQL Command Reference](#)

CREATE RESOURCE GROUP

定义一个新的资源组。

概要

```
CREATE RESOURCE GROUP name WITH (group_attribute=value [, ... ])
```

其中*group_attribute*是：

```
CPU_RATE_LIMIT=integer | CPUSET=tuple  
MEMORY_LIMIT=integer  
[ CONCURRENCY=integer ]  
[ MEMORY_SHARED_QUOTA=integer ]  
[ MEMORY_SPILL_RATIO=integer ]  
[ MEMORY_AUDITOR= {vmtracker | cgroup} ]
```

描述

为MPP数据库资源管理创建一个新的资源组。 您可以创建资源组来管理角色资

源或管理MPP数据库外部组件（例如PL/Container）的资源。

启用资源组后，您创建的用于管理用户角色的资源组将标识该角色的并发事务，内存和CPU限制。您可以将此类资源组分配给一个或多个角色。

您创建的用于管理MPP数据库外部组件（例如PL/Container）的资源的资源组会在启用资源组时标识该组件的内存和CPU限制。这些资源组使用cgroup进行CPU和内存管理。将资源组分配给外部组件是特定于组件的。例如，当您配置PL/Container运行时，您将分配一个PL/Container资源组。您不能将为外部组件创建的资源组分配给角色，也不能将为角色创建的资源组分配给外部组件。

您必须具有SUPERUSER特权才能创建资源组。MPP数据库群集中允许的最大资源组数为100。

MPP数据库预定义了两个默认资源组：admin_group和default_group。这些组名以及组名none均保留。

要为资源组设置适当的限制，MPP数据库管理员必须熟悉通常在系统上执行的查询，以及执行这些查询的用户/角色以及他们可能使用的外部组件，例如PL/Containers。

创建资源组的角色后，组分配给使用[ALTER ROLE](#)或[CREATE ROLE](#)命令一个或多个角色。

创建资源组以管理外部组件的CPU和内存资源后，将外部组件配置为使用该资源组。例如，配置PL/Container运行时resource_group_id。

参数

name

资源组的名称。

CONCURRENCY *integer*

此资源组允许的最大并发事务数，包括活动和空闲事务。

CONCURRENCY值必须是[0 .. max_connections]范围内的整数。为角色定义的资源组的默认CONCURRENCY值为20。

对于为外部组件创建的资源组，必须将CONCURRENCY设置为零（0）。

Note: 您不能将admin_group的CONCURRENCY值设置为零（0）。

CPU_RATE_LIMIT *integer*

CPUSET *tuple*

需要。创建资源组时，只能指定CPU_RATE_LIMIT或CPUSET之一。

CPU_RATE_LIMIT是分配给该资源组的CPU资源的百分比。您可以为资源组指定的最小CPU百分比为1。最大为100。为MPP数据库集群中定义的所有资源组指定的CPU_RATE_LIMIT值的总和必须小于或等于100。

CPUSET标识要为此资源组保留的CPU核心。您在*tuple*中指定的CPU核心必须在系统中可用，并且不能与您为其他资源组指定的任何CPU核心重叠。

*tuple*是用逗号分隔的单个核心号或核心号间隔的列表。您必须将元组用单引号引起来，例如'1,3-4'。

Note: 仅在为MPP数据库集群启用基于资源组的资源管理之后，才能为资源组配置CPUSET。

MEMORY_LIMIT *integer*

需要。分配给该资源组的MPP数据库内存资源的总百分比。您可以为资源组指定的最小内存百分比为1。最大为100。为MPP数据库集群中定义的所有资源组指定的MEMORY_LIMIT值的总和必须小于或等于100。

MEMORY_SHARED_QUOTA *integer*

资源组中共享内存的配额。具有MEMORY_SHARED_QUOTA阈值的资源组预留了分配给资源组以在事务之间共享的内存的百分比。共享内存将按先到先得的原则分配。事务可能不使用任何，部分或全部内存。您可以为资源组指定的最小内存共享配额百分比为0。最大为100。默认的MEMORY_SHARED_QUOTA值为20。

MEMORY_SPILL_RATIO *integer*

事务中内存密集型运算的内存使用量阈值。当达到此阈值时，事务会溢出到磁盘。您可以为资源组指定的最小内存溢出率百分比是0。最大值是100。默认的MEMORY_SPILL_RATIO值为20。

MEMORY_AUDITOR {vmtracker | cgroup}

资源组的内存审核员。MPP数据库对角色资源使用虚拟内存跟踪，对外部组件使用的资源使用cgroup内存跟踪。默认的MEMORY_AUDITOR为vmtracker。使用vmtracker内存审核创建资源组时，MPP数据库在内部跟踪该资源组的内存。

当您创建指定cgroup MEMORY_AUDITOR的资源组时，MPP数据库会将该资源组使用的内存记账推迟到cgroup。对于您为外部组件（例如PL/Container）创建的资源组，CONCURRENCY必须为零（0）。您不能将为外部组件创建的资源组分配给MPP数据库角色。

注解

您不能在显式事务或子事务中提交CREATE RESOURCE GROUP命令。

使用gp_toolkit.gp_resgroup_config系统视图显示所有资源组的限制设置：

```
SELECT * FROM gp_toolkit.gp_resgroup_config;
```

示例

创建一个CPU和内存限制百分比为35的资源组：

```
CREATE RESOURCE GROUP rgroup1 WITH (CPU_RATE_LIMIT=35, MEMORY_LIMIT=35);
```

创建一个资源组，其并发事务限制为20，内存限制为15，CPU限制为25：

```
CREATE RESOURCE GROUP rgroup2 WITH (CONCURRENCY=20,  
MEMORY_LIMIT=15, CPU_RATE_LIMIT=25);
```

创建一个资源组来管理PL/Container资源，指定内存限制为10，CPU限制为10：

```
CREATE RESOURCE GROUP plc_run1 WITH (MEMORY_LIMIT=10, CPU_RATE_LIMIT=10,  
CONCURRENCY=0, MEMORY_AUDITOR=cgroup);
```

创建一个内存限制百分比为11的资源组，并为其分配CPU核心1至3：

```
CREATE RESOURCE GROUP rgroup3 WITH (CPUSET='1-3', MEMORY_LIMIT=11);
```

兼容性

CREATE RESOURCE GROUP是MPP数据扩展。在SQL标准中没有资源组或资源管理。

另见

[ALTER ROLE](#), [CREATE ROLE](#), [ALTER RESOURCE GROUP](#), [DROP RESOURCE GROUP](#)

Parent topic: [SQL Command Reference](#)

CREATE RESOURCE QUEUE

定义一个新的资源队列。

概要

```
CREATE RESOURCE QUEUE name WITH (queue_attribute=value [, ... ])
```

其中`queue_attribute`为：

```
ACTIVE_STATEMENTS=integer
[ MAX_COST=float [ COST_OVERCOMMIT={TRUE|FALSE} ] ]
[ MIN_COST=float ]
[ PRIORITY={MIN|LOW|MEDIUM|HIGH|MAX} ]
[ MEMORY_LIMIT='memory_units' ]

| MAX_COST=float [ COST_OVERCOMMIT={TRUE|FALSE} ]
[ ACTIVE_STATEMENTS=integer ]
[ MIN_COST=float ]
[ PRIORITY={MIN|LOW|MEDIUM|HIGH|MAX} ]
[ MEMORY_LIMIT='memory_units' ]
```

描述

为MPP数据库资源管理创建一个新的资源队列。资源队列必须具有ACTIVE_STATEMENTS或MAX_COST值（或者可以同时具有两者）。只有超级用户才能创建资源队列。

具有ACTIVE_STATEMENTS阈值的资源队列为分配给该队列的角色可以执行的查询数量设置了最大限制。它控制允许同时运行的活动查询的数量。ACTIVE_STATEMENTS的值应为大于0的整数。

具有MAX_COST阈值的资源队列对可以由分配给该队列的角色执行的查询的总

成本设置最大限制。成本以MPP数据库查询优化器确定的查询的估计总成本来衡量（如查询的EXPLAIN输出中所示）。因此，管理员必须熟悉通常在系统上执行的查询，以便为队列设置适当的成本阈值。成本以获取磁盘页面为单位进行衡量；1.0等于读取一个连续的磁盘页面。MAX_COST的值指定为浮点数（例如100.0），也可以指定为指数（例如 $1e + 2$ ）。如果基于成本阈值限制资源队列，则管理员可以允许COST_OVERCOMMIT=TRUE（默认值）。这意味着将允许运行超过允许的成本阈值的查询，但仅在系统空闲时才允许运行。如果指定COST_OVERCOMMIT=FALSE，则超出成本限制的查询将始终被拒绝，并且永远不允许运行。通过为MIN_COST指定一个值，管理员可以为小型查询定义费用，这些费用将免于资源排队。

Note: GPORCA和Postgres查询优化器利用不同的查询成本模型，并且可以为同一查询计算不同的成本。MPP数据库资源队列资源管理方案既不区分GPORCA和Postgres优化器之间的成本，也没有对其进行调整。它使用从优化程序返回的文字成本值来限制查询。

当基于资源队列的资源管理处于活动状态时，请对资源队列使用MEMORY_LIMIT和ACTIVE_STATEMENTS限制，而不是配置基于成本的限制。即使使用GPORCA，MPP数据库也可能会退回到对某些查询使用Postgres查询优化器，因此使用基于成本的限制可能会导致意外结果。

如果未为ACTIVE_STATEMENTS或MAX_COST定义值，则默认情况下将其设置为-1（表示无限制）。定义资源队列后，必须使用[ALTER ROLE](#)或[CREATE ROLE](#)命令将角色分配给队列。

您可以选择将PRIORITY分配给资源队列，以控制与该队列相关联的查询相对于其他资源队列使用的可用CPU资源的相对份额。如果未为PRIORITY定义值，则与队列关联的查询的默认优先级为MEDIUM。

具有可选MEMORY_LIMIT阈值的资源队列设置了对通过资源队列提交的所有查询可以在segment主机上使用的最大内存量的限制。这就决定了一个查询的所有工作进程可以在查询执行期间的segment主机上消耗的存储器的总量。MPP建议将MEMORY_LIMIT与ACTIVE_STATEMENTS结合使用，而不要与MAX_COST结合使用。在基于语句的队列上，每个查询分配的默认内存量

为： $\text{MEMORY\_LIMIT} / \text{ACTIVE\_STATEMENTS}$ 。在基于成本的队列上，每个查询分配的默认内存量为： $\text{MEMORY\_LIMIT} * (\text{query\_cost} / \text{MAX\_COST})$ 。

如果不超过`MEMORY_LIMIT`或`statement_mem`，则可以使用`statement_mem`服务器配置参数在每个查询的基础上覆盖默认内存分配。例如，为一个特定的查询分配更多的内存：

```
=> SET statement_mem='2GB';
=> SELECT * FROM my_big_table WHERE column='value' ORDER BY id;
=> RESET statement_mem;
```

您所有资源队列的`MEMORY_LIMIT`值不应超过`segment`主机的物理内存量。如果工作负载分散在多个队列中，则内存分配可能会超额预定。但是，如果超出了`gp_vmem_protect_limit`中指定的`segment`主机内存限制，则可以在执行期间取消查询。

有关`statement_mem`，`max_statement`和`gp_vmem_protect_limit`的信息，请参阅[服务器配置参数](#)。

参数

name

资源队列的名称。

`ACTIVE_STATEMENTS` *integer*

具有`ACTIVE_STATEMENTS`阈值的资源队列限制了分配给该队列的角色可以执行的查询数量。它控制允许同时运行的活动查询的数量。

`ACTIVE_STATEMENTS`的值应为大于0的整数。

`MEMORY_LIMIT` '*memory_units*'

设置此资源队列中用户提交的所有语句的总内存配额。可以用kB，MB或GB指定存储单元。资源队列的最小内存配额为10MB。没有最大值，但是查询执行时的上限受`segment`主机的物理内存限制。默认值为无限制（-1）。

MAX_COST *float*

具有MAX_COST阈值的资源队列对可以由分配给该队列的角色执行的查询的总成本设置了最大限制。成本以MPP数据库查询优化器确定的查询的估算总成本来衡量（如查询的EXPLAIN输出中所示）。因此，管理员必须熟悉通常在系统上执行的查询，以便为队列设置适当的成本阈值。成本以获取磁盘页面为单位进行衡量；1.0等于读取一个连续的磁盘页面。MAX_COST的值指定为浮点数（例如100.0），也可以指定为指数（例如1e+2）。

COST_OVERCOMMIT *boolean*

如果基于MAX_COST限制资源队列，则管理员可以允许COST_OVERCOMMIT（默认值）。这意味着将允许运行超过允许的成本阈值的查询，但仅在系统空闲时才允许运行。如果指定COST_OVERCOMMIT=FALSE，则超出成本限制的查询将始终被拒绝，并且永远不允许运行。

MIN_COST *float*

小型查询的最低查询成本限制。费用低于此限制的查询将不会排队并立即运行。成本以MPP数据库查询优化器确定的查询的估计总成本来衡量（如查询的EXPLAIN输出中所示）。因此，管理员必须熟悉通常在系统上执行的查询，以便为被认为是小的查询设置适当的成本。成本以获取磁盘页面为单位进行衡量；1.0等于读取一个连续的磁盘页面。MIN_COST 的值指定为浮点数（例如100.0），也可以指定为指数（例如1e+2）。

PRIORITY={MIN|LOW|MEDIUM|HIGH|MAX}

设置与资源队列关联的查询的优先级。发生竞争时，具有较高优先级的队列中的查询或语句将获得更多的可用CPU资源份额。在执行较高优先级的查询时，低优先级队列中的查询可能会延迟。如果未指定优先级，则与队列关联的查询的优先级为MEDIUM。

注解

使用gp_toolkit.gp_resqueue_status系统视图查看资源队列的限制设置和当前状态：

```
SELECT * from gp_toolkit.gp_resqueue_status WHERE  
rsqname='queue_name';
```

还有另一个名为pg_stat_resqueues的系统视图，该视图显示了一段时间内资源队列的统计指标。但是，要使用此视图，必须启用stats_queue_level服务器配置参数。有关使用资源队列的更多信息，请参见*MPP数据库管理员手册*中的“管理工作负载和资源”。

CREATE RESOURCE QUEUE不能在事务中运行。

此外，在执行EXPLAIN ANALYZE命令的过程中运行的SQL语句将从资源队列中排除。

示例

创建一个活动查询限制为20的资源队列：

```
CREATE RESOURCE QUEUE myqueue WITH (ACTIVE_STATEMENTS=20);
```

创建一个资源队列，其活动查询限制为20，总内存限制为2000MB（每个查询在执行时将分配100MB的segment主机内存）：

```
CREATE RESOURCE QUEUE myqueue WITH (ACTIVE_STATEMENTS=20,  
MEMORY_LIMIT='2000MB');
```

创建一个查询费用限制为3000.0的资源队列：

```
CREATE RESOURCE QUEUE myqueue WITH (MAX_COST=3000.0);
```

创建一个查询成本限制为 3^{10} （或30000000000.0）的资源队列，并且不允许过

量使用。 允许成本低于500的小型查询立即运行：

```
CREATE RESOURCE QUEUE myqueue WITH (MAX_COST=3e+10,  
  COST_OVERCOMMIT=FALSE, MIN_COST=500.0);
```

创建同时具有活动查询限制和查询成本限制的资源队列：

```
CREATE RESOURCE QUEUE myqueue WITH (ACTIVE_STATEMENTS=30,  
  MAX_COST=5000.00);
```

创建一个活动查询限制为5且最大优先级设置的资源队列：

```
CREATE RESOURCE QUEUE myqueue WITH (ACTIVE_STATEMENTS=5,  
  PRIORITY=MAX);
```

兼容性

CREATE RESOURCE QUEUE是MPP数据库扩展。 SQL标准中没有提供资源队列或资源管理的规定。

另见

[ALTER ROLE](#), [CREATE ROLE](#), [ALTER RESOURCE QUEUE](#), [DROP RESOURCE QUEUE](#)

Parent topic: [SQL Command Reference](#)

CREATE ROLE

定义一个新的数据库角色（用户或组）。

概要

```
CREATE ROLE name [[WITH] option [ ... ]]
```

其中`option`可以是：

```

    SUPERUSER | NOSUPERUSER
| CREATEDB | NOCREATEDB
| CREATEROLE | NOCREATEROLE
| CREATEUSER | NOCREATEUSER
| CREATEEXTTABLE | NOCREATEEXTTABLE
| [ ( attribute='value'[ , ... ] ) ]
    where attributes and value are:
        type='readable'|'writable'
        protocol='gpfdist'|'http'
| INHERIT | NOINHERIT
| LOGIN | NOLOGIN
| REPLICATION | NOREPLICATION
| CONNECTION LIMIT conlimit
| [ ENCRYPTED | UNENCRYPTED ] PASSWORD 'password'
| VALID UNTIL 'timestamp'
| IN ROLE rolename [ , ... ]
| ROLE rolename [ , ... ]
| ADMIN rolename [ , ... ]
| USER rolename [ , ... ]
| SYSID uid [ , ... ]
| RESOURCE QUEUE queue_name
| RESOURCE GROUP group_name
| [ DENY deny_point ]
| [ DENY BETWEEN deny_point AND deny_point]

```

描述

CREATE ROLE在MPP数据库系统中添加了新角色。角色是可以拥有数据库对象并具有数据库特权的实体。根据角色的使用方式，可以将角色视为用户，组或两者。您必须具有CREATEROLE特权或是数据库超级用户才能使用此命令。

请注意，角色是在系统级别定义的，并且对于MPP数据库系统中的所有数据库均有效。

参数

name

新角色的名称。

SUPERUSER

NOSUPERUSER

如果指定了**SUPERUSER**，则定义的角色将是超级用户，该超级用户可以覆盖数据库中的所有访问限制。超级用户状态很危险，应仅在真正需要时使用。您必须自己是超级用户才能创建新的超级用户。默认值为**NOSUPERUSER**。

CREATEDB

NOCREATEDB

如果指定了**CREATEDB**，将允许所定义的角色创建新数据库。

NOCREATEDB（默认值）将使角色无法创建数据库。如果`restrict_user_privileges`设置为true，则不允许普通用户拥有创建数据库的能力。

CREATEROLE

NOCREATEROLE

如果指定了**CREATEROLE**，则允许定义的角色创建新角色，更改其他角色和删除其他角色。**NOCREATEROLE**（默认值）将拒绝角色创建新角色或修改其他角色。如果`restrict_user_privileges`设置为true，则不允许普通用户拥有创建角色的能力。

CREATEUSER

NOCREATEUSER

这些子句已经过时，但仍然被**SUPERUSER**和**NOSUPERUSER**的拼写接受。请注意，它们不等同于**CREATEROLE**和**NOCREATEROLE**子句。如果`restrict_user_privileges`设置为true，则不允许普通用户拥有创建用户的能力。

CREATEEXTTABLE

NOCREATEEXTTABLE

如果指定了**CREATEEXTTABLE**，则允许定义的角色创建外部表。如果未指定，则默认type为readable，默认protocol为gpfdist。有效类型为gpfdist，gpfdists，http和https。**NOCREATEEXTTABLE**（默认类型）

拒绝该角色创建外部表。 请注意，使用file或execute协议的外部表只能由超级用户创建。

使用GRANT...ON PROTOCOL命令允许用户创建和使用具有自定义协议类型的外部表，包括MPP数据库附带的s3和pxf协议。

INHERIT

NOINHERIT

如果指定了该属性，则INHERIT（默认设置）允许该角色使用为其直接或间接所属的所有角色授予的任何数据库特权。 使用NOINHERIT时，另一个角色的成员资格仅授予SET ROLE权限给该另一个角色。

LOGIN

NOLOGIN

如果指定，则LOGIN允许角色登录数据库。 可以将具有LOGIN属性的角色视为用户。 具有NOLOGIN的角色对于管理数据库特权很有用，并且可以视为组。 如果未指定，则NOLOGIN为默认值，除非CREATE ROLE通过其替代拼写[CREATE USER](#)被调用时。

REPLICATION

NOREPLICATION

这些子句确定是允许角色启动流复制还是使系统进入和退出备份模式。 具有REPLICATION属性的角色是具有很高特权的角色，并且仅应在实际用于复制的角色上使用。 如果未指定，则NOREPLICATION是默认值。

CONNECTION LIMIT *conlimit*

此角色可以建立的并发连接的最大数量。 默认值-1表示没有限制。

PASSWORD *password*

使用LOGIN属性设置角色的用户密码。 如果您不打算使用密码身份验证，则可以忽略此选项。 如果未指定密码，则密码将设置为null，并且该用户的密码身份验证将始终失败。 空密码可以有选择地显式写为PASSWORD NULL。

ENCRYPTED

UNENCRYPTED

这些关键字控制密码是否以加密方式存储在系统catalog中。（如果未指定，则默认行为由配置参数*password_encryption*决定。） 如果显示的密码字符串已经采用MD5加密格式，则将按原样存储加密，而不管是否指定

了ENCRYPTED或UNENCRYPTED（因为系统无法解密指定的加密密码字符串）。这允许在转储/还原期间重新加载加密的密码。

VALID UNTIL *'timestamp'*

VALID UNTIL子句设置日期和时间，之后该角色的密码将不再有效。如果省略此子句，密码将永不过期。

IN ROLE *rolename*

将新角色添加为命名角色的成员。请注意，没有任何选项可以将新角色添加为管理员。使用单独的GRANT命令来执行此操作。

ROLE *rolename*

将命名角色添加为该角色的成员，从而使该新角色成为一个组。

ADMIN *rolename*

ADMIN子句类似于ROLE，但是被提及的角色被使用WITH ADMIN OPTION加入到新角色中，从而赋予他们将这个角色的成员资格授予其他人的权利。

RESOURCE GROUP *group_name*

要分配给新角色的资源组的名称。该角色将受限于资源组配置的并发事务，内存和CPU限制。您可以将一个资源组分配给一个或多个角色。如果未为新角色指定资源组，则会自动为该角色分配角色的默认资源组，为SUPERUSER角色分配admin_group，为非管理员角色分配default_group。

您可以将admin_group资源组分配给具有SUPERUSER属性的任何角色。

您可以将default_group资源组分配给任何角色。

您不能将为外部组件创建的资源组分配给角色。

RESOURCE QUEUE *queue_name*

新用户级别角色将分配到的资源队列的名称。只能将具有LOGIN特权的角色分配给资源队列。特殊关键字NONE表示将角色分配给默认资源队列。一个角色只能属于一个资源队列。

具有SUPERUSER属性的角色不受资源队列限制。对于超级用户角色，无论分配的资源队列施加什么限制，查询总是立即运行。

DENY *deny_point*

DENY BETWEEN *deny_point* AND *deny_point*

DENY和DENY BETWEEN关键字设置在登录时强制执行的基于时间的约束。DENY设置拒绝访问的日期或日期和时间。DENY BETWEEN设置

一个拒绝访问的时间间隔。两者都使用具有以下格式的参数`deny_point`：

```
DAY day [ TIME 'time' ]
```

`deny_point`参数的两个部分使用以下格式：

对于day:

```
{ 'Sunday' | 'Monday' | 'Tuesday' | 'Wednesday' | 'Thursday' | 'Friday' |  
'Saturday' | 0-6 }
```

对于time:

```
{ 00-23 : 00-59 | 01-12 : 00-59 { AM | PM } }
```

DENY BETWEEN子句使用两个`deny_point`参数：

```
DENY BETWEEN deny_point AND deny_point
```

有关基于时间的约束的更多信息和示例，请参阅*MPP数据库管理员手册*中的“管理角色和特权”。

注解

添加和删除角色成员（管理组）的首选方法是使用[GRANT](#)和[REVOKE](#)。

VALID UNTIL子句仅为密码而不是角色定义过期时间。使用非基于密码的身份验证方法登录时，不会强制使用到期时间。

INHERIT属性控制可授予特权（数据库对象和角色成员的访问特权）的继承。它不适用于由CREATE ROLE和ALTER ROLE设置的特殊角色属性。例如，即

使设置了INHERIT，具有CREATEDB特权的角色成员也不会立即授予创建数据库的能力。这些特权/属性永远不会被继承：

SUPERUSER，CREATEDB，CREATEROLE，
CREATEEXTTABLE，LOGIN，RESOURCE GROUP和RESOURCE
QUEUE。必须在每个用户级角色上设置属性。

由于向后兼容，INHERIT属性是默认属性。在以前的MPP数据库版本中，用户始终可以访问其所属组的所有特权。但是，NOINHERIT提供与SQL标准中指定的语义更接近的匹配。

使用CREATEROLE特权时要小心。对于CREATEROLE-role的特权，没有继承的概念。这意味着，即使一个角色没有特定的特权，但被允许创建其他角色，它也可以轻松地创建另一个角色，而该角色的特权不同于其自己的角色（创建具有超级用户特权的角色除外）。例如，如果角色具有CREATEROLE特权，但没有CREATEDB特权，则它可以使用CREATEDB特权创建新角色。因此，将具有CREATEROLE特权的角色视为几乎超级用户角色。

超级用户绝不执行CONNECTION LIMIT选项。

使用此命令指定未加密的密码时必须小心。密码将以明文形式传输到服务器，并且也可能会记录在客户端的命令历史记录或服务器日志中。但是，客户端程序createuser传输加密的密码。另外，psql包含命令\password，可用于稍后安全地更改密码。

示例

创建一个可以登录但不提供密码的角色：

```
CREATE ROLE jonathan LOGIN;
```

创建一个属于资源队列的角色：

```
CREATE ROLE jonathan LOGIN RESOURCE QUEUE poweruser;
```

使用有效期至2016年底的密码创建角色（CREATE USER与CREATE ROLE相同，只不过它暗含了LOGIN）：

```
CREATE USER joelle WITH PASSWORD 'jw8s0F4' VALID UNTIL '2017-01-01';
```

创建一个可以创建数据库并管理其他角色的角色：

```
CREATE ROLE admin WITH CREATEDB CREATEROLE;
```

创建一个角色，该角色在星期日不允许登录访问：

```
CREATE ROLE user3 DENY DAY 'Sunday';
```

创建一个可以创建类型为'gpfdist'的可读可写外部表的角色：

```
CREATE ROLE jan WITH CREATEEXTTABLE(type='readable', protocol='gpfdist')  
CREATEEXTTABLE(type='writable', protocol='gpfdist');
```

创建一个角色，分配一个资源组：

```
CREATE ROLE bill RESOURCE GROUP rg_light;
```

兼容性

SQL标准定义了用户和角色的概念，但是将它们视为不同的概念，并将所有定义

用户的命令留给数据库实现指定。在MPP数据库中，用户和角色被统一为单一类型的对象。因此，角色具有比标准中更多的可选属性。

CREATE ROLE在SQL标准中，但是该标准仅需要以下语法：

```
CREATE ROLE name [WITH ADMIN rolename]
```

MPP数据库扩展是允许多个初始管理员，以及CREATE ROLE的所有其他选项。

通过为用户提供NOINHERIT属性，而为角色赋予INHERIT属性，可以最接近地逼近SQL标准指定的行为。

另见

[SET ROLE](#), [ALTER ROLE](#), [DROP ROLE](#), [GRANT](#), [REVOKE](#), [CREATE RESOURCE QUEUE](#) [CREATE RESOURCE GROUP](#)

Parent topic: [SQL Command Reference](#)

CREATE RULE

定义新的重写规则。

概要

```
CREATE [OR REPLACE] RULE name AS ON event
  TO table_name [WHERE condition]
  DO [ALSO | INSTEAD] { NOTHING | command | (command; command
  ...)
```

描述

CREATE RULE定义了应用于指定表或视图的新规则。CREATE OR REPLACE RULE将创建一个新规则，或为同一表替换一个具有相同名称的现有规则。

MPP数据库规则系统允许您定义对数据库表中的插入，更新或删除执行的另一种操作。当执行给定表上的给定命令时，规则将导致执行附加或替代命令。INSTEAD规则可以用另一个命令替换给定命令，或导致根本不执行命令。规则也可以用于实现SQL视图。重要的是要认识到规则实际上是命令转换机制或命令宏。转换发生在命令执行开始之前。它不会像触发器那样针对每个物理行独立运行。

ON SELECT规则必须是无条件的INSTEAD规则，并且必须具有由单个SELECT命令组成的操作。因此，ON SELECT规则有效地将表变成视图，其可见内容是规则的SELECT命令返回的行，而不是表中存储的内容（如果有的话）。与创建真实表并为其定义ON SELECT规则相比，写CREATE VIEW命令被认为是更好的样式。

您可以通过定义ON INSERT，ON UPDATE和ON DELETE规则将视图上的更新操作替换为其他表上的适当更新，来创建可更新视图的错觉。如果要支持INSERT RETURNING等，请确保在每个规则中放入合适的RETURNING子句。

如果尝试使用条件规则进行视图更新，则有一个陷阱：要允许在视图上执行的

每个操作都必须有一个无条件的INSTEAD规则。如果规则是有条件的，或者不是INSTEAD，则系统仍将拒绝执行更新操作的尝试，因为它认为在某些情况下可能最终尝试对视图的虚拟表执行操作。如果要处理条件规则中的所有有用情况，请添加无条件的DO INSTEAD NOTHING规则，以确保系统理解它将永远不会被调用它来更新虚拟表。然后将条件规则设为non-INSTEAD；在应用它们的情况下，它们会添加到默认的INSTEAD NOTHING操作中。（但是，该方法当前不适用于支持RETURNING查询。）

Note:

这是很简单的自动更新的视图（参见[CREATE VIEW](#)）以可更新并不需要用户创建的规则。尽管您仍然可以创建显式规则，但是自动更新转换通常会胜过显式规则。

参数

name

要创建的规则的名称。此名称必须与同一表的任何其他规则的名称不同。同一表和同一事件类型上的多个规则以字母名称顺序应用。

event

该事件是SELECT，INSERT，UPDATE或DELETE之一。

table_name

规则所适用的表或视图的名称（可以由模式指定）。

condition

任何SQL条件表达式（返回布尔值）。条件表达式可能不引用除NEW和OLD以外的任何表，并且可能不包含聚合函数。NEW和OLD引用参考表中的值。NEW在ON INSERT和ON UPDATE规则中有效，以引用要插入或更新的新行。OLD在ON UPDATE和ON DELETE规则中有效，以引用要更新或删除的现有行。

INSTEAD

INSTEAD NOTHING指示应执行的命令而不是原始命令。

ALSO

ALSO指示除了原始命令外，还应该执行命令。如果未指定ALSO或INSTEAD，则默认为ALSO。

command

组成规则操作的一个或多个命令。有效的命令是SELECT，INSERT，UPDATE或DELETE。特殊表名NEW和OLD可以用来引用引用表中的值。NEW在ON INSERT和ON UPDATE规则中有效，以引用要插入或更新的新行。OLD在ON UPDATE和ON DELETE规则中有效，以引用要更新或删除的现有行。

注解

您必须是表的所有者才能创建或更改该表的规则。

注意避免循环规则非常重要。递归规则在规则创建时未得到验证，但是将在执行时报告错误。

示例

创建一个规则，当用户尝试向分区的父表rank中插入行时，将行插入到子表b2001中：

```
CREATE RULE b2001 AS ON INSERT TO rank WHERE gender='M' and  
year='2001' DO INSTEAD INSERT INTO b2001 VALUES (NEW.id,  
NEW.rank, NEW.year, NEW.gender, NEW.count);
```

兼容性

CREATE RULE是MPP数据库语言的扩展，整个查询重写系统也是如此。

另见

[ALTER RULE DROP RULE](#) , [CREATE TABLE](#) , [CREATE VIEW](#)

Parent topic: [SQL Command Reference](#)

CREATE SCHEMA

定义一个新的模式。

概要

```
CREATE SCHEMA schema_name [AUTHORIZATION username]  
    [schema_element [ ... ]]
```

```
CREATE SCHEMA AUTHORIZATION rolename [schema_element [ ... ]]
```

```
CREATE SCHEMA IF NOT EXISTS schema_name [ AUTHORIZATION user_name ]
```

```
CREATE SCHEMA IF NOT EXISTS AUTHORIZATION user_name
```

描述

CREATE SCHEMA将新模式加入当前数据库。 模式名称必须不同于当前数据库中任何现有模式的名称。

模式本质上是一个名称空间：它包含命名对象（表，数据类型，函数和运算符），其名称可能与其他模式中存在的其他对象的名称重复。通过使用模式名称作为前缀来限定其名称，或设置包含所需模式的搜索路径，可以访问命名对象。使用CREATE命令指定了不合格的对象名称，将在当前模式（搜索路径前面的对象，可以使用current_schema函数确定）中创建该对象。

（可选）CREATE SCHEMA可以包含子命令以在新模式中创建对象。子命令与创建模式后发出的单独命令基本相同，不同之处在于，如果使用AUTHORIZATION子句，则所有创建的对象将归该角色所有。

参数

schema_name

要创建的模式的名称。如果省略，则将用户名用作模式名称。该名称不能以pg_开头，因为此类名称是为系统catalog模式保留的。

user_name

拥有模式的角色的名称。如果省略，则默认为执行命令的角色。只有超级用户可以创建除自己以外的角色所有的模式。

schema_element

定义要在模式内创建的对象SQL语句。当前，在CREATE SCHEMA中，只有CREATE TABLE，CREATE VIEW，CREATE INDEX，CREATE SEQUENCE，CREATE TRIGGER和GRANT被接受为子句。创建模式后，可以在单独的命令中创建其他类型的对象。

Note: MPP数据库不支持触发器。

IF NOT EXISTS

如果已经存在相同名称的模式，则不执行任何操作（发出通知除外）。使用此选项时，不能包含*schema_element*子命令。

注解

要创建模式，调用用户必须对当前数据库具有CREATE特权，或者必须是超级用户。

示例

创建一个模式：

```
CREATE SCHEMA myschema;
```

为角色joe创建一个模式（该模式也将命名为joe）：

```
CREATE SCHEMA AUTHORIZATION joe;
```

创建一个名为test的模式，该模式将由用户joe拥有，除非已经有一个名为test的模式。（joe是否拥有预先存在的模式是没有影响的。）

```
CREATE SCHEMA IF NOT EXISTS test AUTHORIZATION joe;
```

兼容性

SQL标准允许在CREATE SCHEMA中使用DEFAULT CHARACTER SET子句，以及比MPP数据库目前接受的子命令类型更多的子命令类型。

SQL标准指定CREATE SCHEMA中的子命令可以按任何顺序出现。当前的MPP数据库实现并未处理子命令中所有前向引用的情况；有时可能需要对子命令重新排序，以避免前向引用。

根据SQL标准，模式的所有者始终拥有其中的所有对象。MPP数据库允许模式包含模式所有者以外的用户拥有的对象。仅当模式所有者将模式的CREATE特

权授予其他人，或者超级用户选择在其中创建对象时，才会发生这种情况。

IF NOT EXISTS选项是MPP数据库扩展。

另见

[ALTER SCHEMA](#) , [DROP SCHEMA](#)

Parent topic: [SQL Command Reference](#)

CREATE SEQUENCE

定义一个新的序列生成器。

概要

```
CREATE [TEMPORARY | TEMP] SEQUENCE name
  [INCREMENT [BY] value]
  [MINVALUE minvalue | NO MINVALUE]
  [MAXVALUE maxvalue | NO MAXVALUE]
  [START [ WITH ] start]
  [CACHE cache]
  [[NO] CYCLE]
  [OWNED BY { table.column | NONE }]
```

描述

CREATE SEQUENCE创建一个新的序列生成器。这涉及创建和初始化新的特殊单行表。生成器将由发出命令的用户所有。

如果指定了模式名称，则会在指定的模式中创建序列。否则，它将在当前模式中创建。临时序列存在于特殊模式中，因此在创建临时序列时可能不会给出模式名称。序列名称必须与同一模式中任何其他序列，表，索引，视图或外部表的名称不同。

创建序列后，可以使用nextval()函数对序列进行操作。例如，要将行插入到获取序列的下一个值的表中：

```
INSERT INTO distributors VALUES (nextval('myserial'), 'acme');
```

您还可以使用函数setval()对序列进行操作，但仅用于不对分布式数据进行操作的查询。例如，允许以下查询，因为它会重置master上序列生成器进程的序列计数器值：

```
SELECT setval('myserial', 201);
```

但是以下查询在MPP数据库中将被拒绝，因为它对分布式数据进行操作：

```
INSERT INTO product VALUES (setval('myserial', 201), 'gizmo');
```

在常规（非分布式）数据库中，对序列进行操作的函数会转到本地序列表以根据需要获取值。但是，请记住，在MPP数据库中，每个segment都是其自己不同的数据库进程。因此，segment需要一个真实单点来获取序列值，以便所有segment正确递增，并且序列以正确的顺序前进。序列服务器进程在master上运行，并且是MPP分布式数据库中序列的真实点。segment在运行时从master获取序列值。

由于这种分布式序列设计，因此在MPP数据库中对序列操作的函数存在一些限制：

- `lastval()`和`curval()`函数不被支持。
- `setval()`仅可用于在master上设置序列生成器的值，而不能在子查询中用于更新分布式表数据上的记录。
- 根据查询的不同，`nextval()`有时会从master获取一个值块以供segment使用。因此，如果在segment级别不需要所有块，有时可能会跳过序列中的值。请注意，常规PostgreSQL数据库也可以执行此操作，因此这并不是MPP数据库所独有的。

尽管您无法直接更新序列，但可以使用类似以下的查询：

```
SELECT * FROM sequence_name;
```

检查序列的参数和当前状态。特别是，序列的`last_value`字段显示了任何会话分配的最后一个值。

参数

TEMPORARY | TEMP

如果指定，则仅为此会话创建序列对象，并在会话退出时自动将其删除。具有相同名称的现有永久序列在临时序列存在时不可见（在此会话中），除非使用模式限定名称引用它们。

name

要创建的序列的名称（可以由模式指定）。

increment

指定将哪个值添加到当前序列值以创建新值。正值将形成一个升序，负值将形成一个降序。默认值为1。

minvalue

NO MINVALUE

确定序列可以生成的最小值。如果未提供此子句或指定了NO MINVALUE，则将使用默认值。升序和降序的默认值分别为1和-263-1。

maxvalue

NO MAXVALUE

确定序列的最大值。如果未提供此子句或指定了NO MAXVALUE，则将使用默认值。升序和降序的默认值分别为263-1和-1。

start

允许序列从任何地方开始。默认的起始值为升序的最小值和降序的最大值。

cache

指定要预分配多少序号并将其存储在内存中，以加快访问速度。最小（默认）值为1（无高速缓存）。

CYCLE

NO CYCLE

当达到最大值（递增）或最小值（递减）时，允许序列回绕。如果达到限制，则生成的下一个数字将是最小值（升序）或最大值（降序）。如果指定了NO CYCLE，则在序列达到最大值之后，对nextval()的任何调用都将返回错误。如果未指定，则默认为NO CYCLE。

OWNED BY *table.column*

OWNED BY NONE

使序列与特定的表列相关联，这样，如果该列（或其整个表）被删除，该序列也将被自动删除。指定的表必须具有相同的所有者，并且与序列具有相同的模式。OWNED BY NONE（默认值）指定不存在这种关联。

注解

序列基于bigint算术，因此范围不能超过八字节整数的范围（-9223372036854775808至9223372036854775807）。

尽管保证多个会话分配不同的序列值，但是当考虑所有会话时，这些值可能会不按顺序生成。例如，会话A可能保留值1..10并返回nextval=1，然后会话B可能

保留值11..20并在会话A生成nextval=2之前返回nextval=11。因此，您仅应假设nextval()值都是不同的，而不是纯粹按顺序生成它们。同样，*last_value*将反映任何会话保留的最新值，无论nextval()是否已返回该值。

示例

创建一个名为*myseq*的序列：

```
CREATE SEQUENCE myseq START 101;
```

在表中插入一行，以获取名为*idseq*的序列的下一个值：

```
INSERT INTO distributors VALUES (nextval('idseq'), 'acme');
```

在master上重置序列计数器值：

```
SELECT setval('myseq', 201);
```

在MPP数据库中非法使用setval()（在分布式数据上设置序列值）：

```
INSERT INTO product VALUES (setval('myseq', 201), 'gizmo');
```

兼容性

CREATE SEQUENCE符合SQL标准，但以下情况除外：

- 不支持SQL标准中指定的AS*data_type*表达式。
-

- 使用nextval()函数代替SQL标准中指定的NEXT VALUE FOR表达式来获取下一个值。
- OWNED BY子句是MPP数据库扩展。

另见

[ALTER SEQUENCE](#) , [DROP SEQUENCE](#)

Parent topic: [SQL Command Reference](#)

CREATE SERVER

定义一个新的外部服务器。

概要

```
CREATE SERVER server_name [ TYPE 'server_type' ] [ VERSION 'server_version' ]  
  FOREIGN DATA WRAPPER fdw_name  
  [ OPTIONS ( [ mpp_execute { 'master' | 'any' | 'all segments' } [, ] ] option 'value' [, ... ] ) ]
```

描述

CREATE SERVER定义一个新的外部服务器。定义服务器的用户将成为其所有者。

外部服务器通常封装外部数据包装器用来访问外部数据源的连接信息。可以通过用户映射指定其他特定于用户的连接信息。

创建服务器需要对指定的外部数据包装器具有USAGE特权。

参数

server_name

要创建的外部服务器的名称。服务器名称在数据库中必须唯一。

server_type

可选服务器类型，可能对外部数据包装器有用。

server_version

可选服务器版本，可能对外部数据包装器有用。

fdw_name

管理服务器的外部数据包装器的名称。

OPTIONS (*option* '*value*' [, ...])

新的外部服务器的选项。这些选项通常定义服务器的连接详细信息，但是实际的名称和值取决于服务器的外部数据包装器。

mpp_execute { 'master' | 'any' | 'all segments' }

一个选项，用于标识外部数据包装器从其请求数据的主机：

- master（默认设置）- 从master主机请求数据。
-

- any — 向master主机或任一segment请求数据，具体取决于哪条路径的成本更低。
- all segments — 从所有segment中请求数据。要支持此选项值，外部数据包装器必须具有将segment与数据匹配的策略。

可以在多个命令中指定mpp_execute选项：CREATE FOREIGN TABLE，CREATE SERVER和CREATE FOREIGN DATA WRAPPER。外部表设置优先于外部服务器设置，然后是外部数据包装器设置。

注解

使用dblink模块（请参阅[dblink](#)）时，可以将外部服务器名称用作dblink_connect()函数的参数来提供连接参数。您必须在外部服务器上具有USAGE特权才能以这种方式使用它。

示例

创建一个名为myserver的外部服务器，该服务器使用名为pgsql的外部数据包装器并包含连接选项：

```
CREATE SERVER myserver FOREIGN DATA WRAPPER pgsql
  OPTIONS (host 'foo', dbname 'foodb', port '5432');
```

兼容性

CREATE SERVER符合ISO/IEC 9075-9（SQL/MED）。

另见

[ALTER SERVER](#), [DROP SERVER](#), [CREATE FOREIGN DATA WRAPPER](#),
[CREATE USER MAPPING](#)

Parent topic: [SQL Command Reference](#)

CREATE TABLE

定义一个新表。

Note: 引用完整性语法（外键约束）被接受但未强制执行。

概要

```

CREATE [ [GLOBAL | LOCAL] {TEMPORARY | TEMP } | UNLOGGED] TABLE [IF NOT EXISTS]
    table_name (
        [ { column_name data_type [ COLLATE collation ] [column_constraint [ ... ] ]
    [ ENCODING ( storage_directive [, ...] ) ]
        | table_constraint
        | LIKE source_table [ like_option ... ] }
        | [ column_reference_storage_directive [, ...]
        [, ... ]
    ) )
[ INHERITS ( parent_table [, ... ] ) ]
[ WITH ( storage_parameter [=value] [, ... ] )
    | WITH OIDS | WITHOUT OIDS ]
[ ON COMMIT { PRESERVE ROWS | DELETE ROWS | DROP } ]
[ TABLESPACE tablespace_name ]
[ DISTRIBUTED BY (column [opclass], [ ... ] )
    | DISTRIBUTED RANDOMLY | DISTRIBUTED REPLICATED ]
[ PARTITION BY partition_type (column)
    [ SUBPARTITION BY partition_type (column) ]
    [ SUBPARTITION TEMPLATE ( template_spec ) ]
    [...]
    ( partition_spec )
    | [ SUBPARTITION BY partition_type (column) ]
    [...]
    ( partition_spec
    [ ( subpartition_spec
        [(...)]
    ) ]
    ) ]
)]

```

```

CREATE [ [GLOBAL | LOCAL] {TEMPORARY | TEMP} | UNLOGGED ] TABLE [IF NOT EXISTS]
    table_name
    OF type_name [ (
        { column_name WITH OPTIONS [ column_constraint [ ... ] ]
        | table_constraint }
        [, ... ]
    ) ]
[ WITH ( storage_parameter [=value] [, ... ] ) | WITH OIDS | WITHOUT OIDS ]
[ ON COMMIT { PRESERVE ROWS | DELETE ROWS | DROP } ]
[ TABLESPACE tablespace_name ]

```

其中 *column_constraint* 是：

```
[ CONSTRAINT constraint_name
{ NOT NULL
  | NULL
  | CHECK ( expression ) [ NO INHERIT ]
  | DEFAULT default_expr
  | UNIQUE index_parameters
  | PRIMARY KEY index_parameters
  | REFERENCES reftable [ ( refcolumn ) ]
    [ MATCH FULL | MATCH PARTIAL | MATCH SIMPLE ]
    [ ON DELETE key_action ] [ ON UPDATE key_action ] }
[ DEFERRABLE | NOT DEFERRABLE ] [ INITIALLY DEFERRED | INITIALLY IMMEDIATE ]
```

table_constraint 是：

```
[ CONSTRAINT constraint_name ]
{ CHECK ( expression ) [ NO INHERIT ]
  | 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 key_action ] [ ON UPDATE key_action ] }
[ DEFERRABLE | NOT DEFERRABLE ] [ INITIALLY DEFERRED | INITIALLY IMMEDIATE ]
```

like_option 是：

```
{INCLUDING|EXCLUDING} {DEFAULTS|CONSTRAINTS|INDEXES|STORAGE|COMMENTS|A
```

UNIQUE和PRIMARY KEY中的 *index_parameters* 约束为：

```
[ WITH ( storage_parameter [=value] [, ... ] ) ]
[ USING INDEX TABLESPACE tablespace_name ]
```

列的*storage_directive*是：

```
compress_type={ZLIB|ZSTD|QUICKLZ|RLE_TYPE|NONE}  
[compress_level={0-9}]  
[blocksize={8192-2097152}]
```

表的*storage_parameter*是：

```
append_optimized={TRUE|FALSE}  
blocksize={8192-2097152}  
orientation={COLUMN|ROW}  
checksum={TRUE|FALSE}  
compress_type={ZLIB|ZSTD|QUICKLZ|RLE_TYPE|NONE}  
compress_level={0-9}  
fillfactor={10-100}  
oids[=TRUE|FALSE]
```

*key_action*是：

```
ON DELETE  
| ON UPDATE  
| NO ACTION  
| RESTRICT  
| CASCADE  
| SET NULL  
| SET DEFAULT
```

*partition_type*是：

```
LIST | RANGE
```

*partition_specification*是：

```
partition_element [, ...]
```

*partition_element*是：

```

DEFAULT PARTITION
    name
| [PARTITION name] VALUES (list_value [,...])
| [PARTITION name]
    START ([datatype] 'start_value') [INCLUSIVE | EXCLUSIVE]
    [ END ([datatype] 'end_value') [INCLUSIVE | EXCLUSIVE] ]
    [ EVERY ([datatype] [number /INTERVAL] 'interval_value') ]
| [PARTITION name]
    END ([datatype] 'end_value') [INCLUSIVE | EXCLUSIVE]
    [ EVERY ([datatype] [number /INTERVAL] 'interval_value') ]
[ WITH ( partition_storage_parameter=value [, ... ] ) ]
[ column_reference_storage_directive [, ...] ]
[ TABLESPACE tablespace ]

```

其中*subpartition_spec*或*template_spec*是：

```
subpartition_element [, ...]
```

*subpartition_element*是：

```

DEFAULT SUBPARTITION name
| [SUBPARTITION name] VALUES (list_value [,...])
| [SUBPARTITION name]
    START ([datatype] 'start_value') [INCLUSIVE | EXCLUSIVE]
    [ END ([datatype] 'end_value') [INCLUSIVE | EXCLUSIVE] ]
    [ EVERY ([datatype] [number /INTERVAL] 'interval_value') ]
| [SUBPARTITION name]
    END ([datatype] 'end_value') [INCLUSIVE | EXCLUSIVE]
    [ EVERY ([datatype] [number /INTERVAL] 'interval_value') ]
[ WITH ( partition_storage_parameter=value [, ... ] ) ]
[ column_reference_storage_directive [, ...] ]
[ TABLESPACE tablespace ]

```

其中分区的*storage_parameter*是：

```
appendoptimized={TRUE|FALSE}  
blocksize={8192-2097152}  
orientation={COLUMN|ROW}  
checksum={TRUE|FALSE}  
compressype={ZLIB|ZSTD|QUICKLZ|RLE_TYPE|NONE}  
compresslevel={1-19}  
fillfactor={10-100}  
oids[=TRUE|FALSE]
```

描述

CREATE TABLE在当前数据库中创建一个最初为空的表。 执行命令的用户拥有该表。

为了能够创建表，您必须分别对所有列类型或OF子句中的类型具有USAGE特权。

如果指定模式名称，MPP将在指定的模式中创建表。 否则，MPP将在当前模式中创建表。 临时表存在于特殊的模式中，因此在创建临时表时不能指定模式名称。 表名称必须与同一模式中的任何其他表，外部表，序列，索引，视图或外部表的名称不同。

CREATE TABLE还会自动创建一个数据类型，该数据类型表示与表的一行相对应的复合类型。 因此，表不能与同一模式中的任何现有数据类型具有相同的名称。

可选的约束子句指定新行或更新行必须满足的条件才能成功执行插入或更新操作。 约束是一个SQL对象，可以通过多种方式帮助定义表中的有效值集。 约束适用于表，而不适用于分区。 您不能将约束添加到分区或子分区。

引用完整性约束（外键）被接受但不强制执行。 该信息保留在系统catalog中，否则将被忽略。

有两种定义约束的方法：表约束和列约束。列约束被定义为列定义的一部分。表约束定义不与特定列绑定，并且可以包含多个列。每个列约束也可以写为表约束。当约束仅影响一列时，使用列约束只是一种符号上的方便。

创建表时，还有一个附加子句来声明MPP数据库分发策略。如果未提供DISTRIBUTED BY，DISTRIBUTED RANDOMLY或DISTRIBUTED REPLICATED子句，则MPP数据库将通过使用PRIMARY KEY（如果表具有一个）或表的第一列作为分发键，向该表分配哈希分发策略。几何或用户定义数据类型的列不符合MPP分布键列的要求。如果表中没有符合条件的数据类型列，则将根据轮询或随机分布来分配行。为了确保数据在MPP数据库系统中的均匀分配，您希望选择一个对于每个记录都是唯一的分配键，或者如果不可能，则选择DISTRIBUTED RANDOMLY。

如果提供了DISTRIBUTED REPLICATED子句，则MPP数据库会将表的所有行分配给MPP数据库系统中的所有segment。如果用户定义的函数必须在segment上执行，并且这些函数需要访问表的所有行，则可以使用此选项。复制函数还可以用于防止表的broadcast motions，从而提高查询性能。DISTRIBUTED REPLICATED子句不能与PARTITION BY子句或INHERITS子句一起使用。复制的表也不能被另一个表继承。隐藏的系统列

(ctid, cmin, cmax, xmin, xmax和gp_segment_id)无法在复制表的用户查询中引用，因为它们没有单一的，无歧义的值。

通过PARTITION BY子句，您可以将表分为多个子表（或部分），这些子表一起构成父表并共享其模式。尽管子表作为独立表存在，但是MPP数据库以重要方式限制了它们的使用。在内部，分区被实现为继承的一种特殊形式。每个子表分区都是根据不同的CHECK约束创建的，该约束根据一些定义条件限制表可以包含的数据。查询优化器还使用CHECK约束来确定要扫描哪些表分区以满足给定的查询谓词。这些分区约束由MPP数据库自动管理。

参数

GLOBAL | LOCAL

提供这些关键字是为了实现SQL标准兼容性，但在MPP数据库中无效，并且已弃用。

TEMPORARY | TEMP

如果指定，该表将被创建为临时表。临时表在会话结束时或在当前事务结束时自动删除（请参见ON COMMIT）。临时表存在时，具有相同名称的现有永久表在当前会话中不可见，除非使用模式限定名称引用它们。在临时表上创建的所有索引也会自动成为临时索引。

UNLOGGED

如果指定，该表将创建为未记录表。写入未记录表的数据不会写入预写（WAL）日志，这使它们比普通表快得多。但是，未记录表的内容不会复制到mirror实例。同样，未记录的表也不是崩溃安全的。segment实例崩溃或异常关闭后，该segment上未记录表的数据将被截断。在未记录表上创建的所有索引也会自动成为未记录索引。

table_name

要创建的新表的名称（可以由模式指定）。

OF *type_name*

创建一个类型化的表，该表从指定的组合类型（名称可以由模式指定）获取其结构。类型化的表与其类型相关联。例如，如果删除了类型（使用DROP TYPE ... CASCADE），则将删除该表。

当一个类型化的表被创建时，列的数据类型由底层的组合类型决定而没有在CREATE TABLE命令中直接指定。但是CREATE TABLE命令可以对表增加默认值和约束，并且可以指定存储参数。

column_name

要在新表中创建的列的名称。

data_type

列的数据类型。这可能包括数组说明符。

对于包含文本数据的表列，请指定数据类型VARCHAR或TEXT。不建议指定数据类型CHAR。在MPP数据库中，数据类型VARCHAR或TEXT处理作为有效字符添加到数据（在最后一个非空格字符之后添加的空格字符）的填充，而数据类型CHAR不处理。请参阅[注解](#)。

COLLATE *collation*

COLLATE子句为该列分配排序规则（该排序规则必须是可排序的数据类型）。如果未指定，则使用列数据类型的默认排序规则。

Note: 仅当查询中的所有列使用相同的排序规则时，GPORCA才支持排序规则。如果查询中的列使用不同的排序规则，则MPP使用Postgres查询优化器。

DEFAULT *default_expr*

DEFAULT子句为出现在其列定义中的列分配默认数据值。该值是不带变量的表达式（不允许对当前表中的其他列进行子查询和交叉引用）。默认表达式的数据类型必须与列的数据类型匹配。默认表达式将在未为列指定值的任何插入操作中使用。如果列没有默认值，则默认值为null。

ENCODING (*storage_directive* [, ...])

对于列，可选的ENCODING子句指定列数据的压缩类型和块大小。有关compressype，compresslevel和blocksize值，请参见[storage_options](#)。

该子句仅对追加优化的，面向列的表有效。

列压缩设置从表级别继承到分区级别再到子分区级别。最低级别的设置具有优先权。

INHERITS (*parent_table* [, ...])

可选的INHERITS子句指定一个表列表，新表将从中自动继承所有列。使用INHERITS将在新的子表及其父表之间创建持久关系。对父级的模式修改通常也会传播到子级，默认情况下，子级表的数据包含在父级扫描中。在MPP数据库中，创建分区表时不使用INHERITS子句。尽管在分区层次结构中使用了继承的概念，但是使用[PARTITION BY](#)子句创建了分区表的继承结构。

如果一个以上的父表中存在相同的列名，则将报告错误，除非每个父表中的列的数据类型都匹配。如果没有冲突，则将重复的列合并以在新表中形成一个列。如果新表的列名列表包含一个也被继承的列名，则数据类型必须同样与继承的列匹配，并且列定义将合并为一个。如果新表显式指定了该列的默认值，则该默认值将覆盖该列的继承声明中的所有默认值。否则，为该列指定默认值的所有父项都必须指定相同的默认值，否则将报告错误。

CHECK约束基本上以与列相同的方式合并：如果多个父表或新表定义包含名称相同的约束，则这些约束必须全部具有相同的校验表达式，否则将报告错误。具有相同名称和表达式的约束将合并为一个副本。不会考虑在父级中标记为NO INHERIT的约束。请注意，新表中未命名的CHECK约

束将永远不会被合并，因为将始终为其选择唯一的名称。

列STORAGE设置也会从父表中复制。

LIKE *source_table like_option ...*]

LIKE子句指定一个表，新表将从该表中自动复制所有列名，其数据类型，非空约束和分发策略。不会复制诸如追加优化或分区结构之类的存储属性。与INHERITS不同，新表和原始表在创建完成后完全解耦。

仅当指定INCLUDING DEFAULTS时，才会复制复制的列定义的默认表达式。默认行为是排除默认表达式，导致新表中复制的列具有空默认值。

非空约束始终会复制到新表中。仅当指定INCLUDING CONSTRAINTS时，才会复制CHECK约束。列约束和表约束之间没有区别。

仅在指定INCLUDING INDEXES子句的情况下，才会在新表上创建原始表的索引，PRIMARY KEY和UNIQUE约束。不论原始名称如何命名，都会根据默认规则选择新索引和约束的名称。（此行为避免了新索引可能出现的重复名称错误。）

除非指定了INCLUDING INDEXES子句，否则不会在新表上创建原始表上的任何索引。

仅当指定了INCLUDING STORAGE时，才会复制复制的列定义的STORAGE设置。默认行为是排除STORAGE设置，导致新表中复制的列具有特定于类型的默认设置。

仅当指定INCLUDING COMMENTS时，才会复制复制的列，约束和索引的注释。默认行为是排除注释，导致新表中复制的列和约束没有注释。

INCLUDING ALL 是 INCLUDING DEFAULTS INCLUDING CONSTRAINTS INCLUDING INDEXES INCLUDING STORAGE INCLUDING COMMENTS的缩写形式。

请注意，与INHERITS不同，LIKE复制的列和约束不会与名称相似的列和约束合并。如果显式指定了相同的名称，或者在另一个LIKE子句中指定了相同的名称，则将指示错误。

LIKE子句还可用于从视图，外部表或复合类型中复制列。不适用的选项（例如，从视图INCLUDING INDEXES）将被忽略。

CONSTRAINT *constraint_name*

列或表约束的可选名称。如果违反了约束，那么约束名称将出现在错误消息中，因此可以使用约束名称（例如列必须为正）来将有用的约束信息传

达给客户端应用程序。（需要双引号指定包含空格的约束名称。）如果未指定约束名称，则系统将生成一个名称。

Note: 指定的`constraint_name`用于约束，但系统生成的唯一名称用于索引名。在某些以前的版本中，提供的名称同时用于约束名称和索引名称。

NULL | NOT NULL

指定是否允许该列包含空值。默认值为NULL。

CHECK (*expression*) [NO INHERIT]

CHECK子句指定一个生成布尔结果的表达式，新的或更新的行必须满足才能使插入或更新操作成功。计算为TRUE或UNKNOWN的表达式会成功。如果插入或更新操作的任何行都产生FALSE结果，则会引发错误异常，并且插入或更新不会更改数据库。指定为列约束的检查约束应仅引用该列的值，而出现在表约束中的表达式可以引用多个列。

标有NO INHERIT的约束不会传播到子表。

当前，CHECK表达式不能包含子查询，也不能引用当前行的列以外的变量。

UNIQUE (*column_constraint*)

UNIQUE (*column_name* [, ...]) (*table_constraint*)

UNIQUE约束指定表的一组一个或多个列只能包含唯一值。唯一表约束的行为与列约束的行为相同，但具有跨多个列的附加功能。出于唯一约束的目的，空值不视为相等。唯一的列必须包含MPP分布键的所有列。此外，如果表已分区，则<key>必须包含分区键中的所有列。请注意，分区表中的<key>约束与简单的UNIQUE INDEX不同。

有关唯一约束管理和限制的信息，请参见[注解](#)。

PRIMARY KEY (*column_constraint*)

PRIMARY KEY (*column_name* [, ...]) (*table_constraint*)

PRIMARY KEY约束指定表的一列或多列只能包含唯一（非重复），非null值。只能为一个表指定一个主键，无论是作为列约束还是表约束。要使一个表具有主键，它必须是哈希分布的（不是随机分布的），并且主键（唯一的列）必须包含MPP分布键的所有列。此外，如果表已分区，则<key>必须包含分区键中的所有列。请注意，分区表中的<key>约束与简单的UNIQUE INDEX不同。

PRIMARY KEY强制执行与UNIQUE和NOT NULL相同的组合数据约束，但是将一组列标识为主键也可以提供有关模式设计的元数据，因为主键标

识其他表可以依赖这一个列集合作为行的唯一标识符。

有关主键管理和限制的信息，请参阅[注解](#)。

```
REFERENCES reftable [ ( refcolumn ) ]  
[ MATCH FULL | MATCH PARTIAL | MATCH SIMPLE ]  
[ ON DELETE | ON UPDATE ] [ key_action ]  
FOREIGN KEY ( column_name [, ...] )
```

REFERENCES和FOREIGN KEY子句指定引用完整性约束（外键约束）。MPP接受PostgreSQL语法中指定的参照完整性约束，但不强制执行。有关引用完整性约束的信息，请参见PostgreSQL文档。

DEFERRABLE

NOT DEFERRABLE

[NOT] DEFERRABLE子句控制约束是否可以被推迟。一个不可推迟的约束将在每个命令后立即进行检查。可以推迟检查约束，直到事务结束（使用[SET CONSTRAINTS](#)命令）。默认值是NOT DEFERRABLE。当前，只有UNIQUE和PRIMARY KEY约束是可推迟的。NOT NULL和CHECK约束不可延迟。REFERENCES（外键）约束接受此子句，但不强制执行。

INITIALLY IMMEDIATE

INITIALLY DEFERRED

如果约束是可延迟的，则此子句指定检查约束的默认时间。如果该约束是INITIALLY IMMEDIATE，则在每个语句之后对其进行检查。这是默认值。如果约束是INITIALLY DEFERRED，则仅在事务结束时进行检查。可以使用SET CONSTRAINTS命令更改约束检查时间。

WITH (*storage_parameter*=value)

WITH子句可以为表以及与UNIQUE或PRIMARY约束关联的索引指定存储参数。请注意，您还可以通过在分区规范中声明WITH子句来在特定分区或子分区上设置存储参数。最低级别的设置具有优先权。

某些表存储选项的默认值可以使用服务器配置参数

`gp_default_storage_options`指定。有关设置默认存储选项的信息，请参阅[注解](#)。

可以使用以下存储选项：

appendoptimized — 设置为TRUE可将表创建为追加优化的表。如果为FALSE或未声明，则将表创建为常规堆存储表。

blocksize — 设置为表中每个块的大小（以字节为单位）。blocksize必须介于8192和2097152字节之间，并且是8192的倍数。默认值为32768。

orientation — 设置为column以用于列式存储，或设置为row（默认）以用于行式存储。仅当appendoptimized=TRUE时，此选项才有效。堆存储表只能是面向行的。

checksum — 此选项仅对追加优化的表（appendoptimized=TRUE）有效。值TRUE是默认值，并为追加优化表启用CRC校验和验证。校验和是在块创建期间计算的，并存储在磁盘上。在块读取期间执行校验和验证。如果在读取期间计算出的校验和与存储的校验和不匹配，则事务中止。如果将值设置为FALSE以禁用校验和验证，将不会执行检查表数据是否存在磁盘损坏的操作。

compresstype — 设置为ZLIB（默认值），ZSTD，RLE_TYPE或QUICKLZ¹以指定使用的压缩类型。值NONE禁用压缩。Zstd提供速度或良好的压缩率，可通过compresslevel选项进行调整。提供QuickLZ和zlib是为了向后兼容。在通常的工作负载上，Zstd的性能优于这些压缩类型。仅当appendoptimized=TRUE时，compresstype选项才有效。

仅当指定了orientation=column时才支持值RLE_TYPE，它启用游程编码（RLE）压缩算法。当相同的数据值出现在许多连续的行中时，RLE的压缩数据比Zstd，zlib或QuickLZ压缩算法更好。

对于BIGINT，INTEGER，DATE，TIME或TIMESTAMP类型的列，如果将compresstype选项设置为RLE_TYPE压缩，则还将应用增量压缩。增量压缩算法基于连续行中列值之间的增量，旨在改善按排序顺序加载数据或将压缩应用于按排序顺序的列数据时的压缩。

有关使用表压缩的信息，请参阅*MPP数据库管理员手册*中的“选择表存储模型”。

compresslevel — 对于追加优化表的Zstd压缩，请将其设置为1（最快压缩）到19（最高压缩率）之间的整数值。对于zlib压缩，有效范围是1到9。QuickLZ压缩级别只能设置为1。如果未声明，则默认值为1。对于RLE_TYPE，压缩级别可以是1（最快压缩）到4（最高压缩率）之间的

整数值。

仅当appendoptimized=TRUE时，compresslevel选项才有效。

fillfactor — 有关此索引存储参数的更多信息，请参见[CREATE INDEX](#)。

oids — 设置为oids=FALSE（默认值），以便不为行分配对象标识符。

MPP强烈建议您在创建表时不要启用OIDs。在大型表上（例如典型的MPP数据库系统中的表），对表行使用OID可能会导致32位OID计数器的折回。一旦计数器回绕，就不能再认为OID是唯一的，这不仅使它们对用户应用程序无用，而且还会在MPP数据库系统catalog表中引起问题。此外，从表中排除OID会使表每行存储在磁盘上所需的空间减少了每行4个字节，从而略微提高了性能。分区表或追加优化的面向列的表上不允许使用OIDs。

WITH OIDS

WITHOUT OIDS

它们是分别等同于WITH (OIDs)和WITH (OIDs=FALSE)的过时语法。如果希望同时提供OIDs设置和存储参数，则必须使用WITH (...)语法；往上看。MPP强烈建议您在创建表时不要启用OIDs。有关使用**oids**的更多信息，请参见**oids**参数说明。

ON COMMIT

可以使用ON COMMIT控制事务块末尾的临时表的行为。这三个选项是：
PRESERVE ROWS - 临时表的事务结束时不会采取任何特殊操作。这是默认行为。

DELETE ROWS - 临时表中的所有行将在每个事务块的末尾删除。本质上，每次提交都会自动执行一次TRUNCATE。

DROP - 临时表将在当前事务块的末尾删除。

TABLESPACE *tablespace*

要在其中创建新表的表空间的名称。如果未指定，则使用数据库的默认表空间。

USING INDEX TABLESPACE *tablespace*

该子句允许选择将在其中创建与UNIQUE或PRIMARY KEY约束关联的索引的表空间。如果未指定，则使用数据库的默认表空间。

DISTRIBUTED BY (*column* [*opclass*], [...])

DISTRIBUTED RANDOMLY

DISTRIBUTED REPLICATED

用于声明表的MPP数据库分布策略。 DISTRIBUTED BY使用具有一个或多个声明为分布键的列的哈希分布。 为了获得最均匀的数据分配，分布键应为表的主键或唯一列（或一组列）。 如果无法做到这一点，则可以选择DISTRIBUTED RANDOMLY，它将数据轮询发送到segment实例。 此外，可以指定运算符类opclass，以使用非默认哈希函数。

如果在创建表时未指定DISTRIBUTED BY子句，则MPP数据库服务器配置参数gp_create_table_random_default_distribution将控制表默认分布策略。 如果未指定分布策略，MPP数据库将遵循以下规则来创建表。

如果参数的值是off（默认值），则MPP数据库根据以下命令选择表分布键：

- 如果指定了LIKE或INHERITS子句，则MPP从源表或父表复制分布键。
- 如果指定了PRIMARY KEY或UNIQUE约束，则MPP选择所有键列中最大的子集作为分布键。
- 如果既未指定约束，也未指定LIKE或INHERITS子句，则MPP选择第一个合适的列作为分布键。（具有几何或用户定义数据类型的列不符合作为MPP分布键列的条件。）

如果参数的值设置为on，则MPP数据库遵循以下规则：

- 如果未指定PRIMARY KEY或UNIQUE列，则表的分布是随机的（DISTRIBUTED RANDOMLY）。即使表创建命令包含LIKE或INHERITS子句，表分布也是随机的。
- 如果指定了PRIMARY KEY或UNIQUE列，则还必须指定DISTRIBUTED BY子句。如果在表创建命令中未指定DISTRIBUTED BY子句，则该命令将失败。

有关设置默认表分布策略的更多信息，请参

见[gp_create_table_random_default_distribution](#)。

DISTRIBUTED REPLICATED子句将整个表复制到所有MPP数据库segment实例。 当函数需要访问表中的所有行或需要通过阻止broadcast motion来提高查询性能时，必须在segment上执行用户定义的函数时可以使用它。

PARTITION BY

声明用于对表进行分区的一列或多列。

创建分区表时，MPP数据库使用指定的表名创建根分区表（根分区）。MPP数据库还会根据您指定的分区选项创建表，子表的层次结构，这些表是子分区。MPP数据库`pg_partition*`系统视图包含有关子分区表的信息。对于每个分区级别（表的每个层次结构级别），一个分区表最多可以有32,767个分区。

Note: MPP数据库将分区表数据存储在叶子表中，叶子表是子表层次结构中的最低级表，供分区表使用。

partition_type

声明分区类型：LIST（值列表）或RANGE（数字或日期范围）。

partition_specification

声明要创建的各个分区。可以单独定义每个分区，或者对于范围分区，可以使用EVERY子句（带有START和可选END子句）来定义用于创建单个分区的增量模式。

DEFAULT PARTITION *name* — 声明默认分区。当数据与现有分区不匹配时，会将其插入默认分区。没有默认分区的分区设计将拒绝与现有分区不匹配的传入行。

PARTITION *name* — 声明要用于分区的名称。使用以下命名约定创建分区：`parentname_level#_prt_givenname`。

VALUES — 对于列表分区，定义分区将包含的值。

START — 对于范围分区，定义分区的起始范围值。默认情况下，起始值为INCLUSIVE。例如，如果您声明开始日期为'2016-01-01'，则分区将包含所有大于或等于'2016-01-01'的日期。通常，START表达式的数据类型与分区键列的类型相同。如果不是这种情况，则必须显式转换为预期的数据类型。

END — 对于范围分区，定义分区的结束范围值。默认情况下，结束值为EXCLUSIVE。例如，如果您声明结束日期为'2016-02-01'，则分区将包含所有小于但不等于'2016-02-01'的日期。通常，END表达式的数据类型与分区键列的类型相同。如果不是这种情况，则必须显式转换为预期的数据类型。

EVERY — 对于范围分区，定义如何将值从START递增到END以创建单个分区。通常，EVERY表达式的数据类型与分区键列的类型相同。如果不是这种情况，则必须显式转换为预期的数据类型。

WITH — 设置分区的表存储选项。例如，您可能希望将较旧的分区作为追加优化表，而将较新的分区作为常规堆表。

TABLESPACE — 要在其中创建分区的表空间的名称。

SUBPARTITION BY

声明用于对表的第一级分区进行子分区的一个或多个列。子分区的规范格式类似于上述分区的规范格式。

SUBPARTITION TEMPLATE

您可以选择声明一个用于创建子分区的子分区模板（低级别子表），而不是为每个分区分别声明每个子分区定义。然后，此子分区规范将应用于所有父分区。

注解

- 在MPP数据库（基于Postgres的系统）中，数据类型VARCHAR或TEXT处理填充作为有效字符添加到文本的数据（最后一个非空格字符之后添加空格字符）；数据类型CHAR则没有。
在MPP数据库中，CHAR(*n*)类型的值用尾随空格填充到指定的宽度*n*。值将存储并显示为空格。但是，填充空格在语义上无关紧要。分配值时，将忽略尾随空格。比较数据类型CHAR的两个值时，尾随空格在语义上也无关紧要，而将字符值转换为其他字符串类型之一时，尾随空格也将被删除。
- 不建议在新应用程序中使用OID：在可能的情况下，最好使用SERIAL或其他序列生成器作为表的主键。但是，如果您的应用程序确实使用OID来标识表的特定行，则建议在该表的OID列上创建唯一约束，以确保即使在计数器回绕后，表中的OID的确可以唯一地标识行。避免假设OID在表之间是唯一的；如果需要数据库范围的唯一标识符，则可以使用表OID和行OID的组合。
- MPP数据库对于主键和作为MPP表中的分布键的列的唯一约束具有一些特殊条件。为了在MPP数据库中实施唯一约束，表必须是哈希分布的（不是DISTRIBUTED RANDOMLY），并且约束列必须与表的分布键列相同（或作为其的超集）。另外，分布键必须是约束列的左子集，并且列的顺序正确。例如，如果主键是（a，b，c），则分布键只能是以下之一：（a），

(a, b) 或 (a, b, c)。

复制表 (DISTRIBUTED REPLICATED) 可以同时具有PRIMARY KEY和UNIQUE列约束。

主键约束只是唯一约束和非空约束的组合。

MPP数据库自动为每个UNIQUE或PRIMARY KEY约束创建一个UNIQUE索引，以强制执行唯一性。因此，没有必要为主键列显式创建索引。在追加优化表上不允许使用UNIQUE和PRIMARY KEY约束，因为在追加优化表上不允许通过约束创建的UNIQUE索引。

MPP数据库中不支持外键约束。

对于继承的表，当前实现中不会继承唯一约束，主键约束，索引和表特权。

- 对于追加优化的表，可重复读或可序列化事务中不允许UPDATE和DELETE，这将导致事务中止。
CLUSTER，DECLARE...FOR UPDATE和触发器不支持追加优化的表。
- 要将数据插入分区表中，请指定根分区表，即使用CREATE TABLE命令创建的表。您还可以在INSERT命令中指定分区表的叶子表。如果数据对于指定的叶子表无效，则返回错误。不支持在INSERT命令中指定不是叶子表的子表。不支持在分区表的任何子表上执行其他DML命令，例如UPDATE和DELETE。这些命令必须在根分区表（使用CREATE TABLE命令创建的表）上执行。
- 可以使用服务器配置参数gp_default_storage_option指定这些表存储选项的默认值。
 - appendoptimized
 - blocksize
 - checksum
 - compressype
 - compresslevel
 - orientation

可以为系统，数据库或用户设置默认值。有关设置存储选项的信息，请参阅服务器配置参数[gp_default_storage_options](#)。

Important: 当前的MPP数据库Postgres优化器允许使用具有多列（复合）分区键

的列表分区。 GPORCA不支持复合键，因此不建议使用复合分区键。

示例

在名为baby的模式中创建一个名为rank的表，并使用rank，gender和year列分发数据：

```
CREATE TABLE baby.rank (id int, rank int, year smallint,  
gender char(1), count int ) DISTRIBUTED BY (rank, gender,  
year);
```

创建表files和表分配器（默认情况下，主键将用作MPP分布键）：

```
CREATE TABLE films (  
code      char(5) CONSTRAINT firstkey PRIMARY KEY,  
title     varchar(40) NOT NULL,  
did       integer NOT NULL,  
date_prod date,  
kind      varchar(10),  
len       interval hour to minute  
);
```

```
CREATE TABLE distributors (  
did  integer PRIMARY KEY DEFAULT nextval('serial'),  
name varchar(40) NOT NULL CHECK (name <> "")  
);
```

创建一个gzip压缩的，追加优化的表：

```
CREATE TABLE sales (txn_id int, qty int, date date)  
WITH (appendoptimized=true, compresslevel=5)  
DISTRIBUTED BY (txn_id);
```

使用子分区模板和每个级别的默认分区创建一个三级分区表：

```
CREATE TABLE sales (id int, year int, month int, day int,  
region text)  
DISTRIBUTED BY (id)  
PARTITION BY RANGE (year)  
  
SUBPARTITION BY RANGE (month)  
SUBPARTITION TEMPLATE (  
START (1) END (13) EVERY (1),  
DEFAULT SUBPARTITION other_months )  
  
SUBPARTITION BY LIST (region)  
SUBPARTITION TEMPLATE (  
SUBPARTITION usa VALUES ('usa'),  
SUBPARTITION europe VALUES ('europe'),  
SUBPARTITION asia VALUES ('asia'),  
DEFAULT SUBPARTITION other_regions)  
  
( START (2008) END (2016) EVERY (1),  
DEFAULT PARTITION outlying_years);
```

兼容性

CREATE TABLE命令符合SQL标准，但以下情况除外：

- **Temporary Tables** — 在SQL标准中，临时表仅定义一次，并在每个需要它们的会话中自动存在（从空内容开始）。相反，MPP数据库要求每个会话为要使用的每个临时表发出自己的CREATE TEMPORARY TABLE命令。这允许不同的会话出于不同的目的而使用相同的临时表名称，而标准的方法将给定临时表名称的所有实例约束为具有相同的表结构。
全局和本地临时表之间的标准区别不在MPP数据库中。MPP数据库将在临时表声明中接受GLOBAL和LOCAL关键字，但它们无效且已弃用。
如果省略ON COMMIT子句，则SQL标准将默认行为指定为ON COMMIT DELETE ROWS。但是，MPP数据库中的默认行为是ON COMMIT PRESERVE ROWS。SQL标准中不存在ON COMMIT DROP选项。

- **Column Check Constraints** — SQL标准说，CHECK列约束只能引用它们所适用的列。只有CHECK表约束可以引用多个列。MPP数据库不强制执行此限制；它对待列和表检查约束都一样。
- **NULL Constraint** — NULL约束是对SQL标准的MPP数据库扩展，为了与某些其他数据库系统兼容（以及对称的NOT NULL约束）。由于它是任何列的默认值，因此不需要它的存在。
- **Inheritance** — 通过INHERITS子句的多重继承是MPP数据库语言的扩展。SQL：1999及更高版本使用不同的语法和语义定义了单个继承。MPP数据库尚不支持SQL：1999样式的继承。
- **Partitioning** — 通过PARTITION BY子句进行的表分区是MPP数据库语言的扩展。
- **Zero-column tables** — MPP数据库允许创建不包含任何列的表（例如CREATE TABLE foo();）。这是SQL标准的扩展，不允许使用零列表。零列表本身并没有什么用，但是不允许使用零列表在ALTER TABLE DROP COLUMN时会产生奇怪的特殊情况，因此MPP决定忽略此规范限制。
- **LIKE** — 尽管SQL标准中存在LIKE子句，但MPP数据库接受的许多选项都不在该标准中，并且MPP数据库并未实现该标准的某些选项。
- **WITH clause** — WITH子句是MPP数据库扩展。存储参数和OID都不在标准中。
- **Tablespaces** — 表空间的MPP数据库概念不是SQL标准的一部分。子句TABLESPACE和USING INDEX TABLESPACE是扩展。
- **Data Distribution** — 并行或分布式数据库的MPP数据库概念不是SQL标准的一部分。DISTRIBUTED子句是扩展。

另见

[ALTER TABLE](#) , [DROP TABLE](#) , [CREATE EXTERNAL TABLE](#) , [CREATE TABLE AS](#)

Parent topic: [SQL Command Reference](#)

CREATE TABLE AS

根据查询结果定义一个新表。

概要

```
CREATE [ [ GLOBAL | LOCAL ] { TEMPORARY | TEMP } | UNLOGGED ] TABLE table_name
    [ ( column_name [, ...] ) ]
    [ WITH ( storage_parameter [= value] [, ... ] ) | WITH OIDS | WITHOUT OIDS ]
    [ ON COMMIT { PRESERVE ROWS | DELETE ROWS | DROP } ]
    [ TABLESPACE tablespace_name ]
    AS query
    [ WITH [ NO ] DATA ]
    [ DISTRIBUTED BY (column [, ... ] ) | DISTRIBUTED RANDOMLY | DISTRIBUTED REP
```

其中 *storage_parameter* 是：

```
appendoptimized={TRUE|FALSE}
blocksize={8192-2097152}
orientation={COLUMN|ROW}
compresstype={ZLIB|ZSTD|QUICKLZ}
compresslevel={1-19 | 1}
fillfactor={10-100}
oids[=TRUE|FALSE]
```


描述

CREATE TABLE AS创建一个表，并用SELECT命令计算的数据填充该表。表列具有与SELECT的输出列关联的名称和数据类型，但是您可以通过提供新列名称的显式列表来覆盖列名称。

CREATE TABLE AS创建一个新表并仅对查询进行一次取值以填充一次新表。新表将不会跟踪对查询源表的后续更改。

参数

GLOBAL | LOCAL

忽略兼容性。这些关键字已弃用；有关详细信息，请参考CREATE TABLE。

TEMPORARY | TEMP

如果指定，则将新表创建为临时表。临时表在会话结束时或在当前事务结束时自动删除（请参见ON COMMIT）。临时表存在时，具有相同名称的现有永久表在当前会话中不可见，除非使用模式限定名称引用它们。在临时表上创建的所有索引也会自动成为临时索引。

UNLOGGED

如果指定，该表将创建为未记录表。写入未记录表的数据不会写入预写（WAL）日志，这使它们比普通表快得多。但是，未记录表的内容不会复制到mirror实例。同样，未记录的表也不是崩溃安全的。segment实例崩溃或异常关闭后，该segment上未记录表的数据将被截断。在未记录表上创建的所有索引也会自动成为未记录索引。

table_name

要创建的新表的名称（可以由模式指定）。

column_name

新表中的列名。如果未提供列名，则它们取自查询的输出列名。

WITH (*storage_parameter=value*)

WITH子句可用于设置表或其索引的存储选项。请注意，您还可以通过在分区规范中声明WITH子句，在特定分区或子分区上设置不同的存储参

数。可以使用以下存储选项：

appendoptimized — 设置为TRUE可将表创建为追加优化的表。如果为FALSE或未声明，则将表创建为常规堆存储表。

blocksize — 设置为表中每个块的大小（以字节为单位）。blocksize必须介于8192和2097152字节之间，并且是8192的倍数。默认值为32768。

orientation — 设置为column以用于列式存储，或设置为row（默认）以用于行式存储。仅当appendoptimized=TRUE时，此选项才有效。堆存储表只能是面向行的。

compresstype — 设置为ZLIB（默认值），ZSTD或QUICKLZ¹以指定使用的压缩类型。值NONE禁用压缩。Zstd提供速度或良好的压缩率，可通过compresslevel选项进行调整。提供QuickLZ和zlib是为了向后兼容。在通常的工作负载上，Zstd的性能优于这些压缩类型。仅当appendoptimized=TRUE时，compresstype选项才有效。

compresslevel — 对于附加优化表的Zstd压缩，请将其设置为1（最快压缩）到19（最高压缩率）之间的整数值。对于zlib压缩，有效范围是1到9。QuickLZ压缩级别只能设置为1。如果未声明，则默认值为1。

compresslevel选项仅在appendoptimized=TRUE时有效。

fillfactor — 有关此索引存储参数的更多信息，请参见[CREATE INDEX](#)。

oids — 设置为oids=FALSE（默认值），以便不为行分配对象标识符。

MPP强烈建议您在创建表时不要启用OIDS。在大型表上（例如典型的MPP数据库系统中的表），对表行使用OID可能会导致32位OID计数器的折回。一旦计数器回绕，就不能再认为OID是唯一的，这不仅使它们对用户应用程序无用，而且还会在MPP数据库系统catalog表中引起问题。此外，从表中排除OID会使表每行存储在磁盘上所需的空间减少了每行4个字节，从而略微提高了性能。面向列的表不允许使用OIDS。

ON COMMIT

可以使用ON COMMIT控制事务块末尾的临时表的行为。这三个选项是：

PRESERVE ROWS — 临时表的事务结束时不会采取任何特殊操作。这是默认行为。

DELETE ROWS — 临时表中的所有行将在每个事务块的末尾删除。本质上，每次提交都会自动执行一次TRUNCATE。

DROP — 临时表将在当前事务块的末尾删除。

TABLESPACE *tablespace_name*

`tablespace_name`参数是要在其中创建新表的表空间的名称。如果未指定，则使用数据库的默认表空间。

AS query

[SELECT](#)，[TABLE](#)或[VALUES](#)命令，或运行准备的SELECT或VALUES查询的[EXECUTE](#)命令。

DISTRIBUTED BY ({*column* [*opclass*]}, [...])

DISTRIBUTED RANDOMLY

DISTRIBUTED REPLICATED

用于声明表的MPP数据库分发策略。DISTRIBUTED BY使用具有一个或多个声明为分发键的列的哈希分发。为了获得最均匀的数据分配，分发键应为表的主键或唯一列（或一组列）。如果无法做到这一点，则可以选择DISTRIBUTED RANDOMLY，它将数据轮询发送到segment实例。DISTRIBUTED REPLICATED将表中的所有行复制到所有MPP数据库segment。它不能与分区表或从其他表继承的表一起使用。

如果在创建表时未指定DISTRIBUTED BY子句，则MPP数据库服务器配置参数`gp_create_table_random_default_distribution`将控制默认表分发策略。如果未指定分发策略，MPP数据库将遵循以下规则来创建表。

- 如果Postgres查询优化器创建了表，并且该参数的值是off，则根据命令确定表分配策略。
- 如果Postgres查询优化器创建了表，并且该参数的值为on，则表分配策略是随机的。
- 如果GPORCA创建表，则表分配策略是随机的。参数值无效。

有关设置默认表分发策略的更多信息，请参

见[gp_create_table_random_default_distribution](#)。有关Postgres查询优化器和GPORCA的信息，请参阅[MPP数据库管理员手册](#)中的[查询数据](#)。

注解

该命令在功能上类似于[SELECT INTO](#)，但它是首选的，因为它不太可能与SELECT INTO语法的其他用法混淆。此外，CREATE TABLE AS提供

了SELECT INTO提供的功能的超集。

CREATE TABLE AS可用于从外部表数据源快速加载数据。 请参阅[CREATE EXTERNAL TABLE](#)。

示例

创建一个新表films_recent，该表仅包含表films中的最新条目：

```
CREATE TABLE films_recent AS SELECT * FROM films WHERE  
date_prod >= '2007-01-01';
```

使用预编译语句创建一个新的临时表films_recent，该表仅包含表films中的最新条目。 新表具有OID，并将在提交时删除：

```
PREPARE recentfilms(date) AS SELECT * FROM films WHERE  
date_prod > $1;  
CREATE TEMP TABLE films_recent WITH (OIDs) ON COMMIT DROP AS  
EXECUTE recentfilms('2007-01-01');
```

兼容性

CREATE TABLE AS符合SQL标准，但以下情况除外：

- 该标准要求子查询子句两边加上括号；在MPP数据库中，这些括号是可选的。
 - 该标准定义了WITH [NO] DATA子句；MPP数据库当前未实现此功能。 MPP数据库提供的行为等同于标准的WITH DATA情况。 可以通过将LIMIT 0附加到查询来模拟WITH NO DATA。
-

- MPP数据库对临时表的处理方式与标准处理方式有所不同。 有关详细信息，请参见CREATE TABLE。
- WITH子句是MPP数据库扩展。 storage参数和OIDs都不在标准中。
- 表空间的MPP数据库概念不是该标准的一部分。 TABLESPACE子句是扩展。

另见

[CREATE EXTERNAL TABLE](#), [CREATE EXTERNAL TABLE](#), [EXECUTE](#),
[SELECT](#), [SELECT INTO](#), [VALUES](#)

Parent topic: [SQL Command Reference](#)

CREATE TABLESPACE

定义一个新的表空间。

概要

```
CREATE TABLESPACE tablespace_name [OWNER username] LOCATION '/path/to/dir'  
[WITH (contentID_1='/path/to/dir1', contentID_2='/path/to/dir2 ... )]]
```

描述

CREATE TABLESPACE为MPP数据库系统注册并配置新的表空间。表空间名称必须不同于系统中任何现有表空间的名称。表空间是MPP数据库系统对象（全局对象），如果您具有适当的特权，则可以使用任何数据库中的表空间。

表空间允许超级用户定义包含数据库对象（例如表和索引）的数据文件所在的备用主机文件系统位置。

具有适当特权的用户可以将表空间名称传递给[CREATE DATABASE](#)，[CREATE DATABASE](#)或[CREATE INDEX](#)，以将这些对象的数据文件存储在指定的表空间中。

在MPP数据库中，文件系统位置必须存在于所有主机上，包括运行master，standby，每个primary和每个mirror的主机。

参数

tablespacename

要创建的表空间的名称。该名称不能以pg_或gp_开头，因为此类名称是为系统表空间保留的。

OWNER *username*

拥有表空间的用户名。如果省略，则默认为执行命令的用户。只有超级用户可以创建表空间，但是他们可以将表空间的所有权分配给非超级用户。

LOCATION '/path/to/dir'

目录的绝对路径（主机系统文件位置），它将是表空间的根目录。注册表空间时，该目录应该为空，并且必须归MPP数据库系统用户所有。该目录必须由绝对路径名指定。

对于每个segment实例，您可以在WITH子句中为表空间指定不同的目录。

contentID_i='/path/to/dir_i'

值ID_i是segment实例的content ID。/path/to/dir_i是segment实例用作表空间的根目录的主机系统文件位置的绝对路径。您不能指定master实例的

content ID (-1) 。您可以为多个segment指定相同的目录。如果WITH子句中未列出segment实例，则MPP数据库将使用LOCATION子句指定的目录。注册表空间时，目录应为空，并且必须由MPP数据库系统用户拥有。

注解

表空间仅在支持符号链接的系统上受支持。

CREATE TABLESPACE不能在事务块内执行。

创建表空间时，请确保文件系统位置具有足够的I/O速度和可用磁盘空间。

CREATE TABLESPACE创建从master和segment实例数据目录中的pg_tblspc目录到命令中指定目录的符号链接。

系统catalog表pg_tablespace存储表空间信息。此命令显示表空间的OID值，名称和所有者。

```
SELECT oid, spcname, spcowner FROM pg_tablespace ;
```

MPP数据库内置函数gp_tablespace_location(*tablespace_oid*)显示所有segment实例的表空间主机系统文件位置。此命令列出OID为16385的表空间的segment数据库ID和主机系统文件位置。

```
SELECT * FROM gp_tablespace_location(16385)
```

示例

创建一个新表空间，并为master和所有segment实例指定文件系统位置：

```
CREATE TABLESPACE mytblspace LOCATION '/gpdbtspc/mytestspace' ;
```

创建一个新表空间，并为content ID为0和1的segment实例指定一个位置。对于未在WITH子句中列出的master和segment实例，在LOCATION子句中指定该表空间的文件系统位置。

```
CREATE TABLESPACE mytblspace LOCATION '/gpdbtspc/mytestspace' WITH (content0='/te
```

该示例为两个segment实例指定了相同的位置。您可以为每个segment指定不同的位置。

兼容性

CREATE TABLESPACE是MPP数据库扩展。

另见

[CREATE DATABASE](#), [CREATE TABLE](#), [CREATE INDEX](#), [DROP TABLESPACE](#), [ALTER TABLESPACE](#)

Parent topic: [SQL Command Reference](#)

CREATE TEXT SEARCH

CONFIGURATION

定义新的文本搜索配置。

概要

```
CREATE TEXT SEARCH CONFIGURATION name (  
  PARSER = parser_name |  
  COPY = source_config  
)
```

描述

CREATE TEXT SEARCH CONFIGURATION创建一个新的文本搜索配置。文本搜索配置指定了文本搜索解析器，该文本搜索解析器可以将字符串划分为标记，外加一些词典（可被用来决定哪些标记是搜索感兴趣的）。

如果仅指定了解析器，则新的文本搜索配置最初没有从标记类型到字典的映射，因此将忽略所有单词。随后的ALTER TEXT SEARCH CONFIGURATION命令必须用于创建映射以使配置有用。或者，可以复制现有的文本搜索配置。

如果指定了模式名称，则会在指定的模式中创建文本搜索配置。否则，它将在当前模式中创建。

定义文本搜索配置的用户将成为其所有者。

更多信息，请参考[使用全文搜索](#)。

参数

- name*
要创建的文本搜索配置的名称。该名称可以是模式限定的。
- parser_name*
用于此配置的文本搜索解析器的名称。
- source_config*
要复制的现有文本搜索配置的名称。

注解

PARSER和COPY选项是互斥的，因为复制现有配置时，它的解析器选择也会被复制。

兼容性

SQL标准中没有CREATE TEXT SEARCH CONFIGURATION语句。

另见

[ALTER TEXT SEARCH CONFIGURATION](#) , [DROP TEXT SEARCH CONFIGURATION](#)

Parent topic: [SQL Command Reference](#)

CREATE TEXT SEARCH DICTIONARY

定义一个新的文本搜索字典。

概要

```
CREATE TEXT SEARCH DICTIONARY name (  
    TEMPLATE = template  
    [, option = value [, ... ]]  
)
```

描述

CREATE TEXT SEARCH DICTIONARY 创建一个新的文本搜索字典。文本搜索字典指定一种识别有趣或不有趣的单词的搜索方式。词典取决于文本搜索模板，该模板指定了实际执行工作的函数。通常，字典提供一些选项来控制模板函数的详细行为。

如果指定了模式名称，则会在指定的模式中创建文本搜索字典。否则，它将在当前模式中创建。

定义文本搜索词典的用户将成为其所有者。

更多信息请参考[使用全文搜索](#)。

参数

name

要创建的文本搜索字典的名称。该名称可以由模式指定。

template

文本搜索模板的名称，它将定义此词典的基本行为。

option

要为此字典设置的模板特定选项的名称。

value

用于模板特定选项的值。如果该值不是简单的标识符或数字，则必须用引号引起来（但如果需要，您可以始终用引号引起来）。

选项可以按任何顺序出现。

示例

以下示例命令使用不标准的停用词列表创建一个基于Snowball的字典。

```
CREATE TEXT SEARCH DICTIONARY my_russian (  
  template = snowball,  
  language = russian,  
  stopwords = myrussian  
);
```

兼容性

SQL标准中没有CREATE TEXT SEARCH DICTIONARY语句。

另见

[ALTER TEXT SEARCH DICTIONARY](#) , [DROP TEXT SEARCH DICTIONARY](#)

Parent topic: [SQL Command Reference](#)

CREATE TEXT SEARCH PARSER

Description

定义一个新的文本搜索解析器。

概要

```
CREATE TEXT SEARCH PARSER name (  
    START = start_function ,  
    GETTOKEN = gettoken_function ,  
    END = end_function ,  
    LEXTYPES = lextypes_function  
    [, HEADLINE = headline_function ]  
)
```

描述

CREATE TEXT SEARCH PARSER创建一个新的文本搜索解析器。 文本搜索解析器定义了一种方法，用于将文本字符串拆分为标记并为标记分配类型（类

别)。解析器本身并不是特别有用，但必须与一些用于搜索的文本搜索字典一起绑定到文本搜索配置中。

如果指定了模式名称，则会在指定的模式中创建文本搜索解析器。否则，它将在当前模式中创建。

您必须是超级用户才能使用CREATE TEXT SEARCH PARSER。（之所以做出此限制，是因为错误的文本搜索解析器定义可能会使服务器困惑甚至崩溃。）

更多信息请参考[使用全文搜索](#)。

参数

name

要创建的文本搜索解析器的名称。该名称可以由模式指定。

start_function

解析器的启动函数的名称。

gettoken_function

解析器的get-next-token函数的名称。

end_function

解析器的结束函数的名称。

lextypes_function

解析器的lextypes函数的名称（该函数返回有关它产生的标记类型集的信息）。

headline_function

解析器的标题函数的名称（该函数汇总一组标记）。

如有必要，可以对函数名称进行模式限定。由于每种函数类型的参数列表都是预先确定的，因此未提供参数类型。除标题函数外，所有其他函数都是必需的。

参数可以按任何顺序出现，而不仅仅是上面显示的顺序。\\

兼容性

SQL标准中没有CREATE TEXT SEARCH PARSE语句。

另见

[ALTER TEXT SEARCH PARSE](#), [DROP TEXT SEARCH PARSE](#)

Parent topic: [SQL Command Reference](#)

CREATE TEXT SEARCH TEMPLATE

Description

定义一个新的文本搜索模板。

概要

```
CREATE TEXT SEARCH TEMPLATE name (  
  [ INIT = init_function , ]  
  LEXIZE = lexize_function  
)
```

描述

CREATE TEXT SEARCH TEMPLATE创建一个新的文本搜索模板。文本搜索模板定义了实现文本搜索词典的函数。模板本身不是有用的，但必须实例化为要使用的字典。字典通常指定要提供给模板函数的参数。

如果指定了模式名称，则会在指定的模式中创建文本搜索模板。否则，它将在当前模式中创建。

您必须是超级用户才能使用CREATE TEXT SEARCH TEMPLATE。之所以做出此限制，是因为错误的文本搜索模板定义可能会使服务器混乱甚至崩溃。将模板与词典分开的原因是模板封装了定义词典的“不安全”方面。定义字典时可以设置的参数对于没有特权的用户是安全的，因此创建字典不必是特权操作。

更多信息请参考[使用全文搜索](#)。

参数

name

要创建的文本搜索模板的名称。该名称可以由模式指定。

init_function

模板的初始化函数的名称。

lexize_function

模板的lexize函数的名称。

如有必要，可以对函数名称进行模式限定。由于每种函数类型的参数列表都是预先确定的，因此未提供参数类型。lexize函数是必需的，但init函数是可选的。

参数可以以任何顺序出现，而不仅是上面显示的顺序。

兼容性

SQL标准中没有CREATE TEXT SEARCH TEMPLATE语句。

另见

[CREATE TEXT SEARCH TEMPLATE](#), [DROP TEXT SEARCH TEMPLATE](#)

Parent topic: [SQL Command Reference](#)

CREATE TYPE

定义新的数据类型。

概要

```
CREATE TYPE name AS  
( attribute_name data_type [ COLLATE collation ] [, ... ] )
```

```
CREATE TYPE name AS ENUM  
( [ 'label' [, ... ] ] )
```

```
CREATE TYPE name AS RANGE (  
    SUBTYPE = subtype  
    [ , SUBTYPE_OPCLASS = subtype_operator_class ]  
    [ , COLLATION = collation ]  
    [ , CANONICAL = canonical_function ]  
    [ , SUBTYPE_DIFF = subtype_diff_function ]  
)
```

```
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 ]  
    [ , INTERNALLENGTH = {internallength | VARIABLE} ]  
    [ , PASSEDBYVALUE ]  
    [ , ALIGNMENT = alignment ]  
    [ , STORAGE = storage ]  
    [ , LIKE = like_type ]  
    [ , CATEGORY = category ]  
    [ , PREFERRED = preferred ]  
    [ , DEFAULT = default ]  
    [ , ELEMENT = element ]  
    [ , DELIMITER = delimiter ]  
    [ , COLLATABLE = collatable ]  
    [ , COMPRESSTYPE = compression_type ]  
    [ , COMPRESSLEVEL = compression_level ]  
    [ , BLOCKSIZE = blocksize ] )
```

```
CREATE TYPE name
```

CREATE TYPE注册一个新的数据类型以供当前数据库使用。定义类型的用户将成为其所有者。

如果提供了模式名称，则在指定的模式中创建类型。否则，它将在当前模式中创建。类型名称必须不同于同一模式中任何现有类型或域的名称。类型名称还必须与同一模式中任何现有表的名称都不同。

如上面的语法概要所示，有五种形式的CREATE TYPE。它们分别创建组合类型，枚举类型，范围类型，基础类型或shell类型。下面依次讨论其中的前四个。shell类型只是用于稍后定义的类型占位符；它是通过发出CREATE TYPE来创建的，除了类型名称外，不带任何参数。如这些部分所述，在创建范围类型和基础类型时，需要使用Shell类型作为前向引用。

组合类型

第一种形式的CREATE TYPE创建组合类型。组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。组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。单独的组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。

为了能够创建组合类型，必须拥有在其所有属性类型上的USAGE特权。

枚举类型

正如PostgreSQL文档中描述的[枚举类型](#)，第二种形式的CREATE TYPE创建枚举类型。枚举类型需要由一个或者更多带引号的标签构成的列表，每一个标签长度必须不超过NAMEDATALEN字节（在标准的PostgreSQL编译中是64字节）。

范围类型

正如[范围类型](#)描述，第三种形式的CREATE TYPE创建范围类型。

范围类型的`subtype`可以是任何带有一个相关的B树操作符类（用来决定该范围类型值的顺序）的类型。通常，子类型的默认B树操作符类被用来决定顺序。要使用一种非默认操作符类，可以用`subtype_opclass`指定它的名字。如果子类型是可排序的并且希望在该范围的顺序中使用一种非默认的排序规则，可以用`collation`选项来指定。

可选的`canonical`函数必须接受一个所定义的范围类型的参数，并且返回同样类型的一个值。在适用时，它被用来把范围值转换成一种规范的形式。更多信息请见[定义新的范围类型](#)。创建一个`canonical`函数有点棘手，因为必须在声明范围类型之前定义它。要这样做，必须首先创建一种shell类型，它是一种没有属性只有名称和拥有者的占位符类型。这可以通过发出`CREATE TYPE name`来完成。然后可以使用该shell类型作为参数和结果来声明该函数，并且最终用同样的名称来声明范围类型。这会自动用一种合法的范围类型替换shell类型项。

可选的`subtype_diff`函数必须接受两个`subtype`类型的值作为参数，并且返回一个double precision值表示两个给定值之间的差别。虽然这是可选的，但是提供这个函数会让该范围类型列上GiST索引效率更高。更多信息请参考[定义新的范围类型](#)。

Base Types

第四种形式的`CREATE TYPE`创建一种新的基础类型（标量类型）。为了创建一种新的基础类型，你必须是一个超级用户。参数可以以任意顺序出现（而不仅是按照语法所示的顺序），并且大部分是可选的。在定义类型前，必须注册两个或者更多函数（使用`CREATE FUNCTION`）。支持函数`input_function`以及`output_function`是必需的，而函数`receive_function`、`send_function`、`type_modifier_input_function`、`type_modifier_output_function`和`analyze_function`是可选的。通常来说这些函数必须是用C或者另外一种低层语言编写的。在MPP数据库中，用于实现数据类型的任何函数都必须定义为`IMMUTABLE`。

`input_function`将类型的外部文本表达转换成为该类型定义的操作符和函数所使用的内部表达。`output_function`执行反向的转换。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。第一个参数是以C字符串存在的输入文本，第二个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），第三个参数是目标列的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，调用它时第一个参数会是NULL。在这种情况下，该函数必须仍然返回NULL，除非它发生了错误（这种情况主要是想支持域输入函数，它们可能需要拒绝NULL输入）。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。

可选的*receive_function*会把类型的外部二进制表达转换成内部表达。如果没有提供这个函数，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）。接收函数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。第一个参数是一个指向StringInfo缓冲区的指针，其中保存着接收到的字节串。其余可选的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，调用它时第一个参数会是NULL。在这种情况下，该函数必须仍然返回NULL，除非它发生了错误（这种情况主要是想支持域接收函数，它们可能需要拒绝NULL输入）。类似地，可选的*send_function*将内部表达转换成外部二进制表达。如果没有提供这个函数，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。

如果类型支持修饰符，则需要可选

的*type_modifier_input_function*和*type_modifier_output_function*。修饰符是附加到类型声明的可选约束，例如char(5)或numeric(30,2)。尽管MPP数据库允许用户定义的类型将一个或多个简单的常量或标识符用作修饰符，但此信息必须适合单个非负整数值，以存储在系统catalog中。MPP数据库以cstring数组的形式将声明的修饰符传递给*type_modifier_input_function*。修饰符输入函数必须检查值的有效性，如果值不正确，则会引发错误。如果值正确，则修饰符输入

函数将返回单个非负整数值，该值将被MPP数据库存储为typmod列。如果未使用`type_modifier_input_function`定义类型，则拒绝类型修饰符。

`type_modifier_output_function`将内部整数typmod值转换回正确的格式以供用户显示。修饰符输出函数必须返回一个cstring值，该值是要附加到类型名称后的确切字符串。例如，数字函数可能返回(30,2)。

`type_modifier_output_function`是可选的。如果未指定，则默认显示格式为括号内存储的typmod整数值。

到这里你应该在疑惑输入和输出函数是如何能被声明为具有新类型的结果或参数的。因为必须在创建新类型之前创建这两个函数。这个问题的答案是，新类型应该首先被定义为一种shell类型，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I/O函数可以被定义为引用这种shell类型。最后，用带有完整定义的`CREATE TYPE`把该shell类型替换为一个完全的、合法的类型定义，之后新类型就可以正常使用了。

`like_type`参数提供了一种用于指定数据类型的基本表示形式属性的替代方法：从某些现有类型中复制它们。从指定类型复制值`internallength`，`passedbyvalue`，`alignment`和`storage`。（尽管通常不希望这样做，但可以通过与LIKE子句一起指定它们来覆盖其中的某些值。）以这种方式指定表示形式对于在现有类型中新类型“搭载”的低级实现在某种方式上非常有用。

虽然新类型的内部表示的详细信息仅由I/O函数和您创建的与该类型一起使用的其他函数才知道，但是内部表示的一些属性必须声明给MPP数据库。其中最重要的是`internallength`。基础数据类型可以是固定长度，在这种情况下，`internallength`是正整数），也可以是可变长度，通过将`internallength`设置为VARIABLE表示）。（内部，这是通过将`typlen`设置为-1来表示的。）所有可变长度类型的内部表示必须以4字节整数开头，并给出该类型值的总长度。

可选的标志PASSEDBYVALUE表示这种数据类型的值需要被传值而不是传引用。您不能传递内部表示形式大于Datum类型的大小的值类型（大多数计算机上为4字节，少数计算机上为8字节）。

*alignment*参数指定数据类型的存储对齐要求。允许的值等同于以1、2、4或8字节边界对齐。注意变长类型的参数必须至少按4字节对齐，因为它们需要包含一个int4作为它们的第一个组成部分。

*storage*参数允许为变长数据类型选择存储策略（对定长类型只允许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。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，这种存储策略的数据项会被优先考虑保留在主表中）。

如果用户希望数据类型的列默认为空值以外的其他值，则可以指定一个默认值。使用DEFAULT关键字指定默认值。（这种默认值可能会被附加到特定列的显式DEFAULT子句覆盖。）

要指示类型是数组，请使用ELEMENT关键字指定数组元素的类型。例如，要定义一个4字节整数（int4）的数组，请指定ELEMENT = int4。有关数组类型的更多详细信息显示在下面。

*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。每种数据类型均属于以单个ASCII字符命名的类别，并且每种类型均为“首选”或不在其类别内。当此规则有助于解析重载的函数或运算符时，解析器将更喜欢强制转换为首选类型（但只能从同一类别中的其他类型）。对于没有隐式转换为其他类型或从任何其他类型隐式转换的类型，保留默认设置就足够了。但是，对于具有隐式强制转换的一组相关类型，将它们全部标记为属于一个类别并选择一个或两个“最一般”类型作为该类别中的首选类型通常是有帮助的。当您添加用户定义的类型（例如数字或字符串类型）添加到现有内置类别时，*category*参数特别有用。也可以创建新的完全由用户定义的类型类别。选择除大写字母以外的任何ASCII字符以命名此类。

为了指示在此类型的数组的外部表示形式中的值之间使用定界符，可以将定界符设置为特定字符。默认的分隔符是逗号（，）。注意，分隔符与数组元素类型相关联，而不与数组类型本身相关联。

如果可选的布尔参数`collatable`为`true`，则类型的列定义和表达式可以通过使用`COLLATE`子句来携带排序规则信息。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。

数组类型

每当创建用户定义的类型时，MPP数据库都会自动创建一个关联的数组类型，其名称由元素类型的名称组成，该名称前面带有下划线，并在必要时将其截断以使其长度小于`NAMEDATALEN`字节。（如果这样生成的名称与现有类型名称冲突，则重复该过程，直到找到一个非冲突名称为止。）此隐式创建的数组类型为可变长度，并使用内置的输入和输出函数`array_in`和`array_out`。数组类型跟踪其元素类型的所有者或模式中的所有更改，并且在元素类型被删除时也被删除。

如果系统会自动地创建正确的数组类型，你可能会理所当然地问为什么会有一个`ELEMENT`选项。使用`ELEMENT`唯一有用的情况是：当你在创建一种定长类型，它正好在内部是一个多个相同东西的数组，并且除了计划给该类型提供的整体操作之外，你想要允许用下标来直接访问这些东西。例如，类型`point`被表示为两个浮点数，可以使用`point[0]`以及`point[1]`来访问它们。注意，这种功能只适用于内部形式正好是一个相同定长域序列的定长类型。可用下标访问的变长类型必须具有`array_in`以及`array_out`使用的一般化的内部表达。由于历史原因（即很明显是错的，但现在改已经太晚了），定长数组类型的下标是从零开始的，而不是像变长数组那样。

参数

name

要创建的类型的名称（可以由模式指定）。

attribute_name

复合类型的属性（列）的名称。

data_type

要成为复合类型的列的现有数据类型的名称。

collation

与复合类型的列或范围类型关联的现有排序规则的名称。

label

字符串文字，表示与枚举类型的一个值相关联的文本标签。

subtype

范围类型将代表其范围的元素类型的名称。

subtype_operator_class

子类型的b树运算符类的名称。

canonical_function

范围类型的规范化函数的名称。

subtype_diff_function

子类型的差函数的名称。

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的名称。

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。

receive_function

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。

send_function

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。

type_modifier_input_function

将类型的修饰符数组转换为内部形式的函数名。

type_modifier_output_function

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。

internallength

一个数字常量，指定新类型的内部表示形式的长度（以字节为单位）。默认假设是可变长度。

alignment

该数据类型的存储对齐需求。如果被指定，它必须是char、int2、int4或者double。默认是int4。

storage

该数据类型的存储策略。必须是plain、external、extended或者main。

默认是plain。

like_type

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。

category

此类型的类别代码（单个ASCII字符）。默认值为'U'，表示用户定义的类型。您可以在[pg_type类别代码](#)中找到其他标准类别代码。您还可以将未使用的ASCII字符分配给您创建的自定义类别。

preferred

如果此类型是其类型类别中的首选类型，则为true，否则为false。默认值为false。在现有类型类别中创建新的首选类型时要小心；这可能会导致令人惊讶的行为更改。

default

数据类型的默认值。如果省略，则默认为null。

element

创建的类型是一个数组；这指定了数组元素的类型。

delimiter

在此类型的数组中的值之间使用定界符。

collatable

如果此类型的操作可以使用排序规则信息，则为true。默认为false。

compression_type

设置为ZLIB（默认值），ZSTD，RLE_TYPE或QUICKLZ¹以指定在此类型的列中使用的压缩类型。

compression_level

对于Zstd压缩，将其设置为1（最快压缩）到19（最高压缩率）之间的整数值。对于zlib压缩，有效范围是1到9。QuickLZ压缩级别只能设置为1。对于RLE_TYPE，压缩级别可以设置为从1（最快压缩）到4（最高压缩率）的整数值。默认压缩级别为1。

blocksize

设置为列中每个块的大小（以字节为单位）。BLOCKSIZE必须介于8192和2097152字节之间，并且是8192的倍数。默认块大小为32768。

注解

用户定义的类型名称不能以下划线字符（_）开头，并且只能为62个字符长（或者通常为NAMEDATALEN - 2，而不是其他名称允许的NAMEDATALEN - 1个字符）。以下划线开头的类型名称保留给内部创建的数组类型名称。

MPP数据库不支持为行或复合类型添加存储选项。

在表级和列级定义的存储选项将覆盖为标量类型定义的默认存储选项。

因为一旦创建数据类型就没有使用限制，所以创建基础类型或范围类型无异于授予对类型定义中提到的函数的公共执行权限。（因此，类型的创建者必须拥有这些函数。）对于在类型定义中有用的各种函数而言，这通常不是问题。但是您在设计一种在将其转换为外部格式或从外部格式转换要求使用“秘密”信息的类型时，可能需要三思而后行。

示例

本示例创建一个复合类型并将其用于函数定义中：

```
CREATE TYPE compfoo AS (f1 int, f2 text);

CREATE FUNCTION getfoo() RETURNS SETOF compfoo AS $$
    SELECT fooid, foename FROM foo
$$ LANGUAGE SQL;
```

本示例创建枚举类型mood，并在表定义中使用它。

```
CREATE TYPE mood AS ENUM ('sad', 'ok', 'happy');
CREATE TABLE person (
    name text,
    current_mood mood
);
INSERT INTO person VALUES ('Moe', 'happy');
SELECT * FROM person WHERE current_mood = 'happy';
name | current_mood
-----+-----
Moe  | happy
(1 row)
```

本示例创建一个范围类型：

```
CREATE TYPE float8_range AS RANGE (subtype = float8, subtype_diff = float8mi);
```

本示例创建基础数据类型box，然后在表定义中使用该类型：

```
CREATE TYPE box;

CREATE FUNCTION my_box_in_function(cstring) RETURNS box AS
... ;

CREATE FUNCTION my_box_out_function(box) RETURNS cstring AS
... ;

CREATE TYPE box (
    INTERNALLENGTH = 16,
    INPUT = my_box_in_function,
    OUTPUT = my_box_out_function
);

CREATE TABLE myboxes (
    id integer,
    description box
);
```

如果box的内部结构是四个float4元素的数组，则可以改用：

```
CREATE TYPE box (  
    INTERNALLENGTH = 16,  
    INPUT = my_box_in_function,  
    OUTPUT = my_box_out_function,  
    ELEMENT = float4  
);
```

这将允许通过下标访问box值的组件号。 否则，该类型的行为与以前相同。

本示例创建一个大型对象类型，并在表定义中使用它：

```
CREATE TYPE bigobj (  
    INPUT = lo_filein, OUTPUT = lo_fileout,  
    INTERNALLENGTH = VARIABLE  
);  
  
CREATE TABLE big_objs (  
    id integer,  
    obj bigobj  
);
```

兼容性

创建复合类型的CREATE TYPE命令的第一种形式符合SQL标准。 其他形式是MPP数据库扩展。 SQL标准中的CREATE TYPE语句还定义了MPP数据库中未实现的其他形式。

创建具有零属性的复合类型的能力是MPP数据库特定于标准的偏差（类似于CREATE TABLE中的相同情况）。

另见

[ALTER TYPE](#) , [CREATE DOMAIN](#) , [CREATE FUNCTION](#) , [DROP TYPE](#)

Parent topic: [SQL Command Reference](#)

CREATE USER

默认情况下，使用LOGIN特权定义一个新的数据库角色。

概要

```
CREATE USER name [[WITH] option [ ... ]]
```

其中`option`可以是：

```

    SUPERUSER | NOSUPERUSER
| CREATEDB | NOCREATEDB
| CREATEROLE | NOCREATEROLE
| CREATEUSER | NOCREATEUSER
| CREATEEXTTABLE | NOCREATEEXTTABLE
[ ( attribute='value'[ , ... ] ) ]
    where attributes and value are:
        type='readable'|'writable'
        protocol='gpfdist'|'http'
| INHERIT | NOINHERIT
| LOGIN | NOLOGIN
| REPLICATION | NOREPLICATION
| CONNECTION LIMIT connlimit
| [ ENCRYPTED | UNENCRYPTED ] PASSWORD 'password'
| VALID UNTIL 'timestamp'
| IN ROLE role_name [ , ... ]
| IN GROUP role_name
| ROLE role_name [ , ... ]
| ADMIN role_name [ , ... ]
| USER role_name [ , ... ]
| SYSID uid
| RESOURCE QUEUE queue_name
| RESOURCE GROUP group_name
| [ DENY deny_point ]
| [ DENY BETWEEN deny_point AND deny_point ]

```

描述

CREATE USER是CREATE ROLE的别名。

CREATE ROLE和CREATE USER之间的唯一区别是CREATE USER默认情况下假定LOGIN，而CREATE ROLE默认情况下假定NOLOGIN。

兼容性

SQL标准中没有CREATE USER语句。

另见

[CREATE ROLE](#)

Parent topic: [SQL Command Reference](#)

CREATE USER MAPPING

定义用户到外部服务器的新映射。

概要

```
CREATE USER MAPPING FOR { username | USER | CURRENT_USER | PUBLIC }  
    SERVER servername  
    [ OPTIONS ( option 'value' [, ... ] ) ]
```

描述

CREATE USER MAPPING定义用户到外部服务器的映射。 您必须是服务器的所有者才能为其定义用户映射。

参数

username

映射到外部服务器的现有用户的名称。 CURRENT_USER和USER与当前用户的名称匹配。 PUBLIC用于匹配系统中所有当前和将来的用户名。

servername

MPP数据库要为其创建用户映射的现有服务器的名称。

OPTIONS (*option 'value' [, ...]*)

新用户映射的选项。 这些选项通常定义映射的实际用户名和密码。 选项名称必须唯一。 选项名称和值特定于服务器的外部数据包装器。

示例

为用户bob和服务器foo创建用户映射：

```
CREATE USER MAPPING FOR bob SERVER foo OPTIONS (user 'bob', password 'secret');
```

兼容性

CREATE USER MAPPING符合ISO/IEC 9075-9（SQL/MED）。

另见

[ALTER USER MAPPING](#), [DROP USER MAPPING](#), [CREATE FOREIGN](#)

DATA WRAPPER, CREATE SERVER

Parent topic: [SQL Command Reference](#)

CREATE VIEW

定义一个新的视图。

概要

```
CREATE [OR REPLACE] [TEMP | TEMPORARY] [RECURSIVE] VIEW name
    [ ( column_name [, ...] ) ]
    [ WITH ( view_option_name [= view_option_value] [, ... ] ) ]
    AS query
    [ WITH [ CASCADED | LOCAL ] CHECK OPTION ]
```

描述

CREATE VIEW定义查询的视图。该视图未物化。而是，每次在查询中引用视图时运行查询。

CREATE OR REPLACE VIEW类似，但是如果已经存在相同名称的视图，则将其替换。新查询必须生成与现有视图查询生成列相同的列（即，相同的列名以相同的顺序，并且具有相同的数据类型），但是它可能会将其他列添加到列表的末尾。产生输出列的计算可能完全不同。

如果指定了模式名称，则将在指定的模式中创建视图。否则，它将在当前模式中创建。临时视图存在于特殊的模式中，因此在创建临时视图时可能不会给出模式名称。视图的名称必须与同一模式中的任何其他视图，表，序列，索引或外部表的名称不同。

参数

TEMPORARY | TEMP

如果指定，则将视图创建为临时视图。临时视图会在当前会话结束时自动删除。存在临时视图时，当前会话将看不到具有相同名称的已有永久表，除非使用模式限定名称来引用它们。如果该视图引用的任何表都是临时表，则该视图将被创建为临时视图（无论是否指定TEMPORARY）。

RECURSIVE

创建一个递归视图。语法

```
CREATE RECURSIVE VIEW [ schema . ] view_name (column_names) AS SELECT ...
```

等价于

```
CREATE VIEW [ schema . ] view_name AS WITH RECURSIVE view_name (column_n
```

必须为递归视图指定视图列名称列表。

name

要创建的视图的名称（可以由模式指定）。

column_name

用于视图列的名称的可选列表。如果未给出，则从查询中推导出列名。

WITH (*view_option_name* [= *view_option_value*] [, ...])

此子句指定视图的可选参数；支持以下参数：

check_option (string)

该参数可以是local，也可以是cascaded，等效于指定WITH [

CASCADDED | LOCAL] CHECK OPTION（请参阅下文）。可以使用ALTER VIEW在现有视图上更改此选项。

security_barrier (boolean)

如果视图旨在提供行级安全性，则应使用此方法。

query

SELECT或VALUES命令，将提供视图的列和行。

注解

MPP数据库中的视图为只读。系统不允许在视图上插入，更新或删除。通过将视图上的重写规则创建为其他表上的适当操作，可以得到可更新视图的效果。有关更多信息，请参见CREATE RULE。

请注意，视图列的名称和数据类型将按照您想要的方式分配。例如：

```
CREATE VIEW vista AS SELECT 'Hello World';
```

是两种形式的错误形式：列名默认为?column?，列数据类型默认为unknown。如果要在视图结果中使用字符串文字，请使用类似以下内容的方法：

```
CREATE VIEW vista AS SELECT text 'Hello World' AS hello;
```

对视图中引用的表的访问权限是由视图所有者而不是当前用户（即使当前用户是超级用户）的权限决定的。对于超级用户，这可能会造成混淆，因为超级用户通常可以访问所有对象。在视图的情况下，如果超级用户不是视图的所有者，那么即使超级用户也必须被明确授予访问该视图中引用的表的权限。

但是，对视图中调用的函数的处理方式与使用视图直接从查询中直接调用它们的方式相同。因此，视图的用户必须具有调用该视图使用的任何函数的权限。

如果使用ORDER BY子句创建视图，则从视图执行SELECT时将忽略ORDER BY子句。

在现有视图上使用CREATE OR REPLACE VIEW时，仅更改视图定义的SELECT规则。其他视图属性（包括所有权，权限和non-SELECT规则）保持不变。您必须拥有视图才能替换它（包括作为拥有角色的成员）。

示例

创建一个由所有喜剧电影组成的视图：

```
CREATE VIEW comedies AS SELECT * FROM films
WHERE kind = 'comedy';
```

这将创建一个视图，其中包含在创建视图时film表中的列。尽管使用*来创建视图，但是稍后添加到表中的列将不属于视图。

创建一个获取前十名婴儿名字的视图：

```
CREATE VIEW topten AS SELECT name, rank, gender, year FROM
names, rank WHERE rank < '11' AND names.id=rank.id;
```

创建一个由1到100的数字组成的递归视图：

```
CREATE RECURSIVE VIEW public.nums_1_100 (n) AS
VALUES (1)
UNION ALL
SELECT n+1 FROM nums_1_100 WHERE n < 100;
```

请注意，尽管此CREATE VIEW命令中的递归视图名称是模式限定的，但是其内部自引用不是模式限定的。这是因为隐式创建的CTE名称不能通过模式限定。

兼容性

SQL标准为CREATE VIEW语句指定了MPP数据库中没有的一些附加功能。标准中完整SQL命令的可选子句为：

- **CHECK OPTION** — 此选项与可更新视图有关。 将检查视图上的所有INSERT和UPDATE命令， 以确保数据满足视图定义条件（即，新数据将通过视图可见）。 如果不这样做，更新将被拒绝。
- **LOCAL** — 在此视图上检查完整性。
- **CASCADE** — 在此视图和任何从属视图上检查完整性。 如果未指定CASCADE或LOCAL，则假定为CASCADE。

CREATE OR REPLACE VIEW是MPP数据库语言的扩展。临时视图的概念也是如此。

另见

[SELECT , DROP VIEW](#)

Parent topic: [SQL Command Reference](#)

DEALLOCATE

取消分配预编译语句。

概要

DEALLOCATE [PREPARE] *name*

描述

DEALLOCATE用于取消分配先前预编译的SQL语句。 如果未显式取消分配预编译语句，则在会话结束时将其释放。

有关预编译语句的更多信息，请参见[PREPARE](#)。

参数

PREPARE

 可选关键字，将被忽略。

name

 要取消分配的预编译语句的名称。

示例

取消分配先前预编译的名为insert_names的语句：

DEALLOCATE insert_names;

兼容性

SQL标准包括DEALLOCATE语句，但仅用于嵌入式SQL。

另见

[EXECUTE , PREPARE](#)

Parent topic: [SQL Command Reference](#)

DECLARE

定义一个游标。

概要

```
DECLARE name [BINARY] [INSENSITIVE] [NO SCROLL] CURSOR
      [{WITH | WITHOUT} HOLD]
      FOR query [FOR READ ONLY]
```

描述

DECLARE允许用户创建游标，该游标可用于一次从较大查询中检索少量行。游标可以使用[FETCH](#)以文本或二进制格式返回数据。

Note: 本页在SQL命令级别描述游标的用法。如果试图在PL/pgSQL函数中使用游标，则规则是不同的。参见[MPP PL/pgSQL过程语言](#)。

普通游标以文本格式返回数据，与SELECT会产生相同的结果。由于数据本身以二进制格式存储，因此系统必须进行转换以产生文本格式。一旦信息以文本形式返回，客户端应用程序可能需要将其转换为二进制格式以进行操作。另外，文本格式的数据通常比二进制格式的数据大。二进制游标以二进制表示形式返回数据，可能更易于操作。但是，如果您仍然打算将数据显示为文本，则以文本形式检索数据将节省您在客户端的工作量。

例如，如果查询从整数列返回值1，则将使用默认游标获得字符串1，而使用二进制游标将获得包含值的内部表示形式的4字节字段（在大端字节序）。

二进制游标应谨慎使用。许多应用程序，包括psql，都不准备处理二进制游标，并希望数据以文本格式返回。

Note:

当客户端应用程序使用“扩展查询”协议发出FETCH命令时，“绑定协议”消息指定是以文本还是二进制格式检索数据。此选择将覆盖定义光标的方式。因此，当使用扩展查询协议时，二进制游标的概念就已过时了 - 任何游标都可以视为文本或二进制。

可以在[UPDATE](#)或[DELETE](#)语句的WHERE CURRENT OF子句中指定游标，以更新或删除表数据。UPDATE或DELETE语句只能在服务器上执行，例如在交互式psql会话或脚本中。语言扩展（例如PL/pgSQL）不支持可更新的游标。

参数

name

要创建的游标的名称。

BINARY

使光标返回二进制而不是文本格式的数据。

INSENSITIVE

指示在存在游标时，从游标检索的数据应不受游标基于的表更新的影响。在MPP数据库中，所有游标都不敏感。该关键字当前不起作用，出现是为了与SQL标准兼容。

NO SCROLL

游标不能用于以非顺序方式检索行。这是MPP数据库中的默认行为，因为不支持滚动游标（SCROLL）。

WITH HOLD

WITHOUT HOLD

WITH HOLD指定在成功创建游标的事务提交后可以继续使用游标。

WITHOUT HOLD指定不能在创建游标的事务之外使用游标。默认为WITHOUT HOLD。

当query包含FOR UPDATE或FOR SHARE子句时，不能指定WITH HOLD。

query

[SELECT](#)或[VALUES](#)命令，它将提供游标要返回的行。

如果在[UPDATE](#)或[DELETE](#)命令的WHERE CURRENT OF子句中使用了游标，则SELECT命令必须满足以下条件：

- 无法引用视图或外部表。
- 仅引用一张表。
该表必须是可更新的。例如，以下内容不可更新：表函数，返回集合的函数，仅追加表，列式表。
- 不能包含以下任何内容：
 - 分组子句
 - 集合操作，例如UNION ALL或UNION DISTINCT
 - 排序子句
 - 窗口子句
 - 连接或自连接

在SELECT命令中指定FOR UPDATE子句可防止其他会话在获取行和

更新行之间更改行。如果没有FOR UPDATE子句，则在创建游标以后更改了行，随后将UPDATE或DELETE命令与WHERE CURRENT OF子句一起使用将无效。

Note: 在SELECT命令中指定FOR UPDATE子句将锁定整个表，而不只是选定的行。

FOR READ ONLY

FOR READ ONLY表示光标以只读模式使用。

注解

除非指定了WITH HOLD，否则此命令创建的游标只能在当前事务中使用。因此，不带WITH HOLD的DECLARE在事务块之外是无用的：游标只能生存到语句完成。因此，如果在事务块外部使用此命令，MPP数据库将报告错误。使用BEGIN和COMMIT（或ROLLBACK）定义事务块。

如果指定了WITH HOLD，并且成功提交了创建游标的事务，则该游标可以继续被同一会话中的后续事务访问。（但是，如果创建事务被中止，则删除游标。）当对它发出显式CLOSE命令或会话结束时，将关闭用WITH HOLD创建的游标。在当前实现中，由保留的游标表示的行被复制到临时文件或存储区中，以便它们可用于后续事务。

如果在事务中使用DECLARE命令创建游标，则只有在使用CLOSE命令关闭游标后才能在事务中使用SET命令。

MPP数据库当前不支持可滚动光标。您只能使用FETCH将光标位置向前移动，而不能向后移动。

追加优化表不支持DECLARE...FOR UPDATE。

您可以通过查询pg_cursors系统视图来查看所有可用的游标。

示例

声明一个游标：

```
DECLARE mycursor CURSOR FOR SELECT * FROM mytable;
```

兼容性

SQL标准仅允许在嵌入式SQL和模块中使用游标。 MPP数据库允许以交互方式使用游标。

MPP数据库未为游标实现OPEN语句。 当游标被声明时，它被认为是打开的。

SQL标准允许游标向前和向后移动。 所有MPP数据库游标仅向前移动（不可滚动）。

二进制游标是MPP数据库扩展。

另见

[CLOSE](#), [DELETE](#), [FETCH](#), [MOVE](#), [SELECT](#), [UPDATE](#)

Parent topic: [SQL Command Reference](#)

DELETE

从表中删除行。

概要

```
[ WITH [ RECURSIVE ] with_query [, ...] ]  
DELETE FROM [ONLY] table [[AS] alias]  
    [USING usinglist]  
    [WHERE condition | WHERE CURRENT OF cursor_name]  
    [RETURNING * | output_expression [[AS] output_name] [, ...]]
```

描述

DELETE从指定的表中删除满足WHERE子句的行。如果WHERE子句不存在，则结果是删除表中的所有行。结果是一个有效但空的表。

默认情况下，DELETE将删除指定表及其所有子表中的行。如果只希望从提到的特定表中删除，则必须使用ONLY子句。

使用数据库中其他表中包含的信息删除表中的行有两种方法：使用子选择，或在USING子句中指定其他表。哪种技术更合适取决于具体情况。

如果指定了WHERE CURRENT OF子句，则删除的行是从指定游标中最新获取的行。

复制表不支持WHERE CURRENT OF子句。

可选的RETURNING子句使DELETE根据实际删除的每一行计算并返回值。可以计算使用该表的列和/或USING中提到的其他表的列的任何表达式。RETURNING列表的语法与SELECT的输出列表的语法相同。

您必须对表具有DELETE特权才能从中删除。

Note: 默认情况下，MPP数据库为堆表上的DELETE操作获取表上的EXCLUSIVE锁。启用全局死锁检测器后，堆表上DELETE操作的锁定模式为ROW EXCLUSIVE。请参阅[全局死锁检测](#)。

输出

成功完成后，DELETE命令将返回以下形式的命令标签：

```
DELETE count
```

*count*是已删除的行数。如果*count*为0，则查询没有删除任何行（这不视为错误）。

如果DELETE命令包含RETURNING子句，则结果将类似于SELECT语句的结果，该SELECT语句包含RETURNING列表中定义的列和值，该列和值是在该命令删除的行上计算的。

参数

with_query

WITH子句允许您指定一个或多个子查询，这些子查询可以在DELETE查询中按名称引用。
对于包含WITH子句的DELETE命令，该子句只能包含SELECT语句，而WITH子句不能包含数据修改命令（INSERT，UPDATE或DELETE）。
请参见[WITH查询（公用表表达式）](#)和[SELECT](#)获取详细信息。

ONLY

如果指定，则仅从命名表中删除行。如果未指定，还将处理从命名表继承的任何表。

table

现有表的名称（可以用模式指定）。

alias

目标表的替代名称。提供别名后，它将完全隐藏表的实际名称。例如，给定DELETE FROM foo AS f，DELETE语句的其余部分必须将此表称为f而不是foo。

usinglist

表表达式的列表，允许其他表中的列以WHERE条件出现。这类似于可以在SELECT语句的FROM子句中指定的表的列表。例如，可以指定表名的别名。除非您希望设置自连接，否则不要在usinglist中重复目标表。

condition

该表达式返回一个布尔类型的值，该值确定要删除的行。

cursor_name

在WHERE CURRENT OF条件中使用的游标名称。要删除的行是从该游标中最近获取的行。游标必须是对DELETE目标表的简单非分组查询。

不能与布尔条件一起指定WHERE CURRENT OF。

游标语句的DELETE...WHERE CURRENT OF游标语句只能在服务器上执行，例如在交互式psql会话或脚本中。语言扩展（例如PL/pgSQL）不支持可更新的游标。

有关创建游标的更多信息，请参见[DECLARE](#)。

output_expression

删除每一行后，由DELETE命令计算并返回的表达式。该表达式可以使用在USING中列出的一个或多个表的任何列名。输入*以返回所有列。

output_name

用于返回的列的名称。

注解

MPP数据库允许您通过在USING子句中指定其他表来引用WHERE条件下其他表的列。例如，从rank表中将其命名为Hannah，可以这样做：

```
DELETE FROM rank USING names WHERE names.id = rank.id AND
name = 'Hannah';
```

这里实际上发生的是rank和names之间的联接，所有成功联接的行都标记为删除。此语法不是标准语法。但是，这种连接样式通常比更标准的子选择样式更易于编写和执行，例如：

```
DELETE FROM rank WHERE id IN (SELECT id FROM names WHERE name  
= 'Hannah');
```

当使用DELETE删除表的所有行时（例如：DELETE * FROM *table*;），MPP数据库会添加一个隐式的TRUNCATE命令（当用户权限允许时）。添加的TRUNCATE命令释放被删除的行占用的磁盘空间，而无需对表进行VACUUM。这提高了后续查询的扫描性能，并使经常在临时表中插入和删除的ELT工作负载受益。

不支持在分区表的特定分区（子表）上直接执行UPDATE和DELETE命令。而是必须在根分区表（使用CREATE TABLE命令创建的表）上执行这些命令。

示例

删除除musicals以外的所有films：

```
DELETE FROM films WHERE kind <> 'Musical';
```

清除表films：

```
DELETE FROM films;
```

删除已完成的任务，返回已删除行的完整详细信息：

```
DELETE FROM tasks WHERE status = 'DONE' RETURNING *;
```

使用联接删除：

```
DELETE FROM rank USING names WHERE names.id = rank.id AND
name = 'Hannah';
```

兼容性

该命令符合SQL标准，不同之处在于USING和RETURNING子句是MPP数据库扩展， 以及将WITH和DELETE一起使用的功能。

另见

[DECLARE](#), [TRUNCATE](#)

Parent topic: [SQL Command Reference](#)

DISCARD

放弃会话状态。

概要

```
DISCARD { ALL | PLANS | TEMPORARY | TEMP }
```

描述

DISCARD释放与数据库会话关联的内部资源。此命令对于部分或全部重置会话状态很有用。有几个子命令可以释放不同类型的资源。DISCARD ALL变体包含所有其他变体，并且还重置其他状态。

参数

PLANS

释放所有缓存的查询计划，并在下一次使用关联的预编译语句时强制进行重新计划。

SEQUENCES

丢弃所有与缓存序列相关的状态，包括尚未由nextval()返回的任何预分配序列值。（有关预分配序列值的描述，请参见CREATE SEQUENCE。）

TEMPORARY/TEMP

删除在当前会话中创建的所有临时表。

ALL

释放与当前会话关联的所有临时资源，并将会话重置为其初始状态。当前，这与执行以下语句序列具有相同的效果：

```
SET SESSION AUTHORIZATION DEFAULT;  
RESET ALL;  
DEALLOCATE ALL;  
CLOSE ALL;  
UNLISTEN *;  
SELECT pg_advisory_unlock_all();  
DISCARD PLANS;  
DISCARD SEQUENCES;  
DISCARD TEMP;
```

注解

DISCARD ALL不能在事务块内执行。

兼容性

DISCARD是MPP数据库扩展。

Parent topic: [SQL Command Reference](#)

DO

执行匿名代码块作为临时匿名函数。

概要

```
DO [ LANGUAGE lang_name ] code
```

描述

DO执行过程代码中的一个匿名代码块，或一个临时的匿名函数。

将代码块视为没有参数的函数的主体，并返回void。 它被解析并执行一次。

可选的LANGUAGE子句可以出现在代码块之前或之后。

匿名块是过程语言结构，它提供了动态创建和执行过程代码的能力，而无需将代码作为数据库对象持久存储在系统目录中。 匿名块的概念类似于UNIX Shell脚本，它使几个手动输入的命令可以被分组并作为一个步骤执行。 顾名思义，匿名块没有名称，因此，不能从其他对象中引用它们。 尽管匿名块是动态构建的，但可以轻松地将它们作为脚本存储在操作系统文件中以重复执行。

匿名块是标准的过程语言块。 它们带有语法并遵守适用于过程语言的规则，包括变量的声明和范围，执行，异常处理和语言使用。

匿名块的编译和执行在一个步骤中组合在一起，而每次定义更改时，都必须在使用前重新定义用户定义的函数。

参数

code

要执行的过程语言代码。 必须将其指定为字符串文字，就像使用CREATE FUNCTION命令一样。 建议使用美元符号将代码括起来。 可选关键字无效。 支持以下过程语言：PL/pgSQL (plpgsql) ，

PL/Python (plpythonu) 和PL/Perl (plperl和plperlu) 。

lang_name

编写代码的过程语言的名称。默认值为plpgsql。该语言必须安装在MPP数据库系统上并在数据库中注册。

注解

PL/pgSQL语言已安装在MPP数据库系统上，并已在用户创建的数据库中注册。PL/Python和PL/Perl语言是默认安装的，但未注册。未安装或注册其他语言。系统目录pg_language包含有关数据库中已注册语言的信息。

用户必须对过程语言具有USAGE特权，或者如果该语言不受信任，则必须是超级用户。这与使用该语言创建函数需要的特权相同。

匿名块不支持函数易变性或EXECUTE ON属性。

示例

此PL/pgSQL示例向角色webuser授予对模式public的所有视图的所有特权：

```
DO $$DECLARE r record;
BEGIN
  FOR r IN SELECT table_schema, table_name FROM information_schema.tables
    WHERE table_type = 'VIEW' AND table_schema = 'public'
  LOOP
    EXECUTE 'GRANT ALL ON ' || quote_ident(r.table_schema) || '.' || quote_ident(r.table_name) || ' TO webuser;';
  END LOOP;
END$$;
```

该PL/pgSQL示例确定MPP数据库用户是否为超级用户。在该示例中，匿名块从临时表中检索输入值。

```
CREATE TEMP TABLE list AS VALUES ('gpadmin') DISTRIBUTED RANDOMLY;

DO $$
DECLARE
    name TEXT := 'gpadmin' ;
    superuser TEXT := " ;
    t1_row pg_authid%ROWTYPE;
BEGIN
    SELECT * INTO t1_row FROM pg_authid, list
        WHERE pg_authid.rolname = name ;
    IF t1_row.rolsuper = 'f' THEN
        superuser := 'not ' ;
    END IF ;
    RAISE NOTICE 'user % is %a superuser', t1_row.rolname, superuser ;
END $$ LANGUAGE plpgsql ;
```

Note: 示例PL/pgSQL将SELECT与INTO子句一起使用。它与SQL命令SELECT INTO不同。

兼容性

SQL标准中没有DO语句。

另见

CREATE LANGUAGE [MPP PL/pgSQL过程语言](#)

Parent topic: [SQL Command Reference](#)

DROP AGGREGATE

删除聚合函数。

概要

```
DROP AGGREGATE [IF EXISTS] name ( type [, ...] ) [CASCADE | RESTRICT]
```

描述

DROP AGGREGATE将删除现有的聚合函数。 要执行此命令，当前用户必须是聚合函数的所有者。

参数

IF EXISTS

如果聚合不存在，请不要引发错误。在这种情况下会发出通知。

name

现有聚合函数的名称（可以由模式限定）。

type

聚合函数所基于的输入数据类型。 要引用零参数聚合函数，请写*代替输入数据类型列表。

CASCADE

自动删除依赖于聚合函数的对象。

RESTRICT

如果有任何对象依赖聚合函数，则拒绝删除该函数。这是默认值。

示例

要删除整数类型的聚合函数myavg：

```
DROP AGGREGATE myavg(integer);
```

兼容性

SQL标准中没有DROP AGGREGATE语句。

另见

[ALTER AGGREGATE](#)，[CREATE AGGREGATE](#)

Parent topic: [SQL Command Reference](#)

DROP CAST

删除一个造型。

概要

```
DROP CAST [IF EXISTS] (sourcetype AS targettype) [CASCADE | RESTRICT]
```

描述

DROP CAST将删除先前定义的造型。为了能够删除造型，您必须拥有源或目标数据类型。这些与创建造型所需的特权相同。

参数

IF EXISTS

如果造型不存在，请不要抛出错误。在这种情况下会发出通知。

sourcetype

造型的源数据类型的名称。

targettype

造型的目标数据类型的名称。

CASCADE

RESTRICT

这些关键字没有任何作用，因为不依赖造型。

示例

删除从text类型到int类型的造型：

```
DROP CAST (text AS int);
```

兼容性

DROP CAST命令符合SQL标准。

另见

[CREATE CAST](#)

Parent topic: [SQL Command Reference](#)

DROP COLLATION

删除以前定义的排序规则。

概要

```
DROP COLLATION [ IF EXISTS ] name [ CASCADE | RESTRICT ]
```

参数

IF EXISTS

如果排序规则不存在，请不要抛出错误。 在这种情况下会发出通知。

name

排序规则的名称。排序规则名称可以是模式限定的。

CASCADE

自动删除依赖于排序规则的对象。

RESTRICT

如果有任何对象依赖该排序规则，则拒绝删除该排序规则。这是默认值。

注解

DROP COLLATION删除以前定义的排序规则。 要删除排序规则，您必须拥有该排序规则。

示例

删除german的排序规则：

```
DROP COLLATION german;
```

兼容性

除了IF EXISTS选项（这是MPP数据库扩展名）之外，DROP COLLATION命令符合SQL标准。

另见

[ALTER COLLATION](#), [CREATE COLLATION](#)

Parent topic: [SQL Command Reference](#)

DROP CONVERSION

删除转换。

概要

```
DROP CONVERSION [IF EXISTS] name [CASCADE | RESTRICT]
```

描述

DROP CONVERSION删除以前定义的转换。 为了放弃转换，您必须拥有该转换。

参数

IF EXISTS

如果转换不存在，请不要抛出错误。在这种情况下会发出通知。

name

转换的名称。转换名称可以是模式限定的。

CASCADE
RESTRICT

这些关键字无效，因为它们不依赖转换。

示例

删除名为myname的转换：

```
DROP CONVERSION myname;
```

兼容性

SQL标准中没有DROP CONVERSION语句。该标准具有CREATE TRANSLATION和DROP TRANSLATION语句，与MPP数据库的CREATE CONVERSION和DROP CONVERSION语句相似。

另见

[ALTER CONVERSION](#) , [CREATE CONVERSION](#)

Parent topic: [SQL Command Reference](#)

DROP DATABASE

删除数据库。

概要

```
DROP DATABASE [IF EXISTS] name
```

描述

DROP DATABASE删除数据库。它删除数据库的catalog条目，并删除包含数据的目录。它只能由数据库所有者执行。另外，当您或其他任何人连接到目标数据库时，也无法执行该命令。（连接到postgres或任何其他数据库以发出此命令。）

Warning: DROP DATABASE无法撤消。小心使用！

参数

IF EXISTS

如果数据库不存在，请不要报错。在这种情况下会发出通知。

name

要删除的数据库的名称。

注解

无法在事务块内部执行DROP DATABASE。

连接到目标数据库时无法执行此命令。 因此，改为使用程序dropdb可能更方便，该程序是该命令的包装器。

示例

删除名为testdb的数据库：

```
DROP DATABASE testdb;
```

兼容性

SQL标准中没有DROP DATABASE语句。

另见

[ALTER DATABASE](#) , [CREATE DATABASE](#)

Parent topic: [SQL Command Reference](#)

DROP DOMAIN

删除域。

概要

```
DROP DOMAIN [IF EXISTS] name [, ...] [CASCADE | RESTRICT]
```

描述

DROP DOMAIN删除先前定义的域。 您必须是域的所有者才能删除它。

参数

IF EXISTS

如果域不存在，请不要报错。在这种情况下会发出通知。

name

现有域的名称（可以用模式指定）。

CASCADE

自动删除依赖域的对象（例如表列）。

RESTRICT

如果有任何对象依赖该域，则拒绝删除该域。这是默认值。

示例

删除名为zipcode的域：


```
DROP DOMAIN zipcode;
```

兼容性

此命令符合SQL标准，但IF EXISTS选项是MPP数据库扩展。

另见

[ALTER DOMAIN](#) , [CREATE DOMAIN](#)

Parent topic: [SQL Command Reference](#)

DROP EXTENSION

从MPP数据库中删除扩展。

概要

```
DROP EXTENSION [ IF EXISTS ] name [, ...] [ CASCADE | RESTRICT ]
```

描述

DROP EXTENSION从数据库中删除扩展。删除扩展会导致其组成对象也被删除。

Note: 创建扩展所需的支持扩展文件没有被删除。必须从MPP数据库主机中手动删除文件。

您必须拥有该扩展才能使用DROP EXTENSION。

如果数据库中正在使用任何扩展的对象，则此命令将失败。例如，如果用扩展类型的列定义了表。添加CASCADE选项以强制删除那些依赖对象。

Important: 在使用CASCADE关键字发出DROP EXTENSION之前，您应该了解所有依赖于扩展名的对象，以避免意外的结果。

参数

IF EXISTS

如果扩展名不存在，请不要报错。发出通知。

name

已安装扩展的名称。

CASCADE

自动删除依赖于扩展的对象，并删除依赖于这些对象的所有对象。请参阅有关[依赖跟踪](#)的PostgreSQL信息。

RESTRICT

如果扩展成员对象以外的任何对象依赖扩展，则拒绝删除该扩展。这是默

认值。

兼容性

DROP EXTENSION是MPP数据库扩展。

另见

[CREATE EXTENSION](#), [ALTER EXTENSION](#)

Parent topic: [SQL Command Reference](#)

DROP EXTERNAL TABLE

删除外部表定义。

概要

```
DROP EXTERNAL [WEB] TABLE [IF EXISTS] name [CASCADE | RESTRICT]
```

描述

DROP EXTERNAL TABLE从数据库系统中删除现有的外部表定义。外部数据源或文件不会被删除。要执行此命令，您必须是外部表的所有者。

参数

WEB

用于删除外部Web表的可选关键字。

IF EXISTS

如果外部表不存在，不要报错。在这种情况下会发出通知。

name

现有外部表的名称（可以由模式指定）。

CASCADE

自动删除依赖于外部表的对象（例如视图）。

RESTRICT

如果有任何对象依赖外部表，则拒绝删除它。这是默认值。

示例

删除名为staging的外部表（如果存在）：

```
DROP EXTERNAL TABLE IF EXISTS staging;
```

兼容性

SQL标准中没有DROP EXTERNAL TABLE语句。

另见

[CREATE EXTERNAL TABLE](#)

Parent topic: [SQL Command Reference](#)

DROP FOREIGN DATA WRAPPER

删除外部数据包装器。

概要

```
DROP FOREIGN DATA WRAPPER [ IF EXISTS ] name [ CASCADE | RESTRICT ]
```

描述

DROP FOREIGN DATA WRAPPER从当前数据库中删除现有的外部数据包装器。 外部数据包装程序只能由其所有者删除。

参数

IF EXISTS

如果外部数据包装器不存在，不要报错。在这种情况下，MPP数据库会发出通知。

name

现有外部数据包装器的名称。

CASCADE

自动删除依赖于外部数据包装器的对象（例如服务器）。

RESTRICT

如果有任何对象依赖于它，则拒绝删除外部数据包装器。这是默认值。

示例

删除名为dbi的外部数据包装器：

```
DROP FOREIGN DATA WRAPPER dbi;
```

兼容性

DROP FOREIGN DATA WRAPPER符合ISO/IEC 9075-9（SQL/MED）。IF EXISTS子句是MPP数据库扩展。

另见

[CREATE FOREIGN DATA WRAPPER](#), [ALTER FOREIGN DATA WRAPPER](#)

Parent topic: [SQL Command Reference](#)

DROP FOREIGN TABLE

删除外部表。

概要

```
DROP FOREIGN TABLE [ IF EXISTS ] name [, ...] [ CASCADE | RESTRICT ]
```

描述

DROP FOREIGN TABLE删除现有的外部表。 只有外部表的所有者才能将其删除。

参数

IF EXISTS

如果外部表不存在，不要报错。在这种情况下，MPP数据库会发出通知。

name

要删除的外部表的名称（可以由模式指定）。

CASCADE

自动删除依赖于外部表的对象（例如视图）。

RESTRICT

如果有任何对象依赖外部表，则拒绝删除该表。这是默认值。

示例

删除名为films和distributors的外部表：

```
DROP FOREIGN TABLE films, distributors;
```

兼容性

DROP FOREIGN TABLE符合ISO/IEC 9075-9（SQL/MED），但标准仅允许每个命令删除一个外部表。IF EXISTS子句是MPP数据库扩展。

另见

[ALTER FOREIGN TABLE](#), [CREATE FOREIGN TABLE](#)

Parent topic: [SQL Command Reference](#)

DROP FUNCTION

删除函数。

概要

```
DROP FUNCTION [IF EXISTS] name ( [ [argmode] [argname] argtype  
[ , ... ] ] ) [CASCADE | RESTRICT]
```

描述

DROP FUNCTION删除现有函数的定义。 要执行此命令，用户必须是该函数的所有者。 必须指定函数的参数类型，因为可能存在多个具有相同名称和不同参数列表的不同函数。

参数

IF EXISTS

如果该函数不存在，请不要报错。在这种情况下会发出通知。

name

现有函数的名称（可以由模式指定）。

argmode

参数的模式：IN，OUT，INOUT或VARIADIC。 如果省略，则默认值为IN。 请注意，由于仅需要输入参数来确定函数的标识，因此DROP

FUNCTION实际上并不关注OUT参数。因此，列出IN，INOUT和VARIADIC参数就足够了。

argname

参数的名称。请注意，由于仅需要参数数据类型来确定函数的标识，因此DROP FUNCTION实际上并不关注参数名称。

argtype

函数参数的数据类型（可以由模式指定）（如果有）。

CASCADE

自动删除依赖于函数的对象，例如运算符。

RESTRICT

如果有任何对象依赖该函数，则拒绝删除该函数。这是默认值。

示例

删除平方根函数：

```
DROP FUNCTION sqrt(integer);
```

兼容性

SQL标准中定义了DROP FUNCTION语句，但该命令与此命令不兼容。

另见

[CREATE FUNCTION](#) , [ALTER FUNCTION](#)

Parent topic: [SQL Command Reference](#)

DROP GROUP

删除数据库角色。

概要

```
DROP GROUP [IF EXISTS] name [, ...]
```

描述

DROP GROUP是DROP ROLE的别名。 有关更多信息，请参见[DROP ROLE](#)。

兼容性

SQL标准中没有DROP GROUP语句。

另见

DROP ROLE

Parent topic: [SQL Command Reference](#)

DROP INDEX

删除索引。

概要

```
DROP INDEX [ CONCURRENTLY ] [ IF EXISTS ] name [, ...] [ CASCADE | RESTRICT ]
```

描述

DROP INDEX从数据库系统删除现有索引。 要执行此命令，您必须是索引的所有者。

参数

CONCURRENTLY

删除索引而不锁定索引表上的并发选择，插入，更新和删除。 普通的DROP INDEX在表上获取排他锁，从而阻止其他访问，直到可以完成索

引删除为止。使用此选项，命令将一直等到冲突的事务完成。使用此选项时需要注意几个注意事项。只能指定一个索引名称，并且不支持CASCADE选项。（因此，不能以这种方式删除支持UNIQUE或PRIMARY KEY约束的索引。）而且，可以在事务块内执行常规的DROP INDEX命令，但不能以DROP INDEX CONCURRENTLY方式执行。

IF EXISTS

如果该索引不存在，请不要报错。在这种情况下会发出通知。

name

现有索引的名称（可以由模式指定）。

CASCADE

自动删除依赖于索引的对象。

RESTRICT

如果有任何对象依赖该索引，则拒绝删除该索引。这是默认值。

示例

删除索引title_idx：

```
DROP INDEX title_idx;
```

兼容性

DROP INDEX是MPP数据库语言的扩展。SQL标准中没有索引的规定。

另见

DROP LANGUAGE

删除过程语言。

概要

```
DROP [PROCEDURAL] LANGUAGE [IF EXISTS] name [CASCADE | RESTRICT]
```

描述

DROP LANGUAGE将删除先前注册的过程语言的定义。 您必须是超级用户或该语言的所有者才能删除该语言。

参数

- PROCEDURAL
 可选关键字-无效。
- IF EXISTS

如果该语言不存在，请不要报错。在这种情况下会发出通知。

name

现有过程语言的名称。为了向后兼容，名称可以用单引号引起来。

CASCADE

自动删除依赖于该语言的对象（例如用该语言编写的函数）。

RESTRICT

如果有任何对象依赖该语言，则拒绝删除该语言。这是默认值。

示例

删除过程语言plsample：

```
DROP LANGUAGE plsample;
```

兼容性

SQL标准中没有DROP LANGUAGE语句。

另见

[ALTER LANGUAGE](#) , [CREATE LANGUAGE](#)

Parent topic: [SQL Command Reference](#)

DROP OPERATOR

删除运算符。

概要

```
DROP OPERATOR [IF EXISTS] name ( {lefttype | NONE} ,  
                                     {righttype | NONE} ) [CASCADE | RESTRICT]
```

描述

DROP OPERATOR从数据库系统中删除现有的运算符。 要执行此命令，您必须是运算符的所有者。

参数

IF EXISTS

如果该运算符不存在，请不要报错。在这种情况下会发出通知。

name

现有运算符的名称（可以由模式指定）。

lefttype

运算符的左操作数的数据类型；如果运算符没有左操作数，则写NONE。

righttype

运算符的右操作数的数据类型；如果运算符没有右操作数，则写NONE。

CASCADE

自动删除依赖于运算符的对象。

RESTRICT

如果有任何对象依赖于该运算符，请拒绝删除该运算符。这是默认值。

示例

删除integer类型的幂运算符a^b：

```
DROP OPERATOR ^ (integer, integer);
```

删除类型bit的左一元按位补数运算符~b：

```
DROP OPERATOR ~ (none, bit);
```

删除bigint类型的右一元阶乘运算符x!：

```
DROP OPERATOR ! (bigint, none);
```

兼容性

SQL标准中没有DROP OPERATOR语句。

另见

[ALTER OPERATOR](#) , [CREATE OPERATOR](#)

Parent topic: [SQL Command Reference](#)

DROP OPERATOR CLASS

删除运算符类。

概要

```
DROP OPERATOR CLASS [IF EXISTS] name USING index_method [CASCADE | RESTRICT]
```

描述

DROP OPERATOR删除现有的运算符类。 要执行此命令，您必须是运算符类的所有者。

参数

IF EXISTS

如果该运算符类不存在，请不要报错。在这种情况下会发出通知。

name

现有运算符类的名称（可以由模式指定）。

index_method

运算符类用于的索引访问方法的名称。

CASCADE

自动删除依赖于运算符类的对象。

RESTRICT

如果有任何对象依赖于该运算符类，则拒绝删除该类。这是默认值。

示例

删除B-树运算符类widget_ops：

```
DROP OPERATOR CLASS widget_ops USING btree;
```

如果存在使用运算符类的索引，则此命令将不会成功。 添加CASCADE以删除此类索引以及运算符类。

兼容性

SQL标准中没有DROP OPERATOR CLASS语句。

另见

[ALTER OPERATOR CLASS](#) , [CREATE OPERATOR CLASS](#)

Parent topic: [SQL Command Reference](#)

DROP OPERATOR FAMILY

删除一个运算符族

概要

```
DROP OPERATOR FAMILY [IF EXISTS] name USING index_method [CASCADE | RESTRICT]
```

描述

DROP OPERATOR FAMILY删除现有的运算符族。 要执行此命令，您必须是运算符族的所有者。

DROP OPERATOR FAMILY包括删除该族中包含的任何运算符类，但不会删除该族引用的任何运算符或函数。 如果有任何索引取决于该族中的运算符类，则需要指定CASCADE才能完成删除。

参数

IF EXISTS

如果该运算符族不存在，请不要报错。在这种情况下会发出通知。

name

现有运算符族的名称（可以由模式指定）。

index_method

运算符族用于的索引访问方法的名称。

CASCADE

自动删除依赖于运算符族的对象。

RESTRICT

如果有任何对象依赖它，请拒绝删除该运算符族。这是默认值。

示例

删除B-树运算符族float_ops：

```
DROP OPERATOR FAMILY float_ops USING btree;
```

如果存在任何使用运算符族的索引，则此命令将不会成功。 添加CASCADE可以删除此类索引以及运算符族。

兼容性

SQL标准中没有DROP OPERATOR FAMILY语句。

另见

[ALTER OPERATOR FAMILY](#) , [CREATE OPERATOR FAMILY](#) , [ALTER OPERATOR CLASS](#) , [CREATE OPERATOR CLASS](#) , [DROP OPERATOR CLASS](#)

Parent topic: [SQL Command Reference](#)

DROP OWNED

删除数据库角色拥有的数据库对象。

概要

```
DROP OWNED BY name [, ...] [CASCADE | RESTRICT]
```

描述

DROP OWNED删除当前数据库中由指定角色之一拥有的所有对象。 在当前数据库中的对象上授予给定角色的任何特权也将被撤销。

参数

name

一个角色的名称，该角色的对象将被删除，其特权将被撤销。

CASCADE

自动删除依赖于受影响对象的对象。

RESTRICT

如果任何其他数据库对象依赖于受影响的对象之一，则拒绝删除角色拥有的对象。这是默认值。

注解

DROP OWNED通常用于准备删除一个或多个角色。由于DROP OWNED仅影响当前数据库中的对象，因此通常需要在每个数据库中执行此命令，该数据库包含要删除的角色拥有的对象。

使用CASCADE选项可能会使命令递归到其他用户拥有的对象。

[REASSIGN OWNED](#)命令是一种替代方法，它重新分配一个或多个角色拥有的所有数据库对象的所有权。但是，REASSIGN OWNED不处理其他对象的特权。

示例

删除名称为sally的角色拥有的所有数据库对象：

```
DROP OWNED BY sally;
```

兼容性

DROP OWNED命令是MPP数据库扩展。

另见

[REASSIGN OWNED](#) , [DROP ROLE](#)

Parent topic: [SQL Command Reference](#)

DROP PROTOCOL

从数据库中删除外部表数据访问协议。

概要

```
DROP PROTOCOL [IF EXISTS] name
```

描述

DROP PROTOCOL从数据库中删除指定的协议。 可以在CREATE EXTERNAL TABLE命令中指定协议名称，以从外部数据源读取数据或将数据写入外部数据源。

您必须是超级用户或协议所有者才能删除协议。

Warning: 如果删除数据访问协议，则使用该协议定义的外部表将不再能够访问外部数据源。

参数

IF EXISTS

如果该协议不存在，请不要报错。在这种情况下会发出通知。

name

现有数据访问协议的名称。

注解

如果删除数据访问协议，则不会删除数据库中与该协议相关联的调用处理程序。您必须手动删除函数。

协议使用的共享库也应从MPP数据库主机中删除。

兼容性

DROP PROTOCOL是MPP数据库扩展。

另见

[CREATE EXTERNAL TABLE, CREATE PROTOCOL](#)

Parent topic: [SQL Command Reference](#)

DROP RESOURCE GROUP

删除资源组。

概要

```
DROP RESOURCE GROUP group_name
```

描述

此命令从MPP数据库中删除资源组。只有超级用户才能删除资源组。删除资源组时，该组保留的内存和CPU资源将回收回到MPP数据库。

要删除角色资源组，不能将该组分配给任何角色，也不能在该组中有任何待处理或正在运行的语句。如果删除为外部组件创建的资源组，则行为由外部组件决定。例如，删除分配给PL/Container运行时的资源组会杀死该组中正在运行的容器。

您不能删除预定义的admin_group和default_group资源组。

参数

group_name

要删除的资源组的名称。

注解

您不能在显式事务或子事务中提交DROP RESOURCE GROUP命令。

使用ALTER ROLE删除分配给特定用户/角色的资源组。

执行以下查询以查看所有资源组的所有当前活动查询：

```
SELECT username, current_query, waiting, procpid,  
       rsgid, rsgname, rsgqueueduration  
FROM pg_stat_activity;
```

要查看资源组分配，请对pg_roles和pg_resgroup系统catalog表执行以下查询：

```
SELECT rolname, rsgname  
FROM pg_roles, pg_resgroup  
WHERE pg_roles.rolresgroup=pg_resgroup.oid;
```

示例

删除分配给角色的资源组。然后，此操作将默认资源组default_group分配给角色：

```
ALTER ROLE bob RESOURCE GROUP NONE;
```

删除名为adhoc的资源组：

```
DROP RESOURCE GROUP adhoc;
```

兼容性

DROP RESOURCE GROUP语句是MPP数据库扩展。

另见

[ALTER RESOURCE GROUP](#) , [CREATE RESOURCE GROUP](#) , [ALTER ROLE](#)

Parent topic: [SQL Command Reference](#)

DROP RESOURCE QUEUE

删除资源队列。

概要

```
DROP RESOURCE QUEUE queue_name
```

描述

此命令从MPP数据库中删除资源队列。 要删除资源队列，队列不能分配任何角

色，队列中也不能有任何等待语句。只有超级用户才能删除资源队列。

参数

queue_name

要删除的资源队列的名称。

注解

使用[ALTER ROLE](#)从资源队列中删除用户。

要查看所有资源队列的所有当前活动查询，请对与pg_roles和pg_resqueue表连接pg_locks表执行以下查询：

```
SELECT rolname, rsqname, locktype, objid, pid,  
mode, granted FROM pg_roles, pg_resqueue, pg_locks WHERE  
pg_roles.rolresqueue=pg_locks.objid AND  
pg_locks.objid=pg_resqueue.oid;
```

要查看分配给资源队列的角色，请对pg_roles和pg_resqueue系统catalog表执行以下查询：

```
SELECT rolname, rsqname FROM pg_roles, pg_resqueue WHERE  
pg_roles.rolresqueue=pg_resqueue.oid;
```

示例

从资源队列中删除角色（并将角色移至默认资源队列pg_default）：

```
ALTER ROLE bob RESOURCE QUEUE NONE;
```

删除名为adhoc的资源队列：

```
DROP RESOURCE QUEUE adhoc;
```

兼容性

DROP RESOURCE QUEUE语句是MPP数据库扩展。

另见

[ALTER RESOURCE QUEUE](#) , [CREATE RESOURCE QUEUE](#) , [ALTER ROLE](#)

Parent topic: [SQL Command Reference](#)

DROP ROLE

删除数据库角色。

概要

```
DROP ROLE [IF EXISTS] name [, ...]
```

描述

DROP ROLE删除指定的角色。要删除超级用户角色，您必须自己是超级用户。要删除非超级用户角色，您必须具有CREATEROLE特权。

如果仍在任何数据库中引用角色，则无法删除该角色；如果是这样，将引发错误。在删除角色之前，必须删除其拥有的所有对象（或重新分配其所有权），并撤消已授予该角色在其他对象上的所有特权。[REASSIGN OWNED](#)和[DROP OWNED](#)命令可用于此目的。

但是，不必删除涉及该角色的角色成员资格；DROP ROLE自动撤消其他角色中的目标角色以及目标角色中的其他角色的所有成员身份。其他角色不会丢失，也不会受到其他影响。

参数

IF EXISTS

如果该角色不存在，请不要报错。在这种情况下会发出通知。

name

要删除的角色名称。

示例

删除名为sally和bob的角色：

```
DROP ROLE sally, bob;
```

兼容性

SQL标准定义了DROP ROLE，但一次只允许删除一个角色，并且它指定的特权要求与MPP数据库使用的特权要求不同。

另见

[REASSIGN OWNED](#) , [DROP OWNED](#) , [CREATE ROLE](#) , [ALTER ROLE](#) , [SET ROLE](#)

Parent topic: [SQL Command Reference](#)

DROP RULE

删除重写规则。

概要

```
DROP RULE [IF EXISTS] name ON table_name [CASCADE | RESTRICT]
```

描述

DROP RULE删除表或视图中的重写规则。

参数

IF EXISTS

如果该规则不存在，请不要报错。在这种情况下会发出通知。

name

要删除的规则的名称。

table_name

规则适用的表或视图的名称（可以由模式指定）。

CASCADE

自动删除依赖于规则的对象。

RESTRICT

如果有任何对象依赖该规则，则拒绝删除该规则。这是默认值。

示例

删除表sales上的重写规则sales_2006：

```
DROP RULE sales_2006 ON sales;
```

兼容性

DROP RULE是MPP数据库语言的扩展，整个查询重写系统也是如此。

另见

[ALTER RULE](#) , [CREATE RULE](#)

Parent topic: [SQL Command Reference](#)

DROP SCHEMA

删除模式。

概要

```
DROP SCHEMA [IF EXISTS] name [, ...] [CASCADE | RESTRICT]
```

描述

DROP SCHEMA从数据库中删除模式。 模式只能由其所有者或超级用户删除。 请注意，即使所有者不拥有该模式内的某些对象，所有者也可以删除该模式（从而删除所有包含的对象）。

参数

IF EXISTS

如果该模式不存在，请不要报错。在这种情况下会发出通知。

name

要删除的模式的名称。

CASCADE

自动删除模式中包含的所有对象（表，函数等）。

RESTRICT

如果模式包含任何对象，则拒绝删除该模式。这是默认值。

示例

从数据库中删除模式mystuff及其包含的所有内容：

```
DROP SCHEMA mystuff CASCADE;
```

兼容性

DROP SCHEMA完全符合SQL标准，但该标准仅允许每个命令删除一个模式。另外，IF EXISTS选项是MPP数据库扩展。

另见

[CREATE SCHEMA](#) , [ALTER SCHEMA](#)

Parent topic: [SQL Command Reference](#)

DROP SEQUENCE

删除序列。

概要

```
DROP SEQUENCE [IF EXISTS] name [, ...] [CASCADE | RESTRICT]
```

描述

DROP SEQUENCE删除序列生成器表。 您必须拥有删除的序列（或成为超级用户）。

参数

IF EXISTS

如果该序列不存在，请不要报错。在这种情况下会发出通知。

name

要删除的序列的名称（可以由模式指定）。

CASCADE

自动删除依赖于序列的对象。

RESTRICT

如果有任何对象依赖该序列，则拒绝删除该序列。这是默认值。

示例

删除序列myserial :

```
DROP SEQUENCE myserial;
```

兼容性

DROP SEQUENCE完全符合SQL标准，但该标准仅允许每个命令删除一个序列。 另外，IF EXISTS选项是MPP数据库扩展。

另见

[ALTER SEQUENCE](#) , [CREATE SEQUENCE](#)

Parent topic: [SQL Command Reference](#)

DROP SERVER

删除外部服务器描述符。

概要

```
DROP SERVER [ IF EXISTS ] servername [ CASCADE | RESTRICT ]
```

描述

DROP SERVER删除现有的外部服务器描述符。 执行此命令的用户必须是服务器的所有者。

参数

IF EXISTS

如果该服务不存在，请不要报错。在这种情况下会发出通知。

servername

现有服务器的名称。

CASCADE

自动删除依赖于服务器的对象（例如用户映射）。

RESTRICT

如果有任何对象依赖于该服务器，则拒绝删除该服务器。 这是默认值。

示例

删除名为foo的服务器（如果存在）：

```
DROP SERVER IF EXISTS foo;
```

兼容性

DROP SERVER符合ISO/IEC 9075-9（SQL/MED）。IF EXISTS子句是MPP数据库扩展。

另见

[CREATE SERVER](#), [ALTER SERVER](#)

Parent topic: [SQL Command Reference](#)

DROP TABLE

删除表。

概要

```
DROP TABLE [IF EXISTS] name [, ...] [CASCADE | RESTRICT]
```

描述

DROP TABLE从数据库中删除表。只有表所有者，模式所有者和超级用户可以删除表。要清空行表而不删除表定义，请使用DELETE或TRUNCATE。

DROP TABLE始终删除目标表存在的所有索引，规则，触发器和约束。但是，要删除视图引用的表，必须指定CASCADE。CASCADE将完全删除从属视图。

参数

IF EXISTS

如果该表不存在，请不要报错。在这种情况下会发出通知。

name

要删除的表的名称（可以由模式指定）。

CASCADE

自动删除依赖于表的对象（例如视图）。

RESTRICT

如果有任何对象依赖表，则拒绝删除该表。这是默认值。

示例

删除表mytable：

```
DROP TABLE mytable;
```

兼容性

DROP TABLE完全符合SQL标准，但该标准仅允许每个命令删除一个表。 另外，IF EXISTS选项是MPP数据库扩展。

另见

[CREATE TABLE](#) , [ALTER TABLE](#) , [TRUNCATE](#)

Parent topic: [SQL Command Reference](#)

DROP TABLESPACE

删除表空间。

概要

```
DROP TABLESPACE [IF EXISTS] tablespacename
```

描述

DROP TABLESPACE从系统中删除表空间。

表空间只能由其所有者或超级用户删除。 在删除表空间之前，表空间中的所有

数据库对象必须为空。即使当前数据库中没有对象正在使用表空间，其他数据库中的对象也可能仍驻留在表空间中。

参数

IF EXISTS

如果该表空间不存在，请不要报错。在这种情况下会发出通知。

tablespacename

要删除的表空间的名称。

示例

删除表空间mystuff：

```
DROP TABLESPACE mystuff;
```

兼容性

DROP TABLESPACE是MPP数据库扩展。

另见

[CREATE TABLESPACE](#) , [ALTER TABLESPACE](#)

Parent topic: [SQL Command Reference](#)

DROP TEXT SEARCH CONFIGURATION

删除文本搜索配置。

概要

```
DROP TEXT SEARCH CONFIGURATION [ IF EXISTS ] name [ CASCADE | RESTRICT ]
```

描述

DROP TEXT SEARCH CONFIGURATION删除已有的文本搜索配置。 要执行此命令，您必须是配置的所有者。

参数

IF EXISTS

如果该配置不存在，请不要报错。在这种情况下会发出通知。

name

现有文本搜索配置的名称（可以由模式指定）。

CASCADE

自动删除依赖于文本搜索配置的对象。

RESTRICT

如果有任何对象依赖它，则拒绝删除文本搜索配置。这是默认值。

示例

删除文本搜索配置my_english：

```
DROP TEXT SEARCH CONFIGURATION my_english;
```

如果有任何现有索引引用to_tsvector调用中的配置，则此命令将不会成功。 添加CASCADE以删除此类索引以及文本搜索配置。

兼容性

在SQL标准中没有DROP TEXT SEARCH CONFIGURATION语句。

另见

[ALTER TEXT SEARCH CONFIGURATION](#) , [CREATE TEXT SEARCH CONFIGURATION](#)

Parent topic: [SQL Command Reference](#)

DROP TEXT SEARCH DICTIONARY

删除文本搜索字典。

概要

```
DROP TEXT SEARCH DICTIONARY [ IF EXISTS ] name [ CASCADE | RESTRICT ]
```

描述

DROP TEXT SEARCH DICTIONARY删除现有的文本搜索词典。 要执行此命令，您必须是字典的所有者。

参数

IF EXISTS

如果该字典不存在，请不要报错。在这种情况下会发出通知。

name

现有文本搜索字典的名称（可以由模式指定）。

CASCADE

自动删除依赖于文本搜索字典的对象。

RESTRICT

如果有任何对象依赖它，则拒绝删除文本搜索字典。这是默认值。

示例

删除文本搜索词典english :

```
DROP TEXT SEARCH DICTIONARY english;
```

如果存在使用字典的任何现有文本搜索配置，则此命令将不会成功。 添加CASCADE可以将此类配置与字典一起删除。

兼容性

SQL标准中没有CREATE TEXT SEARCH DICTIONARY语句。

另见

[ALTER TEXT SEARCH DICTIONARY](#) , [CREATE TEXT SEARCH DICTIONARY](#)

Parent topic: [SQL Command Reference](#)

DROP TEXT SEARCH PARSER

描述

删除文本搜索解析器。

概要

```
DROP TEXT SEARCH PARSE[ IF EXISTS ] name [ CASCADE | RESTRICT ]
```

描述

DROP TEXT SEARCH PARSE删除现有的文本搜索解析器。 您必须是超级用户才能使用此命令。

参数

IF EXISTS

如果该解析器不存在，请不要报错。在这种情况下会发出通知。

name

现有文本搜索解析器的名称（可以由模式指定）。

CASCADE

自动删除依赖于文本搜索解析器的对象。

RESTRICT

如果有任何对象依赖它，则拒绝删除文本搜索解析器。这是默认值。

示例

删除文本搜索解析器my_parser :

```
DROP TEXT SEARCH PARSER my_parser;
```

如果存在使用解析器的任何现有文本搜索配置，则此命令将不会成功。 添加CASCADE可以将这些配置与解析器一起删除。

兼容性

在SQL标准中没有DROP TEXT SEARCH PARSER语句。

另见

[ALTER TEXT SEARCH PARSER](#), [CREATE TEXT SEARCH PARSER](#)

Parent topic: [SQL Command Reference](#)

DROP TEXT SEARCH TEMPLATE

描述

删除文本搜索模板。

概要

```
DROP TEXT SEARCH TEMPLATE [ IF EXISTS ] name [ CASCADE | RESTRICT ]
```

描述

DROP TEXT SEARCH TEMPLATE删除现有的文本搜索模板。您必须是超级用户才能使用此命令。

您必须是超级用户才能使用ALTER TEXT SEARCH TEMPLATE。

参数

IF EXISTS

如果该模板不存在，请不要报错。在这种情况下会发出通知。

name

现有文本搜索模板的名称（可以由模式指定）。

CASCADE

自动删除依赖于文本搜索模板的对象。

RESTRICT

如果有任何对象依赖文本搜索模板，则拒绝删除它。这是默认值。

兼容性

在SQL标准中没有DROP TEXT SEARCH TEMPLATE语句。

另见

[ALTER TEXT SEARCH TEMPLATE](#), [CREATE TEXT SEARCH TEMPLATE](#)

Parent topic: [SQL Command Reference](#)

DROP TYPE

删除数据类型。

概要

```
DROP TYPE [IF EXISTS] name [, ...] [CASCADE | RESTRICT]
```

描述

DROP TYPE将删除用户定义的数据类型。 只有类型的所有者才能将其删除。

参数

IF EXISTS

如果该类型不存在，请不要报错。在这种情况下会发出通知。

name

要删除的数据类型的名称（可以由模式指定）。

CASCADE

自动删除依赖于类型的对象（例如表列，函数，运算符）。

RESTRICT

如果有任何对象依赖该类型，则拒绝删除该类型。这是默认值。

示例

删除数据类型box；

```
DROP TYPE box;
```

兼容性

除了IF EXISTS选项（这是MPP数据库扩展）之外，此命令与SQL标准中的相应命令相似。但是请注意，MPP数据库中的许多CREATE TYPE命令和数据类型扩展机制与SQL标准不同。

另见

[ALTER TYPE](#) , [CREATE TYPE](#)

Parent topic: [SQL Command Reference](#)

DROP USER

删除数据库角色。

概要

```
DROP USER [IF EXISTS] name [, ...]
```

描述

DROP USER是[DROP ROLE](#)的别名。 有关更多信息，请参见[DROP ROLE](#)。

兼容性

SQL标准中没有DROP USER语句。 SQL标准将用户的定义留给实现。

另见

[DROP ROLE](#)

Parent topic: [SQL Command Reference](#)

DROP USER MAPPING

删除外部服务器的用户映射。

概要

```
DROP USER MAPPING [ IF EXISTS ] { username | USER | CURRENT_USER | PUBLIC }  
SERVER servername
```

描述

DROP USER MAPPING从外部服务器中删除现有的用户映射。 要执行此命令，当前用户必须是包含映射的服务器的所有者。

参数

IF EXISTS

如果该用户映射不存在，请不要报错。在这种情况下会发出通知。

username

映射的用户名。 CURRENT_USER和USER与当前用户的名称匹配。
PUBLIC用于匹配系统中所有当前和将来的用户名。

servername

用户映射的服务器名称。

示例

删除名为bob的用户映射，如果存在，则删除服务器foo：

```
DROP USER MAPPING IF EXISTS FOR bob SERVER foo;
```

兼容性

DROP SERVER符合ISO/IEC 9075-9（SQL/MED）。IF EXISTS子句是MPP数据库扩展。

另见

[CREATE USER MAPPING](#), [ALTER USER MAPPING](#)

Parent topic: [SQL Command Reference](#)

DROP VIEW

删除视图。

概要

```
DROP VIEW [IF EXISTS] name [, ...] [CASCADE | RESTRICT]
```

描述

DROP VIEW将删除现有视图。 仅视图所有者可以将其删除。

参数

IF EXISTS

如果该视图不存在，请不要报错。在这种情况下会发出通知。

name

要删除的视图的名称（可以由模式指定）。

CASCADE

自动删除依赖于视图的对象（例如其他视图）。

RESTRICT

如果有任何对象依赖该视图，则拒绝删除该视图。这是默认值。

示例

删除视图topten；

```
DROP VIEW topten;
```

兼容性

DROP VIEW完全符合SQL标准，但该标准仅允许每个命令删除一个视图。 另外，IF EXISTS选项是MPP数据库扩展。

另见

[CREATE VIEW](#)

Parent topic: [SQL Command Reference](#)

ENABLE USER

允许一个数据库角色重新登录数据库。

概要

```
ENABLE USER name
```

描述

当用户使用password或MD5认证登录时,如果输入密码错误超过指定次数,会被锁

定不允许登录。超户可使用ENABLE USER命令解锁被锁定的用户。被解锁后，用户可重新登录，且其之前的密码错误次数被清零。

参数

name
一个存在的角色的名称。

兼容性

在SQL标准中没有 ENABLE USER 命令。

上级主题：[SQL命令参考](#)

END

提交当前事务。

概要

```
END [WORK | TRANSACTION]
```

描述

END提交当前事务。事务所做的所有更改对其他人都可见，即使发生崩溃，也保证是持久的。此命令是MPP数据库扩展，等效于[COMMIT](#)。

参数

WORK
TRANSACTION
 可选关键字。它们没有作用。

示例

提交当前事务：

```
END;
```

兼容性

END是MPP数据库扩展，它提供的功能等同于SQL标准中指定的[COMMIT](#)。

另见

[BEGIN](#) , [ROLLBACK](#) , [COMMIT](#)

Parent topic: [SQL Command Reference](#)

EXECUTE

执行准备好的SQL语句。

概要

```
EXECUTE name [ (parameter [, ...] ) ]
```

描述

EXECUTE用于执行先前准备的语句。由于准备好的语句仅在会话期间存在，因此准备好的语句必须由当前会话中较早执行的PREPARE语句创建。

如果创建该语句的PREPARE语句指定了一些参数，则必须将一组兼容的参数传递给EXECUTE语句，否则会引发错误。请注意（与函数不同），准备好的语句不会基于其参数的类型或数量而重载。在数据库会话中，准备好的语句的名称必须唯一。

有关准备语句的创建和使用的更多信息，请参见PREPARE。

参数

name

要执行的准备好的语句的名称。

parameter

准备语句的参数的实际值。该表达式必须是产生与该参数的数据类型兼容的值的表达式，这是在创建准备语句时确定的。

示例

为INSERT语句创建一个准备语句，然后执行它：

```
PREPARE fooplan (int, text, bool, numeric) AS INSERT INTO
foo VALUES($1, $2, $3, $4);
EXECUTE fooplan(1, 'Hunter Valley', 't', 200.00);
```

兼容性

SQL标准包含一个EXECUTE语句，但仅用于嵌入式SQL。此版本的EXECUTE语句还使用了一些不同的语法。

另见

[DEALLOCATE](#) , [PREPARE](#)

Parent topic: [SQL Command Reference](#)

EXPLAIN

显示语句的查询计划。

概要

```
EXPLAIN [ ( option [, ...] ) ] statement  
EXPLAIN [ANALYZE] [VERBOSE] statement
```

其中`option`可以是以下之一：

```
ANALYZE [ boolean ]  
VERBOSE [ boolean ]  
COSTS [ boolean ]  
BUFFERS [ boolean ]  
TIMING [ boolean ]  
FORMAT { TEXT | XML | JSON | YAML }
```

描述

EXPLAIN显示MPP或Postgres优化器为提供的语句生成的查询计划。 查询计划是节点的查询树计划。 计划中的每个节点代表一个单独的操作，例如表扫描，连接，聚合或排序。

应从下至上阅读计划，因为每个节点都会向其上方的节点中发送行。 计划的最底层节点通常是表扫描操作（顺序扫描，索引扫描或位图索引扫描）。 如果查

询需要连接，聚集或排序（或原始行上的其他操作），则扫描节点上方将有其他节点来执行这些操作。最顶层的计划节点通常是MPP数据库motion节点（重新分发，显式重新分发，广播或收集motion）。这些操作负责在查询处理期间在segment实例之间移动行。

EXPLAIN的输出对于计划树中的每个节点都有一行，显示基本节点类型以及计划者为执行该计划节点而进行的以下成本估算：

- **cost** — 优化器对运行该语句要花费多长时间的猜测（以任意成本单位衡量，但通常是指磁盘页获取）。显示了两个成本编号：可以返回第一行之前的启动成本，以及返回所有行的总成本。请注意，总成本假定将检索所有行，但并非总是如此（例如，如果使用LIMIT）。
- **rows** — 此计划节点输出的总行数。这通常少于计划节点处理或扫描的实际行数，反映了任何WHERE子句条件的估计选择性。理想情况下，顶级节点估计将近似查询实际返回，更新或删除的行数。
- **width** — 此计划节点输出的所有行的总字节数。

重要的是要注意，上级节点的成本包括其所有子节点的成本。计划的最高节点具有该计划的估计总执行成本。这是计划者要尽量减少的数字。同样重要的是要意识到，成本只反映查询优化器关心的事情。特别是，成本不考虑将结果行传输到客户端所花费的时间。

EXPLAIN ANALYZE导致语句实际执行，而不仅仅是做计划。EXPLAIN ANALYZE优化器会显示实际结果以及计划者的估计。这对于查看优化器的估计是否接近实际很有用。除了EXPLAIN计划中显示的信息之外，EXPLAIN ANALYZE还将显示以下附加信息：

- 运行查询所花费的总时间（以毫秒为单位）。
 - 计划节点操作中涉及的workers（segment）数。仅计算返回行的segment。
 - 操作产生最多行的segment所返回的最大行数。如果多个segment产生相等数量的行，则结束时间最长的一个就是选择的那个。
 - 为一个操作生成最多行的segment的segment ID号。
-

- 对于相关操作，该操作使用的`work_mem`。如果`work_mem`不足以在内存中执行该操作，则该计划将显示有多少数据溢出到磁盘，以及最低性能segment需要多少次数据传递。例如：

```
Work_mem used: 64K bytes avg, 64K bytes max (seg0).  
Work_mem wanted: 90K bytes avg, 90K bytes max (seg0) to abate workfile  
I/O affecting 2 workers.  
[seg0] pass 0: 488 groups made from 488 rows; 263 rows written to  
workfile  
[seg0] pass 1: 263 groups made from 263 rows
```

- 从产生最多行的segment中检索第一行所花费的时间（以毫秒为单位），以及从该segment中检索所有行所花费的总时间。如果`<time> to first row`与`<time> to end`相同，则可以省略。

Important: 请记住，使用ANALYZE时实际上会执行该语句。尽管EXPLAIN ANALYZE将丢弃SELECT将返回的任何输出，但是该语句的其他副作用将照常发生。如果希望在DML语句上使用EXPLAIN ANALYZE而不让命令影响您的数据，请使用以下方法：

```
BEGIN;  
EXPLAIN ANALYZE ...;  
ROLLBACK;
```

仅可以指定ANALYZE和VERBOSE选项，并且只能按该顺序指定，而不要在括号中包含选项列表。

参数

ANALYZE

执行命令并显示实际运行时间和其他统计信息。如果省略此参数，则默认为FALSE。指定ANALYZE true可以启用它。

VERBOSE

显示有关计划的其他信息。具体来说，包括计划树中每个节点的输出列列表，模式限定表和函数名称，始终在表达式中使用范围表别名标记变量，并始终打印要显示其统计信息的每个触发器的名称。如果省略此参数，则默认为FALSE；指定VERBOSE true启用它。

COSTS

包括有关每个计划节点的估计启动成本和总成本以及估计的行数和估计的每行宽度的信息。如果省略此参数，则默认为TRUE；指定COSTS false禁用它。

BUFFERS

包括有关缓冲区使用情况的信息。具体来说，包括命中，读取，弄脏和写入的共享块的数量，命中，读取，弄脏和写入的局部块的数量以及读写的临时块的数量。命中表示避免读取，因为在需要时已在高速缓存中找到该块。共享块包含来自常规表和索引的数据；本地块包含来自临时表和索引的数据；临时块包含用于排序，哈希，物化计划节点和类似情况的短期工作数据。被弄脏的块数表示此查询已更改的先前未修改的块数；而写入的块数则表示此后端在查询处理期间从缓存中逐出的先前处理的块数。上级节点显示的块数包括其所有子节点使用的块数。在文本格式中，仅打印非零值。仅当还启用了ANALYZE时，才可以使用此参数。如果省略此参数，则默认为FALSE；指定BUFFERS true启用它。

TIMING

在输出中包括实际的启动时间和在每个节点上花费的时间。重复读取系统时钟的开销可能会在某些系统上显着降低查询速度，因此，当仅需要实际的行计数而不是确切的时间时，将此参数设置为FALSE可能会很有用。即使使用此选项关闭了节点级计时，也始终会测量整个语句的运行时间。仅当还启用了ANALYZE时，才可以使用此参数。默认为TRUE。

FORMAT

指定输出格式，可以是TEXT，XML，JSON或YAML。非文本输出包含与文本输出格式相同的信息，但程序更易于解析。此参数默认为TEXT。

boolean

指定是打开还是关闭所选选项。您可以写入TRUE，ON或1以启用该选项，而可以写入FALSE，OFF或0以禁用该选项。布尔值也可以省略，在这种情况下，假定为TRUE。

statement

您希望查看其执行计划的任何SELECT，INSERT，UPDATE，DELETE，VALUES，EXECUTE，DECLARE或CREATE TABLE AS语句。

注解

为了使查询优化器在优化查询时能够做出合理的决策，应运行ANALYZE语句以记录有关表内数据分布的统计信息。如果您尚未执行此操作（或者自上次运行ANALYZE以来，表中数据的统计分布已发生重大变化），则估计成本不太可能符合查询的实际属性，因此可能会选一个较差的查询计划。

在执行EXPLAIN ANALYZE命令期间运行的SQL语句从MPP数据库资源队列中排除。

有关查询分析的更多信息，请参阅*MPP数据库管理员手册*中的“查询分析”。有关资源队列的更多信息，请参阅*MPP数据库管理员手册*中的“使用资源队列进行资源管理”。

示例

为了说明如何读取EXPLAIN查询计划，请考虑一个非常简单的查询的示例：

```
EXPLAIN SELECT * FROM names WHERE name = 'Joelle';  
QUERY PLAN
```

```
-----  
Gather Motion 3:1 (slice1; segments: 3) (cost=0.00..431.27 rows=1 width=58)  
-> Seq Scan on names (cost=0.00..431.27 rows=1 width=58)  
    Filter: (name = 'Joelle'::text)  
Optimizer: Optimizer (GPORCA) version 3.23.0  
(4 rows)
```

如果我们自下而上阅读计划，则查询优化器将从对names表的顺序扫描开始。请注意，WHERE子句被用作过滤条件。这意味着扫描操作将检查其扫描的每一行的条件，并仅输出通过条件的行。

扫描操作的结果将传递到`gather motion`操作。在MPP数据库中，`gather motion`是将segment行发送到master。在这种情况下，我们有3个segment实例发送到1个master实例（3：1）。该操作在并行查询执行计划的slice1上进行。在MPP数据库中，查询计划分为多个切片，以便查询计划的各个部分可以由这些segment并行处理。

该计划的估计启动成本为00.00（无成本），总成本为431.27。优化器估计此查询将返回一行。

这是相同的查询，但成本估算被抑制：

```
EXPLAIN (COSTS FALSE) SELECT * FROM names WHERE name = 'Joelle';
      QUERY PLAN
-----
Gather Motion 3:1 (slice1; segments: 3)
-> Seq Scan on names
    Filter: (name = 'Joelle'::text)
Optimizer: Optimizer (GPORCA) version 3.23.0
(4 rows)
```

这是使用JSON格式的相同查询：

EXPLAIN (FORMAT JSON) SELECT * FROM names WHERE name = 'Joelle';
QUERY PLAN

```
-----  
[  
  {  
    "Plan": {  
      "Node Type": "Gather Motion",  
      "Senders": 3,  
      "Receivers": 1,  
      "Slice": 1,  
      "Segments": 3,  
      "Gang Type": "primary reader",  
      "Startup Cost": 0.00,  
      "Total Cost": 431.27,  
      "Plan Rows": 1,  
      "Plan Width": 58,  
      "Plans": [  
        {  
          "Node Type": "Seq Scan",  
          "Parent Relationship": "Outer",  
          "Slice": 1,  
          "Segments": 3,  
          "Gang Type": "primary reader",  
          "Relation Name": "names",  
          "Alias": "names",  
          "Startup Cost": 0.00,  
          "Total Cost": 431.27,  
          "Plan Rows": 1,  
          "Plan Width": 58,  
          "Filter": "(name = 'Joelle'::text)" +  
        }  
      ]  
    },  
    "Settings": {  
      "Optimizer": " Optimizer (GPORCA) version 3.23.0" +  
    }  
  }  
(1 row)
```

如果存在索引，并且我们使用带有可索引WHERE条件的查询，则EXPLAIN可

能会显示不同的计划。 此查询使用YAML格式生成带有索引扫描的计划：

```
EXPLAIN (FORMAT YAML) SELECT * FROM NAMES WHERE LOCATION='Sydney, Australia'
QUERY PLAN
-----
- Plan:
  Node Type: "Gather Motion"
  Senders: 3
  Receivers: 1
  Slice: 1
  Segments: 3
  Gang Type: "primary reader"
  Startup Cost: 0.00
  Total Cost: 10.81
  Plan Rows: 10000
  Plan Width: 70
  Plans:
    - Node Type: "Index Scan"
      Parent Relationship: "Outer"
      Slice: 1
      Segments: 3
      Gang Type: "primary reader"
      Scan Direction: "Forward"
      Index Name: "names_idx_loc"
      Relation Name: "names"
      Alias: "names"
      Startup Cost: 0.00
      Total Cost: 7.77
      Plan Rows: 10000
      Plan Width: 70
      Index Cond: "(location = 'Sydney, Australia'::text)"
  Settings:
    Optimizer: " Optimizer (GPORCA) version 3.23.0"
(1 row)
```

兼容性

在SQL标准中没有定义EXPLAIN语句。

另见

[ANALYZE](#)

Parent topic: [SQL Command Reference](#)

FETCH

使用游标从查询中检索行。

概要

```
FETCH [ forward_direction { FROM | IN } ]  
      cursor_name
```

其中*forward_direction*可以为空或以下之一：

```
NEXT
FIRST
LAST
ABSOLUTE count
RELATIVE count
count
ALL
FORWARD
FORWARD count
FORWARD ALL
```

描述

FETCH使用先前创建的游标检索行。

Note: 本页在SQL命令级别描述游标的用法。 如果试图在PL/pgSQL函数中使用游标，则规则是不同的。 参见[MPP PL/pgSQL过程语言](#)。

游标具有关联的位置，该位置由FETCH使用。 光标位置可以在查询结果的第一行之前，结果的任何特定行上，或者在结果的最后一行之后。 创建后，光标将位于第一行之前。 在获取了一些行之后，将光标定位在最近检索到的行上。 如果FETCH在可用行的末尾运行，则光标将位于最后一行之后。 FETCH ALL将始终使光标位于最后一行之后。

NEXT，FIRST，LAST，ABSOLUTE，RELATIVE形式在适当移动光标后获取单行。 如果没有这样的行，则返回空结果，并且光标将视情况放在第一行之前或最后一行之后。

使用FORWARD的表单将检索指示的向前行的行数，将光标留在最后返回的行上（如果计数超过可用行数，则在所有行之后）。 请注意，由于不支持可滚动光标，因此无法在MPP数据库中后向移动光标位置。 您只能使用FETCH向前移动光标。

RELATIVE 0和FORWARD 0在不移动光标的情况下请求获取当前行，即重新获取最近获取的行。除非光标位于第一行之前或最后一行之后，否则此操作将成功，在这种情况下不会返回任何行。

输出

成功完成后，FETCH命令将返回以下形式的命令标签：

```
FETCH count
```

该计数是获取的行数（可能为零）。请注意，在psql中，实际上不会显示command标记，因为psql只是显示提取的行。

参数

forward_direction

定义获取方向和要获取的行数。MPP数据库中仅允许前向提取。可以是以下之一：

NEXT

获取下一行。如果省略了方向，则为默认设置。

FIRST

提取查询的第一行（与ABSOLUTE 1相同）。仅当它是使用此光标的第一个FETCH操作时才允许。

LAST

获取查询的最后一行（与ABSOLUTE -1相同）。

ABSOLUTE count

获取查询的指定行。如果计数超出范围，则在最后一行之后定位。只有在被指定了*count*的行向前移动游标位置才能使用。

RELATIVE count

在当前光标位置之前获取查询*count*行的指定行。RELATIVE 0重新获取当前行（如果有）。仅当*count*向前移动光标位置时才允许。

count

获取下一个*count*行数（与FORWARD *count*相同）。

ALL

提取所有剩余的行（与FORWARD ALL相同）。

FORWARD

获取下一行（与NEXT相同）。

FORWARD *count*

获取下一个*count*行数。 FORWARD 0重新获取当前行。

FORWARD ALL

提取所有剩余的行。

cursor_name

打开的游标的名称。

注解

MPP数据库不支持可滚动光标，因此只能使用FETCH向前移动光标位置。

ABSOLUTE获取并不比通过相对移动导航到所需行更快：底层实现无论如何都必须遍历所有中间行。

DECLARE用于定义游标。使用MOVE更改光标位置而不检索数据。

示例

-- 开始事务：

```
BEGIN;
```

-- 设置游标：

```
DECLARE mycursor CURSOR FOR SELECT * FROM films;
```

-- 获取游标mycursor中的前5行：

```
FETCH FORWARD 5 FROM mycursor;
code | title | did | date_prod | kind | len
-----+-----+-----+-----+-----+-----
BL101 | The Third Man | 101 | 1949-12-23 | Drama | 01:44
BL102 | The African Queen | 101 | 1951-08-11 | Romantic | 01:43
JL201 | Une Femme est une Femme | 102 | 1961-03-12 | Romantic | 01:25
P_301 | Vertigo | 103 | 1958-11-14 | Action | 02:08
P_302 | Becket | 103 | 1964-02-03 | Drama | 02:28
```

-- 关闭游标并提交事务：

```
CLOSE mycursor;
COMMIT;
```

在c_films光标当前位置的行中更改表films的kind列：

```
UPDATE films SET kind = 'Dramatic' WHERE CURRENT OF c_films;
```

兼容性

SQL标准仅允许在嵌入式SQL和模块中使用游标。MPP数据库允许以交互方式使用游标。

这里描述的FETCH的变量将数据作为SELECT结果返回，而不是将其放在主机变量中。除此之外，FETCH与SQL标准完全向上兼容。

涉及FORWARD的FETCH形式以及其中隐含FORWARD的FETCH

count和FETCH ALL形式都是MPP数据库扩展。 不支持BACKWARD。

SQL标准只允许在游标名称之前使用FROM。 使用IN或完全不使用它们的选项是扩展。

另见

[DECLARE](#) , [CLOSE](#) , [MOVE](#)

Parent topic: [SQL Command Reference](#)

GRANT

定义访问权限。

概要

```
GRANT { {SELECT | INSERT | UPDATE | DELETE | REFERENCES |  
TRIGGER | TRUNCATE } [, ...] | ALL [PRIVILEGES] }  
  ON { [TABLE] table_name [, ...]  
      | ALL TABLES IN SCHEMA schema_name [, ...] }  
  TO { [ GROUP ] role_name | PUBLIC } [, ...] [ WITH GRANT OPTION ]  
  
GRANT { { SELECT | INSERT | UPDATE | REFERENCES } ( column_name [, ...] )  
      [, ...] | ALL [ PRIVILEGES ] ( column_name [, ...] ) }  
  ON [ TABLE ] table_name [, ...]  
  TO { role_name | PUBLIC } [, ...] [ WITH GRANT OPTION ]
```

```
TO { [ role_name | PUBLIC ] [, ...] [ WITH GRANT OPTION ] }
```

```
GRANT { { USAGE | SELECT | UPDATE } [, ...] | ALL [PRIVILEGES] }
  ON { SEQUENCE sequence_name [, ...]
      | ALL SEQUENCES IN SCHEMA schema_name [, ...] }
  TO { [ GROUP ] role_name | PUBLIC } [, ...] [ WITH GRANT OPTION ]
```

```
GRANT { { CREATE | CONNECT | TEMPORARY | TEMP } [, ...] | ALL
[PRIVILEGES] }
  ON DATABASE database_name [, ...]
  TO { [ GROUP ] role_name | PUBLIC } [, ...] [ WITH GRANT OPTION ]
```

```
GRANT { USAGE | ALL [ PRIVILEGES ] }
  ON DOMAIN domain_name [, ...]
  TO { [ GROUP ] role_name | PUBLIC } [, ...] [ WITH GRANT OPTION ]
```

```
GRANT { USAGE | ALL [ PRIVILEGES ] }
  ON FOREIGN DATA WRAPPER fdw_name [, ...]
  TO { [ GROUP ] role_name | PUBLIC } [, ...] [ WITH GRANT OPTION ]
```

```
GRANT { USAGE | ALL [ PRIVILEGES ] }
  ON FOREIGN SERVER server_name [, ...]
  TO { [ GROUP ] role_name | PUBLIC } [, ...] [ WITH GRANT OPTION ]
```

```
GRANT { EXECUTE | ALL [PRIVILEGES] }
  ON { FUNCTION function_name ( [ [ argmode ] [ argname ] argtype [, ...]
] ) [, ...]
      | ALL FUNCTIONS IN SCHEMA schema_name [, ...] }
  TO { [ GROUP ] role_name | PUBLIC } [, ...] [ WITH GRANT OPTION ]
```

```
GRANT { USAGE | ALL [PRIVILEGES] }
  ON LANGUAGE lang_name [, ...]
  TO { [ GROUP ] role_name | PUBLIC } [, ...] [ WITH GRANT OPTION ]
```

```
GRANT { { CREATE | USAGE } [, ...] | ALL [PRIVILEGES] }
  ON SCHEMA schema_name [, ...]
  TO { [ GROUP ] role_name | PUBLIC } [, ...] [ WITH GRANT OPTION ]
```

```
GRANT { CREATE | ALL [PRIVILEGES] }
  ON TABLESPACE tablespace_name [, ...]
  TO { [ GROUP ] role_name | PUBLIC } [, ...] [ WITH GRANT OPTION ]
```

```
GRANT { USAGE | ALL [ PRIVILEGES ] }
  ON TYPE type_name [, ...]
```

```
ON TYPE type_name [, ...]  
TO { [ GROUP ] role_name | PUBLIC } [, ...] [ WITH GRANT OPTION ]  
  
GRANT parent_role [, ...]  
TO member_role [, ...] [WITH ADMIN OPTION]  
  
GRANT { SELECT | INSERT | ALL [PRIVILEGES] }  
ON PROTOCOL protocolname  
TO username
```

描述

MPP数据库将用户和组的概念统一为一种称为角色的实体。因此，不必使用关键字GROUP来标识被授予者是用户还是组。命令中仍然允许使用GROUP，但这是一个干扰词。

GRANT命令具有两种基本变体：一种对数据库对象（表，列，视图，外部表，序列，数据库，外部数据包装器，外部服务器，函数，过程语言，模式或表空间）授予特权，以及授予一个角色中的成员关系。

数据库对象上的GRANT

GRANT命令的此变体将一个数据库对象的特定特权授予一个或多个角色。这些特权将添加到已授予的特权（如果有）中。

还有一个选项可以授予一个或多个模式中所有相同类型的对象的特权。当前仅表，序列和函数支持此功能（但请注意，ALL TABLES被视为包括视图和外部表）。

关键字PUBLIC表示将特权授予所有角色，包括以后可能创建的角色。可以将PUBLIC视为始终包含所有角色的隐式定义的组级别角色。任何特定角色将具有直接授予它的特权，授予它当前所属的任何角色的特权以及授予PUBLIC的特权的总和。

如果指定了WITH GRANT OPTION，则特权的接收者可以依次将其授予其他人。没有grant option，接收者将无法做到这一点。Grant options不能授予PUBLIC。

无需向对象的所有者（通常是创建该对象的角色）授予特权，因为所有者默认情况下具有所有特权。（但是，所有者可以出于安全考虑选择撤销自己的某些特权。）

删除对象或以任何方式更改其定义的权利不视为可授予的特权；它是所有者固有的，不能被授予或撤销。（但是，通过授予或撤销拥有对象的角色成员资格，可以获得类似的效果；请参阅下文。）所有者也隐式拥有该对象的所有授予选项。

MPP数据库将某些类型的对象的默认特权授予PUBLIC。默认情况下，不将在表，表列，序列，外部数据包装器，外部服务器，大型对象，模式或表空间上的权限授予PUBLIC。对于其他类型的对象，授予PUBLIC的默认特权如下：

- 数据库的CONNECT和TEMPORARY（创建临时表）特权，
- 函数的EXECUTE特权
- 语言和数据类型（包括域）的USAGE特权。

当然，对象所有者可以REVOKE默认特权和明确授予的特权。（为了获得最大的安全性，请在创建对象的同一事务中执行REVOKE；因此，没有任何时间窗口可供其他用户使用该对象。）

角色上的GRANT

GRANT命令的此变体将一个角色的成员资格授予一个或多个其他角色。角色的成员资格意义重大，因为它可以将授予角色的特权传达给每个成员。

如果指定了WITH ADMIN OPTION，则成员可以依次将角色的成员资格授予其他人，也可以撤销该角色的成员资格。没有admin选项，普通用户将无法做到这一点。角色不被认为拥有WITH ADMIN OPTION本身，但是它可以从未会话用户与角色匹配的数据库会话中授予或撤销成员资格。数据库超级用户可以以任何

角色向任何人授予或撤销成员资格。具有CREATEROLE特权的角色可以授予或撤销不是超级用户的任何角色的成员资格。

与具有特权的情况不同，不能将角色的成员资格授予PUBLIC。

协议上的GRANT

您还可以使用GRANT命令指定哪些用户可以访问受信任的协议。（如果该协议不受信任，则您不能授予任何其他用户权限以使用它来读取或写入数据。）

- 要允许用户使用受信任的协议创建可读的外部表：

```
GRANT SELECT ON PROTOCOL protocolname TO username
```

- 要允许用户使用受信任的协议创建可写外部表：

```
GRANT INSERT ON PROTOCOL protocolname TO username
```

- 要允许用户使用受信任的协议创建可读和可写的外部表：

```
GRANT ALL ON PROTOCOL protocolname TO username
```

您还可以使用此命令来授予用户创建和使用s3和pxf外部表的权限。但是，类型为http，https，gpfdist和gpfdists的外部表是在MPP数据库内部实现的，而不是作为自定义协议来实现的。对于这些类型，请使用CREATE ROLE或ALTER ROLE命令为每个用户设置CREATEEXTTABLE或NOCREATEEXTTABLE属性。有关语法和示例，请参见[CREATE ROLE](#)。

参数

SELECT

允许从指定表，视图或序列的任何列或列出的特定列中进行SELECT。还

允许使用COPY TO。 引用UPDATE或DELETE中现有的列值也需要此特权。

INSERT

允许将新行INSERT到指定表中。 如果列出了特定的列，则只能在INSERT命令中指定那些列（其他列将接收默认值）。 还允许COPY FROM。

UPDATE

允许UPDATE指定表的任何列或列出的特定列。 SELECT ... FOR UPDATE和SELECT ... FOR SHARE在至少一列上也需要此特权（以及SELECT特权）。 对于序列，此特权允许使用nextval()和setval()函数。

DELETE

允许从指定表中DELETE一行。

REFERENCES

尽管MPP数据库当前不支持外键约束，但仍可接受此关键字。 要创建外键约束，必须在引用列和被引用列上都具有此特权。 可以为表的所有列或仅特定列授予特权。

TRIGGER

允许在指定的表上创建触发器。

Note: MPP数据库不支持触发器。

TRUNCATE

允许对指定表中的所有行进行TRUNCATE。

CREATE

对于数据库，允许在数据库中创建新的schema。

对于schema，允许在schema内创建新对象。 要重命名现有对象，您必须拥有该对象，并对包含的schema具有此特权。

对于表空间，允许在表空间内创建表和索引，并允许创建将表空间作为其默认表空间的数据库。（请注意，撤消此特权不会更改现有对象的位置。）

CONNECT

允许用户连接到指定的数据库。 在连接启动时会检查此特权（除了检查pg_hba.conf施加的任何限制）。

TEMPORARY

TEMP

允许在使用数据库时创建临时表。

EXECUTE

允许使用指定的函数以及使用在该函数之上实现的任何运算符。这是适用于函数的唯一特权类型。（此语法也适用于聚合函数。）

USAGE

对于过程语言，允许使用指定的语言来创建该语言的函数。这是适用于过程语言的唯一特权类型。

对于schema，允许访问指定schema中包含的对象（假设还满足对象自己的特权要求）。从本质上讲，这允许被授予者在schema中查找对象。

对于序列，此特权允许使用curval()和nextval()函数。

对于类型和域，此特权允许在创建表，函数和其他schema对象时使用类型或域。（请注意，它不控制类型的一般“用法”，例如查询中出现的类型的值。它仅防止创建依赖于类型的对象。特权的主要目的是控制哪些用户基于类型创建依赖关系，这可能会阻止所有者稍后更改类型。）

对于外部数据包装器，此特权使被授予者可以使用该外部数据包装器创建新服务器。

对于服务器，此特权使被授予者可以使用服务器创建外部表，还可以创建，更改或删除其自己与该服务器关联的用户的用户映射。

ALL PRIVILEGES

一次授予所有可用特权。PRIVILEGES关键字在MPP数据库中是可选的，尽管严格的SQL要求使用。

PUBLIC

特殊的组级角色，表示将特权授予所有角色，包括以后可能创建的角色。

WITH GRANT OPTION

特权的接收者可以依次将其授予他人。

WITH ADMIN OPTION

角色的成员又可以将角色的成员身份授予其他人。

注解

如果用户对特定列或整个表拥有该特权，则用户可以在该列上执

行SELECT，INSERT等操作。在表级别上授予特权，然后将其撤销一列，这并不像你预想的那样：表级授予不受列级操作的影响。

数据库超级用户可以访问所有对象，而不管对象特权设置如何。该对象的一个例外是视图对象。对视图中引用的表的访问权限是由视图所有者（不是当前用户，即使当前用户是超级用户）的权限决定的。

如果超级用户选择执行GRANT或REVOKE命令，则该命令的执行就像是由受影响对象的所有者执行的一样。特别是，通过此类命令授予的特权似乎已由对象所有者授予。对于角色成员资格，成员资格似乎已由包含它的角色授予。

GRANT和REVOKE也可以由不是受影响对象的所有者完成，但要求是拥有该对象的角色成员，或者是拥有对该对象的WITH GRANT OPTION特权的角色的成员来完成。在这种情况下，特权将被记录为已由实际拥有对象或拥有WITH GRANT OPTION特权的角色授予。

授予表权限不会自动将权限扩展到该表使用的任何序列，包括与SERIAL列绑定的序列。序列的权限必须单独设置。

GRANT命令不能用于为协议file，gpfdist或gpfdists设置特权。这些协议在MPP数据库内部实现。而是使用CREATE ROLE或ALTER ROLE命令来设置角色的CREATEEXTTABLE属性。

使用psql的\dp元命令来获取有关表和列的现有特权的信息。您还可以使用其他\d元命令来显示非表对象的特权。

示例

向表mytable上的所有角色授予插入权限：

```
GRANT INSERT ON mytable TO PUBLIC;
```

将所有可用的特权授予在topten视图中的角色sally。 请注意，如果上述内容的确会由超级用户或topten的所有者执行， 将会授予所有特权，当由其他人执行时，它只会授予那些授予角色具有授予选项的那些权限。

```
GRANT ALL PRIVILEGES ON topten TO sally;
```

将角色admins的成员资格授予用户joe：

```
GRANT admins TO joe;
```

兼容性

PRIVILEGES关键字在SQL标准中是必需的，但在MPP数据库中是可选的。SQL标准不支持为每个命令在多个对象上设置特权。

MPP数据库允许对象所有者撤消他们自己的普通特权： 例如，表所有者可以通过撤消其自己的INSERT，UPDATE，DELETE和TRUNCATE特权使该表对自己只读。 根据SQL标准，这是不可能的。 MPP数据库将所有者的特权视为由所有者授予了所有者；因此他们也可以撤销它们。 在SQL标准中，所有者的特权由假定的`system`实体授予。

SQL标准为其他类型的对象提供USAGE特权：字符集，排序规则，翻译。

在SQL标准中，序列仅具有USAGE特权，该特权控制NEXT VALUE FOR表达式的使用，该表达式等效于MPP数据库中的nextval函数。 序列特权SELECT和UPDATE是MPP数据库扩展。 对currval函数应用序列USAGE特权也是MPP数据库扩展（函数本身也是）。

数据库，表空间，模式和语言的特权是MPP数据库扩展。

另见

[REVOKE](#), [CREATE ROLE](#), [ALTER ROLE](#)

Parent topic: [SQL Command Reference](#)

INSERT

在表中创建新行。

概要

```
[ WITH [ RECURSIVE ] with_query [, ...] ]  
INSERT INTO table [( column [, ...] )]  
  {DEFAULT VALUES | VALUES ( {expression | DEFAULT} [, ...] ) [, ...] | query}  
  [RETURNING * | output_expression [[AS] output_name] [, ...]]
```

描述

INSERT将新行插入表中。 可以插入由值表达式指定的一或多个行，或查询产生的零或多个行。

目标列名称可以按任何顺序列出。 如果根本没有给出任何列名列表，则默认为表中的列按声明的顺序排列。 VALUES子句或查询提供的值与从左到右的显式

或隐式列列表相关联。

在显式或隐式列列表中不存在的每个列都将填充一个默认值，该默认值可以是其声明的默认值，如果没有默认值，则为null。

如果任何列的表达式的数据类型都不正确，则将尝试自动类型转换。

可选的RETURNING子句使INSERT根据实际插入的每一行计算并返回值。这对于获取默认提供的值（例如序列号）很有用。但是，允许使用表列的任何表达式。RETURNING列表的语法与SELECT的输出列表的语法相同。

您必须对表具有INSERT特权才能插入表中。指定列列表后，仅需要对列出的列具有INSERT特权。使用RETURNING子句需要RETURNING中提到的所有列都具有SELECT特权。如果提供查询以插入查询中的行，则必须对查询中引用的任何表或列具有SELECT特权。

输出

成功完成后，INSERT命令将返回以下形式的命令标记：

```
INSERT oid count
```

*count*是插入的行数。如果*count*恰好为1，并且目标表具有OID，则*oid*是分配给插入行的OID。否则，*oid*为零。

参数

with_query

WITH子句允许您指定一个或多个子查询，这些子查询可以在INSERT查询中按名称引用。

对于包含WITH子句的INSERT命令，该子句只能包含SELECT语句，而WITH子句不能包含数据修改命令（INSERT，UPDATE或DELETE）。

查询（SELECT语句）也可能包含WITH子句。在这种情况下，可以在INSERT查询中引用两套 *with_query*，但是第二套优先，因为它的嵌套更紧密。

更多信息，请参考[WITH查询（公用表表达式）](#)和[SELECT](#)。

table

现有表的名称（可以由schema限定）。

column

表中列的名称。如果需要，可以使用子字段名称或数组下标来限定列名称。（仅插入到复合列的某些字段中，未指定的字段为空。）

DEFAULT VALUES

所有列均将填充其默认值。

expression

要分配给相应列的表达式或值。

DEFAULT

相应的列将填充其默认值。

query

提供要插入的行的查询（SELECT语句）。有关语法的说明，请参见SELECT语句。

output_expression

插入每行后，由INSERT命令计算并返回的表达式。该表达式可以使用表的任何列名称。写入*返回插入行的所有列。

output_name

用于返回的列的名称。

注解

要将数据插入分区表中，请指定根分区表，即使用CREATE TABLE命令创建的表。您还可以在INSERT命令中指定分区表的叶子表。如果数据对于指定的叶子表无效，则返回错误。不支持在INSERT命令中指定不是叶子表的子表。不支持在分区表的任何子表上执行其他DML命令，例如UPDATE和DELETE。这些命令必须在根分区表（使用CREATE TABLE命令创建的表）上执行。

对于追加优化表，MPP数据库最多可将127个并发INSERT事务插入单个追加优化表中。

对于可写的S3外部表，INSERT操作将上传到已配置的S3存储桶中的一个或多个文件，如[s3:// 协议](#)中所述。按Ctrl-c取消INSERT，并停止上传到S3。

示例

在表films中插入一行：

```
INSERT INTO films VALUES ('UA502', 'Bananas', 105,  
'1971-07-13', 'Comedy', '82 minutes');
```

在此示例中，省略了length列，因此它将具有默认值：

```
INSERT INTO films (code, title, did, date_prod, kind) VALUES  
( 'T_601', 'Yojimbo', 106, '1961-06-16', 'Drama');
```

本示例对date_prod列使用DEFAULT子句，而不是指定值：

```
INSERT INTO films VALUES ('UA502', 'Bananas', 105, DEFAULT,  
'Comedy', '82 minutes');
```

要插入完全由默认值组成的行：

```
INSERT INTO films DEFAULT VALUES;
```

要使用多行VALUES语法插入多行：

```
INSERT INTO films (code, title, did, date_prod, kind) VALUES  
('B6717', 'Tampopo', 110, '1985-02-10', 'Comedy'),  
('HG120', 'The Dinner Game', 140, DEFAULT, 'Comedy');
```

本示例从表films中向表films中插入一些行，其列布局与films相同：

```
INSERT INTO films SELECT * FROM tmp_films WHERE date_prod <  
'2004-05-07';
```

在表distributors中插入一行，返回由`DEFAULT`子句生成的序列号：

```
INSERT INTO distributors (did, dname) VALUES (DEFAULT, 'XYZ Widgets')  
RETURNING did;
```

兼容性

INSERT符合SQL标准。标准不允许列名列表被省略，但不是所有的列都由VALUES子句或查询填充的情况。

SELECT记录了`query`子句的可能限制。

另见

[COPY](#), [SELECT](#), [CREATE EXTERNAL TABLE](#), [s3:// 协议](#)

Parent topic: [SQL Command Reference](#)

LOAD

加载或重新加载共享库文件。

概要

```
LOAD 'filename'
```

描述

此命令将共享库文件加载到MPP数据库服务器地址空间中。如果文件先前已加载，则首先将其卸载。此命令主要用于卸载和重新加载自服务器首次加载以来已更改的共享库文件。要使用共享库，需要使用CREATE FUNCTION命令声明其中的函数。

文件名的指定方法与CREATE FUNCTION中共享库名的指定方法相同。特别是，可能依赖于搜索路径和系统标准共享库文件名扩展的自动添加。

请注意，在MPP数据库中，共享库文件（.so文件）必须位于MPP数据库阵列（master，segments和mirrors）中每个主机的相同路径。

只有数据库超级用户可以加载共享库文件。

参数

filename

共享库文件的路径和文件名。此文件必须存在于MPP数据库阵列中所有主

机上的相同位置。

示例

加载共享库文件：

```
LOAD '/usr/local/mpp-db/lib/myfuncs.so';
```

兼容性

LOAD是MPP数据库扩展。

另见

[CREATE FUNCTION](#)

Parent topic: [SQL Command Reference](#)

LOCK

锁表。

概要

```
LOCK [TABLE] [ONLY] name [ * ] [, ...] [IN lockmode MODE] [NOWAIT]
```

其中`lockmode`是以下之一：

```
ACCESS SHARE | ROW SHARE | ROW EXCLUSIVE | SHARE UPDATE EXCLUSIVE  
| SHARE | SHARE ROW EXCLUSIVE | EXCLUSIVE | ACCESS EXCLUSIVE
```

描述

LOCK TABLE获取表级别的锁，必要时等待释放任何冲突的锁。如果指定了NOWAIT，则LOCK TABLE不会等待获取所需的锁：如果无法立即获取，则命令将中止并发出错误。一旦获得，该锁将在当前事务的其余部分保持。没有UNLOCK TABLE命令；锁始终在事务结束时释放。

当自动获取引用表的命令的锁时，MPP数据库始终使用限制最小的锁模式。LOCK TABLE提供了您可能需要更多限制性锁定的情况。例如，假设应用程序在`Read Committed`隔离级别上运行事务，并且需要确保表中的数据在事务期间保持稳定。为此，您可以在查询之前在表上获得SHARE锁定模式。这将防止并发数据更改，并确保后续读取表时都能看到已提交数据的稳定视图，因为SHARE锁定模式与写入者获取的ROW EXCLUSIVE锁冲突，并且您的LOCK TABLE `name IN SHARE MODE`语句将等待直到任何并发持有者ROW EXCLUSIVE模式的锁将锁定提交或回滚。因此，一旦获得了锁，就不会有未提交的未完成的写操作。此外，在您释放锁之前，任何人都无法开始。

为了在REPEATABLE READ或SERIALIZABLE隔离级别下运行事务时达到类似的效果，必须在执行任何SELECT或数据修改语句之前执行LOCK TABLE语句。当REPEATABLE READ或SERIALIZABLE事务的数据视图的第一个SELECT或数据修改语句开始时，将被冻结。事务中稍后的LOCK TABLE仍将阻止并发写入 - 但不能确保事务读取的内容与最新的提交值相对应。

如果此类事务要更改表中的数据，则应使用SHARE ROW EXCLUSIVE锁定模式而不是SHARE模式。这样可以确保一次仅运行一个这种类型的事务。否则，可能会导致死锁：两个事务可能都同时获取SHARE模式，然后又无法获取ROW EXCLUSIVE模式来实际执行其更新。请注意，事务自身的锁永远不会发生冲突，因此，当事务拥有SHARE模式时，它可以获取ROW EXCLUSIVE模式，但如果其他人拥有SHARE模式，则不行。为避免死锁，请确保所有事务以相同的顺序获取对相同对象的锁定，并且如果单个对象涉及多个锁定模式，则事务应始终首先获取限制性最强的模式。

参数

name

要锁定的现有表的名称（可以是schema限定）。如果指定ONLY，则仅锁定该表。如果未指定ONLY，则表及其所有子表（如果有）将被锁定。

（可选）可以在表名称后指定*，以明确指示包括子表。

如果给出了多个表，则表将按照LOCK TABLE命令中指定的顺序一张一张地锁定。

lockmode

锁定模式指定与该锁定冲突的锁定。如果未指定锁定模式，则使用限制最大的ACCESS EXCLUSIVE模式。锁定方式如下：

- ACCESS SHARE — 仅与ACCESS EXCLUSIVE锁定模式冲突。
SELECT命令在引用的表上获得此模式的锁定。通常，任何仅读取表而不修改表的查询都将获得此锁定模式。
- ROW SHARE — 与EXCLUSIVE和ACCESS EXCLUSIVE锁定模式冲突。SELECT FOR SHARE命令自动在目标表上获得此模式的锁定（除了在已引用但未选择FOR SHARE的任何其他表上的ACCESS SHARE锁定之外）。
- ROW EXCLUSIVE — 与SHARE，SHARE ROW EXCLUSIVE，EXCLUSIVE和ACCESS EXCLUSIVE锁定模式冲突。
INSERT和COPY命令自动在目标表上获取此锁定模式（除了对任何其他引用表的ACCESS SHARE锁定外），请参见[注解](#)。

- **SHARE UPDATE EXCLUSIVE** — 与**SHARE UPDATE EXCLUSIVE**，**SHARE**，**SHARE ROW EXCLUSIVE**，**EXCLUSIVE**和**ACCESS EXCLUSIVE**锁定模式冲突。此模式可防止表发生并发schema更改和VACUUM运行。由VACUUM（无FULL）在堆表和ANALYZE上获取。
- **SHARE** — 与**ROW EXCLUSIVE**，**SHARE UPDATE EXCLUSIVE**，**SHARE ROW EXCLUSIVE**，**EXCLUSIVE**和**ACCESS EXCLUSIVE**锁定模式冲突。此模式可防止表发生并发数据更改。由CREATE INDEX自动获取。
- **SHARE ROW EXCLUSIVE** — 与**ROW EXCLUSIVE**，**SHARE UPDATE EXCLUSIVE**，**SHARE**，**SHARE ROW EXCLUSIVE**，**EXCLUSIVE**和**ACCESS EXCLUSIVE**锁定模式冲突。任何MPP数据库命令都不会自动获取此锁定模式。
- **EXCLUSIVE** — 与**ROW SHARE**，**ROW EXCLUSIVE**，**SHARE UPDATE EXCLUSIVE**，**SHARE**，**SHARE ROW EXCLUSIVE**，**EXCLUSIVE**和**ACCESS EXCLUSIVE**锁定模式冲突。此模式仅允许并发的ACCESS SHARE锁定，即，只有从表中读取的数据才能与持有该锁定模式的事务并行进行。对于MPP数据库中的UPDATE，SELECT FOR UPDATE和DELETE，此锁定模式是自动获取的（与常规PostgreSQL相比，限制性更强）。看[注解](#)。
- **ACCESS EXCLUSIVE** — 与所有模式的锁定（**ACCESS SHARE**，**ROW SHARE**，**ROW EXCLUSIVE**，**SHARE UPDATE EXCLUSIVE**，**SHARE**，**SHARE ROW EXCLUSIVE**，**EXCLUSIVE**和**ACCESS EXCLUSIVE**）冲突。这种模式保证了持有者是唯一以任何方式访问表的事务。由ALTER TABLE，DROP TABLE，TRUNCATE，REINDEX，CLUSTER和VACUUM FULL命令自动获取。这是未明确指定模式的LOCK TABLE语句的默认锁定模式。VACUUM（无FULL）也会在处理过程中在追加优化表上短暂获取此锁。

Note: 默认情况下，MPP数据库为堆表上的DELETE，

UPDATE和SELECT FOR UPDATE操作获取表上的EXCLUSIVE锁。启用全局死锁检测器后，堆表上操作的锁定模式为ROW EXCLUSIVE。请参阅[全局死锁检测](#)。

NOWAIT

指定LOCK TABLE不等待任何冲突的锁被释放：如果不等待就无法立即获取指定的锁，则事务中止。

注解

LOCK TABLE ... IN ACCESS SHARE MODE需要对目标表具有SELECT特权。所有其他形式的LOCK都需要表级UPDATE，DELETE或TRUNCATE特权。

LOCK TABLE在事务块之外是无用的：该锁定将仅在LOCK语句完成时就释放。因此，如果在事务块外部使用LOCK，MPP数据库将报告错误。使用BEGIN和END定义事务块。

LOCK TABLE仅处理表级锁，因此涉及ROW的模式名称都是错误的。这些模式名称通常应理解为指示用户获取锁定表中的行级锁定的意图。另外，ROW EXCLUSIVE模式是可共享的表锁。请记住，就LOCK TABLE而言，所有锁定模式都具有相同的语义，只是在哪些模式与哪个模式冲突的规则上有所不同。有关如何获取实际的行级锁的信息，请参见[SELECT](#)参考文档中的FOR UPDATE/FOR SHARE子句。

示例

在执行films_user_comments表中的插入操作时，在films表上获得SHARE锁定：

```
BEGIN WORK;
LOCK TABLE films IN SHARE MODE;
SELECT id FROM films
    WHERE name = 'Star Wars: Episode I - The Phantom Menace';
-- Do ROLLBACK if record was not returned
INSERT INTO films_user_comments VALUES
    (_id_, 'GREAT! I was waiting for it for so long!');
COMMIT WORK;
```

执行删除操作时，对表进行SHARE ROW EXCLUSIVE锁定：

```
BEGIN WORK;
LOCK TABLE films IN SHARE ROW EXCLUSIVE MODE;
DELETE FROM films_user_comments WHERE id IN
    (SELECT id FROM films WHERE rating < 5);
DELETE FROM films WHERE rating < 5;
COMMIT WORK;
```

兼容性

SQL标准中没有LOCK TABLE，而是使用SET TRANSACTION来指定事务的并发级别。MPP数据库也支持这一点。

除了ACCESS SHARE，ACCESS EXCLUSIVE和SHARE UPDATE EXCLUSIVE锁定模式外，MPP数据库锁定模式和LOCK TABLE语法与Oracle中的兼容。

另见

[BEGIN](#), [SET TRANSACTION](#), [SELECT](#)

Parent topic: [SQL Command Reference](#)

MOVE

定位游标。

概要

```
MOVE [ forward_direction [ FROM | IN ] ]
      cursor_name
```

其中 *forward_direction* 可以为空或以下之一：

```
NEXT
FIRST
LAST
ABSOLUTE count
RELATIVE count
count
ALL
FORWARD
FORWARD count
FORWARD ALL
```

描述

MOVE会重新定位游标，而不会检索任何数据。 MOVE的工作方式与[FETCH](#)命令完全相同，只不过它仅定位光标而不返回行。

请注意，由于不支持可滚动光标，因此无法在MPP数据库中向后移动光标位置。 您只能使用MOVE向前移动光标。

输出

成功完成后，MOVE命令将返回以下形式的命令标签：

```
MOVE count
```

count是具有相同参数的FETCH命令将返回的行数（可能为零）。

参数

- forward_direction*
MOVE命令的参数与FETCH命令的参数相同。 有关语法和用法的详细信息，请参阅[FETCH](#)。
- cursor_name*
打开的游标的名称。

示例

-- 开始事务：

```
BEGIN;
```

-- 设置游标：

```
DECLARE mycursor CURSOR FOR SELECT * FROM films;
```

-- 在游标mycursor中向前移动5行：

```
MOVE FORWARD 5 IN mycursor;  
MOVE 5
```

-- 之后获取下一行（第6行）：

```
FETCH 1 FROM mycursor;  
code | title | did | date_prod | kind | len  
-----+-----+-----+-----+-----+-----  
P_303 | 48 Hrs | 103 | 1982-10-22 | Action | 01:37  
(1 row)
```

-- 关闭游标并结束事务：

```
CLOSE mycursor;  
COMMIT;
```

兼容性

SQL标准中没有MOVE语句。

另见

[DECLARE](#) , [FETCH](#) , [CLOSE](#)

Parent topic: [SQL Command Reference](#)

PREPARE

准备要执行的语句。

概要

```
PREPARE name [ (datatype [, ...] ) ] AS statement
```

描述

PREPARE创建一个预编译语句。预编译语句是可用于优化性能的服务器端对象。当执行PREPARE语句时，将解析、分析和重写指定的语句。随后发出EXECUTE命令时，将计划并执行预编译语句。这种分工避免了重复的解析分析工作，同时允许执行计划取决于所提供的特定参数值。

预编译语句可以采用参数：执行语句时将用值替换。创建预编译语句时，请使用\$1，\$2等按位置引用参数。可以选择指定相应的参数数据类型列表。如果未指定参数的数据类型或将其声明为未知，则从首次使用该参数的上下文中推断该

类型（如果可能）。执行该语句时，请在EXECUTE语句中指定这些参数的实际值。

预编译语句仅在当前数据库会话期间有效。会话结束时，预编译语句将废弃，因此必须在重新使用之前重新创建它。这也意味着单个预编译语句不能被多个同时的数据库客户端使用。但是，每个客户端可以创建自己预编译语句来使用。可以使用DEALLOCATE命令手动清除预编译语句。

当使用单个会话执行大量相似的语句时，预备语句具有最大的性能优势。如果语句计划或重写很复杂，那么性能差异将特别显著，例如，查询涉及多个表的连接或需要应用多个规则。如果该语句的计划和重写相对简单，但是执行成本相对较高，则预编译语句的性能优势将不太明显。

参数

name

为此特定的预编译语句提供的任意名称。它在单个会话中必须是唯一的，并随后用于执行或取消分配先前预编译语句。

datatype

预编译语句的参数的数据类型。如果未指定特定参数的数据类型或将其指定为未知，则将从首次使用该参数的上下文中推断出来。要在预编译语句本身中引用参数，请使用\$1，\$2等。

statement

任何SELECT，INSERT，UPDATE，DELETE或VALUES语句。

注解

如果预编译语句执行了足够的时间，则服务器可能最终决定保存并重新使用通用计划，而不是每次都重新计划。如果预编译语句没有参数，这将立即发生；否则，仅当通用计划看起来并不比依赖于特定参数值的计划昂贵得多时，它才会发

生。通常，仅当估计查询的性能对提供的特定参数值相当不敏感时，才会选择通用计划。

要检查MPP数据库用于预编译语句的查询计划，请使用[EXPLAIN](#)。如果正在使用通用计划，它将包含参数符号 n ，而自定义计划将使用当前的实际参数值替代它。

有关查询计划和MPP数据库为此目的收集的统计信息的更多信息，请参阅[ANALYZE](#)文档。

尽管预编译语句的主要目的是避免重复分析和规划语句，但只要语句中使用的数据库对象自上次使用预处理语句以来发生了定义（DDL）更改，MPP都会在使用之前强制重新分析和重新规划语句。同样，如果`search_path`的值从一个更改为下一个，则将使用新的`search_path`重新解析该语句。（后一种行为自MPP 6起是新的。）这些规则使用预编译语句在语义上几乎等同于一次又一次地重新提交相同的查询文本，这在语义上几乎等同于使用预编译语句，但是如果不更改对象定义，则可以提高性能，特别是如果最佳计划在不同用途之间保持不变。语义对等不完美的情况的一个示例是，如果该语句使用不合格的名称引用表，然后在较早出现在`search_path`中的模式中创建了一个具有相同名称的新表，则不会自动重新解析，因为语句中使用的对象没有变化。但是，如果其他更改迫使重新解析，则在随后的使用中引用新表。

您可以通过查询`pg_prepared_statements`系统视图来查看会话中所有可用的预编译语句。

示例

为INSERT语句创建一个预编译语句，然后执行它：

```
PREPARE fooplan (int, text, bool, numeric) AS INSERT INTO
foo VALUES($1, $2, $3, $4);
EXECUTE fooplan(1, 'Hunter Valley', 't', 200.00);
```

为SELECT语句创建一个预编译语句，然后执行它。 请注意，未指定第二个参数的数据类型，因此可以从使用\$2的上下文中推断出它：

```
PREPARE usrrptplan (int) AS SELECT * FROM users u, logs l
WHERE u.usrid=$1 AND u.usrid=l.usrid AND l.date = $2;
EXECUTE usrrptplan(1, current_date);
```

兼容性

SQL标准包括一个PREPARE语句，但仅用于嵌入式SQL。 此版本的PREPARE语句还使用了一些不同的语法。

另见

[EXECUTE , DEALLOCATE](#)

Parent topic: [SQL Command Reference](#)

REASSIGN OWNED

更改数据库角色拥有的数据库对象的所有权。

概要

```
REASSIGN OWNED BY old_role [, ...] TO new_role
```

描述

REASSIGN OWNED将任何*old_role*拥有的数据库对象的所有权更改为*new_role*。

参数

old_role

角色名称。该角色所拥有的当前数据库中所有对象以及所有共享对象（数据库，表空间）的所有权将重新分配给*new_role*。

new_role

将成为受影响对象的新所有者的角色的名称。

注解

REASSIGN OWNED通常用于准备删除一个或多个角色。由于REASSIGN OWNED不会影响其他数据库中的对象，因此通常需要在每个数据库中执行此命令，该数据库包含要删除的角色所拥有的对象。

REASSIGN OWNED要求对源角色和目标角色都具有特权。

DROP OWNED命令是一种替代方法，它仅删除一个或多个角色拥有的所有数据库对象。DROP OWNED仅要求对源角色具有特权。

REASSIGN OWNED命令不会影响授予旧角色的不属于它们的对象的任何特权。使用DROP OWNED撤消那些特权。

示例

将由sally和bob角色拥有的所有数据库对象重新分配给admin ；

```
REASSIGN OWNED BY sally, bob TO admin;
```

兼容性

REASSIGN OWNED命令是MPP数据库扩展。

另见

[DROP OWNED](#), [DROP ROLE](#), [ALTER DATABASE](#)

Parent topic: [SQL Command Reference](#)

REINDEX

重建索引。

概要

```
REINDEX {INDEX | TABLE | DATABASE | SYSTEM} name
```

描述

REINDEX使用存储在索引表中的数据重建索引，替换索引的旧副本。在几种情况下可以使用REINDEX：

- 索引变得肿，也就是说，它包含许多空或几乎为空的页面。在某些罕见的访问模式下，MPP数据库中的B树索引可能会发生这种情况。REINDEX提供了一种通过编写没有死页的新版本索引来减少索引空间消耗的方法。
- 您已经更改了索引的FILLFACTOR存储参数，并希望确保所做的更改完全生效。

参数

INDEX

重新创建指定的索引。

TABLE

重新创建指定表的所有索引。如果该表具有辅助TOAST表，则该表也将重新编制索引。

DATABASE

重新创建当前数据库内的所有索引。共享系统目录上的索引也会被处理。这种形式的REINDEX不能在事务块内执行。

SYSTEM

在当前数据库内的系统目录上重新创建所有索引。包括共享系统目录上的索引。用户表上的索引未处理。这种形式的REINDEX不能在事务块内执行。

name

要重新建立索引的特定索引，表或数据库的名称。索引和表名可以是模式限定的。当前，REINDEX DATABASE和REINDEX SYSTEM只能重新索引当前数据库，因此它们的参数必须与当前数据库的名称匹配。

注解

REINDEX与删除索引和重新创建索引相似，因为索引内容是从头开始重建的。但是，锁定注意事项却大不相同。REINDEX锁定对索引的父表的写入但不锁定读取。它还对正在处理的特定索引采取排他锁，这将阻塞该索引的读取。相反，DROP INDEX会立即对父表加排他锁，从而阻止写入和读取。随后的CREATE INDEX锁定写但不锁定读；由于索引不存在，因此不会进行任何读取尝试，这意味着不会有阻塞，但读操作可能会强制进行高代价的顺序扫描。

重新索引单个索引或表要求是该索引或表的所有者。重新索引数据库需要成为数据库的所有者（请注意，所有者可以因此重建其他用户拥有的表的索引）。当然，超级用户可以随时重新索引任何内容。

REINDEX不会更新索引的reltuples和relpages统计信息。要更新这些统计信息，请在重新建立索引后在表上运行ANALYZE。

如果您怀疑共享的全局系统目录索引已损坏，则只能在MPP utility模式下对它们进行索引。共享索引损坏的典型症状是“索引不是btree”错误，否则服务器将在启动时由于依赖损坏的索引而立即崩溃。在这种情况下，请联系MPP客户支持。

示例

重建单个索引：

```
REINDEX INDEX my_index;
```

重建表my_table上的所有索引：

```
REINDEX TABLE my_table;
```

兼容性

SQL标准中没有REINDEX命令。

另见

[CREATE INDEX](#) , [DROP INDEX](#) , [VACUUM](#)

Parent topic: [SQL Command Reference](#)

RELEASE SAVEPOINT

销毁先前定义的保存点。

概要

```
RELEASE [SAVEPOINT] savepoint_name
```

描述

RELEASE SAVEPOINT 销毁先前在当前事务中定义的保存点。

破坏保存点使其无法用作回滚点，但是它没有其他用户可见的行为。它不会撤消在建立保存点后执行的命令的影响。（为此，请参阅[ROLLBACK TO SAVEPOINT](#)。）在不再需要保存点时对其进行销毁，可能会使系统在事务结束之前回收一些资源。

RELEASE SAVEPOINT 还将销毁在建立指定保存点之后建立的所有保存点。

参数

savepoint_name

要销毁的保存点的名称。

示例

建立并随后销毁一个保存点：

```
BEGIN;  
  INSERT INTO table1 VALUES (3);  
  SAVEPOINT my_savepoint;  
  INSERT INTO table1 VALUES (4);  
  RELEASE SAVEPOINT my_savepoint;  
COMMIT;
```

上述事务将同时插入3和4。

兼容性

该命令符合SQL标准。 该标准指定关键字SAVEPOINT是必需的，但是MPP数据库允许省略它。

另见

[BEGIN](#) , [SAVEPOINT](#) , [ROLLBACK TO SAVEPOINT](#) , [COMMIT](#)

Parent topic: [SQL Command Reference](#)

RESET

将系统配置参数的值恢复为默认值。

概要

RESET *configuration_parameter*

RESET ALL

描述

RESET将系统配置参数恢复为其默认值。 RESET是SET *configuration_parameter* TO DEFAULT的替代写法。

缺省值定义为该参数在当前会话中没有为其发出SET时将具有的值。 该值的实际来源可能是已编译的默认值，主postgresql.conf配置文件，命令行选项或每个数据库或每个用户的默认设置。 有关更多信息，请参见[服务器配置参数](#)。

参数

- configuration_parameter*
系统配置参数的名称。 有关详细信息，请参见[服务器配置参数](#)。
- ALL
将所有可设置的配置参数重置为其默认值。

示例

将statement_mem配置参数设置为其默认值：

```
RESET statement_mem;
```

兼容性

RESET是MPP数据库扩展。

另见

SET

Parent topic: [SQL Command Reference](#)

REVOKE

删除访问权限。

概要

```
REVOKE [GRANT OPTION FOR] { {SELECT | INSERT | UPDATE | DELETE
    | REFERENCES | TRIGGER | TRUNCATE } [, ...] | ALL [PRIVILEGES] }

    ON { [TABLE] table_name [, ...]
        | ALL TABLES IN SCHEMA schema_name [, ...] }
    FROM { [ GROUP ] role_name | PUBLIC } [, ...]
    [CASCADE | RESTRICT]

REVOKE [ GRANT OPTION FOR ] { { SELECT | INSERT | UPDATE
    | REFERENCES } ( column_name [, ...] )
    [, ...] | ALL [ PRIVILEGES ] ( column_name [, ...] ) }
    ON [ TABLE ] table_name [, ...]
    FROM { [ GROUP ] role_name | PUBLIC } [, ...]
    [ CASCADE | RESTRICT ]

REVOKE [GRANT OPTION FOR] { {USAGE | SELECT | UPDATE} [,...]}
    | ALL [PRIVILEGES] }
    ON { SEQUENCE sequence_name [, ...]
        | ALL SEQUENCES IN SCHEMA schema_name [, ...] }
```

```

    | ALL SEQUENCES IN SCHEMA schema_name [, ...] }
FROM { [ GROUP ] role_name | PUBLIC } [, ...]
[CASCADE | RESTRICT]

```

```

REVOKE [GRANT OPTION FOR] { {CREATE | CONNECT
    | TEMPORARY | TEMP} [, ...] | ALL [PRIVILEGES] }
ON DATABASE database_name [, ...]
FROM { [ GROUP ] role_name | PUBLIC } [, ...]
[CASCADE | RESTRICT]

```

```

REVOKE [ GRANT OPTION FOR ]
    { USAGE | ALL [ PRIVILEGES ] }
ON DOMAIN domain_name [, ...]
FROM { [ GROUP ] role_name | PUBLIC } [, ...]
[ CASCADE | RESTRICT ]

```

```

REVOKE [ GRANT OPTION FOR ]
    { USAGE | ALL [ PRIVILEGES ] }
ON FOREIGN DATA WRAPPER fdw_name [, ...]
FROM { [ GROUP ] role_name | PUBLIC } [, ...]
[ CASCADE | RESTRICT ]

```

```

REVOKE [ GRANT OPTION FOR ]
    { USAGE | ALL [ PRIVILEGES ] }
ON FOREIGN SERVER server_name [, ...]
FROM { [ GROUP ] role_name | PUBLIC } [, ...]
[ CASCADE | RESTRICT ]

```

```

REVOKE [GRANT OPTION FOR] {EXECUTE | ALL [PRIVILEGES]}
ON { FUNCTION funcname ( [[argmode] [argname] argtype
    [, ...]] ) [, ...]
    | ALL FUNCTIONS IN SCHEMA schema_name [, ...] }
FROM { [ GROUP ] role_name | PUBLIC } [, ...]
[CASCADE | RESTRICT]

```

```

REVOKE [GRANT OPTION FOR] {USAGE | ALL [PRIVILEGES]}
ON LANGUAGE langname [, ...]
FROM { [ GROUP ] role_name | PUBLIC } [, ...]
[ CASCADE | RESTRICT ]

```

```

REVOKE [GRANT OPTION FOR] { {CREATE | USAGE} [, ...]
    | ALL [PRIVILEGES] }
ON SCHEMA schema_name [, ...]

```

```
FROM { [ GROUP ] role_name | PUBLIC } [, ...]  
[CASCADE | RESTRICT]
```

```
REVOKE [GRANT OPTION FOR] { CREATE | ALL [PRIVILEGES] }  
ON TABLESPACE tablespacename [, ...]  
FROM { [ GROUP ] role_name | PUBLIC } [, ...]  
[CASCADE | RESTRICT]
```

```
REVOKE [ GRANT OPTION FOR ]  
{ USAGE | ALL [ PRIVILEGES ] }  
ON TYPE type_name [, ...]  
FROM { [ GROUP ] role_name | PUBLIC } [, ...]  
[ CASCADE | RESTRICT ]
```

```
REVOKE [ADMIN OPTION FOR] parent_role [, ...]  
FROM [ GROUP ] member_role [, ...]  
[CASCADE | RESTRICT]
```

描述

REVOKE命令从一个或多个角色撤消先前授予的特权。关键字PUBLIC指所有角色的隐式定义组。

有关特权类型的含义，请参见[GRANT](#)命令的描述。

请注意，任何特定角色将具有直接授予它的特权，授予该角色当前成员的任何角色的特权以及授予PUBLIC的特权的总和。因此，例如，从PUBLIC撤消SELECT特权并不一定意味着所有角色都对该对象失去SELECT特权：直接或通过其他角色授予它的那些人仍然拥有该特权。同样，如果PUBLIC或其他成员身份角色仍然具有SELECT权限，则从用户撤消SELECT可能不会阻止该用户使用SELECT。

如果指定了GRANT OPTION FOR，则仅撤销特权的授予选项，而不撤销特权本身。否则，特权和授予选项都将被撤销。

如果某个角色拥有带有“授予”选项的特权并将其授予其他角色，则这些其他角色所拥有的特权称为“从属特权”。如果第一个角色持有的特权或授予选项被撤销，并且存在从属特权，则如果指定了CASCADE，那些从属特权也将被撤销，否则撤销操作将失败。此递归撤销仅影响通过可追溯到此REVOKE命令对象的角色角色链授予的特权。因此，如果还通过其他角色授予了特权，则受影响的角色可以有效地保留特权。

当您撤消对表的特权时，MPP数据库也会撤消对表的每一列的相应列特权（如果有）。另一方面，如果已授予角色某个表的特权，则从单独列中撤消相同的特权将无效。

撤销角色成员身份时，GRANT OPTION改为ADMIN OPTION，但行为类似。

参数

参见[GRANT](#)。

注解

用户只能撤消该用户直接授予的那些特权。例如，如果用户A向用户B授予了具有授予选项的特权，而用户B又将其授予了用户C，则用户A无法直接从C撤消该特权。而是，用户A可以撤消用户B的授予选项，并使用CASCADE选项，以便依次从用户C撤消特权。例如，如果A和B都向C授予相同的特权，则A可以撤消他自己的授予，但不能撤消B的授予，因此C实际上仍然有特权。

当对象的非所有者尝试REVOKE对对象的特权时，如果用户对对象没有任何特权，则该命令将彻底失败。只要有某些特权，该命令就会继续执行，但是它将仅撤销用户具有授予选项的那些特权。如果未保留任何授予选项，则REVOKE ALL PRIVILEGES形式将发出警告消息，而如果未保留针对命令中明确指定的任何特权的授予选项，则其他形式将发出警告。（原则上，这些声明也适用于

对象所有者，但是由于MPP数据库始终将所有者视为拥有所有授予选项的对象，因此永远不会发生这种情况。)

如果超级用户选择发出GRANT或REVOKE命令，则MPP数据库将执行该命令，就像它是由受影响对象的所有者发出的一样。由于所有特权最终都来自对象所有者（可能间接地通过授予选项链），因此超级用户可以撤消所有特权，但是如上所述，这可能需要使用CASCADE。

REVOKE也可以由不是受影响对象的所有者，但是拥有该对象的角色成员，或者是持有对该对象具有WITH GRANT OPTION特权的角色成员来调用。在这种情况下，MPP数据库将执行该命令，就像它是由实际拥有该对象或拥有WITH GRANT OPTION特权的用户所包含的角色发出的那样。例如，如果表t1由角色g1拥有，而角色u1是它的成员，则u1可以撤销t1上记录为由g1授予的特权。这包括u1以及角色g1的其他成员所做的授权。

如果执行REVOKE的角色通过多个角色成员路径间接持有特权，则未指定将使用哪个包含角色来执行命令。在这种情况下，最佳实践是使用SET ROLE来成为要REVOKE的特定角色。否则，可能会导致撤销您想要的特权以外的特权，或者根本不撤销任何特权。

使用psql的\dpp元命令来获取有关表和列的现有特权的信息。您还可以使用其他\d元命令来显示非表对象的特权。

示例

撤销public对表films的插入特权：

```
REVOKE INSERT ON films FROM PUBLIC;
```

从topten视图中的角色sally撤消所有特权。请注意，这实际上意味着撤销当前角色（如果不是超级用户）授予的所有特权。

```
REVOKE ALL PRIVILEGES ON topten FROM sally;
```

从用户joe撤消角色admins的成员资格：

```
REVOKE admins FROM joe;
```

兼容性

[GRANT](#)命令的兼容性说明也适用于REVOKE。

根据标准，需要RESTRICT或CASCADE，但是默认情况下MPP数据库假定RESTRICT。

另见

[GRANT](#)

Parent topic: [SQL Command Reference](#)

ROLLBACK

中止当前事务。

概要

ROLLBACK [WORK | TRANSACTION]

描述

ROLLBACK回滚当前事务，并使该事务进行的所有更新都被丢弃。

参数

WORK

TRANSACTION

可选关键字。它们没有作用。

注解

使用COMMIT成功提交当前事务。

如果不在事务内部，则发出ROLLBACK不会造成任何危害，但是会引发警告消息。

示例

要舍弃当前事务中的所有更改：

```
ROLLBACK;
```

兼容性

SQL标准仅指定两种形式ROLLBACK和ROLLBACK WORK。否则，此命令完全符合要求。

另见

[BEGIN](#) , [COMMIT](#) , [SAVEPOINT](#) , [ROLLBACK TO SAVEPOINT](#)

Parent topic: [SQL Command Reference](#)

ROLLBACK TO SAVEPOINT

将当前事务回滚到保存点。

概要

```
ROLLBACK [WORK | TRANSACTION] TO [SAVEPOINT]
        savepoint_name
```

描述

此命令将回滚在建立保存点之后执行的所有命令。该保存点保持有效，可以稍后在需要时回滚。

ROLLBACK TO SAVEPOINT隐式销毁在指定保存点之后建立的所有保存点。

参数

WORK

TRANSACTION

可选关键字。它们没有作用。

savepoint_name

要回滚的保存点的名称。

注解

使用RELEASE SAVEPOINT销毁一个保存点，而不会丢弃建立该保存点后执行的命令的效果。

指定尚未建立的保存点名称是错误的。

游标相对于保存点具有某种非事务性的行为。当回滚保存点时，在保存点内部打开的所有游标都将关闭。如果先前打开的游标受保存点内的FETCH命令影

响，该保存点随后被回滚，则该游标仍将保持在FETCH指向的位置（即，由FETCH引起的光标运动不会回滚）。关闭游标也不会通过回滚来撤消。但是，如果游标的查询所引起的其他副作用（例如查询所调用的volatile函数的副作用）在保存点期间发生，然后又回滚，则这些副作用也会被回滚。由于其执行导致事务中止的游标处于无法执行状态，因此尽管可以使用ROLLBACK TO SAVEPOINT还原该事务，但不能再使用该游标。

示例

要撤消在建立my_savepoint之后执行的命令的影响：

```
ROLLBACK TO SAVEPOINT my_savepoint;
```

光标位置不受保存点回滚的影响：

```
BEGIN;
DECLARE foo CURSOR FOR SELECT 1 UNION SELECT 2;
SAVEPOINT foo;
FETCH 1 FROM foo;
column
-----
      1
ROLLBACK TO SAVEPOINT foo;
FETCH 1 FROM foo;
column
-----
      2
COMMIT;
```

兼容性

SQL标准指定关键字SAVEPOINT是必需的，但是MPP数据库（和Oracle）允许省略它。SQL在ROLLBACK之后只允许WORK，而不允许TRANSACTION作为干扰词。此外，SQL具有可选子句AND [NO] CHAIN，MPP数据库当前不支持该子句。在其他方面，此命令符合SQL标准。

另见

[BEGIN](#) , [COMMIT](#) , [SAVEPOINT](#) , [RELEASE SAVEPOINT](#) , [ROLLBACK](#)

Parent topic: [SQL Command Reference](#)

SAVEPOINT

在当前事务中定义一个新的保存点。

概要

```
SAVEPOINT savepoint_name
```

描述

SAVEPOINT在当前事务中建立一个新的保存点。

保存点是事务内部的特殊标记，它允许回滚在建立事务后执行的所有命令，从而将事务状态恢复到保存点时的状态。

参数

savepoint_name

新保存点的名称。

注解

使用 [ROLLBACK TO SAVEPOINT](#) 回滚到保存点。使用 [RELEASE SAVEPOINT](#) 销毁一个保存点，并在建立保存点后保持命令执行的效果。

只能在事务块内部建立保存点。事务中可以定义多个保存点。

示例

要建立一个保存点，然后撤消建立保存点后执行的所有命令的效果：

```
BEGIN;  
    INSERT INTO table1 VALUES (1);  
    SAVEPOINT my_savepoint;  
    INSERT INTO table1 VALUES (2);  
    ROLLBACK TO SAVEPOINT my_savepoint;  
    INSERT INTO table1 VALUES (3);  
COMMIT;
```

上面的事务将插入值1和3，但不插入2。

要建立并随后销毁一个保存点：

```
BEGIN;  
    INSERT INTO table1 VALUES (3);  
    SAVEPOINT my_savepoint;  
    INSERT INTO table1 VALUES (4);  
    RELEASE SAVEPOINT my_savepoint;  
COMMIT;
```

上述事务将同时插入3和4。

兼容性

建立另一个具有相同名称的保存点时，SQL要求自动删除一个保存点。在MPP数据库中，保留了旧的保存点，尽管在回滚或释放时仅使用较新的保存点。

（释放较新的保存点将使较旧的保存点再次可用于[ROLLBACK TO SAVEPOINT](#)和[RELEASE SAVEPOINT](#)。） 否则，SAVEPOINT完全符合SQL。

另见

[BEGIN](#) , [COMMIT](#) , [ROLLBACK](#) , [RELEASE SAVEPOINT](#) , [ROLLBACK TO SAVEPOINT](#)

Parent topic: [SQL Command Reference](#)

SELECT

从表或视图中检索行。

概要

```
[ WITH [ RECURSIVE1 ] with_query [, ...] ]  
SELECT [ ALL | DISTINCT [ ON (expression [, ...])] ]  
    * | expression[[AS] output_name] [, ...]  
[FROM from_item [, ...]]  
[WHERE condition]  
[GROUP BY grouping_element [, ...]]  
[HAVING condition [, ...]]  
[WINDOW window_name AS (window_definition) [, ...] ]  
[{UNION | INTERSECT | EXCEPT} [ALL | DISTINCT] select]  
[ORDER BY expression [ASC | DESC | USING operator] [NULLS {FIRST | LAST}] [, ...]]  
[LIMIT {count | ALL}]  
[OFFSET start [ ROW | ROWS ] ]  
[FETCH { FIRST | NEXT } [ count ] { ROW | ROWS } ONLY]  
[FOR {UPDATE | NO KEY UPDATE | SHARE | KEY SHARE} [OF table_name [, ...]] [NOV  
  
TABLE { [ ONLY ] table_name [ * ] | with_query_name }
```

其中*with_query*是：

```
with_query_name [( column_name [, ...] )] AS ( select | values | insert | update | delete )
```

其中*from_item*可以是以下之一：

```

[ONLY] table_name [ * ] [ [ AS ] alias [ ( column_alias [, ...] ) ] ]
( select ) [ AS ] alias [ ( column_alias [, ...] ) ]
with_query_name [ [ AS ] alias [ ( column_alias [, ...] ) ] ]
function_name ( [ argument [, ...] ] )
    [ WITH ORDINALITY ] [ [ AS ] alias [ ( column_alias [, ...] ) ] ]
function_name ( [ argument [, ...] ] ) [ AS ] alias ( column_definition [, ...] )
function_name ( [ argument [, ...] ] ) AS ( column_definition [, ...] )
ROWS FROM( function_name ( [ argument [, ...] ] ) [ AS ( column_definition [, ...] ) ] [, ...] )
    [ WITH ORDINALITY ] [ [ AS ] alias [ ( column_alias [, ...] ) ] ]
from_item [ NATURAL ] join_type from_item
    [ ON join_condition | USING ( join_column [, ...] ) ]

```

其中*grouping_element*可以是以下之一：

```

()
expression
ROLLUP (expression [,...])
CUBE (expression [,...])
GROUPING SETS ((grouping_element [, ...]))

```

其中*window_definition*是：

```

[existing_window_name]
[PARTITION BY expression [, ...]]
[ORDER BY expression [ASC | DESC | USING operator]
[NULLS {FIRST | LAST}] [, ...]]
[{ RANGE | ROWS } frame_start
    | {RANGE | ROWS } BETWEEN frame_start AND frame_end]

```

其中*frame_start*和*frame_end*可以是以下之一：

```
UNBOUNDED PRECEDING  
value  
PRECEDING  
CURRENT ROW  
value  
FOLLOWING  
UNBOUNDED FOLLOWING
```

注解：¹RECURSIVE关键字是一个Beta功能。

描述

SELECT从零个或多个表中检索行。SELECT的一般处理如下：

1. WITH子句中的所有查询均被计算。这些可以有效地用作可以在FROM列表中引用的临时表。
2. 将计算FROM列表中的所有元素。（FROM列表中的每个元素都是真实表或虚拟表。）如果FROM列表中指定了多个元素，则它们将交叉连接在一起。
3. 如果指定了WHERE子句，则从输出中排除所有不满足该条件的行。
4. 如果指定了GROUP BY子句，或者存在聚合函数调用，则将输出组合为与一个或多个值匹配的行组，并计算聚合函数的结果。如果存在HAVING子句，它将排除不满足给定条件的组。
5. 实际的输出行是使用SELECT输出表达式为每个选定的行或行组计算的。
6. SELECT DISTINCT从结果中消除重复的行。SELECT DISTINCT ON消除与所有指定表达式匹配的行。SELECT ALL（默认）将返回所有候选行，包括重复项。
7. 如果指定了窗口表达式（和可选的WINDOW子句），则根据位置（行）或基

于值的（范围）窗口框架来组织输出。

8. 使用每个选定行的SELECT输出表达式来计算实际的输出行。
9. 使用运算符UNION，INTERSECT和EXCEPT，可以组合多个SELECT语句的输出以形成单个结果集。 UNION运算符返回一个或两个结果集中的所有行。 INTERSECT运算符返回完全位于两个结果集中的所有行。 EXCEPT运算符返回第一个结果集中的行，而不是第二个结果集中的行。 在所有三种情况下，除非指定了ALL，否则将消除重复的行。 可以添加噪声字DISTINCT来明确指定消除重复行。 请注意，DISTINCT是此处的默认行为，即使ALL是SELECT本身的默认行为。
0. 如果指定了ORDER BY子句，则返回的行将以指定的顺序排序。 如果未给出ORDER BY，则以系统最先找到的顺序返回行。
1. 如果指定了LIMIT（或FETCH FIRST）或OFFSET子句，则SELECT语句仅返回结果行的子集。
2. 如果指定了FOR UPDATE，FOR NO KEY UPDATE，FOR SHARE或FOR KEY SHARE，则SELECT语句将锁定整个表以防止并发更新。

您必须对SELECT命令中使用的每一列都具有SELECT特权。 使用FOR NO KEY UPDATE，FOR UPDATE，FOR SHARE或FOR KEY SHARE也需要UPDATE特权（对于这样选择的每个表的至少一列）。

参数

WITH子句

可选的WITH子句允许您指定一个或多个子查询，这些子查询可以在主查询中按名称引用。 子查询在主查询期间有效地充当临时表或视图。 每个子查询可以是SELECT，INSERT，UPDATE或DELETE语句。 在用WITH编写数据修改语句（INSERT，UPDATE或DELETE）时，通常会包含RETURNING子句。 它

是RETURNING的输出，而不是该语句修改的基础表，形成了由主查询读取的临时表。如果省略RETURNING，则该语句仍将执行，但是不会产生任何输出，因此主查询无法将其引用为表。

对于包含WITH子句的SELECT命令，该子句最多可以包含一个用于修改表数据的子句（INSERT，UPDATE或DELETE命令）。

必须为WITH子句中的每个查询指定一个无schema限定的*with_query_name*。（可选）可以指定列名列表；如果省略列名列表，则从子查询中推断出名称。主查询和WITH查询都（理论上）同时执行。

如果指定了RECURSIVE，则它允许SELECT子查询按名称引用自己。这样的子查询具有一般形式

```
non_recursive_term UNION [ALL | DISTINCT] recursive_term
```

递归自引用出现在UNION的右侧。每个查询仅允许一个递归自引用。不支持递归数据修改语句，但是您可以在数据修改语句中使用递归SELECT查询的结果。

如果指定了RECURSIVE关键字，则无需对WITH查询进行排序：一个查询可以引用列表中后面的另一个查询。但是，不支持循环引用或相互递归。

如果没有RECURSIVE关键字，则WITH查询只能引用WITH列表中较早的同级WITH查询。

WITH RECURSIVE限制。不支持这些项目：

- 一个递归WITH子句，在*recursive_term*中包含以下内容。
 - 具有自引用的子查询
 - DISTINCT 子句
 - GROUP BY 子句
 - 窗口函数

- 递归WITH子句，其中`with_query_name`是集合操作的一部分。

以下是集合操作限制的示例。该查询返回错误，因为集合操作UNION包含对表foo的引用。

```
WITH RECURSIVE foo(i) AS (  
  SELECT 1  
  UNION ALL  
  SELECT i+1 FROM (SELECT * FROM foo UNION SELECT 0) bar  
)  
SELECT * FROM foo LIMIT 5;
```

由于集合操作UNION没有对CTE foo的引用，因此允许此递归CTE。

```
WITH RECURSIVE foo(i) AS (  
  SELECT 1  
  UNION ALL  
  SELECT i+1 FROM (SELECT * FROM bar UNION SELECT 0) bar, foo  
  WHERE foo.i = bar.a  
)  
SELECT * FROM foo LIMIT 5;
```

WITH查询的一个关键属性是，即使主查询多次引用它们，一次执行主查询也只会对它们进行一次评估。特别是，无论主查询是读取全部输出还是输出任何内容，都保证数据修改语句仅执行一次。

主查询和WITH查询都（理论上）同时执行。这意味着，除了通过读取其RETURNING输出之外，不能从查询的其他部分看到WITH中的数据修改语句的效果。如果两个这样的数据修改语句试图修改同一行，则结果不确定。

有关更多信息，请参见[WITH查询（公用表表达式）](#)。[WITH查询（公用表表达式）](#)

SELECT列表

SELECT列表（在关键字SELECT和FROM之间）指定用于形成SELECT语句输

出行的表达式。表达式可以（通常）引用FROM子句中计算的列。

SELECT列表中的`expression`可以是常量值，列引用，运算符调用，函数调用，聚合表达式，窗口表达式，标量子查询等。可以将许多结构体归类为表达式，但不遵循任何常规语法规则。这些通常具有函数或运算符的语义。有关SQL值表达式和函数调用的信息，请参阅*MPP数据库管理员手册*中的“查询数据”。

就像在表中一样，SELECT的每个输出列都有一个名称。在简单的SELECT中，此名称仅用于标记要显示的列，但是当SELECT是较大查询的子查询时，该名称在较大查询中被视为由该子查询生成的虚拟表的列名。要指定用于输出列的名称，请在该列的表达式之后写入AS `output_name`。（您可以省略AS，但只有在所需的输出名称与任何SQL关键字都不匹配时才可以使用。为防止将来可能再添加关键字，您始终可以写AS或在输出名称中用双引号。）如果不指定列名称，MPP数据库自动选择一个名称。如果列的表达式是简单的列引用，则所选名称与该列的名称相同。在更复杂的情况下，可以使用函数或类型名称，或者系统可能会依赖生成的名称（例如?column?或columnN）。

输出列的名称可用于引用ORDER BY和GROUP BY子句中的列的值，但不能用于WHERE或HAVING子句中。在那里，您必须写出表达式。

可以将*而不是表达式写到输出列表，作为所选行的所有列的简写。另外，您可以编写`table_name.*`作为仅来自该表的列的简写。在这种情况下，无法使用AS指定新名称。输出列名称将与表列名称相同。

DISTINCT子句

如果指定了SELECT DISTINCT，则将从结果集中删除所有重复的行（每组重复项中保留一行）。SELECT ALL则相反：所有行都保留；这是默认值。

SELECT DISTINCT ON (`expression` [, ...])仅保留给定表达式等于的每组行的第一行。使用与ORDER BY相同的规则来解释DISTINCT ON表达式（请参见上文）。请注意，除非使用ORDER BY来确保所需的行最先出现，否则每个集合的“第一行”都是不可预测的。例如：

```
SELECT DISTINCT ON (location) location, time, report
FROM weather_reports
ORDER BY location, time DESC;
```

检索每个位置的最新天气报告。但是，如果我们没有使用ORDER BY强制每个位置的时间值按降序排列，那么我们将从每个位置不可预测的时间获得报告。

DISTINCT ON表达式必须与最左边的ORDER BY表达式匹配。ORDER BY子句通常将包含附加表达式，这些表达式确定每个DISTINCT ON组中所需的行优先级。

FROM子句

FROM子句为SELECT指定一个或多个源表。如果指定了多个源，则结果为所有源的笛卡尔乘积（交叉连接）。但通常会添加限定条件（通过WHERE），以将返回的行限制为笛卡尔乘积的一小部分。FROM子句可以包含以下元素：

table_name

现有表或视图的名称（可以有schema修饰）。如果指定ONLY，则仅扫描该表。如果未指定ONLY，则扫描该表及其所有子表（如果有）。

alias

包含别名的FROM项目的替代名称。别名用于简洁起见或用于消除自连接的歧义（多次扫描同一张表）。提供别名后，它将完全隐藏表或函数的实际名称。例如，在给定FROM foo AS f的情况下，SELECT的其余部分必须将此FROM项目引用为f而不是foo。如果写入了别名，则还可以写入列别名列表以为表的一个或多个列提供替代名称。

select

子SELECT可以出现在FROM子句中。这就像在单个SELECT命令期间将其输出创建为临时表一样。请注意，子SELECT必须用括号括起来，并且必须为其提供别名。[VALUES](#)命令也可以在这里使用。有关在MPP数据库中使用相关子选择的限制，请参阅[兼容性](#)部分中的“非标准子句”。

with_query_name

通过指定其*with_query_name*在FROM子句中引用*with_query*，就像该名称是表名一样。*with_query_name*不能包含schema限定符。可以使用与

表相同的方式提供别名。

*with_query*隐藏用于主查询目的的同名表。如有必要，可以通过使用 *schema* 限定表名来引用相同名称的表。

function_name

函数调用可以出现在FROM子句中。（这对于返回结果集的函数特别有用，但是可以使用任何函数。）这就像在单个SELECT命令期间将其输出创建为临时表一样。也可以使用别名。如果编写了别名，则还可以编写列别名列表，以为函数的复合返回类型的一个或多个属性提供替代名称。如果已将函数定义为返回记录数据类型，则必须存在别名或关键字AS，然后是形式为(column_name data_type [, ...])的列定义列表。列定义列表必须匹配该函数返回的实际列数和类型。

join_type

以下之一：

- **[INNER] JOIN**
- **LEFT [OUTER] JOIN**
- **RIGHT [OUTER] JOIN**
- **FULL [OUTER] JOIN**
- **CROSS JOIN**

对于INNER和OUTER连接类型，必须指定一个连接条件，即恰好是NATURAL，ON *join_condition*或USING (*join_column*[, ...])之一。含义如下。对于CROSS JOIN，这些子句都不会出现。

JOIN子句结合了两个FROM项，为方便起见，我们将其称为“表”，尽管实际上它们可以是任何类型的FROM项。如有必要，请使用括号确定嵌套顺序。在没有括号的情况下，JOIN从左到右嵌套。无论如何，JOIN的绑定比逗号分隔FROM列表项的绑定更紧密。

CROSS JOIN和INNER JOIN产生一个简单的笛卡尔积，与在FROM的顶层列出两个表所获得的结果相同，但受连接条件（如果有）的限制。

CROSS JOIN等效于INNER JOIN ON (TRUE)，即，没有行被限定删除。这些连接类型只是一种符号上的方便，因为它们无法执行普通FROM和WHERE无法完成的工作。

LEFT OUTER JOIN返回合格的笛卡尔乘积中的所有行（即，所有通过其

连接条件的组合行），加上左表中没有通过右表连接条件的每行的一个副本。通过为右侧列插入空值，此左侧行将扩展为连接表的整个宽度。请注意，在确定哪些行具有匹配项时，仅考虑JOIN子句自身的条件。之后应用外部条件。

相反，RIGHT OUTER JOIN返回所有已连接的行，并为每个不匹配的右手行加上一行（在左边扩展为空）。这只是一种符号上的便利，因为您可以通过切换左右表将其转换为LEFT OUTER JOIN。

FULL OUTER JOIN返回所有连接的行，再加上一行，用于每行不匹配的左表行（在右边扩展为空），再加上每行不匹配的右表行（扩展在左边为空）。

ON *join_condition*

*join_condition*是一个表达式，其结果为boolean类型的值（类似于WHERE子句），该值指定连接中的哪些行被视为匹配。

USING (*join_column* [, ...])

USING (*a*, *b*, ...)形式的子句是ON *left_table.a* = *right_table.a* AND *left_table.b* = *right_table.b* ...的简写。同样，USING意味着连接输出中 will 仅包括每对等效列中的一对，而不是两者。

NATURAL

NATURAL是USING列表的简写形式，该列表提到两个表中具有相同名称的所有列。如果没有公用的列名，则NATURAL等效于ON TRUE。

WHERE子句

可选的WHERE子句具有一般形式：

```
WHERE condition
```

*condition*是任何计算结果为boolean型结果的表达式。任何不满足此条件的行将从输出中删除。如果用一行的实际值替换其中的变量引用后，该表达式返回真，则该行符合条件。

GROUP BY子句

可选的GROUP BY子句具有一般形式：

```
GROUP BY grouping_element [, ...]
```

其中*grouping_element*可以是以下之一：

```
()  
expression  
ROLLUP (expression [,...])  
CUBE (expression [,...])  
GROUPING SETS ((grouping_element [, ...]))
```

GROUP BY将所有共享有相同值的分组表达式的所有选定行压缩为单行。*expression*可以是输入列名称，也可以是输出列（SELECT列表项）的名称或序号，或者是由输入列值组成的任意表达式。如有歧义，GROUP BY名称将被解释为输入列名称，而不是输出列名称。

汇总函数（如果有的话）在组成每个组的所有行中进行计算，从而为每个组生成单独的值。（如果有聚合函数但没有GROUP BY子句，则该查询被视为具有包含所有选定行的单个组。）可以通过将FILTER子句附加到聚合函数调用来进一步过滤提供给每个聚合函数的行集。如果存在FILTER子句，则只有与之匹配的那些行才包含在该聚合函数的输入中。

如果存在GROUP BY或任何聚合函数，则SELECT列表表达式不能引用未分组的列，除非是在聚合函数内，或者当未分组的列在函数上依赖于分组的列时，否则对于未分组的列将会返回不止一个的可能的值。如果分组的列（或其子集）是包含未分组的列的表的主键，则存在函数依赖。

请记住，在评估HAVING子句或SELECT列表中的任何“标量”表达式之前，将评估所有聚合函数。这意味着，例如，CASE表达式不能用于跳过对聚合函数的求值；见[表达式计算规则](#)。

MPP数据库具有以下附加的OLAP分组扩展（通常称为*supergroups*）：

ROLLUP

ROLLUP分组是对GROUP BY子句的扩展，该子句创建聚合部分和，该部分和从最详细的级别汇总到总计，并遵循分组列（或表达式）的列表。ROLLUP提取分组列的有序列表，计算在GROUP BY子句中指定的标准聚合值，然后逐步创建更高级别的部分和，从列表的右向左移动。最后，它创建了一个总计。可以将ROLLUP分组视为一系列分组集。例如：

```
GROUP BY ROLLUP (a,b,c)
```

等效于：

```
GROUP BY GROUPING SETS( (a,b,c), (a,b), (a), ( ) )
```

请注意，ROLLUP的 n 个元素转换为 $n+1$ 个分组集。同样，在ROLLUP中指定分组表达式的顺序也很重要。

CUBE

CUBE分组是对GROUP BY子句的扩展，它为分组列（或表达式）的给定列表的所有可能组合创建部分和。在多维分析方面，CUBE生成可以为具有指定维的多维数据集计算的所有部分和。例如：

```
GROUP BY CUBE (a,b,c)
```

等效于：

```
GROUP BY GROUPING SETS( (a,b,c), (a,b), (a,c), (b,c), (a),  
(b), (c), ( ) )
```

注意，CUBE的 n 个元素转换为 2^n 个分组集。考虑在需要交叉表报告的任何情况下使用CUBE。CUBE通常最适合使用多个维度列而不是表示单个维度不同级别的列的查询。例如，通常要求的交叉列表可能需要月份，州和产品的所有组合的部分和。

GROUPING SETS

您可以使用GROUP BY子句中的GROUPING SETS表达式有选择地指定

要创建的组集。这样就可以在多个维度上进行精确指定，而无需计算整个ROLLUP或CUBE。例如：

```
GROUP BY GROUPING SETS( (a,c), (a,b) )
```

如果使用分组扩展子句ROLLUP，CUBE或GROUPING SETS，则要面临两个问题。首先，如何确定哪些结果行是部分和，然后确定给定部分和的确切聚合级别。或者，如何区分包含存储的NULL值和由ROLLUP或CUBE创建的“NULL”值的结果行。其次，当在GROUP BY子句中指定重复的分组集时，如何确定哪些结果行是重复的？您可以在SELECT列表中使用两个附加的分组函数来帮助解决此问题：

- **grouping(column [, ...])** — 可以将grouping函数应用于一个或多个分组属性，以将超聚合的行与常规的分组行区分开。这有助于将代表超级汇总行中所有值的集合的“NULL”与常规行中的NULL值区分开。此函数中的每个参数都产生一个位 - 1或0，其中1表示结果行是超级聚合的，而0表示结果行来自常规分组。grouping函数通过将这些位视为二进制数然后将其转换为以10为基的整数来返回整数。
- **group_id()** — 对于包含重复分组集的分组扩展查询，group_id函数用于标识输出中的重复行。所有唯一分组集输出的唯一行的group_id值为0。对于每个检测到的重复分组集，group_id函数分配的group_id编号大于0。特定重复分组集中的所有输出行均由相同的group_id编号标识。

WINDOW子句

可选的WINDOW子句指定出现在查询的SELECT列表或ORDER BY子句中的窗口函数的行为。这些函数可以在其OVER子句中按名称引用WINDOW子句条目。但是，WINDOW子句条目不必在任何地方引用。如果查询中未使用它，则将其忽略。可以根本不使用任何WINDOW子句来使用窗口函数，因为窗口函数调用可以直接在其OVER子句中指定其窗口定义。但是，当多个窗口函数需要相同的窗口定义时，WINDOW子句将保存键入内容。

例如：

```
SELECT vendor, rank() OVER (mywindow) FROM sale
GROUP BY vendor
WINDOW mywindow AS (ORDER BY sum(prc*qty));
```

WINDOW子句具有以下一般形式：

```
WINDOW window_name AS (window_definition)
```

其中*window_name*是可以从OVER子句或后续窗口定义中引用的名称，而*window_definition*为：

```
[existing_window_name]
[PARTITION BY expression [, ...]]
[ORDER BY expression [ASC | DESC | USING operator] [NULLS {FIRST | LAST}] [, ...] ]
[frame_clause]
```

existing_window_name

如果指定了*existing_window_name*，则它必须引用WINDOW列表中的较早条目；新窗口将复制该条目的分区子句以及其排序子句（如果有）。新窗口不能指定自己的PARTITION BY子句，只有复制的窗口没有该子句时，它才能指定ORDER BY。新窗口始终使用其自己的frame子句；复制的窗口不得指定frame子句。

PARTITION BY

PARTITION BY子句根据指定表达式的唯一值将结果集组织为逻辑组。PARTITION BY子句的元素解释方式与GROUP BY子句的元素几乎相同，不同之处在于它们始终是简单表达式，而不是输出列的名称或编号。另一个区别是这些表达式可以包含聚合函数调用，这在常规GROUP BY子句中是不允许的。在此处允许使用它们，因为窗口在分组和聚合之后发生。与窗口函数一起使用时，这些函数将分别应用于每个分区。例如，如果在PARTITION BY之后加上列名，则结果集将按该列的不同值进行分区。如果省略，则将整个结果集视为一个分区。同样，ORDER BY列表中的元素的解释方式与ORDER BY子句中的元素几乎相同，不同之处在于，始终将表达式视为简单表达式，而不使用输出

列的名称或编号。

ORDER BY

ORDER BY子句的元素定义如何对结果集的每个分区中的行进行排序。如果省略，则以最有效的顺序返回行，并且行可能会有所不同。注意：缺少一致顺序（例如time）的数据类型的列不是在窗口规范的ORDER BY子句中使用的良好候选对象。带有或不带有时区的时间缺乏连贯的排序，因为加法和减法没有预期的效果。例如，以下条件通常不成立： $x::time < x::time + '2\ hour'::interval$

frame_clause

可选的*frame_clause*定义依赖于框架的窗口函数的窗口框架（并非全部如此）。窗口框架是查询的每一行（称为当前行）的一组相关行。

*frame_clause*可以是以下之一

```
{ RANGE | ROWS } frame_start  
{ RANGE | ROWS } BETWEEN frame_start AND frame_end
```

其中*frame_start*和*frame_end*可以是以下之一

- UNBOUNDED PRECEDING
- *value* PRECEDING
- CURRENT ROW
- *value* FOLLOWING
- UNBOUNDED FOLLOWING

如果省略*frame_end*，则默认为CURRENT ROW。限制条件是*frame_start*不能为UNBOUNDED FOLLOWING，*frame_end*不能为UNBOUNDED PRECEDING，并且*frame_end*选择不能在上述列表中出现*frame_start*选择之前——例如，RANGE BETWEEN CURRENT ROW AND *value* PRECEDING是不允许的。

默认的框架选项是RANGE UNBOUNDED PRECEDING，与RANGE BETWEEN UNBOUNDED PRECEDING AND CURRENT ROW之间的范围相同；它将框架设置为从分区开始到当前行的最后一个对等方的所有行（ORDER BY的行认为与当前行等效，如果没有ORDER BY则为所有

行)。通常，UNBOUNDED PRECEDING表示框架以分区的第一行开始，类似地UNBOUNDED FOLLOWING表示框架以分区的最后一行结束（与RANGE或ROWS模式无关）。在ROWS模式下，CURRENT ROW表示框架以当前行开始或结束；但是在RANGE模式下，这意味着框架以ORDER BY顺序中当前行的第一个或最后一个对等点开始或结束。当前仅在ROWS模式下允许 *value* PRECEDING和 *value* FOLLOWING情况。它们指示框架以当前行之前或之后许多行的行开始或结束。 *value*必须是不包含任何变量，聚合函数或窗口函数的整数表达式。该值不能为null或负数；但它可以为零，从而选择当前行本身。

请注意，如果ORDER BY排序不能唯一地对行进行排序，则ROWS选项可能会产生不可预测的结果。RANGE选项旨在确保以相同的方式对待在ORDER BY顺序中为对等的行；所有对等行将在同一框架中。

使用ROWS或RANGE子句来表示窗口的边界。窗口边界可以是一个分区的一行，多行或所有行。您可以用偏移当前行值的数据值范围

（RANGE）或偏移当前行的行数（ROWS）来表示窗口的边界。使用RANGE子句时，还必须使用ORDER BY子句。这是因为生成窗口而执行的计算需要对值进行排序。此外，ORDER BY子句不能包含多个表达式，并且该表达式必须导致日期或数字值。使用ROWS或RANGE子句时，如果仅指定起始行，则当前行将用作窗口中的最后一行。

PRECEDING — PRECEDING子句使用当前行作为参考点定义窗口的第一行。起始行以当前行之前的行数表示。例如，对于ROWS框架，5 PRECEDING设置窗口从当前行之前的第五行开始。对于RANGE框架，它将窗口设置为从第一行开始，其排序列值在给定顺序比当前行的顺序高5。如果指定的顺序按日期升序，则它将是当前行之前5天内的第一行。UNBOUNDED PRECEDING将窗口中的第一行设置为分区中的第一行。

BETWEEN — BETWEEN子句使用当前行作为参考点定义窗口的第一行和最后一行。第一行和最后一行分别用当前行之前和之后的行数表示。例如，BETWEEN 3 PRECEDING AND 5 FOLLOWING将窗口设置为从当前行之前的第三行开始，到当前行之后的第五行结束。使用BETWEEN UNBOUNDED PRECEDING AND UNBOUNDED FOLLOWING，可以将窗口中的第一行和最后一行分别设置为分区的第一行和最后一行。如果未指定ROW或RANGE子句，则等效于默认行为。

FOLLOWING — FOLLOWING子句使用当前行作为参考点定义窗口的最后

一行。最后一行以当前行之后的行数表示。例如，对于ROWS框架，5 FOLLOWING将窗口设置为在当前行之后的第五行结束。在RANGE框架的情况下，它将窗口设置为最后一行的末尾，其排序列值在给定顺序的当前行后5个。如果指定的顺序按日期升序，则它将是当前行之后5天内的最后一行。使用UNBOUNDED FOLLOWING将窗口中的最后一行设置为分区中的最后一行。

如果未指定ROW或RANGE子句，并且使用ORDER BY，则窗口绑定从分区的第一行（UNBOUNDED PRECEDING）开始，到当前行（CURRENT ROW）结束。如果未指定ORDER BY，则窗口从分区的第一行开始（UNBOUNDED PRECEDING），然后从分区的最后一行结束（UNBOUNDED FOLLOWING）。

HAVING子句

可选的HAVING子句具有以下一般形式：

```
HAVING condition
```

其中`condition`与为WHERE子句指定的条件相同。HAVING去除不满足条件的组行。HAVING与WHERE不同：WHERE在应用GROUP BY之前过滤单个行，而HAVING过滤GROUP BY创建的组行。`condition`中引用的每个列都必须明确引用一个分组列，除非该引用出现在聚合函数中或未分组的列在函数上依赖于分组列。

即使没有GROUP BY子句，HAVING的存在也会将查询转换为分组查询。这与查询包含聚合函数但不包含GROUP BY子句的情况相同。所有选定的行都被视为形成一个单一的组，并且SELECT列表和HAVING子句只能引用聚合函数中的表列。如果HAVING条件为true，则此类查询将返回单行；如果条件不是true，则将返回零行。

UNION子句

UNION子句具有以下一般形式：

```
select_statement UNION [ALL | DISTINCT] select_statement
```

其中*select_statement*是不带ORDER BY，LIMIT，FOR NO KEY UPDATE，FOR UPDATE，FOR SHARE或FOR KEY SHARE子句的任何SELECT语句。（如果将ORDER BY和LIMIT括在圆括号中，则可以将其附加到子查询表达式。不带圆括号，这些子句将应用于UNION的结果，而不是其右侧输入表达式。）

UNION运算符计算所涉及的SELECT语句返回的行的集合并集。如果行出现在至少一个结果集中，则在两个结果集中的行并集中。表示UNION的直接操作数的两个SELECT语句必须产生相同数量的列，并且对应的列必须具有兼容的数据类型。

除非指定了ALL选项，否则UNION的结果不包含任何重复的行。ALL防止重复项的消除。（因此，UNION ALL通常比UNION快得多；请尽可能使用ALL。）可以编写DISTINCT来明确指定消除重复行的默认行为。

除非括号中另有说明，否则同一SELECT语句中的多个UNION运算符从左到右求值。

当前，不能为UNION结果或UNION的任何输入指定FOR NO KEY UPDATE，FOR UPDATE，FOR SHARE和FOR KEY SHARE。

INTERSECT子句

INTERSECT子句具有以下一般形式：

```
select_statement INTERSECT [ALL | DISTINCT] select_statement
```

其中*select_statement*是不带ORDER BY，LIMIT，FOR NO KEY UPDATE，FOR UPDATE，FOR SHARE或FOR KEY SHARE子句的任何SELECT语句。

INTERSECT运算符计算所涉及的SELECT语句返回的行的交集。如果一行出现在两个结果集中，则该行位于两个结果集中的交集。

除非指定ALL选项，否则INTERSECT的结果不包含任何重复的行。使用ALL，在左表中具有 m 个重复项且在右表中具有 n 个重复项的行将在结果集中出现 $\min(m, n)$ 次。可以编写DISTINCT来明确指定消除重复行的默认行为。

除非括号中另有规定，否则同一SELECT语句中的多个INTERSECT运算符从左到右求值。INTERSECT的优先级比UNION高。即， $A \text{ UNION } B \text{ INTERSECT } C$ 等同于 $A \text{ UNION } (B \text{ INTERSECT } C)$ 。

当前，不能为INTERSECT结果或INTERSECT的任何输入指定FOR NO KEY UPDATE，FOR UPDATE，FOR SHARE和FOR KEY SHARE。

EXCEPT子句

EXCEPT子句具有以下一般形式：

```
select_statement EXCEPT [ALL | DISTINCT] select_statement
```

其中`select_statement`是不带ORDER BY，LIMIT，FOR NO KEY UPDATE，FOR UPDATE，FOR SHARE或FOR KEY SHARE子句的任何SELECT语句。

EXCEPT运算符计算在左SELECT语句的结果但不在右SELECT的结果的交集。

除非指定了ALL选项，否则EXCEPT的结果将不包含任何重复的行。使用ALL，在左表中具有 m 个重复项且在右表中具有 n 个重复项的行将在结果集中出现 $\max(m-n, 0)$ 次。可以编写DISTINCT来明确指定消除重复行的默认行为。

除非括号中另有规定，否则同一SELECT语句中的多个EXCEPT运算符从左到右求值。EXCEPT与UNION优先级相同。

当前，不能为EXCEPT结果或EXCEPT的任何输入指定FOR NO KEY

UPDATE , FOR UPDATE , FOR SHARE和FOR KEY SHARE。

ORDER BY子句

可选的ORDER BY子句具有以下一般形式：

```
ORDER BY expression [ASC | DESC | USING operator] [NULLS {FIRST | LAST}] [...]
```

其中*expression*可以是输出列（SELECT列表项）的名称或序号，也可以是由输入列值组成的任意表达式。

ORDER BY子句使结果行根据指定的表达式排序。如果两行最左边的表达式相等，则根据下一个表达式对它们进行比较，依此类推。如果所有指定的表达式相等，则将根据实现的顺序返回它们。

序号是指输出列的序数位置（从左到右）。通过此特性，可以基于没有唯一名称的列来定义顺序。这绝对不是必须的，因为始终可以使用AS子句为输出列分配名称。

还可以在ORDER BY子句中使用任意表达式，包括未出现在SELECT输出列表中的列。因此，以下语句是有效的：

```
SELECT name FROM distributors ORDER BY code;
```

此特性的局限性在于，应用于UNION，INTERSECT或EXCEPT子句结果的ORDER BY子句只能指定输出列名称或序号，而不能指定表达式。

如果ORDER BY表达式是一个与输出列名和输入列名都匹配的简单名称，则ORDER BY将其解释为输出列名。这与GROUP BY在相同情况下所做的选择相反。这两个语句的不一致行为与SQL标准兼容。

可以选择在ORDER BY子句中的任何表达式之后添加关键字ASC（升序）

或DESC（降序）。如果未指定，则默认采用ASC。或者，可以在USING子句中指定特定的排序运算符名称。ASC通常等效于USING <，而DESC通常等效于USING >。（但是，用户定义数据类型的创建者可以准确定义默认的排序顺序，并且它可能与其他名称的运算符相对应。）

如果指定了NULLS LAST，则空值将在所有非空值之后排序；如果指定了NULLS FIRST，则空值将在所有非空值之前排序。如果都未指定，则默认行为是在指定或隐含ASC时为NULLS LAST，而在指定DESC时为NULLS FIRST（因此，默认值表示空值大于非空值）。指定USING时，默认的空值排序取决于运算符是小于运算符还是大于运算符。

请注意，排序选项仅适用于它们遵循的表达式。例如，ORDER BY x, y DESC的含义与ORDER BY x DESC, y DESC的含义不同。

字符串数据是根据创建数据库时建立的特定于语言环境的排序顺序进行排序的。

字符串数据是根据应用于要排序的列的排序规则进行排序的。可以根据需要通过在`expression`中包含COLLATE子句来覆盖它，例如ORDER BY mycolumn COLLATE "en_US"。有关定义排序规则的信息，请参见[CREATE COLLATION](#)。

LIMIT子句

LIMIT子句包含两个独立的子句：

```
LIMIT {count | ALL}  
OFFSET start
```

其中`count`指定要返回的最大行数，而`start`指定在开始返回行之前要跳过的行数。如果同时指定了两者，则在开始对要返回的count行进行计数之前，将跳过起始行。

如果count表达式的计算结果为NULL，则将其视为LIMIT ALL，即无限制。如果start计算为NULL，则将其与OFFSET 0相同。

SQL : 2008引入了不同的语法来实现相同的结果，MPP数据库也支持该语法。它是：

```
OFFSET start [ ROW | ROWS ]  
      FETCH { FIRST | NEXT } [ count ] { ROW | ROWS } ONLY
```

在这种语法中，标准要求`start`或`count`值是文字常量，参数或变量名；作为MPP数据库扩展，可以使用其他表达式，但通常需要将其括在括号中以避免歧义。如果在FETCH子句中省略了`count`，则其默认值为1。ROW和ROWS以及FIRST和NEXT是不影响这些子句效果的干扰词。根据标准，如果两者都存在，则OFFSET子句必须位于FETCH子句之前。但MPP数据库允许使用任何顺序。

使用LIMIT时，最好使用ORDER BY子句将结果行强制为唯一的顺序。否则，您将获得查询行的不可预测的子集 - 您可能会要求以第十到第二十个行，但以什么顺序要求第十到第二十个行？除非您指定ORDER BY，否则您不知道该如何排序。

查询优化器在生成查询计划时会考虑LIMIT，因此根据LIMIT和OFFSET的使用方式，您很可能会获得不同的计划（产生不同的行顺序）。因此，除非使用ORDER BY强制执行可预测的结果顺序，否则使用不同的LIMIT/OFFSET值选择查询结果的不同子集将产生不一致的结果。这不是缺陷；这是一个正常的结果，即除非使用ORDER BY强制该顺序，否则SQL不会保证以任何特定顺序传递查询结果。

Locking子句

FOR UPDATE，FOR NO KEY UPDATE，FOR SHARE和FOR KEY SHARE是锁定子句；它们会影响SELECT如何锁定从表中获取的行。

锁定子句具有一般形式

```
FOR lock_strength [OF table_name [ , ... ] ] [ NOWAIT ]
```


其中`lock_strength`可以是以下之一

```
UPDATE  
NO KEY UPDATE  
SHARE  
KEY SHARE
```

Note: 默认情况下，MPP数据库获取对表的EXCLUSIVE锁，以对堆表进行DELETE和UPDATE操作。启用全局死锁检测器后，堆表上DELETE和UPDATE操作的锁定模式为ROW EXCLUSIVE。通过将[gp_enable_global_deadlock_detector](#)配置参数设置为true来启用全局死锁检测器。有关全局死锁检测器的信息，请参见*MPP数据库管理员手册*中的[全局死锁检测器](#)。

有关每种行级锁定模式的更多信息，请参见PostgreSQL文档中的[显式锁定](#)。

为防止该操作等待其他事务提交，请使用NOWAIT选项。使用NOWAIT，如果无法立即锁定选定的行，该语句将报告错误，而不是等待。请注意，NOWAIT仅适用于行级锁 - 所需的ROW SHARE表级锁仍以常规方式获取。如果需要不等待就获取表级锁，则可以先将LOCK与NOWAIT选项一起使用。

如果在锁定子句中命名了特定的表，则仅锁定来自那些表的行；其他使用SELECT的表则像往常一样简单地读取。没有表列表的锁定子句会影响该语句中使用的所有表。如果将锁定子句应用于视图或子查询，则它将影响视图或子查询中使用的所有表。但是，这些子句不适用于主查询引用的WITH查询。如果希望在WITH查询中发生行锁定，请在WITH查询中指定一个锁定子句。

如果有必要为不同的表指定不同的锁定行为，则可以编写多个锁定子句。如果同一表被两个以上的锁定子句提及（或隐式影响），则将其视为仅由最强的一个子表指定。同样，如果在影响该表的任何子句中指定了表，则该表将作为NOWAIT处理。

在无法用单独的表行清楚地标识返回的行的上下文中，不能使用锁定子句；例

如，它们不能与聚合一起使用。

当锁定子句出现在SELECT查询的顶层时，被锁定的行与查询返回的行完全相同。对于连接查询，锁定的行是那些促成返回连接行的行。此外，查询快照中满足查询条件的行将被锁定，尽管这些行在快照后对其进行更新并且不再满足查询条件不会返回。如果使用LIMIT，则一旦返回了足够的行以满足限制，锁定就会停止（但请注意，被OFFSET跳过的行将被锁定）。同样，如果在游标的查询中使用锁定子句，则仅锁定实际被游标读取或跳过的行。

当sub-SELECT中出现locking子句时，被锁定的行就是该子查询返回给外部查询的行。这可能涉及的行数少于单独检查子查询所建议的行数，因为外部查询的条件可用于优化子查询的执行。例如，

```
SELECT * FROM (SELECT * FROM mytable FOR UPDATE) ss WHERE col1 = 5;
```

将仅锁定col1 = 5的行，即使该条件在文本上不在子查询中。

SELECT命令可能以READ COMMITTED事务隔离级别运行，并且使用ORDER BY和锁定子句可以使行无序返回。这是因为ORDER BY首先被应用。该命令对结果进行排序，但随后可能会阻止尝试获得对一个或多个行的锁定。一旦SELECT解除阻塞，某些排序列值可能已被修改，从而导致这些行看起来是乱序的（尽管就原始列值而言，它们是有序的）。例如，可以通过在子查询中放置FOR UPDATE/SHARE子句来解决此问题。

```
SELECT * FROM (SELECT * FROM mytable FOR UPDATE) ss ORDER BY column1;
```

请注意，这将导致锁定mytable的所有行，而顶层的FOR UPDATE将仅锁定实际返回的行。这可能会导致明显的性能差异，特别是如果将ORDER BY与LIMIT或其他限制结合使用。因此，仅当期望并发更新排序列并且需要严格排序的结果时才建议使用此技术。

在REPEATABLE READ或SERIALIZABLE事务隔离级别，这将导致序列化失败（SQLSTATE为40001），因此在这些隔离级别下，不可能乱序接收行。

TABLE命令

命令

```
TABLE name
```

完全等同于

```
SELECT * FROM name
```

在部分复杂查询中，它可用作顶级命令或节省空间的语法变体。

示例

连接表films与表distributors:

```
SELECT f.title, f.did, d.name, f.date_prod, f.kind FROM  
distributors d, films f WHERE f.did = d.did
```

将表films的length列求和并将结果按kind分组:

```
SELECT kind, sum(length) AS total FROM films GROUP BY kind;
```

将表films的length列求和并将结果按kind分组并显示总和小于5小时的组：

```
SELECT kind, sum(length) AS total FROM films GROUP BY kind  
HAVING sum(length) < interval '5 hours';
```

计算电影kind和distributor的所有销售部分和与总和。

```
SELECT kind, distributor, sum(prc*qty) FROM sales
GROUP BY ROLLUP(kind, distributor)
ORDER BY 1,2,3;
```

根据总销量计算电影发行商的排名：

```
SELECT distributor, sum(prc*qty),
       rank() OVER (ORDER BY sum(prc*qty) DESC)
FROM sale
GROUP BY distributor ORDER BY 2 DESC;
```

以下两个示例是根据第二列（name）的内容对单个结果进行排序的相同方法：

```
SELECT * FROM distributors ORDER BY name;
SELECT * FROM distributors ORDER BY 2;
```

下一个示例说明如何获取表distributors和actors的并集，将结果限制为每个表中以字母W开头的行。只需要不同的行，因此关键字ALL被省略：

```
SELECT distributors.name FROM distributors WHERE
distributors.name LIKE 'W%' UNION SELECT actors.name FROM
actors WHERE actors.name LIKE 'W%';
```

此示例说明如何在FROM子句中使用函数，无论是否包含列定义列表：

```
CREATE FUNCTION distributors(int) RETURNS SETOF distributors
AS $$ SELECT * FROM distributors WHERE did = $1; $$ LANGUAGE
SQL;
SELECT * FROM distributors(111);

CREATE FUNCTION distributors_2(int) RETURNS SETOF record AS
$$ SELECT * FROM distributors WHERE did = $1; $$ LANGUAGE
SQL;
SELECT * FROM distributors_2(111) AS (dist_id int, dist_name
text);
```

此示例使用一个简单的WITH子句：

```
WITH test AS (
    SELECT random() as x FROM generate_series(1, 3)
)
SELECT * FROM test
UNION ALL
SELECT * FROM test;
```

本示例使用WITH子句仅显示最高销售区域中的每产品销售总额。

```
WITH regional_sales AS
    SELECT region, SUM(amount) AS total_sales
    FROM orders
    GROUP BY region
), top_regions AS (
    SELECT region
    FROM regional_sales
    WHERE total_sales > (SELECT SUM(total_sales) FROM
        regional_sales)
)
SELECT region, product, SUM(quantity) AS product_units,
    SUM(amount) AS product_sales
FROM orders
WHERE region IN (SELECT region FROM top_regions)
GROUP BY region, product;
```

该示例可能编写不包含WITH子句，但需要两级嵌套的sub-SELECT语句。

本示例使用WITH RECURSIVE子句从仅显示直接下属的表中查找员工Mary的所有下属（直接或间接）及其间接级别：

```
WITH RECURSIVE employee_recursive(distance, employee_name, manager_name) AS (  
    SELECT 1, employee_name, manager_name  
    FROM employee  
    WHERE manager_name = 'Mary'  
    UNION ALL  
    SELECT er.distance + 1, e.employee_name, e.manager_name  
    FROM employee_recursive er, employee e  
    WHERE er.employee_name = e.manager_name  
)  
SELECT distance, employee_name FROM employee_recursive;
```

递归查询的典型形式：初始条件，后跟UNION [ALL]，然后是查询的递归部分。确保查询的递归部分最终不会返回任何元组，否则查询将无限期地循环。有关更多示例，请参见[WITH查询（公用表表达式）](#)。

兼容性

SELECT语句与SQL标准兼容，但是有一些扩展和某些缺少的功能。

省略FROM子句

MPP数据库允许省略FROM子句。它可以直接用于计算简单表达式的结果。例如：

```
SELECT 2+2;
```

其他一些SQL数据库无法做到这一点，除非引入一个虚拟的单行表来执行SELECT。

请注意，如果未指定FROM子句，则查询无法引用任何数据库表。例如，以下查询无效：

```
SELECT distributors.* WHERE distributors.name = 'Westward';
```

在较早的版本中，将服务器配置参数`add_missing_from`设置为true允许MPP数据库为查询所引用的每个表向查询的FROM子句添加隐式条目。这个不再允许。

省略AS关键字

在SQL标准中，只要新列名是有效的列名（即与任何保留关键字不同），就可以在输出列名之前省略可选关键字AS。MPP数据库的限制更为严格：如果新列名完全匹配任何关键字（保留与否），则要求使用AS。推荐的做法是使用AS或双引号输出列名，以防止与将来添加关键字的任何可能的冲突。

在FROM项中，SQL标准和MPP数据库都允许在作为未保留关键字的别名之前省略AS。但是由于语法上的歧义，这对于输出列名称是不切实际的。

ONLY和Inheritance

使用ONLY时，SQL标准要求要在表名前后加上括号，例如：

```
SELECT * FROM ONLY (tab1), ONLY (tab2) WHERE ...
```

MPP数据库认为这些括号是可选的。

MPP数据库允许编写尾随*来明确指定包括子表的non-ONLY行为。该标准不允许这样做。

（这些要点同样适用于所有支持ONLY选项的SQL命令。）

Namespace可以用于GROUP BY和ORDER BY

在SQL-92标准中，ORDER BY子句只能使用输出列名称或序号，而GROUP BY子句只能使用基于输入列名称的表达式。MPP数据库扩展了这些子句中的每

一个，以允许其他选择（但是如果有歧义，它将使用标准的解释）。MPP数据库还允许两个子句都指定任意表达式。请注意，出现在表达式中的名称始终被视为输入列名称，而不是输出列名称。

SQL：1999及更高版本使用的定义略有不同，但并不完全与SQL-92向上兼容。但是，在大多数情况下，MPP数据库以与SQL：1999相同的方式解释ORDER BY或GROUP BY表达式。

函数依赖

仅当表的主键包含在GROUP BY列表中时，MPP数据库才能识别函数依赖（允许从GROUP BY省略列）。SQL标准指定了应识别的其他条件。

LIMIT和OFFSET

子句LIMIT和OFFSET是MPP数据库特定的语法，也由MySQL使用。如上所述，SQL：2008标准引入了子句OFFSET .. FETCH {FIRST | NEXT} ...来实现相同的功能。IBM DB2也使用此语法。（Oracle应用程序经常使用一种变通办法来实现这些子句的效果，该变通办法涉及自动生成的rownum列，MPP数据库中不提供该列。）

FOR NO KEY UPDATE, FOR UPDATE, FOR SHARE和FOR KEY SHARE

尽管FOR UPDATE出现在SQL标准中，但该标准仅允许将它作为DECLARE CURSOR的选项。MPP数据库允许在任何SELECT查询以及sub-SELECT中使用它，但这是一个扩展。FOR NO KEY UPDATE，FOR SHARE和FOR KEY SHARE变体以及NOWAIT选项未出现在标准中。

WITH中的数据修改语句

MPP数据库允许将INSERT，UPDATE和DELETE用作WITH查询。在SQL标准中不允许。

非标准子句

在SQL标准中未定义DISTINCT ON子句。

STABLE和VOLATILE函数的限制

为防止数据在MPP数据库中的各个segment之间变得不同步， 如果分类为STABLE或VOLATILE的任何函数包含SQL或以任何方式修改了数据库，则不能在segment数据库级别执行该函数。 有关更多信息，请参见[CREATE FUNCTION](#)。

另见

[EXPLAIN](#)

Parent topic: [SQL Command Reference](#)

SELECT INTO

根据查询结果定义一个新表。

概要

```
[ WITH [ RECURSIVE ] with_query [, ...] ]  
SELECT [ALL | DISTINCT [ON ( expression [, ...] )]]  
    * | expression [AS output_name] [, ...]  
    INTO [TEMPORARY | TEMP | UNLOGGED ] [TABLE] new_table  
    [FROM from_item [, ...]]  
    [WHERE condition]  
    [GROUP BY expression [, ...]]  
    [HAVING condition [, ...]]  
    [{UNION | INTERSECT | EXCEPT} [ALL | DISTINCT ] select]  
    [ORDER BY expression [ASC | DESC | USING operator] [NULLS {FIRST | LAST}] [, ...]]  
    [LIMIT {count | ALL}]  
    [OFFSET start [ ROW | ROWS ] ]  
    [FETCH { FIRST | NEXT } [ count ] { ROW | ROWS } ONLY ]  
    [FOR {UPDATE | SHARE} [OF table_name [, ...]] [NOWAIT]  
    [...]]
```

描述

SELECT INTO创建一个新表，并用查询计算的数据填充该表。数据不会像常规SELECT一样返回给客户端。新表的列具有与SELECT的输出列关联的名称和数据类型。

参数

SELECT INTO的大多数参数与SELECT相同。

TEMPORARY

TEMP

如果指定，该表将创建为临时表。

UNLOGGED

如果指定，该表将创建为不计入日志表。写入不计入日志表的数据不会写入预写（WAL）日志，这使它们比普通表快得多。但是，不计入日志表的

内容不会复制到mirror实例。同样，不计入日志表也不是崩溃安全的。segment实例崩溃或异常关闭后，该segment上不计入日志表的数据将被截断。在不计入日志表上创建的所有索引也会自动取消计入日志。

new_table

要创建的表的名称（可以用schema修饰）。

示例

创建一个新表films_recent，该表仅包含表films中的最新条目：

```
SELECT * INTO films_recent FROM films WHERE date_prod >= '2016-01-01';
```

兼容性

SQL标准使用SELECT INTO表示将值选择到主机程序的标量变量中，而不是创建新表。SELECT INTO在MPP数据库中用于表示表创建的用法是历史性的。为此，最好在新应用程序中使用[CREATE TABLE AS](#)。

另见

[SELECT, CREATE TABLE AS](#)

Parent topic: [SQL Command Reference](#)

SET

更改MPP数据库配置参数的值。

概要

```
SET [SESSION | LOCAL] configuration_parameter {TO | =} value | 'value' | DEFAULT}
SET [SESSION | LOCAL] TIME ZONE {timezone | LOCAL | DEFAULT}
```

描述

SET命令更改服务器配置参数。任何分类为`session`参数的配置参数都可以使用SET即时更改。SET仅影响当前会话使用的值。

如果在稍后中止的事务中发出SET或SET SESSION，则回滚该事务时SET命令的效果会消失。一旦提交了事务，效果将一直持续到会话结束，除非被另一个SET覆盖。

SET LOCAL的影响仅持续到当前事务结束为止，无论是否提交。特殊情况是在单个事务中SET LOCAL紧随SET其后：SET LOCAL值将一直显示到事务结束为止，但是此后（如果提交了事务）SET值将生效。

如果在包含针对同一配置参数的SET选项的函数中使用SET LOCAL（请参见[CREATE FUNCTION](#)），则SET LOCAL命令的效果在函数退出时消失；无论如何，将恢复调用该函数时有有效的值。这允许将SET LOCAL用于函数中参数的动态或重复更改，同时保留使用SET选项保存和恢复调用者值的便利。请注意，常规的SET命令会覆盖周围函数的SET选项。除非回滚，否则其影响持续存在。

如果在事务中使用DECLARE命令创建游标，则只有在使用CLOSE命令关闭游标后才能在事务中使用SET命令。

有关服务器参数的信息，请参阅[服务器配置参数](#)。

参数

SESSION

指定该命令对当前会话生效。这是默认值。

LOCAL

指定该命令仅对当前事务生效。在COMMIT或ROLLBACK之后，会话级设置将再次生效。请注意，如果SET LOCAL在事务外部执行，则似乎无效。

configuration_parameter

MPP数据库配置参数的名称。使用SET只能更改分类为*session*的参数。有关详细信息，请参见[服务器配置参数](#)。

value

参数的新值。可以将值指定为字符串常量，标识符，数字或以逗号分隔的列表。DEFAULT可用于指定将参数重置为其默认值。如果指定内存大小或时间单位，则将值用单引号引起来。

TIME ZONE

SET TIME ZONE值是SET timezone TO *value*。语法SET TIME ZONE允许时区指定使用特殊语法。以下是有效值的示例：

'PST8PDT'

'Europe/Rome'

-7 (UTC以西7小时时区)

INTERVAL '-08:00' HOUR TO MINUTE (UTC以西8小时时区)。

LOCAL

DEFAULT

将时区设置为您的本地时区（即服务器的默认*timezone*）。有关MPP数据库中时区的更多信息，请参见[PostgreSQL文档的时区部](#)。

示例

设置schema搜索路径：

```
SET search_path TO my_schema, public;
```

将每个查询的segment主机内存增加到200 MB：

```
SET statement_mem TO '200MB';
```

把日期风格设置为传统POSTGRES的"日在月之前"的输入习惯：

```
SET datestyle TO postgres, dmy;
```

设置加利福尼亚州圣马特奥市的时区（太平洋时间）：

```
SET TIME ZONE 'PST8PDT';
```

设置意大利的时区：

```
SET TIME ZONE 'Europe/Rome';
```

兼容性

SET TIME ZONE扩展了SQL标准中定义的语法。该标准仅允许数字时区偏移，而MPP数据库允许更灵活的时区规范。SET的所有其他功能都是MPP数据库扩展。

另见

[RESET, SHOW](#)

Parent topic: [SQL Command Reference](#)

SET CONSTRAINTS

设置当前事务的约束检查时间。

Note: 引用完整性语法（外键约束）被接受但未强制执行。

概要

```
SET CONSTRAINTS { ALL | name [, ...] } { DEFERRED | IMMEDIATE }
```

描述

SET CONSTRAINTS设置当前事务中的约束检查行为。 在每个语句的末尾检查IMMEDIATE约束。 在提交事务之前，不检查DEFERRED约束。 每个约束都有其自己的IMMEDIATE或DEFERRED模式。

创建后，将为约束提供以下三个特征之一：DEFERRABLE INITIALLY DEFERRED，DEFERRABLE INITIALLY IMMEDIATE或NOT

DEFERRABLE。第三类始终为IMMEDIATE，不受SET CONSTRAINTS命令的影响。前两个类以指示的模式启动每个事务，但是可以通过SET CONSTRAINTS在事务内更改其行为。

带有约束名称列表的SET CONSTRAINTS会更改那些约束的模式（必须全部推迟）。每个约束名称可以是schema限定的。如果未指定schema名称，则使用当前schema搜索路径查找第一个匹配名称。SET CONSTRAINTS ALL更改所有可延缓约束的模式。

当SET CONSTRAINTS将约束的模式从DEFERRED更改为IMMEDIATE时，新模式将追溯生效：在执行SET CONSTRAINTS命令时，将检查在事务结束时会被检查的所有未完成的数据修改。如果违反了任何此类约束，则SET CONSTRAINTS会失败（并且不会更改约束模式）。因此，SET CONSTRAINTS可以用于强制检查约束以在事务的特定点发生。

当前，此设置仅影响UNIQUE，PRIMARY KEY，REFERENCES（外键）和EXCLUDE约束。当插入或修改一行时（不在语句末尾），始终会立即检查NOT NULL和CHECK约束。还可以立即检查尚未声明为DEFERRABLE的唯一性和排除约束。

声明为“约束触发器”的触发器的触发也由此设置控制-它们在应检查相关约束的同时触发。

注解

因为MPP数据库不要求约束名称在schema中唯一（而仅按表），所以指定约束名称可能有多个匹配项。在这种情况下，SET CONSTRAINTS将对所有匹配起作用。对于非schema限定的名称，一旦在搜索路径中的某些schema中找到一个或多个匹配项，就不会搜索路径中稍后出现的schema。

此命令仅更改当前事务中约束的行为。在事务块外部发出此命令将发出警告，否则无效。

兼容性

该命令符合SQL标准中定义的行为，除了以下限制：在MPP数据库中，该命令不适用于NOT NULL和CHECK约束。 另外，MPP数据库会立即检查不可延展的唯一性约束，而不是像标准建议那样在声明结束时进行检查。

Parent topic: [SQL Command Reference](#)

SET ROLE

设置当前会话的当前角色标识符。

概要

```
SET [SESSION | LOCAL] ROLE rolename  
  
SET [SESSION | LOCAL] ROLE NONE  
  
RESET ROLE
```

描述

此命令将当前SQL会话上下文的当前角色标识符设置为*rolename*。 角色名称可以写为标识符或字符串文字。 在SET ROLE之后，将执行SQL命令的权限检

查，就好像新角色是最初登录的角色一样。

指定的`rolename`必须是当前会话用户所属的角色。如果会话用户是超级用户，则可以选择任何角色。

NONE和RESET表单将当前角色标识符重置为当前会话角色标识符。这些表格可以由任何用户执行。

参数

SESSION

指定该命令对当前会话生效。这是默认值。

LOCAL

指定该命令仅对当前事务生效。在COMMIT或ROLLBACK之后，会话级设置将再次生效。请注意，如果SET LOCAL在事务外部执行，则似乎无效。

rolename

在此会话中用于权限检查的角色名称。

NONE

RESET

将当前角色标识符重置为当前会话角色标识符（用于登录的角色的标识符）。

注解

使用此命令，可以添加特权或限制特权。如果会话用户角色具有INHERITS属性，则它将自动拥有可以SET ROLE的每个角色的所有特权；在这种情况下，SET ROLE有效地放弃直接分配给会话用户及其成员的其他角色的所有特权，仅保留命名角色可用的特权。另一方面，如果会话用户角色具有NOINHERITS属性，则SET ROLE会删除直接分配给会话用户的特权，而是

获取可用于命名角色的特权。

特别是，当超级用户选择将SET ROLE设置为非超级用户角色时，她将失去其超级用户特权。

SET ROLE具有与SET SESSION AUTHORIZATION相同的效果，但是所涉及的特权检查却大不相同。 同样，SET SESSION AUTHORIZATION确定以后的SET ROLE命令允许哪些角色， 而使用SET ROLE不会更改后续SET ROLE可以更改的角色集。

SET ROLE不处理由角色的ALTER ROLE设置指定的会话变量。 会话变量仅在登录期间处理。

示例

```
SELECT SESSION_USER, CURRENT_USER;
session_user | current_user
-----+-----
peter | peter

SET ROLE 'paul';

SELECT SESSION_USER, CURRENT_USER;
session_user | current_user
-----+-----
peter | paul
```

兼容性

MPP数据库允许使用标识符语法 (*rolename*) ，而SQL标准要求将角色名称写为字符串文字。 SQL在事务期间不允许使用此命令。 MPP数据库没有进行此限制。 SESSION和LOCAL修饰符是MPP数据库的扩展名，RESET语法也是如

此。

另见

SET SESSION AUTHORIZATION

Parent topic: [SQL Command Reference](#)

SET SESSION AUTHORIZATION

设置会话角色标识符和当前会话的当前角色标识符。

概要

SET [SESSION | LOCAL] SESSION AUTHORIZATION *rolename*

SET [SESSION | LOCAL] SESSION AUTHORIZATION DEFAULT

RESET SESSION AUTHORIZATION

描述

此命令将会话角色标识符和当前SQL会话上下文的当前角色标识符设置

为`rolename`。角色名称可以写为标识符或字符串文字。使用此命令，例如，可以暂时成为非特权用户，然后再切换回为超级用户。

会话角色标识符最初设置为客户端提供的（可能经过身份验证的）角色名称。当前角色标识符通常等于会话用户标识符，但是在`setuid`函数和类似机制的上下文中可能会暂时更改；也可以通过[SET ROLE](#)进行更改。当前用户标识符与权限检查相关。

仅当初始会话用户（已认证用户）具有超级用户特权时，才可以更改会话用户标识符。否则，仅当命令指定了经过身份验证的用户名时，该命令才会被接受。

DEFAULT和RESET表单将会话和当前用户标识符重置为原始身份验证的用户名。这些表格可以由任何用户执行。

参数

SESSION

指定该命令对当前会话生效。这是默认值。

LOCAL

指定该命令仅对当前事务生效。在COMMIT或ROLLBACK之后，会话级设置将再次生效。请注意，如果SET LOCAL在事务外部执行，则似乎无效。

rolename

指定的角色的名称。

NONE

RESET

将会话和当前角色标识符重置为用于登录的角色的标识符。

示例

```
SELECT SESSION_USER, CURRENT_USER;
session_user | current_user
-----+-----
peter | peter
```

```
SET SESSION AUTHORIZATION 'paul';
```

```
SELECT SESSION_USER, CURRENT_USER;
session_user | current_user
-----+-----
paul | paul
```

兼容性

SQL标准允许其他一些表达式代替文字`rolename`出现，但是这些选项在实践中并不重要。MPP数据库允许使用标识符语法（`rolename`），而SQL则不允许。SQL在事务期间不允许使用此命令。MPP数据库没有进行此限制。SESSION和LOCAL修饰符是MPP数据库的扩展名，RESET语法也是如此。

另见

SET ROLE

Parent topic: [SQL Command Reference](#)

SET TRANSACTION

设置当前事务的特性。

概要

```
SET TRANSACTION [transaction_mode] [READ ONLY | READ WRITE]
```

```
SET TRANSACTION SNAPSHOT snapshot_id
```

```
SET SESSION CHARACTERISTICS AS TRANSACTION transaction_mode  
[READ ONLY | READ WRITE]  
[NOT] DEFERRABLE
```

其中*transaction_mode*为下列之一：

```
ISOLATION LEVEL {SERIALIZABLE | REPEATABLE READ | READ COMMITTED | READ UNCOMMITTED}
```

描述

SET TRANSACTION命令设置当前事务的特性。它对任何后续交易都没有影响。

可用的事务特性是事务隔离级别，事务访问模式（读/写或只读）和可延迟的模式。

Note: 可延迟事务要求该事务为可序列化。MPP数据库不支持可序列化的事务，因此包含DEFERRABLE子句无效。

MPP数据库不支持SET TRANSACTION SNAPSHOT命令。

事务的隔离级别确定当其他事务同时运行时，该事务可以看到哪些数据。

- **READ COMMITTED** — 一条语句只能看到在开始之前提交的行。这是默认值。
- **REPEATABLE READ** — 当前事务中的所有语句只能看到在事务中执行的第一个查询语句之前提交的或该事务中数据修改语句修改的行。

SQL标准定义了两个附加级别，即READ UNCOMMITTED和SERIALIZABLE。在MPP数据库中，READ UNCOMMITTED被视为READ COMMITTED。如果指定SERIALIZABLE，则MPP数据库将退回到REPEATABLE READ。

在执行事务的第一个查询或数据修改语句

(SELECT，INSERT，DELETE，UPDATE，FETCH或COPY) 之后，不能更改事务隔离级别。

事务访问模式确定事务是读/写还是只读。读/写是默认设置。当事务为只读时，不允许使用以下SQL命令：INSERT，UPDATE，DELETE和COPY FROM（如果要写入的表不是临时表）；所有的CREATE，ALTER和DROP命令；GRANT，REVOKE，TRUNCATE；如果EXPLAIN ANALYZE和EXECUTE将要执行的命令在上述命令之中，也不被允许。这是只读的高级概念，不会阻止所有对磁盘的写入。

除非事务既是SERIALIZABLE又是READ ONLY，否则DEFERRABLE事务属性无效。当在事务上设置了所有这些属性后，该事务在首次获取其快照时可能会阻塞，此后它可以在没有SERIALIZABLE事务的正常开销的情况下运行，并且没有任何导致序列化失败或被序列化失败取消的风险。由于MPP数据库不支持可序列化的事务，因此DEFERRABLE事务属性在MPP数据库中无效。

参数

SESSION CHARACTERISTICS

为会话的后续事务设置默认事务特性。

READ UNCOMMITTED

READ COMMITTED

REPEATABLE READ

SERIALIZABLE

SQL标准定义了四个事务隔离级别：READ UNCOMMITTED，READ COMMITTED，REPEATABLE READ和SERIALIZABLE。

READ UNCOMMITTED允许事务查看未提交的并发事务所做的更改。在MPP数据库中这是不可能的，因此READ UNCOMMITTED与READ COMMITTED相同。

READ COMMITTED是MPP数据库中的默认隔离级别，可确保语句只能看到在开始之前提交的行。如果在第一次执行该语句后又提交了另一个并发事务，则在一个事务中执行两次的相同语句可能会产生不同的结果。

REPEATABLE READ隔离级别确保事务只能看到在事务开始之前提交的行。REPEATABLE READ是MPP数据库支持的最严格的事务隔离级别。由于可串行化失败，使用REPEATABLE READ隔离级别的应用程序，必须准备重试事务。

SERIALIZABLE事务隔离级别确保当前事务的所有语句只能看到在此事务中执行第一个查询之前或此事务内数据修改语句提交的行。如果并发可序列化事务之间的读取和写入模式会导致这种事务的任何串行（一次一个）执行都不可能发生的情况，则其中一个事务将回滚，并出现serialization_failure错误。MPP数据库不完全支持标准定义的SERIALIZABLE，因此，如果指定SERIALIZABLE，则MPP数据库将退回到REPEATABLE READ。有关MPP数据库中事务可串行化的更多信息，请参见[兼容性](#)。

READ WRITE

READ ONLY

确定事务是读/写还是只读。读/写是默认设置。当事务为只读时，不允许使用以下SQL命令：INSERT，UPDATE，DELETE和COPY FROM（如果要写入的表不是临时表）；所有的CREATE，ALTER和DROP命令；GRANT，REVOKE，TRUNCATE；如果EXPLAIN ANALYZE和EXECUTE将要执行的命令在上述命令之中，也不被允许。

[NOT] DEFERRABLE

因为不支持SERIALIZABLE事务，所以DEFERRABLE事务属性在MPP数

据库中无效。如果指定了DEFERRABLE且该事务既是SERIALIZABLE也是READ ONLY的，则该事务在首次获取其快照时可能会阻塞，此后它可以在没有SERIALIZABLE事务的正常开销的情况下运行，并且不存在构成或被取消可串行化失败风险。会造成任何贡献或被其取消序列化失败。此模式非常适合长时间运行的报告或备份。

注解

如果在没有事先进行START TRANSACTION或BEGIN的情况下执行SET TRANSACTION，则会发出警告，并且该命令无效。

通过在BEGIN或START TRANSACTION中指定所需的事务模式，可以省去SET TRANSACTION。

也可以通过设置配置参数[`default\_transaction\_isolation`](#)，[`default\_transaction\_read\_only`](#)和[`default\_transaction\_deferrable`](#)来设置会话默认事务模式。

示例

设置当前事务的事务隔离级别：

```
BEGIN;  
SET TRANSACTION ISOLATION LEVEL REPEATABLE READ;
```

兼容性

这两个命令均在SQL标准中定义。SERIALIZABLE是标准中的默认事务隔离级

别。在MPP数据库中，默认值为READ COMMITTED。由于缺少谓词锁定，MPP数据库不完全支持SERIALIZABLE级别，因此当指定SERIALIZABLE时，它退回到REPEATABLE READ级别。本质上，谓词锁定系统通过限制写的内容来防止幻像读取，而MPP数据库中使用的多版本并发控制模型（MVCC）通过限制读取的内容来防止幻像读取。

PostgreSQL提供了一个真正的可序列化隔离级别，称为可序列化快照隔离（SSI），它可以监视并发事务并回滚可能引入序列化异常的事务。MPP数据库未实现此隔离模式。

在SQL标准中，可以使用以下命令设置其他事务特性：诊断区域的大小。此概念是嵌入式SQL特有的，因此未在MPP数据库服务器中实现。

DEFERRABLE事务模式是MPP数据库语言的扩展。

SQL标准要求连续的*transaction_modes*之间使用逗号，但是由于历史原因，MPP数据库允许省略逗号。

另见

[BEGIN, LOCK](#)

Parent topic: [SQL Command Reference](#)

SHOW

显示系统配置参数的值。

概要

```
SHOW configuration_parameter

SHOW ALL
```

描述

SHOW显示MPP数据库系统配置参数的当前设置。 您可以使用SET语句或通过编辑MPP数据库master的postgresql.conf配置文件来设置这些参数。 请注意，SHOW可以查看的某些参数是只读的 - 可以查看但不能设置它们的值。 有关详细信息，请参见MPP数据库参考手册。

参数

- configuration_parameter*
系统配置参数的名称。
- ALL
显示所有配置参数的当前值。

示例

显示参数DateStyle的当前设置：

```
SHOW DateStyle;
DateStyle
-----
ISO, MDY
(1 row)
```

显示参数`geqo`的当前设置：

```
SHOW geqo;
geqo
-----
off
(1 row)
```

显示所有参数的当前设置：

```
SHOW ALL;
name      | setting | description
-----+-----+-----
application_name | psql    | Sets the application name to be reported in sta...
.
.
.
xmlbinary    | base64  | Sets how binary values are to be encoded in XML.
xmloption    | content | Sets whether XML data in implicit parsing and s...
(331 rows)
```

兼容性

SHOW是MPP数据库扩展。

另见

SET , RESET

Parent topic: [SQL Command Reference](#)

START TRANSACTION

启动事务块。

概要

```
START TRANSACTION [transaction_mode] [READ WRITE | READ ONLY]
```

其中*transaction_mode*是：

```
ISOLATION LEVEL {SERIALIZABLE | READ COMMITTED | READ UNCOMMITTED}
```

描述

START TRANSACTION开始一个新的事务块。 如果指定了隔离级别或读/写模式，则新事务具有那些特征， 就像执行[SET TRANSACTION](#)一样。 这与BEGIN命令相同。

参数

READ UNCOMMITTED

READ COMMITTED

REPEATABLE READ

SERIALIZABLE

SQL标准定义了四个事务隔离级别：READ UNCOMMITTED，READ COMMITTED，REPEATABLE READ和SERIALIZABLE。

READ UNCOMMITTED允许事务查看未提交的并发事务所做的更改。在MPP数据库中这是不可能的，因此READ UNCOMMITTED与READ COMMITTED相同。

READ COMMITTED是MPP数据库中的默认隔离级别，可确保语句只能看到在其开始之前提交的行。如果在第一次执行该语句后又提交了另一个并发事务，则在一个事务中执行两次的同一条语句可能会产生不同的结果。

REPEATABLE READ隔离级别确保事务只能看到在事务开始之前提交的行。REPEATABLE READ是MPP数据库支持的最严格的事务隔离级别。由于可串行化失败，必须准备使用REPEATABLE READ隔离级别的应用程序以重试事务。

SERIALIZABLE事务隔离级别确保执行多个并发事务与串行运行这些事务产生相同的效果。如果指定SERIALIZABLE，则MPP数据库将退回到REPEATABLE READ。

READ WRITE

READ ONLY

确定事务是读/写还是只读。读/写是默认设置。当事务为只读时，不允许使用以下SQL命令：INSERT，UPDATE，DELETE和COPY FROM（如果要写入的表不是临时表）；所有的CREATE，ALTER和DROP命令；GRANT，REVOKE，TRUNCATE；如果EXPLAIN ANALYZE和EXECUTE将要执行的命令在上述命令之中，也不被允许。和EXPLAIN ANALYZE和EXECUTE，如果它们将要执行的命令在列出的命令之中。

示例

启动一个事务块：

```
START TRANSACTION;
```

兼容性

在标准中，不必发出START TRANSACTION来启动事务块：任何SQL命令都隐式地开始一个块。 MPP数据库行为可以看作是在每个不遵循START TRANSACTION（或BEGIN）的命令之后隐式发出COMMIT，因此通常被称为“自动提交”。 其他关系数据库系统可能会提供便利的自动提交功能。

SQL标准要求连续的*transaction_modes*之间使用逗号，但是由于历史原因，MPP数据库允许省略逗号。

另请参阅[SET TRANSACTION](#)的兼容性部分。

另见

[BEGIN, SET TRANSACTION](#)

Parent topic: [SQL Command Reference](#)

TRUNCATE

清空表的所有行。

概要

```
TRUNCATE [TABLE] [ONLY] name [ * ] [, ...]  
[ RESTART IDENTITY | CONTINUE IDENTITY ] [CASCADE | RESTRICT]
```

描述

TRUNCATE快速删除一个表或一组表中的所有行。它与在每个表上进行不限定DELETE的效果相同，但是由于它实际上并不扫描表，因此速度更快。这在大型表上最有用。

您必须在表上具有TRUNCATE特权才能清空表行。

TRUNCATE在对其操作的表上获取访问排他锁，这将阻止该表上的所有其他并发操作。如果指定了RESTART IDENTITY，则将要重新计数的所有序列都将被排他地锁定。如果需要并发访问表，则应改用DELETE命令。

参数

name

要清空的表的名称（可以用schema修饰）。如果在表名之前指定了ONLY，则仅该表被清空。如果未指定ONLY，则该表及其所有子表（如果有）将被清空。（可选）可以在表名称后指定*，以显式指示包括

子表。

CASCADE

因为此关键字适用于外键引用（MPP数据库中不支持），所以它无效。

RESTART IDENTITY

自动重新启动被清空表的列所拥有的序列。

CONTINUE IDENTITY

不要更改序列的值。这是默认值。

RESTRICT

因为此关键字适用于外键引用（MPP数据库中不支持），所以它无效。

注解

TRUNCATE将不会运行表可能存在的任何用户定义的ON DELETE触发器。

TRUNCATE不会清空从指定表继承的任何表。仅指定的表被清空，而不是其子表。

TRUNCATE不会清空分区表的任何子表。如果指定分区表的子表，则TRUNCATE不会从该表及其子表中删除行。

TRUNCATE不是MVCC安全的。清空后，如果并发事务使用清空发生之前获取的快照，则该表将对并发事务显示为空。

对于表中的数据，TRUNCATE是事务安全的：如果所属的事务未提交，则清空将被安全地回滚。

TRUNCATE在其操作的每个表上获取ACCESS EXCLUSIVE锁，该锁将阻止该表上的所有其他并发操作。如果需要并发访问表，则应改用DELETE命令。

当指定了RESTART IDENTITY时，隐式的ALTER SEQUENCE RESTART操作也将以事务方式完成；也就是说，如果所在的事务没有提交，它们将被回滚。这不同于ALTER SEQUENCE RESTART的正常行为。请注意，如果在事

务回滚之前对重新启动的序列执行了任何其他序列操作，则这些操作对序列的影响将被回滚，但不会对currval()产生影响。也就是说，在事务currval()之后，它将继续反映在失败的事务中获得的最后一个序列值，即使序列本身可能不再与此一致。这类似于事务失败后currval()的通常行为。

示例

清空表films和distributors：

```
TRUNCATE films, distributors;
```

相同，并重置所有关联的序列生成器：

```
TRUNCATE films, distributors RESTART IDENTITY;
```

兼容性

SQL：2008标准包括带有语法TRUNCATE TABLE *tablename*的TRUNCATE命令。子句CONTINUE IDENTITY/RESTART IDENTITY也出现在该标准中，但是尽管具有相关的含义，但略有不同。该命令的一些并发行为由标准实现定义，因此应考虑上述注意事项，并在必要时与其他实现进行比较。

另见

[DELETE](#) , [DROP TABLE](#)

Parent topic: [SQL Command Reference](#)

UPDATE

更新表的行。

概要

```
[ WITH [ RECURSIVE ] with_query [, ...] ]  
  UPDATE [ONLY] table [[AS] alias]  
    SET {column = {expression | DEFAULT} |  
        (column [, ...]) = ({expression | DEFAULT} [, ...])} [, ...]  
    [FROM fromlist]  
    [WHERE condition] WHERE CURRENT OF cursor_name ]
```

描述

UPDATE更改所有满足条件的行中指定列的值。 只需在SET子句中提及要修改的列； 未显式修改的列将保留其先前的值。

默认情况下，UPDATE将更新指定表及其所有子表中的行。 如果只希望更新提到的特定表，则必须使用ONLY子句。

有两种方法可以使用数据库中其他表中包含的信息来修改表：使用子选择，或在

FROM子句中指定其他表。哪种技术更合适取决于具体情况。

如果指定了WHERE CURRENT OF子句，则更新的行是从指定游标中最新获取的行。

复制表不支持WHERE CURRENT OF子句。

您必须在表上或至少在要更新的列上具有UPDATE特权。您还必须拥有读取*expression*或*condition*中的任何列的SELECT特权。

Note: 默认情况下，MPP数据库为堆表的UPDATE操作获取表的EXCLUSIVE锁。启用全局死锁检测器后，堆表上UPDATE操作的锁定模式为ROW EXCLUSIVE。请参阅[全局死锁检测](#)。

输出

成功完成后，UPDATE命令将返回以下格式的命令标记：

```
UPDATE count
```

其中*count*是更新的行数。如果*count*为0，则没有符合条件的行（这不视为错误）。

参数

with_query

WITH子句允许您指定一个或多个子查询，这些子查询可以在UPDATE查询中按名称进行引用。

对于包含WITH子句的UPDATE命令，该子句只能包含SELECT命令，而WITH子句不能包含数据修改命令（INSERT，UPDATE或DELETE）。查询（SELECT语句）也可能包含WITH子句。在这种情况下，可以在UPDATE查询中引用两组*with_query*，但是第二组优先，因为它的嵌套更紧密。

更多信息，请参考 [WITH查询（公用表表达式）](#) 和 [SELECT](#)。

ONLY

如果指定，则仅更新指定表中的行。如果未指定，还将处理从指定表继承的任何表。

table

现有表的名称（可以用schema修饰）。

alias

目标表的替代名称。提供别名后，它将完全隐藏表的实际名称。例如，在给定UPDATE foo AS f的情况下，UPDATE语句的其余部分必须将此表称为f而不是foo。

column

表中列的名称。如果需要，可以使用子字段名称或数组下标来限定列名称。在目标列的规范中不要包含表的名称。

expression

分配给该列的表达式。该表达式可以使用表中此列和其他列的旧值。

DEFAULT

将列设置为其默认值（如果未分配任何特定的默认表达式，则为NULL）。

fromlist

表表达式的列表，允许其他表中的列出现在WHERE条件和更新表达式中。这类似于可以在SELECT语句的FROM子句中指定的表的列表。请注意，除非您打算进行自连接，否则目标表不得出现在*fromlist*中（在这种情况下，目标表必须带有别名出现在*fromlist*中）。

condition

该表达式返回boolean类型的值。仅此表达式返回true的行将被更新。

cursor_name

在WHERE CURRENT OF条件中使用的游标名称。要更新的行是从游标最近获取的行。游标必须是UPDATE命令目标表上的非分组查询。有关创建游标的更多信息，请参见[DECLARE](#)。

不能与布尔条件一起指定WHERE CURRENT OF。

注意，不能将WHERE CURRENT OF与布尔条件一起指定。

UPDATE...WHERE CURRENT OF语句只能在服务器上执行，例如在交互式psql会话或脚本中。语言扩展（例如PL / pgSQL）不支持可更新的游

标。

有关创建游标的更多信息，请参见[DECLARE](#)。

output_expression

每行更新后，由UPDATE命令计算并返回表达式。该表达式可以使用FROM中列出的一个或多个表的任何列名。输入*以返回所有列。

output_name

用于返回的列的名称。

注解

在表的MPP分布键列上不允许使用SET。

当存在FROM子句时，本质上是将目标表连接到from列表中提到的表，并且连接的每个输出行都代表目标表的更新操作。使用FROM时，应确保该连接为要修改的每一行最多产生一个输出行。换句话说，目标行不应与其他表中的一行连接。如果是这样，那么将仅使用连接行之一来更新目标行，但是将很难预测将使用哪一行。

由于存在这种不确定性，因此仅在子选择内引用其他表会更安全，尽管与使用连接相比，通常更难阅读，也更慢。

不支持在分区表的特定分区（子表）上直接执行UPDATE和DELETE命令。而是在根分区表（使用CREATE TABLE命令创建的表）上执行这些命令。

示例

将表films中的kind列，从Drama改为Dramatic：

```
UPDATE films SET kind = 'Dramatic' WHERE kind = 'Drama';
```

调整温度条目并将表weather的一行中的降水重置为默认值：

```
UPDATE weather SET temp_lo = temp_lo+1, temp_hi =  
    temp_lo+15, prcp = DEFAULT  
    WHERE city = 'San Francisco' AND date = '2016-07-03';
```

使用替代的列列表语法进行相同的更新：

```
UPDATE weather SET (temp_lo, temp_hi, prcp) = (temp_lo+1,  
    temp_lo+15, DEFAULT)  
    WHERE city = 'San Francisco' AND date = '2016-07-03';
```

使用FROM子句语法增加管理Acme Corporation帐户的销售人员的销售数量（假设两个被连接的表在MPP数据库中都基于id列分布）：

```
UPDATE employees SET sales_count = sales_count + 1 FROM  
    accounts  
    WHERE accounts.name = 'Acme Corporation'  
    AND employees.id = accounts.id;
```

使用WHERE子句中的子选择来执行相同的操作：

```
UPDATE employees SET sales_count = sales_count + 1 WHERE id =  
    (SELECT id FROM accounts WHERE name = 'Acme Corporation');
```

尝试插入新的库存商品以及库存数量。如果物料已经存在，请更新现有物料的库存数量。要在不使整个事务失败的情况下执行此操作，请使用保存点。

```
BEGIN;  
-- other operations  
SAVEPOINT sp1;  
INSERT INTO wines VALUES('Chateau Lafite 2003', '24');  
-- Assume the above fails because of a unique key violation,  
-- so now we issue these commands:  
ROLLBACK TO sp1;  
UPDATE wines SET stock = stock + 24 WHERE winename = 'Chateau  
Lafite 2003';  
-- continue with other operations, and eventually  
COMMIT;
```

兼容性

该命令符合SQL标准，但FROM子句是MPP数据库扩展。

根据标准，列列表语法应允许从单个行值表达式（例如子选择）分配列列表：

```
UPDATE accounts SET (contact_last_name, contact_first_name) =  
    (SELECT last_name, first_name FROM salesmen  
     WHERE salesmen.id = accounts.sales_id);
```

当前尚未实现-源必须是独立表达式的列表。

其他一些数据库系统提供FROM选项，该目标表应该在FROM中再次列出。那不是MPP数据库解释FROM的方式。移植使用此扩展名的应用程序时请小心。

另见

[DECLARE](#) , [DELETE](#) , [SELECT](#) , [INSERT](#)

Parent topic: [SQL Command Reference](#)

VACUUM

垃圾收集并可选地分析数据库。

概要

```
VACUUM [{ FULL | FREEZE | VERBOSE | ANALYZE } [, ...]] [table [(column [, ...] )]]
```

```
VACUUM [FULL] [FREEZE] [VERBOSE] [table]
```

```
VACUUM [FULL] [FREEZE] [VERBOSE] ANALYZE  
      [table [(column [, ...] )]]
```

描述

VACUUM回收已删除的元组占用的存储。在正常的MPP数据库操作中，被更新删除或废弃的元组不会从表中物理删除；它们将保留在磁盘上，直到完成VACUUM。因此，有必要定期进行VACUUM，尤其是在频繁更新的表上。

如果没有参数，VACUUM将处理当前数据库中的每个表。使用参数，VACUUM仅处理该表。

VACUUM ANALYZE对每个选定的表执行VACUUM，然后执行ANALYZE。这是日常维护脚本的便捷组合形式。有关处理的更多详细信息，请参见[ANALYZE](#)。

VACUUM（无FULL）在表和索引中标记已删除和过时的数据以供将来重用，并且仅当该空间位于表的末尾并且可以轻松获得独占表锁时，才可以回收该空间以供重用。表开始或中间的未使用空间保持不变。对于堆表，这种形式的命令可以与表的正常读取和写入并行运行，因为不会获得排他锁。但是，在大多数情况下，多余的空间不会返回给操作系统。它只是可以在同一张表中重复使用。VACUUM FULL将表的全部内容重写为一个没有额外空间的新磁盘文件，从而允许将未使用的空间返回给操作系统。这种形式要慢得多，并且在处理过程中需要在每个表上获得排他锁。

使用追加优化表，VACUUM首先清理索引，然后依次压缩每个段文件，最后清理辅助表并更新统计信息，从而压缩表。在每个段上，将可见行从当前段文件复制到新的段文件，然后计划删除当前段文件，并使新的段文件可用。追加优化表的普通VACUUM允许在压缩段文件的同时扫描，插入，删除和更新表。但是，将使用Access Exclusive锁短暂删除当前段文件并激活新的段文件。

VACUUM FULL进行更广泛的处理，包括在块之间移动元组以尝试将表压缩到最小数量的磁盘块。这种形式要慢得多，并且在处理每个表时都需要申请Access Exclusive锁。Access Exclusive锁可确保所有者是以任何方式访问表的唯一事务。

当选项列表用括号括起来时，可以按任何顺序写入选项。不带括号的选项必须严格按照上面显示的顺序指定。括号中的语法是在MPP数据库6.0中添加的；不带括号的语法已弃用。

Important: 有关使用VACUUM，VACUUM FULL和VACUUM ANALYZE的信息，请参阅[注解](#)。

输出

当指定了VERBOSE时，VACUUM发出进度消息以指示当前正在处理哪个表。还将打印有关表格的各种统计信息。

参数

FULL

选择full vacuum，这可以回收更多空间，但是需要更长的时间排他锁定表。此方法还需要额外的磁盘空间，因为它会写入表的新副本，并且在操作完成之前不会释放旧副本。通常，仅在需要从表中回收大量空间时才应使用此选项。

FREEZE

指定FREEZE等效于将vacuum_freeze_min_age服务器配置参数设置为零来执行VACUUM。请参阅[服务器配置参数](#)以获取有关vacuum_freeze_min_age的信息。

VERBOSE

为每个表打印详细的vacuum活动报告。

ANALYZE

更新优化器使用的统计信息，以确定执行查询的最快方法。

table

要vacuum的表的名称（可以用schema修饰）。默认为当前数据库中的所有表。

column

要分析的特定列的名称。默认为所有列。如果指定了列列表，则意味着ANALYZE。

注解

VACUUM无法在事务块内执行。

频繁（至少每晚一次）vacuum活跃数据库，以便删除过期的行。添加或删除大量行后，对受影响的表运行VACUUM ANALYZE命令可能会很有用。这将使用所有最近更改的结果来更新系统catalog，并使MPP数据库查询优化器可以在计划查询中做出更好的选择。

Important: PostgreSQL有一个单独的可选服务器进程，称为*autovacuum*

daemon，其目的是自动执行VACUUM和ANALYZE命令。MPP数据库开启autovacuum守护程序仅在MPP数据库模板数据库template0上执行VACUUM操作。为template0启用了autovacuum，因为不允许连接到template0。autovacuum守护程序在template0上执行VACUUM操作以管理事务ID（XID），并帮助避免template0中的事务ID环绕问题。

必须在用户定义的数据库中执行手动VACUUM操作，以管理这些数据库中的事务ID（XID）。

VACUUM导致I/O流量大幅增加，这可能会导致其他活动会话的性能下降。因此，建议在低使用率时vacuum数据库。

VACUUM命令跳过外部表。

VACUUM FULL回收所有过期的行空间，但是它需要对每个正在处理的表进行独占锁定，这是一项非常昂贵的操作，并且可能需要很长时间才能完成大型分布式MPP数据库表。在数据库维护期间执行VACUUM FULL操作。

不建议例行使用FULL选项，但在特殊情况下可能有用。例如，当您删除或更新了表中的大多数行，并希望该表在物理上缩小以占用更少的磁盘空间并允许更快的表扫描时。VACUUM FULL通常比普通VACUUM将表缩小更多。

作为VACUUM FULL的替代方法，您可以使用CREATE TABLE AS语句重新创建表并删除旧表。

对于附加优化表，VACUUM需要足够的可用磁盘空间以在VACUUM过程中容纳新的段文件。如果段文件中隐藏行与总行的比率小于阈值（默认为10），则不会压缩段文件。可以使用gp_appendonly_compaction_threshold服务器配置参数来配置阈值。VACUUM FULL忽略阈值并重写段文件，而不考虑比率。可以使用gp_appendonly_compaction服务器配置参数为附加优化表禁用VACUUM。有关服务器配置参数的信息，请参阅[服务器配置参数](#)。

如果在清理附加优化表时检测到并发可序列化事务，则不会压缩当前和后续段文件。如果已压缩段文件，但是在删除原始段文件的事务中检测到并发可序列化

事务，则将忽略该删除。清理完成后，这可能会使一个或两个段文件处于“等待删除”状态。

有关MPP数据库中并发控制的更多信息，请参阅*MPP数据库管理员手册*中的“例行系统维护任务”。

示例

要清理单个表onek，请对它进行优化分析并打印详细的vacuum活动报告：

```
VACUUM (VERBOSE, ANALYZE) onek;
```

清理当前数据库中的所有表：

```
VACUUM;
```

仅清理特定表：

```
VACUUM (VERBOSE, ANALYZE) mytable;
```

清理当前数据库中的所有表并收集查询优化器的统计信息：

```
VACUUM ANALYZE;
```

兼容性

SQL标准中没有VACUUM语句。

另见

[ANALYZE](#)

Parent topic: [SQL Command Reference](#)

VALUES

计算一组行。

概要

```
VALUES ( expression [, ...] ) [, ...]  
      [ORDER BY sort_expression [ ASC | DESC | USING operator ] [, ...]  
      ]  
      [LIMIT { count | ALL } ]  
      [OFFSET start [ ROW | ROWS ] ]  
      [FETCH { FIRST | NEXT } [count] { ROW | ROWS } ONLY ]
```

描述

VALUES计算值表达式指定的行值或行值集。 它最常用于在较大的命令中生成“常数表”，但可以单独使用。

指定多于一行时，所有行必须具有相同数量的元素。通过使用与UNION相同的规则，组合出现在该列中的表达式的显式或推断类型，来确定结果表的列的数据类型。

在较大的命令中，语法上允许SELECT所在的任何位置使用VALUES。由于语法将其视为SELECT，因此可以将VALUES命令与ORDER BY，LIMIT（或等效的FETCH FIRST）和OFFSET子句一起使用。

参数

expression

要计算并插入到结果表中指定位置（行集）的常量或表达式。在出现在INSERT顶层的VALUES列表中，可以将表达式替换为DEFAULT以指示应插入目标列的默认值。当VALUES出现在其他上下文中时，不能使用DEFAULT。

sort_expression

表示如何对结果行进行排序的表达式或整数常量。该表达式可以将VALUES结果的列称为column1，column2等。有关更多详细信息，请参见[SELECT](#)参数中的“ORDER BY子句”。

operator

排序运算符。有关更多详细信息，请参见[SELECT](#)参数中的“ORDER BY子句”。

LIMIT count

OFFSET start

要返回的最大行数。有关更多详细信息，请参见[SELECT](#)参数中的“LIMIT子句”。

注解

应该避免使用具有大量行的VALUES列表，因为您可能会遇到内存不足的故障

或性能不佳的情况。出现在INSERT中的VALUES是一种特殊情况（因为所需的列类型是从INSERT的目标表中知道的，不需要通过扫描VALUES列表来推断），因此它可以处理比其他情况下实际更大的列表。

示例

一条简单的VALUES命令：

```
VALUES (1, 'one'), (2, 'two'), (3, 'three');
```

这将返回一个两列三行的表。它实际上等效于：

```
SELECT 1 AS column1, 'one' AS column2
UNION ALL
SELECT 2, 'two'
UNION ALL
SELECT 3, 'three';
```

通常，在较大的SQL命令中使用VALUES。最常见的用法是在INSERT中：

```
INSERT INTO films (code, title, did, date_prod, kind)
VALUES ('T_601', 'Yojimbo', 106, '1961-06-16', 'Drama');
```

在INSERT的上下文中，VALUES列表的条目可以为DEFAULT，以指示此处应使用列默认值而不是指定值：

```
INSERT INTO films VALUES
('UA502', 'Bananas', 105, DEFAULT, 'Comedy', '82
minutes'),
('T_601', 'Yojimbo', 106, DEFAULT, 'Drama', DEFAULT);
```

VALUES也可以用于可能写入sub-SELECT的地方，例如在FROM子句中：

```
SELECT f.* FROM films f, (VALUES('MGM', 'Horror'), ('UA',  
'Sci-Fi')) AS t (studio, kind) WHERE f.studio = t.studio AND  
f.kind = t.kind;  
UPDATE employees SET salary = salary * v.increase FROM  
(VALUES(1, 200000, 1.2), (2, 400000, 1.4)) AS v (depno,  
target, increase) WHERE employees.depno = v.depno AND  
employees.sales >= v.target;
```

请注意，在FROM子句中使用VALUES时，需要AS子句，就像SELECT一样。不需要AS子句为所有列指定名称，但是这样做是一种好习惯。在MPP数据库中，VALUES的默认列名称为column1，column2等，但是在其他数据库系统中，这些名称可能不同。

在INSERT中使用VALUES时，所有值都将自动强制转换为相应目标列的数据类型。在其他上下文中使用它时，可能有必要指定正确的数据类型。如果所有条目都是用引号括起来的文字常量，则强制第一个足以确定所有假定的类型：

```
SELECT * FROM machines WHERE ip_address IN  
(VALUES('192.168.0.1'::inet), ('192.168.0.10'),  
(192.0.2.43));
```

Note: 对于简单的IN测试，最好依靠IN的标量列表形式，而不要像上面所示编写VALUES查询。标量方法列表需要较少的编写，并且通常效率更高。

兼容性

VALUES符合SQL标准。LIMIT和OFFSET是MPP数据库扩展；另请参见[SELECT](#)下的内容。

另见

INSERT , SELECT

Parent topic: SQL Command Reference

SQL 2008可选特性兼容性

下表列出了2008 SQL标准中描述的特性。 MPP数据库支持的特性在“支持”列中标记为“是”，未实现的特性标记为“否”。

有关MPP特性和SQL兼容性的信息，请参阅MPP数据库管理员手册。

Table 1. SQL 2008可选特性合规细节

ID	特性	支持	注解
B011	嵌入式Ada	NO	
B012	嵌入式C	NO	由于PostgreSQL ecpg的问题
B013	嵌入式COBOL	NO	
B014	嵌入式Fortran	NO	
B015	嵌入式MUMPS	NO	
B016	嵌入式Pascal	NO	
B017	嵌入式PL/I	NO	
B021	直接SQL	YES	
B031	基本动态SQL	NO	

ID	特性	支持	注解
B032	扩展动态SQL	NO	
B033	无类型的SQL调用函数参数	NO	
B034	游标属性的动态规范	NO	
B035	非扩展描述符名称	NO	
B041	嵌入式SQL异常声明的扩展	NO	
B051	增强的执行权限	NO	
B111	模块语言Ada	NO	
B112	模块语言C	NO	
B113	模块语言COBOL	NO	
B114	模块语言Fortran	NO	
B115	模块语言MUMPS	NO	
B116	模块语言Pascal	NO	
B117	模块语言PL/I	NO	
B121	常规语言Ada	NO	
B122	常规语言C	NO	
B123	常规语言COBOL	NO	
B124	常规语言Fortran	NO	
B125	常规语言MUMPS	NO	
B126	常规语言Pascal	NO	
B127	常规语言PL/I	NO	
B128	常规语言SQL	NO	
E011	数值数据类型	YES	

ID	特性	支持	注解
E011-01	INTEGER和SMALLINT数据类型	YES	
E011-02	DOUBLE PRECISION和FLOAT数据类型	YES	
E011-03	DECIMAL和NUMERIC数据类型	YES	
E011-04	算术操作符	YES	
E011-05	数值比较	YES	
E011-06	数值数据类型之间的隐式转换	YES	
E021	字符数据类型	YES	
E021-01	CHARACTER数据类型	YES	
E021-02	CHARACTER VARYING数据类型	YES	
E021-03	字符文字	YES	
E021-04	CHARACTER_LENGTH函数	YES	在计数之前修剪CHARACTER值的尾随空格
E021-05	OCTET_LENGTH函数	YES	
E021-06	SUBSTRING函数	YES	

ID	特性	支持	注解
E021-07	字符连接	YES	
E021-08	UPPER和LOWER函数	YES	
E021-09	TRIM函数	YES	
E021-10	字符串类型之间隐式转换	YES	
E021-11	POSITION函数	YES	
E021-12	字符比较	YES	
E031	标识符	YES	
E031-01	分隔标识符	YES	
E031-02	小写标识符	YES	
E031-03	尾随下划线	YES	
E051	基本查询规范	YES	
E051-01	SELECT DISTINCT	YES	
E051-02	GROUP BY子句	YES	
E051-03	GROUP BY可以包含不在SELECT列表中的列	YES	

ID	特性	支持	注解
E051-04	可以重命名SELECT列表项	YES	
E051-05	HAVING子句	YES	
E051-06	SELECT列表中受限定的*	YES	
E051-07	FROM子句中的关联名称	YES	
E051-08	重命名FROM子句中的列	YES	
E061	基本谓词和搜索条件	YES	
E061-01	比较谓词	YES	
E061-02	BETWEEN谓词	YES	
E061-03	带有值列表的IN谓词	YES	
E061-04	LIKE谓词	YES	
E061-05	LIKE谓词ESCAPE 子句	YES	
E061-06	NULL谓词	YES	
E061-07	量化比较谓词	YES	

ID	特性	支持	注解
E061-08	EXISTS谓词	YES	并非所有用途都适用于MPP
E061-09	比较谓词中的子查询	YES	
E061-11	IN谓词中的子查询	YES	
E061-12	量化比较谓词中的子查询	YES	
E061-13	相关子查询	YES	
E061-14	搜索条件	YES	
E071	基本查询表达式	YES	
E071-01	UNION DISTINCT表操作符	YES	
E071-02	UNION ALL表操作符	YES	
E071-03	EXCEPT DISTINCT表操作符	YES	
E071-05	通过表操作符组合的列不需要具有完全相同的数据类型	YES	
E071-06	子查询中的表操作符	YES	
E081	基本权限	NO	支持部分子特性
E081-01	SELECT权限	YES	

ID	特性	支持	注解
E081-02	DELETE权限	YES	
E081-03	INSERT表级别的权限	YES	
E081-04	UPDATE表级别的权限	YES	
E081-05	UPDATE列级别的权限	YES	
E081-06	REFERENCES表级的权限	NO	
E081-07	REFERENCES列级别的权限	NO	
E081-08	WITH GRANT OPTION	YES	
E081-09	USAGE权限	YES	
E081-10	EXECUTE权限	YES	
E091	集合函数	YES	
E091-01	AVG	YES	
E091-02	COUNT	YES	
E091-03	MAX	YES	

ID	特性	支持	注解
E091-04	MIN	YES	
E091-05	SUM	YES	
E091-06	ALL量词	YES	
E091-07	DISTINCT量词	YES	
E101	基本数据操作	YES	
E101-01	INSERT语句	YES	
E101-03	搜索UPDATE语句	YES	
E101-04	搜索DELETE语句	YES	
E111	单行SELECT语句	YES	
E121	基本光标支持	YES	
E121-01	DECLARE CURSOR	YES	
E121-02	ORDER BY列不需要在选择列表中	YES	
E121-03	ORDER BY子句的值表达式	YES	
E121-04	OPEN语句	YES	

ID	特性	支持	注解
E121-06	Positioned UPDATE语句	NO	
E121-07	Positioned DELETE语句	NO	
E121-08	CLOSE语句	YES	
E121-10	FETCH语句隐含 NEXT	YES	
E121-17	WITH HOLD游标	YES	
E131	空值支持	YES	
E141	基本完整性约束	YES	
E141-01	NOT NULL约束	YES	
E141-02	UNIQUE NOT NULL 列约束	YES	必须与MPP分布键相同或是它的超集
E141-03	PRIMARY KEY约束	YES	必须与MPP分布键相同或是它的超集
E141-04	参考删除操作和参照更新操作的NO ACTION默认值的基本FOREIGN KEY约束	NO	
E141-06	CHECK约束	YES	
E141-07	列默认值	YES	

ID	特性	支持	注解
E141-08	NOT NULL推断PRIMARY KEY	YES	
E141-10	外键中的名称可以按任何顺序指定	YES	可以声明外键，但在MPP中不会强制执行
E151	事务支持	YES	
E151-01	COMMIT语句	YES	
E151-02	ROLLBACK语句	YES	
E152	基本SET TRANSACTION语句	YES	
E152-01	ISOLATION LEVEL SERIALIZABLE子句	NO	可以声明，但被视为REPEATABLE READ的同义词
E152-02	READ ONLY和READ WRITE 子句	YES	
E153	具有子查询的可更新查询	NO	
E161	使用前导双减去的SQL注释	YES	
E171	SQLSTATE支持	YES	
E182	模块语言	NO	
F021	基本信息模式	YES	
F021-01	COLUMNS视图	YES	
F021-02	TABLES视图	YES	
F021-03	VIEWS视图	YES	

ID	特性	支持	注解
F021-04	TABLE_CONSTRAINTS视图	YES	
F021-05	REFERENTIAL_CONSTRAINTS视图	YES	
F021-06	CHECK_CONSTRAINTS视图	YES	
F031	基本模式操作	YES	
F031-01	CREATE TABLE语句创建持久性基表	YES	
F031-02	CREATE VIEW语句	YES	
F031-03	GRANT语句	YES	
F031-04	ALTER TABLE语句：ADD COLUMN子句	YES	
F031-13	DROP TABLE语句：RESTRICT子句	YES	
F031-16	DROP VIEW语句：RESTRICT子句	YES	
F031-19	REVOKE语句：RESTRICT子句	YES	
F032	CASCADE删除行为	YES	
F033	ALTER TABLE语句：DROP COLUMN子句	YES	
F034	扩展的REVOKE语句	YES	

ID	特性	支持	注解
F034-01	REVOKE语句由schema对象的所有者以外的其他语句执行	YES	
F034-02	REVOKE语句：GRANT OPTION FOR子句	YES	
F034-03	REVOKE语句撤销授权人WITH GRANT OPTION的权限	YES	
F041	基本连接表	YES	
F041-01	内连接（但不一定是INNER关键字）	YES	
F041-02	INNER关键字	YES	
F041-03	LEFT OUTER JOIN	YES	
F041-04	RIGHT OUTER JOIN	YES	
F041-05	外连接可以嵌套	YES	
F041-07	左或右外连接中的内表也可用于内连接	YES	
F041-08	支持所有比较操作符(而不仅是=)	YES	
F051	基本的日期和时间	YES	
F051-01	DATE数据类型（包括支持DATE文字）	YES	
F051-02	TIME数据类型（包括支持TIME文字）分秒精确度至少为0	YES	

ID	特性	支持	注解
F051-03	TIMESTAMP数据类型（包括支持TIMESTAMP文字）分秒精确度至少为0和6	YES	
F051-04	DATE，TIME和TIMESTAMP数据类型的比较谓词	YES	
F051-05	日期时间类型和字符串类型之间的显式CAST	YES	
F051-06	CURRENT_DATE	YES	
F051-07	LOCALTIME	YES	
F051-08	LOCALTIMESTAMP	YES	
F052	间隔和日期时间算术	YES	
F053	OVERLAPS谓词	YES	
F081	视图中的UNION和EXCEPT	YES	
F111	除SERIALIZABLE之外的隔离级别	YES	
F111-01	READ UNCOMMITTED 隔离级别	NO	可以声明但是被视为READ COMMITTED的同义词
F111-02	READ COMMITTED隔离级别	YES	
F111-03	REPEATABLE READ隔离级别	YES	
F121	基本诊断管理	NO	
F122	增强诊断管理	NO	

ID	特性	支持	注解
F123	所有诊断	NO	
F131 -	分组操作	YES	
F131 -01	在分组视图的查询中支持的WHERE，GROUP BY和 HAVING子句	YES	
F131 -02	在具有分组视图的查询中支持多个表	YES	
F131 -03	设置具有分组视图的查询支持的函数	YES	
F131 -04	具有GROUP BY和HAVING子句以及分组视图的子查询	YES	
F131 -05	具有GROUP BY和HAVING子句以及分组视图的单行SELECT	YES	
F171	每个用户的多个模式	YES	
F181	多模块支持	NO	
F191	引用删除操作	NO	
F200	TRUNCATE TABLE语句	YES	
F201	CAST函数	YES	
F202	TRUNCATE TABLE：标识列重新启动选项	NO	
F221	显式默认值显式默认值	YES	
F222	INSERT语句：DEFAULT VALUES子句	YES	
F231	特权表	YES	

ID	特性	支持	注解
F231-01	TABLE_PRIVILEGES视图	YES	
F231-02	COLUMN_PRIVILEGES视图	YES	
F231-03	USAGE_PRIVILEGES视图	YES	
F251	域支持		
F261	CASE表达式	YES	
F261-01	简单CASE	YES	
F261-02	搜索CASE	YES	
F261-03	NULLIF	YES	
F261-04	COALESCE	YES	
F262	扩展的CASE表达式	NO	
F263	简单CASE表达式中逗号分隔的谓词	NO	
F271	复合字符文字	YES	
F281	LIKE增强	YES	
F291	UNIQUE谓词	NO	
F301	查询表达式中的CORRESPONDING	NO	
F302	INTERSECT表操作符	YES	

ID	特性	支持	注解
F302-01	INTERSECT DISTINCT表操作符	YES	
F302-02	INTERSECT ALL表操作符	YES	
F304	EXCEPT ALL table operator		
F311	模式定义语句	YES	支持部分子特性
F311-01	CREATE SCHEMA	YES	
F311-02	CREATE TABLE用于持久性基表	YES	
F311-03	CREATE VIEW	YES	
F311-04	CREATE VIEW: WITH CHECK OPTION	NO	
F311-05	GRANT语句	YES	
F312	MERGE语句	NO	
F313	增强的MERGE语句	NO	
F321	用户授权	YES	
F341	用法表	NO	
F361	子程序支持	YES	
F381	扩展的模式操作	YES	
F381-01	ALTER TABLE语句：ALTER COLUMN子句		更改分布键列的一些限制

ID	特性	支持	注解
F381-02	ALTER TABLE语句：ADD CONSTRAINT子句		
F381-03	ALTER TABLE语句：DROP CONSTRAINT子句		
F382	更改列数据类型	YES	更改分布键列的一些限制
F391	长标识符	YES	
F392	Unicode转义为标识符	NO	
F393	Unicode以文字形式转义	NO	
F394	可选的正常形式规格	NO	
F401	扩展连接表	YES	
F401-01	NATURAL JOIN	YES	
F401-02	FULL OUTER JOIN	YES	
F401-04	CROSS JOIN	YES	
F402	命名列连接LOB，数组和多重集	NO	
F403	分区连接表	NO	
F411	时区规格	YES	关于字面解释的差异
F421	国家特征	YES	
F431	只读可滚动光标	YES	仅向前滚动
01	有显式NEXT的FETCH	YES	
02	FETCH FIRST	NO	

ID	特性	支持	注解
03	FETCH LAST	YES	
04	FETCH PRIOR	NO	
05	FETCH ABSOLUTE	NO	
06	FETCH RELATIVE	NO	
F441	扩展集函数支持	YES	
F442	集合函数中的混合列引用	YES	
F451	字符集定义	NO	
F461	命名字符集	NO	
F471	标量子查询值	YES	
F481	扩展的NULL谓词	YES	
F491	约束管理	YES	
F501	特征和一致性视图	YES	
F501-01	SQL_FEATURES视图	YES	
F501-02	SQL_SIZING视图	YES	
F501-03	SQL_LANGUAGES视图	YES	
F502	增强的文档表	YES	
F502-01	SQL_SIZING_PROFILES视图	YES	
F502-02	SQL_IMPLEMENTATION_INFO视图	YES	

ID	特性	支持	注解
F502-03	SQL_PACKAGES视图	YES	
F521	断言	NO	
F531	临时表	YES	非标准表格
F555	增强秒精度	YES	
F561	全值表达式	YES	
F571	真值测试	YES	
F591	派生表	YES	
F611	指标数据类型	YES	
F641	行和表构造函数	NO	
F651	Catalog名称限定	YES	
F661	简单表	NO	
F671	CHECK中的子查询	NO	故意省略
F672	回溯检查限制	YES	
F690	排序规则支持	NO	
F692	增强排序规则支持	NO	
F693	SQL会话和客户端模块排序规则	NO	
F695	翻译支持	NO	
F696	附加翻译文件	NO	
F701	引用更新动作	NO	
F711	ALTER域	YES	
F721	可延迟的约束	NO	
F731	INSERT列权限	YES	

ID	特性	支持	注解
F741	引用MATCH类型	NO	没有部分匹配
F751	视图CHECK增强	NO	
F761	会话管理	YES	
F762	CURRENT_CATALOG	NO	
F763	CURRENT_SCHEMA	NO	
F771	连接管理	YES	
F781	自引用操作	YES	
F791	不敏感的游标	YES	
F801	全套函数	YES	
F812	基本标记	NO	
F813	扩展标记	NO	
F831	完整游标更新	NO	
F841	LIKE_REGEX谓词	NO	正则表达式的非标准语法
F842	OCCURENCES_REGEX函数	NO	
F843	POSITION_REGEX函数	NO	
F844	SUBSTRING_REGEX函数	NO	
F845	TRANSLATE_REGEX函数	NO	
F846	正则表达式操作符中的八位字节支持	NO	
F847	非常规正则表达式	NO	
F850	查询表达式中的顶级ORDER BY子句	YES	
F851	子查询中的顶级ORDER BY子句	NO	

ID	特性	支持	注解
F852	视图中的顶级ORDER BY子句	NO	
F855	查询表达式中的嵌套ORDER BY子句	NO	
F856	查询表达式中的嵌套FETCH FIRST子句	NO	
F857	查询表达式中的顶级FETCH FIRST子句	NO	
F858	子查询中的FETCH FIRST子句	NO	
F859	视图中的顶级FETCH FIRST子句	NO	
F860	FETCH FIRST子句中的FETCH FIRST ROW数量	NO	
F861	查询表达式中的顶级RESULT OFFSET子句	NO	
F862	子查询中的RESULT OFFSET子句	NO	
F863	查询表达式中的嵌套RESULT OFFSET子句	NO	
F864	视图中的顶级RESULT OFFSET子句	NO	
F865	RESULT OFFSET子句中的OFFSET ROW数量	NO	
S011	不同的数据类型	NO	
S023	基本结构化类型	NO	
S024	增强的结构化类型	NO	
S025	最终结构化类型	NO	

ID	特性	支持	注解
S026	自引用结构化类型	NO	
S027	按特定方法名称创建方法	NO	
S028	可置换的UDT选项列表	NO	
S041	基本参考类型	NO	
S043	增强的参考类型	NO	
S051	创建类型表	NO	
S071	函数和类型名称解析中的SQL路径	YES	
S091	基本数组支持	NO	MPP有数组，但不完全符合标准
S091-01	内置数据类型的数组	NO	部分兼容
S091-02	不同类型的数组	NO	
S091-03	数组表达式	NO	
S092	用户定义类型的数组	NO	
S094	参考类型的数组	NO	
S095	通过查询的数组构造函数	NO	
S096	可选数组边界	NO	
S097	数组元素赋值	NO	

ID	特性	支持	注解
S098	ARRAY_AGG	部分支持	支持：使用没有窗口规范的array_agg；例如 <pre>SELECT array_agg(x) FROM ...</pre> <pre>SELECT array_agg (x order by y) FROM ..</pre> 不支持：使用array_agg作为聚合派生窗口函数；例如 <pre>SELECT array_agg(x) over (ORDER BY y) FROM ...</pre> <pre>SELECT array_agg(x order by y) over (PARTITION BY z) FROM ...</pre> <pre>SELECT array_agg(x order by y) over (ORDER BY z) FROM ..</pre>
S111	ONLY在查询表达式中	YES	
S151	类型谓词	NO	

ID	特性	支持	注解
S161	子类型处理	NO	
S162	引用的子类型处理	NO	
S201	SQL上调用的数组例程	NO	函数可以传递MPP数组类型
S202	多集上的SQL调用例程	NO	
S211	用户定义的强制转换函数	YES	
S231	结构类型定位器	NO	
S232	数组定位器	NO	
S233	Multiset定位器	NO	
S241	转换函数	NO	
S242	改变转换语句	NO	
S251	用户定义的命令	NO	
S261	特定类型方法	NO	
S271	基本的多重集支持	NO	
S272	用户自定义类型的多重集	NO	
S274	引用类型的多重集	NO	
S275	高级多重集支持	NO	
S281	嵌套集合类型	NO	
S291	整个行的唯一约束	NO	
S301	增强的UNNEST	NO	
S401	基于数组类型的不同类型	NO	
S402	基于不同类型的不同类型	NO	
S403	MAX_CARDINALITY	NO	

ID	特性	支持	注解
S404	TRIM_ARRAY	NO	
T011	信息模式中的时间戳	NO	
T021	BINARY和VARBINARY 数据类型	NO	
T022	支持高级的BINARY和VARBINARY数据类型	NO	
T023	复合二进制文字	NO	
T024	二进制文字中的空格	NO	
T031	BOOLEAN数据类型	YES	
T041	支持基本的LOB数据类型	NO	
T042	支持扩展的LOB数据类型	NO	
T043	乘数T	NO	
T044	乘数P	NO	
T051	行类型	NO	
T052	用于行类型的MAX和MIN	NO	
T053	全域引用的显式别名	NO	
T061	支持UCS	NO	
T071	BIGINT数据类型	YES	
T101	增强的可空性确定	NO	
T111	可更新的连接，联合和列	NO	
T121	查询表达式中的WITH(除了RECURSIVE)	NO	
T122	子查询中的WITH(除了RECURSIVE)	NO	
T131	递归查询	NO	

ID	特性	支持	注解
T132	子查询中的递归查询	NO	
T141	SIMILAR谓词	YES	
T151	DISTINCT谓词	YES	
T152	DISTINCT否定谓词	NO	
T171	表定义中的LIKE子句	YES	
T172	在表中定义AS子查询子句	YES	
T173	在表定义中扩展LIKE子句	YES	
T174	身份列	NO	
T175	生成列	NO	
T176	支持序列生成器	NO	
T177	支持顺序发生器：简单的重启选项	NO	
T178	身份列：简单重启选项	NO	
T191	参照活动RESTRICT	NO	
T201	参照约束的可比较数据类型	NO	
T211	基本触发功能	NO	
T211-01	一个基本表的UPDATE, INSERT, 或 DELETE上激活触发器	NO	
T211-02	BEFORE触发器	NO	
T211-03	AFTER触发器	NO	
T211-04	FOR EACH ROW触发器	NO	

ID	特性	支持	注解
T211-05	能够在调用触发器之前指定一个必须为true的搜索条件	NO	
T211-06	支持触发器和约束的交互的运行时规则	NO	
T211-07	TRIGGER特权	YES	
T211-08	同一事件的多个触发器按其在catalog中创建的顺序执行	NO	故意省略
T212	增强的触发器功能	NO	
T213	INSTEAD OF触发器	NO	
T231	敏感游标	YES	
T241	START TRANSACTION语句	YES	
T251	SET TRANSACTION语句：LOCAL选项	NO	
T261	联锁事务	NO	
T271	保存点	YES	
T272	增强的保存点管理	NO	
T281	SELECT具有列粒度的特权	YES	
T285	增强派生列名称	NO	
T301	函数依赖	NO	
T312	OVERLAY函数	YES	
T321	基本的SQL-invoked例程	NO	部分支持
T321-01	不带重载的用户定义的函数	YES	

ID	特性	支持	注解
T321-02	不带重载的用户定义的存储过程	NO	
T321-03	函数调用	YES	
T321-04	CALL语句	NO	
T321-05	RETURN语句	NO	
T321-06	ROUTINES视图	YES	
T321-07	PARAMETERS视图	YES	
T322	SQL-invoked的函数和过程的重载	YES	
T323	外部程序的显式安全性	YES	
T324	SQL例程的显式安全性	NO	
T325	合格的SQL参数引用	NO	
T326	表函数	NO	
T331	基本角色	NO	
T332	扩展角色	NO	
T351	括号SQL注释 (/*...*/ 注释)	YES	
T431	扩展分组容量	NO	
T432	嵌套和级联GROUPING SETS	NO	
T433	多参数GROUPING函数	NO	
T434	GROUP BY DISTINCT	NO	

ID	特性	支持	注解
T441	ABS和MOD函数	YES	
T461	对称的BETWEEN谓词	YES	
T471	结果集返回值	NO	
T491	LATERAL派生表	NO	
T501	增强的EXISTS谓词	NO	
T511	事务总数	NO	
T541	可更新表引用	NO	
T561	可定位器	NO	
T571	Array-returning外部SQL-invoked 函数	NO	
T572	Multiset-returning外部SQL-invoke d函数	NO	
T581	正则表达式子串函数	YES	
T591	UNIQUE可能为空列的约束	YES	
T601	本地游标引用	NO	
T611	初级OLAP操作	YES	
T612	高级OLAP操作	NO	部分支持
T613	采样	NO	
T614	NTILE函数	YES	
T615	LEAD和LAG函数	YES	
T616	LEAD和LAG函数的空值处理选项	NO	
T617	FIRST_VALUE和LAST_VALUE 函数	YES	

ID	特性	支持	注解
T618	NTH_VALUE	NO	函数存在于MPP中，但不支持所有选项
T621	增强数字函数	YES	
T631	N谓词与一个列表元素	NO	
T641	多列分配	NO	支持一些语法变体
T651	SQL例程中的SQL-schema语句	NO	
T652	SQL例程中的SQL动态语句	NO	
T653	外部例程中的SQL-schema语句	NO	
T654	外部例程中的SQL动态语句	NO	
T655	循环依赖例程	NO	
M001	数据链路	NO	
M002	数据链路通过SQL/CLI	NO	
M003	数据链路通过嵌入式SQL	NO	
M004	外部数据支持	NO	
M005	外部模式支持	NO	
M006	GetSQLString例程	NO	
M007	TransmitRequest	NO	
M009	GetOpts和GetStatistics例程	NO	
M010	外部数据包装支持	NO	
M011	通过Ada的数据链接	NO	
M012	数据链通过C	NO	
M013	数据链通过COBOL	NO	
M014	数据链路通过Fortran	NO	

ID	特性	支持	注解
M015	数据链路通过M	NO	
M016	数据链路通过Pascal	NO	
M017	数据链路通过PL/I	NO	
M018	Ada中的外部数据包装器接口例程	NO	
M019	C中的外部数据包装器接口例程	NO	
M020	COBOL中的外部数据包装器接口例程	NO	
M021	Fortran中的外部数据包装器接口例程	NO	
M022	MUMPS中的外部数据包装器接口例程	NO	
M023	Pascal中的外部数据包装器接口例程	NO	
M024	PL/I中的外部数据包装器接口例程	NO	
M030	SQL-server外部数据支持	NO	
M031	外部数据包装程序一般例程	NO	
X010	XML类型	YES	
X011	XML类型的数组	YES	
X012	XML类型的多重集	NO	
X013	XML类型的不同类型	NO	
X014	XML类型的属性	NO	
X015	XML类型的字段	NO	
X016	持久的XML值	YES	
X020	XMLConcat	YES	支持xmlconcat2()

ID	特性	支持	注解
X025	XMLCast	NO	
X030	XMLDocument	NO	
X031	XMLElement	YES	
X032	XMLForest	YES	
X034	XMLAgg	YES	
X035	XMLAgg: ORDER BY option	YES	
X036	XMLComment	YES	
X037	XMLPI	YES	
X038	XMLText	NO	
X040	基本表映射	NO	
X041	基本表映射: 空值不存在	NO	
X042	基本表映射 : null as nil	NO	
X043	基本表映射 : 表作为森林	NO	
X044	基本表映射 : 表作为元素	NO	
X045	基本表映射 : 具有目标命名空间	NO	
X046	基本表映射 : 数据映射	NO	
X047	基本表映射 : 元数据映射	NO	
X048	基本表映射 : 二进制字符串的base 64编码	NO	
X049	基本表映射 : 二进制字符串的十六进制编码	NO	
X051	高级表映射 : 空值不存在	NO	
X052	高级表映射 : null as nil	NO	

ID	特性	支持	注解
X053	高级表映射：表作为森林	NO	
X054	高级表映射：表作为元素	NO	
X055	高级表映射：目标命名空间	NO	
X056	高级表映射：数据映射	NO	
X057	高级表映射：元数据映射	NO	
X058	高级表映射：二进制字符串的base 64编码	NO	
X059	高级表映射：二进制字符串的十六进制编码	NO	
X060	XMLParse: 字符串输入和CONTENT选项	YES	
X061	XMLParse: 字符串输入和DOCUMENT选项	YES	
X065	XMLParse: BLOB输入和CONTENT选项	NO	
X066	XMLParse: BLOB输入和DOCUMENT选项	NO	
X068	XMLSerialize: BOM	NO	
X069	XMLSerialize: INDENT	NO	
X070	XMLSerialize: 字符串序列化和CONTENT选项	YES	
X071	XMLSerialize: 字符串序列化和DOCUMENT选项	YES	
X072	XMLSerialize: 字符串序列化	YES	

ID	特性	支持	注解
X073	XMLSerialize: BLOB序列化和CONTENT选项	NO	
X074	XMLSerialize: BLOB序列化和DOCUMENT选项	NO	
X075	XMLSerialize: BLOB序列化	NO	
X076	XMLSerialize: VERSION	NO	
X077	XMLSerialize : 显式ENCODING选项	NO	
X078	XMLSerialize : 显式XML声明	NO	
X080	XML发布中的命名空间	NO	
X081	查询级XML命名空间声明	NO	
X082	DML中的XML命名空间声明	NO	
X083	DDL中的XML命名空间声明	NO	
X084	复合语句中的XML命名空间声明	NO	
X085	预定义的命名空间前缀	NO	
X086	XMLTable中的XML命名空间声明	NO	
X090	XML文档谓词	NO	支持xml_is_well_formed_document()
X091	XML内容谓词	NO	支持xml_is_well_formed_content()
X096	XMLExists	NO	支持xmlexists()
X100	主机语言支持XML : CONTENT选项	NO	
X101	主机语言支持XML : DOCUMENT选项	NO	

ID	特性	支持	注解
X110	主机语言支持XML：VARCHAR映射	NO	
X111	主机语言支持XML：CLOB映射	NO	
X112	主机语言支持XML：BLOB映射	NO	
X113	主机语言支持XML：STRIP WHITESPACE选项	YES	
X114	主机语言支持XML：PRESERVE WHITESPACE选项	YES	
X120	SQL例程中的XML参数	YES	
X121	外部例程中的XML参数	YES	
X131	查询级XMLBINARY子句	NO	
X132	DML中的XMLBINARY子句	NO	
X133	DDL中的XMLBINARY子句	NO	
X134	复合语句中的XMLBINARY子句	NO	
X135	子查询中的XMLBINARY子句	NO	
X141	IS VALID谓词：数据驱动案例	NO	
X142	IS VALID谓词：ACCORDING TO子句	NO	
X143	IS VALID谓词：ELEMENT子句	NO	
X144	IS VALID谓词：模式位置	NO	
X145	IS VALID谓词外部检查约束	NO	
X151	带有DOCUMENT选项的IS VALID谓词	NO	

ID	特性	支持	注解
X152	带有CONTENT选项的IS VALID谓词	NO	
X153	带有SEQUENCE选项的IS VALID谓词	NO	
X155	IS VALID谓词：不带ELEMENT子句的NAMESPACE	NO	
X157	IS VALID谓词：NO NAMESPACE与ELEMENT子句	NO	
X160	注册XML模式的基本信息模式	NO	
X161	注册XML模式的高级信息模式	NO	
X170	XML null处理选项	NO	
X171	NIL ON NO CONTENT选项	NO	
X181	XML(DOCUMENT (UNTYPED))类型	NO	
X182	XML(DOCUMENT (ANY))类型	NO	
X190	XML(SEQUENCE) type	NO	
X191	XML(DOCUMENT (XMLSCHEMA))类型	NO	
X192	XML(CONTENT (XMLSCHEMA))类型	NO	
X200	XMLQuery	NO	
X201	XMLQuery: RETURNING CONTENT	NO	
X202	XMLQuery: RETURNING SEQUENCE	NO	

ID	特性	支持	注解
X203	XMLQuery : 传递上下文条目	NO	
X204	XMLQuery: 初始化XQuery变量	NO	
X205	XMLQuery : EMPTY ON EMPTY 选项	NO	
X206	XMLQuery : NULL ON EMPTY选 项	NO	
X211	支持XML 1.1	NO	
X221	XML传递机制BY VALUE	NO	
X222	XML传递机制BY REF	NO	
X231	XML (CONTENT (UNTYPED)) 类型	NO	
X232	XML (CONTENT (ANY)) 类型	NO	
X241	在XML发布中返回内容	NO	
X242	在XML发布中返回序列	NO	
X251	XML (DOCUMENT (UNTYPED)) 类型的持久XML值	NO	
X252	XML (DOCUMENT (ANY)) 类 型的持久XML值	NO	
X253	XML (CONTENT (UNTYPED)) 类型的持久XML值	NO	
X254	XML (CONTENT (ANY)) 类型 的持久XML值	NO	
X255	XML (SEQUENCE) 类型的持久 XML值	NO	

ID	特性	支持	注解
X256	XML (DOCUMENT (XMLSCHEMA)) 类型的持久XML值	NO	
X257	XML的持久XML值 (CONTENT (XMLSCHEMA))	NO	
X260	XML类型：ELEMENT子句	NO	
X261	XML类型：不带ELEMENT子句的NAMESPACE	NO	
X263	XML类型：NO NAMESPACE与ELEMENT子句	NO	
X264	XML type: schema位置	NO	
X271	XMLValidate: 数据驱动的案例	NO	
X272	XMLValidate: ACCORDING TO子句	NO	
X273	XMLValidate: ELEMENT子句	NO	
X274	XMLValidate: schema位置	NO	
X281	XMLValidate: 具有DOCUMENT选项	NO	
X282	带有CONTENT选项的XMLValidat	NO	
X283	带有SEQUENCE选项的XMLValidat	NO	
X284	不带ELEMENT子句的XMLValidat e NAMESPACE	NO	
X286	XMLValidate: 带ELEMENT子句的 NO NAMESPACE	NO	
X300	XMLTable	NO	

ID	特性	支持	注解
X301	XMLTable：派生列列表选项	NO	
X302	XMLTable：ordinality列选项	NO	
X303	XMLTable：列默认选项	NO	
X304	XMLTable：传递上下文条目	NO	
X305	XMLTable：初始化XQuery变量	NO	
X400	名称和标识符映射	NO	

Parent topic: [MPP数据库参考手册](#)

MPP环境变量

该参考列出并描述了MPP中可设置的环境变量。 如果想为所有用户设置这些环境变量，需要在用户的启动shell配置文件（如~/.bashrc或~/.bash_profile或者在/etc/profile中进行设置这些环境变量。

- [必需的环境变量](#)
- [可选的环境变量](#)

Parent topic: [MPP数据库参考手册](#)

必需的环境变量

Note: GPHOME，PATH和LD_LIBRARY_PATH能够通过引入MPP安装目录下的mpp_path.sh文件来设置。

GPHOME

该参数指明了MPP安装位置。例如：

```
GPHOME=/usr/local/mpp-db-<version>  
export GPHOME
```

PATH

PATH环境变量指出MPP的bin目录所在位置。例如：

```
PATH=$GPHOME/bin:$PATH  
export PATH
```

LD_LIBRARY_PATH

LD_LIBRARY_PATH环境变量指出了MPP/PostgreSQL库文件所在位置。例如：

```
LD_LIBRARY_PATH=$GPHOME/lib  
export LD_LIBRARY_PATH
```

MASTER_DATA_DIRECTORY

该变量指出在master数据目录中通过gpinitssystem创建的目录。例如：

```
MASTER_DATA_DIRECTORY=/data/master/gpseg-1  
export MASTER_DATA_DIRECTORY
```

可选的环境变量

以下是标准的PostgreSQL环境变量，它们也在MPP数据库中被识别。为方便起见，您可能希望将与连接相关的环境变量添加到配置文件中，因此您无需在命令行上为客户端连接键入这么多选项。请注意，这些环境变量应仅在MPP数据库master主机上设置。

PGAPPNAME

应用的名称，通常当一个应用连接到服务器时设置的。该名称在活动视图和日志条目中有显示。PGAPPNAME环境变量和application_name连接参数有相同的行为。application_name的默认值是psql。该名称不能超过63个字符。

PGDATABASE

默认连接到的数据库名称。

PGHOST

MPP数据库的Master主机名称。

PGHOSTADDR

Master主机的数字IP地址。 设置该变量可以用来代替或者在PGHOST之上从而避免DNS查询的过多开销。

PGPASSWORD

服务器要求密码验证时使用的密码。 出于安全原因，不建议使用此环境变量（某些操作系统允许非root用户通过ps查看进程环境变量）。 而是考虑使用~/.pgpass文件。

PGPASSFILE

用于查找的密码文件的名称。 如果未设置，则默认为~/.pgpass。 有关详细信息，请参阅PostgreSQL文档中有关[密码文件](#)的主题。

PGOPTIONS

为MPP数据库master服务器设置其他配置参数。

PGPORT

master主机上MPP数据库服务器的端口号。 默认端口为5432。

PGUSER

用于连接的MPP数据库用户名。

PGDATESTYLE

设置会话的日期/时间表示的默认样式。（相当于SET datestyle TO...）

PGTZ

设置会话的默认时区。（相当于SET timezone TO...）

PGCLIENTENCODING

设置会话的默认客户端字符集编码。（相当于SET client_encoding TO...）

保留标识符和SQL关键字

本主题描述了MPP数据库保留标识符和对象名称，以及MPP数据库和PostgreSQL命令解析器识别的SQL关键字。

保留标识符

在MPP数据库系统中，以gp_和pg_开头的名称是保留的，不应用作用户创建的

对象的名称，例如表，视图和函数。

资源组名称admin_group，default_group和none都是保留的。资源队列名称pg_default是保留的。

表空间名称pg_default和pg_global是保留的。

角色名称gpadmin和gpmon是保留的。gpadmin是默认的MPP数据库超级用户角色。gpmon角色拥有gpperfmon数据库。

在数据文件中，分隔字段（列）和行的字符具有特殊含义。如果它们出现在数据中，则必须将它们转义，以便MPP数据库将它们视为数据而不是分隔符。反斜杠字符（\）是默认的转义字符。有关详细信息，请参见[转义](#)。

有关SQL标识符，常量，运算符和表达式的更多信息，请参阅PostgreSQL文档中的[SQL语法](#)。

SQL关键词

[Table 1](#)列出了所有token，这些token是MPP数据库6和PostgreSQL 9.4中的关键词。

ANSI SQL区分保留和未保留的关键字。根据标准，保留关键词是唯一真正的关键词;永远不允许他们作为标识符。未保留的关键词在特定上下文中仅具有特殊含义，并且可以在其他上下文中用作标识符。大多数未保留的关键字实际上是SQL指定的内置表和函数的名称。无保留关键词的概念基本上只存在于声明在某些上下文中某些预定义含义附加到单词上。

在MPP数据库和PostgreSQL解析器中，有几种不同类别的token，从永远不能用作标识符的token到与普通标识符相比在解析器中完全没有特殊状态的token。（后者通常是SQL指定的函数的情况。）即使保留的关键字也没有完全保留，但可以用作列标签（例如，SELECT 55 AS CHECK，即使CHECK是保留关键

字) 。

Table 1将那些解析器明确知道但允许作为列或表名称的关键字归类为"unreserved"。 一些未经保留的关键字不能用作函数或数据类型名称，并相应标记。（这些单词中的大部分代表具有特殊语法的内置函数或数据类型。函数或类型仍然可用，但用户无法重新定义。） 标记为“reserved”的关键字不允许作为列或表名。 一些保留的关键字可以作为函数或数据类型的名称;这也显示在表中。 如果没有这么标记，则保留的关键字仅允许作为“AS”列标签名称。

如果您为包含任何列出的关键字作为标识符的命令获得虚假解析器错误，您应该尝试引用标识符以查看问题是否消失。

在研究表之前，请注意关键字未保留的事实并不意味着未实现与该字相关的特性。 相反，关键词的存在并不表示特性存在。

Table 1. SQL Key Words

关键字	MPP数据库	PostgreSQL 9.4
ABORT	unreserved	unreserved
ABSOLUTE	unreserved	unreserved
ACCESS	unreserved	unreserved
ACTION	unreserved	unreserved
ACTIVE	unreserved	
ADD	unreserved	unreserved
ADMIN	unreserved	unreserved
AFTER	unreserved	unreserved
AGGREGAT E	unreserved	unreserved
ALL	reserved	reserved

关键字	MPP数据库	PostgreSQL 9.4
ALSO	unreserved	unreserved
ALTER	unreserved	unreserved
ALWAYS	unreserved	unreserved
ANALYSE	reserved	reserved
ANALYZE	reserved	reserved
AND	reserved	reserved
ANY	reserved	reserved
ARRAY	reserved	reserved
AS	reserved	reserved
ASC	reserved	reserved
ASSERTION	unreserved	unreserved
ASSIGNMENT	unreserved	unreserved
ASYMMETRIC	reserved	reserved
AT	unreserved	unreserved
ATTRIBUTE	unreserved	unreserved
AUTHORIZATION	reserved (可以是函数或类型名称)	reserved (可以是函数或类型名称)
BACKWARD	unreserved	unreserved
BEFORE	unreserved	unreserved
BEGIN	unreserved	unreserved
BETWEEN	unreserved (不能是函数或类型名称)	unreserved (不能是函数或类型名称)

关键字	MPP数据库	PostgreSQL 9.4
BIGINT	unreserved (不能是函数或类型名称)	unreserved (不能是函数或类型名称)
BINARY	reserved (可以是函数或类型名称)	reserved (可以是函数或类型名称)
BIT	unreserved (不能是函数或类型名称)	unreserved (不能是函数或类型名称)
BOOLEAN	unreserved (不能是函数或类型名称)	unreserved (不能是函数或类型名称)
BOTH	reserved	reserved
BY	unreserved	unreserved
CACHE	unreserved	unreserved
CALLED	unreserved	unreserved
CASCADE	unreserved	unreserved
CASCADED	unreserved	unreserved
CASE	reserved	reserved
CAST	reserved	reserved
CATALOG	unreserved	unreserved
CHAIN	unreserved	unreserved
CHAR	unreserved (不能是函数或类型名称)	unreserved (不能是函数或类型名称)
CHARACTER	unreserved (不能是函数或类型名称)	unreserved (不能是函数或类型名称)
CHARACTERISTICS	unreserved	unreserved
CHECK	reserved	reserved

关键字	MPP数据库	PostgreSQL 9.4
CHECKPOINT	unreserved	unreserved
CLASS	unreserved	unreserved
CLOSE	unreserved	unreserved
CLUSTER	unreserved	unreserved
COALESCE	unreserved (不能是函数或类型名称)	unreserved (不能是函数或类型名称)
COLLATE	reserved	reserved
COLLATION	reserved (可以是函数或类型名称)	reserved (可以是函数或类型名称)
COLUMN	reserved	reserved
COMMENT	unreserved	unreserved
COMMENTS	unreserved	unreserved
COMMIT	unreserved	unreserved
COMMITTED	unreserved	unreserved
CONCURRENCY	unreserved	
CONCURRENTLY	reserved (可以是函数或类型名称)	reserved (可以是函数或类型名称)
CONFIGURATION	unreserved	unreserved
CONNECTION	unreserved	unreserved

关键字	MPP数据库	PostgreSQL 9.4
CONSTRAINT	reserved	reserved
CONSTRAINTS	unreserved	unreserved
CONTAINS	unreserved	
CONTENT	unreserved	unreserved
CONTINUE	unreserved	unreserved
CONVERSION	unreserved	unreserved
COPY	unreserved	unreserved
COST	unreserved	unreserved
CPU_RATE_LIMIT	unreserved	
CPUSET	unreserved	
CREATE	reserved	reserved
CREATEEXTTABLE	unreserved	
CROSS	reserved (可以是函数或类型名称)	reserved (可以是函数或类型名称)
CSV	unreserved	unreserved
CUBE	unreserved (不能是函数或类型名称)	
CURRENT	unreserved	unreserved
CURRENT_CATALOG	reserved	reserved

关键字	MPP数据库	PostgreSQL 9.4
CURRENT_DATE	reserved	reserved
CURRENT_ROLE	reserved	reserved
CURRENT_SCHEMA	reserved (可以是函数或类型名称)	reserved (可以是函数或类型名称)
CURRENT_TIME	reserved	reserved
CURRENT_TIMESTAMP	reserved	reserved
CURRENT_USER	reserved	reserved
CURSOR	unreserved	unreserved
CYCLE	unreserved	unreserved
DATA	unreserved	unreserved
DATABASE	unreserved	unreserved
DAY	unreserved	unreserved
DEALLOCATE	unreserved	unreserved
DEC	unreserved (不能是函数或类型名称)	unreserved (不能是函数或类型名称)
DECIMAL	unreserved (不能是函数或类型名称)	unreserved (不能是函数或类型名称)
DECLARE	unreserved	unreserved
DECODE	reserved	

关键字	MPP数据库	PostgreSQL 9.4
DEFAULT	reserved	reserved
DEFAULTS	unreserved	unreserved
DEFERRABLE	reserved	reserved
DEFERRED	unreserved	unreserved
DEFINER	unreserved	unreserved
DELETE	unreserved	unreserved
DELIMITER	unreserved	unreserved
DELIMITERS	unreserved	unreserved
DENY	unreserved	
DESC	reserved	reserved
DICTIONARY	unreserved	unreserved
DISABLE	unreserved	unreserved
DISCARD	unreserved	unreserved
DISTINCT	reserved	reserved
DISTRIBUTED	reserved	
DO	reserved	reserved
DOCUMENT	unreserved	unreserved
DOMAIN	unreserved	unreserved
DOUBLE	unreserved	unreserved
DROP	unreserved	unreserved

关键字	MPP数据库	PostgreSQL 9.4
DXL	unreserved	
EACH	unreserved	unreserved
ELSE	reserved	reserved
ENABLE	unreserved	unreserved
ENCODING	unreserved	unreserved
ENCRYPTED	unreserved	unreserved
END	reserved	reserved
ENUM	unreserved	unreserved
ERRORS	unreserved	
ESCAPE	unreserved	unreserved
EVENT	unreserved	unreserved
EVERY	unreserved	
EXCEPT	reserved	reserved
EXCHANGE	unreserved	
EXCLUDE	reserved	unreserved
EXCLUDING	unreserved	unreserved
EXCLUSIVE	unreserved	unreserved
EXECUTE	unreserved	unreserved
EXISTS	unreserved (不能是函数或类型名称)	unreserved (不能是函数或类型名称)
EXPAND	unreserved	
EXPLAIN	unreserved	unreserved

关键字	MPP数据库	PostgreSQL 9.4
EXTENSION	unreserved	unreserved
EXTERNAL	unreserved	unreserved
EXTRACT	unreserved (不能是函数或类型名称)	unreserved (不能是函数或类型名称)
FALSE	reserved	reserved
FAMILY	unreserved	unreserved
FETCH	reserved	reserved
FIELDS	unreserved	
FILL	unreserved	
FILTER	unreserved	unreserved
FIRST	unreserved	unreserved
FLOAT	unreserved (不能是函数或类型名称)	unreserved (不能是函数或类型名称)
FOLLOWING	reserved	unreserved
FOR	reserved	reserved
FORCE	unreserved	unreserved
FOREIGN	reserved	reserved
FORMAT	unreserved	
FORWARD	unreserved	unreserved
FREEZE	reserved (可以是函数或类型名称)	reserved (可以是函数或类型名称)
FROM	reserved	reserved
FULL	reserved (可以是函数或类型名称)	reserved (可以是函数或类型名称)

关键字	MPP数据库	PostgreSQL 9.4
FULLSCAN	unreserved	
FUNCTION	unreserved	unreserved
FUNCTIONS	unreserved	unreserved
GLOBAL	unreserved	unreserved
GRANT	reserved	reserved
GRANTED	unreserved	unreserved
GREATEST	unreserved (不能是函数或类型名称)	unreserved (不能是函数或类型名称)
GROUP	reserved	reserved
GROUP_ID	unreserved (不能是函数或类型名称)	
GROUPING	unreserved (不能是函数或类型名称)	
HANDLER	unreserved	unreserved
HASH	unreserved	
HAVING	reserved	reserved
HEADER	unreserved	unreserved
HOLD	unreserved	unreserved
HOST	unreserved	
HOURL	unreserved	unreserved
IDENTITY	unreserved	unreserved
IF	unreserved	unreserved
IGNORE	unreserved	

关键字	MPP数据库	PostgreSQL 9.4
ILIKE	reserved (可以是函数或类型名称)	reserved (可以是函数或类型名称)
IMMEDIATE	unreserved	unreserved
IMMUTABLE	unreserved	unreserved
IMPLICIT	unreserved	unreserved
IN	reserved	reserved
INCLUDING	unreserved	unreserved
INCLUSIVE	unreserved	
INCREMENT T	unreserved	unreserved
INDEX	unreserved	unreserved
INDEXES	unreserved	unreserved
INHERIT	unreserved	unreserved
INHERITS	unreserved	unreserved
INITIALLY	reserved	reserved
INLINE	unreserved	unreserved
INNER	reserved (可以是函数或类型名称)	reserved (可以是函数或类型名称)
INOUT	unreserved (不能是函数或类型名称)	unreserved (不能是函数或类型名称)
INPUT	unreserved	unreserved
INSENSITIV E	unreserved	unreserved
INSERT	unreserved	unreserved

关键字	MPP数据库	PostgreSQL 9.4
INSTEAD	unreserved	unreserved
INT	unreserved (不能是函数或类型名称)	unreserved (不能是函数或类型名称)
INTEGER	unreserved (不能是函数或类型名称)	unreserved (不能是函数或类型名称)
INTERSECT	reserved	reserved
INTERVAL	unreserved (不能是函数或类型名称)	unreserved (不能是函数或类型名称)
INTO	reserved	reserved
INVOKER	unreserved	unreserved
IS	reserved (可以是函数或类型名称)	reserved (可以是函数或类型名称)
ISNULL	reserved (可以是函数或类型名称)	reserved (可以是函数或类型名称)
ISOLATION	unreserved	unreserved
JOIN	reserved (可以是函数或类型名称)	reserved (可以是函数或类型名称)
KEY	unreserved	unreserved
LABEL	unreserved	unreserved
LANGUAGE	unreserved	unreserved
LARGE	unreserved	unreserved
LAST	unreserved	unreserved
LATERAL	reserved	reserved
LC_COLLATE	unreserved	unreserved

关键字	MPP数据库	PostgreSQL 9.4
LC_CTYPE	unreserved	unreserved
LEADING	reserved	reserved
LEAKPROOF	unreserved	unreserved
LEAST	unreserved (不能是函数或类型名称)	unreserved (不能是函数或类型名称)
LEFT	reserved (可以是函数或类型名称)	reserved (可以是函数或类型名称)
LEVEL	unreserved	unreserved
LIKE	reserved (可以是函数或类型名称)	reserved (可以是函数或类型名称)
LIMIT	reserved	reserved
LIST	unreserved	
LISTEN	unreserved	unreserved
LOAD	unreserved	unreserved
LOCAL	unreserved	unreserved
LOCALTIME	reserved	reserved
LOCALTIME STAMP	reserved	reserved
LOCATION	unreserved	unreserved
LOCK	unreserved	unreserved
LOG	reserved (可以是函数或类型名称)	
MAPPING	unreserved	unreserved

关键字	MPP数据库	PostgreSQL 9.4
MASTER	unreserved	
MATCH	unreserved	unreserved
MATERIALIZED	unreserved	unreserved
MAXVALUE	unreserved	unreserved
MEDIAN	unreserved (不能是函数或类型名称)	
MEMORY_LIMIT	unreserved	
MEMORY_SHARED_QUOTA	unreserved	
MEMORY_SPILL_RATIO	unreserved	
MINUTE	unreserved	unreserved
MINVALUE	unreserved	unreserved
MISSING	unreserved	
MODE	unreserved	unreserved
MODIFIES	unreserved	
MONTH	unreserved	unreserved
MOVE	unreserved	unreserved
NAME	unreserved	unreserved
NAMES	unreserved	unreserved
NATIONAL	unreserved (不能是函数或类型名称)	unreserved (不能是函数或类型名称)

关键字	MPP数据库	PostgreSQL 9.4
NATURAL	reserved (可以是函数或类型名称)	reserved (可以是函数或类型名称)
NCHAR	unreserved (不能是函数或类型名称)	unreserved (不能是函数或类型名称)
NEWLINE	unreserved	
NEXT	unreserved	unreserved
NO	unreserved	unreserved
NOCREATE EXTTABLE	unreserved	
NONE	unreserved (不能是函数或类型名称)	unreserved (不能是函数或类型名称)
NOOVERCO MMIT	unreserved	
NOT	reserved	reserved
NOTHING	unreserved	unreserved
NOTIFY	unreserved	unreserved
NOTNULL	reserved (可以是函数或类型名称)	reserved (可以是函数或类型名称)
NOWAIT	unreserved	unreserved
NULL	reserved	reserved
NULLIF	unreserved (不能是函数或类型名称)	unreserved (不能是函数或类型名称)
NULLS	unreserved	unreserved
NUMERIC	unreserved (不能是函数或类型名称)	unreserved (不能是函数或类型名称)

关键字	MPP数据库	PostgreSQL 9.4
OBJECT	unreserved	unreserved
OF	unreserved	unreserved
OFF	unreserved	unreserved
OFFSET	reserved	reserved
OIDS	unreserved	unreserved
ON	reserved	reserved
ONLY	reserved	reserved
OPERATOR	unreserved	unreserved
OPTION	unreserved	unreserved
OPTIONS	unreserved	unreserved
OR	reserved	reserved
ORDER	reserved	reserved
ORDERED	unreserved	
ORDINALITY	unreserved	unreserved
OTHERS	unreserved	
OUT	unreserved (不能是函数或类型名称)	unreserved (不能是函数或类型名称)
OUTER	reserved (可以是函数或类型名称)	reserved (可以是函数或类型名称)
OVER	unreserved	unreserved
OVERCOMMIT	unreserved	

关键字	MPP数据库	PostgreSQL 9.4
OVERLAPS	reserved (可以是函数或类型名称)	reserved (可以是函数或类型名称)
OVERLAY	unreserved (不能是函数或类型名称)	unreserved (不能是函数或类型名称)
OWNED	unreserved	unreserved
OWNER	unreserved	unreserved
PARSER	unreserved	unreserved
PARTIAL	unreserved	unreserved
PARTITION	reserved	unreserved
PARTITION S	unreserved	
PASSING	unreserved	unreserved
PASSWORD	unreserved	unreserved
PERCENT	unreserved	
PLACING	reserved	reserved
PLANS	unreserved	unreserved
POSITION	unreserved (不能是函数或类型名称)	unreserved (不能是函数或类型名称)
PRECEDIN G	reserved	unreserved
PRECISION	unreserved (不能是函数或类型名称)	unreserved (不能是函数或类型名称)
PREPARE	unreserved	unreserved
PREPARED	unreserved	unreserved

关键字	MPP数据库	PostgreSQL 9.4
PRESERVE	unreserved	unreserved
PRIMARY	reserved	reserved
PRIOR	unreserved	unreserved
PRIVILEGE S	unreserved	unreserved
PROCEDUR AL	unreserved	unreserved
PROCEDUR E	unreserved	unreserved
PROGRAM	unreserved	unreserved
PROTOCOL	unreserved	
QUEUE	unreserved	
QUOTE	unreserved	unreserved
RANDOMLY	unreserved	
RANGE	unreserved	unreserved
READ	unreserved	unreserved
READABLE	unreserved	
READS	unreserved	
REAL	unreserved (不能是函数或类型名称)	unreserved (不能是函数或类型名称)
REASSIGN	unreserved	unreserved
RECHECK	unreserved	unreserved
RECURSIV E	unreserved	unreserved

关键字	MPP数据库	PostgreSQL 9.4
REF	unreserved	unreserved
REFERENC ES	reserved	reserved
REFRESH	unreserved	unreserved
REINDEX	unreserved	unreserved
REJECT	unreserved	
RELATIVE	unreserved	unreserved
RELEASE	unreserved	unreserved
RENAME	unreserved	unreserved
REPEATAB LE	unreserved	unreserved
REPLACE	unreserved	unreserved
REPLICA	unreserved	unreserved
REPLICATE D	unreserved	
RESET	unreserved	unreserved
RESOURCE	unreserved	
RESTART	unreserved	unreserved
RESTRICT	unreserved	unreserved
RETURNIN G	reserved	reserved
RETURNS	unreserved	unreserved
REVOKE	unreserved	unreserved

关键字	MPP数据库	PostgreSQL 9.4
RIGHT	reserved (可以是函数或类型名称)	reserved (可以是函数或类型名称)
ROLE	unreserved	unreserved
ROLLBACK	unreserved	unreserved
ROLLUP	unreserved (不能是函数或类型名称)	
ROOTPARTITION	unreserved	
ROW	unreserved (不能是函数或类型名称)	unreserved (不能是函数或类型名称)
ROWS	unreserved	unreserved
RULE	unreserved	unreserved
SAVEPOINT	unreserved	unreserved
SCATTER	reserved	
SCHEMA	unreserved	unreserved
SCROLL	unreserved	unreserved
SEARCH	unreserved	unreserved
SECOND	unreserved	unreserved
SECURITY	unreserved	unreserved
SEGMENT	unreserved	
SEGMENTS	unreserved	
SELECT	reserved	reserved
SEQUENCE	unreserved	unreserved

关键字	MPP数据库	PostgreSQL 9.4
SEQUENCE S	unreserved	unreserved
SERIALIZAB LE	unreserved	unreserved
SERVER	unreserved	unreserved
SESSION	unreserved	unreserved
SESSION_U SER	reserved	reserved
SET	unreserved	unreserved
SETOF	unreserved (不能是函数或类 型名称)	unreserved (不能是函数或类 型名称)
SETS	unreserved (不能是函数或类 型名称)	
SHARE	unreserved	unreserved
SHOW	unreserved	unreserved
SIMILAR	reserved (可以是函数或类型 名称)	reserved (可以是函数或类型 名称)
SIMPLE	unreserved	unreserved
SMALLINT	unreserved (不能是函数或类 型名称)	unreserved (不能是函数或类 型名称)
SNAPSHOT	unreserved	unreserved
SOME	reserved	reserved
SPLIT	unreserved	
SQL	unreserved	
STABLE	unreserved	unreserved

关键字	MPP数据库	PostgreSQL 9.4
STANDALO NE	unreserved	unreserved
START	unreserved	unreserved
STATEMEN T	unreserved	unreserved
STATISTICS	unreserved	unreserved
STDIN	unreserved	unreserved
STDOUT	unreserved	unreserved
STORAGE	unreserved	unreserved
STRICT	unreserved	unreserved
STRIP	unreserved	unreserved
SUBPARTIT ION	unreserved	
SUBSTRING	unreserved (不能是函数或类 型名称)	unreserved (不能是函数或类 型名称)
SYMMETRI C	reserved	reserved
SYSID	unreserved	unreserved
SYSTEM	unreserved	unreserved
TABLE	reserved	reserved
TABLES	unreserved	unreserved
TABLESPA CE	unreserved	unreserved
TEMP	unreserved	unreserved

关键字	MPP数据库	PostgreSQL 9.4
TEMPLATE	unreserved	unreserved
TEMPORARY	unreserved	unreserved
TEXT	unreserved	unreserved
THEN	reserved	reserved
THRESHOLD	unreserved	
TIES	unreserved	
TIME	unreserved (不能是函数或类型名称)	unreserved (不能是函数或类型名称)
TIMESTAMP	unreserved (不能是函数或类型名称)	unreserved (不能是函数或类型名称)
TO	reserved	reserved
TRAILING	reserved	reserved
TRANSACTION	unreserved	unreserved
TREAT	unreserved (不能是函数或类型名称)	unreserved (不能是函数或类型名称)
TRIGGER	unreserved	unreserved
TRIM	unreserved (不能是函数或类型名称)	unreserved (不能是函数或类型名称)
TRUE	reserved	reserved
TRUNCATE	unreserved	unreserved
TRUSTED	unreserved	unreserved
TYPE	unreserved	unreserved

关键字	MPP数据库	PostgreSQL 9.4
TYPES	unreserved	unreserved
UNBOUNDED	reserved	unreserved
UNCOMMITTED	unreserved	unreserved
UNENCRYPTED	unreserved	unreserved
UNION	reserved	reserved
UNIQUE	reserved	reserved
UNKNOWN	unreserved	unreserved
UNLISTEN	unreserved	unreserved
UNLOGGED	unreserved	unreserved
UNTIL	unreserved	unreserved
UPDATE	unreserved	unreserved
USER	reserved	reserved
USING	reserved	reserved
VACUUM	unreserved	unreserved
VALID	unreserved	unreserved
VALIDATE	unreserved	unreserved
VALIDATION	unreserved	
VALIDATOR	unreserved	unreserved
VALUE	unreserved	unreserved

关键字	MPP数据库	PostgreSQL 9.4
VALUES	unreserved (不能是函数或类型名称)	unreserved (不能是函数或类型名称)
VARCHAR	unreserved (不能是函数或类型名称)	unreserved (不能是函数或类型名称)
VARIADIC	reserved	reserved
VARYING	unreserved	unreserved
VERBOSE	reserved (可以是函数或类型名称)	reserved (可以是函数或类型名称)
VERSION	unreserved	unreserved
VIEW	unreserved	unreserved
VIEWS	unreserved	unreserved
VOLATILE	unreserved	unreserved
WEB	unreserved	
WHEN	reserved	reserved
WHERE	reserved	reserved
WHITESPACE	unreserved	unreserved
WINDOW	reserved	reserved
WITH	reserved	reserved
WITHIN	unreserved	unreserved
WITHOUT	unreserved	unreserved
WORK	unreserved	unreserved
WRAPPER	unreserved	unreserved
WRITABLE	unreserved	

关键字	MPP数据库	PostgreSQL 9.4
WRITE	unreserved	unreserved
XML	unreserved	unreserved
XMLATTRIBUTES	unreserved (不能是函数或类型名称)	unreserved (不能是函数或类型名称)
XMLCONCAT	unreserved (不能是函数或类型名称)	unreserved (不能是函数或类型名称)
XMLELEMENT	unreserved (不能是函数或类型名称)	unreserved (不能是函数或类型名称)
XML EXISTS	unreserved (不能是函数或类型名称)	unreserved (不能是函数或类型名称)
XMLFOREST	unreserved (不能是函数或类型名称)	unreserved (不能是函数或类型名称)
XMLPARSE	unreserved (不能是函数或类型名称)	unreserved (不能是函数或类型名称)
XMLPI	unreserved (不能是函数或类型名称)	unreserved (不能是函数或类型名称)
XMLROOT	unreserved (不能是函数或类型名称)	unreserved (不能是函数或类型名称)
XMLSERIALIZE	unreserved (不能是函数或类型名称)	unreserved (不能是函数或类型名称)
YEAR	unreserved	unreserved
YES	unreserved	unreserved
ZONE	unreserved	unreserved

Parent topic: [MPP数据库参考手册](#)

系统目录参考

该参考文献描述了MPP数据库系统目录表和视图。以gp_为前缀的系统表与MPP数据库的并行特性有关。以pg_为前缀的表是MPP数据库中支持的标准PostgreSQL系统目录表，或与MPP数据库性能有关，以提升PostgreSQL的数据仓库工作负载。请注意，MPP数据库的全局系统目录位于主实例上。

Warning: 不支持对MPP数据库系统目录表或视图的更改。如果更改目录表或视图，则必须重新初始化并还原集群。

- [系统表](#)
- [系统视图](#)
- [系统目录定义](#)

Parent topic: [MPP数据库参考手册](#)

系统表

- [gp_configuration_history](#)
 - [gp_distribution_policy](#)
 - [gp_fastsequence](#)
 - [gp_global_sequence](#)
 - [gp_id](#)
 - [gp_stat_replication](#)
-

- [gp_segment_configuration](#)
 - [gp_version_at_initdb](#)
 - [gpexpand.status](#)
 - [gpexpand.status_detail](#)
 - [pg_aggregate](#)
 - [pg_am](#)
 - [pg_amop](#)
 - [pg_amproc](#)
 - [pg_appendonly](#)
 - pg_appendonly_alter_column (not supported)
 - [pg_attrdef](#)
 - [pg_attribute](#)
 - [pg_auth_members](#)
 - [pg_authid](#)
 - pg_autovacuum (not supported)
 - [pg_cast](#)
 - [pg_class](#)
 - [pg_constraint](#)
 - [pg_conversion](#)
 - [pg_database](#)
 - [pg_db_role_setting](#)
 - [pg_depend](#)
 - [pg_description](#)
 - [pg_exttable](#)
 - [pg_foreign_data_wrapper](#)
-

- [pg_foreign_server](#)
 - [pg_foreign_table](#)
 - [pg_index](#)
 - [pg_inherits](#)
 - [pg_language](#)
 - [pg_largeobject](#)
 - [pg_listener](#)
 - [pg_namespace](#)
 - [pg_opclass](#)
 - [pg_operator](#)
 - [pg_opfamily](#)
 - [pg_partition](#)
 - [pg_partition_rule](#)
 - [pg_pltemplate](#)
 - [pg_proc](#)
 - [pg_resgroup](#)
 - [pg_resgroupcapability](#)
 - [pg_resourcetype](#)
 - [pg_resqueue](#)
 - [pg_resqueuecapability](#)
 - [pg_rewrite](#)
 - [pg_shdepend](#)
 - [pg_shdescription](#)
 - [pg_stat_last_operation](#)
 - [pg_stat_last_shoperation](#)
-

- [pg_stat_replication](#)
- [pg_statistic](#)
- [pg_tablespace](#)
- [pg_trigger](#)
- [pg_type](#)
- [pg_user_mapping](#)

Parent topic: [系统目录参考](#)

系统视图

MPP数据库提供了如下在PostgreSQL中不存在的系统视图。

- [gp_distributed_log](#)
 - [gp_distributed_xacts](#)
 - [gp_pgdatabase](#)
 - [gp_resgroup_config](#)
 - [gp_resgroup_status](#)
 - [gp_resgroup_status_per_host](#)
 - [gp_resgroup_status_per_segment](#)
 - [gp_resqueue_status](#)
 - [gp_transaction_log](#)
 - [gpexpand.expansion_progress](#)
-

- [pg_max_external_files](#)
- [pg_partition_columns](#)
- [pg_partition_templates](#)
- [pg_partitions](#)
- [pg_resqueue_attributes](#)
- [pg_resqueue_status](#) (弃用。使用[gp_toolkit.gp_resqueue_status.](#))
- [pg_stat_activity](#)
- [pg_stat_replication](#)
- [pg_stat_resqueues](#)
- [session_level_memory_consumption](#)(见*MPP数据库管理员手册*中的“查看会话内存使用信息”。)

有关PostgreSQL和MPP数据库支持的标准系统视图的更多信息，请参阅PostgreSQL文档的以下部分：

- [系统视图](#)
- [统计收集器视图](#)
- [信息模式](#)

Parent topic: [系统目录参考](#)

系统目录定义

系统目录表和视图按字母顺序定义。

- [foreign_data_wrapper_options](#)
 - [foreign_data_wrappers](#)
 - [foreign_server_options](#)
 - [foreign_servers](#)
 - [foreign_table_options](#)
 - [foreign_tables](#)
 - [gp_configuration_history](#)
 - [gp_distributed_log](#)
 - [gp_distributed_xacts](#)
 - [gp_distribution_policy](#)
 - [gpexpand.expansion_progress](#)
 - [gpexpand.status](#)
 - [gpexpand.status_detail](#)
 - [gp_fastsequence](#)
 - [gp_id](#)
 - [gp_pgdatabase](#)
 - [gp_resgroup_config](#)
 - [gp_resgroup_status](#)
 - [gp_resgroup_status_per_host](#)
 - [gp_resgroup_status_per_segment](#)
 - [gp_resqueue_status](#)
 - [gp_stat_replication](#)
 - [gp_segment_configuration](#)
 - [gp_transaction_log](#)
-

- [gp_version_at_initdb](#)
 - [pg_aggregate](#)
 - [pg_am](#)
 - [pg_amop](#)
 - [pg_amproc](#)
 - [pg_appendonly](#)
 - [pg_attrdef](#)
 - [pg_attribute](#)
 - [pg_attribute_encoding](#)
 - [pg_auth_members](#)
 - [pg_authid](#)
 - [pg_available_extension_versions](#)
 - [pg_available_extensions](#)
 - [pg_cast](#)
 - [pg_class](#)
 - [pg_compression](#)
 - [pg_constraint](#)
 - [pg_conversion](#)
 - [pg_database](#)
 - [pg_db_role_setting](#)
 - [pg_depend](#)
 - [pg_description](#)
 - [pg_enum](#)
 - [pg_extension](#)
-

- [pg_exttable](#)
 - [pg_foreign_data_wrapper](#)
 - [pg_foreign_server](#)
 - [pg_foreign_table](#)
 - [pg_index](#)
 - [pg_inherits](#)
 - [pg_language](#)
 - [pg_largeobject](#)
 - [pg_listener](#)
 - [pg_locks](#)
 - [pg_max_external_files](#)
 - [pg_namespace](#)
 - [pg_opclass](#)
 - [pg_operator](#)
 - [pg_opfamily](#)
 - [pg_partition](#)
 - [pg_partition_columns](#)
 - [pg_partition_encoding](#)
 - [pg_partition_rule](#)
 - [pg_partition_templates](#)
 - [pg_partitions](#)
 - [pg_pltemplate](#)
 - [pg_proc](#)
 - [pg_resgroup](#)
-

- [pg_resgroupcapability](#)
 - [pg_resourcetype](#)
 - [pg_resqueue](#)
 - [pg_resqueue_attributes](#)
 - [pg_resqueuecapability](#)
 - [pg_rewrite](#)
 - [pg_roles](#)
 - [pg_shdepend](#)
 - [pg_shdescription](#)
 - [pg_stat_activity](#)
 - [pg_stat_last_operation](#)
 - [pg_stat_last_shoperation](#)
 - [pg_stat_operations](#)
 - [pg_stat_partition_operations](#)
 - [pg_stat_replication](#)
 - [pg_stat_statements](#)
 - [pg_statistic](#)
 - [pg_stat_resqueues](#)
 - [pg_tablespace](#)
 - [pg_trigger](#)
 - [pg_type](#)
 - [pg_type_encoding](#)
 - [pg_user_mapping](#)
 - [pg_user_mappings](#)
-

- [user_mapping_options](#)
- [user_mappings](#)

Parent topic: [系统目录参考](#)

foreign_data_wrapper_options

foreign_data_wrapper_options视图包含为当前数据库中的外部数据包装器定义的所有选项。 MPP数据库仅显示当前用户可以访问的外部数据包装器（通过作为所有者或具有某些权限）。

Table 1. foreign_data_wrapper_options

列	类型	参考	描述
foreign_data_wrapp er_catalog	sql_identif ier		定义外部数据包装器的数据库的名称（始终是当前数据库）。
foreign_data_wrapp er_name	sql_identif ier		外部数据包装器的名称。
option_name	sql_identif ier		选项的名称。
option_value	character _data		选项的值。

Parent topic: [系统目录定义](#)

foreign_data_wrappers

foreign_data_wrappers视图包含当前数据库中定义的所有外部数据包装器。MPP数据库仅显示当前用户可以访问的外部数据包装器（通过作为所有者或具有某些权限）。

Table 1. foreign_data_wrappers

列	类型	参考	描述
foreign_data_wrapp er_catalog	sql_identif ier		定义外部数据包装器的数据库的名称（始终是当前数据库）。
foreign_data_wrapp er_name	sql_identif ier		外部数据包装器的名称。
authorization_identif ier	sql_identif ier		外部服务器所有者的名称。
library_name	character _data		实现此外部数据包装器的库的文件名。
foreign_data_wrapp er_language	character _data		用于实现外部数据包装器的语言。

Parent topic: [系统目录定义](#)

foreign_server_options

foreign_servers视图包含当前数据库中定义的所有外部服务器。 MPP数据库仅显示当前用户可以访问的外部服务器（通过作为所有者或具有某些权限）。 foreign_server_options视图包含为当前数据库中的外部服务器定义的所有选项。 MPP数据库仅显示当前用户可以访问的外部服务器（通过作为所有者或具有某些权限）。

Table 1. foreign_server_options

列	类型	参考	描述
foreign_server_catalog	sql_identifier		定义外部服务器的数据库的名称（始终是当前数据库）。
foreign_server_name	sql_identifier		外部服务器的名称。
option_name	sql_identifier		选项的名称。
option_value	character_data		选项的值。

Parent topic: [系统目录定义](#)

foreign_servers

foreign_servers视图包含当前数据库中定义的所有外部服务器。 MPP数据库仅显示当前用户可以访问的外部服务器（通过作为所有者或具有某些权限）。

Table 1. foreign_servers

列	类型	参考	描述
foreign_server_catalog	sql_identifier		定义外部服务器的数据库的名称（始终是当前数据库）。
foreign_server_name	sql_identifier		外部服务器的名称。
foreign_data_wrapper_catalog	sql_identifier		定义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	character_data		外部服务器类型信息，如果在创建时指定。
foreign_server_version	character_data		外部服务器版本信息，如果在创建时指定。
authorization_identifier	sql_identifier		外部服务器所有者的名称。

Parent topic: [系统目录定义](#)

foreign_table_options

foreign_table_options视图包含为当前数据库中的外部表定义的所有选项。
MPP数据库仅显示当前用户可以访问的外部表（通过作为所有者或具有某些权限）。

Table 1. foreign_table_options

列	类型	参考	描述
foreign_table_catalog	sql_identifier		定义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schema的名称。
foreign_table_name	sql_identifier		外部表的名称。
option_name	sql_identifier		选项的名称。
option_value	character_data		选项的值。

Parent topic: [系统目录定义](#)

foreign_tables

foreign_tables视图包含当前数据库中定义的所有外部表。 MPP数据库仅显示当前用户可以访问的外部表（通过作为所有者或具有某些权限）。

Table 1. foreign_tables

列	类型	参考	描述
foreign_table_catalog	sql_identifier		定义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schema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（始终是当前数据库）。
foreign_server_name	sql_identifier		外部服务器的名称。

Parent topic: [系统目录定义](#)

gp_configuration_history

gp_configuration_history表包含有关故障检测和恢复操作相关的系统更改信息。fts_probe进程将数据记录到此表，和相关的gpcheck, gprecoverseg和gpinitssystem之类的管理工具一样。例如，当用户向系统添加新的Segment和镜像Segment时，这些事件会被记录到gp_configuration_history。

该表仅在Master上有数据。该表在pg_global表空间中定义, 这意味着它在系统中的所有数据库之间全局共享。

Table 1. pg_catalog.gp_configuration_history

列	类型	参考	描述
time	timestamp with time zone		记录事件的时间戳。
dbid	smallint	gp_segment_configuration.dbid	系统分配的ID。Segment（或者Master）实例的唯一标识符。
desc	text		时间的文本描述。

有关gpcheck、 gprecoverseg和gpinitssystem的信息，请参阅MPP数据库工具手册。

Parent topic: [系统目录定义](#)

gp_distributed_log

gp_distributed_log视图包含有关分布式事务及其关联的本地事务的状态信息。分布式事务是涉及修改Segment实例上数据的事务。 MPP的分布式事务管理器确保了这些Segment保持同步。 此视图允许用户查看分布式事务的状态。

Table 1. pg_catalog.gp_distributed_log

列	类型	参考	描述

列	类型	参考	描述
segment_id	smallint	gp_segment_configuration.content	如果是Segment，则是其内容ID。Master总是为-1（无内容）。
dbid	small_int	gp_segment_configuration.dbid	Segment实例的唯一ID。
distributed_xid	xid		全局事务ID。
distributed_id	text		分布式事务的系统分配ID。
status	text		分布式事务的状态（提交或者中止）。
local_transaction	xid		本地事务ID。

Parent topic: [系统目录定义](#)

gp_distributed_xacts

gp_distributed_xacts视图包含有关MPP数据库分布式事务的信息。 分布式事务是涉及修改Segment实例上数据的事务。 MPP的分布式事务管理器确保了这些Segment保持同步。 此视图允许用户查看当前活动的会话及其关联的分布式事务。

Table 1. pg_catalog.gp_distributed_xacts

列	类型	参考	描述

列	类型	参考	描述
distributed_xid	xid		分布式事务在MPP数据库阵列中使用的事务ID。
distributed_id	text		分布式事务标识符。它有2个部分 - 一个唯一的时间戳和分布式事务编号。
state	text		本次会话对于分布式事务的当前状态。
gp_session_id	int		与此事务关联的MPP数据库会话的ID编号。
xmin_distributed_snapshot	xid		在此事务开始时，所有打开事务中发现的最小分布式事务号。它用于MVCC分布式快照目的。

Parent topic: [系统目录定义](#)

gp_distribution_policy

gp_distribution_policy表包含有关MPP数据库表及其分布表数据的策略的信息。该表仅在Master上填充。该表不是全局共享的，这意味着每个数据库都有自己的副本。

Table 1. pg_catalog.gp_distribution_policy

列	类型	参考	描述
localoid	oid	pg_class.oid	表对象标识符（OID）。
policytype	char		表分布策略： • p - 分区策略。表数据在segment实例中分布。 • r - 复制策略。表数据在每个segment实例中冗余分布。
numsegments	integer		有表数据分布的节点数量。
distkey	int2vector	pg_attribute.attnum	分布列的列编号。
distclass	oidvector	pg_opclass.oid	分布列的运算符类标识符。

Parent topic: [系统目录定义](#)

gpexpand.expansion_progress

gpexpand.expansion_progress视图包含有关系统扩容操作状态的信息。该表提供了表重新分布速度和完成扩容的时间的估算。

扩容涉及的特定表的状态存储在[gpexpand.status_detail](#)中。

Table 1. gpexpand.expansion_progress

列	类型	参考	描述
name	text		提供的数据域的名称，包括： 剩余字节数 完成字节数 扩展速度估计 估计完成时间 已扩展的表 剩余的表
value	text		进度数据的值。例如： 扩展速度估计 - 9.756 67095996092 MB/s

Parent topic: [系统目录定义](#)

gpexpand.status

gpexpand.status表包含有关系统扩容操作状态的信息。扩容中涉及的特定表的状态存储在[gpexpand.status_detail](#)。

在正常的扩容操作中，不需要修改存储在该表中的数据。

Table 1. gpexpand.status

列	类型	参考	描述
status	text		跟踪扩展操作的状态。 有效值为： SETUP SETUP DONE EXPANSION STARTED EXPANSION STOPPED COMPLETED
updated	timestamp without time zone		最后状态变化的时间戳。

Parent topic: [系统目录定义](#)

gpexpand.status_detail

gpexpand.status_detail表包含了有关系统扩容操作所涉及的表的状态的信息。用户可以查询此表以确定正在扩容的表的状态，或查看已完成表的开始和结束时间。

此表还存储表的相关信息，如OID、磁盘尺寸。扩容的整体状态信息存储在[gpexpand.status](#)中。

In a normal expansion operation it is not necessary to modify the data stored in this table.

Table 1. gpexpand.status_detail

列	类型	参考	描述
dbname	text		表所属数据库的名称。
fq_name	text		表的完全限定名称。
table_oid	oid		表的OID.
root_partition_name	text		对于分区表来说，根分区的名称。否则，None。
rank	int		等级决定表被扩容的顺序。扩容工具将在rank上排序，并首先扩容排名最低的表。
external_writable	boolean		标识表是否是外部可写表。（外部可写表需要不同的语法来扩容）。

列	类型	参考	描述
status	text		此表的扩容状态。有效值为: NOT STARTED IN PROGRESS FINISHED NO LONGER EXISTS
expansion_started	timestamp without time zone		此表扩容开始的时间戳。此字段仅在表成功扩容后填充。
expansion_finished	timestamp without time zone		此表扩容完成的时间戳。
source_bytes			与源表相关的磁盘空间尺寸。由于堆表中的表膨胀和扩容后不同的Segment数量，最终的字节数是否与源字节相同是不可预测的。跟踪此信息有助于提供进度度量，以帮助进行端到端扩容操作的持续时间估计。

Parent topic: [系统目录定义](#)

gp_fastsequence

gp_fastsequence表包含有关追加优化表和列存表的信息。last_sequence值表示该表当前使用的最大行号。

Table 1. pg_catalog.gp_fastsequence

列	类型	参考	描述
objid	oid	pg_class.oid	用于跟踪追加优化文件段的pg_aoseg.pg_aocsseg_*表的对象ID。
objmod	bigint		对象修饰符。
last_sequence	bigint		对象使用的最后一个序列号。

Parent topic: [系统目录定义](#)

gp_id

gp_id系统目录表标识了MPP数据库系统名称和系统的Segment数量。它还具有表所在的特定数据库实例（Segment或Master）的本地值。该表在pg_global表空间中定义，这意味着它在系统中的所有数据库之间全局共享。

Table 1. pg_catalog.gp_id

列	类型	参考	描述
gpname	name		这个MPP数据库系统的名称。
numsegments	integer		MPP数据库系统中的Segment数量。
dbid	integer		此Segment（或Master）实例的唯一标识符。
content	integer		该Segment实例上的这部份数据的ID。主Segment及其镜像Segment将具有相同的内容ID。 对于Segment，值为0-<i>N-1</i>，其中<i>N</i>是MPP数据库中的Segment数量。 对于Master,其值为-1。

Parent topic: [系统目录定义](#)

gp_pgdatabase

gp_pgdatabase视图显示MPP中Segment实例的状态信息，以及它们是作为镜

像Segment还是主Segment。 MPP的故障检测和恢复工具在内部使用此视图来确定失效的Segment。

Table 1. pg_catalog.gp_pgdatabase

列	类型	参考	描述
dbid	smallint	gp_segment_configuration.dbid	系统分配的ID。Segment（或者Master）实例的唯一标识符。
isprimary	boolean	gp_segment_configuration.role	该实例是否处于活动状态。它目前是否作为主Segment（还是镜像Segment）。
content	smallint	gp_segment_configuration.content	实例上部分数据的ID。主Segment实例及其镜像将具有相同的内容ID。 对于Segment，值为0-<i>N</i>-1, 其中<i>N</i>是MPP数据库中的Segment数量。 对于Master，值为-1。
definedprimary	boolean	gp_segment_configuration.preferred_role	在初始化系统时，该实例是否被定义为主Segment（而不是镜像Segment）。

Parent topic: [系统目录定义](#)

gp_resgroup_config

gp_toolkit.gp_resgroup_config视图允许管理员查看资源组的当前CPU，内存和并发限制。该视图还显示建议的限制设置。当限制被更改时，建议的限制将不同于当前限制，但新值不能立即应用。

Note: 仅当基于资源组的资源管理处于活动状态时，gp_resgroup_config视图才有效。

Table 1. gp_toolkit.gp_resgroup_config

列	类型	参考	描述
groupid	oid	pg_resgroup.oid	资源组的OID。
groupname	name	pg_resgroup.rsgname	资源组的名称。
concurrency	text	pg_resgroupcapability.value for pg_resgroupcapability.reslimittype = 1	为资源组指定的并发（CONCURRENCY）值。
proposed_concurrency	text	pg_resgroupcapability.proposed for pg_resgroupcapability.reslimittype = 1	资源组的挂起并发值。

列	类型	参考	描述
cpu_rate_limit	text	pg_resgroupcapability.value for pg_resgroupcapability.reslimittype = 2	为资源组指定的CPU限制（CPU_RATE_LIMIT）值，或-1。
memory_limit	text	pg_resgroupcapability.value for pg_resgroupcapability.reslimittype = 3	为资源组指定的内存限制（MEMORY_LIMIT）值。
proposed_memory_limit	text	pg_resgroupcapability.proposed for pg_resgroupcapability.reslimittype = 3	资源组的挂起内存限制值。
memory_shared_quota	text	pg_resgroupcapability.value for pg_resgroupcapability.reslimittype = 4	为资源组指定的共享内存配额（MEMORY_SHARED_QUOTA）值。
proposed_memory_shared_quota	text	pg_resgroupcapability.proposed for pg_resgroupcapability.reslimittype = 4	资源组的挂起共享内存配额值。
memory_spill_ratio	text	pg_resgroupcapability.value for pg_resgroupcapability.reslimittype = 5	为资源组指定的内存溢出率（MEMORY_SPILL_RATIO）值。
proposed_memory_spill_ratio	text	pg_resgroupcapability.proposed for pg_resgroupcapability.reslimittype >= 5	资源组的待处理内存溢出比率值。

列	类型	参考	描述
memory_auditor	text	pg_resgroupcapability.value for pg_resgroupcapability.reslimittype = 6	用于资源组的内存审计器。
cpuset	text	pg_resgroupcapability.value for pg_resgroupcapability.reslimittype = 7	为资源组保留的CPU核心，或-1。

Parent topic: [系统目录定义](#)

gp_resgroup_status

gp_toolkit.gp_resgroup_status视图允许管理员查看资源组的状态和活动。它显示了等待运行的查询数以及系统中每个资源组当前有多少查询处于活动状态。该视图还显示资源组的当前内存和CPU使用情况。

Note: 仅当基于资源组的资源管理处于活动状态时，gp_resgroup_status视图才有效。

Table 1. gp_toolkit.gp_resgroup_status

列	类型	参考	描述
rsgname	name	pg_resgroup.rsgname	资源组的名称。
groupid	oid	pg_resgroup.oid	资源组的ID。

列	类型	参考	描述
num_running	integer		当前在资源组中执行的事务数。
num_queueing	integer		资源组的当前排队事务数。
num_queued	integer		自MPP数据库集群上次启动以来资源组的排队事务总数，不包括num_queueing。
num_executed	integer		自MPP数据库集群上次启动以来资源组中执行的事务总数，不包括num_running。
total_queue_duration	interval		自上次启动MPP数据库集群以来，任何事务排队的总时间。
cpu_usage	json		每个MPP数据库网段主机上资源组的实时CPU使用情况。
memory_usage	json		每个MPP数据库segment主机上资源组的实时内存使用情况。

cpu_usage字段是JSON格式的key : value字符串，用于为每个资源组标识每个segment的CPU使用率百分比。key是segment ID，value是segment主机上资源组的CPU使用率百分比。在segment主机上运行的所有segment的CPU总使用率不应超过gp_resource_group_cpu_limit。示例cpu_usage列输出：

```
{"-1":0.01, "0":0.31, "1":0.31}
```

memory_usage字段也是JSON格式的key : value字符串。字符串内容因资源组的类型而异。对于分配给角色的每个资源组（默认内存审核器vmtracker），此字符串标识每个segment上已使用，可用，已授予和建议的固定和共享内存配额分配。key是segment ID，value是以MB为单位显示的内存值。下面的示例显示分配给角色的资源组的单个segment的memory_usage列输出：

```
"0":{"used":0, "available":76, "quota_used":-1, "quota_available":60, "quota_granted":60, "quota_reserved":0}
```

对于分配给外部组件的每个资源组，memory_usage JSON格式的字符串标识了每个segment上使用的内存和内存限制。以下示例显示单个segment的外部组件资源组的memory_usage列输出：

```
"1":{"used":11, "limit_granted":15}
```

Parent topic: [系统目录定义](#)

gp_resgroup_status_per_host

gp_toolkit.gp_resgroup_status_per_host视图允许管理员基于每个主机查看每个资源组的当前内存和CPU使用率和分配。

内存量以MB为单位指定。

Note: 仅当基于资源组的资源管理处于活动状态时，gp_resgroup_status_per_host视图才有效。

Table 1. gp_toolkit.gp_resgroup_status_per_host

列	类型	参考	描述
rsgname	name	pg_resgroup.rsgname	资源组的名称。
groupid	oid	pg_resgroup.oid	资源组的ID。
hostname	text	gp_segment_configuration.hostname	segment主机的主机名。
cpu	numeric		主机上资源组的实时CPU使用情况。
memory_used	integer		主机上资源组的实时内存使用情况。此总数包括资源组固定和共享内存。它还包括资源组使用的全局共享内存。
memory_available	integer		主机上可用的资源组的未使用的固定和共享内存。此总计不包括可用资源组全局共享内存。
memory_quota_used	integer		主机上资源组的实时固定内存使用情况。
memory_quota_available	integer		主机上资源组可用的固定内存。
memory_quota_proposed	integer		MPP数据库分配给主机上资源组的固定内存总量。

列	类型	参考	描述
memory_shared _used	integer		该组共享主机上资源组使用的内存。 如果资源组使用任何全局共享内存，则此数量也包含在总计中。
memory_shared _available	integer		主机上资源组可用的组共享内存量。 资源组全局共享内存不包含在此总计中。
memory_shared _granted	integer		MPP数据库分配给主机上资源组的组共享内存部分。 资源组全局共享内存不包含在此值中。
memory_shared _proposed	integer		主机上资源组请求的组共享内存总量。

Parent topic: [系统目录定义](#)

gp_resgroup_status_per_segment

gp_toolkit.gp_resgroup_status_per_segment视图允许管理员基于每个主机和每个segment查看每个资源组的当前内存和CPU使用率和分配。

内存量以MB为单位指定。

Note: 仅当基于资源组的资源管理处于活动状态时，gp_resgroup_status_per_segment视图才有效。

Table 1. gp_toolkit.gp_resgroup_status_per_segment

列	类型	参考	描述
rsgname	name	pg_resgroup.rsgname	资源组的名称。
groupid	oid	pg_resgroup.oid	资源组的ID。
hostname	text	gp_segment_configuration.hostname	segment主机的主机名。
segment_id	smallint	gp_segment_configuration.content	segment主机上的segment实例的内容ID。
cpu	numeric		主机上的segment实例的资源组的实时CPU使用情况。
memory_used	integer		主机上的segment实例的资源组的实时内存使用情况。此总数包括资源组固定和共享内存。它还包括资源组使用的全局共享内存。
memory_available	integer		主机上的segment实例的资源组的未使用的固定和共享内存。
memory_quota_used	integer		主机上的segment实例的资源组的实时固定内存使用情况。

列	类型	参考	描述
memory_quota_available	integer		主机上的segment实例的资源组可用的固定内存。
memory_quota_proposed	integer		MPP数据库分配给主机上的segment的资源组的固定内存总量。对于主机上的所有资源组和segment实例组合，此值将相同。
memory_shared_used	integer		该组共享资源组用于主机上的segment实例的内存。
memory_shared_available	integer		主机上的segment实例可用的组共享内存量。资源组全局共享内存不包含在此总计中。
memory_shared_granted	integer		组共享内存的部分，MPP数据库已分配给主机上的segment实例的资源组。资源组全局共享内存不包含在此值中。
memory_shared_proposed	integer		主机上资源组请求的组共享内存总量。对于主机上的所有资源组和segment实例组合，此值将相同。

Parent topic: [系统目录定义](#)

gp_resqueue_status

gp_toolkit.gp_resqueue_status视图允许管理员查看资源队列的状态和活动。它显示了有多少查询正在等待运行以及系统中当前有多少查询来自特定资源队列。

Note: The gp_resqueue_status view is valid only when resource queue-based resource management is active.

Table 1. gp_toolkit.gp_resqueue_status

列	类型	参考	描述
queueid	oid	gp_toolkit.gp_resqueue_queueid	资源队列的ID。
rsqname	name	gp_toolkit.gp_resqueue_rsqname	资源队列的名称。
rsqcountlimit	real	gp_toolkit.gp_resqueue_rsqcountlimit	资源队列的活动查询阈值。 值-1表示没有限制。
rsqcountvalue	real	gp_toolkit.gp_resqueue_rsqcountvalue	当前在资源队列中使用的活动查询槽的数量。
rsqcostlimit	real	gp_toolkit.gp_resqueue_rsqcostlimit	资源队列的查询开销阈值。 值-1表示没有限制。

列	类型	参考	描述
rsqcostvalue	real	gp_toolkit.gp_resqueue_rsqcostvalue	当前在资源队列中的所有语句的总成本。
rsqmemorylimit	real	gp_toolkit.gp_resqueue_rsqmemorylimit	资源队列的内存限制。
rsqmemoryvalue	real	gp_toolkit.gp_resqueue_rsqmemoryvalue	当前在资源队列中的所有语句使用的总内存。
rsqwaiters	integer	gp_toolkit.gp_resqueue_rsqwaiter	当前在资源队列中等待的语句数。
rsqholders	integer	gp_toolkit.gp_resqueue_rsqholders	当前在此资源队列中在系统上运行的语句数。

Parent topic: [系统目录定义](#)

gp_stat_replication

gp_stat_replication视图包含walsender进程的复制统计信息，该进程在启用master或segment镜像时用于MPP数据库预写日志记录（WAL）复制。

Table 1. gp_catalog.gp_stat_replication

列	类型	参考	描述

列	类型	参考	描述
gp_segment_id	integer		segment（或master）实例的唯一标识符。
pid	integer		walsender后端进程的进程ID。
usesysid	oid		运行walsender后端进程的用户系统标识。
username	name		运行walsender后端进程的用户名。
application_name	text		客户端应用名称。
client_addr	inet		客户端IP地址。
client_hostname	text		客户端主机名。
client_port	integer		客户端端口号。
backend_start	timestamp		操作开始时间戳。
state	text		walsender state. The value can be: startup backup catchup streaming
sent_location	text		walsender xlog record sent location.

列	类型	参考	描述
write_location	text		walreceiver xlog record write location.
flush_location	text		walreceiver xlog record flush location.
replay_location	text		Master standby or segment mirror xlog record replay location.
sync_priority	integer		Priority. The value is 1.
sync_state	text		walsendersynchronization state. The value is sync.
sync_error	text		walsender synchronization error. none if no error.

Parent topic: [系统目录定义](#)

gp_segment_configuration

gp_segment_configuration表包含有关Mirror和Segment实例配置的信息。

Table 1. pg_catalog.gp_segment_configuration

列	类型	参考	描述
dbid	smallint		Segment（或Master）实例的唯一标识符。
content	smallint		Segment实例的内容标识符。主Segment实例及其镜像将始终具有相同的内容标识符。 对于Segment，值为0到N-1，其中N是系统中主Segment的数量。 对于master，值始终为-1。
role	char		Segment当前正在运行的角色。值为p（主）或m（镜像）。
preferred_role	char		最初在初始化时分配Segment的角色。值为p（主）或m（镜像）。
mode	char		Segment实例及其镜像副本的同步状态。值为s（同步）或n（未同步）。

列	类型	参考	描述
status	char		Segment实例的故障状态。 值是u（存活）或d（宕机）。
port	integer		数据库监听的TCP端口。 T
hostname	text		Segment主机的主机名。
address	text		用于访问Segment主机上的特定Segment实例的主机名。 此值可能与未配置每个接口主机名的系统上的主机名相同。
datadir	text		Segment实例数据目录。

Parent topic: [系统目录定义](#)

gp_transaction_log

gp_transaction_log视图包含特定Segment的本地事务的状态信息。 此视图允许用户查看本地事务的状态。

Table 1. pg_catalog.gp_transaction_log

列	类型	参考	描述
segment_id	smallint	gp_segment_configuration.content	如果是Segment，则是内容ID。Master总为-1 (没有内容)。
dbid	smallint	gp_segment_configuration.dbid	Segment实例的唯一ID。
transaction	xid		本地事务的ID。
status	text		本地事务的状态（提交或者中止）。

Parent topic: [系统目录定义](#)

gp_version_at_initdb

gp_version_at_initdb表在MPP数据库系统的Master和每一个Segment上被填充。它标识着系统初始化时使用的MPP数据库版本。这个表被定义在pg_global表空间中，意味着它在系统中的所有数据库中全局共享。

Table 1. pg_catalog.gp_version

列	类型	参考	描述
schemaversion	integer		Schema版本号。
productversion	text		产品版本号。

Parent topic: [系统目录定义](#)

pg_aggregate

pg_aggregate存储关于聚集函数的信息。聚集函数是对一个值集合（通常是来自于匹配某个查询条件的每个行的一个列值）进行操作并且返回从这些值计算出的一个值的函数。典型的聚集函数是sum、count和max。pg_aggregate里的每个项都是一个pg_proc项的扩展。pg_proc的项记载该聚集的名字、输入输出的数据类型以及其他和普通函数类似的信息。

Table 1. pg_catalog.pg_aggregate

列	类型	参考	描述
aggfnoid	regproc	pg_proc.oid	聚集函数的OID
aggtransfn	regproc	pg_proc.oid	转移函数的OID
aggcombinefn	regproc		预备函数的OID（如果没有就为0）
aggfinalfn	regproc	pg_proc.oid	最终函数的OID（如果没有就为0）
agginitval	text		转移状态的初始值。这是一个文本域，它包含初始值的外部字符串表现形式。如果这个域为NULL，则转移状态从NULL开始

列	类型	参考	描述
aggordered	Boolean		如果为true，则聚集定义为ORDERED。
aggstoptop	oid	pg_operator.oid	相关的排序操作符的OID（如果没有则为零）
aggtranstype	oid	pg_type.oid	聚集函数的内部转移（状态）数据的数据类型
aggtransspace	int4	pg_type.int4	估计的状态数据大小（默认估计为0）

Parent topic: [系统目录定义](#)

pg_am

pg_am表存储有关索引访问方法的信息。 系统所支持的每一种索引访问方法都有一行。

Table 1. pg_catalog.pg_am

列	类型	参考	描述
oid	oid		行标识符（隐藏属性;必须显式选择）
amname	name		访问方法的名称

列	类型	参考	描述
amstrategies	int2		此访问方法的运算符策略数，如果访问方法没有一组固定的运算符策略，则为零
amsupport	int2		此访问方法的支持例程数量
amcanorder	boolean		访问方法是否支持按索引列的值排序的有序扫描？
amcanorderbyop	boolean		访问方法是否支持按索引列上的运算符结果排序的有序扫描？
amcanbackward	boolean		访问方法是否支持后向扫描？
amcanunique	boolean		访问方法是否支持唯一索引？
amcanmulticol	boolean		访问方法是否支持多列索引？
amoptionalkey	boolean		访问方法是否支持对第一个索引列没有任何约束的扫描？
amsearcharray	boolean		访问方法是否支持ScalarArrayOpExpr搜索？
amsearchnulls	boolean		访问方法是否支持IS NULL/NOT NULL搜索？
amstorage	boolean		索引存储数据类型可以与列数据类型不同吗？

列	类型	参考	描述
amclusterable	boolean		这种类型的索引可以聚集在一起吗？
ampredlocks	boolean		这种类型的索引是否管理细粒度的谓词锁？
amkeytype	oid	pg_type.oid	存储在索引中的数据类型，如果不是固定类型，则为零
aminsert	regproc	pg_proc.oid	“插入此元组”函数
ambeginscan	regproc	pg_proc.oid	“准备索引扫描”函数
amgettupple	regproc	pg_proc.oid	“下一个有效元组”函数，如果没有则为零
amgetbitmap	regproc	pg_proc.oid	“获取所有元组”函数，如果没有，则返回零
amrescan	regproc	pg_proc.oid	“（重新）启动索引扫描”函数
amendscan	regproc	pg_proc.oid	“索引扫描后清理”函数
ammarkpos	regproc	pg_proc.oid	“标记当前扫描位置”函数
amrestrpos	regproc	pg_proc.oid	“恢复标记的扫描位置”函数
ambuild	regproc	pg_proc.oid	“建立新的索引”函数
ambuildempty	regproc	pg_proc.oid	“构建空索引”函数
ambulkdelete	regproc	pg_proc.oid	批量删除函数
amvacuumcleanup	regproc	pg_proc.oid	Post-VACUUM清理功能

列	类型	参考	描述
amcanreturn	regproc	pg_proc.oid	用于检查索引是否支持仅索引扫描的函数，如果没有则支持零
amcostestimate	regproc	pg_proc.oid	用于估计索引扫描成本的函数
amoptions	regproc	pg_proc.oid	为索引解析和验证relop tions的函数

Parent topic: [系统目录定义](#)

pg_amop

pg_amop表存储有关与索引访问方法操作符类关联的运算符的信息。 每个运算符都有一行是运算符类的成员。

条目的amopmethod必须与其包含的运算符族的opfmethod匹配（包括amopmethod这里是出于性能原因故意对catalog结构进行非规范化）。此外，amoplefttype和amoprightrighttype必须与引用的pg_operator条目的oprleft和oprright字段匹配。

Table 1. pg_catalog.pg_amop

列	类型	参考	描述
oid	oid		行标识符（隐藏属性;必须显式选择）

列	类型	参考	描述
amopfamily	oid	pg_opfamily.oid	此条目所针对的运算符系列
amoplefttype	oid	pg_type.oid	左侧输入数据类型的运算符
amoprightrighttype	oid	pg_type.oid	右侧输入数据类型的运算符
amopstrategy	int2		运算符策略编号
amoppurpose	char		运算符目的，s用于搜索或o用于订购
amopopr	oid	pg_operator.oid	运算符的OID
amopmethod	oid	pg_am.oid	运算符成员的索引访问方法
amopsortfamily	oid	pg_opfamily.oid	如果是一个排序运算符，则该条目按此排序的B树运算符系列; 如果是搜索运算符则为零

Parent topic: [系统目录定义](#)

pg_amproc

pg_amproc表存储与索引访问操作符相关的支持过程的信息。 对于属于一个操作符类的每个支持过程都有一行。

Table 1. pg_catalog.pg_amproc

列	类型	参考	描述
oid	oid		行标识符（隐藏属性;必须显式选择）
amprocfamily	oid	pg_opfamily.oid	此条目适用的运算符系列
amproclefttype	oid	pg_type.oid	左侧输入数据类型的运算符
amprocrighttype	oid	pg_type.oid	右侧输入数据类型的运算符
amprocnum	int2		支持过程编号
amproc	regproc	pg_proc.oid	过程的OID

Parent topic: [系统目录定义](#)

pg_appendonly

pg_appendonly表包含有关追加优化表的储存选项和其他特性的信息。

Table 1. pg_catalog.pg_appendonly

列	类型	参考	描述
relid	oid		压缩表的表对象标识符（OID）。

列	类型	参考	描述
blocksize	integer		用于追加优化表压缩的块尺寸。有效值为8K - 2M。默认值为32 K。
safefswritesize	integer		在非成熟文件系统中追加优化表的安全写操作的最小尺寸。通常设置为文件系统扩展块尺寸的倍数，例如Linux ext3是4096 字节，所以通常使用3 2768的值。
compresslevel	smallint		压缩级别，压缩比从1 增加到19。
majorversion	smallint		pg_appendonly表的主要版本号。
minorversion	smallint		pg_appendonly表的次要版本号。
checksum	boolean		存储的校验和值，在压缩和扫描时用来比较数据块的状态，以确保数据完整性。
compresstype	text		用于压缩追加优化表的压缩类型。有效值为： • zlib (gzip压缩) • zstd (Zstandard压缩)

列	类型	参考	描述
columnstore	boolean		1为列存，0为行存。
segrelid	oid		表在磁盘上的Segment文件ID。
segidxid	oid		索引在磁盘上的Segment文件ID。
blkdirrelid	oid		用于磁盘上列存表文件的块。
blkdiridxid	oid		用于磁盘上列存索引文件的块。
visimaprelid	oid		表的可见性映射。
visimapidxid	oid		可见性映射上的B-树索引。

Parent topic: [系统目录定义](#)

pg_attrdef

pg_attrdef表储存列的默认值，关于列的主要信息储存在[pg_attribute](#)中。只有那些显式指定了默认值的列（在创建表或者添加列时）才会在该表中有一项。

Table 1. pg_catalog.pg_attrdef

列	类型	参考	描述
adrelid	oid	pg_class.oid	该列所属的表

列	类型	参考	描述
adnum	int2	pg_attribute.att num	列编号
adbin	text		列默认值的内部表示
adsrc	text		人类可读的默认值表示。 这个字段是历史遗留下来的，最好别用。A h uman-readable represe ntation of the default val ue.

Parent topic: [系统目录定义](#)

pg_attribute

pg_attribute存储关于表列的信息。数据库中每个表的每一列都正好对应pg_attribute表的一行（还有有索引的属性项，以及所有有pg_class 项的对象的属性）。术语属性等效于列。

Table 1. pg_catalog.pg_attribute

列	类型	参考	描述
attrelid	oid	pg_class.oi d	该列所属的表。
attname	name		列名。
atttypid	oid	pg_type.oid	该列的数据类型。

列	类型	参考	描述
attstattarget	int4		控制由ANALYZE为此列积累的统计信息的详细程度。 0值表示不应收集统计信息。 负值表示使用系统默认的统计信息目标。 正值的确切含义依赖于数据类型。 对于标量数据类型，它既是要收集的“最常用值”的目标，也是要创建的柱状图的目标。
attlen	int2		该列类型的pg_type.typelen的副本。
attnum	int2		列编号。普通列从1开始编号。系统列（如OID），具有（任意）负编号。
attndims	int4		如果列是一个数组类型则是维度数；否则为0（目前，数组的维数不是强制的，所以任何非0值都能有效地表示它为一个数组）。
attcacheoff	int4		在存储中始终为-1，但是当加载到内存中的行描述符时，这可能会被更新以缓存该属性在行中的偏移量。

列	类型	参考	描述
atttypmod	int4		记录在表创建时提供的特定类型的数据（例如，varchar列的最大长度）。它被传递到特定类型的输入函数和长度强制函数。对于不需要它的类型，该值通常为-1。
attbyval	boolean		该列类型的pg_type.typbyval副本。
attstorage	char		通常是该列类型的pg_type.typstorage副本。对于可TOAST的数据类型来说，可以在列创建之后更改这些数据类型，以控制存储策略。
attalign	char		该列类型的pg_type.typalign副本。
attnotnull	boolean		这表示一个非空约束。可以更改此列以启用或禁用该约束。
atthasdef	boolean		此列具有默认值，这种情况下，将在pg_attrdef catalog中存在相应的条目实际定义默认值。

列	类型	参考	描述
attisdropped	boolean		该列已被删除，不再有效。已删除的列仍然物理存在于表中，但是会被解析器忽略，所以无法通过SQL访问。
attislocal	boolean		该列在表中本地定义。请注意，列可以同时在本地图定义和继承。
attinhcount	int4		这列的直接祖先的数量。具有非0数量祖先的列不能被删除或重命名。
attcollation	oid	pg_collation.oid	列的已定义排序规则，如果不是可合并数据类型，则为零。
attacl	aclitem[]		列级访问权限（如果已在此列上专门授予）。
attoptions	text[]		属性级选项，作为“key word = value”字符串。
attfdwoptions	text[]		属性级外部数据包装器选项，作为“keyword = value”字符串。

Parent topic: [系统目录定义](#)

pg_attribute_encoding

pg_attribute_encoding系统目录表包含列存储信息。

Table 1. pg_catalog.pg_attribute_encoding

列	类型	修正值	存储	描述
attrelid	oid	not null	plain	外键于pg_attribute.attrelid
attnum	smallint	not null	plain	外键于pg_attribute.attnum
attoptions	text []		extended	选项

Parent topic: [系统目录定义](#)

pg_auth_members

pg_auth_members系统目录表显示角色之间的成员关系。 允许任何非循环关系。因为角色是系统范围的，所以pg_auth_members表在MPP数据库系统的所有数据库之间共享。

Table 1. pg_catalog.pg_auth_members

列	类型	参考	描述
roleid	oid	pg_authid.oid	父级（组）角色的ID

列	类型	参考	描述
member	oid	pg_authid.oid	成员角色的ID
grantor	oid	pg_authid.oid	授予此成员关系的角色的ID
admin_option	boolean		如果角色成员可以向其他人授予成员关系，则为true

Parent topic: [系统目录定义](#)

pg_authid

pg_authid表包含了有关数据库认证标识符（角色）的信息。角色包含用户和组的概念。用户是设置了rolcanlogin标志的角色。任何角色（有或者没有rolcanlogin）可能有其他角色作为其成员。请参阅[pg_auth_members](#)。

由于此目录包含密码，因此不是公众可读的。 [pg_roles](#)是pg_authid上的一个公开可读的视图，其中模糊化了密码字段。

由于用户身份是全系统范围的，因此pg_authid在MPP数据库系统中的所有数据库之间共享：每个系统只有一个pg_authid副本，而不是每个数据库一个。

Table 1. pg_catalog.pg_authid

列	类型	参考	描述

列	类型	参考	描述
oid	oid		行标识符（隐藏属性;必须显式选择）
rolname	name		角色名称
rolsuper	boolean		角色具有超级用户特权
rolinherit	boolean		角色自动继承其所属角色的权限
rolcreatorole	boolean		角色可以创建其他更多角色
rolcreatedb	boolean		角色可以创建数据库
rolcatupdate	boolean		角色可以直接更新系统目录（即使超级用户也不能这样做，除非此列为真）
rolcanlogin	boolean		角色可以登录。也就是说，该角色可以作为初始会话授权标识符
rolreplication	boolean		角色是复制角色。也就是说，此角色可以使用pg_start_backup和pg_stop_backup启动流复制并设置/取消设置系统备份模式。
rolconndefault	int4		对于那些可以登录的角色，这一列设置此角色可以创建的最大并发连接数。 -1表示没有限制。

列	类型	参考	描述
rolpassword	text		密码（可能是加密的）；如果没有则为NULL。如果密码已加密，则此列将以字符串md5开头，后跟32个字符的十六进制MD5哈希。MD5哈希将是用户的密码连接到他们的用户名。例如，如果用户joe的密码为xyzzzy，则MPP数据库将存储xyzzzyjoe的md5哈希值。MPP假定不遵循该格式的密码未加密。
rolvaliduntil	timestampz		密码到期时间（仅用于密码验证）；不过期则为NULL
rolresqueue	oid		<i>pg_resqueue</i> 中关联资源队列ID的对象ID
rolcreaterextgpfd	boolean		使用gpfdist或gpfdists协议创建读取外部表的权限
rolcreaterexhttp	boolean		使用http协议创建读取外部表的权限
rolcreatewextgpfd	boolean		使用gpfdist或gpfdists协议创建写入外部表的权限
rolresgroup	oid		<i>pg_resgroup</i> 中关联资源组ID的对象ID

Parent topic: [系统目录定义](#)

pg_available_extension_versions

pg_available_extension_versions视图列出了可用于安装的特定扩展版本。
[pg_extension](#)系统目录表显示当前安装的扩展。

该视图是只读的。

Table 1. pg_catalog.pg_available_extension_versions

列	类型	描述
name	name	扩展名称。
version	text	版本名称。
installed	boolean	如果此扩展程序的此版本已安装，则为True，否则为False。
superuser	boolean	如果只允许超级用户安装此扩展，则为True，否则为False。
relocatable	boolean	如果扩展可以重新定位到另一个schema，则为True，否则为False。

列	类型	描述
schema	name	扩展必须装入的schema的名称，如果部分或完全可重定位，则为NULL。
requires	name[]	必备扩展的名称，如果没有则为NULL。
comment	text	扩展控制文件的注释字符串。

Parent topic: [系统目录定义](#)

pg_available_extensions

该pg_available_extensions视图列出了可用于安装的扩展。该[pg_extension](#)系统目录表显示当前安装的扩展。

该视图是只读的。

Table 1. pg_catalog.pg_available_extensions

列	类型	描述
name	name	扩展名称
default_version	text	默认版本名，如果没指定的话为NULL。
installed_version	text	当前安装的扩展的版本，如果未安装则为NULL。
comment	text	扩展控制文件的注释字符串。

Parent topic: [系统目录定义](#)

pg_cast

pg_cast表存储数据类型转换路径，包括内置路径和用CREATE CAST定义的路径。

请注意，pg_cast不代表系统已知的每种类型转换，只代表那些无法从某些通用规则推导出的转换。例如，在pg_cast中未显式表示域与其基类型之间的转换。另一个重要的例外是“自动I/O转换强制转换”，使用数据类型自己的I/O函数执行转换为文本或其他字符串类型的那些，在pg_cast中没有显式表示。

pg_cast中列出的强制转换函数必须始终将强制转换源类型作为其第一个参数类型，并返回强制转换目标类型作为其结果类型。强制转换函数最多可以包含三个参数。第二个参数（如果存在）必须是整数类型；它接收与目标类型关联的类型修饰符，如果没有，则返回-1。第三个参数（如果存在）必须是boolean类型；如果强制转换是显式转换，则接收为true，否则为false。

如果关联函数需要多个参数，则创建一个源和目标类型相同的pg_cast条目是合法的。这些条目表示“长度强制函数”，其强制该类型的值对于特定类型修饰符值是合法的。

当pg_cast条目具有不同的源和目标类型以及具有多个参数的函数时，该条目将从一种类型转换为另一种类型，并在单个步骤中应用长度强制。当没有这样的条目可用时，对使用类型修饰符的类型的强制涉及两个步骤，一个用于在数据类型之间进行转换，另一个用于应用修饰符。

Table 1. pg_catalog.pg_cast

列	类型	参考	描述
castsource	oid	pg_type.oid	源数据类型的OID。
casttarget	oid	pg_type.oid	目标数据类型的OID。
castfunc	oid	pg_proc.oid	用于执行此强制转换的函数的OID。 如果强制转换方法不需要函数，则存储零。
castcontext	char		指示可以调用强制转换的上下文。 e仅表示显式强制转换（使用CAST或::语法）。 a表示隐式分配给目标列，和显式一样。 i表示隐式在表达式中，以及其他情况。
castmethod	char		指出转换的执行方式： f - 使用castfunc字段中标识的函数。 i - 使用输入/输出函数。 b - 这些类型是二进制可强制的，不需要转换。

Parent topic: [系统目录定义](#)

pg_class

系统目录表pg_class记录表以及其他大部分具有列或者与表（也称为关系）相似的东西。这包括索引（另见[pg_index](#)）、序列、视图、组合类型和TOAST表。并不是所有的列对所有的关系类型都有意义。

Table 1. pg_catalog.pg_class

列	类型	参考	描述
relname	name		表、索引、视图等的名字。
relnamespace	oid	pg_namespace.oid	包含这个关系的命名空间（schema）的OID
reltype	oid	pg_type.oid	如果有的话（索引为0，没有pg_type项），对应与此表的行类型的数据类型的OID
reloftype	oid	pg_type.oid	pg_type中条目的OID，用于隐含的复合类型。
relowner	oid	pg_authid.oid	关系的所有者
relam	oid	pg_am.oid	如果这是一个索引，则表示访问方法（B树、位图、哈希等。）
relfilenode	oid		此关系的磁盘文件的名称，如果没有则为0。

列	类型	参考	描述
reltablespace	oid	pg_tablespace.oid	存储此关系的表空间。如果为0，则表示数据库的默认表空间（如果关系没有磁盘文件，则无意义）。The tablespace in which this relation is stored. If zero, the
relpages	int4		该表的磁盘尺寸，以页面为单位（每页32k）。这只是优化器使用的估计值。它由VACUUM、ANALYZE和一些DDL命令更新。
reltuples	float4		表中的行数。这只是优化器使用的估计值。它由VACUUM、ANALYZE和一些DDL命令更新。
relallvisible	int32		全部可见块的数量（此值可能不是最新的）。
reltoastrelid	oid	pg_class.oid	与这张表关联的TOAST表的OID，没有的就为0。TOAST表在辅助表中存储“行溢出”的大型属性。

列	类型	参考	描述
relhasindex	boolean		如果这是一个表并且它（或最近有）任何索引，则为True。这由CREATE INDEX设置，但不会立即由DROP INDEX清除。如果发现表没有索引，VACUUM将清除。
relisshared	boolean		如果此表在系统中的所有数据库之间共享，则为True。仅共享某些系统目录表。
relpersistence	char		对象持久性的类型：p = 堆或追加优化表，u = 未记录的临时表，t = 临时表。
relkind	char		对象的类型 r = 堆或追加优化表，i = 索引，S = 序列，t = TOAST值，v = 视图，c = 组合类型，f = 外部表，u = 未登记的临时堆表，o = 内部追加优化的segment文件和EOF，b = 追加的块目录，M = 追加的可视化映射

列	类型	参考	描述
relstorage	char		表的存储模式 a= 追加优化，c= 面向列的，h = 堆表，v = 虚拟，x= 外部表。
relnatts	int2		关系中用户列的数量（系统列不计入）。pg_attribute中必须有这么多个相应的项。
relchecks	int2		表中检查约束的个数。
relhasoids	boolean		如果为关系的每一行生成OID，则为True。
relhaspkey	boolean		如果表具有（或曾经拥有）主键，则为True。
relhasrules	boolean		如果表有规则，则为True。
relhastriggers	boolean		如果表有（或曾经有）触发器，则为True。
relhassubclass	boolean		如果表具有（或曾经有）任何继承子项，则为True。
relispopulated	boolean		如果关系被填充，则为真（对于除某些物化视图之外的所有关系都是如此）。

列	类型	参考	描述
relreplident	char		用于为行形成“副本标识”的列：d =默认（主键，如果有），n =无，f =所有列 i =具有indisreplident设置的索引，或默认值
relfrozenxid	xid		此表中该值之前的所有事务ID都已替换为此表中的永久（冻结）事务ID。 这用于跟踪表是否需要被清理以防止事务ID环绕或允许pg_clog收缩。 如果关系不是表，或者表不需要清理以防止事务ID环绕，则值为0（InvalidTransactionId）。该表仍可能需要清理才能回收磁盘空间。
relminmxid	xid		此表中该值之前的所有multixact ID都被此表中的事务ID替换。 这用于跟踪表是否需要被清理以防止多重ID环绕或允许pg_multixact收缩。 如果关系不是表，则为零（InvalidMultiXactId）。
relacl	aclitem[]		GRANT和REVOKE分配的访问权限。

列	类型	参考	描述
reloptions	text[]		特定于访问方法的选项，作为“keyword = value”字符串。

Parent topic: [系统目录定义](#)

pg_compression

pg_compression系统目录表描述了可用的压缩方法。

Table 1. pg_catalog.pg_compression

列	类型	修饰符	存储	描述
compname	name	not null	plain	压缩方法的名字
compconstruc tor	regproc	not null	plain	压缩方法构造函数名称
compdestruct or	regproc	not null	plain	压缩方法析构函数名称
compcompres sor	regproc	not null	plain	压缩器的名称
compdecomp ressor	regproc	not null	plain	解压缩器的名称
compvalidato r	regproc	not null	plain	压缩验证器的名称

列	类型	修饰符	存储	描述
compowner	oid	not null	plain	pg_authid的oid

Parent topic: [系统目录定义](#)

pg_constraint

pg_constraint系统目录表储存表上的检查、主键、唯一和外键约束。列约束不被特别对待。每个列约束都等效于某种表约束。非空约束在[pg_attribute](#)目录表中表示。域上的检查约束也储存在这里。

Table 1. pg_catalog.pg_constraint

列	类型	参考	描述
conname	name		约束名称（不一定唯一！）
connamespace	oid	pg_namespace.oid	包含约束的命名空间（schema）的OID。
contype	char		c = 检查约束，f = 外键约束，p = 主键约束，u = 唯一约束。
condeferrable	boolean		约束是否可以延迟？
condeferred	boolean		约束是否默认延迟？
conrelid	oid	pg_class.oid	约束所在的表；如果不是一个表约束则为0。

列	类型	参考	描述
contypid	oid	pg_type.oid	该约束所在的域；如果不是一个域约束则为0。
confrelid	oid	pg_class.oid	如果是一个外键，则表示所引用的表；否则为0。
confupdtype	char		外键更新动作代码。
confdeltype	char		外键删除动作代码。
confmacthype	char		外键匹配类型。
conkey	int2[]	pg_attribute.att num	如果是一个表约束，则表示所约束的列的列表。
confkey	int2[]	pg_attribute.att num	如果是一个外键，则表示所引用列的列表。
conbin	text		如果是一个检查约束，则表示表达式的内部表示。
consrc	text		如果是一个检查约束，则表示表达式的人类可读的表示。 当被引用对象改变时候，这个属性不会被更新；例如，它不会跟踪列的重命名。最好使用pg_get_constraintdef()来提取检查约束的定义，而不是依赖此字段。

Parent topic: [系统目录定义](#)

pg_conversion

pg_conversion系统目录表描述了由CREATE CONVERSION定义的可用编码转换过程。

Table 1. pg_catalog.pg_conversion

列	类型	参考	描述
conname	name		转换的名称（在命名空间中唯一）。
connamespace	oid	pg_namespace.oid	包含此转换的命名空间（schema）的OID。
conowner	oid	pg_authid.oid	转换的拥有者。
conforencoding	int4		源编码ID。
contoencoding	int4		目标编码ID。
conproc	regproc	pg_proc.oid	转换过程。
condefault	boolean		如果是默认转换，则为真。

Parent topic: [系统目录定义](#)

pg_database

pg_database系统目录表储存了可用数据库的信息。数据库由SQL命令CREATE DATABASE创建。和大多数系统目录不同，pg_database在系统中所有数据库之间共享。每个系统只有一个pg_database副本，而不是每个数据库一个。

Table 1. pg_catalog.pg_database

列	类型	参考	描述
datname	name		数据库名称。
datdba	oid	pg_authid.oid	数据库的拥有者，通常是创建它的人。
encoding	int4		数据库的字符编码。 pg_encoding_to_char()可以将此编号转换为编码名称。
datcollate	name		此数据库的LC_COLLATE。
datctype	name		此数据库的LC_CTYPE。
datistemplate	boolean		如果为真则这个数据库可以被用在CREATE DATABASE的TEMPLATE子句中来创建一个新的数据库作为这个数据库的克隆体。

列	类型	参考	描述
datallowconn	boolean		如果为假，则该数据库不可连接。 这用于保护数据库template0不被修改。
datconndefault	int4		设置该数据库最大并发连接数。 -1表示没有限制。
datlastsysoid	oid		数据库中的最后一个系统OID。
datfrozenxid	xid		这个数据库中在此值之前的所有事务ID已被替换为永久（冻结）事务ID。 这用于追踪该数据库是否需要清理，以防止事务ID回卷或允许pg_clog收缩，它是每个表的pg_class.relfrozeoid的最小值。

列	类型	参考	描述
datminmxid	xid		<i>A Multixact ID is used to support row locking by multiple transactions. All multixact IDs before this one have been replaced with a transaction ID in this database. This is used to track whether the database needs to be vacuumed in order to prevent multixact ID wraparound or to allow pg_multixact to be shrunk. It is the minimum of the per-table <code>pg_class.relmminxid</code> values.</i>
dattablespace	oid	pg_tablespace.oid	数据库的默认表空间。在此数据库中，所有 <code>pg_class.reltablespace</code> 为0的表都将存储在此表空间中。所有非共享的系统目录也将放在那里。
datacl	aclitem[]		由GRANT和REVOKE所给予的数据库访问特权。

Parent topic: [系统目录定义](#)

pg_db_role_setting

pg_db_role_setting系统目录表记录每个角色和数据库组合的服务器配置设置的默认值。

每个MPP数据库集群都有一个pg_db_role_settings副本。 此系统目录表在所有数据库中共享。

您可以使用psql的\drds元命令查看MPP数据库集群的服务器配置设置。

Table 1. pg_catalog.pg_database

列	类型	参考	描述
setdatabase	oid	pg_databas e.oid	设置适用的数据库，如 果设置不是特定于数据 库，则为零。
setrole	oid	pg_authid.oi d	设置适用的角色，如果 设置不是特定于角色， 则为零。
setconfig	text[]		用户可设置的服务器配 置参数的每个数据库和 每个角色特定的默认值 。

Parent topic: [系统目录定义](#)

pg_depend

pg_depend系统目录表记录数据库对象之间依赖关系。此信息允许DROP命令查找哪些其他对象必须由DROP CASCADE删除或者必须防止在DROP RESTRICT情况中被删除。另请参见[pg_shdepend](#)，它对涉及MPP系统共享对象的依赖执行类似的功能。

在所有的情况下，pg_depend项表示在不删除依赖对象的前提下也不能删除被引用对象。不过，有几种由deptype区分的子类型：

- **DEPENDENCY_NORMAL (n)** — 单独创建的对象之间的正常关系。可以删除依赖对象而不影响被引用的对象。被引用对象只能通过指定CASCADE来删除，在这种情况下依赖对象也会被删除。例如：表列对其数据类型具有正常的依赖性。
- **DEPENDENCY_AUTO (a)** — 依赖对象可以独立于被引用对象而删除，并且如果被引用对象被删除，则依赖对象应该自动被删除（不管是RESTRICT还是CASCADE模式）。例如：表上的命名约束自动依赖表，因此如果表被删除，该约束也将消失。
- **DEPENDENCY_INTERNAL (i)** — 依赖对象作为被引用对象的一部分创建，实际只是其内部实现的一部分。依赖对象的DROP操作将被完全禁止（我们会告诉用户针对被引用对象发出DROP命令来代替）。被引用对象的DROP命令会被传播以删除依赖对象，不管是否指定CASCADE（级联删除）。
- **DEPENDENCY_PIN (p)** — 没有依赖对象；这种类型的条目是系统本身依赖于引用对象的信号，因此绝不能删除该对象。此类型的条目仅由系统初始化创建。从属对象的列包含零。

Table 1. pg_catalog.pg_depend

列	类型	参考	描述
classid	oid	pg_class.oid	依赖对象所在的系统目录的OID。

列	类型	参考	描述
objid	oid	any OID column	特定依赖对象的OID。
objsubid	int4		对于表列，这是列编号。对于其他对象类型，此列为0。
refclassid	oid	pg_class.oid	被引用对象所在的系统目录的OID。
refobjid	oid	any OID column	特定被引用对象的OID。
refobjsubid	int4		对于表列，这是被引用的列编号。对于其他对象类型，该列为0。
deptype	char		定义此依赖关系的特定语义的代码。

Parent topic: [系统目录定义](#)

pg_description

pg_description系统目录表存储每个数据库对象的可选描述（注释）。可以使用COMMENT命令来操作描述，并使用psql的\d元命令来查看。

pg_description的初始内容中提供了许多内建系统对象的描述。另请参见[pg_shdescription](#)，它对涉及MPP数据库系统共享对象的描述执行类似的功能。

Table 1. pg_catalog.pg_description

列	类型	参考	描述
objoid	oid	any OID column	该描述所涉及对象的OID。
classoid	oid	pg_class.oid	该对象所出现的系统目录的OID。
objsubid	int4		对于表的注释，这是列编号。 对于其他对象类型，此列为0。
description	text		作为此对象描述的任意文本。

Parent topic: [系统目录定义](#)

pg_enum

该pg_enum表包含与其相关联的值和标签匹配的枚举类型条目。 给定枚举值的内部表示其实是pg_enum表中相关行的OID。 特定枚举类型的Oid保证按照应有类型排序方式进行排序，但是不保证不相关枚举类型Oid的排序。

Table 1. pg_catalog.pg_enum

列	类型	参考	描述
enumtypid	oid	pgtype.oid	拥有此枚举类型pg_type条目的OID

列	类型	参考	描述
enumlabel	name		此枚举值的文本标签

Parent topic: [系统目录定义](#)

pg_extension

该系统目录表pg_extension存储了关于安装的扩展的信息。

Table 1. pg_catalog.pg_extension

列	类型	参考	描述
extname	name		扩展的名称。
extowner	oid	pg_authid.oid	扩展所有者
extnamespace	oid	pg_namespace.oid	包含扩展导出对象的Schema
extrelocatable	boolean		如果该扩展能够重定位到其他schema，则为真。
extversion	text		扩展的版本名字
extconfig	oid[]	pg_class.oid	扩展配置表的regclass OIDs 数组，如果为空则为NULL。

列	类型	参考	描述
extcondition	text[]		扩展配置表的WHERE-clause过滤条件数组，如果没有则为NULL。

与大多数命名空间列的目录不同，extnamespace并不意味着该扩展属于这个schema。扩展的名称不符合schema限制。extnamespace schema指示其包含大部分或全部扩展对象的schema。如果extrelocatable为true，则该schema必须包含所有属于扩展名的符合schema限定的对象。

Parent topic: [系统目录定义](#)

pg_exttable

pg_exttable系统目录表用来追踪由CREATE EXTERNAL TABLE命令创建的外部表和Web表。

Table 1. pg_catalog.pg_exttable

列	类型	参考	描述
relid	oid	pg_class.oid	该外部表的OID。
urilocation	text[]		外部表文件的URI地址。
execlocation	text[]		为外部表定义的ON Segment位置。

列	类型	参考	描述
fmttype	char		外部表文件的格式：t是文本，c是csv。
fmtopts	text		外部表文件的格式化选项，如字段定界符、空字符串、转义字符等。
options	text[]		为外部表定义的选项。
command	text		访问外部表所执行的OS命令。
rejectlimit	integer		每个Segment拒绝错误行的限制，之后装载将失败。
rejectlimittype	char		拒绝限制的阈值类型：r表示行数。
logerrors	bool		1记录错误，0不记录。
encoding	text		客户端编码。
writable	boolean		0表示可读外部表，1表示可写外部表。

Parent topic: [系统目录定义](#)

pg_foreign_data_wrapper

系统目录表pg_foreign_data_wrapper存储外部数据包装器定义。 外部数据包装器是一种访问驻留在外部服务器上的外部数据的机制。

Table 1. pg_catalog.pg_foreign_data_wrapper

列	类型	参考	描述
fdwname	name		外部数据包装器的名称。
fdwowner	oid	pg_authid.oid	外部数据包装器的所有者。
fdwhandler	oid	pg_proc.oid	对处理程序函数的引用，该函数负责为外部数据包装器提供执行例程。如果没有提供处理程序则为零。
fdwvalidator	oid	pg_proc.oid	对验证器函数的引用，该函数负责检查提供给外部数据包装器的选项的有效性。此函数还使用外部数据包装器检查外部服务器和用户映射的选项。如果未提供验证器，则为零。
fdwacl	aclitem[]		访问权限; 有关详细信息，请参阅 GRANT 和 REVOKE 。
fdwoptions	text[]		外部数据包装器特定选项，作为“keyword = value”字符串。

Parent topic: [系统目录定义](#)

pg_foreign_server

系统目录表pg_foreign_server存储外部服务器定义。 外部服务器描述外部数据源，例如远程服务器。 您可以通过外部数据包装器访问外部服务器。

Table 1. pg_catalog.pg_foreign_server

列	类型	参考	描述
srvname	name		外部服务器的名称。
srvowner	oid	pg_authid.oid	外部服务器的所有者。
srvfdw	oid	pg_foreign_data_wrapper.oid	此外部服务器的外部数据包装器的OID。
srvtype	text		服务器类型（可选）。
srvversion	text		服务器版本（可选）。
srvaccl	aclitem[]		访问权限; 有关详细信息，请参阅 GRANT 和 EVOKE 。
srvoptions	text[]		外部服务器特定选项，作为“keyword = value”字符串。

Parent topic: [系统目录定义](#)



系统目录表pg_foreign_table包含有关外部表的辅助信息。 外表主要由pg_class条目表示，就像常规表一样。 它的pg_foreign_table条目包含仅与外表相关的信息，而不包含任何其他类型的关系。

Table 1. pg_catalog.pg_foreign_table

列	类型	参考	描述
ftrelid	oid	pg_class.oid	此外表的pg_class条目的OID。
ftserver	oid	pg_foreign_server.oid	此外表的外部服务器的OID。
ftoptions	text[]		外表选项，如“keyword = value”字符串。 .

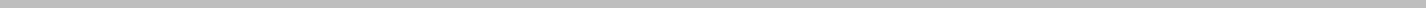
Parent topic: [系统目录定义](#)

pg_index

pg_index系统目录表包含部分关于索引的信息。 剩下的信息主要在[pg_class](#)表中。

Table 1. pg_catalog.pg_index

列	类型	参考	描述
indexrelid	oid	pg_class.oid	该索引的pg_class项的OID。



列	类型	参考	描述
indrelid	oid	pg_class.oid	该索引所对应的表的pg_class项的OID。
indnatts	int2		索引中的列数（与其pg_class.relnatts重复）。
indisunique	boolean		如果为真，则这是一个唯一索引。
indisprimary	boolean		如果为真，则这个索引表示表的主键（此属性为真时indisunique总是为真）。
indisexclusion	boolean		如果为true，则此索引支持排除约束
indimmediate	boolean		如果为true，则在插入时立即强制执行唯一性检查（如果indisunique不成立则无关紧要）
indisclustered	boolean		如果为真，表最后一次是通过CLUSTER命令在此索引上聚簇。
indisvalid	boolean		如果为真，则该索引目前对查询有效。 如果为假，意味着该索引目前可能不完整：它仍然必须被INSERT/UPDATE操作所修改，但是它不能被安全地用于查询。

列	类型	参考	描述
indcheckxmin	boolean		如果为true，则查询必须不使用索引，直到此pg_index行的xmin低于其TransactionXmin事件范围，因为该表可能包含具有可以看到的不兼容行的断开的HOT链
indisready	boolean		如果为true，则索引当前已准备好进行插入。False意味着INSERT/UPDATE操作必须忽略索引
indislive	boolean		如果为false，则索引正在被删除，并且应该为所有目的而忽略
indisreplident	boolean		如果为true，则使用ALTER TABLE ... REPLICA IDENTITY USING INDEX ...将此索引选为“副本标识”
indkey	int2vector	pg_attribute.attnum	这是一个indnatts值数组，指示此索引索引的表列。例如，值为1 3意味着第一个和第三个表列构成索引键。此数组中的零表示相应的索引属性是表列上的表达式，而不是简单的列引用。

列	类型	参考	描述
indcollation	oidvector		对于索引键中的每一列，它包含用于索引的排序规则的OID。
indclass	oidvector	pg_opclass.oid	对于索引键中的每一列，它包含要使用的运算符类的OID。
indoption	int2vector		这是一个indnatts值数组，用于存储每列标志位。位的含义由索引的访问方法定义。
indexprs	text		表达式树（在nodeToString()表示中），用于不是简单列引用的索引属性。这是一个列表，其中包含indkey中每个零条目的一个元素。如果所有索引属性都是简单引用，则为NULL
indpred	text		部分索引谓词的表达式树（在nodeToString()表示中）。如果不是部分索引，则为NULL。

Parent topic: [系统目录定义](#)

pg_inherits系统目录表记录有关表继承层次结构的信息。数据库中的每个直接子表都有一个条目。（间接继承可以通过以下条目链来确定。）在MPP数据库中，继承关系由CREATE TABLE的INHERITS子句（独立继承）和PARTITION BY 子句（分区子表继承）创建。

Table 1. pg_catalog.pg_inherits

列	类型	参考	描述
inhrelid	oid	pg_class.oid	子表的OID。
inhparent	oid	pg_class.oid	父表的OID。
inhseqno	int4		如果对于一个子表有多个直接父表（多继承），则这个数字将指出被继承列的排列顺序。计数从1开始。

Parent topic: [系统目录定义](#)

pg_language

pg_language系统目录表注册用户可以用来编写函数和存储程序的语言。它由CREATE LANGUAGE填充。

Table 1. pg_catalog.pg_language

列	类型	参考	描述
lanname	name		语言的名称。
lanowner	oid	pg_authid.oid	语言的拥有者。
lanispl	boolean		对内部语言（如SQL）而言，值为假。而对于用户自定义的语言为真。目前，pg_dump仍然使用它来确定哪些语言需要被转存，但是，在将来它可能会被不同的机制所代替。
lanpltrusted	boolean		如果这是一种可信的语言，则为真，表示它不会为正常SQL执行环境之外的任何东西授予访问。只有超级用户才能用不可信语言创建函数。
lanplcallfoid	oid	pg_proc.oid	对于非内部的语言，该属性引用了一个语言处理程序，该程序是一个特殊的函数，负责执行所有以特定语言编写的函数。

列	类型	参考	描述
laninline	oid	pg_proc.oid	这个属性引用一个函数负责执行内联匿名代码块的函数（参见DO命令）。 如果不支持匿名块，则为0。
lanvalidator	oid	pg_proc.oid	这个属性引用一个语言验证器函数，负责在创建新函数时检查新函数的语法和合法性。 如果没有提供验证器，则为0。
lanacl	aclitem[]		语言的访问特权。

Parent topic: [系统目录定义](#)

pg_largeobject

Note: MPP数据库不支持PostgreSQL[大对象工具](#)，用于流式传输存储在大型对象结构中的用户数据。

pg_largeobject系统目录表保存构成“大对象”的数据。 大对象由创建时分配的OID来识别。 每个大对象被分解成便于在pg_largeobject表中储存为行的足够小的段或者“页”。 每页的数据量被定义为LOBLKSIZE（当前是BLCKSZ/4，或者通常是 8K）。

pg_largeobject的每一行都保存一个大对象的一个页，从对象内的字节偏移量（*pageno* * LOBLKSIZE）开始。 该实现允许稀疏存储：页面可以丢失，并且即

使它们不是对象的最后一页也可能比LOBLKSIZE字节更短。 大对象中缺少的区域被读作0。

Table 1. pg_catalog.pg_largeobject

列	类型	参考	描述
loid	oid		包含这一页的大对象的标识符。
pageno	int4		大对象中该页的页号（从0开始计数）。
data	bytea		存储在大对象中的实际数据。该数据不会超过LOBLKSIZE字节数而且可能会更小。

Parent topic: [系统目录定义](#)

pg_listener

pg_listener系统目录表支持LISTEN和NOTIFY命令。 监听器在pg_listener中为每个正在监听的通知名称创建一项。 通知者扫描和更新每个匹配的项以显示通知已经发生。 通知者也会发送一个信号（使用表中记录的PID）将监听器从睡眠中唤醒。

此表目前不在MPP数据库中使用。

Table 1. pg_catalog.pg_listener

列	类型	参考	描述
relname	name		通知条件名称（该名称不需要匹配数据库中任何实际关系）。
listenerpid	int4		创建这个项的服务器进程的PID。
notification	int4		如果这个监听器没有事件待处理，则为0。如果有待处理事件，则为发送通知的服务器进程的PID。

Parent topic: [系统目录定义](#)

pg_locks

pg_locks视图提供了有关在MPP数据库中由打开的事务持有的锁的信息的访问。

pg_locks包含一行关于每个积极可锁对象，请求的锁模式和相关事务。 因此，如果多个事务正在持有或等待其上的锁，同样的可锁对象可能会出现多次。 但是，目前没有锁的对象根本就不会出现。

存在几种不同类型的可锁定对象：完整关系（例如表），关系的各个页面，关系的单个元组，事务ID（虚拟和永久ID）以及通用数据库对象。 此外，扩展关系

的权利表示为单独的可锁定对象。

Table 1. pg_catalog.pg_locks

列	类型	参考	描述
locktype	text		可锁对象的类型：relation, extend, page, tuple, transactionid, object, userlock, resource queue 或 advisory
database	oid	pg_database.oid	该对象存在的数据库的Oid，如果该对象是共享对象，则为0。如果对象是事务ID，则为空。
relation	oid	pg_class.oid	关系的OID，如果对象不是关系或者关系的一部分，则为NULL。
page	integer		关系中的页码，如果对象不是元组或者关系页则为NULL
tuple	smallint		页中的元组号，如果该对象不是个元组则为NULL
virtualxid	text		事务的虚拟ID，如果对象不是虚拟事务ID，则为NULL
transactionid	xid		事务的ID，如果该对象不是一个事务ID，则为NULL

列	类型	参考	描述
classid	oid	pg_class.oid	包含对象的系统目录的OID，如果对象不是一般数据库对象，则为NULL
objid	oid	any OID column	其系统目录中对象的OID，如果对象不是一般数据库对象，则为NULL
objsubid	smallint		对于一个表列来说，这是列号（classid和objid引用表本身）。对于所有其他的对象类型，此列为0。如果对象不是数据库对象，则为NULL
virtualtransaction	text		持有或等待此锁定的事务的虚拟ID
pid	integer		持有或等待该锁的事务进程的进程Id。如果锁定由准备好的事务持有，则为NULL
mode	text		该进程所持有或期望的锁模式的名称
granted	boolean		锁被持有为真，锁为等待为假
fastpath	boolean		如果通过快速路径进行锁定则为True，如果通过主锁定表进行锁定则为false。

列	类型	参考	描述
mppsessionid	integer		与锁相关的客户端会话的id。
mppiswriter	boolean		指明该锁是否由一个写进程所持有。
gp_segment_id	integer		MPP持有该锁的segment的id（dbid）。

Parent topic: [系统目录定义](#)

pg_max_external_files

pg_max_external_files视图显示使用外部表文件协议时每台Segment主机允许的外部表文件最大数量。

Table 1. pg_catalog.pg_max_external_files

列	类型	参考	描述
hostname	name		在Segment主机上访问特定Segment实例所使用的主机名。
maxfiles	bigint		该主机上的主Segment实例数。

Parent topic: [系统目录定义](#)

pg_namespace

pg_namespace系统目录表存储命名空间。命名空间是SQL模式的底层结构：每个命名空间都可以具有单独的没有命名冲突的关系、类型等集合。

Table 1. pg_catalog.pg_namespace

列	类型	参考	描述
oid	oid		行标识符（隐藏属性;必须显式选择）
nspname	name		命名空间的名字
nspowner	oid	pg_authid.oid	命名空间的所有者
nspacl	aclitem[]		由GRANT和REVOKE给予的访问特权

Parent topic: [系统目录定义](#)

pg_opclass

pg_opclass系统目录表定义索引访问方法操作符类。每个运算符类为特定数据类型的索引列和特定索引访问方法定义语义。运算符类实质上指定特定运算符族适用于特定的可索引列数据类型。系列中实际可用于索引列的运算符集是那

些接受列的数据类型作为其左侧输入的运算符。

运算符类的opcmethod必须与其包含运算符族的opfmethod匹配。此外，对于任何给定的opcmethod和opcintype组合，必须有不超过一个pg_opclass行具有opcdefault为true。

Table 1. pg_catalog.pg_opclass

列	类型	参考	描述
oid	oid		行标识符（隐藏属性;必须显式选择）
opcmethod	oid	pg_am.oid	运算符类用的索引访问方法
opcname	name		运算符类的名称
opcnamespace	oid	pg_namespace.oid	运算符类所属的命名空间
opcowner	oid	pg_authid.oid	运算符类的拥有者
opcfamily	oid	pg_opfamily.oid	包含运算符类的运算符族
opcintype	oid	pg_type.oid	运算符类索引的数据类型
opcdefault	boolean		如果此运算符类是数据类型opcintype的缺省值，则为True
opckeytype	oid	pg_type.oid	存储在索引中的数据类型，如果与opcintype相同则为零

Parent topic: [系统目录定义](#)

pg_operator

系统目录表pg_operator存储关于操作符的信息，包括内建的和通过CREATE OPERATOR语句定义的操作符。未用的列包含零值。例如，一个前缀操作符的oprleft为0。

Table 1. pg_catalog.pg_operator

列	类型	参考	描述
oid	oid		行标识符（隐藏属性;必须显式选择）
oprname	name		操作符的名称
oprnamespace	oid	pg_namespace.oid	操作符所属的名字空间的OID
oprowner	oid	pg_authid.oid	操作符的拥有者
oprkind	char		b = 中缀 (前缀和后缀) ，l = 前缀 ("左")，r = 后缀 ("右")
oprcanmerge	boolean		此运算符是否支持合并连接
oprcanhash	boolean		该操作符是否支持哈希连接

列	类型	参考	描述
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	此运算符的转换器，如果有的话
oprnegate	oid	pg_operator.oid	该操作符的求反器，如果有的话
oprcode	regproc	pg_proc.oid	实现该操作符的函数
oprrest	regproc	pg_proc.oid	该运算符的限制选择性估计函数
oprjoin	regproc	pg_proc.oid	加入此运算符的选择性估计函数

Parent topic: [系统目录定义](#)

pg_opfamily

目录pg_opfamily定义了运算符系列。 每个运算符族都是运算符和相关支持例程的集合，这些例程实现了为特定索引访问方法指定的语义。 此外，一个系列中的运算符都以访问方法指定的方式兼容。 运算符族概念允许跨数据类型的运算符与索引一起使用，并被推理为使用访问方法语义的知识。

定义运算符族的大多数信息不在其pg_opfamily行中，而是在[pg_amop](#)，[pg_amproc](#)和[pg_opclass](#)中的关联行中。

Table 1. pg_opfamily

名称	类型	参考	描述
oid	oid		行标识符（隐藏属性;必须显式选择）
opfmethod	oid	pg_am.oid	此系列的索引访问方法运算符
opfname	name		此运算符系列的名称
opfnamespace	oid	pg_namespace.oid	此运算符系列的命名空间
opfowner	oid	pg_authid.oid	此运算符系列的所有者

Parent topic: [系统目录定义](#)

pg_partition

pg_partition系统目录表被用来跟踪分区表以及它们的继承层级关系。
pg_partition中的每一行要么代表了一个分区表在分区层级关系中的等级，要么是一个子分区模板的描述。 paristemplate的属性值决定了一个特定行代表的含义。

Table 1. pg_catalog.pg_partition

列	类型	参考	描述
parrelid	oid	pg_class.oid	表的对象标识符。

列	类型	参考	描述
parkind	char		分区类型 - R表示范围或 L表示列表。
parlevel	smallint		该行的分区级别：0代表最顶层的父表，1代表父表下的第一个级别，2代表第二个级别，以此类推。
paristemplate	boolean		该行是否代表一个子分区模板定义（true）或者实际分区级别（false）。
parnatts	smallint		定义该级别的属性数目。
paratts	smallint()		参与定义该级别的属性编号（正如在pg_attribute.attnum中的）数组。
parclass	oidvector	pg_opclass.oid	分区列上的操作符类的标识符。

Parent topic: [系统目录定义](#)

pg_partition_columns

pg_partition_columns系统视图被用来显示一个分区表的分区键列。

Table 1. pg_catalog.pg_partition_columns

列	类型	参考	描述
schemaname	name		该分区表所在schema的名称。
tablename	name		顶层父表的表名。
columnname	name		分区键列的名称。
partitionlevel	smallint		该子分区在层次中的级别。
position_in_partition_key	integer		对于列表分区可以有组合（多列）分区键。此处显示了列在组合键中的位置。

Parent topic: [系统目录定义](#)

pg_partition_encoding

pg_partition_encoding系统目录表描述一个分区模板的可用的列压缩选项。

Table 1. pg_catalog.pg_attribute_encoding

列	类型	修饰符	存储	描述
parencoid	oid	not null	plain	
parencattnum	smallint	not null	plain	

列	类型	修饰符	存储	描述
parencattoptions	text []		extended	

Parent topic: [系统目录定义](#)

pg_partition_rule

pg_partition_rule系统目录表被用来跟踪分区表、它们的检查约束以及数据包含规则。 pg_partition_rule表中的每一行要么代表了一个叶子分区（最底层包含数据的分区），要么是一个分支分区（用于定义分区层次的顶层或者中间层分区，但不包含数据）。

Table 1. pg_catalog.pg_partition_rule

列	类型	参考	描述
paroid	oid	pg_partition.oid	这个分区所属的分区级别的行标识符（来自 pg_partition ）。对于分支分区，相应的表（由pg_partition_rule标识）是一个空的容器表。对于叶子分区，这个表含有分区包含规则的行。
parchildrelid	oid	pg_class.oid	分区（子表）的表标识符。

列	类型	参考	描述
parparentrule	oid	pg_partition_rule. paroid	与该分区的父表相关的规则的标识符。
parname	name		该分区的给定名称。
parisdefault	boolean		该分区是否为默认分区。
parruleord	smallint		对于范围分区表，该分区在分区层次的这个级别上的排名。
parrangestartincl	boolean		对于范围分区表，是否包括起始值。
parrangeendincl	boolean		对于范围分区表，是否包括结束值。
parrangestart	text		对于范围分区表，范围的开始值。
parrangeend	text		对于范围分区表，范围的结束值。
parrangeevery	text		对于范围分区表，EVERY子句的间隔值。
parlistvalues	text		对于列表分区表，指派给该分区的值列表。
parreloptions	text		一个描述特定分区存储特性的数组。

Parent topic: [系统目录定义](#)

pg_partition_templates

pg_partition_templates系统视图用来显示通过子分区模板创建的子分区。

Table 1. pg_catalog.pg_partition_templates

列	类型	参考	描述
schemaname	name		该分区表所在的schema的名称。
tablename	name		顶层父表的表名。
partitionname	name		子分区的名称（如果在ALTER TABLE命令中引用分区，用的就是这个名称）。如果该分区在创建时或者通过EVERY子句产生时没有给定一个名称则为NULL。
partitiontype	text		子分区的类型（范围或者列表）。
partitionlevel	smallint		该分区在层次中的级别。
partitionrank	bigint		对于范围分区，该分区相对于同级中其他分区的排名。
partitionposition	smallint		该子分区的规则顺序位置。
partitionlistvalues	text		对于列表分区，该子分区相关的列表值。

列	类型	参考	描述
partitionrangestart	text		对于范围分区，该子分区的开始值。
partitionstartinclusive	boolean		如果该子分区包含了起始值则为T，否则为F。
partitionrangeend	text		对于范围分区，该子分区的结束值。
partitionendinclusive	boolean		如果该子分区包含了结束值则为T，否则为F。
partitioneveryclause	text		该子分区的EVERY子句（间隔）。
partitionisdefault	boolean		如果这是一个默认子分区则为T，否则为F。
partitionboundary	text		该子分区的整个分区说明。

Parent topic: [系统目录定义](#)

pg_partitions

pg_partitions系统视图被用于显示分区表的结构。

Table 1. pg_catalog.pg_partitions

列	类型	参考	描述
schemaname	name		分区表所属schema的名称。
tablename	name		顶层父表的名称。
partitiontablename	name		分区表的关系名（直接访问分区时使用的表名）。
partitionname	name		分区的名称（在ALTER TABLE命令引用分区时，使用该名称）。如果在分区创建时或者由EVERY子句产生时没有给定名称则为NULL。
parentpartitiontablename	name		该分区上一层父表的关系名。
parentpartitionname	name		该分区上一层父表给定的名称。
partitiontype	text		分区的类型（范围或者列表）。
partitionlevel	smallint		该分区在层次中的级别。
partitionrank	bigint		对于范围分区，该分区相对于同级其他分区的排名。
partitionposition	smallint		该分区的规则顺序位置。

列	类型	参考	描述
partitionlistvalues	text		对于列表分区，与该分区相关的列表值。
partitionrangestart	text		对于范围分区，该子分区的开始值。
partitionstartinclusive	boolean		如果该子分区包含了起始值则为T，否则为F。
partitionrangeend	text		对于范围分区，该子分区的结束值。
partitionendinclusive	boolean		如果该子分区包含了结束值则为T，否则为F。
partitioneveryclause	text		该子分区的EVERY子句（间隔）。
partitionisdefault	boolean		如果这是一个默认子分区则为T，否则为F。
partitionboundary	text		该子分区的整个分区说明。

Parent topic: [系统目录定义](#)

pg_pltemplate

pg_pltemplate系统目录表存储了过程语言的模板信息。 一种语言的模板允许我

们在一个特定数据库中以简单的CREATE LANGUAGE命令创建该语言，而不需要指定实现细节。 和大部分系统目录不同，pg_pltemplate在MPP系统的所有数据库之间共享：在一个系统中只有一份pg_pltemplate拷贝，而不是每个数据库一份。 这使得在每个需要的数据库中都可以访问该信息。

目前任何命令都不能操纵过程语言模板。 要改变内建信息，超级用户必须使用普通的INSERT、DELETE或UPDATE命令修改该表。

Table 1. pg_catalog.pg_pltemplate

列	类型	参考	描述
tmplname	name		该模板适用的语言名称
tmpltrusted	boolean		如果该语言被认为是可信的则为真
tmplhandler	text		调用处理器函数的名字
tmplvalidator	text		验证器函数的名字，如果没有则为空
tmpllibrary	text		实现语言的共享库的路径
tmplacl	aclitem[]		模板的访问特权（并未实现）。

Parent topic: [系统目录定义](#)

pg_proc系统目录表存储关于函数（或过程）的信息，包括所有的内建函数以及由CREATE FUNCTION定义的函数。该表也包含了聚集和窗口函数以及普通函数的数据。如果proisagg为真，在pg_aggregate中应该有一个相匹配的行。

对于内置和动态加载的已编译函数，prosrc包含函数的C语言名称（链接符号）。对于所有其他当前已知的语言类型，prosrc包含函数的源文本。除了动态加载的C函数之外，probin是未使用的，它为此提供了包含该函数的共享库文件的名称。

Table 1. pg_catalog.pg_proc

列	类型	参考	描述
oid	oid		行标识符（隐藏属性;必须显式选择）
proname	name		函数的名字
pronamespace	oid	pg_namespace.oid	函数所属的名字空间的OID
proowner	oid	pg_authid.oid	函数的拥有者
prolang	oid	pg_language.oid	该函数的实现语言或调用接口
procost	float4		估计的执行代价（以cpu_operator_cost为单位），如果proretset为true，这是返回每行的代价
prorows	float4		估计的结果行数（如果不是proretset则为零）
provariadic	oid	pg_type.oid	可变数组参数的元素的数据类型，如果函数没有可变参数则为0

列	类型	参考	描述
protransform	regproc	pg_proc.oid	这个函数可以简化对此函数的调用
proisagg	boolean		函数是否为一个聚集函数
proiswindow	boolean		功能是否为一个窗口函数
prosecdef	boolean		函数是一个安全性定义器（例如，一个"setuid"函数）
proleakproof	boolean		该功能没有副作用。除了通过返回值之外，不会传达有关参数的信息。任何可能根据其参数的值抛出错误的函数都不是防漏的。
proisstrict	boolean		当任意调用参数为空时，函数是否会返回空值。在那种情况下函数实际上根本不会被调用。非“strict”函数必须准备好处理空值输入。
proretset	boolean		函数是否返回一个集合（即，指定数据类型的多个值）

列	类型	参考	描述
provolatile	char		说明函数的结果是仅依赖于它的输入参数，还是会被外部因素影响。值i表示“不变的”函数，它对于相同的输入总是输出相同的结果。值s表示“稳定的”函数，它的结果（对于固定输入）在一次扫描内不会变化。值v表示“不稳定的”函数，它的结果在任何时候都可能变化（使用具有副作用，结果可能在任何时刻发生改变）。
pronargs	int2		输入参数的个数
pronargdefaults	int2		具有默认值的参数个数
prorettype	oid	pg_type.oid	返回值的数据类型
proargtypes	oidvector	pg_type.oid	函数参数的数据类型的数组。这只包括输入参数（包括INOUT和VAR IADIC参数），因此也表现了函数的调用特征。

列	类型	参考	描述
proallargtypes	oid[]	pg_type.oid	函数参数的数据类型的数组。这包括所有参数（含OUT和INOUT参数）。不过，如果所有参数都是IN参数，这个属性将为空。注意下标是从1开始，然而由于历史原因proargtypes的下标是从0开始。
proargmodes	char[]		函数参数的模式的数组：i表示IN参数，o表示OUT参数，b表示INOUT参数，v表示VARIADIC参数。如果所有的参数都是IN参数，这个属性为空。注意这里的下标对应着proallargtypes而不是proargtypes中的位置。
proargnames	text[]		函数参数的名字的数组。没有名字的参数在数组中设置为空字符串。如果没有一个参数有名字，这个属性为空。注意这里的下标对应着proallargtypes而不是proargtypes中的位置。

列	类型	参考	描述
proargdefaults	pg_node_tree		表达式树（在nodeToString()表示中）用于默认参数值。 这是一个带有pronargdefaults元素的列表，对应于最后N个输入参数（即最后N个proargtypes位置）。 如果没有参数具有默认值，则此字段将为null。
prosrc	text		这个域告诉函数处理者如何调用该函数。 它可能是针对解释型语言的实际源码、一个符号链接、一个文件名或任何其他东西，这取决于实现语言/调用规范。
probin	text		关于如何调用函数的附加信息。 同样，其解释是与语言相关的。
proconfig	text[]		函数的运行时配置变量的本地设置。
proacl	aclitem[]		GRANT/REVOKE给出的函数访问权限
prodataaccess	char		提供有关函数中包含的SQL语句类型的提示： n - 不包含SQL，c - 包含SQL，r - 包含读取数据的SQL，m - 包含修改数据的SQL

列	类型	参考	描述
proexeclocation	char		函数在调用时执行的位置：m - 仅限master，a - 任何段实例，s - 所有段实例。

Parent topic: [系统目录定义](#)

pg_resgroup

Note: 仅当基于资源组的资源管理处于活动状态时，pg_resgroup系统目录表才有效。

pg_resgroup系统目录表包含有关MPP数据库资源组的信息，这些资源组用于管理并发语句，CPU和内存资源。 此表在pg_global表空间中定义，在系统中的所有数据库之间全局共享。

Table 1. pg_catalog.pg_resgroup

列	类型	参考	描述
rsgname	name		资源组的名称。
parent	oid		未使用; 留作将来使用。

Parent topic: [系统目录定义](#)

pg_resgroupcapability

Note: 仅当基于资源组的资源管理处于活动状态时，pg_resgroupcapability系统目录表才有效。

pg_resgroupcapability系统目录表包含有关已定义的MPP数据库资源组的功能和限制的信息。 您可以按资源组对象ID将此表连接到[pg_resgroup](#)表。

pg_global表空间中定义的pg_resgroupcapability表在系统中的所有数据库之间全局共享。

Table 1. pg_catalog.pg_resgroupcapability

列	类型	参考	描述
resgroupid	oid	pg_resgroup.oid	关联资源组的对象ID。
reslimittype	smallint		资源组限制类型： 0 - 未知 1 - 并发度 2 - CPU 3 - 内存 4 - 内存共享配额 5 - 内存溢出比例 6 - 内存审查 7 - CPU集合

列	类型	参考	描述
value	opaque type		为此记录中引用的资源限制设置的特定值。此值具有固定类型文本，并将根据引用的限制转换为不同的数据类型。
proposed	opaque type		如果您更改了资源限制并且无法立即更新限制，此记录中引用的限制的proposed值。否则，proposed反映当前设定的value。

Parent topic: [系统目录定义](#)

pg_resourcetype

pg_resourcetype系统目录表包含有关可以分配给MPP数据库资源队列的扩展属性的信息。 每行详细说明属性和固有质量，例如其默认设置，是否需要，以及禁用它的值（如果允许）。

此表仅在master服务器上填充。 此表在pg_global表空间中定义，这意味着它在系统中的所有数据库中全局共享。

Table 1. pg_catalog.pg_resourcetype

列	类型	参考	描述
restypid	smallint		资源类型ID。
resname	name		资源类型名称。
resrequired	boolean		资源类型是否是有效资源队列所必需的。
reshasdefault	boolean		资源类型是否具有默认值。 如果为true，则在reshasdefaultsetting中指定默认值。
rescandisable	boolean		是否可以删除或禁用该类型。 如果为true，则在resdisabledsetting中指定默认值。
resdefaultsetting	text		资源类型的默认设置（如果适用）。
resdisabledsetting	text		禁用此资源类型的值（如果允许）。

Parent topic: [系统目录定义](#)

pg_resqueue

Note: 仅当基于资源队列的资源管理处于活动状态时，pg_resqueue系统目录表才有效。

pg_resqueue系统目录表包含有关MPP数据库资源队列的信息，这些队列用于资源管理功能。此表仅在master服务器上填充。此表在pg_global表空间中定义，这意味着它在系统中的所有数据库中全局共享。

Table 1. pg_catalog.pg_resqueue

列	类型	参考	描述
rsqname	name		资源队列的名称。
rsqcountlimit	real		资源队列的活跃查询阈值。
rsqcostlimit	real		资源队列的查询开销阈值。
rsqovercommit	boolean		允许超出成本阈值的查询在系统空闲时运行。
rsqignorecostlimit	real		被视为“小查询”的查询成本限制。成本低于此限制的查询将不会排队并立即运行。

Parent topic: [系统目录定义](#)

pg_resqueue_attributes

Note: 仅当基于资源队列的资源管理处于活动状态时，pg_resqueue_attributes视图才有效。

pg_resqueue_attributes视图允许管理员查看为资源队列设置的属性，例如其活

动语句限制，查询成本限制和优先级。

Table 1. pg_catalog.pg_resqueue_attributes

列	类型	参考	描述
rsqname	name	pg_resqueue.rs qname	资源队列的名称。
resname	text		资源队列属性的名称。
resetting	text		资源队列属性的当前值。 。
restypid	integer		系统分配的资源类型ID。 。

Parent topic: [系统目录定义](#)

pg_resqueuecapability

Note: 仅当基于资源队列的资源管理处于活动状态时，pg_resqueuecapability系统目录表才有效。

pg_resqueuecapability系统目录表包含有关现有MPP数据库资源队列的扩展属性或功能的信息。 只有已分配了扩展功能的资源队列（例如优先级设置）才会记录在此表中。 此表通过资源队列对象ID连接到[pg_resqueue](#)表，并通过资源类型ID（restypid）连接到[pg_resourcetype](#)表。

此表仅在master服务器上填充。 此表在pg_global表空间中定义，这意味着它在系统中的所有数据库中全局共享。

Table 1. pg_catalog.pg_resqueuecapability

列	类型	参考	描述
rsqueueid	oid	pg_resqueue.oid	关联资源队列的对象ID。
restypid	smallint	pg_resourcetype.restypid	资源类型，派生自 pg_resqueuecapability 系统表。
resetting	opaque type		为此记录中引用的功能设置的特定值。根据实际资源类型，此值可能具有不同的数据类型。

Parent topic: [系统目录定义](#)

pg_rewrite

pg_rewrite系统目录表存储表和视图的重写规则。 如果一个表在这个目录中有任何规则，其pg_class.relhasrules必须为真。

Table 1. pg_catalog.pg_rewrite

列	类型	参考	描述
rulename	name		规则名称。

列	类型	参考	描述
ev_class	oid	pg_class.oid	使用该规则的表。
ev_attr	int2		使用该规则的列（当前总是0，表示在整个表上使用）。
ev_type	char		使用该规则的事件类型： 1 = SELECT，2 = UPDATE，3 = INSERT，4 = DELETE
is_instead	boolean		如果规则是一个INSTEAD规则，则为真。
ev_qual	text		规则条件的表达式树（按照nodeToString()的表现形式）。
ev_action	text		规则动作的查询树（按照nodeToString()的表现形式）。

Parent topic: [系统目录定义](#)

pg_roles

pg_roles视图提供关提供了关于数据库角色的信息。这是[pg_authid](#)的一个公共可读视图，它隐去了口令域。此视图显示了低层表的OID列，因为需要它来和其他目录做连接。

Table 1. pg_catalog.pg_roles

列	类型	参考	描述
rolname	name		角色名
rolsuper	bool		角色是否具有超级用户权限
rolinherit	bool		如果此角色是另一个角色的成员，角色是否能自动继承另一个角色的权限
rolcreatorole	bool		角色能否创建更多角色
rolcreatedb	bool		角色能否创建数据库
rolcatupdate	bool		角色能够更新直接更新系统目录（除非该列设置为真，否则超级用户也不能执行该操作）
rolcanlogin	bool		角色是否能登录？即此角色能否被作为初始会话授权标识符
rolconnlimit	int4		对于一个可登录的角色，这里设置角色可以发起的最大并发连接数。-1表示无限制。
rolpassword	text		不是口令（看起来是*****)
rolvaliduntil	timestampz		口令失效时间（只用于口令认证），如果永不失效则为NULL

列	类型	参考	描述
rolconfig	text[]		运行时配置变量的角色特定默认值
rolresqueue	oid	pg_resqueue.oid	该角色指定的资源队列的对象ID
oid	oid	pg_authid.oid	角色的ID
rolcreaterextgpfd	bool		角色能够创建使用gpfdist协议的可读的外部表
rolcreaterexthttp	bool		角色能够创建使用http协议的可读的外部表
rolcreatewextgpfd	bool		角色能否创建使用了gpfdist协议的可写外部表
rolresgroup	oid	pg_resgroup.oid	分配了此角色的资源组的对象ID

Parent topic: [系统目录定义](#)

pg_shdepend

pg_shdepend系统目录表记录数据库对象和共享对象（例如角色）之间的依赖关系。 这些信息使得MPP数据库可以确保对象在被删除时没有被其他对象引用。 另见[pg_depend](#)，它对单个数据库中对象之间的依赖提供了相似的功能。与大部分其他系统目录不同，pg_shdepend在MPP系统的所有数据库之间共享：在每一个系统中只有一份pg_shdepend拷贝，而不是每个数据库一份。

在所有情况下，一个pg_shdepend项表明被引用对象不能在删除其依赖对象的情况下被删除。但是，其中也有多种依赖类型，由deptype标识：

- **SHARED_DEPENDENCY_OWNER (o)** — 被引用对象（必须是一个角色）是依赖对象的拥有者。
- **SHARED_DEPENDENCY_ACL (a)** — 被引用对象（必须是一个角色）在依赖对象的ACL（访问控制列表）中被提到。
- **SHARED_DEPENDENCY_PIN (p)** — 没有依赖对象；这种类型的项是系统本身依赖被引用对象的信号，因此对象绝不能被删除。此类型的项仅通过系统初始化创建，依赖对象列包含0。

Table 1. pg_catalog.pg_shdepend

列	类型	参考	描述
dbid	oid	pg_databases.oid	依赖对象所在的数据库OID，如果是一个共享对象则值为0。
classid	oid	pg_class.oid	依赖对象所在的系统目录的OID。
objid	oid	any OID column	特定依赖对象的OID。
objsubid	int4		对于一个表列，这将是列编号。对于所有其他对象类型，该列值为0。
refclassid	oid	pg_class.oid	被引用对象所在的系统目录的OID（必须是一个共享的目录）。
refobjid	oid	any OID column	被引用对象的OID。

列	类型	参考	描述
refobjsubid	int4		对于一个表列，这将是被引用列的列编号。对于所有其他对象类型，该列值为0。
deptype	char		一个定义该依赖关系的特定语义的代码。

Parent topic: [系统目录定义](#)

pg_shdescription

pg_shdescription系统目录表存储共享数据库对象的可选描述（注释）。描述可以通过COMMENT命令操作，并且可以使用psql的\d元命令来查看。另见[pg_description](#)，它对单个数据库中对象的描述提供了相似的功能。与大部分其他系统目录不同，pg_shdescription在MPP系统的所有数据库之间共享：在每一个系统中只有一份pg_shdescription拷贝，而不是每个数据库一份。

Table 1. pg_catalog.pg_shdescription

列	类型	参考	描述
objoid	oid	any OID column	该描述所属的对象的OID。
classoid	oid	pg_class.oid	该对象所在系统目录的OID。

列	类型	参考	描述
description	text		作为该对象描述的任意文本。

Parent topic: [系统目录定义](#)

pg_stat_activity

视图pg_stat_activity每行显示一个服务器进程同时详细描述与之关联的用户会话和查询。 这些列报告当前查询上可用的数据，除非参数stats_command_string被关闭。 此外，只有在检查视图的用户是超级用户或者是正在报告的进程的拥有者时，这些列才可见。

列current_query中存储的查询文本字符串的最大长度可以通过服务器配置参数track_activity_query_size来控制。

Table 1. pg_catalog.pg_stat_activity

列	类型	参考	描述
datid	oid	pg_database.oid	数据库OID
datname	name		数据库名称
pid	integer		服务进程的进程ID
sess_id	integer		会话ID
usesysid	oid	pg_authid.oid	登录此后端的用户的OID

列	类型	参考	描述
username	name		登录到此后端的用户的名称
application_name	text		连接到此后端的应用程序的名称
client_addr	inet		连接到此后端的客户端的IP地址。 如果此字段为空，则表示客户端通过服务器计算机上的Unix套接字连接，或者这是内部进程（如autovacuum）。
client_hostname	text		客户端的主机名，由client_addr的反向DNS查找报告。 对于IP连接，此字段仅为非null，并且仅在启用log_hostname时才为空。
client_port	integer		客户端用于与此后端通信的TCP端口号，如果使用Unix套接字，则为-1
backend_start	timestampz		后端进程启动时间
xact_start	timestampz		事务开始时间
query_start	timestampz		查询开始执行时间
state_change	timestampz		状态最后一次改变的时间
waiting	boolean		如果等待一个锁为True，否则为false

列	类型	参考	描述
state	text		此后端的当前整体状态。可能的值是： • active: 后端正在执行查询。 • idle: 后端正在等待新的客户端命令。 • idle in transaction: 后端处于事务中，但当前未执行查询。 • idle in transaction (aborted): 此状态类似于事务中的空闲，除了事务中的一个语句导致错误。 • fastpath function call: 后端正在执行快速路径功能。 • disabled: 如果在此后端禁用track_activities，则报告此状态。
query	text		此后端的最新查询的文本。如果状态为活跃，则此字段显示当前正在执行的查询。在所有其他状态中，它显示最后执行的查询。

列	类型	参考	描述
waiting_reason	text		服务器进程正在等待的原因。 值可以是： lock， replication或res group
rsgid	oid	pg_resgroup.oid	资源组OID
rsgname	text	pg_resgroup.rsgname	资源组名称
rsgqueueduration	interval		对于排队查询，查询排队的总时间。

Parent topic: [系统目录定义](#)

pg_stat_last_operation

pg_stat_last_operation表包含关于数据库对象（表、视图等）的元数据跟踪信息。

Table 1. pg_catalog.pg_stat_last_operation

列	类型	参考	描述
classid	oid	pg_class.oid	包含对象的系统目录的OID。

列	类型	参考	描述
objid	oid	any OID column	对象在其系统目录内的对象OID。
staactionname	name		在一个对象上采取的动作。
stasysid	oid	pg_authid.oid	pg_authid.oid的外键。
stausename	name		在该对象上执行操作的角色的名称。
stasubtype	text		被执行操作的对象的类型或者被执行操作的子类。
statime	timestamp with timezone		操作的时间戳。 这和写到MPP数据库服务器日志文件的时间戳是相同的，以便在日志中查询更多关于操作细节的信息。

Parent topic: [系统目录定义](#)

pg_stat_last_shoperation

pg_stat_last_shoperation表包含全局对象（角色，表空间等）的元数据跟踪信息。

Table 1. pg_catalog.pg_stat_last_shoperation

列	类型	参考	描述
classid	oid	pg_class.oid	包含该对象的系统目录的OID。
objid	oid	any OID column	系统目录中对象的OID。
staactionname	name		对该对象采取的操作。
stasysid	oid		
stausename	name		对此对象执行操作的角色的名称。
stasubtype	text		操作对象的类型或执行的操作的子类。
statime	timestamp with timezone		操作的时间戳。这与写入MPP数据库服务器日志文件的时间戳相同，以防您需要在日志中查找有关操作的更多详细信息。

Parent topic: [系统目录定义](#)

pg_stat_operations

视图pg_stat_operations显示了关于最后一个执行在数据库对象（例如，表、索引）上的操作。

引、视图或者数据库) 上或者全局对象 (例如角色) 的操作的细节信息。

Table 1. pg_catalog.pg_stat_operations

列	类型	参考	描述
classname	text		pg_catalog schema中 存储有关此对象的记录 的系统表的名称 (pg_cl ass=关系, pg_database =数据库, pg_namespace=schem as, pg_authid=角色)
objname	name		对象的名称。
objid	oid		对象的OID。
schemaname	name		对象所在的schema的 名称。
usestatus	text		对对象执行最后一次操 作的角色的状态 (CUR RENT =系统中当前活 动的角色, DROPPED =系统中不再存在的角 色, CHANGED =系统 中存在的角色名称, 但 自上次操作以来已更改) 。
username	name		对此对象执行操作的角 色的名称。

列	类型	参考	描述
actionname	name		对该对象采取的操作。
subtype	text		操作对象的类型或执行的操作的子类。
statime	timestamptz		操作的时间戳。这与写入MPP数据库服务器日志文件的时间戳相同，以防您需要在日志中查找有关操作的更多详细信息。

Parent topic: [系统目录定义](#)

pg_stat_partition_operations

pg_stat_partition_operations视图显示了执行在一个分区表上的上一个操作的细节信息。

Table 1. pg_catalog.pg_stat_partition_operations

列	类型	参考	描述
classname	text		pg_catalog schema中系统表的名称，其中存储有关此对象的记录（对于表和分区，始终为pg_class）。

列	类型	参考	描述
objname	name		对象的名称。
objid	oid		对象的OID。
schemaname	name		对象所在的schema的名称。
usestatus	text		对对象执行最后一次操作的角色的状态（CURRENT =系统中当前活动的角色，DROPPED =系统中不再存在的角色，CHANGED =系统中存在的角色名称，但自上次操作以来已更改）。
username	name		对此对象执行操作的角色的名称。
actionname	name		对该对象采取的操作。
subtype	text		操作对象的类型或执行的操作的子类。
statime	timestampz		操作的时间戳。这与写入MPP数据库服务器日志文件的时间戳相同，以防您需要在日志中查找有关操作的更多详细信息。
partitionlevel	smallint		层次结构中此分区的级别。
parenttablename	name		父表的关系名称从此分区向上一级。

列	类型	参考	描述
parentschemaname	name		父表所在的schema的名称。
parent_relid	oid		父表的OID从此分区向上一级。

Parent topic: [系统目录定义](#)

pg_stat_replication

pg_stat_replication视图包含用于MPP数据库主镜像的walsender进程的元数据。

Table 1. pg_catalog.pg_stat_replication

列	类型	参考	描述
procpid	integer		WAL-sender后台进程的进程ID。
usesysid	integer		运行WAL-sender后台进程的用户系统ID。
username	name		运行WAL-sender后台进程的用户名。
application_name	oid		客户端应用名。
client_addr	name		客户端IP地址。
client_port	integer		客户端端口号。

列	类型	参考	描述
backend_start	timestamp		操作开始的时间戳。
state	text		WAL发送状态。可取值有： startup backup catchup streaming
sent_location	text		WAL-sender已发送的xlog记录位置。
write_location	text		WAL-receiver的xlog记录写位置。
flush_location	text		WAL-receiver的xlog记录刷入位置。
replay_location	text		Standby xlog记录的重放位置。
sync_priority	text		优先级，值为1。
sync_state	text		WAL-sender的同步状态。该值为sync。

Parent topic: [系统目录定义](#)

pg_stat_statements

pg_stat_statements模块提供一种跟踪执行统计服务器执行的所有SQL语句的手段。

该模块收集的统计通过一个名为pg_stat_statements的系统视图使其可用。 这个模块为每个不同的查询、数据库ID和用户ID（取决于该模块可以追踪的不同语句的最大值） 包含一行。视图的字段显示在下表中。

表 1. pg_catalog.pg_stat_statements

名称	类型	引用	描述
userid	oid	pg_authid.oid	执行该语句的用户的OID
dbid	oid	pg_database.oid	执行该语句的数据库的OID
queryid	bigint		内部哈希代码，从声明的分析树中计算而来
query	text		有代表性的语句的文本
calls	bigint		执行的次数
total_time	double precision		该语句花费的总时间，以毫秒计
rows	bigint		该语句恢复或影响的行的总数
shared_blks_hit	bigint		该语句命中的共享块缓存的总数

名称	类型	引用	描述
shared_blks_read	bigint		该语句读取的共享块的总数
shared_blks_dirtied	bigint		该语句弄脏的共享块的总数
shared_blks_written	bigint		该语句写入的共享块的总数
local_blks_hit	bigint		该语句命中的本地块缓存的总数
local_blks_read	bigint		该语句读取的本地块的总数
local_blks_dirtied	bigint		该语句弄脏的本地块的总数
local_blks_written	bigint		该语句写入的本地块的总数
temp_blks_read	bigint		该语句读取的临时块的总数
temp_blks_written	bigint		该语句写入的临时块的总数
blk_read_time	double precision		该语句读取块花费的总时间，以毫秒计（如果启用了track_io_timing，否则为0）
blk_write_time	double precision		该语句写入块花费的总时间，以毫秒计（如果启用了track_io_timing，否则为0）

这个视图和函数pg_stat_statements_reset和 pg_stat_statements，只有在通过

安装 `pg_stat_statements` 扩展特别安装到的数据库中可用。不过，当 `pg_stat_statements` 模块加载到服务器中时，统计跟踪该服务器中的所有数据库，不管该视图是否存在。

为了安全起见，不允许非超级用户查看其它用户执行的查询的SQL文本或 `queryid`。不过，如果视图已经安装到他们的数据库中，那么他们可以看到统计。

可计划的查询（也就是，`SELECT`, `INSERT`, `UPDATE`, 和 `DELETE`）组合成为一个 `pg_stat_statements`，当它们根据一个内部哈希计算有相同的查询结构时。典型的，如果两个查询语义上相等，除了查询中字面常量的值之外，我们认为这两个查询相同。工具命令（也就是，所有其他命令）是直接基于它们的文本查询字符串比较的。

当一个常量的值为了匹配其他查询而忽略时，该常量在 `pg_stat_statements` 的显示中被 `?` 替代。查询文本的剩余部分是第一个查询特定 `queryid` 散列值与 `pg_stat_statements` 相关条目。

在一些情况下，带有明显不同文本的查询可能合并到一个 `pg_stat_statements`。通常这只在语义相等的查询上发生，但是有很小的可能哈希冲突导致不相关的查询被合并到一个条目。（不过，这对于属于不同用户或数据库的查询来说是不会发生的。）

因为哈希值是基于分析查询的表示法之后来计算的，相反的也是可能的：带有相同文本的查询可能表现为单独的条目，如果它们因为一个因素的结果有不同的含义，比如不同的 `search_path` 设置。

`pg_stat_statements` 的消耗者可能希望将 `queryid`（可能和 `dbid` 和 `userid` 结合使用）用作每个条目更稳定可靠的标识符而不是它的查询文本。但是，重要的是要理解 `queryid` 哈希值的稳定性保证是有限的。因为标识符来源于 `post-parse-analysis` 树，它的值是在其他事物之上，出现在这个表示法中的内部对象标识符的一个函数。这有一些违背常理。例如，如果在两个查询的执行期间它们引用的表删除之后又重新创建了，那么 `pg_stat_statements` 将认为这两个看起来相同的查询是不同的。哈希过程对机器架构和平台其他方面的不同也是敏感的。另

外，对queryid跨PostgreSQL的主要版本还是稳定的这种假设是不安全的。

根据经验，只要底层服务器版本和目录元数据细节保持完全相同，queryid的值就可以假设为稳定和可比较的。参与到基于物理WAL重放复制中的两个服务器，对于相同的查询可以预计为有不同的queryid值。不过，逻辑复制模式并不保证复制品在所有相关细节上都相同，所以queryid对于累加一系列逻辑复制的消耗来说不是一个有用的标识符。如果有怀疑，建议直接做测试。

函数

`pg_stat_statements_reset()` returns void

`pg_stat_statements_reset`抛弃所有`pg_stat_statements`到目前为止收集的统计。缺省的，这个函数只能被超级用户执行。

`pg_stat_statements(showtext boolean)` returns setof record

`pg_stat_statements`视图的定义是为了一个也叫做 `pg_stat_statements` 的函数。客户端可以直接调用 `pg_stat_statements` 函数，并且通过声明 `showtext := false` 省略查询文本（也就是，对应于视图的 `query` 字段的OUT参数将返回空）。这个特性是为了支持想要避免重复检索不确定查询文本长度的总开销的外部工具。这样的工具可以为每个条目本身缓存首先看到的查询文本，所有 `pg_stat_statements` 本身也是这样做的，然后只在需要时检索查询文本。因为服务器将查询文本存储在一个文件中，所以这个方法可以减少重复检查 `pg_stat_statements` 数据的物理I/O。

配置参数

`pg_stat_statements.enable` (boolean)

`pg_stat_statements.enable`控制是否开启语句追踪功能。缺省值是不开启，这个参数可在配置文件中设置。

`pg_stat_statements.max` (integer)

`pg_stat_statements.max`是该模块追踪语句的最大值（也就是，`pg_stat_statements`视图中的最大行数）。如果观察了比这更多的不同的语句，则会抛弃执行最少的语句的信息。缺省值是5000。这个参数只能在服务器

启动时设置。

`pg_stat_statements.track` (enum)

`pg_stat_statements.track`控制哪个语句可以被该模块计数。声明`top`来跟踪顶级的语句（直接通过客户端发出的语句）。`all`也跟踪嵌套的语句（比如包含在函数中的语句），或`none`禁用语句状态收集。缺省值是`top`。只有超级用户可以更改这个设置。

`pg_stat_statements.track_utility` (boolean)

`pg_stat_statements.track_utility`控制该模块是否追踪工具命令。工具命令是除了`SELECT`, `INSERT`, `UPDATE` 和 `DELETE`的所有命令。缺省值是`on`。只有超级用户可以更改这个设置。

`pg_stat_statements.save` (boolean)

`pg_stat_statements.save`指定在服务器关闭时是否保存语句状态。如果是`off`，那么在服务器关闭时不保存状态，在服务器启动时也不重新加载。缺省值是`on`。这个参数只可以在`postgresql.conf` 文件中或者服务器命令行中设置。

该模块需要额外的共享内存与 `pg_stat_statements.max`成比例。请注意，这个内存在该模块加载时被消耗，即使`pg_stat_statements.track` 设置为`none`。

上级主题：[系统目录定义](#)

pg_statistic

`pg_statistic`系统目录表存储有关数据库内容的统计数据。其中的项由`ANALYZE`创建，查询规划器会使用这些数据来进行查询规划。每个已经被分析的表列都有一项。注意所有的统计数据天然就是近似的，即使它刚刚被更新。

`pg_statistic`还存储有关索引表达式值的统计数据。这些被描述为它们是实际的

数据列;特别是，starelid引用了索引。但是，没有为普通的非表达式索引列创建条目，因为它对于基础表列的条目是多余的。目前，索引表达式的条目始终具有stainherit = false。

当stainherit = false时，每个表格列通常有一个条目已被分析。如果表具有继承子项，则MPP数据库会创建第二个条目，其中stainherit = true。此行表示列在继承树上的统计信息，例如，您将使用SELECT column FROM table*看到的数据的统计信息，而stainherit = false行表示SELECT column FROM ONLY table表的结果。

由于不同类型的统计数据可能适用于不同类型的数据，因此pg_statistic不会非常假设它存储的是哪种统计数据。在pg_statistic中只给出了非常普通的统计信息（例如nullness）。其他所有内容都存储在插槽中，插槽是相关列的组，其内容由插槽列之一中的代码编号标识。

pg_statistic不应该被公众读取，因为即使关于表的内容的统计信息也可能被认为是敏感的（例如：工资列的最小值和最大值）。pg_stats是pg_statistic上的公共可读视图，它只公开有关当前用户可读的表的信息。

Table 1. pg_catalog.pg_statistic

列	类型	参考	描述
starelid	oid	pg_class.oid	被描述列所属的表或索引。
staattnum	int2	pg_attribute.attnum	被描述列的编号。
stainherit	bool		如果为true，则统计信息包括继承子列，而不仅仅是指定关系中的值。
stanullfrac	float4		列项为空的比例。

列	类型	参考	描述
stawidth	int4		非空项的平均存储宽度，以字节计。
stadistinct	float4		列中不同的非空数据值的数量。大于零的值是不同值的实际数量。小于零的值是表中行数的乘数的负值（例如，其中值平均显示两次的列可以由stadistinct = -0.5表示）。零值表示不同值的数量未知。
stakindN	int2		一个代码，它表示存储在该pg_statistic行中第N个“槽位”的统计类型。
staopN	oid	pg_operator.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，一个柱状图槽位会用<操作符，该操作符定义了该数据的排序顺序。
stanumbersN	float4[]		第N个“槽位”的类型的数值类型统计，如果该槽位不涉及数值类型则为NULL。

列	类型	参考	描述
stavaluesN	anyarray		第N个“槽位”的类型的列值，如果该槽位类型不存储任何数据值则为NULL。 每个数组的元素值实际上都是指定列的数据类型，因此，除了把这些列的类型定义成anyarray之外别无他法。

Parent topic: [系统目录定义](#)

pg_stat_resqueues

Note: 仅当基于资源队列的资源管理处于活动状态时，pg_stat_resqueues视图才有效。

pg_stat_resqueues视图允许管理员查看有关资源队列工作负载的指标。 要允许为此视图收集统计信息，必须在MPP数据库master实例上启用stats_queue_level服务器配置参数。 启用这些度量标准的收集会导致性能损失很小，因为通过资源队列提交的每个语句都必须记录在系统目录表中。

Table 1. pg_catalog.pg_stat_resqueues

列	类型	参考	描述
queueoid	oid		资源队列的OID。
queuename	name		资源队列的名称。

列	类型	参考	描述
n_queries_exec	bigint		已提交的要从该资源队列中执行的查询数目。
n_queries_wait	bigint		已提交到该资源队列中，在执行之前必须进行等待的查询数目。
elapsed_exec	bigint		通过该资源队列提交的语句的总执行时间。
elapsed_wait	bigint		通过该资源队列提交的语句，在执行之前必须进行等待的总时间。

Parent topic: [系统目录定义](#)

pg_tablespace

pg_tablespace系统目录表存储关于可用表空间的信息。 表可以被放置在特定表空间中以实现磁盘布局的管理。 与大部分其他系统目录不同，pg_tablespace在MPP系统的所有数据库之间共享：在每一个系统中只有一份pg_tablespace的拷贝，而不是每个数据库一份。

Table 1. pg_catalog.pg_tablespace

列	类型	参考	描述
spcname	name		表空间名。

列	类型	参考	描述
spcowner	oid	pg_authid.oid	表空间的拥有者，通常是创建它的用户。
spcacl	aclitem[]		表空间访问权限。
spcoptions	text[]		表空间contentID位置。

Parent topic: [系统目录定义](#)

pg_trigger

pg_trigger系统目录表存储表上的触发器。

Note: MPP数据库不支持触发器。

Table 1. pg_catalog.pg_trigger

列	类型	参考	描述
tgrelid	oid	<i>pg_class.oid</i> 请注意，MPP数据库不强制引用完整性。	触发器所在的表。
tgname	name		触发器名（同一个表的触发器名必须唯一）。

列	类型	参考	描述
tgfoid	oid	<i>pg_proc.oid</i> 请注意，MP P数据库不 强制引用完 整性。	被触发器调用的函数。
tgtype	int2		触发器触发条件的位掩 码。
tgenabled	boolean		启用触发器则为True。
tgisinternal	boolean		如果触发器是内部生成 的，则为True（通常， 强制执行tgconstraint标 识的约束）。
tgconstrrelid	oid	<i>pg_class.oi d</i> 请注意，MP P数据库不 强制引用完 整性。	被一个参照完整性约束 引用的表。
tgdeferrable	boolean		如果可延迟则为True。
tginitdeferred	boolean		如果初始可延迟则为Tr ue。
tgargs	int2		传递给触发器函数的参 数字符串个数。
tgattr	int2vector		当前没有使用。

列	类型	参考	描述
tgargs	bytea		传递给触发器的参数字符串，每一个都以NUL结尾。

Parent topic: [系统目录定义](#)

pg_type

pg_type系统目录表存储有关数据类型的信息。基类（标量类型）由CREATE TYPE创建。而域由CREATE DOMAIN创建。数据库中的每一个表都会有一个自动创建的组合类型，用于表示表的行结构。也可以使用CREATE TYPE AS创建组合类型。

Table 1. pg_catalog.pg_type

列	类型	参考	描述
oid	oid		行标识符（隐藏属性;必须显示选择）
typname	name		数据类型名称
typnamespace	oid	pg_namespace.oid	包含此类型的命名空间的OID
typowner	oid	pg_authid.oid	类型的拥有者

列	类型	参考	描述
typlen	int2		对于一个固定尺寸的类型，typlen是该类型内部表示的字节数。 对于一个变长类型，typlen为负值。 -1表示一个“varlena”类型（具有长度字），-2表示一个以null结尾的C字符串。
typbyval	boolean		决定内部例程传递这个类型的数值时是通过传值还是传引用方式。 如果typlen不是1、2或4（或者在Datum为8字节的机器上为8），typlen最好是假。 变长类型总是传引用。 注意即使长度允许传值，typbyval也可以为假。
typtype	char		b表示基类，c表示组合类型，d表示域，e表示枚举类型，p表示伪类型，或r表示范围类型。 另见typrelid和typbasetype。
typcategory	char		解析器使用的数据类型的任意分类，以确定应首选哪些隐式转换。 请参阅 类别代码 。
typispreferred	boolean		如果类型是其typcategory中的首选转换目标，则为True

列	类型	参考	描述
typisdefined	boolean		如果定义了类型，则为True;如果是尚未定义的类型占位符条目，则为false。 如果为false，则可以依赖除类型名称，名称空间和OID之外的任何内容。
typdelim	char		解析数组输入时分隔此类型的两个值的字符。 请注意，分隔符与数组元素数据类型相关联，而不是与数组数据类型相关联。
typrelid	oid	pg_class.oid	如果这是复合类型（请参阅type），则此列指向定义相应表的pg_class条目。（对于独立的复合类型，pg_class条目实际上并不代表表，但无论如何都需要将类型的pg_attribute条目链接到。）非复合类型为零。

列	类型	参考	描述
typelem	oid	pg_type.oid	如果不为0，则它标识pg_type中的另一行。然后，当前类型可以像数组一样下标，产生类型为typelem的值。“true”数组类型是可变长度（typlen = -1），但某些固定长度（typlen > 0）类型也具有非零typelem，例如name和point。如果固定长度类型有typelem，则其内部表示必须是typelem数据类型的某些值，而没有其他数据。可变长度数组类型具有由数组子例程定义的头。
typarray	oid	pg_type.oid	如果不为0，则标识pg_type中的另一行，这是具有此类型作为其元素的“true”数组类型。使用pg_type.typarray查找与特定类型关联的数组类型。
typinput	regproc	pg_proc.oid	输入转换函数（文本格式）
typoutput	regproc	pg_proc.oid	输出转换函数（文本格式）
typreceive	regproc	pg_proc.oid	输入转换函数（二进制格式），如果没有则为0
typsend	regproc	pg_proc.oid	输出转换函数（二进制格式），如果没有则为0
typmodin	regproc	pg_proc.oid	键入修饰符输入函数，如果类型不支持修饰符，则为0

列	类型	参考	描述
typmodout	regpro	pg_proc.oid	键入修饰符输出函数，或0以使用标准格式
typanalyze	regpro	pg_proc.oid	自定义ANALYZE函数，0表示使用标准函数
typalign	char		存储此类型的值时所需的对齐方式。它适用于磁盘上的存储以及MPP数据库中值的大多数表示。当连续存储多个值时，例如在磁盘上的完整行的表示中，在此类型的数据之前插入填充，以便它在指定的边界上开始。对齐参考是序列中第一个数据的开头。可能的值是： c = char对齐，即不需要对齐。 s = short对齐（在大部分机器上为2字节）。 i = int对齐（在大部分机器上为4字节）。 d = double对齐（在很多机器上为8字节，但绝不是全部）。

列	类型	参考	描述
typstorage	char		对于varlena类型（具有typlen = -1的那些），告知该类型是否准备好toasting以及该类型的属性的默认策略应该是什么。可能的值是： p: 值必须明文存储。 e: 值可以存储在辅助关系中（如果关系有一个，请参阅pg_class.reltoastrelid）。 m: 值可以压缩内联存储。 x: 值可以内联压缩存储或存储在二级存储中。 请注意，m列也可以移出到辅助存储，但仅作为最后的手段（首先移动e和x列）。
typnotnull	boolean		表示类型的非空约束。仅用于域。
typbasetype	oid	pg_type.oid	标识域所基于的类型。如果此类型不是域，则为零。
typtypmod	int4		域使用typtypmod来记录要应用于其基类型的typmod（如果基类型不使用typmod，则为-1）。-1如果此类型不是域。

列	类型	参考	描述
typndims	int4		域上数组的数组维数（如果typbasetype是数组类型）。除了数组类型的域以外的类型为零。
typcollation	oid	pg_collation.oid	指定类型的排序规则。如果类型不支持排序规则，则为零。对于支持排序规则的基本类型，该值为DEFAULT_COLLATION_OID。如果为域指定了一个域，则通过可折叠类型的域可以具有一些其他排序规则OID。
typdefaultbin	pg_node_tree		如果不为null，则为该类型的默认表达式的nodeToString()表示。这仅用于域。
typdefault	text		如果类型没有关联的默认值，则为null。如果typdefaultbin不为null，则typdefault必须包含由typdefaultbin表示的默认表达式的人类可读版本。如果typdefaultbin为null且typdefault不为null，则typdefault是类型默认值的外部表示，可以将其输入到类型的输入转换器以生成常量。
typacl	aclitem[]		访问权限; 有关详细信息，请参阅 GRANT 和 REVOKE 。

下表列出了typcategory的系统定义值。此列表的任何未来添加项也将是大写ASCII字母。所有其他ASCII字符都保留给用户定义类别。

Table 2. typcategory Codes

代码	种类
A	数组类型
B	布尔类型
C	复合类型
D	日期/时间类型
E	枚举类型
G	几号类型
I	网络地址类型
N	数字类型
P	伪类型
R	范围类型
S	字符串类型
T	时间跨度类型
U	用户自定义类型
V	位串类型
X	未知类型

Parent topic: [系统目录定义](#)

pg_type_encoding系统目录表包含了列存储类型信息。

Table 1. pg_catalog.pg_type_encoding

列	类型	描述符	存储	描述
typeid	oid	not null	plain	pg_attribute 的外键
typoptions	text []		extended	实际的选项

Parent topic: [系统目录定义](#)

pg_user_mapping

系统目录表pg_user_mapping存储从本地用户到远程用户的映射。 您必须具有管理员权限才能查看此目录。 普通用户仅限访问此目录，请改用pg_user_mappings视图。

Table 1. pg_catalog.pg_user_mapping

列	类型	参考	描述
umuser	oid	pg_authid.oid	要映射的本地角色的OID，如果用户映射是公共的，则为0。
umserver	oid	pg_foreign_server.oid	包含此映射的外部服务器的OID。

列	类型	参考	描述
umoptions	text[]		特定于用户映射的选项，作为“keyword = value”字符串。

Parent topic: [系统目录定义](#)

pg_user_mappings

pg_user_mappings视图提供对用户映射信息的访问。此视图本质上是pg_user_mappings系统目录表的公共可读视图，如果用户没有查看它的访问权限，则会忽略选项字段。

Table 1. pg_user_mappings

列	类型	参考	描述
umid	oid	pg_user_mapping.oid	用户映射的OID。
srvid	oid	pg_foreign_server.oid	包含此映射的外部服务器的OID。
srvname	text	pg_foreign_server.srvname	外部服务器的名称。
umuser	oid	pg_authid.oid	要映射的本地角色的OID，如果用户映射是public的，则为0。

列	类型	参考	描述
username	name		要映射的本地用户的名称。
umoptions	text[]		特定于用户映射的选项，作为“keyword = value”字符串。

要保护存储为用户映射选项的密码信息，umoptions列将显示为null，除非以下应用情况之一：

- 当前用户是被映射的用户，拥有服务器或拥有USAGE权限。
- 当前用户是服务器所有者，映射是PUBLIC。
- 当前用户是超级用户。

Parent topic: [系统目录定义](#)

user_mapping_options

user_mapping_options视图包含为当前数据库中的用户映射定义的所有选项。MPP数据库仅显示当前用户可以访问的那些用户映射（通过作为所有者或具有某些权限）。

Table 1. user_mapping_options

列	类型	参考	描述
authorization_identif ier	sql_identif ier		要映射的用户名称， 如果映射是公共的，则 为PUBLIC。
foreign_server_catal og	sql_identif ier		定义此映射使用的外部 服务器的数据库的名称 (始终是当前数据库) 。
foreign_server_nam e	sql_identif ier		此映射使用的外部服务 器的名称。
option_name	sql_identif ier		选项的名称。
option_value	character _data		选项的值。此列将显示 null，除非： • 当前用户是被映射的 用户。 • 映射适用于PUBLIC ，当前用户是外部服 务器所有者。 • 当前用户是超级用户 。 目的是保护存储为用户 映射选项的密码信息。

Parent topic: [系统目录定义](#)

user_mappings

user_mappings视图包含当前数据库中定义的所有用户mappgins。 MPP数据库仅显示当前用户可以访问的那些用户映射（通过作为所有者或具有某些权限）。

Table 1. user_mappings

列	类型	参考	描述
authorization_identif ier	sql_identif ier		要映射的用户的名称， 如果映射是公共的，则 为PUBLIC。
foreign_server_catal og	sql_identif ier		定义此映射使用的外部 服务器的数据库的名称 （始终是当前数据库） 。
foreign_server_nam e	sql_identif ier		此映射使用的外部服务 器的名称。

Parent topic: [系统目录定义](#)

gp_toolkit管理模式

MPP数据库提供了一个名为gp_toolkit的管理模式，可用于查询系统catalog，日志文件和操作环境中的系统状态信息。 gp_toolkit模式包含许多视图，您可以使用SQL命令访问这些视图。 尽管某些对象可能需要超级用户权限，但所有数据库用户都可以访问gp_toolkit模式。 为了方便起见，您可能需要将gp_toolkit模式

添加到模式搜索路径。 例如：

```
=> ALTER ROLE myrole SET search_path TO myschema,gp_toolkit;
```

本文档介绍了gp_toolkit中最有用的视图。 您可能会注意到gp_toolkit模式中的其他对象（视图，函数和外部表）在本文档中未描述（这些是本节中描述的视图的支持对象）。

Warning: 不要在gp_toolkit模式中更改数据库对象。 不要在模式中创建数据库对象。 对模式对象的更改可能会影响模式对象返回的管理信息的准确性。 备份数据库，然后使用gpbackup和gprestore实用程序还原时， 在gp_toolkit模式中所做的任何更改都将丢失。

- [检查需要例行维护的表](#)
- [检查锁](#)
- [检查追加优化表](#)
- [查看MPP数据库服务器日志文件](#)
- [检查服务器配置文件](#)
- [检查失败的segments](#)
- [检查资源组活动和状态](#)
- [检查资源队列活动和状态](#)
- [检查查询磁盘溢出空间使用情况](#)
- [查看用户和组（角色）](#)
- [检查数据库对象大小和磁盘空间](#)
- [检查数据分布不均](#)

Parent topic: [MPP数据库参考手册](#)

检查需要例行维护的表

以下视图可以帮助您识别需要日常维护的表（VACUUM和/或ANALYZE）。

- [gp_bloat_diag](#)
- [gp_stats_missing](#)

VACUUM或VACUUM FULL命令可回收已删除或过时的行所占用的磁盘空间。由于MPP数据库中使用了MVCC事务并发模型，即使任何新事务都不可见，被删除或更新的数据行仍占据磁盘上的物理空间。过期的行会增加磁盘上的表大小，并最终减慢表的扫描速度。

ANALYZE命令收集查询优化器所需的列级统计信息。MPP数据库使用依赖数据库统计信息的基于成本的查询优化器。准确的统计信息使查询优化器可以更好地估计选择性和查询操作检索到的行数，从而选择最有效的查询计划。

Parent topic: [gp_toolkit管理模式](#)

gp_bloat_diag

此视图显示膨胀的常规堆存储表（给定表统计信息，磁盘上的实际页面数超过了预期的页面数）。膨胀的表需要VACUUM或VACUUM FULL，才能回收已删除或过时的行所占用的磁盘空间。所有用户都可以访问该视图，但是非超级用户将只能看到他们有权访问的表。

Note: 有关返回追加优化表信息的诊断函数，请参阅[检查追加优化表](#)。

Table 1. gp_bloat_diag视图

列	描述
bdirelid	表OID。
bdinspname	模式名称。

列	描述
bdirelname	表名。
bdirelpages	磁盘上实际页的数量。
bdiexppages	给定表数据的预期页数。
bdidiag	膨胀诊断消息。

gp_stats_missing

该视图显示的表没有统计信息，因此可能需要对该表运行ANALYZE。

Table 2. gp_stats_missing view

列	描述
smischema	模式名称。
smitable	表名。
smisize	该表是否有统计信息？ 如果该表没有在系统catalog中记录行数和行大小统计信息，则为False，这可能表明该表需要进行分析。 如果表不包含任何行，这也将为false。 例如，分区表的父表始终为空，并且始终返回false。
smicols	表中的列数。
smirecs	表中的行数。

检查锁

当事务访问关系（例如表）时，它获取一个锁。 根据获取的锁的类型，后续事

务可能必须等待才能访问相同的关系。 有关锁类型的更多信息，请参阅*MPP数据库管理员手册*中的“管理数据”。 MPP数据库资源队列（用于资源管理）还使用锁来控制查询进入系统的权限。

gp_locks_*系列视图可以帮助诊断由于锁定而正在等待访问对象的查询和会话。

- [gp_locks_on_relation](#)
- [gp_locks_on_resqueue](#)

Parent topic: [gp_toolkit管理模式](#)

gp_locks_on_relation

此视图显示当前在某个关系上持有的所有锁，以及有关与该锁关联的查询的关联会话信息。 有关锁类型的更多信息，请参阅*MPP数据库管理员手册*中的“管理数据”。 所有用户都可以访问此视图，但是非超级用户将只能看到他们有权访问的关系的锁。

Table 3. gp_locks_on_relation视图

列	描述
lorlocktype	可锁定对象的类型：relation, extend, page, tuple, transactionid, object, userlock, resource queue, 或 advisory
lordatabase	对象所在的数据库的对象ID，如果该对象是共享对象，则为零。
lorrelname	关系的名称。
lorrelation	关系的OID。
lortransaction	受锁影响的事务ID。

列	描述
lorpid	持有或正在等待此锁定的服务器进程的进程ID。 如果锁由准备好的事务持有，则为NULL。
lormode	此进程持有或需要的锁定模式的名称。
lorgranted	显示是授予锁（true）还是不授予锁（false）。
lorcurrentquery	会话中的当前查询。

gp_locks_on_resqueue

Note: 仅当基于资源队列的资源管理处于活动状态时，gp_locks_on_resqueue视图才有效。

该视图显示了资源队列中当前持有的所有锁，以及与该锁关联的查询的关联会话信息。 所有用户都可以访问此视图，但是非超级用户将只能看到与其自己的会话相关联的锁。

Table 4. gp_locks_on_resqueue视图

列	描述
lorusename	执行会话的用户名。
lorrsqname	资源队列名称。
lorlocktype	可锁定对象的类型：资源队列
lorobjid	锁定事务的ID。
lortransaction	受锁影响的事务的ID。
lorpid	受锁影响的事务的进程ID。
lormode	此进程持有或需要的锁定模式的名称。
lorgranted	显示是授予锁（true）还是不授予锁（false）。

列	描述
lorwaiting	显示会话是否正在等待。

检查追加优化表

gp_toolkit模式包含一组诊断功能，可用于调查追加优化表的状态。

创建追加优化表（或面向列的追加优化表）时，将隐式创建另一个表，其中包含有关该表当前状态的元数据。元数据包括诸如表的每个segment中的记录数之类的信息。

追加优化表可能包含不可见的行，这些行已被更新或删除，但是保留在存储中，直到使用VACUUM压缩表为止。使用辅助可见性映射表或visimap跟踪隐藏的行。

通过以下函数，您可以访问针对追加优化和面向列的表的元数据，并查看不可见的行。其中一些函数具有两个版本：一个采用表的oid，而另一个采用表的名称。后一种版本在函数名称后附加了“_name”。

Parent topic: [gp_toolkit管理模式](#)

__gp_aovisimap_compaction_info(oid)

此函数显示追加优化表的压缩信息。该信息适用于MPP数据库segment中存储表数据的磁盘上数据文件。您可以使用该信息来确定将通过VACUUM操作对追加优化表进行压缩的数据文件。

Note: 在VACUUM操作从数据文件中删除该行之前，已删除或已更新的数据行将占据磁盘上的物理空间，即使它们已被新事务隐藏。配置参数gp_appendonly_compaction控制VACUUM命令的功能。

下表描述了__gp_aovisimap_compaction_info函数输出表。

Table 5. __gp_aovisimap_compaction_info输出表

列	描述
content	MPP数据库segment ID。
datafile	segment上数据文件的ID。
compaction_possible	该值为t或f。 值t表示执行VACUUM操作时压缩数据文件中的数据。 服务器配置参数gp_appendonly_compaction_threshold会影响此值。
hidden_tupcount	在数据文件中，隐藏（删除或更新）的行数。
total_tupcount	在数据文件中，总行数。
percent_hidden	在数据文件中，隐藏（已删除或更新）行与总行的比率（百分比）。

__gp_aoseg_name('table_name')

此函数返回追加优化表的磁盘段文件中包含的元数据信息。

Table 6. __gp_aoseg_name输出表

列	描述
segno	文件段号。
eof	该文件段的文件有效结尾。
tupcount	段中的元组总数，包括不可见的元组。

列	描述
varblockcount	文件段中的varblocks总数。
eof_uncompressed	文件末尾（如果文件段未压缩）。
modcount	数据修改操作的数量。
state	文件段的状态。 指示段是活动的还是压缩后可以删除的。

__gp_aoseg_history(oid)

此函数返回追加优化表的磁盘段文件中包含的元数据信息。 它显示aoseg元信息的所有不同版本（堆元组）。 数据很复杂，但是对系统有深入了解的用户可能会发现它对于调试很有用。

输入参数是追加优化表的oid。

调用__gp_aoseg_history_name ('table_name') 可获得与表名参数相同的结果。

Table 7. __gp_aoseg_history输出表

列	描述
gp_tid	元组的id。
gp_xmin	最早事务的id。
gp_xmin_status	gp_xmin事务的状态。
gp_xmin_commit_	gp_xmin事务的提交分布式id。
gp_xmax	最晚的事务id。
gp_xmax_status	最晚的事务的状态。

列	描述
gp_xmax_commit_	gp_xmax事务的提交分布式id。
gp_command_id	查询命令的id。
gp_infomask	包含状态信息的位图。
gp_update_tid	更新的元组的ID（如果该行已更新）。
gp_visibility	元组可见性状态。
segno	段文件中的段号。
tupcount	元组数量，包括隐藏元组。
eof	段的文件的有效结尾。
eof_uncompressed	该段的文件末尾，如果数据未压缩。
modcount	数据修改计数。
state	段的状态。

__gp_aocsseg(oid)

此函数返回包含在面向列的追加优化表的磁盘段文件中的元数据信息，不包括不可见的行。 每行描述表中一系列的段。

输入参数是面向列的追加优化表的oid。 以表名作为参数调用 __gp_aocsseg_name ('table_name') 获得相同的结果。

Table 8. __gp_aocsseg(oid)输出表

列	描述
gp_tid	表的ID。
segno	段号。

列	描述
column_num	列号。
physical_segno	段文件中段的编号。
tupcount	段中的行数，不包括隐藏的元组。
eof	段的文件的有效结尾。
eof_uncompressed	该段文件的末尾（如果数据未压缩）。
modcount	段的数据修改操作计数。
state	段的状态。

__gp_aocsseg_history(oid)

此函数返回包含在面向列的追加优化表的磁盘段文件中的元数据信息。 每行描述表中一列的段。 数据很复杂，但是对系统有深入了解的用户可能会发现它对于调试很有用。

输入参数是面向列的追加优化表的oid。 以表名称作为参数调用 __gp_aocsseg_history_name ('table_name') 获得相同的结果。

Table 9. __gp_aocsseg_history输出表

列	描述
gp_tid	元组的oid。
gp_xmin	最早的事务。
gp_xmin_status	gp_xmin事务的状态。
gp_xmin_	gp_xmin的文本表示形式。
gp_xmax	最近的事务。

列	描述
gp_xmax_status	gp_xmax事务的状态。
gp_xmax_	gp_max的文本表示形式。
gp_command_id	在元组上运行的命令的ID。
gp_infomask	包含状态信息的位图。
gp_update_tid	新的元组的ID（如果该行已更新）。
gp_visibility	元组可见性状态。
segno	段文件中的段号。
column_num	列号。
physical_segno	包含列数据的段。
tupcount	段中的元组总数。
eof	段的文件的有效结尾。
eof_uncompressed	该段文件的末尾（如果数据未压缩）。
modcount	数据修改操作的计数。
state	段的状态。

__gp_aovisimap(oid)

此函数根据可见性映射返回元组id，段文件和每个不可见元组的行号。

输入参数是追加优化表的oid。

使用__gp_aovisimap_name ('table_name') 获得与表名作为参数相同的结果。

列	描述
tid	元组ID。
segno	段文件的编号。
row_num	已删除或更新的行的行号。

__gp_aovisimap_hidden_info(oid)

此函数返回段文件中针对追加优化表的隐藏和可见元组数。

输入参数是追加优化表的oid。

调用__gp_aovisimap_hidden_info_name ('table_name') 可获得与表名参数相同的结果。

列	描述
segno	段文件的编号。
hidden_tupcount	段文件中隐藏元组的数量。
total_tupcount	段文件中的元组总数。

__gp_aovisimap_entry(oid)

此函数返回有关表的每个可见性映射条目的信息。

输入参数是追加优化表的oid。

调用__gp_aovisimap_entry_name ('table_name') 可获得与表名参数相同的结果。

Table 10. __gp_aovisimap_entry输出表

列	描述
segno	可见性映射条目的段号。
first_row_num	条目的第一行号。
hidden_tupcount	条目中的隐藏元组数。
bitmap	可见性位图的文本表示。

查看MPP数据库服务器日志文件

MPP数据库系统的每个组件（master，standby，primary和mirror）均保留其自己的服务器日志文件。使用gp_log_*系列视图，您可以对服务器日志文件执行SQL查询，以找到感兴趣的特定条目。这些视图的使用需要超级用户权限。

- [gp_log_command_timings](#)
- [gp_log_database](#)
- [gp_log_master_concise](#)
- [gp_log_system](#)

Parent topic: [gp_toolkit管理模式](#)

gp_log_command_timings

该视图使用外部表来读取主数据库上的日志文件，并报告在数据库会话中执行的SQL命令的执行时间。使用此视图需要超级用户权限。

Table 11. gp_log_command_timings视图

列	描述
logsession	会话标识符（以"con"为前缀）。
logcmdcount	会话中的命令号（以"cmd"为前缀）。
logdatabase	数据库的名称。
loguser	数据库用户的名称。
logpid	进程ID（前缀为"p"）。
logtimemin	此命令的第一条日志消息的时间。
logtimemax	该命令的最后一条日志消息的时间。
logduration	语句从开始到结束时间的持续时间。

gp_log_database

该视图使用外部表读取整个MPP系统的服务器日志文件（master，primary和mirror），并列出了与当前数据库关联的日志条目。可以通过会话ID（logsession）和命令ID（logcmdcount）标识关联的日志条目。使用此视图需要超级用户权限。

Table 12. gp_log_database视图

列	描述
logtime	日志消息的时间戳。
loguser	数据库用户的名称。
logdatabase	数据库的名称。
logpid	关联的进程ID（前缀为"p"）。

列	描述
logthread	关联的线程数（以"th"为前缀）。
loghost	segment或master主机名。
logport	segment或master端口号。
logsessiontime	会话连接打开时间。
logtransaction	全局事务ID。
logsession	会话标识符（以"con"为前缀）。
logcmdcount	会话中的命令号（以"cmd"为前缀）。
logsegment	segment内容标识符（对于primary标识符，前缀为"seg"，对于mirror，前缀为"mir"。master服务器的内容ID始终为-1）。
logslice	切片ID（正在执行的查询计划的一部分）。
logdistxact	分布式事务ID。
loglocalxact	本地事务ID。
logsubxact	子事务ID。
logseverity	LOG, ERROR, FATAL, PANIC, DEBUG1或DEBUG 2。
logstate	与日志消息关联的SQL状态代码。
logmessage	日志或错误消息文本。
logdetail	与错误消息关联的详细消息文本。
loghint	与错误消息关联的提示消息文本。
logquery	内部生成的查询文本。
logquerypos	游标索引到内部生成的查询文本中。
logcontext	生成此消息的上下文。
logdebug	用于调试的带有详细信息的查询字符串。

列	描述
logcursorpos	查询字符串中的游标索引。
logfunction	生成此消息的函数。
logfile	生成此消息的日志文件。
logline	日志文件中生成此消息的行。
logstack	与该消息关联的堆栈跟踪的全文。

gp_log_master_concise

该视图使用外部表从master日志文件读取日志字段的子集。 使用此视图需要超级用户权限。

Table 13. gp_log_master_concise视图

列	描述
logtime	日志消息的时间戳。
logdatabase	数据库的名称。
logsession	会话标识符（以"con"为前缀）。
logcmdcount	会话中的命令号（以"cmd"为前缀）。
logmessage	日志或错误消息文本。

gp_log_system

该视图使用一个外部表来读取整个MPP系统（master，segment和mirror）的服务器日志文件，并列出所有日志条目。 可以通过会话ID（logsession）和命令ID（logcmdcount）标识关联的日志条目。 使用此视图需要超级用户权限。

Table 14. gp_log_system视图

列	描述
logtime	日志消息的时间戳。
loguser	数据库用户的名称。
logdatabase	数据库的名称。
logpid	关联的进程ID（前缀为"p"）。
logthread	关联的线程数（以"th"为前缀）。
loghost	segment或master主机名。
logport	segment或master端口号。
logsessiontime	会话连接打开时间。
logtransaction	全局事务ID。
logsession	会话标识符（以"con"为前缀）。
logcmdcount	会话中的命令号（以"cmd"为前缀）。
logsegment	segment内容标识符（对于primary标识符，前缀为"seg"，对于mirror，前缀为"mir"。master服务器的内容ID始终为-1）。
logslice	切片ID（正在执行的查询计划的一部分）。
logdistxact	分布式事务ID。
loglocalxact	本地事务ID。
logsubxact	子事务ID。
logseverity	LOG, ERROR, FATAL, PANIC, DEBUG1或DEBUG2。
logstate	与日志消息关联的SQL状态代码。
logmessage	日志或错误消息文本。

列	描述
logdetail	与错误消息关联的详细消息文本。
loghint	与错误消息关联的提示消息文本。
logquery	内部生成的查询文本。
logquerypos	游标索引到内部生成的查询文本中。
logcontext	生成此消息的上下文。
logdebug	用于调试的带有详细信息的查询字符串。
logcursorpos	查询字符串中的游标索引。
logfunction	生成此消息的函数。
logfile	生成此消息的日志文件。
logline	日志文件中生成此消息的行。
logstack	与该消息关联的堆栈跟踪的全文。

检查服务器配置文件

MPP数据库系统的每个组件（master，standby，primary和mirror）都有其自己的服务器配置文件（postgresql.conf）。以下gp_toolkit对象可用于检查系统中所有primary的postgresql.conf文件的参数设置：

- [gp_param_setting\('parameter_name'\)](#)
- [gp_param_settings_seg_value_diffs](#)

Parent topic: [gp_toolkit管理模式](#)

gp_param_setting('parameter_name')

此函数采用服务器配置参数的名称，并返回master和每个活动segment的postgresql.conf值。 所有用户均可使用此函数。

Table 15. gp_param_setting('parameter_name')函数

列	描述
paramsegment	segment内容ID（仅显示活动segment）。 master内容ID始终为-1。
paramname	参数名称。
paramvalue	参数值。

示例：

```
SELECT * FROM gp_param_setting('max_connections');
```

gp_param_settings_seg_value_diffs

分类为本地参数的服务器配置参数（意味着每个segment都从其自己的postgresql.conf文件获取参数值），应在所有segment上进行相同的设置。该视图显示的本地参数设置不一致。 不包括应该具有不同值的参数（例如port）。 所有用户均可访问此视图。

Table 16. gp_param_settings_seg_value_diffs视图

列	描述
psdname	参数的名称。
psdvalue	参数的值。

列	描述
psdcount	具有此值的segment数。

检查失败的segments

[gp_pgdatabase_invalid](#)视图可用于检查宕机的segment。

Parent topic: [gp_toolkit管理模式](#)

gp_pgdatabase_invalid

此视图显示有关在系统目录中标记为宕机的segment的信息。 所有用户均可访问此视图。

Table 17. gp_pgdatabase_invalid视图

列	描述
pgdbidbid	segment的dbid。 每个segment都有唯一的dbid。
pgdbiisprimary	segment的角色是否是primary？（t或f）
pgdbicontent	该segment的内容ID。 primary和mirror将具有相同的内容ID。
pgdbivalid	segment是否在线有效？（t或f）
pgdbidefinedprimary	在系统初始化时，此segment是否分配为primary？（t或f）

资源组/占内存的心

Note: 仅当基于资源组的资源管理处于活动状态时，本节中描述的资源组活动和状态视图才有效。

资源组管理事务，以避免耗尽系统CPU和内存资源。 每个数据库用户都被分配了一个资源组。 在运行事务之前，MPP数据库会根据为用户资源组配置的限制评估用户提交的每个事务。

您可以使用gp_resgroup_config视图检查每个资源组的配置。 您可以使用gp_resgroup_status*视图显示当前事务状态和每个资源组的资源使用情况。

- [gp_resgroup_config](#)
- [gp_resgroup_status](#)
- [gp_resgroup_status_per_host](#)
- [gp_resgroup_status_per_segment](#)

Parent topic: [gp_toolkit管理模式](#)

gp_resgroup_config

使用gp_resgroup_config视图，管理员可以查看资源组的当前CPU，内存和并发限制。 该视图还显示建议的限制设置。 更改限制后，建议的限制将不同于当前限制，但是无法立即应用新值。

所有用户都可以访问该视图。

Table 18. gp_resgroup_config

列	描述
groupid	资源组的ID。

列	描述
groupname	资源组的名称。
concurrency	为资源组指定的并发（CONCURRENCY）值。
proposed_concurrency	资源组的待定并发值。
cpu_rate_limit	为资源组指定的CPU限制（CPU_RATE_LIMIT）值或-1。
memory_limit	为资源组指定的内存限制（MEMORY_LIMIT）值。
proposed_memory_limit	资源组的待定内存限制值。
memory_shared_quota	为资源组指定的共享内存配额（MEMORY_SHARED_QUOTA）值。
proposed_memory_shared_quota	资源组的待定共享内存配额值。
memory_spill_ratio	为资源组指定的内存溢出率（MEMORY_SPILL_RATIO）值。
proposed_memory_spill_ratio	资源组的待定内存溢出率值。
memory_auditor	资源组的内存审核。
cpuset	为资源组保留的CPU核心，或-1。

gp_resgroup_status

使用gp_resgroup_status视图，管理员可以查看资源组的状态和活动。 它显示了每个资源组正在等待运行的查询数量以及系统中当前正在活动的查询数量。

该视图还显示资源组的当前内存和CPU使用率。

Note: 资源组使用主机系统上配置的Linux控制组（cgroup）。 cgroup用于管理主机系统资源。 当资源组使用作为嵌套cgroup组的一部分的cgroup时，资源组限制是相对于父cgroup分配的。 有关嵌套cgroup和MPP数据库资源组限制的信息，请参阅[理解角色和组件资源组](#)。

所有用户均可访问此视图。

Table 19. gp_resgroup_status视图

列	描述
rsgname	资源组的名称。
groupid	资源组的ID。
num_running	资源组中当前正在执行的事务数。
num_queueing	资源组当前排队的事务数。
num_queued	自上次启动MPP数据库集群以来，资源组的排队事务总数，不包括num_queueing。
num_executed	自MPP数据库集群上次启动以来，资源组中已执行事务的总数，不包括num_running。
total_queue_duration	自上次启动MPP数据库集群以来，所有事务排队的总时间。
cpu_usage	每个MPP数据库segment主机上资源组的实时CPU使用率。
memory_usage	每个MPP数据库segment主机上资源组的实时内存使用情况。

cpu_usage字段是JSON格式的key : value字符串， 用于为每个资源组标识每个

segment的CPU使用率百分比。key是segment ID，value是segment主机上资源组的CPU使用率百分比。在segment主机上运行的所有segment的CPU总使用率不应超过gp_resource_group_cpu_limit。示例cpu_usage列输出：

```
{"-1":0.01, "0":0.31, "1":0.31}
```

在此示例中，segment 0和segment 1在同一主机上运行；它们的CPU使用率是相同的。

memory_usage字段也是JSON格式的key：value字符串。字符串内容因资源组的类型而异。对于分配给角色的每个资源组（默认内存审核器vmtracker），此字符串标识每个segment上已使用，可用，已授予和建议的固定和共享内存配额分配。key是segment ID，value是以MB为单位显示的内存值。下面的示例显示分配给角色的资源组的单个segment的memory_usage列输出：

```
"0":{"used":0, "available":76, "quota_used":-1, "quota_available":60, "quota_granted":60, "quota_reserved":0, "limit_granted":60, "limit_reserved":0}
```

对于分配给外部组件的每个资源组，memory_usage JSON格式的字符串标识了每个segment上使用的内存和内存限制。以下示例显示单个segment的外部组件资源组的memory_usage列输出：

```
"1":{"used":11, "limit_granted":15}
```

Note: 请参阅下文所述

的gp_resgroup_status_per_host和gp_resgroup_status_per_segment视图，以更友好地显示CPU和内存使用情况。

gp_resgroup_status_per_host

[gp_resgroup_status_per_host](#)视图以主机为单位显示每个资源组的实时CPU和内存使用率（MB）。该视图还显示主机上每个资源组的可用和已授予组的固定

和共享内存。

Table 20. gp_resgroup_status_per_host视图

列	描述
rsgname	资源组的名称。
groupid	资源组的ID。
hostname	segment主机的主机名。
cpu	主机上资源组的实时CPU使用率。
memory_used	主机上资源组的实时内存使用情况。 此总数包括资源组固定和共享内存。 它还包括资源组使用的全局共享内存。
memory_available	主机上可用的资源组的未使用的固定和共享内存。 此总数不包括可用资源组全局共享内存。
memory_quota_used	主机上资源组的实时固定内存使用情况。
memory_quota_available	主机上资源组可用的固定内存。
memory_quota_proposed	MPP数据库已分配给主机上资源组的固定内存总量。
memory_shared_used	主机上的资源组使用的组共享内存。 如果资源组使用任何全局共享内存，则此数量也将包括在总数中。
memory_shared_available	主机上资源组可用的组共享内存量。 资源组全局共享内存不包括在此总计中。
memory_shared_granted	MPP数据库已分配给主机上资源组的组共享内存部分。 资源组全局共享内存不包括在此值中。
memory_shared_proposed	主机上的资源组请求的组共享内存的总数。

如果群集没有足够的内存来分配，则memory_shared_granted值可能小于memory_shared_proposed。当资源组使用全局共享内存时，它可能大于memory_shared_proposed。随着时间的推移，memory_shared_granted和memory_shared_proposed应该达到相同的值。

gp_resgroup_status_per_host视图的示例输出：

rsgname	groupid	hostname	cpu	memory_used	memory_available	memory_quota
admin_group	6438	my-desktop	0.84	1	271	68
default_group	6437	my-desktop	0.00	0	816	0
(2 rows)						

gp_resgroup_status_per_segment

[gp_resgroup_status_per_segment](#)视图基于每个segment实例和每个主机显示每个资源组的实时CPU和内存使用率（MB）。该视图还显示主机上每个资源组和segment实例组合的可用和已授予组的固定和共享内存。

Table 21. gp_resgroup_status_per_segment视图

列	描述
rsgname	资源组的名称。
groupid	资源组的ID。
hostname	segment主机的主机名。
segment_id	segment主机上segment实例的内容ID。
cpu	主机上的segment实例的资源组的实时CPU使用率。

列	描述
memory_used	主机上的segment实例的资源组的实时内存使用情况。此总数包括资源组固定和共享内存。它还包括资源组使用的全局共享内存。
memory_available	主机上segment实例的资源组的未使用的固定和共享内存。
memory_quota_used	主机上segment实例的资源组的实时固定内存使用情况。
memory_quota_available	资源组可用于主机上的segment实例的固定内存。
memory_quota_proposed	MPP数据库已为主机上的segment分配给资源组的固定内存总量。对于主机上的所有资源组和segment实例组合，此值都相同。
memory_shared_used	资源组用于主机上的segment实例的组共享内存。
memory_shared_available	主机上的segment实例可用的组共享内存量。资源组全局共享内存不包括在此总计中。
memory_shared_granted	MPP数据库已为主机上的segment实例分配给资源组的组共享内存部分。资源组全局共享内存不包括在此值中。
memory_shared_proposed	主机上的资源组请求的组共享内存的总数。对于主机上的所有资源组和segment实例组合，此值都相同。

该视图的查询输出与gp_resgroup_status_per_host视图的查询输出相似，并针对每个主机上的每个segment实例划分CPU和内存（已使用和可用）。

检查资源队列活动和状态

Note: 仅当基于资源队列的资源管理处于活动状态时，本节中描述的资源队列活动和状态视图才有效。

资源队列的目的是在任何给定时间限制系统中活动查询的数量，以避免耗尽内存，CPU和磁盘I/O等系统资源。所有数据库用户都分配给一个资源队列，并且在运行该用户之前，将首先针对资源队列限制评估用户提交的每个语句。
gp_resq_*系列视图可用于检查通过它们各自的资源队列当前提交给系统的语句的状态。请注意，超级用户执行的语句免于资源排队。

- [gp_resq_activity](#)
- [gp_resq_activity_by_queue](#)
- [gp_resq_priority_statement](#)
- [gp_resq_role](#)
- [gp_resqueue_status](#)

Parent topic: [gp_toolkit管理模式](#)

gp_resq_activity

对于具有活动工作负载的资源队列，此视图为通过资源队列提交的每个活动语句显示一行。所有用户均可访问此视图。

Table 22. gp_resq_activity视图

列	描述
resqprocpid	分配给该语句的进程ID（在master上）。
resqrole	用户名。
resqoid	资源队列对象标识。

列	描述
resqname	资源队列名称。
resqstart	已向系统发布时间声明。
resqstatus	语句状态：running,waiting或cancelled。

gp_resq_activity_by_queue

对于具有活动工作负载的资源队列，此视图显示队列活动的摘要。 所有用户均可访问此视图。

Table 23. gp_resq_activity_by_queue视图

列	描述
resqoid	资源队列对象标识。
resqname	资源队列名称。
resqlast	发布到队列的最后一条语句的时间。
resqstatus	最后一条语句的状态：running,waiting或cancelled。
resqtotal	此队列中的语句总数。

gp_resq_priority_statement

该视图显示了MPP数据库系统中当前正在运行的所有语句的资源队列优先级，会话ID和其他信息。 所有用户均可访问此视图。

Table 24. gp_resq_priority_statement视图

列	描述
rqpdname	会话连接到的数据库名称。
rqpusename	执行语句的用户。
rqpsession	会话ID。
rqpcommand	该会话中的语句编号（命令ID和会话ID唯一标识一条语句）。
rqppriority	此语句的资源队列优先级（MAX，HIGH，MEDIUM，LOW）。
rqpweight	与该语句的优先级关联的整数值。
rqpquery	语句的查询文本。

gp_resq_role

此视图显示与角色关联的资源队列。 所有用户均可访问此视图。

Table 25. gp_resq_role视图

列	描述
rrrolname	角色（用户）名称。
rrrsqname	分配给此角色的资源队列名称。 如果尚未将角色明确分配给资源队列，则它将位于默认资源队列（ <i>pg_default</i> ）中。

gp_resqueue_status

该视图使管理员可以查看资源队列的状态和活动。 它显示了特定资源队列中有多少查询正在等待运行以及当前系统中有多少查询处于活动状态。

Table 26. gp_resqueue_status视图

列	描述
queueid	资源队列的ID。
rsqname	资源队列的名称。
rsqcountlimit	资源队列的活动查询阈值。 值-1表示没有限制。
rsqcountvalue	资源队列中当前正在使用的活动查询槽的数量。
rsqcostlimit	资源队列的查询成本阈值。 值-1表示没有限制。
rsqcostvalue	资源队列中当前所有语句的总成本。
rsqmemorylimit	资源队列的内存限制。
rsqmemoryvalue	资源队列中当前所有语句使用的总内存。
rsqwaiters	资源队列中当前正在等待的语句数。
rsqholders	此资源队列中当前在系统上运行的语句数。

检查查询磁盘溢出空间使用情况

`gp_workfile_*`视图显示有关当前正在使用磁盘溢出空间的所有查询的信息。 如

果MPP数据库没有足够的内存来在内存中执行查询，则会在磁盘上创建工作文件。此信息可用于故障排除和优化查询。视图中的信息还可用于指定MPP数据库配置参数gp_workfile_limit_per_query和gp_workfile_limit_per_segment的值。

- [gp_workfile_entries](#)
- [gp_workfile_usage_per_query](#)
- [gp_workfile_usage_per_segment](#)

Parent topic: [gp_toolkit管理模式](#)

gp_workfile_entries

该视图为每个运算符在当前时间使用一个磁盘空间来存储segment上工作文件的每一行。所有用户均可访问该视图，但是非超级用户只能查看其有权访问的数据库信息。

Table 27. gp_workfile_entries

列	类型	参考	描述
command_cnt	integer		查询的命令ID。
content	smallint		segment实例的内容标识符。
current_query	text		进程正在执行的查询。
datname	name		MPP数据库名称。
directory	text		工作文件的路径。
optype	text		创建工作文件的查询运算符类型。

列	类型	参考	描述
procpid	integer		服务器进程的进程ID。
sess_id	integer		会话ID。
size	bigint		工作文件的大小（以字节为单位）。
numfiles	bigint		创建的文件数。
slice	smallint		查询计划切片。 查询计划中正在执行的部分。
state	text		创建工作文件的查询状态。
username	name		角色名称。
workmem	integer		分配给运算符的内存量，以KB为单位。

gp_workfile_usage_per_query

对于当前segment上使用工作文件的磁盘空间的每个查询，此视图都包含一行。所有用户均可访问该视图，但是非超级用户只能查看其有权访问的数据库信息。

Table 28. gp_workfile_usage_per_query

列	类型	参考	描述
command_cnt	integer		查询的命令ID。
content	smallint		segment实例的内容标识符。

列	类型	参考	描述
current_query	text		进程正在执行的查询。
datname	name		MPP数据库名称。
procpid	integer		服务器进程的进程ID。
sess_id	integer		会话ID。
size	bigint		工作文件的大小（以字节为单位）。
numfiles	bigint		创建的文件数。
state	text		创建工作文件的查询状态。
username	name		角色名称。

gp_workfile_usage_per_segment

该视图为每个segment包含一行。 每行显示当前时间segment上用于工作文件的磁盘空间总量。 所有用户均可访问该视图，但是非超级用户只能查看其有权访问的数据库信息。

Table 29. gp_workfile_usage_per_segment

列	类型	参考	描述
content	smallint		segment实例的内容标识符。
size	bigint		segment上工作文件的总大小。

列	类型	参考	描述
numfiles	bigint		创建的文件数。

查看用户和组（角色）

将用户（角色）分组在一起以简化对象特权的管理通常很方便：通过这种方式，可以将特权授予或撤消整个组。 在MPP数据库中，这是通过创建代表组的角色，然后将组角色的成员身份授予各个用户角色来完成的。

[gp_roles_assigned](#)视图可用于查看系统中的所有角色及其分配的成员（如果角色也是组角色）。

Parent topic: [gp_toolkit管理模式](#)

gp_roles_assigned

此视图显示系统中的所有角色及其分配的成员（如果角色也是组角色）。 所有用户均可访问此视图。

Table 30. gp_roles_assigned视图

列	描述
raroleid	角色对象ID。 如果此角色具有成员（用户），则将其视为组角色。
rarolename	角色（用户或组）名称。
ramemberid	属于此角色成员的角色的角色对象ID。
ramembername	属于此角色的角色的名称。

检查数据库对象大小和磁盘空间

gp_size_*系列视图可用于确定分布式MPP数据库，模式，表或索引的磁盘空间使用情况。以下视图计算了所有primary中对象的总大小（大小计算中不包括mirror）。

- [gp_size_of_all_table_indexes](#)
- [gp_size_of_database](#)
- [gp_size_of_index](#)
- [gp_size_of_partition_and_indexes_disk](#)
- [gp_size_of_schema_disk](#)
- [gp_size_of_table_and_indexes_disk](#)
- [gp_size_of_table_and_indexes_licensing](#)
- [gp_size_of_table_disk](#)
- [gp_size_of_table_uncompressed](#)
- [gp_disk_free](#)

表和索引大小视图按对象ID（而不是名称）列出关系。要按名称检查表或索引的大小，必须在pg_class表中查找关系名称（relname）。例如：

```
SELECT relname as name, sotdsize as size, sotdtoastsize as  
toast, sotdadditionalsize as other  
FROM gp_size_of_table_disk as sotd, pg_class  
WHERE sotd.sotdoid=pg_class.oid ORDER BY relname;
```

Parent topic: [gp_toolkit管理模式](#)

gp_size_of_all_table_indexes

此视图显示表的所有索引的总大小。所有用户都可以访问此视图，但是非超级用户将只能看到他们有权访问的关系。

Table 31. gp_size_of_all_table_indexes视图

列	描述
soatoid	表的对象ID
soatisize	所有表索引的总大小（以字节为单位）
soatischemaname	模式名称
soatitablename	表名

gp_size_of_database

此视图显示数据库的总大小。所有用户都可以访问该视图，但是非超级用户将只能看到他们有权访问的数据库。

Table 32. gp_size_of_database视图

列	描述
sodddatname	数据库名称
sodddatsize	数据库大小（以字节为单位）

gp_size_of_index

此视图显示索引的总大小。所有用户都可以访问此视图，但是非超级用户将只

能看到他们有权访问的关系。

Table 33. gp_size_of_index视图

列	描述
soioid	索引的对象ID
soitableoid	索引所属表的对象ID
soisize	索引大小（以字节为单位）
soiindexschemaname	索引模式的名称
soiindexname	索引名称
soitableschema	表模式的名称
soitablename	表名称

gp_size_of_partition_and_indexes_disk

此视图显示分区子表及其索引在磁盘上的大小。所有用户均可访问此视图，但是非超级用户将只能看到他们有权访问的关系。

Table 34. gp_size_of_partition_and_indexes_disk视图

列	描述
sopaidparentoid	父表的对象ID
sopaidpartitionoid	分区表的对象ID
sopaidpartitiontablesize	分区表大小（以字节为单位）
sopaidpartitionindexessize	该分区上所有索引的总大小

列	描述
Sopaidparentschemaname	父模式的名称
Sopaidparenttablename	父表的名称
Sopaidpartitionsschemaname	分区模式的名称
sopaidpartitiontablename	分区表的名称

gp_size_of_schema_disk

此视图显示当前数据库中公共模式和用户创建的模式的模式大小。 所有用户都可以访问此视图，但是非超级用户将只能看到他们有权访问的架构。

Table 35. gp_size_of_schema_disk视图

列	描述
sosdnsp	模式名称
sosdschematablesize	模式中表的总大小（以字节为单位）
sosdschemaidxsize	模式中索引的总大小（以字节为单位）

gp_size_of_table_and_indexes_disk

此视图显示表及其索引在磁盘上的大小。 所有用户都可以访问此视图，但是非超级用户将只能看到他们有权访问的关系。

Table 36. gp_size_of_table_and_indexes_disk视图

列	描述
sotaidoid	父表的对象ID
sotaidtablesize	表的磁盘大小
sotaididxsize	表上所有索引的总大小
sotaidschemaaname	模式名称
sotaidtablename	表名

gp_size_of_table_and_indexes_licensing

此视图显示用于许可目的的表及其索引的总大小。 使用此视图需要超级用户权限。

Table 37. gp_size_of_table_and_indexes_licensing视图

列	描述
sotailoid	表的对象ID
sotailtablesize	表的总磁盘大小
sotailtablesizeuncompressed	如果该表是压缩的追加优化表，则以字节为单位显示未压缩的表大小。
sotailindexessize	表中所有索引的总大小
sotailschemaaname	模式名称
sotailtablename	表名

gp_size_of_table_disk

此视图显示磁盘上表的大小。 所有用户均可访问此视图，但是非超级用户将只能看到他们有权访问的表

Table 38. gp_size_of_table_disk视图

列	描述
sotdoid	表的对象ID
sotdsize	表的大小（以字节为单位）。 该大小仅是主表的大小。 该大小不包括辅助对象（例如，超大（toast）属性）或AO表的其他存储对象。
sotdtoastsize	TOAST表的大小（超大属性存储）（如果有）。
sotdadditionalsize	反映追加优化（AO）表的段和块目录表的大小。
sotdschemaname	模式名称
sotdtablename	表名

gp_size_of_table_uncompressed

此视图显示追加优化（AO）表的未压缩表大小。 否则，将显示磁盘上的表大小。 使用此视图需要超级用户权限。

Table 39. gp_size_of_table_uncompressed视图

列	描述
sotuoid	表的对象ID
sotusize	如果表是压缩的AO表，则表示表的未压缩大小（以字节为单位）。 否则，为磁盘上表的大小。

列	描述
sotuschemaname	模式名称
sotutablename	表名

gp_disk_free

该外部表在活动segment主机上运行df（disk free）命令，并报告结果。非活动mirror不包括在计算中。使用此外部表需要超级用户权限。

Table 40. gp_disk_free外部表

列	描述
dfsegment	segment的内容ID（仅显示活动segment）
dfhostname	segment主机的主机名
dfdevice	设备名称
dfspace	segment文件系统中的可用磁盘空间（以千字节为单位）

检查数据分布不均

MPP数据库中的所有表都是分布式的，这意味着它们的数据被划分到系统中的所有segment中。如果数据分布不均匀，则查询处理性能可能会受到影响。以下视图可以帮助诊断表是否具有不均匀的数据分布：

- [gp_skew_coefficients](#)
- [gp_skew_idle_fractions](#)

Parent topic: [gp_toolkit管理模式](#)

gp_skew_coefficients

该视图通过计算存储在每个segment上的数据的变异系数（CV）来显示数据分布偏斜。所有用户均可访问此视图，但是非超级用户将只能看到他们有权访问的表

Table 41. gp_skew_coefficients视图

列	描述
skcoid	表的对象ID。
skcnamespace	定义表的命名空间。
skcrelname	表名。
skccoeff	变异系数（CV）计算为标准偏差除以平均值。它同时考虑了平均值和围绕数据序列平均值的变异性。值越低越好。较高的值表示较大的数据偏斜。

gp_skew_idle_fractions

该视图通过计算表扫描期间空闲系统的百分比来显示数据分布偏差，这是处理数据偏差的指标。所有用户均可访问此视图，但是非超级用户将只能看到他们有权访问的表

Table 42. gp_skew_idle_fractions视图

列	描述
sifoid	表的对象ID。

列	描述
sifnamespace	定义表的名称空间。
sifrelname	表名。
siffraction	在表扫描期间空闲的系统百分比，这表示数据分布不均匀或查询处理偏斜。例如，值0.1表示10%的歪斜，值0.5表示50%的歪斜，依此类推。歪斜率超过10%的表应评估其分配策略。

gpperfmon 数据库

gpperfmon 数据库是一个专用数据库，MPP segment 主机上的数据收集代理 程序将查询和系统统计信息保存在这个数据库中。

gpperfmon 数据库通过 gpperfmon_install 命令行工具创建。这个工具然后创建 gpmon 数据库角色并启用 master 和 segment 主机上的数据收集代理 程序。更多关于该工具使用和数据收集代理配置的信息，请参考 *MPP 数据库工具手册* 中的 gpperfmon_install 参考手册。

gpperfmon 数据库由三组表组成。它们分别用于在不同阶段捕捉查询和系统状态信息。

- `_now` 系列表存储当前系统指标，例如活动查询等。
- `_tail` 系列表用于在数据保存到 `_history` 系列表之前暂存数据。`_tail` 系列表仅供内部使用，不提供用户查询。
- `_history` 系列表用于存储历史数据。

`_now` 和 `_tail` 表的数据存储为文本文件，保存在 master 主机 文件系统中，通过

外部表在 gpperfmon 数据库中访问。history 系列表 是 gpperfmon 数据库的常规内存表。只有运行超过一定时间(默认是 20 秒)的查询才会 保存到历史数据表中。你可以通过设置 \$MASTER_DATA_DIRECTORY/gpperfmon/conf/gpperfmon.conf 中的 min_query_time 参数来修改这个门限值。设置为 0 将会保存所有查询历史信息。

Note: gpperfmon 不支持 ALTER SQL 命令。ALTER 查询不会被记录在 gpperfmon 历史查询表中。

历史查询 表按月进行分区。关于移除老分区的信息，请参见 [历史查询表分区保留](#)。

该数据库包含以下几类表：

- [database_*](#) 系列表存储一个 MPP 数据库实例的查询负载信息。
- [diskspace_*](#) 系列表存储磁盘 空间指标。
- [log_alert_*](#) 系列表存储 pg_log 中的错误和警告消息。
- [queries_*](#) 系列表存储高级查询 状态信息。
- [segment_*](#) 系列表存储 MPP 数据库 segment 实例的内存分配和统计信息。
- [socket_stats_*](#) 系列表存储一个 MPP 数据库实例中 socket 使用统计指标信息。注意: 这些表为将来使用保留，当前没有填充信息。
- [system_*](#) 系列表存储系统工具程序指标。

gpperfmon 数据库还包含下列视图：

- [dynamic_memory_info](#) 视图展示每个主机中所有 segments 的汇总, 以及每个主机中 动态内存使用量。
 - [memory_info](#) 视图展示来自 system_history 和 segment_history 表的每个主机的内存信息。
-

历史查询表分区保留

gpperfmon 数据库中的 历史查询 表按月进行分区。分区 在需要时会以两个月为增量自动添加。

\$MASTER_DATA_DIRECTORY/gpperfmon/conf/gpperfmon.conf 配置文件中的 partition_age 参数可以设置为要保留的每月最大分区数。添加新分区时，将自动删除早于指定值的分区。

partition_age 的默认值是 0, 这意味着管理员必须手动删除 不需要的分区。

警报日志处理和日志轮换

当 gp_gpperfmon_enable 服务期配置参数设置为 true 时, MPP 数据库 syslogger 会把警报消息写到一个 .csv 文件中。该文件位于 \$MASTER_DATA_DIRECTORY/gpperfmon/logs 目录。

通过修改 postgresql.conf 文件中的 gpperfmon_log_alert_level 服务期配置参数，可以将写入日志消息级别设置为 none, warning, error, fatal, 或者 panic。默认的消息基本是 warning。

日志存储的目录可以通过设置

\$MASTER_DATA_DIRECTORY/gpperfmon/conf/gpperfmon.conf 配置文件中的 log_location 配置变量来改变。

syslogger 每 24 小时，或在当前日志文件达到或超过 1MB 时轮换警报日志。

如果单条错误消息包含一个长 SQL 语句或者长堆栈信息时，轮换日志文件可能超过 1MB。此外，syslogger 会分批处理错误消息，每个日志记录过程分别对应一个单独的分块。日志块的大小取决于操作系统; 例如，在 Red Hat

Enterprise Linux 系统中,它是 4096 字节。如果许多 MPP 数据库会话同时生成错误消息,则在检查文件大小并触发日志轮换之前,日志文件可能会显着增长。

gpperfmon 数据收集过程

当 MPP 数据库启用 gpperfmon 支持下启动时,它将创建一个 gpmmon 代理进程。gpmmon 然后在 MPP 数据库集群的 master 主机和每个 segment 主机上启动一个 gpsmon 代理进程。MPP 数据库的 postmaster 进程监视 gpmmon 进程并在需要时重启该进程;gpmmon 进程监视和在需要时重启 gpsmon 进程。

gpmmon 进程以循环方式运行,并按一个可配置间隔检索 gpsmon 系列进程聚合的数据,并将其添加到 `_now` 和 `_tail` 外部数据库表的数据文件,然后存入 `_history` 常规内存数据库表中。

Note: gpperfmon 中的 `log_alert` 表流程不同于此,因为警报消息由 MPP 数据库系统 logger 发送,而不是通过 gpsmon 发送。参见 [警报日志处理和日志轮换](#) 了解更多信息。

`$MASTER_DATA_DIRECTORY/gpperfmon/conf/gpperfmon.conf` 配置文件中的两个配置参数控制着 gpmmon 被触发的频度:

- `quantum` 参数是以秒为单位的频率, gpmmon 依此频率向 segment 节点的 gpsmon 代理进程请求数据,并将获取到的数据添加到 `_now` 和 `_tail` 外部表数据文件中。 `quantum` 参数的有效设定值是 10, 15, 20, 30, 以及 60. 默认值是 15.
- `harvest_interval` 参数是以秒为单位的频率,依此频率将 `_tail` 表中的数据移动到 `_history` 表中。 `harvest_interval` 至少是 30. 默认值为 120.

参见 *MPP 数据库工具手册* 中的 `gpperfmon_install` 管理工具 参考手册, 查阅 gpperfmon 配置参数完整列表。

以下步骤描述了，当 gpperfmon 支持启用时，数据从 MPP 数据库进入 gpperfmon 的流程。

1. 在执行查询时, MPP 数据库查询调度程序和查询执行程序进程以 UDP 报文形式发送查询状态消息。 `gp_gpperfmon_send_interval` 服务器配置变量决定数据库发送这些消息的频率。默认为每秒发送一次。
2. 每个主机上的 `gpsmon` 进程接收 UDP 报文, 合并并汇总所包含的数据, 并添加其他主机指标，例如 CPU 和内存使用率。
3. `gpsmon` 进程继续累积数据，直到它们从 `gpmmon` 接收到转储命令为止。
4. `gpsmon` 进程通过发送其累积的状态数据和日志警报到一个 `gpmmon` 事件侦听线程来响应 `dump` 命令。
5. `gpmmon` 事件处理程序将指标保存到 master 主机的 `$MASTER_DATA_DIRECTORY/gpperfmon/data` 目录下的 .txt 文件。

在每一个 quantum 间隔 (默认为 15 秒), `gpmmon` 执行以下步骤:

1. 发送一个 `dump` 命令到 `gpsmon` 进程。
2. 收集和转换 .txt 文件 (保存在 the `$MASTER_DATA_DIRECTORY/gpperfmon/data` 目录中) 为 .dat 外部数据文件, 支持通过 `gpperfmon` 数据库中的 `_now` 和 `_tail` 外部表访问。
例如, 磁盘空间指标被添加到 `diskspace_now.dat` 和 `_diskspace_tail.dat` 带分隔符的文本文件中. 这些文本文件经由 `gpperfmon` 数据库中的 `diskspace_now` 和 `_diskspace_tail` 表访问.

在每一个 `harvest_interval` (默认为 120 秒), `gpmmon` 对每一个 `_tail` 文件执行以下步骤:

1. 重命名 `_tail` 为一个 `_stage` 文件.
 2. 创建一个新的 `_tail` 文件.
-

3. 把数据从 `_stage` 文件追加进 `_tail` 文件.
4. 允许一个 SQL 命令把数据从 `_tail` 外部表插入进相应的 `_history` 表中.
例如, `_database_tail` 外部表的内容被插进 `database_history` 常规 (内存) 表中.
5. Deletes the `_tail` file 在内容被加载进数据库表中后, 删除 `_tail` 文件.
6. 收集所有 `$MASTER_DATA_DIRECTORY/gpperfmon/logs` 目录下的 `gpdb-alert-*.csv` 文件 (除了 `syslogger` 已经打开并正在写入的最新文件) 到单个文件 `alert_log_stage` 中.
7. 把 `alert_log_stage` 文件内容加载进 `gpperfmon` 数据库的 `log_alert_history` 表中.
8. 清空 `alert_log_stage` 文件.

以下主题分别描述 `gpperfmon` 数据库每表中的内容。

- [database_* 表](#)
.....
- [diskspace_* 表](#)
.....
- [interface_stats_* 表](#)
.....
- [log_alert_* 表](#)
.....
- [queries_* 表](#)
.....
- [segment_* 表](#)
.....
- [socket_stats_* 表](#)
.....
- [system_* 表](#)
.....
- [dynamic_memory_info 视图](#)
.....
- [memory_info 视图](#)
.....

database_* 表

database_* 表存储一个 MPP 数据库实例的工作负载信息。这里一共有三张表，每张表都具有相同的结构(列):

- database_now 是一个外部表，其数据文件位于 \$MASTER_DATA_DIRECTORY/gpperfmon/data. 从数据采集代理程序获得数据以后，自动提交到 database_history 表之前，当前查询工作负载数据存储在 database_now 表中。
- database_tail 是一个外部表, 其数据文件位于 \$MASTER_DATA_DIRECTORY/gpperfmon/data. 它是一个过渡表，当数据已经从 database_now 中清除，但还没有提交到 database_history 表中时，暂存在这里。它通常仅包含数据几分钟时间。
- database_history 是一个常规表，用于存储历史查询工作负载数据。它已预先设置为按月分区。分区会根据需要以两个月为增量自动添加。

列	类型	说明
ctime	times tamp	该行的创建时间.
queries_t otal	int	采集数据时，MPP 数据库中的查询总数量.
queries_r unning	int	采集数据时，活动的查询数量.

列	类型	说明
queries_queued	int	采集数据时，资源组或资源队列中处于等待状态的查询数量(与当前使用的资源管理器相关).

Parent topic: [gpperfmon 数据库](#)

diskspace_* 表

diskspace_* 表存储磁盘空间指标。

- diskspace_now 是一个外部表, 其数据文件位于 \$MASTER_DATA_DIRECTORY/gpperfmon/data. 在数据从 gpperfmon 数据采集代理程序获得数据以后，自动提交到 diskspace_history 表之前，当前磁盘空间指标数据存储在 database_now 表中。
- diskspace_tail 是一个外部表, 其数据文件位于 \$MASTER_DATA_DIRECTORY/gpperfmon/data. 它是一个过渡表，当数据已经从 diskspace_now 中清除，但还没有提交到 diskspace_history表中时，暂存在这里。它通常仅包含数据几分钟时间。
- diskspace_history 是一个常规表，用于存储历史磁盘空间指标。它已预先设置为按月分区。分区会根据需要以两个月为增量自动添加。

列	类型	说明
ctime	timestamp(0) without time zone	磁盘空间测量时间.
hostname	varchar(64)	磁盘空间测量对应的主机名称.

列	类型	说明
Filesystem	text	磁盘空间测量对应的文件系统名称.
total_bytes	bigint	文件系统的总字节数.
bytes_used	bigint	文件系统中已使用字节数.
bytes_available	bigint	文件系统中可用的字节数.

Parent topic: [gpperfmon 数据库](#)

interface_stats_* 表

interface_stats_* 表存储 一个 MPP 数据库实例在每个活动网络接口上 的通信统计指标。

这些表为将来使用保留，当前没有填充信息。

一共有三张 interface_stats 表, 它们具有相同的结构(列):

- interface_stats_now 是一个外部表, 其数据文件位于 \$MASTER_DATA_DIRECTORY/gpperfmon/data.
- interface_stats_tail 是一个外部表, 其数据文件位于 in \$MASTER_DATA_DIRECTORY/gpperfmon/data. 它是一个存储接口统计指标的过渡表，当数据已经从 interface_stats_now 中清除，但还没有提交到 interface_stats_history表中时，暂存在这里。它通常仅包含数据几分钟时间。

- interface_stats_history 是一个常规表，用于存储历史接口统计指标。它已预先设置为按月分区。分区会根据需要以两个月为增量自动添加。

列	类型	说明
interface_name	string	接口名称. 例如: eth0, eth1, lo.
bytes_received	bigint	接收字节数.
packets_received	bigint	接收报文数.
receive_errors	bigint	接收数据时，发生的错误数.
receive_drops	bigint	接收数据时，丢弃报文的次数.
receive_fifo_errors	bigint	接收数据时，FIFO (先进先出) 错误发生的次数.
receive_frame_errors	bigint	接收数据时，帧错误发生的次数.
receive_compressed_packets	int	接受的压缩格式报文数.
receive_multicast_packets	int	接受的多播报文数.
bytes_transmitted	bigint	传输字节数.
packets_transmitted	bigint	传输报文数.
transmit_errors	bigint	数据传输时，发生的错误数.

列	类型	说明
transmit_drops	bigint	数据传输时，丢弃报文的次数.
transmit_fifo_errors	bigint	数据传输时，FIFO (先进先出) 错误发生的次数.
transmit_collision_errors	bigint	数据传输时，碰撞错误发生的次数.
transmit_carrier_errors	bigint	数据传输时，载波错误发生的次数.
transmit_compressed_packets	int	传输的压缩格式报文数.

Parent topic: [gpperfmon 数据库](#)

log_alert_* 表

log_alert_* 表存储 pg_log 错误和警报.

参考 [警报日志处理和日志轮换](#) 了解关于配置 gpperfmon 系统 logger 的更多信息.

一共有三张 log_alert 表, 所有这些表具有相同的列:

- log_alert_now 是一个外部表，它的数据存储在 \$MASTER_DATA_DIRECTORY/gpperfmon/logs 目录下的 .csv 文件中。在数据从 gpperfmon 代理自动提交到 log_alert_history 表期间，当前的 pg_log

错误和警报数据在 log_alert_now 中。 .

- log_alert_tail 是一个外部表，它的数据存储在 \$MASTER_DATA_DIRECTORY/gpperfmon/logs/alert_log_stage 中. 这是一个过渡表，当数据已经从 log_alert_now 中清除，但还没有提交到 log_alert_history 中时，暂存在这里。该表包括所有警报日志中的记录，但最新的除外。它通常仅包含数据几分钟时间。
- log_alert_history 是一个常规表，用于存储数据库范围内的历史错误和警告数据。 它已预先设置为按月分区。分区会根据需要以两个月为增量自动添加。

列	类型	说明
logtime	timestamp with time zone	此日志的时间戳
loguser	text	查询的用户
logdatabase	text	查询的用户
logpid	text	进程 ID
logthread	text	线程号
loghost	text	主机名或 IP 地址
logport	text	端口号
logsessiontime	timestamp with time zone	会话时间戳
logtransaction	integer	事务 ID
logsession	text	会话 ID
logcmdcount	text	命令数量
logsegment	text	Segment 编号
logslice	text	Slice 编号
logdistxact	text	分布式事务
loglocalxact	text	本地事务
logsubxact	text	子事务

列	类型	说明
logseverity	text	日志级别
logstate	text	日志状态
logmessage	text	日志信息
logdetail	text	详细信息
loghint	text	提示信息
logquery	text	查询内容
logquerypos	text	查询位置
logcontext	text	上下文信息
logdebug	text	调试
logcursorpos	text	光标位置
logfunction	text	函数信息
logfile	text	源代码文件
logline	text	源代码行
logstack	text	堆栈信息

Parent topic: [gpperfmon 数据库](#)

queries_* 表

queries_* 系列表存储高级查询状态信息.

tmid, ssid 和 ccnt 列是唯一标识特定查询的组合键.

一共有三张查询表，所有查询表都具有相同的列:

- queries_now 是一个外部表, 其数据文件位于 \$MASTER_DATA_DIRECTORY/gpperfmon/data. 在数据从 gpperfmon 代理自动提交到 queries_history 表期间，当前查询状态存储在 queries_now 中。
- queries_tail 是一个外部表, 其数据文件位于 \$MASTER_DATA_DIRECTORY/gpperfmon/data. 它是一个过渡表。当数据已经从 queries_now 中清除，但还没有提交到 queries_history 时，查询状态数据保存在这里。它通常仅包含数据几分钟时间。
- queries_history 是存储历史查询状态数据的常规表。 它被预定义为按月进行分区. 分区会根据需要以两个月为增量自动添加。

列	类型	说明
ctime	timestamp	该行的创建时间.
tmid	int	特定查询的时间标识符. 与特定查询关联的所有记录具有相同的 tmid.
ssid	int	与 gp_session_id 关联的会话 ID. 与特定查询关联的所有记录具有相同的 ssid.
ccnt	int	与 gp_command_count 关联的命令号. 与特定查询关联的所有记录具有相同的 ccnt.

列	类型	说明
use rna me	va rc h ar (6 4)	发出查询的 MPP 角色名称.
db	va rc h ar (6 4)	查询的数据库名称.
cost	int	在此版本中未实现.
tsu bmi t	ti m es ta m p	查询的提交时间.
tsta rt	ti m es ta m p	查询的开始时间.

列	类型	说明
tfinish	timestamp	查询的结束时间.
status	varchar(64)	查询状态 -- start, done, or abort.
rows_out	bigint	查询输出的行数.
cpu_elapsed	bigint	执行该查询的全部 segments 的全部进程的 CPU 时间 (单位: 秒). 它是从数据库系统中所有活动的主 segments 获取的 CPU 时间总和。 请注意，如果查询运行时间短于 quantum 的值，则该值记录为 0。即使查询运行时间大于 min_query_time 值, 但是低于 quantum 的值, 也会记录为 0。

列	类型	说明
cpu_cur_pct	float	当前执行此查询的所有进程的 CPU 平均百分比。对在每个 segment 上运行的所有进程的百分比求平均值，然后计算所有这些值的平均值来获得该指标。 当前的 CPU 平均百分比在 history 数据和 tail 数据中始终为零。
skew_cpu	float	显示该查询在系统中的处理偏斜量。当一个 segment 对查询执行的处理量不成比例时，就会发生处理 / CPU 偏斜。此值是此查询所有 segment 中 CPU % 度量的方差系数乘以 100。例如，.95 的值表示为 95。
skew_rows	float	显示系统中的行偏斜量。当一个 segment 产生的查询行数不成比例时，就会发生行偏斜。该值是该查询所有 segment 的 rows_in 指标的方差系数乘以100。例如，.95 的值显示为 95。
query_hash	bigint	在此版本中未实现.
query_text	text	该查询的 SQL 文本.
query_plan	text	查询计划文本. 在此版本中未实现.

列	类型	说明
app licat ion _na me	va rc h ar (6 4)	应用程序的名称.
rsq na me	va rc h ar (6 4)	如果基于资源队列的资源管理方案被使用，则此列为资源队列的名称.
rqp prio rity	va rc h ar (6 4)	如果基于资源队列的资源管理方案被使用，则此列为查询优先级 - - max, high, med, low, 或 min.

Parent topic: [gpperfmon 数据库](#)

segment_* 表

The segment_* 包含 MPP 数据库 segment 实例的内存分配统计信息。它跟踪每个特定 segment 实例上全部 postgres 进程的内存消费数量, 以及由当前资源

管理方案(基于资源组或基于资源队列)设置的每个 segment 可用的剩余内存数量. 参见 *MPP 数据库管理员手册* 了解更多关于资源管理方案的详细信息.

一共有三张 segment 表, 每张表都具有相同的结构(列):

- segment_now 是一个外部表, 其数据文件位于 \$MASTER_DATA_DIRECTORY/gpperfmon/data. 在数据从 gpperfmon 数据采集代理程序获得以后, 自动提交到 segment_history 表之前, 当前系统使用指标数据存储在 segment_now 表中。
- segment_tail 是一个外部表, 其数据文件位于 \$MASTER_DATA_DIRECTORY/gpperfmon/data. 它是一个内存分配信息的过渡表, 当数据已经从 segment_now 中清除, 但还没有提交到 segment_history 表中时, 暂存在这里。它通常仅包含数据几分钟时间。
- segment_history 是一个常规表, 用于存储历史内存分配指标。它已预先设置为按月分区。分区会根据需要以两个月为增量自动添加。

每个特定 segment 实例通过 hostname 和 dbid (gp_segment_configuration 系统目录表中的 segment 唯一标识) 定义.

列	类型	说明
ctime	timestamp(0) (without time zone)	该行的创建时间.
dbid	int	segment ID (gp_segment_configuration 中的 dbid).
hostname	charvar (64)	segment 主机名称.

列	类型	说明
dynamic_memory_used	bigint	当前 segment 上分配给查询进程的动态内存数量(单位: 字节).
dynamic_memory_available	bigint	在到达当前资源管理方案(基于资源组或基于资源队列)设置限制前，segment 上还可分配的动态内存数量(单位: 字节).

参见 memory_info 视图和 dynamic_memory_info 视图了解更多主机上内存分配汇总和使用信息.

Parent topic: [gpperfmon 数据库](#)

socket_stats_* 表

socket_stats_* 表存储一个 MPP 数据库实例中 socket 使用统计指标。一共有三张表，所有查询表都具有相同的列:

这些表为将来使用保留，当前没有填充信息。

- socket_stats_now 是一个外部表，它的数据存储在 \$MASTER_DATA_DIRECTORY/gpperfmon/data.
- socket_stats_tail 是一个外部表，它的数据存储在 \$MASTER_DATA_DIRECTORY/gpperfmon/data. 这是一个过渡表，当数据已经从 socket_stats_now 中清除，但还没有提交到 socket_stats_history 中时，暂存在这里。它通常仅包含数据几分钟时间。

- `socket_stats_history` 是一个常规表，用于存储 socket 历史统计指标。它已预先设置为按月分区。分区会根据需要以两个月为增量自动添加。

列	类型	说明
<code>total_sockets_used</code>	<code>int</code>	系统中的 socket 总数.
<code>tcp_sockets_inuse</code>	<code>int</code>	使用中的 TCP socket 数量.
<code>tcp_sockets_orphan</code>	<code>int</code>	孤儿 TCP socket 数量.
<code>tcp_sockets_timewait</code>	<code>int</code>	Time-Wait 状态的 TCP socket 数量.
<code>tcp_sockets_alloc</code>	<code>int</code>	已分配的 TCP socket 数量.
<code>tcp_sockets_memusage_inbytes</code>	<code>int</code>	TCP socket 消耗的内存总量.
<code>udp_sockets_inuse</code>	<code>int</code>	使用中的 UDP socket 数量.
<code>udp_sockets_memusage_inbytes</code>	<code>int</code>	UDP socket 消耗的内存总量.
<code>raw_sockets_inuse</code>	<code>int</code>	使用中的 RAW socket 数量.
<code>frag_sockets_inuse</code>	<code>int</code>	使用中的 FRAG socket 数量.
<code>frag_sockets_memusage_inbytes</code>	<code>int</code>	FRAG socket 消耗的内存总量.

Parent topic: [gpperfmon 数据库](#)

system_* 表

system_* 表存储系统使用指标。 这里一共有三张表，每张表都具有相同的结构(列):

- system_now 是一个外部表, 其数据文件位于 \$MASTER_DATA_DIRECTORY/gpperfmon/data. 在数据从 gpperfmon 数据采集代理程序获得以后，自动提交到 system_history 表之前， 当前系统使用指标数据存储在 system_now 表中。
- system_tail 是一个外部表, 其数据文件位于 \$MASTER_DATA_DIRECTORY/gpperfmon/data. 它是一个过渡表，当数据已经从 system_now 中清除，但还没有提交到 system_history 表中时，暂存在这里。它通常仅包含数据几分钟时间。
- system_history 是一个常规表， 用于存储历史系统使用指标。 它已预先设置为按月分区。 分区会根据需要以两个月为增量自动添加。

列	类型	说明
ctime	timestamp	该行的创建时间.
hostname	varchar(64)	与系统指标相关的 Segment 或 master 主机名称.
mem_total	bigint	主机全部系统内存(单位: 字节).
mem_used	bigint	主机已使用的系统内存(单位: 字节).
mem_actual_used	bigint	主机实际已使用的内存(单位: 字节) (不包括缓存和缓冲保留内存).
mem_actual_free	bigint	主机空闲内存(单位: 字节) (不包括缓存和缓冲保留内存).
swap_total	bigint	主机全部交换内存(单位: 字节).
swap_used	bigint	主机已使用的交换内存(单位: 字节).
swap_page_in	bigint	交换进的页数量.

列	类型	说明
swap_page_out	bigint	交换出的页数量.
cpu_user	float	MPP 系统用户的 CPU 使用量.
cpu_sys	float	主机 CPU 使用量.
cpu_idle	float	指标收集时空闲的 CPU 容量.
load0	float	前一分钟的 CPU 平均负载.
load1	float	前五分钟的 CPU 平均负载.
load2	float	前十五分钟的 CPU 平均负载.
quantum	int	此指标的指标采集间隔.
disk_ro_rate	bigint	每秒磁盘读取操作次数.
disk_wo_rate	bigint	每秒磁盘写入操作次数.
disk_rb_rate	bigint	磁盘每秒读取的字节数.
disk_wb_rate	bigint	磁盘每秒写入的字节数.
net_rp_rate	bigint	系统网络每秒读取的报文数量.
net_wp_rate	bigint	系统网络每秒写入的报文数量.
net_rb_rate	bigint	系统网络每秒读取的字节数.
net_wb_rate	bigint	系统网络每秒写入的字节数.

Parent topic: [gpperfmon 数据库](#)

dynamic_memory_info 视图

dynamic_memory_info 视图展示一个 segment 主机上所有 segment 实例已用内存和可用动态内存的总和。 动态内存指 MPP 数据库实例在启动取消进程前，允许一个 segment 实例上查询进程使用的内存(超过该数量则启动取消进程)。这个限制有当前所使用的资源管理方案决定 (基于资源组或基于资源队列), 按每个 segment 进行评估.

列	类型	说明
ctime	timest amp(0) witho ut time zone	在 segment_history 表中的创建时间.
hostna me	varch ar(64)	与这些系统内存指标相关的 Segment 或 master 主机名称.
dynam ic_me mory_ used_ mb	numer ic	segment 节点上分配给查询进程的动态内存大小(单位: MB) .
dynam ic_me mory_ availa ble_m b	numer ic	segment 节点上可分配给查询进程的动态内存大小(单位: M B). 注意这个值是所有 segment 节点上可分配内存的总和. 虽然报告了这么多可分配内存，也可能一个或多个 segme nt 在主机上分配的内存已经超过了它们的限制。

Parent topic: [gpperfmon 数据库](#)

memory_info 视图

memory_info 视图展示 system_history 表和 segment_history 表中的每个主机上的内存信息. 它允许管理员比较 segment 主机上的总可以内存, 总使用内存, 以及查询进程已使用的动态内存.

列	类型	说明
ctime	timestamp(0) without time zone	segment_history 表中该行数据的创建时间.
hostname	varchar(64)	这些系统内存指标相关的 Segment or master 主机名称.
mem_total_mb	numeric	segment 主机上的总系统内存(单位: MB).
mem_used_mb	numeric	segment 主机上的已使用内存(单位: MB).
mem_actual_used_mb	numeric	segment 主机上的实际使用内存(单位: MB).
mem_actual_free_mb	numeric	segment 主机上的实际可用内存(单位: MB).
swap_total_mb	numeric	segment 主机上的总交换内存(单位: MB).

列	类型	说明
swap_used_mb	numeric	segment 主机上的已使用交换内存(单位: MB).
dynamic_memory_used_mb	numeric	segment 主机上分配给查询进程的动态内存(单位: MB).
dynamic_memory_available_mb	numeric	segment 主机上查询进程还可分配的动态内存(单位: MB). 注意这个值是一个主机上所有 segment 的可用内存总和. 即使报告为可用内存, 也可能主机上的一个或多个 segment 已经超过了它们自身的内存限制.

Parent topic: [gpperfmon 数据库](#)

MPP 数据库数据类型

MPP 数据库有一套丰富的本地数据类型供用户使用。用户也可以使用 CREATE TYPE 命令定义新的数据类型。本参考手册对所有内置数据类型进行说明。除了这里列出的类型外，还有一些内部使用的数据类型，例如 *oid* (对象标识符)，但这些不在本手册中介绍。

在 contrib 目录中的可选模块也可能安装新的数据类型。例如 *hstore* 模块引入了一种新数据类型和 连带函数来与 key-value 键值对一起工作。参见 [hstore](#)。
citext 模块添加了一种大小写不敏感的文本数据类型。参见 [citext](#)。

下面的数据类型在SQL中使用: *bit*, *bit varying*, *boolean*, *character varying*, *varchar*, *character*, *char*, *date*, *double precision*, *integer*, *interval*, *numeric*,

decimal, *real*, *smallint*, *time* (带时区或不带时区), 以及 *timestamp* (带时区或不带时区)。

每一种数据类型都有一种外部表示格式，这由数据类型的输入函数和输出函数决定。许多内置数据类型有明显的外部格式。然而，几种特别的数据类型要么只存在于 PostgreSQL（以及 MPP 数据库），例如 *geometric paths*，要么有几种可能的表示格式，例如时间和日期类型。一些输入和输出函数是不可逆转的，也就是说，相比较原始输入，输出函数的结果可能存在精度损失。

表 1. MPP 数据库内置数据类型

名称	别名	大小	范围	说明
bigint	int8	8 字节	-9223372036854775808 到 9223372036854775807	大范围整数
bigserial	serial8	8 字节	1 到 9223372036854775807	大范围自增型整数
bit [(n)]		<i>n</i> 位	位串常量	定长位串
bit varying [(n)] ¹	varbit	实际位数量	位串常量	变长位串
boolean	bool	1 字节	true/false, t/f, yes/no, y/n, 1/0	逻辑布尔值(true/false)
box		32 字节	((x1,y1),(x2,y2))	平面内长方形 - 不能在分布键列使用中。

名称	别名	大小	范围	说明
bytea ¹		1 字节 + 二进制字符串大小	8比特字符 序列	变长二进制字符串
character [(n)] ¹	char [(n)]	1 字节 + <i>n</i>	最长 <i>n</i> 个字符的字符串	定长, (输入不足时自动用) 空白填充
character varying [(n)] ¹	varchar [(n)]	1 字节 + 字符串大小	最长 <i>n</i> 个字符的字符串	有限制的变长字符串
cidr		12 或 24 字节		IPv4 和 IPv6 网络地址
circle		24 字节	<(x,y),r> (中心和半径)	平面内圆 - 不能在分布键列使用中。
date		4 字节	4713 BC - 294, 277 AD	日历日期 (年, 月, 日)
decimal [(p, s)] ¹	numeric [(p, s)]	可变	无限制	用户指定精度, 精度确切
double precision	float8 float	8 字节	15位十进制数字精度	可变精度, 精度不确切

名称	别名	大小	范围	说明
inet		12 或 24 字节		IPv4 和 IPv6 主机和网络
integer	int, int4	4 字节	-2147483648 to +2147483647	常用整数类型
interval [fields][(p)]		16 字节	-178000000 年到 178000000 年	时间跨度(间隔)
json		1 字节 + json 大小	任意长度 json	不定长（无限制）
jsonb		???	???	???
lseg		32 字节	((x1,y1),(x2,y2))	平面内线段 - 不能在分布键列使用中。
macaddr		6 字节		MAC 地址
money		8 字节	-92233720368547758.08 到 +92233720368547758.07	货币数量
path ¹		16+16n 字节	[(x1,y1),...]	平面内几何路径 - 不能在分布键列使用中。
point		16 字节	(x,y)	平面内几何点 - 不能在分布键列使用中。

名称	别名	大小	范围	说明
polygon		40+1 6n 字节	((x1,y1),...)	平面内封闭几何路径 - 不能在分布键列使用中。
real	float4	4 字节	6 十进制数字精度	可变精度，精度不确切
serial	serial4	4 字节	1 到 2147483647	自增型整数
smallint	int2	2 字节	-32768 到 +32767	小范围整数
text ¹		1 字节 + 字符串长度	任意长度字符串	不定长（无限制）
time [(p)] [without time zone]		8 字节	00:00:00[.000000] - 24:00:00[.000000]	一天内的时间
time [(p)] with time zone	timetz	12 字节	00:00:00+1359 - 24:00:00-1359	一天内的时间, 带时区
timestamp [(p)] [without time zone]		8 字节	4713 BC - 294,277 AD	日期和时间
timestamp [(p)] with time zone	timestamp tz	8 字节	4713 BC - 294,277 AD	日期和时间, 带时区

名称	别名	大小	范围	说明
uuid		32 字节		RFC 4122, ISO/IEC 9834-8:2005 标准定义的通用唯一标识(UUID)
xml ¹		1 字节 + xml 大小	任意长度 xml	变长，无限制
txid_snapshot				用户级事务 ID 快照

- [日期/时间类型](#)
- [伪类型](#)
- [全文搜索类型](#)
- [范围类型](#)

父主题：[MPP数据库参考手册](#)

¹ 对于变长数据类型，如果大于或等于 127 字节，需要(除数据本身外)额外 4 字节的存储空间（小于 127 字节时额外只需要 1 字节存储空间）。

日期/时间类型

MPP 完整支持全部 SQL 日期和时间类型, 参见 [Table 1](#)。对于这些数据类型可用的操作说明, 参见 PostgreSQL 文档 [日期/时间函数和运算符](#)。日期根据罗马日历计算，在日历引入之前的年也这样计算 (参见 PostgreSQL 文档 [日期单位的历史](#)了解更多信息)。

Table 1. 日期/时间类型

名称	存储大小	描述	最小值	最大值	精度
timestamp [(p)][without time zone]	8 字节	包括时间和日期 (无时区)	4713 BC	294276 AD	1 微秒 / 14 位数字
timestamp [(p)]with time zone	8 字节	包括时间和日期 (有时区)	4713 BC	294276 AD	1 微秒 / 14 位数字
date	4 字节	日期 (无一天内的时间)	4713 BC	5874897 AD	1 天
time [(p)][without time zone]	8 字节	一天内的时间 (无日期和时区)	00:00:00	24:00:00	1 微秒 / 14 位数字
time [(p)]with time zone	12 字节	一天内的时间, 无日期, 有时区	00:00:00+1459	24:00:00-1459	1 微秒 / 14 位数字
interval [fields][(p)]	16 字节	时间间隔	-178000000 年	178000000 年	1 微秒 / 14 位数字

Note: SQL 标准要求 timestamp 与 timestamp without time zone 写法一致, MPP 实现了这一点。timestamp_{tz} 是 timestamp with time zone 的缩写; 这是 PostgreSQL 的一点扩展。

time, timestamp, 和 interval 接受一个可选的精度值 *p* , 来指定秒的小数部分数字。默认情况下, 没有明确精度指定。对于 timestamp 和 interval 类型, 允许的 *p* 值范围为 0 到 6。

Note: 当 timestamp 值按八字节整数存储(当前默认), 精度可全范围(0 到 6 位)任意设置。当 timestamp 值按双精度浮点数存储(一个过时的编译选项), 精度有效值可能小于 6 位。timestamp 值存储的是从 2000-01-01 午夜开始的秒的数值。当 timestamp 值按浮点数实现时, 2000-01-01 附近的一些年可以达到微秒精

度，但是距离这个日期稍远的一些时间精度就会有所降低。注意：浮点存储的 timestamp 值允许更大的日期范围，比上表中给出的要大，即可以表示从 4713 BC 到 5874897 AD 的日期。

同样的编译选项也决定 time 和 interval 类型值存储为浮点数还是八字节整数。当存储位浮点数的时候，当时间间隔增大时，interval 的精度有所降低。

对于 time 类型，使用八字节整数存储时，允许的 p 值范围从 0 到 6；使用浮点数存储时，允许的 p 值范围从 0 到 10。

另外，interval 数据类型还有一个附加选项，可以通过书写以下时间单位后缀(*fields*)来来限制存储内容：

```
YEAR  
MONTH  
DAY  
HOUR  
MINUTE  
SECOND  
YEAR TO MONTH  
DAY TO HOUR  
DAY TO MINUTE  
DAY TO SECOND  
HOUR TO MINUTE  
HOUR TO SECOND  
MINUTE TO SECOND
```

注意：如果同时指定 *fields* 与 p 参数，*fields* 必须包含 SECOND，因为精度参数 p 只对秒起作用。

time with time zone 类型由 SQL 标准定义，但是定义给出的属性导致人们怀疑它的有用性。大部分情况下，复合使用 date, time, timestamp without time zone, 以及 timestamp with time zone 已经提供了任何应用程序所需的完整日期/时间功能。

abstime 和 reltime 是内部使用的低精度数据类型。不建议你在应用程序中使用；这些内部数据类型可能在未来的某一个版本中消失。

日期/时间输入

几乎任何合理的日期/时间输入格式都可以被接受。包括 ISO 8601, SQL-兼任格式, 传统 POSTGRES 格式, 等等。对于一些格式, 数据输入中的年、月、日的顺序非常含糊，所以支持指定这些字段的期望顺序。参见 [DateStyle](#) 通过参数 MDY 指定 月-日-年 格式, DMY 指定 日-月-年 格式，或者 YMD 指定 年-月-日 格式。

MPP 在处理日期/时间输入上比 SQL 标准要求的更加弹性。参见 PostgreSQL 文档 [附录 B. 时间/日期支持](#)，了解时间/日期输入的确切解析规则，以及如何识别文本字段，包括月份, 星期和时区。

记住，任何日期或时间字面值需要用单引号引起来, 就像字符串一样. SQL 需要这样的语法

```
type[(p)] 'value'
```

这里，*p* 是一个可选精度参数，指定秒的小数位数。它对以下类型起作用，time, timestamp, 以及 interval 类型。允许设定的值范围已经在上面文档中说明；如果没有指定精度，默认为字面值的精度。

日期

[Table 2](#) 一些可能的 date 类型输入格式.

Table 2. 日期输入

举例	说明
1999-01-08	ISO 8601; 1 月 8 日 (推荐格式)

举例	说明
January 8, 1999	对任何 日期风格 都很明确的输入模式
1/8/1999	MDY 模式：1 月 8 日；DMY 模式：8 月 1 日
1/18/1999	MDY 模式：1 月 18 日；其他模式：无效
01/02/03	MDY 模式：2003 年 1 月 2 日；DMY 模式：2003 年 2 月 1 日；YMD 模式：2001 年 2 月 3 日
1999-Jan-08	任何模式：1999 年 1 月 8 日
Jan-08-1999	任何模式：1999 年 1 月 8 日
08-Jan-1999	任何模式：1999 年 1 月 8 日
99-Jan-08	YMD 模式：1999 年 1 月 8 日, 其他：错误
08-Jan-99	YMD 模式：错误；其他模式：1999 年 1 月 8 日
Jan-08-99	YMD 模式：错误；其他模式：1999 年 1 月 8 日
19990108	ISO 8601; 任何模式：1999 年 1 月 8 日
990108	ISO 8601; 任何模式：1999 年 1 月 8 日
1999.008	年和一年中的天
J2451187	儒略日期（天文学常用）
January 8, 99 BC	公元前 99 年 1 月 8 日

时间

时间类型包括 `time [ ( p ) ] without time zone` 和 `time [ ( p ) ] with time zone`。
`time` 与 `time without time zone` 等价。

这些类型的有效的输入格式由时间加上一个可选的时区. (参见 [Table 3](#) 和 [Table 4](#).) 如果在输入中给 `time without time zone` 类型指定了时区, 则时区会被忽略。你如果指定了一个日期, 也会被忽略, 除非你用了包含夏令时的时区, 例如 `America/New_York`. 这种情况下, 指定日期是必要的, 因为需要决定当前是否是标准时间, 还是夏令时时间. 使用 `time with time zone` 类型时, 合适的时区偏移会被记录.

Table 3. 时间输入

举例	说明
04:05:06.789	ISO 8601
04:05:06	ISO 8601
04:05	ISO 8601
040506	ISO 8601
04:05 AM	与 04:05 相同; AM 不影响值
04:05 PM	与 16:05 相同; 小时值必须 <= 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	时区用缩写指定
2003-04-12 04:05:06 America/New_York	时区用全称指定

Table 4. 时区输入

举例	说明
PST	缩写 (太平洋标准时间)
America/New_York	时区全称
PST8PDT	POSIX-样式时区格式
-8:00	ISO-8601 偏移(PST)
-800	ISO-8601 偏移(PST)
-8	ISO-8601 偏移(PST)
zulu	军方缩写(UTC)
z	zulu 的短格式

参考 [时区](#) 了解更多时区格式输入信息.

时间戳

时间戳的有效输入格式由以下几个部分组成：日期，时间，时区(可选), 接着可选的 AD 或 BC. (另外, AD / BC 也可以出现在时区之前, 但不是推荐的顺序.) 因而: 1999-01-08 04:05:06 和: 1999-01-08 04:05:06 -8:00 都是有效的时间戳值, 它们满足 ISO 8601 标准要求. 另外, 常用格式: January 8 04:05:06 1999 PST 也被支持.

SQL 标准中 timestamp without time zone 与 timestamp with time zone 字面值的差异主要体现在时间后面由一个 + 或 - 号标识的时区. 因此, 根据标准, `TIMESTAMP '2004-10-19 10:23:54'` 是一个 timestamp without time zone 类型 字面值, 而 `TIMESTAMP '2004-10-19 10:23:54+02'` 是一个 timestamp with time zone 类型的字面值. MPP 在确定类型之前从不检查字面值内容, 所以上面两个字面值都会被认为是 timestamp without time zone 类型. 为确保当作 timestamp

with time zone 类型对待, 请像这样给出明确类型: `TIMESTAMP WITH TIME ZONE '2004-10-19 10:23:54+02'` 对于一个已经确定为 `timestamp without time zone` 类型的字面值, MPP 会抛弃时区相关信息. 也就是说, 结果时间戳仅仅根据日期/时间确定, 不会根据时区进行调整.

对于 `timestamp with time zone` 类型, 内部值总是按照 UTC (统一协调时间, 通常叫做格林威治时间, GMT) 存储. 带有明确时区的输入值会使用合适的时区偏移转为 UTC 时间. 如果输入字符串中没有指定时区, 会假定为当前系统的 `TimeZone` 参数, 并使用这个参数的偏移转换为 UTC 时间.

当输出一个 `timestamp with time zone` 类型值, 总是根据当前 `timezone` 时区值进行转换, 并显示为本地时间. 要想以另一个时区查看时间, 要么改变 `timezone` 设置, 要么使用 `AT TIME ZONE` 构造 (参见 PostgreSQL 文档的[AT TIME ZONE](#)).

在 `timestamp without time zone` 与 `timestamp with time zone` 间转换通常假定 `timestamp without time zone` 中的值应该是以 `timezone` 为时区的本地时间. 当然, 也可以用 `AT TIME ZONE` 指定一个不同的时区.

特殊值

为了方便, MPP 支持几种特殊的日期/时间输入值格式, 参见 [Table 5](#). `infinity` 和 `-infinity` 值是用于系统内部的特殊表示, 会被直接显示; 但其他都只是些快捷记号, 会在读取的时候被转换为常规的日期/时间值. (特别地, `now` 和相关字符串会被转换为读取时的当前特定时间值.) 所有这些常量值在 SQL 命令中都需要用单引号引起来.

Table 5. 特殊日期/时间输入

输入字符串	有效类型	说明
epoch	date, timestamp	1970-01-01 00:00:00+00 (Unix 系统时间零点)

输入字符串	有效类型	说明
infinity	date, timestamp	比任何时间戳都晚的时间
-infinity	date, timestamp	比任何时间戳都早的时间
now	date, time, timestamp	当前事务的开始时间
today	date, timestamp	今天午夜
tomorrow	date, timestamp	明天午夜
yesterday	date, timestamp	昨天午夜
allballs	time	00:00:00.00 UTC

下面这些 SQL-兼容函数也可以用来获取相应类型的当前时间: CURRENT_DATE, CURRENT_TIME, CURRENT_TIMESTAMP, LOCALTIME, LOCALTIMESTAMP. 后面四个可接受一个可选的亚秒级精度参数. (参见 PostgreSQL 文档的[当前日期/时间](#).) 注意 有一些 SQL 函数在输入字符串中 不被接受.

日期/时间输出

日期/时间类型的输出格式可以被设置为以下四种样式: ISO 8601, SQL (Ingres), 传统 POSTGRES (Unix date 格式), 或者 German. 默认是 ISO 格式. (SQL 标准需要使用 ISO 8601 格式. SQL 输出格式这个名字是由于偶然的历史原因.) [Table 6](#) 给出了各种输出格式的例子. date 和 time 类型的输出格式通常为对应例子中的日期或时间部分. 然而, POSTGRES 格式的纯日期输出用的是 ISO 格式.

Table 6. 日期/时间输出格式

样式规范	说明	举例
ISO	ISO 8601, SQL 标准	1997-12-17 07:37:16-08

样式规范	说明	举例
SQL	传统样式	12/17/1997 07:37:16.00 PST
Postgres	原始样式	Wed Dec 17 07:37:16 1997 PST
German	区域样式	17.12.1997 07:37:16.00 PST

Note: ISO 8601 规定使用大写字母 T 隔开日期和时间. MPP 支持这个输入格式, 但是输出时使用空格而不是 T, 上面您已经看到了. 这是为了可读性, 同时与 RFC 3339 以及一些其他数据库系统保持一致.

在 SQL 和 POSTGRES 样式中, 如果指定 DMY 模式, 日会出现在月之前; 其他情况下月在日之前. (参见 [Table 2](#) 关于这些设置如何影响输入值解释.) [Table 7](#) 中的例子.

Table 7. 日期顺序约定

<i>datestyle</i> 设置	输入顺序	输出举例
SQL, DMY	<i>day / month / year</i>	17/12/1997 15:37:16.00 CET
SQL, MDY	<i>month / day / year</i>	12/17/1997 07:37:16.00 PST
Postgres, DMY	<i>day / month / year</i>	Wed 17 Dec 07:37:16 1997 PST

用户可以使用以下方式选择日期/时间样式: SET *datestyle* 命令, postgresql.conf 配置文件中的 *DateStyle* 参数, 或者服务器或客户端的 PGDATESTYLE 环境变量.

格式化函数 `to_char` (参见 [数据类型格式化函数](#)) 是一种格式化日期/时间输出的更加灵活的方式.

时区

时区和时区转换，不仅仅是地理位置问题，也受政治决定影响. 从 1900 年以来，世界上的时区稍微标准了一点, 但还是在持续的改变, 尤其是夏令时规则. MPP 使用广泛采纳的 IANA (Olson) 时区数据库来定义历史时区规则. 对于未来时间, 假定给定时区的现行已知规则会无限期持续.

MPP 保持与 SQL 标准兼容的典型用法定义. 然而, SQL 标准偶尔也有一点混淆时间和日期类型和能力. 两个明显的问题是:

1. 虽然 `date` 类型不能带时区, 但是 `time` 类型是可以的. 不与日期和时间关联, 时区在真实世界意义不大, 因为偏移变化与夏令时边界相关.
2. 默认时区是与 UTC 之间差异的常量数值. 因而, 当进行跨 DST 边界的日期/时间数值计算时, 可能采用夏令时.

为了解决这些困难, 当使用时区的时候, 我们推荐用同时包含日期和时间的日期/时间类型. 我们 不 推荐使用 `time with time zone` 类型(虽然 MPP 支持, 但主要是为了一个老应用, 同时保持与 SQL 标准兼容). 对于任何只包含日期或时间的类型, MPP 假定是你的本地时区(的时间或日期).

所有时区感知的日期和时间内部都存储为 UTC 值. 当被显示到客户端时, 才通过 `TimeZone` 配置参数的值转换为指定时区的日期或时间.

MPP 允许你以三种不同的形式指定时区:

1. 时区全称, 例如 `America/New_York`. 在 `pg_timezone_names` 视图里列出了所有可被识别的时区名称. 为了这个目的, MPP 使用广泛采纳的 IANA 的时区数据, 因此同样的时区名称也可以被很多其他软件识别.
 2. 另一个形式时时区缩写, 例如 `PST`. 对于 时区全称, 能够暗示夏令时规则, 而时区缩写这个规范只能表示与 UTC 直接的时区偏移. 可识别的缩写列在 `pg_timezone_abbrevs` 视图中. 你不能给时区缩写设置配置参数 `TimeZone` 或 `log_timezone`, 但你可以在时间/日期输入值中通过 `AT TIME ZONE` 操作符使用缩写.
 3. 除了时区全称和时区缩写外, MPP 也接受 POSIX-风格的时区规范, 形如 `STD offset` 或者 `STD offset DST`, 这里 `STD` 是一个时区缩写, `offset` 是 UTC
-

往西偏移的小时数值, *DST* 是一个可选的夏令时时区缩写, 假定为比给定偏移提前一个小时. 举个例子, 如果 EST5EDT 还没有被识别为时区名称, 它能被接受, 相当于美国东岸时间(United States East Coast time). 在这种语法中, 时区缩写可以是一串字母, 或者包围在尖括号中的任意字符串(<>). 当存在夏令时时区缩写时, 假定使用与 IANA 时区数据库中相同的夏令时转换规则. 在标准 MPP 安装时, 与 US/Eastern 相同, 因此 POSIX-风格时区规范遵守 USA 夏令时规则. 如果需要, 你也可以通过替换文件来调整这个行为.

总之, 这就是时区全称与缩写的不同: 缩写表示与 UTC 之间的偏移, 而全称还暗示了夏令时规则, 因此有两种可能的 UTC 偏移. 例如, 2014-06-04 12:00 America/New_York 表示纽约本地时间中午, 指定为东部夏令时间 Eastern Daylight Time (UTC-4). 因此 2014-06-04 12:00 EDT 指定了同样的时间常数. 但是 2014-06-04 12:00 EST 指定为东部标准时间 (UTC-5) 中午, 不管那天夏令时是否正常生效与否.

另外复杂情况是, 有些管辖范围使用同样的时区缩写表示不同的 UTC 偏移和不同的时间; 例如, 在 MSK 有些年头表示 UTC+3, 有些年头又表示 UTC+4. 对于这些缩写, MPP 根据日期来进行解释(对于未来时间, 根据最近的情况解释); 但是, 就像上面 EST 这个例子, 这不一定与那个日期的本地时间相同.

应该小心 POSIX-风格时区特征很容易导致错误输入被默默接受, 因为很难检查时区输入的合理性. 例如, SET TIMEZONE TO FOOBAR0 能够正常工作, 让系统有效使用相当特别的 UTC 缩写. 另一个要记住的问题是在 POSIX 时区名称中, 正数被用在(相对于)格林威治时间. 其他任何地方, MPP 遵守 ISO-8601 约定, 正时区偏移都是(相对于)格林威治时间.

所有情况中, 失去名称和缩小都是大小写无关的.

不论时区名称还是缩写, 都没有硬编码进服务程序; 我们是从配置文件.

TimeZone 配置参数可以在文件中设定, 或者其他任何标准设置配置参数的方式. 另外也有一个特殊方式来设置:

1. SQL 命令 SET TIME ZONE 可以设置当前会话的时区. 这是比 SET TIMEZONE TO 更符合 SQL 兼容规范的一种形式.
-

- PGTZ 环境变量被 libpq 客户端用来, 在初始化与服务器的连接时, 发送 SET TIME ZONE 命令.

时间段输入

时间段 值可以被写成下面的详细语法:

```
@ quantity unit quantity unit... direction
```

这里 *quantity* 是一个有符号数值; *unit* 是 microsecond, millisecond, second, minute, hour, day, week, month, year, decade, century, millennium, 或者这些单位的缩写或复数形式; *direction* 可以是 ago 或者 为空. at 符号 (@) 是一个可选的无用符号. 不同单位的值会被隐式的加在一起(考虑符号和单位). ago 导致所有单位的值都变号. 如果 [IntervalStyle](#) 设置为了 postgres_verbose, 这个语法也用于时间段输出.

天, 小时, 分, 和秒的数量都可以不用单位指定. 例如, '1 12:59:10' 与这个 '1 day 12 hours 59 min 10 sec' 是一样的. 另外, 年和月可以通过一个划线(减号)来指定; 例如 '200-10' 读出来与 '200 years 10 months' 一样. (这些短格式事实上是 SQL 标准仅允许的格式. 当 *IntervalStyle* 设置给 sql_standard 时, 也按这样的格式输出.)

时间段值也可以写成 ISO 8601 时间段的格式, 用标准中的 4.4.3.2 节内容 format with designators, 或者用 4.4.3.3 节说明的 alternative format. format with designators 看起来像这个样子:

```
P quantity unit quantity unit... T quantity unit...
```

这个时间段字符串必须以 P 开头, 可以包含一个 T 来隔开时间部分. 可以使用的缩写列在 [Table 8](#) 中. 单位可以忽略, 也可以按任意顺序指定, 但是时间单位必须出现在 T 后面. 特别地, M 的含义依赖于它出现在 T 之前还是之后.

Table 8. ISO 8601 时间段单位缩写

缩写	含义
Y	年
M	月 (在日期部分)
W	周
D	天
H	小时
M	分 (在时间部分)
S	秒

另一种格式:

P years - months - days T hours : minutes : seconds

字符串必须以 P 开头, 用字母 T 隔开时间段的日期和时间部分. 数值按类似于 ISO 8601 日期格式书写.

当书写一个带有 *fields* 格式的时间段常量, 或者当把一个字符串赋给带有 *fields* 格式的时间段列时, 没有标志单位的数值根据 *fields* 解释. 例如 INTERVAL '1' YEAR 读成 1 年, 而 INTERVAL '1' 意味着 1 秒. 另外, 带有 to the right 字段值中 *fields* 允许的右边最不重要的字段会被简单的忽略. 例如, 写 INTERVAL '1 day 2:03:04' HOUR TO MINUTE 结果或丢弃秒部分, 而不会丢掉天的部分.

根据 SQL 标准, 时间段中所有部分必须有同样的符号, 开头的负号会对所有部分起作用; 例如, 在时间段字面值 '-1 2:03:04' 中, 负号应用于天, 以及小时/分/秒部分. MPP 允许各部分有不同的符号, 并且传统认为文本中每个部分有独立的符号, 因此上面例子中小时/分/秒会被认为是正值. 如果 *IntervalStyle* 设置给 *sql_standard* 那么开头的负号就应用到所有部分 (只有当别的地方没有符号时). 要不然, 就用传统 MPP 解释. 为了避免模糊, 如果时间段值是负的, 建议每个部分都带上负号.

在详细日期格式中, 对于更紧凑的输入格式的几个部分值可以带小数; 例如 '1.5 week' 或者 '01:02:03.45'. 这些输入被转换为月, 天和秒来存储. 当这些导致结果的月或者天有小数时, 小数部分被加到低一级的部分上. 转换规则是 1 月 = 30 天, 1 天 = 24 小时. 例如, '1.5 month' 变成 1 月 15 天. 输出中秒从来不会显示小数部分.

Table 9 这里给出 时间段 输入的有效示例.

Table 9. 时间段输入

举例	说明
1-2	SQL 标准格式: 1 年 2 个月
3 4:05:06	SQL 标准格式: 3 天 4 小时 5 分 6 秒
1 year 2 months 3 days 4 hours 5 minutes 6 seconds	传统 Postgres 格式: 1 年 2 月 3 天 4 小时 5 分 6 秒
P1Y2M3DT4H5M6S	ISO 8601 format with designators : 同上
P0001-02-03T04:05:06	ISO 8601 alternative format : 同上

内部地 时间段 值被存为月, 天, 和秒. 这是因为一个月中的天数是变化的, 另外因为夏令时的原因, 一天可能有23, 24 或 25 小时. 月 and 天部分是整数, 秒可以包含小数. 因为时间段通常从常量字符串或 时间戳 相减得来, 大部分情况下这种存储挺好, 但有时也有一些意外: `SELECT EXTRACT(hours from '80 minutes'::interval); date_part ----- 1` `SELECT EXTRACT(days from '80 hours'::interval); date_part ----- 0` 当超过值的正常范围时, 可以用 `justify_days` 函数和 `justify_hours` 函数用来调整天和小时.

时间段输出

时间段类型的输出格式可以设置为以下四种格式之一: `sql_standard`, `postgres`,

postgres_verbose, 或者 iso_8601, 通过命令 SET intervalstyle 来设置. 默认是 postgres 格式. [Table 10](#) 里有每种输出格式的一些例子.

如果时间段值符合标准要求 (年-月 或者 天-时间, 没有混合正负数), sql_standard 样式产生的输出遵从 SQL 标准中对于时间段字面值的规范. 要不然输出看起来像一个标准的 年-月 字面值, 后面跟着一个 天-时间 字面值字符串, 并且标记上明确的符号来消除混合符号带来的歧义.

当 [DateStyle](#) 参数设置为 ISO 时, postgres 样式输出与 PostgreSQL 8.4 发布之前的输出一致.

当 *DateStyle* 参数设置为 非- ISO 时, postgres_verbose 样式的输出与 PostgreSQL 8.4 之前版本的输出一致.

iso_8601 样式与 ISO 8601 标准中 4.4.3.2 节关于 format with designators 的描述一致.

Table 10. 时间段输出样式举例

样式规范	年-月 时间段	天-时间 时间段	混合时间段
sql_standard	1-2	3 4:05:06	-1-2 +3 -4:05:06
postgres	1 year 2 mons	3 days 04:05:06	-1 year -2 mons +3 days -04:05:06
postgres_verbose	@ 1 year 2 mons	@ 3 days 4 hours 5 mins 6 secs	@ 1 year 2 mons -3 days 4 hours 5 mins 6 secs ago
iso_8601	P1Y2M	P3DT4H5M6S	P-1Y-2M3DT-4H-5M-6S

Parent topic: [MPP 数据库数据类型](#)

伪类型

MPP 数据库支持一个特殊目的数据类型集合，叫做 *伪类型*。一个伪类型不能被用作列数据类型，但是可以用来声明一个函数的参数或返回值。当一个函数的行为并不是简单获取或者返回一个特定 SQL 数据类型值的时候，这些伪类型非常有用。

过程语言实现的函数只能使用实现语言中支持的伪类型。过程与语言全都禁止使用伪类型作为参数类型，仅仅允许 *void* 和 *record* 作为返回类型。

一个使用 *record* 伪类型作为返回数据类型的函数返回一个待定的行数据类型。*record* 表示可能的匿名复合类型数组。由于复合数据带有自己的类型标识，因此不需要额外的数组级信息。

伪类型 *void* 表示一个函数不返回值。

Note: MPP 数据库不支持触发器和 *trigger* 伪类型。

另外，*anyelement*，*anyarray*，*anynonarray*，和 *anyenum* 几个伪类型被称为多态类型。一些过程语言也支持多态函数使用 *anyarray*，*anyelement*，*anyenum*，和 *anynonarray* 函数。

anytable 伪类型是一个 MPP 数据库类型，表示一个二维表表达式 — 评估为一张二维表的表达式。MPP 数据库只允许这个类型作为用户定义函数的参数。参见 [二维表表达式](#) 了解更多关于 *anytable* 伪类型信息。

更多关于伪类型的介绍，查看 PostgreSQL 文档中 关于 [伪数据类型](#)。

Parent topic: [MPP 数据库数据类型](#)

多态类型

四种特殊目的的数据类型: *anyelement*, *anyarray*, *anynonarray*, 和 *anyenum*, 被称作 多态 类型. 任何使用这些类型的函数被称做多态函数. 通过在运行时检查实际传递的数据, 一个多态函数可以操作许多不同的数据类型.

多态参数和返回值互相关联, 当查询调用一个多态函数的时候, 真实类型才被决定. 每个位置 (或者参数或者返回值) 声明为 *anyelement* 类型时, 允许有任何特定的实际类型, 但是, 任何调用中, 必须是同一种真实数据类型. 每个位置声明为 *anyarray* 允许有任何类型的数组, 但同样的, (在一个特定查询中,) 必须是同一种数据类型. 如果有位置声明为 *anyarray* 同时别的位置声明为 *anyelement*, 在 *anyarray* 位置中的实际数组类型的元素必须与 *anyelement* 位置 中的类型相同. *anynonarray* 与 *anyelement* 类型同样对待, 但是包含附加的约束就是, 实际类型不能是数组类型. *anyenum* 被当作 *anyelement* 同样对待, 但是实际类型必须是 enum 类型.

当超过一个参数位置声明为多态类型, 最终效果是只允许仅有特定的实际参数组合. 举例说, 函数声明为 `equal( anyelement, anyelement )` 则这个函数接受两个输入值, 并且他们是同样的数据类型就可以.

当一个函数的返回值声明为多态类型时, 必须有一个参数位置也声明为多态类型, 并且实际参数的数据类型决定调用的实际返回值类型. 举例说, 如果还没有数组订阅机制, 你可以像这样定义一个函数实现订阅: `subscript( anyarray, integer ) returns anyelement`. 这里声明约束第一个参数必须是数组类型, 并且允许解析器根据第一个实际参数类型, 推断新的正确的返回值类型. 另一个例子是一个函数声明为 `myfunc( anyarray ) returns anyenum` 则只能接受一个 enum 类型数组(作为参数).

注意: *anynonarray* 和 *anyenum* 不表示单个类型变量; 可以用 *anyelement*, 只是带一点附加约束. 例如, 声明函数为 `myfunc( anyelement, anyenum )` 与声明为 `myfunc( anyenum, anyenum )` 是一样的: 两者的实际参数都必须是同样的 enum 类型.

一个可变参数函数 (接收可变数目参数) 中, 如果最后的参数声明为 `VARIADIC`

anyarray , 就是多态的. 为了匹配和决定实际参数类型, 这样的函数与你生命合适数量的 *anynonarray* 参数的函数一样.

了解更多多态类型信息, 参见 PostgreSQL 文档关于 [多态参数和返回值](#).

二维表表达式

当用 *anytable* 伪类型声明了一个函数参数, 这就是一个二维表表达式. 一个二维表表达式的写法是把 SELECT 语句放在一个 TABLE() 函数中. 你可以通过添加这样的语句来指定表的分布规则: SCATTER RANDOMLY, 或者 SCATTER BY 子句以及列列表指定分布键.

SELECT 语句在函数被调用时执行, 结果集被分布到 segments 节点中, 因此每一个 segment 节点在一部分结果表上执行函数.

举例说, 下面这个表表达式从一个叫做 customer 的表中选了三列, 并将分布键设置为第一列:

```
TABLE(SELECT cust_key, name, address FROM customer SCATTER BY 1)
```

SELECT 语句可以包括连接多个基础表, WHERE 语句, 聚集, 以及任何有效的查询语法.

对于 C 或 C++ 语言实现的函数, *anytable* 类型 是唯一允许的(多态)类型. 函数体可以用 MPP 数据库服务器编程接口 (SPI) 或者 MPP 伙伴连接器接口 (GPPC) API 访问.

全文搜索类型

MPP 数据库提供两种数据类型，这两种数据类型被设计来支持全文搜索，即搜索自然语言 文档集合并定位与一个 查询最匹配内容。 tsvector 类型以一种专为全文搜索优化的形式表示一个文档; 类似地， tsquery 类型表示一个文本查询。[使用全文搜索](#) 有这方面支持的详细解释, [文本搜索函数和操作符](#) 总结了相关函数和操作。

tsvector 和 tsquery 类型不能是 MPP 数据库表的分布键的一部分。

Parent topic: [MPP 数据库数据类型](#)

tsvector

一个 tsvector 值是不同 词素(lexemes) 的排序列表, 也就是由对不同形式的词进行合并，已经被 标准化的单词组成的列表 (更多细节，请参考 [使用全文搜索](#))。排序和去重在输入过程中已经自动完成。请看以下例子：

```
SELECT 'a fat cat sat on a mat and ate a fat rat'::tsvector;
      tsvector
-----
'a' 'and' 'ate' 'cat' 'fat' 'mat' 'on' 'rat' 'sat'
```

要表示包含空格和标点符号的词素，可以用单引号把它们包围起来：

```
SELECT $$the lexeme ' ' contains spaces$$::tsvector;
      tsvector
-----
' ' 'contains' 'lexeme' 'spaces' 'the'
```

(在这个例子中，我们使用了 \$\$ 引起来的字符串字面值。下一个例子中，我们使用常规的单引号引起来的字符串，所以字面值中嵌入的单引号必须双写进行转义。) 单引号引起来的词素中嵌入的单引号(')和反斜杠(\)必须双写:

```
SELECT $$the lexeme 'Joe"s' contains a quote$$::tsvector;
           tsvector
-----
'Joe"s' 'a' 'contains' 'lexeme' 'quote' 'the'
```

可选的, 整数 *定位数值* 可以被附加到词素:

```
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
```

一个定位数值通常表示源单词在文档中的位置。定位信息能够在 *邻近排名* 时使用。定位数值的范围可以从 1 到 16383; 更大的数值自动归整到 16383。相同定位数值的同一词素会被丢弃。

包含定位数值的词素可以进一步使用一个 *权重* 进行标记, 权重可以是 A, B, C, 或 D。D 是默认权重因此在输出中不会显示:

```
SELECT 'a:1A fat:2B,4C cat:5D'::tsvector;
           tsvector
-----
'a':1A 'cat':5 'fat':2B,4C
```

典型地，权重被用来反映文档结构, 例如, 可以使用不同权重标记标题与正文中的单词。全文搜索排名函数能够对不同权重标记指派不同优先级。

理解 tsvector 类型本身并不进行任何标准化非常重要; 它假定给它的单词已经为应用进行了适当地标准化。例如,


```
select 'The Fat Rats'::tsvector;
      tsvector
-----
'Fat' 'Rats' 'The'
```

对大多数英文全文搜索应用，上面的单词被认为是非标准化的，但是 `tsvector` 并不在意。原始文档文本应该总是通过 `to_tsvector` 函数来为搜索进行适当地标准化：

```
SELECT to_tsvector('english', 'The Fat Rats');
      to_tsvector
-----
'fat':2 'rat':3
```

tsquery

一个 `tsquery` 值为搜索存储词素，同时结合它们实现布尔操作 `&` (与), `|` (或), 以及 `!` (非)。 括号可以用来表示分组操作：

```
SELECT 'fat & rat'::tsquery;
      tsquery
-----
'fat' & 'rat'

SELECT 'fat & (rat | cat)'::tsquery;
      tsquery
-----
'fat' & ( 'rat' | 'cat' )

SELECT 'fat & rat & ! cat'::tsquery;
      tsquery
-----
'fat' & 'rat' & '!cat'
```

不使用括号时,!(非) 具有最高优先级, 另外 & (与) 优先级高于 |(或)。

可选地, 一个 tsquery 中的词素可以带一个或多个权重标记字母, 用来限制他们只与合适权重的 tsvector 相匹配:

```
SELECT 'fat:ab & cat':tsquery;  
      tsquery  
-----  
'fat':AB & 'cat'
```

Also, 一个 tsquery 中的词素也可以用星号(*)指定前缀匹配:

```
SELECT 'super:*':tsquery;  
      tsquery  
-----  
'super':*
```

这个查询会匹配一个以 "super" 开头的 tsvector 中的任何单词。 注意前缀会被全文搜索配置首先处理, 也就是说这个比较会返回 true (真):

```
SELECT to_tsvector( 'postgraduate' ) @@ to_tsquery( 'postgres:*' );  
?column?  
-----  
t  
(1 row)
```

因为 postgres 中提取的词干是 postgr :

```
SELECT to_tsquery('postgres:*');  
      to_tsquery  
-----  
'postgr':*  
(1 row)
```

所以就匹配到了 postgraduate 。

词素引用规则与前面 tsvector 中的规则一样; 同时, 也和 tsvector 一样, 任何需要标准化的单词必须在转换为 tsquery 类型之前完成。to_tsquery 函数可以方便地用来进行这类标准化操作:

```
SELECT to_tsquery('Fat:ab & Cats');
       to_tsquery
-----
'fat':AB & 'cat'
```

范围类型

范围类型是一种数据类型, 它代表一种元素类型 (即范围类型的子类型) 的一个范围. 举例说, timestamp 的范围类型可以用来表示一个会议室预定的时间范围. 在这个例子中, 数据类型是 tsrange (“timestamp range” 的缩写), 这里 timestamp 是子数据类型. 子数据类型必须有一个整体顺序, 因此很好确定一个元素值是否在范围值域内, 范围值域前, 还是范围值域后面.

范围类型表示单个范围内的许多元素, 所以非常有用, 因为一些概念, 像重叠范围, 能够用范围类型清晰表达. 为调度目的使用时间和日期是一个更清楚的例子; 对于价格范围, 仪器的测量范围, 等等, 也很有用.

父主题: [MPP 数据库数据类型](#)

内置范围类型

MPP 数据库带有以下内置范围类型:

- int4range -- integer 范围
- int8range -- bigint 范围
- numrange -- numeric 范围
- tsrange -- timestamp without time zone 范围
- tstzrange -- timestamp with time zone 范围
- daterange -- date 范围

除此之外, 你也可以定义你自己的范围类型; 参见 [CREATE TYPE](#) 了解更多信息.

举例

```
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);

-- 提取上边界
SELECT upper(int8range(15, 25));

-- 计算交集
SELECT int4range(10, 20) * int4range(15, 25);

-- 是否为空?
SELECT isempty(numrange(1, 5));
```

参见 [范围函数和运算符](#) 查看范围类型的操作符和函数完整列表。

包含边界与非包含边界

每一个非空范围有两个边界, 下边界和上边界. 所有在这两个边界之间的点都包含在范围内. 包含边界意味着边界点本身也包括在范围内, 而非包含边界意思是边界点不包括在范围内.

对于一个范围的文本格式, 包含下边界用 [表示, 而非包含下边界用 (表示. 同样, 包含上边界用] 表示, 而非包含上边界用) 表示. (参见 [范围函数和运算符](#) 了解更多信息.)

函数 `upper_inc` 和 `lower_inc` 分别用来测试一个范围值的上下边界包含关系.

无限 (无边界) 范围

范围的下边界可以忽略, 意味着所有小于上边界的点都包含在范围内. 同样地, 如果范围的上边界被忽略, 那么所有大于下边界的点都包含在范围内. 如果上下边界都忽略了, 那么这个元素类型的所有值都被认为包含在范围内.

可以等价地分别认为, 这个时候下边界是“负无穷”, 或者上边界是“正无穷”. 但是请注意, 这些无穷值不是一个范围的元素类型的值, 也不是范围的一部分. (所有无穷边界不存在包含问题 -- 如果你这样写的话, 会自动转换为非包含边界.)

有些元素类型也有“infinity”的表示, 但那仅仅是一个值, 与范围边界无关. 举例说, 在时间戳范围中, `[today,]` 与 `[today,)` 含义相同. 但是 `[today,infinity]` 与 `[today,infinity)` 含义就不同了 -- 后者排除了特殊的时间戳值 `infinity`.

函数 `lower_inf` 和 `upper_inf` 可以分别检测一个范围的无限上边界和下边界.

范围类型输入/输出

范围类型的输入必须遵循下面的模式:

```
( lower-bound, upper-bound )  
( lower-bound, upper-bound ]  
[ lower-bound, upper-bound )  
[ lower-bound, upper-bound ]  
empty
```

和前面说的一样, 方括号或圆括号表示是否上下边界是包含, 非包含关系. 注意最后一种模式是 `empty`, 表示一个空范围 (一个不包含任何点的范围).

lower-bound 可以是一个子类型的有效字符串, 或者用 `empty` 表示没有下边界. 同样地, *upper-bound* 可以是一个子类型的有效字符串, 或者用 `empty` 表示没有上边界.

每一个边界值都可以是用 " (双引号) 引起来的字符串. 如果边界值包含圆括号, 方括号, 逗号, 双引号, 反斜杠的时候, 则必须引起来, 因为这些字符可能是范围语法的一部分. 如果引起来的边界值有双引号或者反斜杠, 需要在前面插入额外的反斜杠进行转义. (在双引号引起来的边界值中两个双引号也用来表示一个双引号字符, 类似于 SQL 字面值中的单引号规则.) 当然, 你也可以避免引号, 只用反斜杠把所有会被认为是范围语法一部分的字符进行转义. 另外, 边界值也可以写成空字符串 "", 因为啥也不写意味着无限边界.

在范围值的前后, 允许出现空格, 但是任何在圆括号或方括号之间的空格会被认为是上边界或下边界的一部分. (与元素类型有关, 或许有或许没意义.)

例如:

```
-- 包含 3, 不包含 7, 并且包含期间的所有点
SELECT '[3,7)':int4range;

-- 不包含 3 和 7, 单包含之间的所有点
SELECT '(3,7)':int4range;

-- 只包含单个点 4
SELECT '[4,4)':int4range;

-- 不包含任何点 (会被规范化为 'empty')
SELECT '[4,4)':int4range;
```

构造范围

每个范围类型有一个与范围类型同名的构造函数. 由于避免了边界值的额外引号 (和转义), 通常使用构造函数比写一个范围字面常量更方便. 构造函数接受二或三个参数. 两个参数形式构造一个标准格式的范围 (包含下边界, 不包含上边界), 而三个参数格式构造一个通过第三个参数指定的边界(包含关系). 第三个参数必须是下面的字符串之一: `()`, `[)`, `]`, 或者 `[]`. 举例如下:

```
-- 全格式是: 下边界, 上边界, 以及文本参数指示
-- 包含/非包含边界.
SELECT numrange(1.0, 14.0, '[');

-- 如果第三个参数忽略, 默认是 '['.
SELECT numrange(1.0, 14.0);

-- 虽然这里指定 '[', 显示值会被转换成经典格式
-- 因为 int8range 是一个离散范围类型 (见下).
SELECT int8range(1, 14, '[');

-- 对某一边界使用 NULL 导致这一边无边界.
SELECT numrange(NULL, 2.2);
```

离散范围类型

离散范围类型的元素类型有一个明确定义的“步长”，例如 `integer` 或 `date`. 在这些类型中，当在两个元素间没有合法的值时，我们就可以说这两个元素是相邻的. 这个与连续范围相对，在连续范围中，两个元素之间总是可以找到其它元素值. 例如，`numeric` 类型的范围就是连续的，`timestamp` 类型的范围也是连续的. (即使 `timestamp` 有有限的精度，理论上可以认为它是离散的，但由于步长值通常不重要，所以最好认为它是连续的.)

另一个理解离散范围类型的方法是，对于每一个元素值，它有清晰的“后一个”和“前一个”值. 了解了这些，就可以通过选择后一个或前一个元素值，取代原始元素值，来对范围边界的包含和非包含关系进行转换. 举例说，一个整数范围类型 `[4,8]` 和 `(3,9)` 表示的是同样值的集合；但对于一个 `numeric` 类型的范围就没有这样的关系.

一个离散范围类型应该有一个 *canonicalization* 函数来感知元素类型的步长. *canonicalization* 函数用于把等价范围类型值转换为同一表示，特别是一致的包含或非包含边界. 如果没有指定 *canonicalization* 函数，那么不同格式总是当成不相等对待，即使他们实际上表示同样的值域.

内置范围类型 `int4range`, `int8range`, 和 `daterange` 全部使用一个包含下边界，不包含上边界的一个经典格式；即 `[]`. 然而，用户定义的范围类型可以使用其他转换形式.

定义新的范围类型

用户可以定义他们自己的范围类型. 这样做的通常原因是要使用的子类型的范围没有内置提供. 例如，定义子类型为 `float8` 的新的范围类型：

```
CREATE TYPE floatrange AS RANGE (  
    subtype = float8,  
    subtype_diff = float8mi  
);  
  
SELECT '[1.234, 5.678]':floatrange;
```

因为 float8 没有“步长”，这个例子中我们不能定义一个 canonicalization 函数给它。

定义你自己的范围类型也允许你指定一个不同子类型 B-树 操作类或校对方式 (collation) 来使用, 可以改变排序规则, 以决定哪些值会落在给定范围之内。

如果子类型被认为是离散值而不是连续值, CREATE TYPE 命令应该指定一个 canonicalization 函数. canonicalization 函数接受一个范围值, 返回一个具有不同边界和格式的等价范围值. 对于表示同一值集的两个范围, canonicalization 函数输出一致., 例如整数范围 [1, 7] 和 [1, 8), 一定是相同的. 不论你选择哪一个作为标准输出, 只要不同格式的等价值总是映射到同样格式的相同值(就可以). 除了调整包含/非包含边界格式, canonicalization 函数还可以圆整(四舍五入)边界值, 例如设定的步长比子类型存储分辨能力宽(就需要这个功能). 例如, 一个 timestamp 范围类型, 可以定义为步长为 1 小时, 这种情况下, canonicalization 函数需要对不是整小时的边界进行圆整, 当然也可以抛出一个异常。

另外, 与 GiST 或 SP-GiST 索引一起使用的范围类型应该定义一个子类型差异函数, 即 subtype_diff 函数. (没有 subtype_diff 函数索引也能工作, 但可能被认为没有提供这个函数时高效.) 子类型差异函数接受两个子类型输入值, 同时返回他们之间的差 (即, X 减去 Y) 用 float8 值表示. 在我们上面的例子中, 函数 float8mi 隐含有常规 float8 相减操作来用; 但对于任何其他子类型, 某种类型的转换是必要的. 关于如何表示数值之间差异的一些创造性想法也很需要. 为了最大限度扩展这种可能, subtype_diff 函数应该与选定操作类和校对函数一致; 也就是说, 根据排序顺序, 当第一个参数比第二个大时, 这个函数应该范围正数。

一个关于 subtype_diff 函数不那么简单的例子是：

```
CREATE FUNCTION time_subtype_diff(x time, y time) RETURNS float8 AS  
'SELECT EXTRACT(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;
```

参见 [CREATE TYPE](#) 了解更多关于创建范围类型的详细信息.

索引

GiST 和 SP-GiST 索引可以在范围类型列上创建. 例如, 创建一个 GiST 索引:

```
CREATE INDEX reservation_idx ON reservation USING GIST (during);
```

一个 GiST 或者 SP-GiST 索引能够加速包含以下操作的范围运算: =, &&, <@, @>, <<, >>, -|-, &<, 以及 &> (参见 [范围函数和运算符](#) 了解更多信息).

另外, B-树 和 hash 索引也可以创建在范围类型的表列上. 对于这些索引类型, 基本上只对相等范围操作有用. 也有一个范围值的 B-树 排序定义, 具有相应的 < 和 > 操作, 但是排序方法相当简单粗暴, 现实生活中并不常用. 范围类型的 B-树 和 hash 支持主要允许排序和查询内部建立 hash, 而不是为了创建实际的索引.

字符集支持

MPP数据库中的字符集支持允许您以各种字符集存储文本, 包括单字节字符集

（如ISO 8859系列）和多字节字符集（如EUC（扩展Unix代码），UTF-8），和Mule内部代码。 客户端可以透明地使用所有支持的字符集，但少数只能在服务器上使用（即作为一种服务器端编码）。 使用gpinitssystem初始化MPP数据库阵列时，会选择默认字符集。 创建数据库时可以覆盖它，因此您可以拥有多个数据库，每个数据库具有不同的字符集。

Table 1. MPP数据库字符集 ¹

名称	描述	语言	是否 服务端?	字节/ 字符	别名
BIG5	Big Five	繁体中文	No	1-2	WIN950, Windows9 50
EUC_CN	Extended UNIX Co de-CN	简体中文	Yes	1-3	
EUC_JP	Extended UNIX Co de-JP	日文	Yes	1-3	
EUC_KR	Extended UNIX Co de-KR	韩文	Yes	1-3	
EUC_TW	Extended UNIX Co de-TW	繁体中文， 台湾	Yes	1-3	
GB18030	国家标准	中文	No	1-2	
GBK	扩展国标	简体中文	No	1-2	WIN936, Windows9 36
ISO_8859 _5	ISO 8859-5, ECMA 113	Latin/Cyrilli c	Yes	1	

名称	描述	语言	是否服务端?	字节/字符	别名
ISO_8859_6	ISO 8859-6, ECMA 114	Latin/Arabic	Yes	1	
ISO_8859_7	ISO 8859-7, ECMA 118	Latin/Greek	Yes	1	
ISO_8859_8	ISO 8859-8, ECMA 121	Latin/Hebrew	Yes	1	
JOHAB	JOHA	Korean (Hangul)	Yes	1-3	
KOI8	KOI8-R(U)	Cyrillic	Yes	1	KOI8R
LATIN1	ISO 8859-1, ECMA 94	Western European	Yes	1	ISO88591
LATIN2	ISO 8859-2, ECMA 94	Central European	Yes	1	ISO88592
LATIN3	ISO 8859-3, ECMA 94	South European	Yes	1	ISO88593
LATIN4	ISO 8859-4, ECMA 94	North European	Yes	1	ISO88594
LATIN5	ISO 8859-9, ECMA 128	Turkish	Yes	1	ISO88599
LATIN6	ISO 8859-10, ECMA 144	Nordic	Yes	1	ISO885910
LATIN7	ISO 8859-13	Baltic	Yes	1	ISO885913

名称	描述	语言	是否服务端?	字节/字符	别名
LATIN8	ISO 8859-14	Celtic	Yes	1	ISO885914
LATIN9	ISO 8859-15	LATIN1 with Euro and accents	Yes	1	ISO885915
LATIN10	ISO 8859-16, ASRO SR 14111	Romanian	Yes	1	ISO885916
MULE_INTERNAL	Mule internal code	Multilingual Emacs	Yes	1-4	
SJIS	Shift JIS	Japanese	No	1-2	Mskanji, ShiftJIS, WIN932, Windows932
SQL_ASCII	unspecified ²	any	No	1	
UHC	Unified Hangul Code	Korean	No	1-2	WIN949, Windows949
UTF8	Unicode, 8-bit	all	Yes	1-4	Unicode
WIN866	Windows CP866	Cyrillic	Yes	1	ALT
WIN874	Windows CP874	Thai	Yes	1	
WIN1250	Windows CP1250	Central European	Yes	1	
WIN1251	Windows CP1251	Cyrillic	Yes	1	WIN

名称	描述	语言	是否服务端?	字节/字符	别名
WIN1252	Windows CP1252	Western European	Yes	1	
WIN1253	Windows CP1253	Greek	Yes	1	
WIN1254	Windows CP1254	Turkish	Yes	1	
WIN1255	Windows CP1255	Hebrew	Yes	1	
WIN1256	Windows CP1256	Arabic	Yes	1	
WIN1257	Windows CP1257	Baltic	Yes	1	
WIN1258	Windows CP1258	Vietnamese	Yes	1	ABC, TCVN, TCVN5712, VSCII

Parent topic: [MPP数据库参考手册](#)

设置字符集

gpinitssystem通过在初始化时读取gp_init_config文件中ENCODING参数的设置来定义MPP数据库系统的默认字符集。默认字符集是UNICODE或UTF8。

除了用作系统范围的默认值之外，您还可以创建具有不同字符集的数据库。例如：

```
=> CREATE DATABASE korean WITH ENCODING 'EUC_KR';
```

重点: 虽然您可以为数据库指定所需的任何编码，但选择与您选择的语言环境不同的编码是不明智的。 LC_COLLATE和LC_CTYPE设置意味着特定的编码，

并且依赖于区域设置的操作（例如排序）可能会错误解释处于不兼容编码的数据。

由于这些语言环境设置被gpinitssystem冻结，因此在不同数据库中使用不同编码的明显的灵活性更具理论性而非实用性。

安全使用多种编码的一种方法是在初始化期间将语言环境设置为C或POSIX，从而禁用任何真实的语言环境感知。

服务器和客户端之间的字符集转换t

MPP数据库支持服务器和客户端之间针对特定字符集组合的自动字符集转换。转换信息存储在master *pg_conversion*系统catalog表中。 MPP数据库附带了一些预定义的转换，或者您可以使用SQL命令CREATE CONVERSION创建新的转换。

Table 2. 客户端/服务器字符集转换

服务器字符集	可用的客户端字符集
BIG5	不支持作为服务器编码
EUC_CN	EUC_CN, MULE_INTERNAL, UTF8
EUC_JP	EUC_JP, MULE_INTERNAL, SJIS, UTF8
EUC_KR	EUC_KR, MULE_INTERNAL, UTF8
EUC_TW	EUC_TW, BIG5, MULE_INTERNAL, UTF8
GB18030	不支持作为服务器编码
GBK	不支持作为服务器编码
ISO_8859_5	ISO_8859_5, KOI8, MULE_INTERNAL, UTF8, WIN866, WIN1251

服务器字符集	可用的客户端字符集
ISO_8859_6	ISO_8859_6, UTF8
ISO_8859_7	ISO_8859_7, UTF8
ISO_8859_8	ISO_8859_8, UTF8
JOHAB	JOHAB, UTF8
KOI8	KOI8, ISO_8859_5, MULE_INTERNAL, UTF8, WIN866, WIN1251
LATIN1	LATIN1, MULE_INTERNAL, UTF8
LATIN2	LATIN2, MULE_INTERNAL, UTF8, WIN1250
LATIN3	LATIN3, MULE_INTERNAL, UTF8
LATIN4	LATIN4, MULE_INTERNAL, UTF8
LATIN5	LATIN5, UTF8
LATIN6	LATIN6, UTF8
LATIN7	LATIN7, UTF8
LATIN8	LATIN8, UTF8
LATIN9	LATIN9, UTF8
LATIN10	LATIN10, UTF8
MULE_INTERNAL	MULE_INTERNAL, BIG5, EUC_CN, EUC_JP, EUC_KR, EUC_TW, ISO_8859_5, KOI8, LATIN1 to LATIN4, SJIS, WIN866, WIN1250, WIN1251
SJIS	不支持作为服务器编码
SQL_ASCII	不支持作为服务器编码
UHC	不支持作为服务器编码
UTF8	所有支持的编码
WIN866	WIN866

服务器字符集	可用的客户端字符集
ISO_8859_5	KOI8, MULE_INTERNAL, UTF8, WIN1251
WIN874	WIN874, UTF8
WIN1250	WIN1250, LATIN2, MULE_INTERNAL, UTF8
WIN1251	WIN1251, ISO_8859_5, KOI8, MULE_INTERNAL, UTF8, WIN866
WIN1252	WIN1252, UTF8
WIN1253	WIN1253, UTF8
WIN1254	WIN1254, UTF8
WIN1255	WIN1255, UTF8
WIN1256	WIN1256, UTF8
WIN1257	WIN1257, UTF8
WIN1258	WIN1258, UTF8

要启用自动字符集转换，您必须告诉MPP数据库您要在客户端中使用的字符集（编码）。有几种方法可以实现此目的：

- 在psql中使用\encoding命令，它允许您动态更改客户端编码。
 - 使用 SET client_encoding TO.
- 要设置客户端编码，请使用以下SQL命令

```
=> SET CLIENT_ENCODING TO 'value';
```

要查询当前的客户端编码：

```
=> SHOW client_encoding;
```

要返回到默认编码：

```
=> RESET client_encoding;
```

- 使用PGCLIENTENCODING环境变量。在客户端环境中定义PGCLIENTENCODING时，将在与服务器建立连接时自动选择该客户端编码。（这可以随后使用上面提到的任何其他方法覆盖。）
- 设置配置参数client_encoding。如果在master postgresql.conf文件中设置了client_encoding，则在建立与MPP数据库的连接时会自动选择该客户端编码。（这可以随后使用上面提到的任何其他方法覆盖。）

如果无法转换特定字符“假设您为服务器选择了EUC_JP而对客户端选择了LATIN1，那么某些日文字符在LATIN1中没有表示”，则会报告错误。

如果客户端字符集定义为SQL_ASCII，则无论服务器的字符集如何，都将禁用编码转换。除非使用all-ASCII数据，否则使用SQL_ASCII是不明智的。

¹ 并非所有API都支持所有列出的字符集。例如，JDBC驱动程序不支持MULE_INTERNAL，LATIN6，LATIN8和LATIN10。

² SQL_ASCII设置与其他设置的行为大不相同。字节值0-127根据ASCII标准进行解释，字节值128-255作为未解释的字符。如果使用任何非ASCII数据，则将SQL_ASCII设置用作客户端编码是不明智的。不支持SQL_ASCII作为服务器编码。

服务器配置参数

有很多MPP服务器配置参数能够影响MPP数据库系统的行为。如同常规的PostgreSQL数据库系统一样，许多配置参数有相同的名称、设置以及行为。

- [参数类型和值](#)描述参数数据类型和值。
- [设置参数](#)描述了谁可以改变它们以及何时何处可以设置它们的限制。

- [参数类别](#)根据功能组织参数。
- [配置参数](#)以字母顺序列出参数描述。

参数类型和值

所有的参数不区分大小写。每个参数取值为四种类型之一：布尔, 整数, 浮点, 或字符串。

布尔值可能被写成ON, OFF, TRUE, FALSE, YES, NO, 1, 0(所有都是不区分大小写)。

枚举类型参数的指定方式与字符串参数相同，但仅限于一组有限的值。枚举参数值不区分大小写。

某些设置指定内存或者时间值。每个都有一个隐式单位，它是千字节，块（通常为8千字节），毫秒，秒，或者分钟。有效内存单位是kB（千字节），MB（兆字节），和GB（千兆字节）。有效的时间单位是ms（毫秒），s（秒），min（分钟），h（小时），和d（天）。注意内存单元的乘数为1024，而不是1000。有效的时间表达式包含数字和单位。当指定内存和时间单位时使用SET命令，将值括在引号中。例如：

```
SET statement_mem TO '200MB';
```

Note: 值和单位名字之间没有空格。

设置参数

许多配置参数对谁可以更改它们，何处改变它们以及何时改变他们都做了限制，例如，用户必须是MPP 数据库的超级用户。其他参数需要重启系统才能更改生

效。分类为session级别的参数可以在系统层级设置（在postgresql.conf文件中），在数据库层级设置（使用ALTER DATABASE），在角色层级设置（使用ALTER ROLE），在数据库角色级设置（ALTER ROLE...IN DATABASE...SET），或者在会话层级设置（使用using SET）。系统参数只能在postgresql.conf文件中设置。

在MPP数据库中，Master实例和Segment实例都有它自己的postgresql.conf文件（位于各自的数据目录中）。一些参数被认为是local参数，意味着每个Segment实例都查看自己的postgresql.conf文件来获取该参数的值。用户必须在系统上的每个实例中都设置local参数（Master实例和Segment实例）。其他参数被认为是master参数。master参数只能在Master实例上设置。

这个表描述了服务器配置参数描述中“可设置分类”列中的值。

Table 1. Settable Classifications

设置分类	描述
master或local	master参数只能在MPP的Master实例的postgresql.conf文件中设置。然后，该参数的值在运行时被传递到（或者忽略）到Segment实例。 local参数必须在Master实例和每个Segment实例的postgresql.conf文件中设置。每个Segment实例会在其自己的配置文件中去获得参数。local参数设置总是需要系统重启才能生效。

设置分类	描述
session或system	<i>Session</i>参数可以在数据库会话中即时更改，并且可以具有以下设置的层次结构：在系统层级（postgresql.conf）、在数据库层级（ALTER DATABASE...SET）、在角色层级（ALTER ROLE...SET）、在数据库和角色层级或者会话层级（SET）。如果参数设置为多个级别，则最细度的设置优先（例如，会话覆盖数据库和角色，数据库和角色覆盖角色，角色覆盖数据库，数据库覆盖系统）。 <i>system</i>参数只能通过postgresql.conf文件修改。
restart或reload	当更改postgresql.conf文件中的参数值时，有些需要重启MPP数据库才能生效。其他参数只需要重新加载服务器配置文件就能刷新（使用gpstop -u），不需要停止系统。
superuser	这些会话参数只能通过数据库超级用户才能设置，常规用户不能设置这个参数。
read only	这些参数不能被数据库用户和超级用户设置，当前的参数值可以显示但是不会更改。

参数类别

配置参数会影响服务器行为的类别，例如资源消耗，查询调整 and 身份验证。以下主题描述了MPP配置参数类别。

- [连接和验证参数](#)
- [系统资源消耗参数](#)

- [GPORCA参数](#)
- [查询调优参数](#)
- [错误报告和日志参数](#)
- [系统监控参数](#)
- [运行时统计信息收集参数](#)
- [自动统计收集参数](#)
- [客户端连接默认参数](#)
- [锁管理参数](#)
- [资源管理参数（资源队列）](#)
- [资源管理参数（资源组）](#)
- [外部表参数](#)
- [数据库表参数](#)
- [历史版本兼容性参数](#)
- [MPP数据库阵列配置参数](#)
- [master和segment的MPP镜像参数](#)
- [MPP数据库扩展参数](#)

连接和验证参数

这些参数控制客户端如何连接和验证MPP数据库。

Connection Parameters

<code>gp_connection_send_timeout</code>	<code>tcp_keepalives_count</code>
<code>gp_vmem_idle_resource_timeout</code>	<code>tcp_keepalives_idle</code>
<code>listen_addresses</code>	<code>tcp_keepalives_interval</code>
<code>max_connections</code>	<code>unix_socket_directory</code>
<code>max_prepared_transactions</code>	<code>unix_socket_group</code>
<code>superuser_reserved_connections</code>	<code>unix_socket_permissions</code>

安全和身份验证参数

<code>authentication_timeout</code>	<code>password_encryption</code>
<code>db_user_namespace</code>	<code>password_hash_algorithm</code>
<code>krb_caseins_users</code>	<code>ssl</code>
<code>krb_server_keyfile</code>	<code>ssl_ciphers</code>

系统资源消耗参数

这些参数设置MPP数据库消耗的系统资源限制。

内存消耗参数

这些参数控制系统内存使用。

gp_vmem_idle_resource_timeout	gp_workfile_limit_per_query
gp_resource_group_memory_limit (基于资源组的资源管理)	gp_workfile_limit_per_segment
gp_vmem_protect_limit (基于资源队列的资源管理)	maintenance_work_mem
gp_vmem_protect_segworker_cache_limit	max_stack_depth
gp_workfile_limit_files_per_query	shared_buffers
	temp_buffers

OS资源参数

max_files_per_process
shared_preload_libraries

基于成本的清理延迟参数

Warning: 不要使用基于成本的清理延迟，因为它在segment实例之间异步运

行。在segment级别调用清理成本限制和延迟，而不考虑整个MPP数据库阵列的状态

您可以配置VACUUM和ANALYZE命令的执行成本，以减少I/O对并发数据库活动的影响。当I/O操作的累计成本达到限制时，执行操作的进程会暂停一段时间，然后重置计数器并继续执行

vacuum_cost_delay	vacuum_cost_page_hit
vacuum_cost_limit	vacuum_cost_page_miss
vacuum_cost_page_dirty	

事务ID管理参数

xid_stop_limit
xid_warn_limit

GPORCA参数

这些参数控制MPP数据库对GPORCA的使用。有关GPORCA的信息，请参阅[关于GPORCA](#)。

gp_enable_relsizes_collection	optimizer_join_arity_for_associativity_commutativity
optimizer	optimizer_join_order
optimizer_analyze_root_partition	optimizer_join_order_threshold
optimizer_array_expansion_threshold	optimizer_mdcache_size
optimizer_cte_inlining_bound	optimizer_metadata_caching
optimizer_control	optimizer_parallel_union
optimizer_enable_associativity	optimizer_print_missing_stats
optimizer_enable_master_only_queries	optimizer_print_optimization_stats
optimizer_force_agg_skew_avoidance	optimizer_sort_factor
optimizer_force_multistage_agg	
optimizer_force_three_stage_scalar_dqa	

查询调优参数

这些参数控制SQL查询处理的各个方面，例如查询运算符和运算符设置以及统计

采样。

Postgres查询优化器运算符控制参数

以下参数控制Postgres查询优化器可以使用的计划操作类型。 启用或禁用计划操作以强制Postgres优化程序选择其他计划。 这对于使用不同计划类型测试和比较查询性能非常有用。

enable_bitmapscan	gp_enable_agg_distinct_pruning
enable_groupagg	gp_enable_direct_dispatch
enable_hashagg	gp_enable_fast_sri
enable_hashjoin	gp_enable_groupect_distinct_gather
enable_indexscan	gp_enable_groupect_distinct_pruning
enable_mergejoin	gp_enable_multiphase_agg
enable_nestloop	gp_enable_predicate_propagation
enable_seqscan	gp_enable_preunique
enable_sort	gp_enable_relsizes_collection
enable_tidscan	gp_enable_sort_distinct
gp_enable_adaptive_nestloop	gp_enable_sort_limit
gp_enable_agg_distinct	

Postgres查询优化器成本计算参数

Warning: 请勿调整这些查询成本计算参数。 它们经过调整以反映MPP数据库硬件配置和典型工作负载。 所有这些参数都是相关的。 更改一个而不更改其他一个可能会对性能产生负面影响。

cpu_index_tuple_cost	gp_motion_cost_per_row
cpu_operator_cost	gp_segments_for_planner
cpu_tuple_cost	random_page_cost
cursor_tuple_fraction	seq_page_cost
effective_cache_size	

数据库统计采样参数

这些参数调整ANALYZE操作采样的数据量。 调整这些参数会影响系统范围内的统计信息收集。 您可以使用ALTER TABLE SET STATISTICS子句在特定表和列上配置统计信息收集。

default_statistics_target

排序运算符配置参数

[gp_enable_sort_distinct](#)

[gp_enable_sort_limit](#)

聚合运算符配置参数

[gp_enable_agg_distinct](#)

[gp_enable_agg_distinct_pruning](#)

[gp_enable_multiphase_agg](#)

[gp_enable_preunique](#)

[gp_enable_grouptext_distinct_gather](#)

[gp_enable_grouptext_distinct_pruning](#)

[gp_workfile_compression](#)

连接运算符配置参数

[join_collapse_limit](#)

[gp_adjust_selectivity_for_outerjoins](#)

[gp_hashjoin_tuples_per_bucket](#)

[gp_statistics_use_fkeys](#)

[gp_workfile_compression](#)

其他Postgres查询优化器配置参数

- [from_collapse_limit](#)
- [gp_enable_predicate_propagation](#)
- [gp_max_plan_size](#)
- [gp_statistics_pullup_from_child_partition](#)

查询计划执行

控制查询计划的执行。

- [gp_max_slices](#)

错误报告和日志参数

这些配置参数控制MPP数据库日志记录。

日志轮换

log_rotation_age	log_truncate_on_rotation
log_rotation_size	

何时记录

<code>client_min_messages</code>	<code>log_min_error_statement</code>
<code>gp_interconnect_debug_retry_interval</code>	<code>log_min_messages</code>
<code>log_error_verbosity</code>	<code>optimizer_minidump</code>
<code>log_min_duration_statement</code>	

记录什么

debug_pretty_print	log_executor_stats
debug_print_parse	log_hostname
debug_print_plan	gp_log_interconnect
debug_print_prelim_plan	log_parser_stats
debug_print_rewritten	log_planner_stats
debug_print_slice_table	log_statement
log_autostats	log_statement_stats
log_connections	log_timezone
log_disconnections	gp_debug_linger
log_dispatch_stats	gp_log_format
log_duration	gp_reraise_signal

系统监控参数

这些配置参数控制MPP数据库数据收集和与数据库监视相关的通知。

MPP性能数据库

以下参数配置填充gpperfmon数据库的数据收集代理。

gp_enable_gpperfmon	gpperfmon_log_alert_level
gp_gpperfmon_send_interval	gpperfmon_port

查询指标收集参数

这些参数启用和配置查询指标收集。 启用后，MPP数据库会在查询执行期间将指标保存到共享内存。

gp_enable_query_metrics	gp_instrument_shmem_size

运行时统计信息收集参数

这些参数控制服务器统计信息收集功能。 启用统计信息收集后，可以使用pg_stat系列目录视图访问统计信息数据。

stats_queue_level	update_process_title

自动统计收集参数

启用自动统计信息收集时，您可以在与某个阈值的行数被更新（on_change）或新生成没有统计信息的表时，在
与INSERT，UPDATE，DELETE，COPY或CREATE TABLE ... AS

SELECT语句相同的事务中自动运行ANALYZE。要启用此功能，请在MPP数据库master的postgresql.conf文件中设置以下服务器配置参数，然后重新启动MPP数据库：

gp_autostats_mode

gp_autostats_mode_in_functions

gp_autostats_on_change_threshold

log_autostats

Warning: 根据数据库操作的特定性质，自动统计信息收集可能会对性能产生负面影响。仔细评估on_no_stats的默认设置是否适合您的系统。

客户端连接默认参数

这些配置参数设置用于客户端连接的默认值。

语句行为参数

check_function_bodies default_tablespace default_transaction_deferrable default_transaction_isolation	default_transaction_read_only search_path statement_timeout vacuum_freeze_min_age

区域设置和格式化参数

client_encoding	lc_messages
DateStyle	lc_monetary
extra_float_digits	lc_numeric
IntervalStyle	lc_time
lc_collate	TimeZone
lc_ctype	

其他客户端默认参数

dynamic_library_path	local_preload_libraries
explain_pretty_print	

锁管理参数

这些配置参数设置锁和死锁的限制。

<code>deadlock_timeout</code> <code>gp_enable_global_deadlock_detector</code> <code>gp_global_deadlock_detector_period</code>	<code>lock_timeout</code> <code>max_locks_per_transaction</code>

资源管理参数（资源队列）

以下配置参数配置MPP数据库资源管理功能（资源队列），查询优先级，内存利用率和并发控制。

<code>gp_resqueue_memory_policy</code> <code>gp_resqueue_priority</code> <code>gp_resqueue_priority_cpucores_per_segment</code> <code>gp_resqueue_priority_sweeper_interval</code> <code>gp_vmem_idle_resource_timeout</code> <code>gp_vmem_protect_limit</code> <code>gp_vmem_protect_segworker_cache_limit</code> <code>max_resource_queues</code>	<code>max_resource_portals_per_transaction</code> <code>max_statement_mem</code> <code>resource_cleanup_gangs_on_wait</code> <code>resource_select_only</code> <code>runaway_detector_activation_percent</code> <code>statement_mem</code> <code>stats_queue_level</code> <code>vmem_process_interrupt</code>

资源管理参数（资源组）

以下参数配置MPP数据库资源组工作负载管理功能。

gp_resgroup_memory_policy	gp_vmem_idle_resource_timeout
gp_resource_group_bypass	gp_vmem_protect_segworker_cache_limit
gp_resource_group_cpu_limit	memory_spill_ratio
gp_resource_group_memory_limit	vmem_process_interrupt
gp_resource_manager	

外部表参数

以下参数配置MPP数据库的外部表功能。

gp_external_enable_exec	readable_external_table_timeout
gp_external_enable_filter_pushdown	writable_external_table_bufsize
gp_external_max_segs	verify_gpfdists_cert
gp_initial_bad_row_limit	
gp_reject_percent_threshold	

数据库表参数

以下参数配置MPP数据库表的默认选项设置。

[gp_create_table_random_default_distribution](#)

[gp_default_storage_options](#)

[gp_enable_exchange_default_partition](#)

[gp_enable_segment_copy_checking](#)

[gp_use_legacy_hashops](#)

追加优化表参数

以下参数配置MPP数据库的追加优化表功能。

[max_appendonly_tables](#)

[gp_appendonly_compaction](#)

[gp_appendonly_compaction_threshold](#)

[validate_previous_free_tid](#)

历史版本兼容性参数

以下参数提供与较旧的PostgreSQL和MPP数据库版本的兼容性。 您无需在MPP数据库中更改这些参数。

PostgreSQL

<code>array_nulls</code>	<code>regex_flavor</code>
<code>backslash_quote</code>	<code>standard_conforming_strings</code>
<code>escape_string_warning</code>	<code>transform_null_equals</code>

MPP数据库

<code>gp_ignore_error_table</code>

MPP数据库阵列配置参数

本主题中的参数控制MPP数据库阵列及其组件的配置：segment，master，分布式事务管理器，主镜像和互连。

互连配置参数

gp_interconnect_fc_method	gp_interconnect_setup_timeout
gp_interconnect_hash_multiplier	gp_interconnect_snd_queue_depth
gp_interconnect_queue_depth	gp_interconnect_type
	gp_max_packet_size

Note: MPP数据库仅支持UDPIFC（默认）和TCP互连类型。

调度配置参数

gp_cached_segworkers_threshold	gp_segment_connect_timeout
gp_enable_direct_dispatch	gp_set_proc_affinity

故障操作参数

gp_set_read_only	gp_fts_probe_timeout
gp_fts_probe_interval	gp_fts_probe_threadcount
gp_fts_probe_retries	gp_log_fts

分布式事务管理参数

`gp_max_local_distributed_cache`

只读参数

`gp_command_count`

`gp_content`

`gp_dbid`

`gp_role`

`gp_session_id`

`gp_server_version`

`gp_server_version_num`

master和segment的MPP镜像参数

这些参数控制MPP数据库master和standby之间的复制配置。

`keep_wal_segments`

`repl_catchup_within_range`

`replication_timeout`

`wal_receiver_status_interval`

MPP数据库扩展参数

本主题中的参数控制MPP数据库扩展的配置。

- [pljava_classpath](#)
- [pljava_classpath_insecure](#)
- [pljava_statement_cache_size](#)
- [pljava_release_lingering_savepoints](#)
- [pljava_vmoptions](#)

XML数据参数

- [xmlbinary](#)
- [xmloption](#)

配置参数

按字母顺序列出MPP数据库服务器配置参数的说明。

- [application_name](#)
- [array_nulls](#)
- [authentication_timeout](#)
- [backslash_quote](#)
- [block_size](#)
- [bonjour_name](#)
- [check_function_bodies](#)
- [client_encoding](#)
- [client_min_messages](#)
- [cpu_index_tuple_cost](#)
- [cpu_operator_cost](#)
- [cpu_tuple_cost](#)
- [cursor_tuple_fraction](#)
- [data_checksums](#)
- [DateStyle](#)
- [db_user_namespace](#)
- [deadlock_timeout](#)
- [debug_assertions](#)
- [debug_pretty_print](#)
- [debug_print_parse](#)
- [gp_ignore_error_table \(不推荐使用\)](#)
- [gp_initial_bad_row_limit](#)
- [gp_instrument_shmem_size](#)
- [gp_interconnect_debug_retry_interval](#)
- [gp_interconnect_fc_method](#)
- [gp_interconnect_hash_multiplier](#)
- [gp_interconnect_queue_depth](#)
- [gp_interconnect_setup_timeout](#)
- [gp_interconnect_snd_queue_depth](#)
- [gp_interconnect_type](#)
- [gp_log_format](#)
- [gp_log_fts](#)
- [gp_log_interconnect](#)
- [gp_log_gang](#)
- [gp_max_local_distributed_cache](#)
- [gp_max_packet_size](#)
- [gp_max_plan_size](#)
- [max_files_per_processes](#)
- [max_function_args](#)
- [max_identifier_length](#)
- [max_index_keys](#)
- [max_locks_per_transaction](#)
- [max_prepared_transactions](#)
- [max_resource_portals_per_transaction](#)
- [max_resource_queues](#)
- [max_stack_depth](#)
- [max_statement_mem](#)
- [memory_spill_ratio](#)
- [optimizer](#)
- [optimizer_array_expansion_threshold](#)
- [optimizer_analyze_root_partition](#)
- [optimizer_control](#)
- [optimizer_cte_inlining_bound](#)
- [optimizer_enable_associativity](#)
- [optimizer_enable_ma](#)

- [debug_print_plan](#)
- [debug_print_prelim_plan](#)
- [debug_print_rewritten](#)
- [debug_print_slice_table](#)
- [default_statistics_target](#)
- [default_tablespace](#)
- [default_text_search_config](#)
- [#default_transaction_deferrable](#)
- [default_transaction_isolation](#)
- [default_transaction_read_only](#)
- [dynamic_library_path](#)
- [effective_cache_size](#)
- [enable_bitmapscan](#)
- [enable_groupagg](#)
- [enable_hashagg](#)
- [enable_hashjoin](#)
- [gp_max_slices](#)
- [gp_motion_cost_per_row](#)
- [gp_recursive_cte](#)
- [gp_reject_percent_threshold](#)
- [gp_reraise_signal](#)
- [gp_resgroup_memory_policy](#)
- [gp_resource_group_bypass](#)
- [gp_resource_group_cpu_limit](#)
- [gp_resource_group_memory_limit](#)
- [gp_resource_manager](#)
- [gp_resqueue_memory_policy](#)
- [gp_resqueue_priority](#)
- [gp_resqueue_priority_cpucore_per_segment](#)
- [gp_resqueue_priority_sweeper_interval](#)
- [gp_role](#)
- [gp_safefswritesize](#)
- [ster_only_queries](#)
- [optimizer_force_agg_skew_avoidance](#)
- [optimizer_force_multistage_agg](#)
- [optimizer_force_three_stage_scalar_dqa](#)
- [optimizer_join_arity_for_associativity_commutativity](#)
- [optimizer_join_order](#)
- [optimizer_join_order_threshold](#)
- [optimizer_mdcache_size](#)
- [optimizer_metadata_caching](#)
- [optimizer_minidump](#)
- [optimizer_nestloop_factor](#)
- [optimizer_parallel_union](#)
- [optimizer_print_missing_stats](#)
- [optimizer_print_optimization_stats](#)
- [optimizer_sort_factor](#)
- [password_encryption](#)

- [enable_indexscan](#)
- [enable_mergejoin](#)
- [enable_nestloop](#)
- [enable_seqscan](#)
- [enable_sort](#)
- [enable_tidscan](#)
- [escape_string_warning](#)
- [explain_pretty_print](#)
- [extra_float_digits](#)
- [from_collapse_limit](#)
- [gp_adjust_selectivity_for_outerjoins](#)
- [gp_appendonly_compaction](#)
- [gp_appendonly_compaction_threshold](#)
- [gp_autostats_mode](#)
- [gp_autostats_mode_in_functions](#)
- [gp_autostats_on_change_threshold](#)
- [gp_cached_segworkers_threshold](#)
- [gp_segment_connect_timeout](#)
- [gp_segments_for_planner](#)
- [gp_server_version](#)
- [gp_server_version_num](#)
- [gp_session_id](#)
- [gp_set_proc_affinity](#)
- [gp_set_read_only](#)
- [gp_statistics_pullup_from_child_partition](#)
- [gp_statistics_use_fkeys](#)
- [gp_use_legacy_hash_ops](#)
- [gp_vmem_idle_resource_timeout](#)
- [gp_vmem_protect_limit](#)
- [gp_vmem_protect_segworker_cache_limit](#)
- [gp_workfile_compression](#)
- [gp_workfile_limit_files_per_query](#)
- [gp_workfile_limit_per_query](#)
- [password_hash_algorithm](#)
- [pljava_classpath](#)
- [pljava_classpath_insecure](#)
- [pljava_statement_cache_size](#)
- [pljava_release_lingering_savepoints](#)
- [pljava_vmoptions](#)
- [port](#)
- [random_page_cost](#)
- [readable_external_table_timeout](#)
- [repl_catchup_within_range](#)
- [replication_timeout](#)
- [regex_flavor](#)
- [resource_cleanup_gangs_on_wait](#)
- [resource_select_only](#)
- [runaway_detector_activation_percent](#)
- [search_path](#)
- [seq_page_cost](#)
- [server_encoding](#)
- [server_version](#)

- [gp_command_count](#)
- [gp_connection_send_timeout](#)
- [gp_content](#)
- [gp_create_table_random_default_distribution](#)
- [gp_dbid](#)
- [gp_debug_linger](#)
- [gp_default_storage_options](#)
- [gp_dynamic_partition_pruning](#)
- [gp_enable_adaptive_nestloop](#)
- [gp_enable_agg_distinct](#)
- [gp_enable_agg_distinct_pruning](#)
- [gp_enable_direct_dispatch](#)
- [gp_enable_exchange_default_partition](#)
- [gp_enable_fast_serialize](#)
- [gp_enable_global_deadlock_detector](#)
- [gp_workfile_limit_per_segment](#)
- [gpperfmon_port](#)
- [ignore_checksum_failure](#)
- [integer_datetimes](#)
- [IntervalStyle](#)
- [join_collapse_limit](#)
- [keep_wal_segments](#)
- [krb_caseins_users](#)
- [krb_server_keyfile](#)
- [lc_collate](#)
- [lc_ctype](#)
- [lc_messages](#)
- [lc_monetary](#)
- [lc_numeric](#)
- [lc_time](#)
- [listen_addresses](#)
- [local_preload_libraries](#)
- [lock_timeout](#)
- [log_autostats](#)
- [log_connections](#)
- [log_disconnections](#)
- [log_dispatch_stats](#)
- [server_version_num](#)
- [session_idle_timeout](#)
- [shared_buffers](#)
- [shared_preload_libraries](#)
- [ssl](#)
- [ssl_ciphers](#)
- [standard_conforming_strings](#)
- [statement_mem](#)
- [statement_timeout](#)
- [stats_queue_level](#)
- [superuser_reserved_connections](#)
- [tcp_keepalives_count](#)
- [tcp_keepalives_idle](#)
- [tcp_keepalives_interval](#)
- [temp_buffers](#)
- [TimeZone](#)
- [timezone_abbreviations](#)
- [track_activity_query_size](#)
- [transaction_isolation](#)
- [transaction_read_only](#)

- [gp_enable_gppperfmon](#)
- [gp_enable_group_ext_distinct_gather](#)
- [gp_enable_group_ext_distinct_pruning](#)
- [gp_enable_multiphase_agg](#)
- [gp_enable_predicate_propagation](#)
- [gp_enable_preunique](#)
- [gp_enable_query_metrics](#)
- [gp_enable_relsizes_collection](#)
- [gp_enable_segment_copy_checking](#)
- [gp_enable_sort_distinct](#)
- [gp_enable_sort_limit](#)
- [gp_external_enable_exec](#)
- [gp_external_max_segs](#)
- [gp_external_enable_filter_pushdown](#)
- [log_duration](#)
- [log_error_verbosity](#)
- [log_executor_stats](#)
- [log_hostname](#)
- [log_min_duration_statement](#)
- [log_min_error_statement](#)
- [log_min_messages](#)
- [log_parser_stats](#)
- [log_planner_stats](#)
- [log_rotation_age](#)
- [log_rotation_size](#)
- [log_statement](#)
- [log_statement_stats](#)
- [log_temp_files](#)
- [log_timezone](#)
- [log_truncate_on_rotation](#)
- [maintenance_work_mem](#)
- [max_appendonly_tables](#)
- [max_connections](#)
- [transform_null_equals](#)
- [unix_socket_directory](#)
- [unix_socket_group](#)
- [unix_socket_permissions](#)
- [update_process_title](#)
- [vacuum_cost_delay](#)
- [vacuum_cost_limit](#)
- [vacuum_cost_page_dirty](#)
- [vacuum_cost_page_hit](#)
- [vacuum_cost_page_miss](#)
- [vacuum_freeze_min_age](#)
- [validate_previous_free_tid](#)
- [verify_gpfdists_cert](#)
- [vmem_process_interrupt](#)
- [wal_receiver_status_interval](#)
- [writable_external_table_bufsize](#)
- [xid_stop_limit](#)
- [xid_warn_limit](#)

• gp_fts_probe_interval • gp_fts_probe_retrievals • gp_fts_probe_threadcount • gp_fts_probe_timeout • gp_global_deadlock_detector_period • gp_gpperfmon_send_interval • gpperfmon_log_alert_level • gp_hashjoin_tuples_per_bucket		• xmlbinary • xmloption

application_name

设置客户端会话的应用程序名称。 例如，如果通过psql连接，则将其设置为psql。 设置应用程序名称允许在日志消息和统计信息视图中报告。

取值范围	默认值	设置分类

取值范围	默认值	设置分类
string		master session reload

array_nulls

这可以控制数组输入解析器是否将未加引号的NULL识别为指定空数组元素。默认情况下，此选项处于启用状态，允许输入包含空值的数组值。3.0之前的MPP数据库版本不支持数组中的空值，因此将NULL视为指定字符串值为“NULL”的普通数组元素。

取值范围	默认值	设置分类
Boolean	on	master session reload

authentication_timeout

完成客户端认证的最大时间，这样可以防止挂起的客户端无限期占用连接。

取值范围	默认值	设置分类

取值范围	默认值	设置分类
任何有效的时间表达式 (数字和单位)	1分钟	local system restart

backslash_quote

这可以控制是否在字符串中可以用\表示引号。 代表引号的首选SQL标准是用“”表示，但是PostgreSQL历来也使用'。 但是，使用'会导致安全风险，因为在一些客户端字符集编码中，有很多多字节字符，其中最后一个字节等同于ASCII字符\。

取值范围	默认值	设置分类
on (总是允许 \')	safe_encoding	master
off (总是拒绝)		session
safe_encoding (只有客户端编码不允许多字节中的ASCII字符\才允许)		reload

block_size

报告磁盘块大小。

取值范围	默认值	设置分类
字节数	32768	read only

bonjour_name

指定Bonjour广播名称。默认情况下，使用计算机名称，指定为空字符串。如果服务器未支持Bonjour服务，则忽略此选项。

取值范围	默认值	设置分类
string	unset	master system restart

check_function_bodies

设置为off时，在CREATE FUNCTION期间禁用函数体字符串的验证。在从转储中恢复函数定义时，禁用验证有时可以避免诸如前向引用之类的问题。

取值范围	默认值	设置分类
Boolean	on	master session reload

client_encoding

设置客户端编码（字符集）。默认是使用与数据库相同的编码。 请参阅 PostgreSQL文档中的[支持的字符集](#)。

取值范围	默认值	设置分类
字符集	UTF8	master session reload

client_min_messages

控制哪些消息级别发送到客户端。每个级别包括跟它随后的所有级别，越往后的级别，发送的消息就越少。

取值范围	默认值	设置分类
DEBUG5	NOTICE	master
DEBUG4		session
DEBUG3		reload
DEBUG2		
DEBUG1		
LOG		
NOTICE		
WARNING		
ERROR		
FATAL		
PANIC		

cpu_index_tuple_cost

对于传统的查询优化器（planner），在索引扫描期间设置对处理每个索引行代价的估计。这是作为顺序页面提取代价的一部分来衡量的。

取值范围	默认值	设置分类

取值范围	默认值	设置分类
浮点数	0.005	master session reload

cpu_operator_cost

对于传统的查询优化器（planner），设置对处理WHERE语句中每个操作符代价的估计。这是作为顺序页面提取代价的一部分来衡量的。

取值范围	默认值	设置分类
浮点数	0.0025	master session reload

cpu_tuple_cost

对于传统的查询优化器（planner），设置对处理一个查询中每行（元组）代价的估计。这是作为顺序页面提取代价的一部分来衡量的。

取值范围	默认值	设置分类

取值范围	默认值	设置分类
浮点数	0.01	master session reload

cursor_tuple_fraction

告知传统查询优化器（planner）预期在游标查询中提取多少行，从而允许传统优化器使用此信息来优化查询计划。默认值为1表示获取所有行。

取值范围	默认值	设置分类
整数	1	master session reload

data_checksums

报告是否为数据库系统中的堆数据存储启用了校验和。 初始化数据库系统且无法更改时，将启用或禁用堆数据的校验和。

堆数据页存储堆表，目录表，索引和数据库元数据。 追加优化存储具有与此参数无关的内置校验和支持。

MPP数据库使用校验和来防止将文件系统中损坏的数据加载到由数据库进程管

理的内存中。 启用堆数据校验和时，MPP Database会在堆数据页写入磁盘时计算并存储校验和。 从磁盘检索页面时，将验证校验和。 如果验证失败，则会生成错误，并且不允许将页面加载到托管内存中。

如果ignore_checksum_failure配置参数已设置为on，则校验和验证失败会生成警告，但允许将页面加载到托管内存中。 如果页面随后更新，则会刷新到磁盘并复制到镜像。 这可能导致数据损坏传播到镜像并阻止完全恢复。 由于可能会丢失数据，因此只应在需要恢复数据时启用ignore_checksum_failure参数。 有关更多信息，请参阅[ignore_checksum_failure](#)。

取值范围	默认值	设置分类
Boolean	on	read only

DateStyle

设置日期和时间值的显示格式，以及解释模糊日期输入值的规则。该变量值包含两个独立的而部分：输出格式规范和输入输出规范中年月日的顺序。

取值范围	默认值	设置分类
<format>, <date style> 其中： <format> 是ISO, Postgres, SQL或German <date style> 是DMY, MDY或 YMD	ISO, MDY	master session reload

db_user_namespace

这启用了每个数据库的用户名。 如果打开，用户应该以`username@dbname`创建用户。 要创建普通的全局用户，只需要在客户端指定用户名时附加`@`。

取值范围	默认值	设置分类
Boolean	off	local system restart

deadlock_timeout

在检查以查看是否存在死锁情况之前等待锁的时间。 在一个比较重的服务器上，用户可能希望提高此值。 理想的情况下，设置的值应该超过用户的典型处理时间，以此提高在等待线程在决定检查死锁之前自动解锁的几率。

取值范围	默认值	设置分类
任何有效时间的表达式 (数字或者单位)。	1s	local system restart

debug_assertions

打开各种断言检查。

取值范围	默认值	设置分类
Boolean	off	local system restart

debug_pretty_print

缩进调试输出产生更可读但是更长的输出格式。 *client_min_messages* 或者 *log_min_messages* 必须是DEBUG1或者更低。

取值范围	默认值	设置分类
Boolean	on	master session reload

debug_print_parse

对于每一个执行的查询，打印出结果分析树。
*client_min_messages*或*log_min_messages*必须是DEBUG1或者更低。

取值范围	默认值	设置分类

取值范围	默认值	设置分类
Boolean	off	master session reload

debug_print_plan

对于每个执行的查询，打印出MPP并行查询执行计划。
*client_min_messages*或*log_min_messages*必须是DEBUG1或者更低。

取值范围	默认值	设置分类
Boolean	off	master session reload

debug_print_prelim_plan

对于每个执行的查询，打印出初步查询计划。
*client_min_messages*或*log_min_messages*必须是DEBUG1或者更低。

取值范围	默认值	设置分类

取值范围	默认值	设置分类
Boolean	off	master session reload

debug_print_rewritten

对于每个执行的查询，打印出查询重写输出。
*client_min_messages*或*log_min_messages*必须是DEBUG1或者更低。

取值范围	默认值	设置分类
Boolean	off	master session reload

debug_print_slice_table

对于每个执行的查询，打印MPP查询分片计划。
*client_min_messages*或*log_min_messages*必须是DEBUG1或者更低。

取值范围	默认值	设置分类

取值范围	默认值	设置分类
Boolean	off	master session reload

default_statistics_target

通过ALTER TABLE SET STATISTICS设置未具有列特定目标集的表列的默认统计信息采样目标（存储在公共值列表中的值的数量）。较大的值可能会提高Postgres查询优化器（规划器）估计的质量。

取值范围	默认值	设置分类
0 < 整数 < 10000	100	master session reload

default_tablespace

当CREATE命令没有明确指定一个表空间，会在默认的表空间创建对象（表和索引）。

取值范围	默认值	设置分类

取值范围	默认值	设置分类
表空间的名字	unset	master session reload

default_text_search_config

选择文本搜索功能的那些变体使用的文本搜索配置，这些变体没有指定配置的显式参数。 有关详细信息，请参阅[使用全文搜索](#)。 内置缺省值为pg_catalog.simple，但是如果可以识别与该语言环境匹配的配置，则initdb将使用与所选lc_ctype语言环境对应的设置初始化配置文件。

取值范围	默认值	设置分类
文本搜索配置的名称。	pg_catalog.simple	master session reload

default_transaction_deferrable

Note: 只有只读和可串行化事务才可以被推迟。 MPP数据库不支持SERIALIZABLE事务隔离级别，因此将default_transaction_deferrable设置为on对MPP数据库没有影响。

在可序列化隔离级别运行时，可延迟的只读SQL事务可能会在允许继续之前被延

迟。但是，一旦它开始执行，它不会产生确保可串行化所需的任何开销; 因此，序列化代码没有理由强制它因并发更新而中止，所以此选项适用于长时间运行的只读事务。

此参数控制每个新事务的默认可延迟状态。它目前对读写事务或低于可序列化的隔离级别的操作没有影响。默认为关闭。

取值范围	默认值	设置分类
Boolean	off	master session reload

default_transaction_isolation

控制每个新事务的默认隔离级别。MPP数据库将read uncommitted视为与read committed相同，并将serializable视为与repeatable read相同。

取值范围	默认值	设置分类
read committed	read committed	master
read uncommitted		session
repeatable read		reload
serializable		

default_transaction_read_only

default_transaction_read_only

控制每个新事务的默认只读状态。只读的SQL事务 不能修改非临时表。

取值范围	默认值	设置分类
Boolean	off	master session reload

dynamic_library_path

如果需要打开动态加载的模块，并且在CREATE FUNCTION或LOAD命令中指定的文件名没有目录部分（即：目录不包括斜杠），系统会搜索该路径以获取所需的文件。此时，PostgreSQL 内置编译的包库目录会替换 \$libdir。这是由标准PostgreSQL发行版提供的模块安装位置。

取值范围	默认值	设置分类
由冒号分隔的绝对目录 路径列表	\$libdir	local system restart

effective_cache_size

设置有关Postgres查询优化器（计划程序）的单个查询可用的磁盘高速缓存的有效大小的假设。这是用于估算使用指数的成本的因素; 较高的值使得更有可能使

用索引扫描，较低的值使得更有可能使用顺序扫描。 此参数对MPP服务器实例分配的共享内存大小没有影响，也不保留内核磁盘缓存; 它仅用于估算目的。

将此参数设置为32K块的数量（例如，对于16MB，为512），或指定有效缓存的大小（例如，对于1024个块，为'32MB'）。 gpconfig工具和SHOW命令以“MB”或“kB”为单位显示有效的高速缓存大小值。

取值范围	默认值	设置分类
浮点数	512 (16GB)	master session reload

enable_bitmapscan

启用或禁用Postgres查询优化器（规划器）使用位图扫描计划类型。 请注意，这与位图索引扫描不同。 位图扫描意味着索引将在适当的时候在内存中动态转换为位图，从而在针对非常大的表的复杂查询上提供更快索引性能。 当不同的索引列上有多个谓词时使用它。 可以比较每列的每个位图以创建所选元组的最终列表。

取值范围	默认值	设置分类
Boolean	on	master session reload

enable_groupagg

启用或禁用Postgres查询优化器（规划器）使用组聚合计划类型。

取值范围	默认值	设置分类
Boolean	on	master session reload

enable_hashagg

启用或禁用Postgres查询优化器（规划器）使用哈希聚合计划类型。

取值范围	默认值	设置分类
Boolean	on	master session reload

enable_hashjoin

启用或禁用Postgres查询优化器（规划器）使用哈希连接计划类型。

取值范围	默认值	设置分类

取值范围	默认值	设置分类
Boolean	on	master session reload

enable_indexscan

启用或禁用Postgres查询优化器（规划器）使用索引扫描计划类型。

取值范围	默认值	设置分类
Boolean	on	master session reload

enable_mergejoin

启用或禁用Postgres查询优化器（规划器）使用合并连接计划类型。合并连接基于将左侧和右侧表按顺序排序然后并行扫描的想法。因此，两种数据类型必须能够完全排序，并且连接运算符必须是只能成功处于排序顺序中“相同位置”的值对的连接运算符。实际上，这意味着连接运算符必须表现得像相等一样。

取值范围	默认值	设置分类

取值范围	默认值	设置分类
Boolean	off	master session reload

enable_nestloop

启用或禁用Postgres查询优化器（规划器）使用嵌套循环连接计划。不可能完全抑制嵌套循环连接，但如果有其他可用方法，则关闭此变量会阻止Postgres优化器使用它。

取值范围	默认值	设置分类
Boolean	off	master session reload

enable_seqscan

启用或禁用Postgres查询优化器（规划器）使用顺序扫描计划类型。不可能完全抑制顺序扫描，但如果有其他方法可用，则关闭此变量会阻止Postgres优化器使用。

取值范围	默认值	设置分类
Boolean	on	master session reload

enable_sort

启用或禁用Postgres查询优化器（规划器）使用显式排序步骤。完全禁止显式排序是不可能的，但如果有其他可用方法，则关闭此变量会阻止Postgres优化器使用。

取值范围	默认值	设置分类
Boolean	on	master session reload

enable_tidscan

启用或禁用Postgres查询优化器（规划器）使用元组标识符（TID）扫描计划类型。

取值范围	默认值	设置分类

取值范围	默认值	设置分类
Boolean	on	master session reload

escape_string_warning

打开的时候，如果在普通字符串文字（'...'语法）中出现反斜杠（\）,则会发出警告。转义字符语法（E'...'）应用于转义，因为在将来的版本中，普通字符串将具有字面上处理反斜杠的符合SQL标准的行为。

取值范围	默认值	设置分类
Boolean	on	master session reload

explain_pretty_print

确定 EXPLAIN VERBOSE 是否使用缩进或非缩进格式显示详细的查询树信息。

取值范围	默认值	设置分类

取值范围	默认值	设置分类
Boolean	on	master session reload

extra_float_digits

调整浮点值显示的位数，包括float4，float8，和几何数据类型。将参数将加到数位上。该值可以设置为高达2，包括部分有效位。这对于转储需要精确恢复的浮点数据尤其有用。或者设置为负以摒弃不需要的位。

取值范围	默认值	设置分类
integer	0	master session reload

from_collapse_limit

Postgres查询优化器（规划器）将子查询合并到上层查询中，如果生成的FROM列表只有这么多项。较小的值会减少计划时间，但可能会产生较差的查询计划。

取值范围	默认值	设置分类

取值范围	默认值	设置分类
1- <i>n</i>	20	master session reload

gp_adjust_selectivity_for_outerjoins

在外连接上启用NULL测试的选择性。

取值范围	默认值	设置分类
Boolean	on	master session reload

gp_appendonly_compaction

在VACUUM命令期间启用压缩segment文件。 禁用时，VACUUM仅将segment文件截断为EOF值，当前行为也是如此。 管理员可能希望在高I/O负载情况或低空间情况下禁用压缩。

取值范围	默认值	设置分类

取值范围	默认值	设置分类
Boolean	on	master session reload

gp_appendonly_compaction_threshold

指定在没有FULL选项（惰性VACUUM）的情况下运行VACUUM时隐藏行与触发压缩segment文件的总行的阈值比率（百分比）。如果segment上的segment文件中的隐藏行的比率小于此阈值，则不压缩段文件，并发出日志消息。

取值范围	默认值	设置分类
整数(%)	10	master session reload

gp_autostats_mode

指定使用ANALYZE触发自动统计信息收集的模式。 on_no_stats选项可以触发对任何没有统计信息的表上的CREATE TABLE AS SELECT，INSERT，或COPY操作的统计信息收集。

当受影响的行数超过由gp_autostats_on_change_threshold定义的阈值时，on_change选项才会触发统计信息收集。 可以使用on_change触发自动统

计信息收集的操作有：

CREATE TABLE AS SELECT

UPDATE

DELETE

INSERT

COPY

默认值是on_no_stats。

Note: 对于分区表来说，如果从分区表的顶级父表插入数据，则不会触发自动统计信息收集。

如果数据直接插入到分区表的叶表（数据的存储位置）中，则触发自动统计信息收集。 统计数据仅在叶表上收集。

取值范围	默认值	设置分类
none	on_no_stats	master
on_change		session
on_no_stats		reload

gp_autostats_mode_in_functions

指定使用过程语言函数中的ANALYZE语句触发自动统计信息收集的模式。
none选项禁用统计信息收集。 on_no_stats选项在任何没有现有统计信息表上的函数中执行的CREATE TABLE AS SELECT，INSERT，或COPY操作触发统

计信息收集。

只有当受影响的行数超过由gp_autostats_on_change_threshold定义的阈值时，on_change选项才会触发统计信息收集。 可以使用on_change触发自动信息统计收集功能的操作有：

CREATE TABLE AS SELECT

UPDATE

DELETE

INSERT

COPY

取值范围	默认值	设置分类
none	none	master
on_change		session
on_no_stats		reload

gp_autostats_on_change_threshold

当gp_autostats_mode设定为on_change时，指明自动统计信息收集的阈值。 当触发表操作影响超过此阈值的行数时，将添加ANALYZE并收集表的统计信息。

取值范围	默认值	设置分类

取值范围	默认值	设置分类
整数	2147483647	master session reload

gp_cached_segworkers_threshold

当用户使用MPP数据库启动会话并发出查询时，系统会在每个segment上创建工作进程的组或“gangs”来完成工作。完成工作后，除了由此参数设置的缓存数之外，将销毁段工作进程。较低的设置可以节省段主机上的系统资源，但更高的设置可以提高想要连续发出许多复杂查询的高级用户的性能。

取值范围	默认值	设置分类
整数 > 0	5	master session reload

gp_command_count

显示主服务器从客户端收到的命令数。请注意，单个SQL命令实际上可能在内部涉及多个命令，因此对于单个查询，计数器可能会增加多个。该计数器也由处理该命令的所有segment进程共享。

取值范围	默认值	设置分类
整数 > 0	1	read only

gp_connection_send_timeout

在查询处理期间向无响应的MPP数据库用户客户端发送数据的超时。 值为0将禁用超时，MPP数据库将无限期地等待客户端。 达到超时后，将使用以下消息取消查询：

Could not send data to client: Connection timed out.

取值范围	默认值	设置分类
秒数	3600 (1小时)	master system reload

gp_content

Segment的本地内容ID。

取值范围	默认值	设置分类
整数		read only

gp_create_table_random_default_distribution

使用不包含DISTRIBUTED BY子句的CREATE TABLE或CREATE TABLE AS创建MPP数据库表时，控制表的创建。

对于CREATE TABLE，如果参数的值为off（缺省值），并且创建表命令不包含DISTRIBUTED BY子句，MPP数据库将根据以下命令选择表分布键：

- 如果指定了LIKE或INHERITS子句，则MPP将从源表或父表复制分布键。
- 如果指定了PRIMARY KEY或UNIQUE约束，则MPP会选择所有键列的最大子集作为分布键。
- 如果既没有指定约束也没有指定LIKE或INHERITS子句，则MPP选择第一个合适的列作为分布键。（具有几何或用户定义数据类型的列不符合MPP分布键列的条件。）

如果参数的值设置为on，则当未指定DISTRIBUTED BY子句时，MPP数据库将遵循这些规则来创建表：

- 如果未指定PRIMARY KEY或UNIQUE列，则表的分布是随机的（DISTRIBUTED RANDOMLY）。即使表创建命令包含LIKE或INHERITS子句，表分发也是随机的。
- 如果指定了PRIMARY KEY或UNIQUE列，则还必须指定DISTRIBUTED BY子句。如果未将DISTRIBUTED BY子句指定为表创建命令的一部分，则该命令将失败。

对于不包含分布子句的CREATE TABLE AS命令：

- 如果Postgres查询优化器创建表，并且参数的值为off，则根据该命令确定表分发策略。
- 如果Postgres查询优化器创建表，并且参数的值为on，则表分发策略是随机的。
- 如果GPORCA创建表，则表分发策略是随机的。参数值没有影响。

有关Postgres查询优化器和GPORCA的信息，请参阅*MPP数据库管理员手册*中的“查询数据”。

取值范围	默认值	设置分类
boolean	off	master system reload

gp_dbid

对于segment，则为本地dbid。

取值范围	默认值	设置分类
整数		read only

gp_debug_linger

在致命的内部错误之后，MPP进程保留的秒数。

取值范围	默认值	设置分类
任何有效的时间表达式 (数字和单位)	0	master session reload

gp_default_storage_options

使用CREATE TABLE命令创建表时，设置以下表存储选项的默认值。

- appendoptimized

Note: 您可以使用appendoptimized=*value*语法指定追加优化的表存储类型。appendoptimized是appendonly传统存储选项的精简别名。MPP数据库在catalog中存储appendonly，并在列出追加优化表的存储选项时显示相同内容。

- blocksize
- checksum
- compresstype
- compresslevel
- orientation

将多个存储选项值指定为逗号分隔列表。

您可以使用此参数设置存储选项，而不是在CREATE TABLE命令的WITH中指定表存储选项。使用CREATE TABLE命令指定的表存储选项会覆盖此参数指定的值。

并非所有存储选项值组合都有效。如果指定的存储选项无效，则返回错误。有关表存储选项的信息，请参阅CREATE TABLE命令。

可以为数据库和用户设置默认值。如果服务器配置参数设置在不同的级别，则当用户登录到数据库并创建表时，这是表存储值的优先顺序，从最高到最低：

1. CREATE TABLE命令中使用WITH子句或ENCODING子句指定的值
 2. 使用ALTER ROLE...SET命令为用户设置的gp_default_storage_options的值
 3. 使用ALTER DATABASE...SET命令为数据库设置的gp_default_storage_options的值
 4. 使用gpconfig工具为MPP数据库系统设置的gp_default_storage_options的值
-

参数值不是累积的。例如，如果参数指定数据库的appendoptimized和compressstype选项并且用户登录并设置参数以指定orientation选项的值，则忽略在数据库级别设置的appendoptimized和compressstype值。

此示例ALTER DATABASE命令为数据库mytest设置默认orientation和compressstype表存储选项。

```
ALTER DATABASE mytest SET gp_default_storage_options = 'orientation=column, compress
```

使用面向列的表和RLE压缩在mytest数据库中创建追加优化的表。用户需要在WITH子句中仅指定appendoptimized=TRUE。

此示例gpconfig工具命令设置MPP数据库系统的默认存储选项。如果为多个表存储选项设置默认值，则该值必须用单引号括起来。

```
gpconfig -c 'gp_default_storage_options' -v 'appendoptimized=true, orientation=column'
```

此示例gpconfig工具命令显示参数的值。参数值必须在MPP数据库主数据库和所有segment之间保持一致。

```
gpconfig -s 'gp_default_storage_options'
```

取值范围	默认值	设置分类 ¹
appendoptimized= TRUE FALSE	appendoptimized=FALSE	master
blocksize= integer between 8192 and 2097152	blocksize=32768	session
checksum= TRUE FALSE	checksum=TRUE	reload
compresstype= ZLIB ZSTD QUICKLZ ² RLE_TYPE NONE	compresstype=none	
compresslevel= integer between 0 and 19	compresslevel=0	
orientation= ROW COLUMN	orientation=ROW	

Note: ¹当参数使用gpconfig工具设置在系统级时，为集合分类。
Note:

gp_dynamic_partition_pruning

启用可以动态消除分区扫描的计划。

取值范围	默认值	设置分类

取值范围	默认值	设置分类
on/off	on	master session reload

gp_enable_adaptive_nestloop

允许在Postgres查询优化器（规划器）的查询执行时使用称为“Adaptive Nestloop”的新类型的连接节点。 如果连接外侧的行数超过预先计算的阈值，这会导致Postgres优化器相比嵌套循环连接更偏爱哈希连接。 此参数提高了索引操作的性能，这些操作以前支持较慢的嵌套循环连接。

取值范围	默认值	设置分类
Boolean	on	master session reload

gp_enable_agg_distinct

启用或禁用两阶段聚合以计算单个不同限定的聚合。 这仅适用于包含单个不同限定聚合函数的子查询。

取值范围	默认值	设置分类

取值范围	默认值	设置分类
Boolean	on	master session reload

gp_enable_agg_distinct_pruning

启用或禁用三阶段聚合和连接以计算不同限定的聚合。 这仅适用于包含一个或多个不同限定聚合函数的子查询。

取值范围	默认值	设置分类
Boolean	on	master session reload

gp_enable_direct_dispatch

启用或禁用为访问单个segment上的数据的查询调度目标查询计划。 启用时，单个segment上的目标行的查询将仅将其查询计划分派到该segment（而不是所有segment）。 这明显缩短了限定查询的响应时间，因为没有涉及互连设置。直接分发确实需要主服务器上更多的CPU利用率。

取值范围	默认值	设置分类
Boolean	on	master system restart

gp_enable_exchange_default_partition

控制ALTER TABLE的EXCHANGE DEFAULT PARTITION子句的可用性。 参数的默认值为off。 如果在ALTER TABLE命令中指定了该子句，该子句不可用，MPP数据库将返回错误。

如果该值为on，则MPP数据库会返回一条警告，指出由于默认分区中的数据无效，交换默认分区可能会导致错误的结果。

Warning: 在交换默认分区之前，必须确保要交换的表中的数据（新的默认分区）对默认分区有效。 例如，新默认分区中的数据不得包含在分区表的其他叶子分区中有效的数据。 否则，使用GPORCA执行的交换的默认分区对分区表的查询可能会返回不正确的结果。

取值范围	默认值	设置分类
Boolean	off	master session reload

gp_enable_parallel_insert

当设置为on时，Postgres查询优化器（planner）计划单行插入，以便将它们直接发送到正确的segment实例（无需motion操作）。这显着提高了单行插入语句的性能。

取值范围	默认值	设置分类
Boolean	on	master session reload

gp_enable_global_deadlock_detector

控制是否启用MPP数据库全局死锁检测器来管理堆表上的并发UPDATE和DELETE操作以提高性能。见[全局死锁检测](#)。默认为off，全局死锁检测被禁用。

如果禁用全局死锁检测器（默认），MPP数据库将串行地对堆表执行并发更新和删除操作。

如果启用了全局死锁检测器，则允许并发更新，并且全局死锁检测器确定何时存在死锁，并通过终止与所涉及的最新事务关联的一个或多个后端进程来中断死锁。

取值范围	默认值	设置分类

取值范围	默认值	设置分类
Boolean	off	master system restart

gp_enable_gpperfmon

启用或禁用为MPP Command Center填充gpperfmon数据库的数据收集代理。

取值范围	默认值	设置分类
Boolean	off	local system restart

gp_enable_groupect_distinct_gather

启用或禁用将数据收集到单个节点以计算分组扩展查询的不限定聚合。 当此参数和gp_enable_groupect_distinct_pruning都启用时，Postgres查询优化器（规划器）使用成本更低的计划。

取值范围	默认值	设置分类

取值范围	默认值	设置分类
Boolean	on	master session reload

gp_enable_groupect_distinct_pruning

启用或禁用三阶段聚合和联接以计算分组扩展查询的不同限定聚合。通常，启用此参数会生成代价更小的查询计划，Postgres查询优化程序（优化器）将优先使用现有计划。

取值范围	默认值	设置分类
Boolean	on	master session reload

gp_enable_multiphase_agg

启用或禁用使用两阶段或三阶段并行聚合计划Postgres查询优化器（planner）。此方法适用于具有聚合的任何子查询。如果gp_enable_multiphase_agg关闭，则禁用gp_enable_agg_distinct和gp_enable_agg_distinct_pruning。

取值范围	默认值	设置分类
Boolean	on	master session reload

gp_enable_predicate_propagation

启用后，Postgres查询优化器（planner）会在表格在其分布键列上连接的情况下将查询谓词应用于两个表表达式。 在进行连接之前过滤两个表（如果可能）更有效。

取值范围	默认值	设置分类
Boolean	on	master session reload

gp_enable_preunique

为SELECT DISTINCT查询启用两阶段重复删除（不是SELECT COUNT(DISTINCT)）。 启用后，它会在移动之前添加一组额外的SORT DISTINCT计划节点。 在不同操作大大减少行数的情况下，这个额外的SORT DISTINCT比通过互连发送行的成本代价低得多。

取值范围	默认值	设置分类
Boolean	on	master session reload

gp_enable_query_metrics

启用查询指标的收集。 启用查询指标收集后，MPP数据库会在查询执行期间收集指标。 默认为关闭。

更改此配置参数后，必须重新启动MPP数据库才能使更改生效。

MPP数据库指标收集扩展在启用时，会将收集的指标通过UDP发送到 MPP Command Center代理¹。

MPP Command Center仅支持 MPP数据库。

取值范围	默认值	设置分类
Boolean	off	master system restart

gp_enable_resize_collection

如果表没有统计信息，则允许GPORCA和Postgres查询优化器（planner）使用

表的估计大小（pg_relation_size函数）。默认情况下，如果统计信息不可用，GPORCA和计划程序将使用默认值来估计行数。默认行为可以改善查询优化时间并减少繁重工作负载中的资源队列使用，但可能导致计划不理想。

对于分区表的根分区，将忽略此参数。启用GPORCA且根分区没有统计信息时，GPORCA始终使用默认值。您可以使用ANALYZE ROOTPARTITION收集根分区的统计信息。请参见ANALYZE。

取值范围	默认值	设置分类
Boolean	off	master session reload

gp_enable_segment_copy_checking

控制在使用COPY FROM...ON SEGMENT命令将数据复制到表中时是否检查表的分发策略（来自表DISTRIBUTED子句）。如果为true，则如果一行数据违反了segment实例的分发策略，则会返回错误。默认值为true。

如果值为false，则不检查分发策略。添加到表中的数据可能违反了segment实例的表分发策略。可能需要手动重新分配表数据。请参阅ALTER TABLE子句WITH REORGANIZE。

The parameter can be set for a database system or a session. The parameter cannot be set for a specific database.

取值范围	默认值	设置分类

取值范围	默认值	设置分类
Boolean	true	master session reload

gp_enable_sort_distinct

在排序时启用删除重复项。

取值范围	默认值	设置分类
Boolean	on	master session reload

gp_enable_sort_limit

在排序时启用LIMIT操作。 当计划最多需要第一行`limit_number`时，排序更有效。

取值范围	默认值	设置分类

取值范围	默认值	设置分类
Boolean	on	master session reload

gp_external_enable_exec

启用或禁用在segment主机上执行OS命令或脚本的外部表的使用（CREATE EXTERNAL TABLE EXECUTE语法）。 如果使用Command Center或MapReduce功能，则必须启用。

取值范围	默认值	设置分类
Boolean	on	master system restart

gp_external_max_segs

设置在外表操作期间扫描外部表数据的segment数量，目的是使系统不要超载扫描数据并从其他并发操作中夺走资源。 这仅适用于使用gpfdist://协议访问外部表数据的外部表。

取值范围	默认值	设置分类

取值范围	默认值	设置分类
integer	64	master session reload

gp_external_enable_filter_pushdown

从外部表读取数据时启用过滤器下推。 如果下推失败，则执行查询而不将过滤器推送到外部数据源（相反，MPP数据库会对结果应用相同的约束）。 有关更多信息，请参见[定义外部表](#)。

取值范围	默认值	设置分类
Boolean	on	master session reload

gp_fts_probe_interval

指定故障检测过程的轮询间隔（ftsprobe）。 ftsprobe过程将花费大约这段时间来检测段故障。

取值范围	默认值	设置分类

取值范围	默认值	设置分类
10 - 3600秒	1分钟	master system restart

gp_fts_probe_retries

指定在报告segment故障之前故障检测过程（ftsprobe）尝试连接到segment的次数。

取值范围	默认值	设置分类
integer	5	master system restart

gp_fts_probe_threadcount

指定要创建的ftsprobe线程数。 此参数应设置为等于或大于每个主机的segment数量的值。

取值范围	默认值	设置分类

取值范围	默认值	设置分类
1 - 128	16	master system restart

gp_fts_probe_timeout

指定故障检测过程（ftsprobe）允许的超时，以在声明它之前建立与segment的连接。

取值范围	默认值	设置分类
10 - 3600秒	20秒	master system restart

gp_global_deadlock_detector_period

指定全局死锁检测器后端进程的执行间隔（以秒为单位）。

取值范围	默认值	设置分类

取值范围	默认值	设置分类
5 - INT_MAX秒	120秒	master system reload

gp_log_fts

控制故障检测过程（ftsprobe）写入日志文件的详细信息量。

取值范围	默认值	设置分类
OFF	TERSE	master
TERSE		system
VERBOSE		restart
DEBUG		

gp_log_interconnect

控制写入日志文件的有关MPP数据库segment实例工作进程之间通信的信息量。
默认值为terse。 日志信息将写入master实例日志和segment实例日志。

增加日志记录量可能会影响性能并增加磁盘空间使用量。

取值范围	默认值	设置分类
off terse verbose debug	terse	master session reload

gp_log_gang

控制写入日志文件的有关查询工作进程创建和查询管理的信息量。 默认值为OFF，不记录信息。

取值范围	默认值	设置分类
OFF TERSE VERBOSE DEBUG	OFF	master session restart

gp_gpperfmon_send_interval

设置MPP数据库服务器将查询执行更新发送到用于为Command Center填充gpperfmon数据库的数据收集代理的频率。 在此间隔期间执行的查询操作通过

UDP发送到segment监视器代理。如果在长时间运行的复杂查询期间发现丢弃了过多的UDP数据包，则可以考虑增加此值。

取值范围	默认值	设置分类
任何有效的时间表达式 (数量和单位)	1sec	master system restart

gpperfmon_log_alert_level

控制将哪些消息级别写入gpperfmon日志。每个级别包括其后的所有级别。级别越靠后，发送到日志的消息越少。

Note: 如果安装了gpperfmon数据库并且正在监视数据库，则默认值为warning。

取值范围	默认值	设置分类
none warning error fatal panic	none	local system restart

gp hashjoin tuples per bucket

设置HashJoin操作使用的哈希表的目标密度。 较小的值往往会产生较大的哈希表，这可以提高连接性能。

取值范围	默认值	设置分类
integer	5	master session reload

gp_ignore_error_table (不推荐使用)

在CREATE EXTERNAL TABLE或COPY命令中指定不推荐使用的INTO ERROR TABLE子句时，控制MPP数据库行为。

默认值为false，如果在命令中指定了INTO ERROR TABLE子句，MPP数据库将返回错误。

如果值为true，则MPP数据库会忽略该子句，发出警告，并在没有INTO ERROR TABLE子句的情况下执行该命令。 在MPP数据库5.x及更高版本中，您可以使用内置SQL函数访问错误日志信息。 请参见 [CREATE EXTERNAL TABLE](#)或[COPY](#)命令。

您可以将此值设置为true，以避免在运行包含MPP数据库4.3.x INTO ERROR TABLE子句的CREATE EXTERNAL TABLE或COPY命令的应用程序时出现MPP数据库错误。

在MPP数据库5中不推荐使用INTO ERROR TABLE子句并将其删除。 在MPP数据库7中，此参数也将被删除，导致所有INTO ERROR TABLE调用产生语法错误。

取值范围	默认值	设置分类
Boolean	false	master session reload

gp_initial_bad_row_limit

对于参数值 n ，MPP数据库在使用COPY命令或从外部表导入数据时，如果处理的前 n 行包含格式错误，则停止处理输入行。 如果在前 n 行中处理有效行，则MPP数据库继续处理输入行。

将值设置为0将禁用此限制。

还可以为COPY命令或外部表定义指定SEGMENT REJECT LIMIT子句，以限制被拒绝的行数。

INT_MAX是可以作为整数存储在系统中的最大值。

取值范围	默认值	设置分类
整数 0 - INT_MAX	1000	master session reload

gp_instrument_shmem_size

为查询指标分配的共享内存量（以KB为单位）。默认值为5120，最大值为131072。启动时，如果`gp_enable_query_metrics`设置为on，则MPP数据库会在共享内存中分配空间以保存查询指标。该内存被组织为标题和插槽列表。所需的插槽数取决于并发查询的数量和每个查询的执行计划节点的数量。默认值5120基于MPP数据库系统，该系统最多执行约250个并发查询，每个查询120个节点。如果`gp_enable_query_metrics`配置参数关闭，或者插槽耗尽，则指标将保留在本地内存中而不是共享内存中。

取值范围	默认值	设置分类
整数0 - 131072	5120	master system restart

gp_interconnect_debug_retry_interval

指定在服务器配置参数`gp_log_interconnect`设置为DEBUG时记录MPP数据库interconnect调试消息的时间间隔（以秒为单位）。默认值为10秒。

日志消息包含有关MPP数据库segment实例工作进程之间的interconnect通信的信息。调试segment实例之间的网络问题时，这些信息可能会有所帮助。

取值范围	默认值	设置分类
1 <= 整数 < 4096	10	master session reload

gp_interconnect_fc_method

指定用于默认MPP数据库UDPIFC interconnect的流控制方法。

对于基于容量的流量控制，当接收器没有容量时，发送器不发送数据包。

基于损耗的流量控制基于基于容量的流量控制，并且还根据包丢失来调整发送速度。

取值范围	默认值	设置分类
CAPACITY	LOSS	master
LOSS		session
		reload

gp_interconnect_hash_multiplier

设置MPP数据库用于跟踪与默认UDPIFC interconnect的连接的哈希表的大小。此数字乘以segment数以确定哈希表中的桶数。 增加该值可能会增加复杂多切片查询的interconnect性能（同时在segment主机上消耗更多内存）。

取值范围	默认值	设置分类
2-25	2	master
		session
		reload

gp_interconnect_queue_depth

设置接收器上的排队的MPP数据库每方用于默认UDPIFC的interconnect数据量（当接收到数据但没有可用于接收数据的空间时，数据将被丢弃，并且发送器将需要重新发送它）。将深度从其默认值增加将导致系统使用更多内存，但可能会提高性能。将此值设置在1到10之间是合理的。具有数据倾斜的查询可能会随着队列深度的增加而更好地执行。增加这可能会从根本上增加系统使用的内存量。

取值范围	默认值	设置分类
1-2048	4	master session reload

gp_interconnect_setup_timeout

指定在超时之前等待MPP数据库interconnect完成设置的时间。

取值范围	默认值	设置分类
任何有效的时间表达式 (数量和单位)	2 hours	master session reload

gp_interconnect_snd_queue_depth

设置发件人上默认UDPIFC interconnect排队的每方的数据量。 将深度从其默认值增加将导致系统使用更多内存，但可能会提高性能。 此参数的合理值介于1和4之间。 增加该值可能会从根本上增加系统使用的内存量。

取值范围	默认值	设置分类
1 - 4096	2	master session reload

gp_interconnect_type

设置用于MPP数据库interconnect流量的网络协议。 UDPIFC指定使用UDP和流量控制来实现interconnect流量，并且是唯一支持的值。

UDPIFC（默认值）指定使用UDP和流量控制来实现interconnect流量。 使用[gp_interconnect_fc_method](#)指定interconnect流控制方法。

使用TCP作为interconnect协议，MPP数据库的上限为1000个segment实例 - 如果查询工作负载涉及复杂的多切片查询时，上限小于该值。

取值范围	默认值	设置分类
UDPIFC TCP	UDPIFC	local system restart

gp_log_format

指定服务器日志文件的格式。 如果使用gp_toolkit管理模式，则日志文件必须为CSV格式。

取值范围	默认值	设置分类
CSV	CSV	local
text		system
		restart

gp_max_local_distributed_cache

设置要在segment实例的后端进程内存中缓存的最大分布式事务日志条目数。

日志条目包含有关SQL语句正在访问的行的状态的信息。 该信息用于确定在MVCC环境中多个SQL语句同时执行时SQL事务可见的行。 通过提高行可见性确定过程的性能，本地缓存分布式事务日志条目可提高事务处理速度。

默认值适用于各种SQL处理环境。

取值范围	默认值	设置分类
整数	1024	local
		system
		restart

gp_max_packet_size

设置MPP数据库interconnect的元组序列化块大小。

取值范围	默认值	设置分类
512-65536	8192	master system restart

gp_max_plan_size

指定查询执行计划的最大未压缩总大小乘以计划中的Motion运算符（切片）数。如果查询计划的大小超过该值，则取消查询并返回错误。 值为0表示不监视计划的大小。

您可以指定以kB，MB或GB为单位的值。 默认单位是kB。 例如，值200是200kB。 值1GB与1024MB或1048576kB相同。

取值范围	默认值	设置分类
整数	0	master superuser session

gp_max_slices

指定可以由查询生成的最大切片数（在segment实例上执行的查询计划的部分）。如果查询生成的切片数超过指定数量，则MPP数据库将返回错误并且不执行查询。默认值为0，没有最大值。

执行生成大量切片的查询可能会影响MPP数据库性能。例如，在多个复杂视图上包含UNION或UNION ALL运算符的查询可以生成大量切片。您可以在查询上运行EXPLAIN ANALYZE以查看查询的切片统计信息。

取值范围	默认值	设置分类
0 - INT_MAX	0	master session reload

gp_motion_cost_per_row

设置Motion运算符的Postgres查询优化器（planner）成本估算，以将行从一个segment传输到另一个segment，以连续页面提取的成本的小数部分来衡量。如果为0，则使用的值是cpu_tuple_cost值的两倍。

取值范围	默认值	设置分类
浮点数	0	master session reload

gp_recursive_cte

控制SELECT [INTO]命令的WITH子句或DELETE，INSERT或UPDATE命令中RECURSIVE关键字的可用性。 该关键字允许命令的WITH子句中的子查询引用自身。 默认值为false，命令的WITH子句中不允许使用RECURSIVE关键字。

有关RECURSIVE关键字（Beta）的信息，请参阅[SELECT命令](#)和[WITH查询（公用表表达式）](#)。

可以为数据库系统，单个数据库或会话或查询设置该参数。

Note: 此参数之前名为gp_recursive_cte_prototype，但已重命名以反映实现的当前状态。

取值范围	默认值	设置分类
Boolean	true	master session restart

gp_reject_percent_threshold

对于COPY和外部表SELECT上的单行错误处理，设置在SEGMENT REJECT LIMIT *n* PERCENT开始计算之前处理的行数。

取值范围	默认值	设置分类

取值范围	默认值	设置分类
1- <i>n</i>	300	master session reload

gp_reraise_signal

如果启用，将在发生致命服务器错误时尝试生成core文件。

取值范围	默认值	设置分类
Boolean	on	master session reload

gp_resgroup_memory_policy

Note: 仅当资源管理设置为资源组时，gp_resgroup_memory_policy服务器配置参数才会生效。

由资源组用于管理查询运算符的内存分配。

设置为auto时，MPP数据库使用资源组内存限制在查询运算符之间分配内存，为非内存密集型运算符分配固定大小的内存，其余内存为内存密集型运算符。

当您指定eager_free时，MPP数据库通过将已完成处理的运算符释放的内存重新

分配给稍后查询阶段的运算符，从而更好地在运算符之间分配内存。

取值范围	默认值	设置分类
auto, eager_free	eager_free	local system superuser restart/reload

gp_resource_group_bypass

Note: 仅当资源管理设置为资源组时，gp_resource_group_bypass服务器配置参数才会生效。

启用或禁用对MPP数据库资源的资源组并发事务限制的实施。默认值为false，这会强制执行资源组事务限制。资源组管理资源，例如CPU，内存以及查询和外部组件（如PL / Container）使用的并发事务数。

您可以将此参数设置为true以绕过资源组并发事务限制，以便查询可以立即运行。例如，您可以将参数设置为true，以便会话运行系统目录查询或需要最少量资源的类似查询。

将此参数设置为true并运行查询时，查询将在此环境中运行：

- 查询在资源组内运行。查询的资源组分配不会更改。
- 查询内存配额大约为每个查询10 MB。内存从资源组共享内存或全局共享内存中分配。如果没有足够的共享内存可用于满足内存分配请求，则查询将失败。

可以为会话设置此参数。无法在事务或函数中设置该参数。

取值范围	默认值	设置分类
Boolean	false	session

gp_resource_group_cpu_limit

Note: 仅当资源管理设置为资源组时，gp_resource_group_cpu_limit服务器配置参数才会生效。

标识要分配给每个MPP数据库segment节点上的资源组的系统CPU资源的最大百分比。

取值范围	默认值	设置分类
0.1 - 1.0	0.9	local system restart

gp_resource_group_memory_limit

Note: 仅当资源管理设置为资源组时，gp_resource_group_memory_limit服务器配置参数才会生效。

标识要分配给每个MPP数据库segment节点上的资源组的系统内存资源的最大百分比。

取值范围	默认值	设置分类

取值范围	默认值	设置分类
0.1 - 1.0	0.7	local system restart

Note: 当基于资源组的资源管理处于活动状态时，分配给segment主机的内存将由活动primary segment平均共享。 当segment为primary时，MPP数据库会将内存分配给primary segment。 即使在故障转移情况下，primary segment的初始内存分配也不会更改。 这可能导致segment主机使用比gp_resource_group_memory_limit设置允许的更多内存。

例如，假设您的MPP数据库群集使用默认的gp_resource_group_memory_limit为0.7，而名为segghost1的segment主机有4个primary和4个mirror。 MPP数据库在segghost1上为每个primary segment分配整个系统内存的(0.7 / 4 = 0.175%)。 如果发生故障转移并且segghost1上的两个mirror故障转移成为primary，则原始4个primary中的每一个保留其0.175的存储器分配，并且两个新的primary segment分别被分配系统内存的(0.7 / 6 = 0.116%)。 segghost1在这种情况下的总体内存分配是

0.7 + (0.116 * 2) = 0.932%

它高于gp_resource_group_memory_limit设置中配置的百分比。

gp_resource_manager

标识当前在MPP数据库群集中启用的资源管理方案。 默认方案是使用资源队列。

取值范围	默认值	设置分类
group	queue	local
queue		system
		restart

gp_resqueue_memory_policy

Note: 仅当资源管理设置为资源队列时，gp_resqueue_memory_policy服务器配置参数才会生效。

启用MPP内存管理功能。 分配算法eager_free利用了并非所有运算符同时执行的事实（在MPP Database 4.2及更高版本中）。 查询计划分为几个阶段，MPP数据库急切地释放在该阶段执行结束时分配给前一阶段的内存，然后将急切释放的内存分配给新阶段。

When set to none, memory management is the same as in MPP Database releases prior to 4.1.

When set to auto, query memory usage is controlled by [statement_mem](#) and resource queue memory limits.

取值范围	默认值	设置分类
none, auto, eager_free	eager_free	local
		system
		restart/reload

gp_resqueue_priority

Note: 仅当资源管理设置为资源队列时，gp_resqueue_priority服务器配置参数才会生效。

启用或禁用查询优先级。禁用此参数时，不会在查询运行时计算现有优先级设置。

取值范围	默认值	设置分类
Boolean	on	local system restart

gp_resqueue_priority_cpucores_per_segment

Note: 仅当资源管理设置为资源队列时，gp_resqueue_priority_cpucores_per_segment服务器配置参数才会生效。

指定每个segment实例分配的CPU单元数。例如，如果MPP数据库群集具有配置有四个segment的10核心segment主机，请将segment实例的值设置为2.5。对于master实例，值为10。master主机通常只运行master实例，因此主机的值应反映所有可用CPU内核的使用情况。

设置不正确可能导致CPU使用率不足或查询优先级不按设计工作。

MPP数据计算设备V2的默认值为segment 4，master 25。

取值范围	默认值	设置分类

取值范围	默认值	设置分类
0.1 - 512.0	4	local system restart

gp_resqueue_priority_sweeper_interval

Note: 仅当资源管理设置为资源队列时，gp_resqueue_priority_sweeper_interval服务器配置参数才会生效。

指定清理程序进程评估当前CPU使用率的时间间隔。 当新语句变为活动状态时，将评估其优先级，并在达到下一个间隔时确定其CPU份额。

取值范围	默认值	设置分类
500 - 15000毫秒	1000	local system restart

gp_role

此服务器进程的角色。master设置为*dispatch*，segment设置为*execute*。

取值范围	默认值	设置分类
dispatch execute utility		read only

gp_safefswritesize

指定在非成熟文件系统中对追加优化表进行安全写入操作的最小尺寸。当指定大于零的字节数时，追加优化的写入器将填充数据添加到该数字，以防止由于文件系统错误导致的数据损坏。每个非成熟文件系统都具有已知的安全写入大小，在使用具有该类型文件系统的MPP数据库时必须在此处指定。这通常设置为文件系统的范围大小的倍数；例如，Linux ext3是4096字节，因此通常使用值32768。

取值范围	默认值	设置分类
整数	0	local system restart

gp_segment_connect_timeout

MPP interconnect将尝试通过网络连接到segment实例的超时时间。控制master和primary之间的网络连接超时，以及primary到mirror复制过程的网络连接超

时。

取值范围	默认值	设置分类
任何有效的时间表达式 (数量和单位)	10分钟	local system reload

gp_segments_for_planner

设置Postgres查询优化器（planner）在其成本和大小估计中假设的primary实例数。 如果为0，则使用的值是primary的实际数量。 此变量影响Postgres优化器对Motion运算符中每个发送和接收过程处理的行数的估计。

取值范围	默认值	设置分类
0- <i>n</i>	0	master session reload

gp_server_version

以字符串形式报告服务器的版本号。 版本修饰符参数可能会附加到版本字符串的数字部分，例如：*5.0.0 beta*。

取值范围	默认值	设置分类
字符串。示例：5.0.0	n/a	read only

gp_server_version_num

以整数形式报告服务器的版本号。对于每个版本，该数字始终保持增加，并可用于数字比较。主要版本按原样表示，次要版本和补丁版本为零填充，始终为两位数宽。

取值范围	默认值	设置分类
<i>Mmmpp</i> ，其中 <i>M</i> 是主要版本， <i>mm</i> 是零填充的次要版本， <i>pp</i> 是零填充的补丁版本。示例：50000	n/a	read only

gp_session_id

系统为客户端会话分配的ID号。首次启动master实例时从1开始计数。

取值范围	默认值	设置分类
1- <i>n</i>	14	read only

gp_set_proc_affinity

如果启用，当启动MPP服务器进程（postmaster）时，它将绑定到一个CPU。

取值范围	默认值	设置分类
Boolean	off	master system restart

gp_set_read_only

设置为on以禁用对数据库的写入。任何正在进行的事务必须在只读模式生效之前完成。

取值范围	默认值	设置分类
Boolean	off	master system restart

gp_statistics_pullup_from_child_partition

在Postgres查询优化器（planner）规划父表的查询时，允许使用子表中的统计信息。

取值范围	默认值	设置分类

取值范围	默认值	设置分类
Boolean	on	master session reload

gp_statistics_use_fkeys

启用后，允许Postgres查询优化器（planner）使用存储在系统目录中的外键信息来优化外键和主键之间的连接。

取值范围	默认值	设置分类
Boolean	off	master session reload

gp_use_legacy_hashops

对于使用DISTRIBUTED BY *key_column*子句定义的表，此参数控制用于在segment实例之间分发表数据的哈希算法。默认值为false，使用跳转一致性哈希算法。

将值设置为true使用与MPP数据库5.x和早期版本兼容的取模哈希算法。

取值范围	默认值	设置分类
Boolean	false	master session reload

gp_vmem_idle_resource_timeout

如果数据库会话空闲的时间超过指定的时间，则会话将释放系统资源（例如共享内存），但仍保持与数据库的连接。这允许一次更多并发连接到数据库。

取值范围	默认值	设置分类
任何有效的时间表达式 (数量和单位)	18s	master system reload

gp_vmem_protect_limit

Note: 仅当资源管理设置为资源队列时，gp_vmem_protect_limit服务器配置参数才会生效。

设置活动segment实例的所有postgres进程可以使用的内存量（以MB为单位）。如果查询导致超出此限制，则不会分配内存，查询将失败。请注意，这是一个本地参数，必须为系统中的每个segment（primary和mirror）设置。设置参数值时，仅指定数值。例如，要指定4096MB，请使用值4096.不要将单位MB添加到该值。

为了防止内存过度分配，这些计算可以估计安全的`gp_vmem_protect_limit`值。

首先计算值`gp_vmem`。这是主机上可用的MPP数据库内存

$$gp\_vmem = ((SWAP + RAM) - (7.5GB + 0.05 * RAM)) / 1.7$$

其中`SWAP`是主机交换空间，`RAM`是主机上的RAM，以GB为单位。

接下来，计算`max_acting_primary_segments`。这是由于故障而激活mirror时主机上可以运行的primary的最大数量。例如，如果mirror布置在4个主机块中，每个主机有8个primary，则单个segment主机故障将激活故障主机块中每个剩余主机上的两个或三个mirror。此配置的`max_acting_primary_segments`值为11（8个primary加上3个失败时激活的mirror）。

这是`gp_vmem_protect_limit`的计算。该值应转换为MB。

$$gp\_vmem\_protect\_limit = gp\_vmem / acting\_primary\_segments$$

对于生成大量工作文件的情况，这是计算工作文件的`gp_vmem`的计算。

$$gp\_vmem = ((SWAP + RAM) - (7.5GB + 0.05 * RAM - (300KB * total\_#\_workfiles))) / 1.7$$

有关监视和管理工作文件使用情况的信息，请参阅***MPP数据库管理员手册***。

根据`gp_vmem`值，您可以计算`vm.overcommit_ratio`操作系统内核参数的值。配置每个MPP数据库主机时，将设置此参数。

$$vm.overcommit\_ratio = (RAM - (0.026 * gp\_vmem)) / RAM$$

Note: Red Hat Enterprise Linux中内核参数`vm.overcommit_ratio`的默认值为50。

有关内核参数的信息，请参阅***MPP数据库安装手册***。

取值范围	默认值	设置分类
整数	8192	local system restart

gp_vmem_protect_segworker_cache_limit

如果查询执行程序进程消耗的数量超过此配置的数量，则在进程完成后，将不会高速缓存该进程以用于后续查询。 具有大量连接或空闲进程的系统可能希望减少此数量以释放segment上的更多内存。 请注意，这是一个本地参数，必须为每个segment设置。

取值范围	默认值	设置分类
MB	500	local system restart

gp_workfile_compression

指定在哈希聚合或哈希连接操作溢出到磁盘时创建的临时文件是否压缩。

如果您的MPP数据库安装使用串行ATA（SATA）磁盘驱动器，则启用压缩可能有助于避免使用IO操作使磁盘子系统过载。

取值范围	默认值	设置分类
Boolean	off	master session reload

gp_workfile_limit_files_per_query

设置每个segment的每个查询允许的临时溢出文件（也称为工作文件）的最大数量。溢出文件会在执行需要更多内存的查询时被创建。超出限制时终止当前查询。

将值设置为0（零）以允许无限数量的溢出文件。主会话重新加载

取值范围	默认值	设置分类
整数	100000	master session reload

gp_workfile_limit_per_query

设置允许单个查询用于在每个segment创建临时溢出文件的最大磁盘大小。默认值为0，表示不强制执行限制。

取值范围	默认值	设置分类
KB	0	master session reload

gp_workfile_limit_per_segment

设置允许所有正在运行的查询用于在每个segment创建临时溢出文件的最大总磁盘大小。默认值为0，表示不强制执行限制。

取值范围	默认值	设置分类
KB	0	local system restart

gpperfmon_port

设置所有数据收集代理（用于Command Center）与master通信的端口。

取值范围	默认值	设置分类

取值范围	默认值	设置分类
整数	8888	master system restart

ignore_checksum_failure

仅在启用了[data_checksums](#)时才有效。

MPP数据库使用校验和来防止将文件系统中已损坏的数据加载到由数据库进程管理的内存中。

默认情况下，当读取堆数据页时发生校验和验证错误时，MPP数据库会生成错误并阻止将页面加载到托管内存中。当ignore_checksum_failure设置为on并且校验和验证失败时，MPP Database会生成警告，并允许将页面读入托管内存。如果页面随后更新，则会将其保存到磁盘并复制到mirror。如果页眉已损坏，即使启用此选项，也会报告错误。

Warning: 将ignore_checksum_failure设置为on可能会传播或隐藏数据损坏或导致其他严重问题。但是，如果已检测到校验和失败且页眉未损坏，则将ignore_checksum_failure设置为on可能允许您绕过错误并恢复表中可能仍存在的未损坏元组。

默认设置为关闭，只能由超级用户更改。

取值范围	默认值	设置分类

取值范围	默认值	设置分类
Boolean	off	local system restart

integer_datetimes

报告PostgreSQL是否构建为支持64位整数日期和时间。

取值范围	默认值	设置分类
Boolean	on	read only

IntervalStyle

设置间隔值的显示格式。 值*sql_standard*生成与SQL标准区间文字匹配的输出。 当DateStyle参数设置为ISO时， *postgres*值会生成与8.4之前的PostgreSQL版本匹配的输出。

当DateStyle参数设置为非ISO输出时， *postgres_verbose*值会产生与3.3之前的MPP版本匹配的输出。

值*iso_8601*将生成与ISO 8601第4.4.3.2节中定义的指示符匹配时间间隔格式的输出。 有关详细信息，请参阅[PostgreSQL 8.4文档](#)。

取值范围	默认值	设置分类
postgres	postgres	master
postgres_verbose		session
sql_standard		reload
iso_8601		

join_collapse_limit

Postgres查询优化器（planner）会在总项不超过配置就显式将内部JOIN结构重写为FROM项列表。默认情况下，此变量的设置与*from_collapse_limit*相同，适用于大多数用途。将其设置为1可防止对内部JOIN进行任何重新排序。将此变量设置为介于1和*from_collapse_limit*之间的值可能有助于将计划时间与所选计划的质量进行权衡（更高的值会产生更好的计划）。

取值范围	默认值	设置分类
1- <i>n</i>	20	master
		session
		reload

keep_wal_segments

对于MPP数据库主镜像，如果发生检查点操作，则设置由活动MPP数据库主机保存的已处理WAL段文件的最大数量。

段文件用于在standby上同步活跃的master。

取值范围	默认值	设置分类
整数	5	master system reload superuser

krb_caseins_users

设置是否应对区分大小写的Kerberos用户名进行处理。 默认值区分大小写（关闭）。

取值范围	默认值	设置分类
Boolean	off	master system reload

krb_server_keyfile

设置Kerberos服务器密钥文件的位置。

取值范围	默认值	设置分类
路径和文件名	unset	master system restart

lc_collate

报告完成文本数据排序的区域设置。 初始化MPP数据库阵列时确定该值。

取值范围	默认值	设置分类
<系统依赖>		read only

lc_ctype

报告确定字符分类的区域设置。 初始化MPP数据库阵列时确定该值。

取值范围	默认值	设置分类
<系统依赖>		read only

lc_messages

设置消息的显示语言。 可用的语言环境取决于操作系统安装的内容 - 使用*locale -a*列出可用的语言环境。 默认值继承自服务器的执行环境。 在某些系统上，此区域设置类别不存在。 设置此变量仍然有效，但不会产生任何影响。 此外，有可能不存在所需语言的翻译消息。 在这种情况下，您将继续看到英文消息。

取值范围	默认值	设置分类
<系统依赖>		local system restart

lc_monetary

设置用于格式化货币金额的区域设置，例如使用`to_char`系列函数。 可用的语言环境取决于操作系统安装的内容 - 使用`locale -a`列出可用的语言环境。 默认值继承自服务器的执行环境。

取值范围	默认值	设置分类
<系统依赖>		local system restart

lc_numeric

设置用于格式化数字的语言环境，例如使用`to_char`系列函数。 可用的语言环境取决于操作系统安装的内容 - 使用`locale -a`列出可用的语言环境。 默认值继承自服务器的执行环境。

取值范围	默认值	设置分类
<系统依赖>		local system restart

lc_time

此参数目前不执行任何操作，但可能在将来执行。

取值范围	默认值	设置分类
<系统依赖>		local system restart

listen_addresses

指定服务器将侦听来自客户端应用程序的连接的TCP/IP地址 - 以逗号分隔的主机名和/或数字IP地址列表。 特殊条目*对应于所有可用的IP接口。 如果列表为空，则只能连接UNIX域套接字。

取值范围	默认值	设置分类
localhost, host names, IP addresses, * (all available IP interfa ces)	*	master system restart

local_preload_libraries

以逗号分隔的共享库文件列表，以便在客户端会话开始时进行预加载。

取值范围	默认值	设置分类
		local system restart

lock_timeout

尝试获取表，索引，行或其他数据库对象的锁时，中止任何等待时间超过指定毫秒数的语句。 时间限制在每次锁定获取尝试是独立的。 该限制既适用于显式锁定请求（例如LOCK TABLE，也适用于没有NOWAIT的SELECT FOR UPDATE），也适用于隐式获取的锁定。 如果log_min_error_statement设置

为ERROR或更低，则MPP数据库会记录超时的语句。 值为零（默认值）将关闭此锁定等待监视。

与statement_timeout不同，此超时只能在等待锁定时发生。 请注意，如果statement_timeout非零，则将lock_timeout设置为相同或更大的值是没有意义的，因为语句超时始终会先触发。

MPP数据库使用deadlock_timeout和gp_global_deadlock_detector_period来触发本地和全局死锁检测。 请注意，如果打开lock_timeout并将其设置为小于这些死锁检测超时的值，则MPP数据库将在该会话中触发死锁检查之前中止语句。

Note: 建议不要在postgresql.conf中设置lock_timeout，因为它会影响所有会话

取值范围	默认值	设置分类
0 - INT_MAX毫秒	0毫秒	master session reload

log_autostats

记录有关与gp_autostats_mode和gp_autostats_on_change_threshold相关的自动ANALYZE操作的信息。

取值范围	默认值	设置分类

取值范围	默认值	设置分类
Boolean	off	master session reload superuser

log_connections

这将向服务器日志输出一行，详细说明每个成功的连接。某些客户端程序（如psql）在确定是否需要密码时尝试连接两次，因此重复的“已接收连接”消息并不总是表示存在问题。

取值范围	默认值	设置分类
Boolean	off	local system restart

log_disconnections

这在客户端会话终止时在服务器日志中输出一行，并包括会话的持续时间。

取值范围	默认值	设置分类

取值范围	默认值	设置分类
Boolean	off	local system restart

log_dispatch_stats

设置为“on”时，此参数将添加一条日志消息，其中包含有关语句分派的详细信息。

取值范围	默认值	设置分类
Boolean	off	local system restart

log_duration

导致记录满足`log_statement`的每个已完成语句的持续时间。

取值范围	默认值	设置分类

取值范围	默认值	设置分类
Boolean	off	master session reload superuser

log_error_verbosity

控制记录的每条消息在服务器日志中写入的详细信息量。

取值范围	默认值	设置分类
TERSE DEFAULT VERBOSE	DEFAULT	master session reload superuser

log_executor_stats

对于每个查询，将查询执行程序的性能统计信息写入服务器日志。 这是一种原始的分析工具。 无法与log_statement_stats一起启用。

取值范围	默认值	设置分类
Boolean	off	local system restart

log_hostname

默认情况下，连接日志消息仅显示连接主机的IP地址。 打开此选项会导致记录MPP数据库主服务器的IP地址和主机名。 请注意，根据您的主机名解析设置，这可能会造成不可忽视的性能损失。

取值范围	默认值	设置分类
Boolean	off	master system restart

log_min_duration_statement

如果语句的持续时间大于或等于指定的毫秒数，则将语句及其持续时间记录在单个日志行上。 将此值设置为0将打印所有语句及其持续时间。 -1禁用该功能。 例如，如果将其设置为250，则将记录运行250毫秒或更长时间的所有SQL语句。 启用此选项可用于跟踪应用程序中未优化的查询。

取值范围	默认值	设置分类
毫秒数, 0, -1	-1	master session reload superuser

log_min_error_statement

控制是否还将在服务器日志中记录导致错误情况的SQL语句。 将记录导致指定级别或更高级别的错误的所有SQL语句。 默认为PANIC（有效关闭此功能以供正常使用）。 启用此选项有助于跟踪服务器日志中出现的任何错误的来源。

取值范围	默认值	设置分类
DEBUG5	ERROR	master
DEBUG4		session
DEBUG3		reload
DEBUG2		superuser
DEBUG1		
INFO		
NOTICE		
WARNING		
ERROR		
FATAL		
PANIC		

log_min_messages

控制将哪些消息级别写入服务器日志。 每个级别包括其后的所有级别。 级别越靠后，发送到日志的消息越少。

如果安装了MPP数据库PL/Container扩展。 此参数还控制PL/Container日志级别。 有关扩展的信息，请参阅[MPP PL/Container语言扩展语言扩展](#)。

取值范围	默认值	设置分类
DEBUG5	WARNING	master
DEBUG4		session
DEBUG3		reload
DEBUG2		superuser
DEBUG1		
INFO		
NOTICE		
WARNING		
LOG		
ERROR		
FATAL		
PANIC		

log_parser_stats

对于每个查询，将查询解析器的性能统计信息写入服务器日志。这是一种原始的分析工具。 无法与log_statement_stats一起启用。

取值范围	默认值	设置分类
Boolean	off	master session reload superuser

log_planner_stats

对于每个查询，将Postgres查询优化器（planner）的性能统计信息写入服务器日志。 这是一种原始的分析工具。 无法与log_statement_stats一起启用。

取值范围	默认值	设置分类
Boolean	off	master session reload superuser

log_rotation_age

确定MPP数据库将消息写入活动日志文件的时间。 经过这段时间后，文件将关闭并创建一个新的日志文件。 设置为零以禁用基于时间的新日志文件创建。

取值范围	默认值	设置分类
任何有效的时间表达式 (数量和单位)	1d	local system restart

log_rotation_size

确定触发旋转的单个日志文件的大小。当日志文件大小等于或大于此大小时，将关闭该文件并创建新的日志文件。设置为零以禁用基于大小的新日志文件的创建。

最大值为INT_MAX/1024。如果指定了无效值，则使用默认值。INT_MAX是可以作为整数存储在系统中的最大值。

取值范围	默认值	设置分类
kb	1048576	local system restart

log_statement

控制记录哪些SQL语句。DDL记录所有数据定义命令，如CREATE，ALTER和DROP命令。MOD记录所有DDL语句，以及INSERT，UPDATE，DELETE，TRUNCATE和COPY FROM。如果包含的命

令属于适当的类型，也会记录PREPARE和EXPLAIN ANALYZE语句。

取值范围	默认值	设置分类
NONE	ALL	master
DDL		session
MOD		reload
ALL		superuser

log_statement_stats

对于每个查询，将查询解析器，优化器和执行器的总体性能统计信息写入服务器日志。 这是一种原始的分析工具。

取值范围	默认值	设置分类
Boolean	off	master
		session
		reload
		superuser

log_temp_files

控制临时文件名和大小的记录。 可以为排序，哈希，临时查询结果和溢出文件

创建临时文件。删除时，每个临时文件都会在pg_log中创建一个日志条目。根据临时文件的来源，可以在master和/或segment上创建日志条目。
log_temp_files值为零会记录所有临时文件信息，而正值仅记录大小大于或等于指定KB数的文件。默认设置为-1，禁用日志记录。只有超级用户才能更改此设置。

取值范围	默认值	设置分类
整数	-1	local system restart

log_timezone

设置用于写入日志的时间戳的时区。与TimeZone不同，此值是系统范围的，因此所有会话都将一致地报告时间戳。默认值是unknown，这意味着使用系统环境指定的任何时区。

取值范围	默认值	设置分类
字符串	unknown	local system restart

log_truncate_on_rotation

截断（覆盖）而不是附加到任何现有的同名日志文件。 仅当由于基于时间的旋转而打开新文件时才会发生截断。 例如，将此设置与log_filename（例如gpseg#-%H.log）结合使用将导致生成二十四小时的日志文件，然后循环覆盖它们。 禁用时，将在所有情况下追加预先存在的文件。

取值范围	默认值	设置分类
Boolean	off	local system restart

maintenance_work_mem

指定要在维护操作中使用的最大内存量，例如VACUUM和CREATE INDEX。默认为16兆字节（16MB）。 较大的设置可能会提高清理和恢复数据库转储的性能。

取值范围	默认值	设置分类
整数	16	local system restart

max_appendonly_tables

设置可以写入或更新追加优化表的最大并发事务数。 超过最大值的事务将返回

错误。

计算的操作是INSERT，UPDATE，COPY和VACUUM操作。 限制仅适用于正在进行的交易。 一旦交易结束（中止或提交），就不再计入此限制。

对于针对分区表的操作，作为追加优化表并且被更改的每个子分区（子表）计为最大的单个表。 例如，分区表p_tbl被定义为具有三个子分区， 这些子分区是追加优化表p_tbl_ao1，p_tbl_ao2和p_tbl_ao3。 针对分区表p_tbl的INSERT或UPDATE命令将更改追加优化表p_tbl_ao1和p_tbl_ao2计为两个事务。

增加限制会在服务器启动时在master主机上分配更多共享内存。

取值范围	默认值	设置分类
整数 > 0	10000	master system restart

max_connections

与数据库服务器的最大并发连接数。 在MPP数据库系统中，用户客户端连接仅通过MPP master实例。 segment实例应该允许相比于master实例5-10倍的数量。 增加此参数时，还必须增加[max_prepared_transactions](#)。 有关限制并发连接的详细信息，请参阅[MPP数据库管理员手册](#)中的“配置客户端身份验证”。

增加此参数可能会导致MPP数据库请求更多共享内存。 有关MPP服务器实例共享内存缓冲区的信息，请参阅[shared_buffers](#)。

取值范围	默认值	设置分类
10- <i>n</i>	250 on master 750 on segments	local system restart

max_files_per_process

设置允许每个服务器子进程的最大同时打开文件数。如果内核强制执行每个进程的安全限制，则无需担心此设置。一些平台，如BSD，内核将允许单个进程打开比系统真正支持的更多文件。

取值范围	默认值	设置分类
整数	1000	local system restart

max_function_args

报告函数参数的最大数量。

取值范围	默认值	设置分类
整数	100	read only

max_identifier_length

报告最大标识符长度。

取值范围	默认值	设置分类
整数	63	read only

max_index_keys

报告索引键的最大数量。

取值范围	默认值	设置分类
整数	32	read only

max_locks_per_transaction

创建共享锁定表时，可以在 $max\_locks\_per\_transaction * (max\_connections + max\_prepared\_transactions)$ 对象上描述锁定，因此在任何时候都不能锁定这么多不同的对象。这不是任何一个事务所采用的锁数的硬限制，而是最大平均值。如果您的客户端在单个事务中触及许多不同的表，则可能需要提高此值。

取值范围	默认值	设置分类
整数	128	local
		system
		restart

max_prepared_transactions

设置可以同时处于准备状态的最大事务数。MPP在内部使用准备好的事务来确保各个segment的数据完整性。该值必须至少与master上的max_connections值一样大。segment实例应设置为与master相同的值。

取值范围	默认值	设置分类
整数	250 on master 250 on segments	local system restart

max_resource_portals_per_transaction

Note: 仅当资源管理设置为资源队列时，max_resource_portals_per_transaction服务器配置参数才会生效。

设置每个事务允许的同时打开的用户声明游标的最大数量。请注意，打开的游标将在资源队列中保留活动的查询槽。用于资源管理。

取值范围	默认值	设置分类
整数	64	master system restart

max resource queues

Note: 仅当资源管理设置为资源队列时，max_resource_portals_per_transaction服务器配置参数才会生效。

设置可以在MPP数据库系统中创建的最大资源队列数。 请注意，资源队列是系统范围的（角色也是如此），因此它们适用于系统中的所有数据库。

取值范围	默认值	设置分类
整数	9	master system restart

max_stack_depth

指定服务器执行堆栈的最大安全深度。 此参数的理想设置是内核强制实际的堆栈大小限制（由ulimit -s或本地等效项设置），减去一兆字节左右的安全边际。 将参数设置为高于实际内核限制将意味着失控的递归函数可能导致单个后端进程崩溃。

取值范围	默认值	设置分类
KB	2MB	local system restart

max statement mem

Note: 仅当资源管理设置为资源队列时，max_statement_mem服务器配置参数才会生效。

设置查询的最大内存限制。 由于将statement_mem设置得太高，有助于在查询处理期间避免segment主机上的内存不足错误。

当gp_resqueue_memory_policy=auto时， statement_mem和资源队列内存限制控制查询内存使用量。 考虑到单个segment主机的配置，请按如下方式计算此设置：

$$(\text{segghost\_physical\_memory}) / (\text{average\_number\_concurrent\_queries})$$

更改max_statement_mem和statement_mem时， 必须先更改max_statement_mem，或者先在postgresql.conf文件中列出。

取值范围	默认值	设置分类
KB	2000MB	master session reload superuser

memory_spill_ratio

Note: 仅当资源管理设置为资源组时，memory_spill_ratio服务器配置参数才会生效。

设置事务中内存密集型运算符的内存使用阈值百分比。 当事务达到此阈值时，它将溢出到磁盘。

默认memory_spill_ratio百分比是为分配给当前活动角色的资源组定义的值。您可以在会话级别设置memory_spill_ratio，以便在每个查询的基础上有选择地设置此限制。例如，如果您有一个特定的查询溢出到磁盘并需要更多内存，您可以选择设置更大的memory_spill_ratio来增加初始内存分配。

在会话级别设置memory_spill_ratio时，MPP数据库不会对新值执行语义验证，直到您下次执行查询为止。

取值范围	默认值	设置分类
0 - 100	20	master session reload

optimizer

运行SQL查询时启用或禁用GPORCA。默认on。如果禁用GPORCA，MPP数据库仅使用Postgres查询优化器。

GPORCA与Postgres查询优化器共存。启用GPORCA后，MPP数据库会尽可能使用GPORCA为查询生成执行计划。如果无法使用GPORCA，则使用Postgres查询优化器。

可以为数据库系统，单个数据库或会话或查询设置optimizer参数。

有关Postgres查询优化器和GPORCA的信息，请参阅[Querying Data](#)。

取值范围	默认值	设置分类

取值范围	默认值	设置分类
Boolean	on	master session reload

optimizer_analyze_root_partition

对于分区表，控制在表上运行ANALYZE命令时是否需要ROOTPARTITION关键字来收集根分区统计信息。 GPORCA在生成查询计划时使用根分区统计信息。Postgres查询优化器不使用这些统计信息。

参数的默认设置为on，ANALYZE命令可以在不使用ROOTPARTITION关键字的情况下收集根分区统计信息。 在根分区上运行ANALYZE时，或者在分区表的子叶子分区上运行ANALYZE并且其他子叶子分区具有统计信息时，将收集根分区统计信息。 当该值为off时，您必须运行ANALZYE ROOTPARTITION以收集根分区统计信息。

当服务器配置参数optimizer的值为on（默认值）时，此参数的值也应该为on。有关收集分区表的表统计信息的信息，请参阅ANALYZE。

有关Postgres查询优化器和GPORCA的信息，请参阅查询数据。

取值范围	默认值	设置分类
Boolean	on	master session reload

optimizer_array_expansion_threshold

启用GPORCA（默认值）并处理包含具有常量数组的谓词的查询时，optimizer_array_expansion_threshold参数会根据数组中的常量数限制优化过程。如果查询谓词中的数组包含多于参数指定的数字元素，则GPORCA会在查询优化期间禁用谓词转换为其析取范式。

默认值是100。

例如，当GPORCA执行包含具有100个以上元素的IN子句的查询时，GPORCA在查询优化期间不会将谓词转换为其析取范式，以减少优化时间消耗更少的内存。查询处理的差异可以在查询EXPLAIN计划的IN子句的过滤条件中看到。

更改此参数的值会更改较短的优化时间和较低的内存消耗之间的权衡，以及查询优化期间约束派生的潜在好处，例如冲突检测和分区消除。

可以为数据库系统，单个数据库或会话或查询设置该参数。

取值范围	默认值	设置分类
整数 > 0	25	master session reload

optimizer_control

控制是否可以使用SET，RESET命令或MPP数据库实用程序gpconfig更改服务器配置参数optimizer。如果启用了optimizer_control参数值，则用户可以设置优化程序参数。如果optimizer_control参数值为off，则无法更改优化程序参数。

取值范围	默认值	设置分类
Boolean	on	master system restart superuser

optimizer_cte_inlining_bound

启用GPORCA（默认值）时，此参数控制为公用表表达式（CTE）查询（包含WHERE子句的查询）执行的内联量。默认值0禁用内联。

可以为数据库系统，单个数据库或会话或查询设置该参数。

取值范围	默认值	设置分类
Decimal >= 0	0	master session reload

optimizer_enable_associativity

启用GPORCA（默认设置）时，此参数控制是否在查询优化期间启用连接关联变换。转换分析连接顺序。对于默认值off，仅启用用于分析连接顺序的GPORCA动态编程算法。连接关联变换很大程度上复制了较新的动态编程算法

的功能。

如果值为on，GPORCA可以在查询优化期间使用关联变换。

可以为数据库系统，单个数据库或会话或查询设置该参数。

有关GPORCA的信息，请参阅[关于GPORCA](#)。

取值范围	默认值	设置分类
Boolean	off	master session reload

optimizer_enable_master_only_queries

启用GPORCA（默认值）时，此参数允许GPORCA执行仅在MPP数据库master上运行的catalog查询。对于默认值off，只有Postgres查询优化器可以执行仅在MPP数据库master上运行的catalog查询。

可以为数据库系统，单个数据库或会话或查询设置该参数。

Note: 启用此参数会降低短期运行的catalog查询的性能。要避免此问题，请仅为会话或查询设置此参数。

有关GPORCA的信息，请参阅[关于GPORCA](#)。

取值范围	默认值	设置分类

取值范围	默认值	设置分类
Boolean	off	master session reload

optimizer_force_agg_skew_avoidance

启用GPORCA（默认值）时，此参数会影响GPORCA在生成3阶段聚合计划时考虑的查询计划备选方案。当值为true时，默认情况下，GPORCA仅考虑3阶段聚合计划，其中中间聚合使用GROUP BY和DISTINCT列进行分发，以减少处理偏斜的影响。

如果值为false，GPORCA还可以考虑使用GROUP BY列进行分发的计划。当处理偏斜存在时，这些计划可能表现不佳。

有关GPORCA的信息，请参阅[关于GPORCA](#)。

取值范围	默认值	设置分类
Boolean	true	master session reload

optimizer_force_multistage_agg

对于默认设置，启用GPORCA且此参数为true，GPORCA在生成此类计划备选项时为标量不同的合格聚合选择3阶段聚合计划。 当值为false时，GPORCA会根据成本选择而不是启发式选择。

可以为数据库系统，单个数据库或会话或查询设置该参数。

取值范围	默认值	设置分类
Boolean	true	master session reload

optimizer_force_three_stage_scalar_dqa

对于默认设置，GPORCA已启用且此参数为true，GPORCA会在生成此类计划备选项时选择具有多级聚合的计划。 当值为false时，GPORCA会根据成本选择而不是启发式选择。

可以为数据库系统，单个数据库或会话或查询设置该参数。

取值范围	默认值	设置分类
Boolean	true	master session reload

optimizer_join_arity_for_associativity_commutativ

optimizer_join_order_associativity_commutativity

该值是一个优化提示，用于限制在查询优化期间探索的连接关联性和连接可交换性转换的数量。该限制控制GPORCA在查询优化期间考虑的备选计划。例如，默认值18是GPORCA在优化期间n-ary连接运算符具有超过18个子节点时停止探索连接关联性和连接交换变换的优化提示。

对于具有大量连接的查询，指定较低的值可通过限制GPORCA评估的备用查询计划的数量来提高查询性能。但是，将值设置得太低可能会导致GPORCA生成次优执行的查询计划。

当optimizer_join_order参数设置为query或greedy时，此参数无效。

可以为数据库系统或会话设置此参数。

取值范围	默认值	设置分类
整数 > 0	18	local system reload

optimizer_join_order

启用GPORCA时，此参数通过指定要评估的连接排序备选类型来设置查询优化期间的连接排序的优化级别。

- query - 使用查询中指定的连接顺序。
- greedy - 根据连接中关系的最小基数，计算查询中指定的连接顺序和备选方案。
- exhaustive - 应用转换规则以查找和评估所有连接排序备选方案。

默认值是exhaustive。 将此参数设置为query或greedy可以生成次优查询计划。但是，如果管理员确信使用query或greedy设置生成了令人满意的计划， 则可以通过将参数设置为较低的优化级别来提高查询优化时间。

将此参数设置为query或greedy会覆盖optimizer_join_order_threshold和optimizer_join_arity_for_associativity_column数。

可以为单个数据库，会话或查询设置此参数。

取值范围	默认值	设置分类
query	exhaustive	master
greedy		session
exhaustive		reload

optimizer_join_order_threshold

启用GPORCA（默认设置）时，此参数设置GPORCA将使用基于动态编程的连接排序算法的最大连接子数。 您可以为单个查询或整个会话设置此值。

当optimizer_join_query参数设置为query或greedy时，此参数无效。

取值范围	默认值	设置分类
0 - 12	10	master
		session
		reload

optimizer_mdcache_size

设置MPP数据库master上GPORCA在查询优化期间用于缓存查询元数据（优化数据）的最大内存量，。 基于内存限制会话。 GPORCA使用默认设置在查询优化期间缓存查询元数据： 启用GPORCA并启用[optimizer_metadata_caching](#)。

默认值为16384（16MB）。 这是通过性能分析确定的最佳值。

您可以指定KB，MB或GB的值。 默认单位是KB。 例如，值16384是16384KB。 值1GB与1024MB或1048576KB相同。 如果值为0，则不限制高速缓存的大小。

可以为数据库系统，单个数据库或会话或查询设置此参数。

取值范围	默认值	设置分类
整数 >= 0	16384	master session reload

optimizer_metadata_caching

启用GPORCA（默认值）时，此参数指定GPORCA在查询优化期间是否在MPP数据库master上的内存中缓存查询元数据（优化数据）。 此参数的默认值为on，启用缓存。 缓存是基于会话的。 会话结束时，缓存将被释放。 如果查询元数据的数量超过缓存大小，则从缓存中逐出旧的未使用的元数据。

如果该值为off，则GPORCA在查询优化期间不会缓存元数据。

可以为数据库系统，单个数据库或会话或查询设置此参数。

服务器配置参数[optimizer_mdcache_size](#)控制查询元数据缓存的大小。

取值范围	默认值	设置分类
Boolean	on	master session reload

optimizer_minidump

GPORCA生成minidump文件以描述给定查询的优化上下文。 文件中的信息格式不易于调试或故障排除。 minidump文件位于主数据目录下，并使用以下命名格式：

Minidump_ *date_time*.mdp

minidump文件包含此查询相关信息：

- catalog对象，包括GPORCA所需的数据类型，表，运算符和统计信息
- 查询的内部表示（DXL）
- GPORCA制定的计划的内部代表（DXL）
- 传递给GPORCA的系统配置信息，例如服务器配置参数，成本和统计信息配置以及segment数
- 优化查询时生成的堆栈错误跟踪

将此参数设置为ALWAYS会为所有查询生成一个minidump。 将此参数设置为ONERROR以最小化总优化时间。

有关GPORCA的信息，请参阅[关于GPORCA](#)。

取值范围	默认值	设置分类
ONERROR	ONERROR	master
ALWAYS		session
		reload

optimizer_nestloop_factor

此参数向GPORCA添加成本因子，以在查询优化期间优先考虑哈希连接而不是嵌套循环连接。 在使用均匀分布的数据评估大量工作负载之后，选择了默认值1024。 应将1024视为此参数的实际上限设置。 如果您发现GPORCA选择哈希连接的频率超过预期，则减小该值以转换成本因子，转而使用嵌套循环连接。

The parameter can be set for a database system, an individual database, or a session or query.

取值范围	默认值	设置分类
INT_MAX > 1	1024	master
		session
		reload

optimizer_parallel_union

启用GPORCA（默认值）时，optimizer_parallel_union控制包含UNION或UNION ALL子句的查询的并行化数量。

当值为off时，默认GPORCA会生成一个查询计划，其中APPEND（UNION）运算符的每个子项与APPEND运算符位于同一个切片中。在查询执行期间，子节点以顺序方式执行。

当该值打开时，GPORCA会生成一个查询计划，其中重新分配motion节点位于APPEND（UNION）运算符下。在查询执行期间，子项和父APPEND运算符位于不同的切片上，允许APPEND（UNION）运算符的子项在segment实例上并行执行。

可以为数据库系统，单个数据库或会话或查询设置该参数。

取值范围	默认值	设置分类
boolean	off	master session reload

optimizer_print_missing_stats

启用GPORCA（默认设置）时，此参数控制有关查询缺少统计信息的列的表列信息的显示。默认值为true，将列信息显示给客户端。当值为false时，不会将信息发送到客户端。

在查询执行期间或使用EXPLAIN或EXPLAIN ANALYZE命令显示信息。

可以为数据库系统，单个数据库或会话设置该参数。

取值范围	默认值	设置分类

取值范围	默认值	设置分类
Boolean	true	master session reload

optimizer_print_optimization_stats

启用GPORCA（默认设置）时，此参数可以为查询的各种优化阶段记录GPORCA查询优化统计信息。默认值为off，请勿记录优化统计信息。要记录优化统计信息，必须将此参数设置为on，并且必须将参数client_min_messages设置为log。

- set optimizer_print_optimization_stats = on;
- set client_min_messages = 'log';

在查询执行期间或使用EXPLAIN或EXPLAIN ANALYZE命令记录信息。

可以为数据库系统，单个数据库或会话或查询设置此参数。

取值范围	默认值	设置分类
Boolean	off	master session reload

optimizer_sort_factor

启用GPORCA（默认值）时，optimizer_sort_factor控制在查询优化期间应用于排序操作的成本因子。默认值1指定默认排序成本因子。该值是默认因子的增加或减少的比率。例如，值2.0将成本因子设置为默认值的两倍，值0.5将因子设置为默认值的一半。

可以为数据库系统，单个数据库或会话或查询设置该参数。

取值范围	默认值	设置分类
Decimal > 0	1	master session reload

password_encryption

如果在CREATE USER或ALTER USER中指定了密码而未写入ENCRYPTED或UNENCRYPTED，则此选项确定是否要加密密码。

取值范围	默认值	设置分类
Boolean	on	master session reload

password_hash_algorithm

指定存储加密的MPP数据库用户密码时使用的加密哈希算法。默认算法是MD5。

有关设置密码哈希算法以保护用户密码的信息，请参阅*MPP数据库管理员手册*中的“保护MPP数据库中的密码”。

取值范围	默认值	设置分类
MD5	MD5	master
SHA-256		session
		reload
		superuser

pljava_classpath

包含jar文件或目录的冒号 (:)分隔列表，其中包含PL/Java函数所需的jar文件。必须指定jar文件或目录的完整路径，但\$GPHOME/lib/postgresql/java目录中的jar文件的路径可以省略。jar文件必须安装在所有MPP主机上的相同位置，并且可由gpadmin用户读取。

pljava_classpath参数用于在每个用户会话开始时组装PL/Java类路径。会话启动后添加的Jar文件不可用于该会话。

如果在pljava_classpath中指定了jar文件的完整路径，则会将其添加到PL/Java类路径中。指定目录时，该目录包含的任何jar文件都将添加到PL/Java类路径中。搜索不会下降到指定目录的子目录中。如果jar文件的名称包含在没有路径的pljava_classpath中，则jar文件必须位于\$GPHOME/lib/postgresql/java目录中。

Note: 如果要搜索的目录很多或者有大量的jar文件，性能会受到影响。

如果pljava_classpath_insecure为false，则设置pljava_classpath参数需要超级用户权限。当没有超级用户权限的用户执行代码时，在SQL代码中设置类路径将失败。必须先由超级用户或postgresql.conf文件设置pljava_classpath参数。更改postgresql.conf文件中的类路径需要重新加载（gpstop -u）。

取值范围	默认值	设置分类
string		master session reload superuser

pljava_classpath_insecure

控制是否可以由没有MPP数据库超级用户权限的用户设置服务器配置参数pljava_classpath。如果为true，则pljava_classpath可由常规用户设置。否则，pljava_classpath只能由数据库超级用户设置。默认值为false。

Warning: 启用此参数会使非管理员数据库用户能够运行未经授权的Java方法，从而暴露出安全风险。

取值范围	默认值	设置分类

取值范围	默认值	设置分类
Boolean	false	master session restart superuser

pljava_statement_cache_size

为预准备语句设置JRE MRU（最近使用）缓存的大小（以KB为单位）。

取值范围	默认值	设置分类
KB	10	master system restart superuser

pljava_release_lingering_savepoints

如果为true，则PL/Java函数中使用的延迟保存点将在函数退出时释放。 如果为false，则将回滚保存点。

取值范围	默认值	设置分类
Boolean	true	master system restart superuser

pljava_vmoptions

定义Java VM的启动选项。 默认值为空字符串（""）。

取值范围	默认值	设置分类
string		master system reload superuser

port

MPP实例的数据库侦听器端口。 master和每个segment都有自己的端口。 还必须在gp_segment_configuration中更改MPP系统的端口号。 您必须在更改端口号之前关闭MPP数据库系统。

取值范围	默认值	设置分类
任何有效的端口号	5432	local system restart

random_page_cost

设置Postgres查询优化器（planner）的非顺序读取磁盘页面的成本估计。这是连续页面提取的成本的倍数。较高的值使得更有可能使用顺序扫描，较低的值使得更有可能使用索引扫描。

取值范围	默认值	设置分类
浮点数	100	master session reload

readable_external_table_timeout

当SQL查询从外部表读取时，参数值指定当数据停止从外部表返回时，MPP数据库在取消查询之前等待的时间（以秒为单位）。

默认值0，指定没有超时。MPP数据库不会取消查询。

如果使用gpfdist的查询运行很长时间然后返回错误“间歇性网络连接问题”，则可以认为readable_external_table_timeout设定值。如果gpfdist在指定的时间内没有返回任何数据，MPP数据库将取消该查询。

取值范围	默认值	设置分类
整数 ≥ 0	0	master system reload

repl_catchup_within_range

对于MPP数据库主镜像，控制活跃从节点的更新。如果walsender尚未处理的WAL段文件数超过此值，MPP数据库将更新活跃从节点。

如果段文件的数量不超过该值，则MPP数据库会阻止更新，以允许walsender处理文件。如果已处理所有WAL段，则更新活动主站。 If the number of segment files does not exceed the value, MPP Database blocks updates to the to allow the walsender process the files. If all WAL segments have been processed, the active master is updated.

取值范围	默认值	设置分类
0 - 64	1	master system reload superuser

replication_timeout

对于MPP数据库主镜像，设置活动主服务器上的walsender进程等待来自备用主服务器上的walreceiver进程的状态消息的最长时间（以毫秒为单位）。如果未收到消息，则walsender会记录一条错误消息。

The [wal_receiver_status_interval](#) controls the interval between walreceiver status messages.

取值范围	默认值	设置分类
0 - INT_MAX	60000 ms (60 seconds)	master system reload superuser

regex_flavor

“扩展”设置可能对于与PostgreSQL 7.4之前版本的精确向后兼容非常有用。

取值范围	默认值	设置分类
advanced	advanced	master
extended		session
basic		reload

resource_cleanup_gangs_on_wait

Note: 仅当资源管理设置为资源队列时，resource_cleanup_gangs_on_wait服务器配置参数才会生效。

如果通过资源队列提交语句，请在锁定资源队列之前清除所有空闲的查询执行程序工作进程。

取值范围	默认值	设置分类
Boolean	on	master system restart

resource_select_only

Note: 仅当资源管理设置为资源队列时，resource_select_only服务器配置参数才会生效。

设置资源队列管理的查询类型。 如果设置为on，则评估SELECT，SELECT INTO，CREATE TABLE AS SELECT和DECLARE CURSOR命令。 如果设置为off，则还将评估INSERT，UPDATE和DELETE命令。

取值范围	默认值	设置分类
Boolean	off	master system restart

runaway_detector_activation_percent

Note: 仅当资源管理设置为资源队列时，runaway_detector_activation_percent服务器配置参数才会生效。

设置触发查询终止的MPP数据库vmem内存的百分比。 如果用于MPP数据库segment的vmem内存百分比超过指定值， MPP数据库将根据内存使用情况终止查询，从占用最大内存量的查询开始。 查询将终止，直到使用的vmem百分比低于指定的百分比。

使用服务器配置参数[gp_vmem_protect_limit](#)指定活动MPP数据库segment实例的最大vmem值。

例如，如果vmem内存设置为10GB， 并且runaway_detector_activation_percent的值为90（90 %）， 则当使用的vmem内存超过9GB时，MPP数据库将开始终止查询。

值0禁用基于所使用的vmem的百分比自动终止查询。

取值范围	默认值	设置分类
百分比(整数)	90	local system restart

search_path

指定在没有schema组件的简单名称引用对象时搜索schema的顺序。 当在不同schema中存在具有相同名称的对象时，将使用在搜索路径中首先找到的对象。 始终搜索系统catalog schema *pg_catalog*，无论路径中是否提及它。 在未指定

特定目标schema的情况下创建对象时，它们将被放置在搜索路径中列出的第一个schema中。 可以通过SQL函数`current_schemas()`检查搜索路径的当前有效值。 `current_schemas()`显示了`search_path`中出现的请求是如何解析的。

取值范围	默认值	设置分类
以逗号分隔的schema名称列表	<code>\$user,public</code>	master session reload

seq_page_cost

对于Postgres查询优化器（planner），设置作为一系列连续提取的一部分的磁盘页面提取的成本估计。

取值范围	默认值	设置分类
浮点数	1	master session reload

server_encoding

报告数据库编码（字符集）。 确定何时初始化MPP数据库阵列。 通常，客户只需要关注`client_encoding`的值。

取值范围	默认值	设置分类
<系统依赖>	UTF8	read only

server_version

报告此版本的MPP数据库所基于的PostgreSQL版本。

取值范围	默认值	设置分类
字符串	9.4.20	read only

server_version_num

报告此版本的MPP数据库基于整数的PostgreSQL版本。

取值范围	默认值	设置分类
整数	90420	read only

session_idle_timeout

终止处于IDLE状态时长超过指定毫秒数的数据库连接。0为关闭该限制。

取值范围	默认	设置分类

取值范围	默认	设置分类
毫秒数	0	master session reload

shared_buffers

设置MPP数据库segment实例用于共享内存缓冲区的内存量。 此设置必须至少为128KB且至少为max_connections的16KB倍。

每个MPP数据库segment实例根据segment配置计算并尝试分配一定量的共享内存。 shared_buffers的值是此共享内存计算的重要部分，但并非全部。 设置shared_buffers时，可能还需要调整操作系统参数SHMMAX或SHMALL的值。

操作系统参数SHMMAX指定单个共享内存分配的最大大小。 SHMMAX的值必须大于此值：

$$\text{shared\_buffers} + \text{other\_seg\_shmem}$$

other_seg_shmem的值是MPP数据库共享内存计算的部分，而shared_buffers值没有考虑该部分。 other_seg_shmem值将根据segment配置而有所不同。

使用默认的MPP数据库参数值，对于MPP数据库segment，other_seg_shmem的值大约为111MB，对于MPP数据库master，大约为79MB。

操作系统参数SHMALL指定主机上的最大共享内存量。 SHMALL的值必须大于

此值：

$$(num\_instances\_per\_host * (shared\_buffers + other\_seg\_shmem)) + other\_app\_shared\_mem$$

*other_app_shared_mem*的值是主机上其他应用程序和进程使用的共享内存量。

发生共享内存分配错误时，解决共享内存分配问题的可能方法是增加SHMMAX或SHMALL，或减少shared_buffers或max_connections。

有关参数SHMMAX和SHMALL的MPP数据库值的信息，请参阅*MPP数据库安装手册*。

取值范围	默认值	设置分类
整数 > 16K * <i>max_connections</i>	125MB	local system restart

shared_preload_libraries

以逗号分隔的共享库列表，这些共享库将在服务器启动时预加载。 PostgreSQL 过程语言库可以通过这种方式预加载，通常使用语法'*\$libdir/plXXX*'，其中XXX是pgsql，perl，tcl或python。通过预加载共享库，首次使用库时可以避免库启动时间。如果找不到指定的库，则服务器将无法启动。

取值范围	默认值	设置分类

取值范围	默认值	设置分类
		local system restart

ssl

启用SSL连接。

取值范围	默认值	设置分类
Boolean	off	master system restart

ssl_ciphers

指定允许在安全连接上使用的SSL密码列表。 有关支持的密码列表，请参见 openssl手册页。

取值范围	默认值	设置分类

取值范围	默认值	设置分类
字符串	ALL	master system restart

standard_conforming_strings

确定普通的字符串文字 ('...') 是否按字面意思处理反斜杠，如SQL标准中所指定的那样。 默认值为on。 关闭此参数可将字符串文字中的反斜杠视为转义字符而不是文字反斜杠。 应用程序可以检查此参数以确定如何处理字符串文字。 此参数的存在也可以作为支持转义字符串语法 (E'...') 的指示。

取值范围	默认值	设置分类
Boolean	on	master session reload

statement_mem

Note: 仅当资源管理设置为资源队列时，statement_mem服务器配置参数才会生效。

为每个查询分配segment主机内存。 使用此参数分配的内存量不能超过[max_statement_mem](#)或提交查询的资源队列的内存限制。

当`gp_resqueue_memory_policy=auto`时，`statement_mem`和资源队列内存限制控制查询内存使用量。

如果查询需要额外的内存，则使用磁盘上的临时溢出文件。

该计算可用于估计各种情况的合理值。

(gp_vmem_protect_limitGB * .9) / max_expected_concurrent_queries

将`gp_vmem_protect_limit`设置为8192MB（8GB）并假设最多包含10%缓冲的40个并发查询

(8GB * .9) / 40 = .18GB = 184MB

更改`max_statement_mem`和`statement_mem`时，必须先更改`max_statement_mem`，或者先在`postgresql.conf`文件中列出。

取值范围	默认值	设置分类
KB	128MB	master session reload

statement_timeout

中止任何超过指定毫秒数的语句。 0关闭限制。

取值范围	默认值	设置分类

取值范围	默认值	设置分类
毫秒数	0	master session reload

stats_queue_level

Note: 仅当资源管理设置为资源队列时，stats_queue_level服务器配置参数才会生效。

收集有关数据库活动的资源队列统计信息。

取值范围	默认值	设置分类
Boolean	off	master session reload

superuser_reserved_connections

确定为MPP数据库超级用户保留的连接插槽数。

取值范围	默认值	设置分类

取值范围	默认值	设置分类
整数 < <i>max_connection</i> s	3	local system restart

tcp_keepalives_count

在连接被认为死亡之前，可能会丢失多少个Keepalive。 值0使用系统默认值。 如果不支持TCP_KEEPCNT，则此参数必须为0。

将此参数用于不在primary和mirror之间的所有连接。

取值范围	默认值	设置分类
丢失的Keepalive数量	0	local system restart

tcp_keepalives_idle

在空闲连接上发送Keepalive之间的秒数。 值0使用系统默认值。 如果不支持TCP_KEEPIDLE，则此参数必须为0。

将此参数用于不在primary和mirror之间的所有连接。

取值范围	默认值	设置分类
秒数	0	local system restart

tcp_keepalives_interval

在重新传输之前等待keepalive响应的秒数。 值0使用系统默认值。 如果不支持TCP_KEEPINTVL，则此参数必须为0。

将此参数用于不在primary和mirror之间的所有连接。

取值范围	默认值	设置分类
number of seconds	0	local system restart

temp_buffers

以块为单位设置最大内存，以允许每个数据库会话使用临时缓冲区。 这些是仅用于访问临时表的会话本地缓冲区。 可以在单个会话中更改设置，但只能在会话中首次使用临时表之前更改。 在实际上不需要大量临时缓冲区的会话中设置大值的成本只是每个块的缓冲区描述符，或每个增量大约64个字节。 但是，如果实际使用了缓冲区，则将消耗额外的32768字节。

您可以将此参数设置为32K块的数量（例如，1024以允许32MB用于缓冲区），或指定允许的最大内存量（例如，对于1536块，为'48MB'）。 gpconfig实用程序和SHOW命令报告临时缓冲区允许的最大内存量。

取值范围	默认值	设置分类
整数	1024 (32MB)	master session reload

TimeZone

设置显示和解释时间戳的时区。 默认设置是使用系统环境指定的任何时区。 请参阅PostgreSQL文档中的[日期/时间关键字](#)。

取值范围	默认值	设置分类
时区缩写		local restart

timezone_abbreviations

设置服务器为日期时间输入接受的时区缩写集合。 默认值为Default，这是一个适用于世界大部分地区的集合。 可以为特定安装定义Australia和India以及其他集合。 可能的值是存储在\$GPHOME/share/postgresql/timezonesets/中的配置文件的名称。

要将MPP数据库配置为使用自定义时区集合，请将包含时区定义的文件复制到MPP数据库master和segment主机上的\$GPHOME/share/postgresql/timezonesets/目录中。然后将服务器配置参数timezone_abbreviations的值设置为该文件。例如，要使用包含默认时区和WIB（Waktu Indonesia Barat）时区的文件custom。

1. 从目录\$GPHOME/share/postgresql/timezonesets/复制文件Default到文件custom。将文件Asia.txt中的WIB时区信息添加到custom。
2. 将文件custom复制到MPP数据库master和segment主机上的\$GPHOME/share/postgresql/timezonesets/目录。
3. 将服务器配置参数timezone_abbreviations的值设置为custom。
4. 重新加载服务器配置文件（gpstop -u）。

取值范围	默认值	设置分类
字符串	默认值	master session reload

track_activity_query_size

设置存储在系统catalog视图pg_stat_activity的current_query列中的查询文本的最大长度限制。最小长度为1024个字符。

取值范围	默认值	设置分类

取值范围	默认值	设置分类
整数	1024	local system restart

transaction_isolation

设置当前事务隔离级别。

取值范围	默认值	设置分类
read committed serializable	read committed	master session reload

transaction_read_only

设置当前事务的只读状态。

取值范围	默认值	设置分类

取值范围	默认值	设置分类
Boolean	off	master session reload

transform_null_equals

启用时，表达式`expr = NULL`（或`NULL = expr`）被视为`expr IS NULL`，即，如果`expr`计算为空值，则返回`true`，否则返回`false`。正确的符合SQL规范的`expr = NULL`行为始终返回`null`（未知）。

取值范围	默认值	设置分类
Boolean	off	master session reload

unix_socket_directory

指定服务器要侦听来自客户端应用程序的连接的UNIX域套接字的目录。

取值范围	默认值	设置分类

取值范围	默认值	设置分类
目录路径	unset	local system restart

unix_socket_group

设置UNIX域套接字的拥有组。默认情况下，这是一个空字符串，它使用当前用户的默认组。

取值范围	默认值	设置分类
UNIX组名称	unset	local system restart

unix_socket_permissions

设置UNIX域套接字的访问权限。UNIX域套接字使用通常的UNIX文件系统权限集。 请注意，对于UNIX域套接字，只有写权限才对。

取值范围	默认值	设置分类

取值范围	默认值	设置分类
数字UNIX文件权限模式 (由chmod或umask命令接受)	511	local system restart

update_process_title

每次服务器收到新的SQL命令时，都可以更新进程标题。通常由ps命令查看进程标题。

取值范围	默认值	设置分类
Boolean	on	local system restart

vacuum_cost_delay

超出成本限制时vacuum进程将休眠的时间长度。 0禁用基于成本的vacuum延迟功能。

取值范围	默认值	设置分类

取值范围	默认值	设置分类
毫秒 < 0 (以10的倍数表示)	0	local system restart

vacuum_cost_limit

导致vacuum进程进入睡眠状态的累积成本。

取值范围	默认值	设置分类
整数 > 0	200	local system restart

vacuum_cost_page_dirty

vacuum修改先前干净的块时收取的估计成本。 它表示将脏块再次刷新到磁盘所需的额外I/O.

取值范围	默认值	设置分类

取值范围	默认值	设置分类
整数 > 0	20	local system restart

vacuum_cost_page_hit

vacuum共享缓冲区缓存中找到的缓冲的估计成本。 它表示锁定缓冲池，查找共享哈希表和扫描页面内容的成本。

取值范围	默认值	设置分类
整数 > 0	1	local system restart

vacuum_cost_page_miss

vacuum必须从磁盘读取的缓冲的估计成本。 这表示锁定缓冲池，查找共享哈希表，从磁盘读取所需块并扫描其内容的代价。

取值范围	默认值	设置分类

取值范围	默认值	设置分类
整数 > 0	10	local system restart

vacuum_freeze_min_age

指定VACUUM在扫描表时决定是否用*FrozenXID*替换事务ID时应使用的截止年龄（在事务中）。

有关VACUUM和事务ID管理的信息，请参阅*MPP数据库管理员手册*和[PostgreSQL文档](#)中的“管理数据”。

取值范围	默认值	设置分类
整数 0-1000000000000	100000000	local system restart

validate_previous_free_tid

启用验证空闲元组ID（TID）列表的测试。该列表由MPP数据库维护和使用。MPP数据库通过确保当前空闲元组的先前空闲TID是有效的空闲元组来确定空闲TID列表的有效性。默认值为true，启用测试。

如果MPP数据库检测到空闲TID列表中的损坏，则会重建空闲TID列表，并记录警告，并且检查失败的查询会返回警告。 MPP数据库尝试执行查询。

取值范围	默认值	设置分类
Boolean	true	master session reload

verify_gpfdists_cert

使用gpfdists协议定义MPP数据库外部表以使用SSL安全性时，此参数控制是否启用SSL证书身份验证。 默认设置为true，当MPP数据库与gpfdist实用程序通信以从外部数据源读取数据或将数据写入外部数据源时，将启用SSL身份验证。

值false将禁用SSL证书身份验证。 忽略这些SSL异常：

- Greenpum数据库不信任gpfdist使用的自签名SSL证书。
- SSL证书中包含的主机名与运行gpfdist的主机名不匹配。

您可以将值设置为false以在测试MPP数据库外部表与提供外部数据的gpfdist实用程序之间的通信时禁用身份验证。

Warning: 通过不验证gpfdists SSL证书禁用SSL证书身份验证会暴露安全风险。

有关gpfdists协议的信息，请参阅[gpfdists:// Protocol](#)。 有关运行gpfdist实用程序的信息，请参阅[gpfdist](#)。

取值范围	默认值	设置分类

取值范围	默认值	设置分类
Boolean	true	master session reload

vmem_process_interrupt

在MPP数据库查询执行期间，为查询预留vmem内存之前，启用检查中断。在为查询保留进一步的vmem之前，请检查查询的当前会话是否具有挂起的查询取消或其他挂起的中断。这确保了更具响应性的中断处理，包括查询取消请求。默认为关闭。

取值范围	默认值	设置分类
Boolean	off	master session reload

wal_receiver_status_interval

对于MPP数据库主镜像，设置发送到活动master的walreceiver进程状态消息之间的间隔（以秒为单位）。在高负载下，时间可能会更长。

[replication_timeout](#)的值控制walsender进程等待walreceiver消息的时间。

取值范围	默认值	设置分类
整数 0- INT_MAX/1000	10 sec	master system reload superuser

writable_external_table_bufsize

MPP数据库用于网络通信的缓冲区大小（以KB为单位），例如gpfdist实用程序和外部Web表（使用http）。MPP数据库在写入数据之前将数据存储在缓冲区中。有关gpfdist的信息，请参阅*MPP数据库实用程序手册*。

取值范围	默认值	设置分类
整数 32 - 131072 (32KB - 128MB)	64	local session reload

xid_stop_limit

发生事务ID环绕的ID之前的事务ID数。达到此限制时，MPP数据库会停止创建新事务，以避免因事务ID环绕而导致数据丢失。

取值范围	默认值	设置分类
整数 10000000 - 200000000	1000000000	local system restart

xid_warn_limit

[xid_stop_limit](#)指定的限制之前的事务ID数。 当MPP数据库达到此限制时，它会发出警告以执行VACUUM操作，以避免因事务ID环绕而导致数据丢失。

取值范围	默认值	设置分类
整数 10000000 - 200000000	500000000	local system restart

xmlbinary

指定如何在XML数据中编码二进制值。 例如，当bytea值转换为XML时。 二进制数据可以转换为base64编码或十六进制编码。 默认值为base64。

可以为数据库系统，单个数据库或会话设置该参数。

取值范围	默认值	设置分类
base64	base64	master
hex		session
		reload

xmloption

指定是否将XML数据视为执行隐式解析和序列化的操作的XML文档（document）或XML内容片段（content）。默认为content。

此参数会影响xml_is_well_formed()执行的验证。如果值是document，则函数将检查格式良好的XML文档。如果值是content，则函数检查格式良好的XML内容片段。

Note: 包含文档类型声明（DTD）的XML文档不被视为有效的XML内容片段。如果xmloption设置为content，则包含DTD的XML不被视为有效的XML。

要将包含DTD的字符串转换为xml数据类型，请将xmlparse函数与document关键字一起使用，或将xmloption值更改为document。

可以为数据库系统，单个数据库或会话设置该参数。MPP数据库中也提供了为会话设置此选项的SQL命令。

```
SET XML OPTION { DOCUMENT | CONTENT }
```

取值范围	默认值	设置分类
document	content	master
content		session
		reload

内置函数摘要

MPP数据库支持内置函数和运算符，包括可用于窗口表达式的分析函数和窗口函数。 有关使用内置MPP数据库函数的信息，请参阅*MPP数据库管理员手册*中的“使用函数和运算符”。

- [MPP数据库函数类型](#)
- [内置函数和运算符](#)
- [JSON函数和操作符](#)
- [窗口函数](#)
- [高级聚合函数](#)
- [文本搜索函数和操作符](#)
- [范围函数和运算符](#)

Parent topic: [MPP数据库参考手册](#)

MPP数据库函数类型

MPP数据库评估SQL表达式中使用的函数和运算符。 某些函数和运算符只允许

在master服务器上执行，因为它们可能导致MPP数据库segment实例中的不一致。该表描述了MPP数据库函数类型。

Table 1. MPP数据库中的函数

函数类型	MPP是否支持	描述	注释
IMMUTABLE	是	仅依赖于其参数列表中的信息。给定相同的参数值，始终返回相同的结果。	
STABLE	在大多数情况下是的	在单个表扫描中，对相同的参数值返回相同的结果，但结果将通过SQL语句进行更改。	结果取决于数据库查找或参数值。current_timestamp系列函数是STABLE; 值在执行中不会改变。
VOLATILE	受限制的	函数值可以在单个表扫描中更改。例如: random(),timeofday()。	任何具有副作用的函数都是易变的，即使其结果是可预测的。例如: setval()。

在MPP数据库中，数据跨segment分割 - 每个segment是不同的PostgreSQL数据库。为了防止出现不一致或意外的结果，如果它们包含SQL命令或以任何方式修改数据库，则不要在segment级别执行归类为VOLATILE的函数。例如，不允许在MPP数据库中对分布式数据执行诸如setval()之类的函数，因为它们可能导致segment实例之间的数据不一致。

为确保数据一致性，可以在master服务器上评估和运行的语句中安全地使用VOLATILE和STABLE函数。例如，以下语句在master上运行（没有FROM子句的语句）：

```
SELECT setval('myseq', 201);
SELECT foo();
```

如果语句具有包含分布式表的FROM子句，并且FROM子句中的函数返回一组行，则该语句可以在segment上运行：

```
SELECT * from foo();
```

MPP数据库不支持返回表引用（rangeFuncs）的函数或使用refCursor数据类型的函数。

内置函数和运算符

下表列出了PostgreSQL支持的内置函数和运算符的类别。 MPP数据库支持所有函数和运算符，如PostgreSQL，但STABLE和VOLATILE函数除外，它们受[MPP数据库函数类型](#)中的限制。 有关这些内置函数和运算符的更多信息，请参阅PostgreSQL文档的[函数和运算符](#)部分。

Table 2. 内置函数和运算符

操作符/函数类别	VOLATILE函数	STABLE函数	限制
逻辑操作符			
比较操作符			
数学函数和操作符	random setseed		

操作符/函数类别	VOLATILE函数	STABLE函数	限制
字符串函数和操作符	所有内建转换函数	convert pg_client_encoding	
二进制字符串和函数操作符			
位串函数和操作符			
模式匹配			
数据类型格式函数		to_char to_timestamp	
日期/时间函数和操作符	timeofday	age current_date current_time current_timestamp localtime localtimestamp now	
枚举支持函数			
几何函数和操作符			
网络地址函数和操作符			

操作符/函数类别	VOLATILE函数	STABLE函数	限制
序列操作函数	nextval() setval()		
条件表达式			
数组函数和操作符		所有数组函数	
聚合函数			
子查询表达式			
行和数组比较			
返回集合函数	generate_series		
系统信息函数		所有会话信息函数 所有访问特权查询函数 所有模式可见性查询函数 所有系统目录信息函数 所有注释信息函数 所有事务ID和快照	

操作符/函数类别	VOLATILE函数	STABLE函数	限制
系统管理函数	set_config pg_cancel_backend pg_reload_conf pg_rotate_logfile pg_start_backup pg_stop_backup pg_size_pretty pg_ls_dir pg_read_file pg_stat_file	current_setting 所有数据库对象尺寸函数	注意：函数pg_column_size显示存储值所需的字节，可能使用TOAST压缩。
XML函数和类似函数的表达式		cursor_to_xml(cursor refcursor, count int, nulls boolean, tablefor est boolean, targetns text) cursor_to_xmlschema(cursor refcursor, nul	

操作符/函数类别	VOLATILE函数	STABLE函数	限制
		tableforest boolean, targetns text) database_to_xml(nulls boolean, tableforest boolean, targetns text) database_to_xmlschema(nulls boolean, tableforest boolean, targetns text) database_to_xml_and_xmlschema(nulls boolean, tableforest boolean, targetns text) query_to_xml(query text, nulls boolean, tableforest boolean, targetns text) query_to_xmlschema(query text, nulls boolean, tableforest boolean, targetns text) query_to_xml_and_xmlschema(query text, nulls boolean, tableforest boolean, targetns text)	

操作符/函数类别	VOLATILE函数	STABLE函数	限制
		schema_name, nulls boolean, tableforest boolean, targetns text) schema_to_xmlschema(schema_name, nulls boolean, tableforest boolean, targetns text) schema_to_xml_and_xmlschema(schema_name, nulls boolean, tableforest boolean, targetns text) table_to_xml(tbl regclass, nulls boolean, tableforest boolean, targetns text) table_to_xmlschema(tbl regclass, nulls boolean, tableforest boolean, targetns text) table_to_xml_and_xmlschema(tbl regclass, nulls boolean, tableforest boolean, targetns text) xmlagg(xml)	

操作符/函数类别	VOLATILE函数	STABLE函数	限制
		xmlelement(name name [, xmlattributes(value [AS attname] [, ...]) [, content, ...]) xmlexists(text, xml) xmlforest(content [AS name] [, ...]) xml_is_well_formed(text) xml_is_well_formed_document(text) xml_is_well_formed_content(text) xmlparse ({ DOCUMENT CONTENT } value) xpath(text, xml) xpath(text, xml, text[]) xpath_exists(text, xml) xpath_exists(text, xml , text[])	

操作符/函数类别	VOLATILE函数	STABLE函数	限制
		xmlroot(xml, version text no value [, standalone yes no no value]) xmlserialize ({ DOCUMENT CONTENT } value AS type) xml(text) text(xml) xmlcomment(xml) xmlconcat2(xml, xml)	

JSON函数和操作符

MPP数据库包含用于创建和操作JSON数据的内置函数和运算符。

- [JSON操作符](#)
- [JSON创建函数](#)
- [JSON聚合函数](#)
- [JSON处理函数](#)

Note: 对于json数据类型值，即使JSON对象包含重复键，也会保留所有键/值

对。对于重复键，JSON处理函数将最后一个值视为可操作的值。对于jsonb数据类型，不保留重复的对象键。如果输入包含重复键，则仅保留最后一个值。参考 [关于JSON数据](#)。

JSON操作符

此表描述了可用于json和jsonb数据类型的运算符。

Table 3. json和jsonb操作符

操作符	右操作数类型	描述	示例	示例结果
->	int	获取JSON数组元素（从零索引）。	'{"a":"foo"},"b":"bar"},"c":"baz"}'::json->2	{"c":"baz"}
->	text	通过键获取JSON对象字段。	'{"a":{"b":"foo"}}'::json->'a'	{"b":"foo"}
->>	int	获取JSON数组元素作为text。	'[1,2,3]'::json->>2	3
->>	text	获取JSON对象字段作为text。	'{"a":1,"b":2}'::json->>'b'	2
#>	text[]	在指定的路径获取JSON对象。	'{"a":{"b":{"c":"foo"}}}'::json#>{'a,b'}	{"c":"foo"}
#>>	text[]	以指定路径获取JSON对象作为text。	'{"a":[1,2,3],"b":[4,5,6]}'::json#>>{'a,2'}	3

Note: 对于json和jsonb数据类型，这些运算符都有并行变体。 字段，元素和路径提取运算符返回与其左侧输入（json或jsonb）相同的数据类型，但指定为返回text的那些除外，它将值强制转换为text。 如果JSON输入没有与请求匹配的正确结构，则字段，元素和路径提取操作符将返回NULL，而不是失败; 例如，

如果不存在这样的元素。

需要jsonb数据类型作为左操作数的运算符如下表所述。 其中许多运算符可以由jsonb运算符类索引。 有关jsonb包含和存在语义的完整描述，请参阅[jsonb容器与存在](#)。 有关如何使用这些运算符有效地索引jsonb数据的信息，请参阅[jsonb索引](#)。

Table 4. jsonb操作符

操作符	右操作数类型	描述	示例
@>	jsonb	左JSON值是否包含正确的值？	'{"a":1, "b":2}'::jsonb @> '{"b":2}'::jsonb
<@	jsonb	左JSON值是否包含在右值内？	'{"b":2}'::jsonb < @ '{"a":1, "b":2}'::jsonb
?	text	键/元素字符串是否存在于JSON值中？	'{"a":1, "b":2}'::jsonb ? 'b'
?	text[]	是否存在任何这些键/元素字符串？	'{"a":1, "b":2, "c":3}'::jsonb ? array ['b', 'c']
?&	text[]	是否存在所有这些键/元素字符串？	'["a", "b"]'::jsonb ?& array ['a', 'b']

下表中的标准比较运算符仅适用于jsonb数据类型，不适用于json数据类型。 它们遵循[jsonb索引](#)中描述的B树操作的排序规则。

Table 5. jsonb比较操作符

操作符	描述
<	小于
>	大于
<=	小于等于
>=	大于等于
=	等于
<>或!=	不等于

Note: !=运算符在解析器阶段转换为<>。不可能实现执行不同操作的!=和<>运算符。

JSON创建函数

此表描述了创建json数据类型值的函数。（目前，jsonb没有等效函数，但您可以将其中一个函数的结果转换为jsonb。）

Table 6. JSON创建函数

函数	描述	示例	示例结果
to_json(anyelement)	以JSON对象的形式返回值。数组和复合体以递归方式处理，并转换为数组和对象。如果输入包含从类型到json的强制转换，则使用强制转换函数执行转换;否则，将生成JSON标量值。对于除数字，布尔值或空值之外的任何标量类型，将使用正确引用和转义文本表示，以使其成为有效的JSON字符串。	to_json('Fred said "Hi."':text)	"Fred said \"Hi.\""

函数	描述	示例	示例结果
array_to_json(anyarray [, pretty_bool])	将数组作为JSON数组返回。 多维数组成为数组的JSON数组。 如果pretty_bool为true，则会在dimension-1元素之间添加换行符。	array_to_json('{1,5},{99,100}':int[])	[[1, 5],[99, 10 0]]
row_to_json(record [, pretty_bool])	将行作为JSON对象返回。 如果pretty_bool为true，则会在level-1元素之间添加换行符。	row_to_json(row(1,'foo'))	{"f1":1, "f2": "foo"}
json_built_array(VARIADIC "any")	从VARIADIC参数列表构建可能异构类型的JSON数组。	json_built_array(1,2,'3',4,5)	[1, 2, " 3", 4, 5]

函数	描述	示例	示例结果
json_build_object(VARIADIC "any")	从VARIADIC参数列表中构建JSON对象。 参数列表按顺序获取并转换为一组键/值对。	json_build_object('foo', 1, 'bar', 2)	{"foo": 1, "bar": 2}
json_object(text[])	从文本数组中构建JSON对象。 该数组必须是一维或二维数组。 一维数组必须具有偶数个元素。 元素被视为键/值对。 对于二维数组，每个内部数组必须具有恰好两个元素，这些元素被视为键/值对。	json_object({'a', 1, 'b', 'def', 'c', 3.5}) json_object({'{a, 1}', '{b, "def"', '{c, 3.5}'})	{"a": 1, "b": "def", "c": 3.5}

函数	描述	示例	示例结果
json_object(keys text[], values text[])	从文本数组中构建JSON对象。 这种形式的json_object从两个独立的数组中成对获取键和值。 在所有其他方面，它与单参数形式相同。	json_object('{a, b}', '{1, 2}')	{"a": "1", "b": "2"}

Note: array_to_json和row_to_json与to_json具有相同的行为，除了提供漂亮的打印选项。 为to_json描述的行为同样适用于由其他JSON创建函数转换的每个单独的值。

Note: [hstore](#)扩展具有从hstore到json的强制转换，因此通过JSON创建函数转换的hstore值将表示为JSON对象，而不是原始字符串值。

JSON聚合函数

此表显示函数将记录聚合到JSON对象数组和值对聚合到JSON对象

Table 7. JSON聚合函数

函数	参数类型	返回类型	描述
json_agg(record)	record	json	将记录聚合为JSON对象数组。
json_object_agg(name, value)	("any", "any")	json	将名称/值对聚合为JSON对象。

JSON处理函数

此表显示可用于处理json和jsonb值的函数。

许多这些处理函数和运算符将JSON字符串中的Unicode转义转换为适当的单个字符。 如果输入数据类型是jsonb，则这不是问题，因为转换已经完成。 但是，对于json数据类型输入，这可能会导致抛出错误。 请参阅 [关于JSON数据](#)。

Table 8. JSON处理函

函数	返回类型	描述	示例	示例结果
json_array_length(json) jsonb_array_length(jsonb)	int	返回最外层JSON数组中的元素数。	json_array_length('[1,2,3,{"f1":1,"f2":[5,6]},4]')	5
json_each(json) jsonb_each(jsonb)	setof key text, value json setof key text, value jsonb	将最外层的JSON对象扩展为一组键/值对。	select * from json_each('{"a":"foo", "b":"bar"}')	key value ----+----- a "foo" b "bar"

函数	返回类型	描述	示例	示例结果
json_each_text(json) jsonb_each_text(jsonb)	setof key text, value text	将最外层的JSON对象扩展为一组键/值对。返回的值将是text类型。	select * from json_each_text('{"a":"foo", "b":"bar"}')	key value -----+----- a foo b bar
json_extract_path(from_json json, VARIADIC path_elems text[]) jsonb_extract_path(from_json jsonb, VARIADIC path_elems text[])	json jsonb	返回path_elems指向的JSON值（相当于#>运算符）。	json_extract_path('{"f2":{"f3":1},"f4":{"f5":99,"f6":"foo"}}','f4')	{"f5":99,"f6":"foo"}

函数	返回类型	描述	示例	示例结果
json_extract_path_text(from_json json, VARADIC path_elems text[]) jsonb_extract_path_text(from_json jsonb, VARADIC path_elems text[])	text	返回path_elems指向的JSON值作为文本。相当于#>>运算符。	json_extract_path_text('{"f2":{"f3":1},"f4":{"f5":99,"f6":"foo"}}','f4','f6')	foo
json_object_keys(json) jsonb_object_keys(jsonb)	setof text	返回最外层JSON对象中的键集。	json_object_keys('{"f1":"abc","f2":{"f3":"a","f4":"b"}}')	json_object_keys ----- f1 f2

函数	返回类型	描述	示例	示例结果
json_populate_record(base anyelement, from _json json) jsonb_populate_record(base anyelement, from _json jsonb)	anyelement	将从_jsonn中的对象扩展为其列与base定义的记录类型匹配的行。 见 注1 。	select * from json_populate_record(null::myrowtype, '{"a":1,"b":2}')	a b ---+--- 1 2
json_populate_recordset(base anyelement, from_json json) jsonb_populate_recordset(base anyelement, from_json jsonb)	setof anyelement	将从_jsonn中最外层对象的数组扩展为一组行，这些行的列与base定义的记录类型匹配。 见 注1 。	select * from json_populate_recordset(null::myrowtype, '[{"a":1,"b":2},{ "a":3,"b":4}]')	a b ---+--- 1 2 3 4

函数	返回类型	描述	示例	示例结果
json_array_elements(json) jsonb_array_elements(jsonb)	setof json setof jsonb	将JSON数组扩展为一组JSON值。	select * from json_array_elements('[1,true,[2,false]]')	value ----- 1 true [2,false]
json_array_elements_text(json) jsonb_array_elements_text(jsonb)	setof text	将JSON数组扩展为一组文本值。	select * from json_array_elements_text(['foo','bar'])	value ----- foo bar
json_typeof(json) jsonb_typeof(jsonb)	text	以文本字符串形式返回最外层JSON值的类型。可能的类型是object，array，string，number，boolean和null。 见注2	json_typeof('123.4')	number

函数	返回类型	描述	示例	示例结果
json_to_record(json) jsonb_to_record(jsonb)	record	从JSON对象构建任意记录。 见注1。 与返回记录的所有函数一样，调用者必须使用AS子句显式定义记录的结构。	select * from json_to_record('{\"a\":1,\"b\":[1,2,3],\"c\":\"bar\"}') as x(a int, b text, d text)	a b d ---+-----+--- 1 [1,2,3]
json_to_recordset(json) jsonb_to_recordset(jsonb)	setof record	从JSON对象数组构建任意记录集请参见注1。 与返回记录的所有函数一样，调用者必须使用AS子句显式定义记录的结构。	select * from json_to_recordset('[{\"a\":1,\"b\":\"foo\"},{\"a\":\"2\",\"c\":\"bar\"}]') as x(a int, b text);	a b ---+----- 1 foo 2

Note:

1. 函数json_populate_record(), json_populate_recordset(), json_to_record()和json_to_recordset()的示例使用常量。但是，典型的用法

是引用FROM子句中的表，并使用其中一个json或jsonb列作为函数的参数。然后可以在查询的其他部分中引用所提取的键值。 例如，可以在WHERE子句和目标列表中引用该值。 以这种方式提取多个值可以提高性能，而不是使用每个键操作符单独提取它们。

JSON键与目标行类型中的相同列名匹配。 这些函数的JSON类型强制转换可能不会产生某些类型的期望值。 将不会在输出中省略未出现在目标行类型中的JSON字段，并且与任何JSON字段不匹配的目标列将为NULL。

2. 不应将json_typeof函数null返回值null与SQL NULL混淆。 调用json_typeof('null'::json)将返回null， 调用json_typeof(NULL::json)将返回SQLNULL。

窗口函数

以下是MPP数据库内置窗口函数。 所有窗口函数都是不可变的。 有关窗口函数的更多信息，请参阅MPP数据库管理员手册中的“窗口表达式”。

Table 9. 窗口函数

函数	返回类型	完整语法	描述
cume_dist()	double precision	CUME_DIST() OVER ([PARTITION BY expr] ORDER BY expr)	计算一组值中的值的累积分布。 具有相等值的行始终评估为相同的累积分布值。
dense_rank()	bigint	DENSE_RANK () OVER ([PARTITION BY expr] ORDER BY expr)	计算有序行组中行的等级而不跳过等级值。 具有相等值的行被赋予相同的等级值。

函数	返回类型	完整语法	描述
first_value(<i>expr</i>)	与输入 <i>expr</i> 类型相同	FIRST_VALUE(<i>expr</i>) OVER ([PARTITION BY <i>expr</i>] ORDER BY <i>expr</i> [ROWS RANGE frame_expr])	返回有序值集中的第一个值。
lag(<i>expr</i> [, <i>offset</i>] [, <i>default</i>])	与输入 <i>expr</i> 类型相同	LAG(<i>expr</i> [, <i>offset</i>] [, <i>default</i>]) OVER ([PARTITION BY <i>expr</i>] ORDER BY <i>expr</i>)	提供对同一个表的多个行的访问，而无需进行自连接。给定从查询返回的一系列行和光标的位置，LAG提供对该位置之前给定物理偏移处的行的访问。默认offset为1。 <i>default</i> 设置当偏移量超出窗口范围时返回的值。如果未指定 <i>default</i> ，则默认值为null。
last_value(<i>expr</i>)	与输入 <i>expr</i> 类型相同	LAST_VALUE(<i>expr</i>) OVER ([PARTITION BY <i>expr</i>] ORDER BY <i>expr</i> [ROWS RANGE frame_expr])	返回有序值集中的最后一个值。

函数	返回类型	完整语法	描述
<code>lead(<i>expr</i> [,<i>offset</i>] [,<i>default</i>])</code>	与输入 <i>expr</i> 类型相同	<code>LEAD(<i>expr</i> [,<i>offset</i>] [,<i>exprdefault</i>]) OVER ([PARTITION BY <i>expr</i>] ORDER BY <i>expr</i>)</code>	提供对同一个表的多个行的访问，而无需进行自连接。给定从查询返回的一系列行和光标的位置， <code>lead</code> 提供对该位置之后给定物理偏移处的行的访问。如果未指定 <i>offset</i> ，则默认偏移量为1。 <i>default</i> 设置当偏移量超出窗口范围时返回的值。如果未指定 <i>default</i> ，则默认值为null。
<code>ntile(<i>expr</i>)</code>	bigint	<code>NTILE(<i>expr</i>) OVER ([PARTITION BY <i>expr</i>] ORDER BY <i>expr</i>)</code>	将有序数据集划分为多个存储桶（由 <i>expr</i> 定义），并为每行分配存储桶编号。
<code>percent_rank()</code>	double precision	<code>PERCENT_RANK () OVER ([PARTITION BY <i>expr</i>] ORDER BY <i>expr</i>)</code>	计算假定行R减1的等级，除以小于被评估的行数（在窗口分区内）的1。

函数	返回类型	完整语法	描述
rank()	bigint	RANK () OVER ([PARTITION BY <i>expr</i>] ORDER BY <i>expr</i>)	计算有序值组中行的等级。排名标准具有相同值的行具有相同的排名。绑定行的数量被添加到等级编号以计算下一个等级值。在这种情况下，排名可能不是连续的数字。
row_number()	bigint	ROW_NUMBER () OVER ([PARTITION BY <i>expr</i>] ORDER BY <i>expr</i>)	为应用它的每一行分配一个唯一的编号（窗口分区中的每一行或查询的每一行）。

高级聚合函数

以下内置高级分析函数是PostgreSQL数据库的MPP扩展。分析函数是 *不可变的*。

Note: MPP MADlib分析扩展提供了额外的高级功能，可以使用MPP数据库数据执行统计分析和机器学习。 请参阅[MPP MADlib分析扩展](#)。

Table 10. 高级聚合函数

函数	返回类型	完整语法	描述
MEDIAN(<i>expression</i>)	timestamp, interval, float	MEDIAN (<i>expression</i>) Example: <pre>SELECT department_id, MEDIAN(salary) FROM employees GROUP BY department_id;</pre>	可以将二维数组作为输入。将这些数组视为矩阵。
PERCENTILE	timestamp	PERCENTILE_CONT(<i>percentage</i>) WITHIN GROUP (ORDER BY <i>expression</i>) Example:	执行假设连

函数 LE E _ C O N T (e xp r) W I T H I N G R O U P (O R D E R B Y ex pr [D E S	返回类型 es ta m pt z, int er va l, f lo at	完整语法 SELECT department_id, PERCENTILE_CONT (0.5) WITHIN GROUP (ORDER BY salary DESC) "Median_cont"; FROM employees GROUP BY department_id;	统计分布模型的逆分布函数。它采用百分位值和排序规范，并返回与参数的n

函数 S C])	返回类型	完整语法	描述 er ic 数据类型相同的数据类型。该返回值是执行线性插值后的计算结果

函数	返回类型	完整语法	描述 此计算中忽略 Null
PERCENTILE_DISC	timestamp, timestamp, interval, float	PERCENTILE_DISC(<i>percentage</i>) WITHIN GROUP (ORDER BY <i>expression</i>) <i>Example:</i> <pre>SELECT department_id, PERCENTILE_DISC (0.5) WITHIN GROUP (ORDER BY salary DESC) "Median_desc"; FROM employees GROUP BY department_id;</pre>	执行假设离散分布模型的逆分布函数。它需要百分位

函数 (ORDER BY expr [DESC/ASC])	返回类型	完整语法	描述 排序规范。此返回值是集合中的元素。在此计算中忽略Null。

函数	返回类型	完整语法	描述
sum(array[])	smallint[], bigint[], float[]	sum(array[[1,2],[3,4]]) Example: <pre>CREATE TABLE mymatrix (myvalue int[]); INSERT INTO mymatrix VALUES (array[[1,2],[3,4]]); INSERT INTO mymatrix VALUES (array[[0,1],[1,0]]); SELECT sum(myvalue) FROM mymatrix; sum ----- {{1,3},{4,4}}</pre>	执行矩阵求和。可以将被视为矩阵的二维数组作为输入。

函数	返回类型	完整语法	描述
<code>pivot_sum(label[, label, expr])</code>	<code>int</code> <code>[]</code> , <code>bigint</code> <code>[]</code> , <code>float</code> <code>[]</code>	<code>pivot_sum(array['A1','A2'], attr, value)</code>	使用 <code>sum</code> 的数据透视聚合来解决重复的条目。
<code>unnest(array[])</code>	<code>set of any element</code>	<code>unnest(array['one', 'row', 'per', 'item'])</code>	将一维数组转换为行

函数	返回类型	完整语法	描述 回
			组 a ny el e m e nt , 多态的 P os tg re S Q L 中的假型

文本搜索函数和操作符

下表总结了为全文搜索提供的函数和运算符。 有关MPP数据库文本搜索工具的详细说明，请参阅[使用全文搜索](#)。

Table 11. 文本搜索运算符

操作符	描述	示例	结果
@@	tsvector匹配tsquery ?	to_tsvector('fat cats ate rats') @@ to_tsquery('cat & rat')	t
@@@	@@的弃用同义词	to_tsvector('fat cats ate rats') @@@ to_tsquery('cat & rat')	t
	连接 tsvectors	'a:1 b:2':tsvector 'c:1 d:2 b:3':tsvector	'a:1 'b':2,5 'c':3 'd':4
&&	AND tsqueries 一起	'fat rat':tsquery && 'cat':tsquery	('fat' 'rat') & 'cat'
	OR tsqueries 一起	'fat rat':tsquery 'cat':tsquery	('fat' 'rat') 'cat'
!!	否定一个 tsquery	!! 'cat':tsquery	!'cat'
@>	tsquery是否包含另一个？	'cat':tsquery @> 'cat & rat':tsquery	f
<@	tsquery是否包含在？	'cat':tsquery <@ 'cat & rat':tsquery	t

Note: tsquery包含运算符仅考虑两个查询中列出的词位，忽略组合运算符。

除了表中显示的运算符之外，还为类型tsvector和tsquery定义了普通的B树比较运算符（=, <, 等）。 这些对于文本搜索不是很有用，但允许在这些类型的列上构建唯一索引。

Table 12. 文本搜索函数

函数	返回类型	描述	示例	结果
get_current_ts_config()	regconfig	获取默认文本搜索配置	get_current_ts_config()	english
length(tsvector)	integer	tsvector中的词位数	length('fat:2,4 cat:3 rat:5A':::tsvector)	3
numnode(tsquery)	integer	词位数加tsquery中的运算符的数量	numnode('(fat & rat) cat':::tsquery)	5
plainto_tsquery([config regconfig ,] querytext)	tsquery	产生忽略标点符号的tsquery	plainto_tsquery('english', 'The Fat Rats')	'fat' & 'rat'
querytree(query tsquery)	text	获得tsquery的可索引部分	querytree('foo & ! bar':::tsquery)	'foo'

函数	返回类型	描述	示例	结果
setweight(tsvector, "char")	tsvector	为tsvector的每个元素赋予权重	setweight('fat:2,4 cat:3 rat:5B':::tsvector, 'A')	'cat':3 A 'fat':2A,4 A 'rat':5A
strip(tsvector)	tsvector	从tsvector中删除位置和权重	strip('fat:2,4 cat:3 rat:5A':::tsvector)	'cat' 'fat' 'rat'
to_tsquery([config regconfig ,] query text)	tsquery	规范化单词并转换为tsquery	to_tsquery('english', 'The & Fat & Rats')	'fat' & 'rat'
to_tsvector([config regconfig ,] documenttext)	tsvector	将文档文本缩减为tsvector	to_tsvector('english', 'The Fat Rats')	'fat':2 'rat':3
ts_headline([config regconfig,] documenttext , query tsquery [, options text])	text	显示查询匹配	ts_headline('x y z', 'z':::tsquery)	x y < b>z
ts_rank([weights float4 [],] vector tsvector, query tsquery [, normalization integer])	float4	用于查询的排名文档	ts_rank(textsearch, query)	0.818

函数	返回类型	描述	示例	结果
ts_rank_cd([weights float4[],] vectortsvector, query tsquery [, normalizationinteger])	float4	使用封面密度对查询进行排名	ts_rank_cd('{0.1, 0.2, 0.4, 1.0}', textsearch, query)	2.01317
ts_rewrite(query tsquery, target tsquery, substitute tsquery)	tsquery	在查询中用 substitute 替换目标	ts_rewrite('a & b'::tsquery, 'a'::tsquery, 'foo bar'::tsquery)	'b' & ('foo' 'bar')
ts_rewrite(query tsquery, select text)	tsquery	使用 targets 和 substitute s 从 SELECT 命令中替换	SELECT ts_rewrite('a & b'::tsquery, 'SELECT t,s FROM aliases')	'b' & ('foo' 'bar')
tsvector_update_trigger()	trigger	用于自动 tsvector 列更新的触发器函数	CREATE TRIGGER ... tsvector_update_trigger (tsvcol, 'pg_catalog.swedish', title, body)	
tsvector_update_trigger_column()	trigger	用于自动 tsvector 列更新的触发器函数	CREATE TRIGGER ... tsvector_update_trigger _column(tsvcol, configcol, title, body)	

Note: 接受可选的 regconfig 参数的所有文本搜索函数将在省略该参数时使用 [default_text_search_config](#) 指定的配置。

下表中的功能单独列出，因为它们通常不用于日常文本搜索操作。它们有助于开发和调试新的文本搜索配置。

Table 13. 文本搜索调试函数

函数	返回类型	描述	示例	结果
ts_debug([config regconfig,] document text, OUT alias text, OUT description text, OUT token text, OUT dictionaries regdictionary[], OUT dictionary regdictionary, OUT lexemes text[])	set of record	测试配置	ts_debug('english', 'The Brightest supernovaes')	(asciiword," Word, all ASCII",The,{english_stem}, english_stem,{}) ...
ts_lexize(dict regdictionary, token text)	text[]	测试字典	ts_lexize('english_stem', 'stars')	{star}
ts_parse(parser_name text, document text, OUT tokid integer, OUT token text)	set of record	测试解析器	ts_parse('default', 'foo - bar')	(1,foo) ...
ts_parse(parser_oid oid, document text, OUT tokid integer, OUT token text)	set of record	测试解析器	ts_parse(3722, 'foo - bar')	(1,foo) ...

函数	返回类型	描述	示例	结果
ts_token_type(<i>parser_name</i> text, OUT <i>tokid</i> integer, OUT <i>alias</i> text, OUT <i>description</i> text)	set of record	获取解析器定义的 token 类型	ts_token_type('default')	(1,asciiword,"Word, all ASCII") ...
ts_token_type(<i>parser_oid</i> oid, OUT <i>tokid</i> integer, OUT <i>alias</i> text, OUT <i>description</i> text)	set of record	获取解析器定义的 token 类型	ts_token_type(3722)	(1,asciiword,"Word, all ASCII") ...
ts_stat(<i>sqlquery</i> text, [<i>weights</i> text,] OUT <i>word</i> text, OUT <i>ndocinteger</i> , OUT <i>nentry</i> integer)	set of record	获取 tsvectorcolumn 的统计信息	ts_stat('SELECT vector from apod')	(foo,10,15) ..

范围函数和运算符

有关范围类型的概述，请参阅[范围类型](#)。

下表显示了可用于范围类型的运算符。

Table 14. 范围运算符

运算符	描述	示例	结果
=	相等	<code>int4range(1,5) = '[1,4]':int4range</code>	t
<>	不相等	<code>numrange(1.1,2.2) <> numrange(1.1,2.3)</code>	t
<	小于	<code>int4range(1,10) < int4range(2,3)</code>	t
>	大于	<code>int4range(1,10) > int4range(1,5)</code>	t
<=	小于等于	<code>numrange(1.1,2.2) <= numrange(1.1,2.2)</code>	t
>=	大于等于	<code>numrange(1.1,2.2) >= numrange(1.1,2.0)</code>	t
@>	包含范围	<code>int4range(2,4) @> int4range(2,3)</code>	t
@>	包含元素	<code>'[2011-01-01,2011-03-01]'::tsrange @> '2011-01-10'::timestamp</code>	t
<@	范围包含在	<code>int4range(2,4) <@ int4range(1,7)</code>	t
<@	元素包含在	<code>42 <@ int4range(1,7)</code>	f
&&	重叠（有共同点）	<code>int8range(3,7) && int8range(4,12)</code>	t
<<	严格小于	<code>int8range(1,10) << int8range(100,110)</code>	t
>>	严格大于	<code>int8range(50,60) >> int8range(20,30)</code>	t
&<	没有超越右边	<code>int8range(1,20) &< int8range(18,20)</code>	t
&>	没有超越左边	<code>int8range(7,20) &> int8range(5,10)</code>	t
- -	毗邻	<code>numrange(1.1,2.2) - - numrange(2.2,3.3)</code>	t

运算符	描述	示例	结果
+	并集	numrange(5,15) + numrange(10,20)	[5,20)
*	交集	int8range(5,15) * int8range(10,20)	[10,15)
-	差集	int8range(5,15) - int8range(10,20)	[5,10)

简单比较运算符<，>，<=和>=首先比较下限，只有在相等的情况下，才比较上限。 这些比较通常对范围不是很有用，但提供的是允许在范围上构建B树索引。

当涉及空范围时，左/右/邻接运算符总是返回false; 也就是说，空范围不被认为是在任何其他范围之前或之后。

如果结果范围需要包含两个不相交的子范围，则并集和差异运算符将失败，因为无法表示这样的范围。

下表显示了可用于范围类型的函数。

Table 15. 范围函数

函数	返回类型	描述	示例	结果
lower(anyrange)	范围的元素类型	范围的下限	lower(numrange(1.1,2.2))	1.1
upper(anyrange)	范围的元素类型	范围的上限	upper(numrange(1.1,2.2))	2.2
isempty(anyrange)	boolean	范围是否空？	isempty(numrange(1.1,2.2))	false

函数	返回类型	描述	示例	结果
lower_inc(anyrange)	boolean	是否包含下限？	lower_inc(numrange(1.1,2.2))	true
upper_inc(anyrange)	boolean	是否包含上限？	upper_inc(numrange(1.1,2.2))	false
lower_inf(anyrange)	boolean	下限是否无穷小？	lower_inf('(,)'::daterange)	true
upper_inf(anyrange)	boolean	上限是否无穷大？	upper_inf('(,)'::daterange)	true
range_merge(anyrange, anyrange)	anyrange	包括两个给定范围的最小范围	range_merge('[1,2]'::int4range, '[3,4]'::int4range)	[1, 4)

如果范围为空或请求的边界为无限，则lower和upper函数返回null。对于空范围，lower_inc，upper_inc，lower_inf和upper_inf函数都返回false。

MPP MapReduce规范

该规范描述了用于定义MPP MapReduce作业的文档格式和模式。

[MapReduce](#)是由Google开发的一种编程模型，用于在商用服务器阵列上处理和生成大型数据集。MPP MapReduce允许熟悉MapReduce模型的程序员编写map和reduce函数，并将它们提交给MPP数据库并行引擎进行处理。

要使MPP能够处理MapReduce函数，请在文档中定义函数，然后将文档传递给MPP MapReduce程序gmpareduce，以便由MPP数据库并行引擎执行。MPP数据库系统分配输入数据，在一组机器上执行程序，处理机器故障，并管理所需的机器间通信。

关于gmpareduce的信息请见*MPP数据库工具手册*。

Parent topic: [MPP数据库参考手册](#)

MPP MapReduce文档格式

本节介绍MPP MapReduce文档格式的一些基础知识，以帮助您开始创建自己的MPP MapReduce文档。MPP使用[YAML 1.1](#)文档格式，然后实现自己的模式，以定义MapReduce作业的各个步骤。

所有MPP MapReduce文件必须首先声明它们正在使用的YAML规范的版本。之后，三个破折号（---）表示文档的开头，三个点（...）表示文档的结尾而不启动新文档。注释行以井号（#）为前缀。可以在同一个文件中声明多个MPP MapReduce文档：

```
%YAML 1.1
---
# Begin Document 1
# ...
---
# Begin Document 2
# ...
```

在MPP MapReduce文档中，有三种基本类型的数据结构或节点：标量，序列和映射。

标量是由空格缩进的基本文本字符串。如果您有跨越多行的标量输入，则前面

的管道 (|) 表示文字样式，其中所有换行符都很重要。或者，前一个尖括号 (>) 将单个换行符折叠到具有相同缩进级别的后续行的空格。如果字符串包含具有保留含义的字符，则必须引用该字符串，或者必须使用反斜杠 (\) 转义特殊字符。

```
# Read each new line literally
somekey: / this value contains two lines
    and each line is read literally
# Treat each new line as a space
anotherkey: >
    this value contains two lines
    but is treated as one continuous line
# This quoted string contains a special character
ThirdKey: "This is a string: not a mapping"
```

序列是列表，列表中的每个条目都在其自己的行上，用短划线和空格 (-) 表示。或者，您可以将内联序列指定为方括号内的逗号分隔列表。序列提供一组数据并为其提供订单。将列表加载到MPP MapReduce程序时，将保留订单。

```
# list sequence
- this
- is
- a list
- with
- five scalar values
# inline sequence
[this, is, a list, with, five scalar values]
```

映射用于将数据值与称为键的标识符配对。映射对每个key: value对使用冒号和空格 (:), 或者也可以内联指定为花括号内的逗号分隔列表。该密钥用作从映射中检索数据的索引。

```
# a mapping of items
title: War and Peace
author: Leo Tolstoy
date: 1865
# same mapping written inline
{title: War and Peace, author: Leo Tolstoy, date: 1865}
```

密钥用于将元信息与每个节点相关联，并指定预期的节点类型（标量，序列或映射）。有关MPP MapReduce程序所需的键，请参阅[MPP MapReduce文档模式](#)。

MPP MapReduce程序按顺序处理文档的节点，并使用缩进（空格）来确定文档层次结构和节点之间的关系。使用空白区域非常重要。不应仅将白色空间用于格式化目的，并且根本不应使用制表符。

MPP MapReduce文档模式

MPP MapReduce使用YAML文档框架并实现自己的YAML模式。MPP MapReduce文档的基本结构是：

```
%YAML 1.1
---
VERSION: 1.0.0.2
DATABASE: dbname
USER: db_username
HOST: master_hostname
PORT: master_port
```

DEFINE:**- INPUT:****NAME:** *input_name***FILE:****-** *hostname:/path/to/file***GPFDIST:****-** *hostname:port/file_pattern***TABLE:** *table_name***QUERY:** *SELECT_statement***EXEC:** *command_string***COLUMNS:****-** *field_name data_type***FORMAT:** TEXT | CSV**DELIMITER:** *delimiter_character***ESCAPE:** *escape_character***NULL:** *null_string***QUOTE:** *csv_quote_character***ERROR_LIMIT:** *integer***ENCODING:** *database_encoding***- OUTPUT:****NAME:** *output_name***FILE:** *file_path_on_client***TABLE:** *table_name***KEYS:****-** *column_name***MODE:** REPLACE | APPEND**- MAP:****NAME:** *function_name***FUNCTION:** *function_definition***LANGUAGE:** perl | python | c**LIBRARY:** */path/filename.so***PARAMETERS:****-** *nametype***RETURNS:****-** *nametype***OPTIMIZE:** STRICT IMMUTABLE**MODE:** SINGLE | MULTI

- TRANSITION | CONSOLIDATE | FINALIZE:

NAME: *function_name*

FUNCTION: *function_definition*

LANGUAGE: *perl | python | c*

LIBRARY: */path/filename.so*

PARAMETERS:

- *nametype*

RETURNS:

- *nametype*

OPTIMIZE: STRICT IMMUTABLE

MODE: SINGLE | MULTI

- REDUCE:

NAME: *reduce_job_name*

TRANSITION: *transition_function_name*

CONSOLIDATE: *consolidate_function_name*

FINALIZE: *finalize_function_name*

INITIALIZE: *value*

KEYS:

- *key_name*

- TASK:

NAME: *task_name*

SOURCE: *input_name*

MAP: *map_function_name*

REDUCE: *reduce_function_name*

EXECUTE

- RUN:

SOURCE: *input_or_task_name*

TARGET: *output_name*

MAP: *map_function_name*

REDUCE: *reduce_function_name...*

VERSION

必须。MPP MapReduce YAML规范版本。当前版本为1.0.0.1。

DATABASE

可选。指定MPP中要连接的数据库。如果未指定，则默认为默认数据库或\$PGDATABASE（如果已设置）。

USER

可选。指定要用于连接的数据库角色。如果未指定，则默认为当前用户或\$PGUSER（如果已设置）。您必须是MPP超级用户才能运行用不受信任的Python和Perl编写的函数。常规数据库用户可以运行用可信Perl编写的函数。您还必须是数据库超级用户才能运行包含[FILE](#)，[GPFDIST](#)和[EXEC](#)输入类型的MapReduce作业。

HOST

可选。指定MPP master主机名。如果未指定，则默认为localhost或\$PGHOST（如果已设置）。

PORT

可选。指定MPP主端口。如果未指定，则默认为5432或\$PGPORT（如果已设置）。

DEFINE

必须。此MapReduce文档的一系列定义。DEFINE部分必须至少有一个INPUT定义。

INPUT

必须。定义输入数据。每个MapReduce文档必须至少定义一个输入。文档中允许多个输入定义，但每个输入定义只能指定其中一种访问类型：文件，gpfdist文件分发程序，数据库中的表，SQL命令或操作系统命令。有关gpfdist的信息，请参阅*MPP数据库实用程序手册*。

NAME

此输入的名称。关于此MapReduce作业中其他对象的名称（例如map函数，task，reduce函数和输出名称），名称必须是唯一的。此外，名称不能与数据库中的现有对象（例如表，函数或视图）冲突。

FILE

一个或多个输入文件的序列，格式为：`seghostname:/path/to/filename`。您必须是MPP数据库超级用户才能使用FILE输入运行MapReduce作业。该文件必须位于MPP segment主机上。

GPFDIST

一个或多个运行gpfdist文件分发程序的序列，格式为：`hostname[:port]/file_pattern`。除非服务器配置参数[服务器配置参数](#)设置为on，否则您必须是MPP数据库超级用户才能使用GPFDIST输入运行MapReduce作业。

TABLE

数据库中现有表的名称。

QUERY

要在数据库中运行的SQL SELECT命令。

EXEC

要在MPP segment主机上运行的操作系统命令。默认情况下，该命令由系统中的所有segment实例运行。例如，如果每个segment主机有四个segment实例，则该命令将在每个主机上运行四次。您必须是MPP数据库超级用户才能使用EXEC输入运行MapReduce作业，并且服务器配置参数[服务器配置参数](#)设置为on。

COLUMNS

可选。列指定为：`column_name [data_type]`。如果未指定，则默认值为value text。[DELIMITER](#)字符用于分隔两个数据值字段（列）。行由换行符（0x0a）确定。

FORMAT

可选。指定数据的格式 - 分隔文本（TEXT）或逗号分隔值（CSV）格式。如果未指定数据格式，则默认为TEXT。

DELIMITER

可选[FILE](#)，[GPFDIST](#)和[EXEC](#)输入。指定用于分隔数据值的单个字符。默认值为TEXT模式下的制表符，CSV模式下为逗

号。分隔符字符只能出现在任意两个数据值字段之间。不要在行的开头或结尾放置分隔符。

ESCAPE

对于FILE，GPFDIST和EXEC输入可选。指定用于C转义序列的单个字符（例如\n,\t,\100等）以及转义可能以行或列分隔符形式取出的数据字符。确保选择实际列数据中未使用的转义字符。默认转义字符是文本格式文件的\（反斜杠）和csv格式文件的"（双引号），但是可以指定另一个字符来表示转义。也可以通过指定禁用转义值'OFF'作为转义值。这对于诸如文本格式的Web日志数据之类的数据非常有用，这些数据具有许多不打算转义的嵌入式反斜杠。

NULL

对于FILE，GPFDIST和EXEC输入可选。指定表示空值的字符串。默认值为TEXT格式的\N，以及CSV格式没有引号的空值。如果您不想将空值与空字符串区分开来，即使在TEXT模式下，您可能更喜欢空字符串。与此字符串匹配的任何输入数据项都将被视为空值。

QUOTE

对于FILE，GPFDIST和EXEC输入可选。指定CSV格式文件的引用字符。默认值为双引号（"）。在CSV格式的文件中，如果数据值字段包含任何逗号或嵌入的新行，则必须用双引号括起来。包含双引号字符的字段必须用双引号括起来，并且嵌入双引号必须由一对连续的双引号表示。始终正确打开和关闭引号以便正确解析数据行非常重要。

ERROR_LIMIT

如果输入行具有格式错误，则只要在输入处理期间未在任何MPP segment实例上达到错误限制计数，它们将被丢弃。如果未达到错误限制，则将处理所有正常行并丢弃任何错误行。

ENCODING

用于数据的字符集编码。指定字符串常量（例如'SQL_ASCII'），整数编码号或DEFAULT以使用默认客户端

编码。有关更多信息，请参阅[字符集支持](#)。

OUTPUT

可选。定义输出此MapReduce作业的格式化数据的位置。如果未定义输出，则默认为STDOUT（客户端的标准输出）。您可以将输出发送到客户端主机上的文件或数据库中的现有表。

NAME

此输出的名称。默认输出名称为STDOUT。关于MapReduce作业中其他对象的名称（例如map函数，task，reduce函数和输入名称），名称必须是唯一的。此外，名称不能与数据库中的现有对象（例如表，函数或视图）冲突。

FILE

指定MapReduce客户端计算机上的文件位置，以如下格式输出数据：/path/to/filename。

TABLE

指定数据库中用于输出数据的表的名称。如果在运行MapReduce作业之前该表不存在，则将使用[KEYS](#)指定的分发策略创建该表。

KEYS

[TABLE](#)输出的可选项。指定要用作MPP数据库分发键的列。如果[EXECUTE](#)任务包含[REDUCE](#)定义，则默认情况下REDUCE键将用作表分发键。否则，表的第一列将用作分发键。

MODE

[TABLE](#)输出的可选项。如果未指定，则默认为创建表（如果该表尚不存在），但如果表存在则输出错误。声明APPEND将输出数据添加到现有表（前提是表模式与输出格式匹配），而不删除任何现有数据。如果表存在，则声明REPLACE将删除该表，然后重新创建它。如果不存在，APPEND和REPLACE都将创建一个新表。

MAP

必须。每个MAP函数采用以（key, value）对构造的数据，处理每

对，并生成零个或多个输出 (key, value) 对。然后，MPP MapReduce框架从所有输出列表中收集具有相同密钥的所有对，并将它们组合在一起。然后将此输出传递给REDUCE任务，该任务由TRANSITION | CONSOLIDATE | FINALIZE函数组成。有一个名为IDENTITY的预定义MAP函数，它返回的 (key, value) 对不变。虽然 (key, value) 是默认参数，但您可以根据需要指定其他原型。

TRANSITION | CONSOLIDATE | FINALIZE

TRANSITION，CONSOLIDATE和FINALIZE都是REDUCE的组成部分。需要TRANSITION函数。CONSOLIDATE和FINALIZE函数是可选的。默认情况下，所有将state作为其输入PARAMETERS的第一个，但也可以定义其他原型。

TRANSITION函数遍历给定键的每个值，并在state变量中累积值。当在键的第一个值上调用转换函数时，state将设置为REDUCE作业的INITIALIZE指定的值（或数据类型的默认状态值）。转换需要两个参数作为输入；密钥减少的当前状态和下一个值，然后产生一个新state。

如果指定了CONSOLIDATE函数，则在segment级别执行TRANSITION处理，然后在MPP互连上重新分配密钥以进行最终聚合（两阶段聚合）。仅重新分配给定密钥的结果state值，从而导致更低的互连流量和更高的并行度。

CONSOLIDATE像TRANSITION一样处理，除了 (state + value) => state，它是 (state + state) => state。

如果指定了FINALIZE函数，它将采用CONSOLIDATE（如果存在）或TRANSITION生成的最终state，并在发出最终结果之前进行任何最终处理。TRANSITION和CONSOLIDATE函数不能返回一组值。如果需要REDUCE作业来返回一个集合，则需要FINALIZE将最终状态转换为一组输出值。

NAME

必须。函数的名称。关于此MapReduce作业中其他对象的名称（例如函数，任务，输入和输出名称），名称必须是唯一的。您还可以指定MPP数据库内置函数的名称。如果使用内置函

数，请不要提供LANGUAGE或FUNCTION正文。

FUNCTION

可选。使用指定的LANGUAGE指定函数的完整主体。如果未指定FUNCTION，则使用与NAME对应的内置数据库函数。

LANGUAGE

使用FUNCTION时需要。指定用于解释函数的实现语言。此版本具有对perl，python和C的语言支持。如果调用内置数据库函数，则不应指定LANGUAGE。

LIBRARY

LANGUAGE为C时必需（不允许用于其他语言函数）。要使用此属性，VERSION必须为1.0.0.2。必须在运行MapReduce作业之前安装指定的库文件，并且该文件必须存在于所有MPP主机（master和segment）上的相同文件系统位置。

PARAMETERS

可选。函数输入参数。默认类型是text。

MAP default - key text, value text

TRANSITION default - state text, value text

CONSOLIDATE default - state1 text, state2 text (必须具有相同数据类型的两个输入参数)

FINALIZE default - state text (仅限单个参数)

RETURNS

可选。默认返回类型是text。

MAP default - key text, value text

TRANSITION default - state text (仅限单个参数)

CONSOLIDATE default - state text (仅限单个参数)

FINALIZE default - value text

OPTIMIZE

该函数的可选优化参数：

STRICT - 函数不受NULL值的影响

IMMUTABLE - 函数将始终返回给定输入的相同值

MODE

可选。指定函数返回的行数。

MULTI - 每个输入记录返回0行或更多行。函数的返回值必须是要返回的行数组，或者必须使用Python中的yield或Perl中的return_next将函数写为迭代器。

MULTI是MAP和FINALIZE函数的默认模式。

SINGLE - 每个输入记录只返回一行。

SINGLE是TRANSITION和CONSOLIDATE函数支持的唯一模式。当与MAP和FINALIZE函数一起使用时，SINGLE模式可以提供适度的性能改进。

REDUCE

必须。REDUCE定义命名[TRANSITION | CONSOLIDATE | FINALIZE](#)函数，包括将 (key, value) 对缩减到最终结果集。您还可以执行几个预定义的REDUCE作业，这些作业都在名为value的列上运行：

IDENTITY - 返回 (键, 值) 对不变

SUM - 计算数值数据的总和

AVG - 计算数字数据的平均值

COUNT - 计算输入数据的计数

MIN - 计算数值数据的最小值

MAX - 计算数值数据的最大值

NAME

必须。这个REDUCE工作的名称。关于此MapReduce作业中的其他对象的名称（函数，任务，输入和输出名称），名称必须是唯一的。此外，名称不能与数据库中的现有对象（例如表，函数或视图）冲突。

TRANSITION

必须。TRANSITION函数名称。

CONSOLIDATE

可选。CONSOLIDATE函数名称。

FINALIZE

可选。FINALIZE函数名称。

INITIALIZE

text和float数据类型的可选项。所有其他数据类型都需要。文本的默认值为"。float的默认值为0.0。设置TRANSITION函数的初始state值。

KEYS

可选。默认为[key, *]。使用多列缩减时，可能需要指定哪些列是键列，哪些列是值列。默认情况下，未传递给TRANSITION函数的任何输入列都是键列，名为key的列始终是键列，即使它传递给TRANSITION函数也是如此。特殊指示符*表示未传递给TRANSITION函数的所有列。如果该指示符不存在于键列表中，则丢弃任何不匹配的列。

TASK

可选。TASK在MPP MapReduce作业管道中定义了完整的端到端INPUT/MAP/REDUCE阶段。它与EXECUTE类似，但不会立即执行。可以被称为INPUT的任务对象进入进一步处理阶段。

NAME

必须。此任务的名称。关于此MapReduce作业中其他对象的名称（例如map函数，reduce函数，输入和输出名称），名称必须是唯一的。此外，名称不能与数据库中的现有对象（例如

表，函数或视图) 冲突。

SOURCE

[INPUT](#)或其他TASK的名称。

MAP

可选。 [MAP](#)函数的名称。 如果未指定，则默认为IDENTITY。

REDUCE

可选。 [REDUCE](#)函数的名称。 如果未指定，则默认为IDENTITY。

EXECUTE

必须。EXECUTE定义MPP MapReduce作业管道中的最终INPUT/MAP/REDUCE阶段。

RUN

SOURCE

必须。 [INPUT](#)或[TASK](#)的名称。

TARGET

可选。 [OUTPUT](#)的名称。默认值为STDOUT。

MAP

可选。 [MAP](#)函数名称。 如果未指定，默认为IDENTITY。

REDUCE

可选。 [REDUCE](#)函数的名称。默认为IDENTITY。

示例MPP MapReduce文档

```
# This example MapReduce job processes documents and looks for keywords in them.  
# It takes two database tables as input:  
# - documents (doc_id integer, url text, data text)
```



```
# data: "a computer is a machine that manipulates data"
# "a" [1, 4]
# "computer" [2]
# "machine" [3]
# ...
    if term in terms:
        terms[term] += ','+str(i)
    else:
        terms[term] = str(i)

# Return multiple lines for each document. Each line consists of
# the doc_id, a term and the positions in the data where the term appeared.
# For example:
# (doc_id => 100, term => "a", [1,4]
# (doc_id => 100, term => "computer", [2]
# ...
for term in terms:
yield([doc_id, term, terms[term]])
    OPTIMIZE:STRICT IMMUTABLE
    PARAMETERS:
- doc_id integer
  - data text
RETURNS:
- doc_id integer
  - term text
  - positions text

# The map function for keywords is almost identical to the one for documents
# but it also counts of the number of terms in the keyword.
- MAP:
NAME:kw_map
LANGUAGE:python
FUNCTION:|
    i = 0
    terms = {}
    for term in keyword.lower().split():
        i = i + 1
        if term in terms:
            terms[term] += ','+str(i)
        else:
            terms[term] = str(i)

# output 4 values including i (the total count for term in terms):
```

```
yield([keyword_id, i, term, terms[term]])
```

```
    OPTIMIZE:STRICT IMMUTABLE
```

```
PARAMETERS:
```

```
- keyword_id integer
```

```
    - keyword text
```

```
RETURNS:
```

```
- keyword_id integer
```

```
    - nterms integer
```

```
    - term text
```

```
    - positions text
```

```
# A TASK is an object that defines an entire INPUT/MAP/REDUCE stage
```

```
# within a MPP MapReduce pipeline. It is like EXECUTION, but it is
```

```
# executed only when called as input to other processing stages.
```

```
# Identify a task called 'doc_prep' which takes in the 'doc' INPUT defined earlier
```

```
# and runs the 'doc_map' MAP function which returns doc_id, term, [term_position]
```

```
- TASK:
```

```
NAME:doc_prep
```

```
SOURCE:doc
```

```
MAP:doc_map
```

```
# Identify a task called 'kw_prep' which takes in the 'kw' INPUT defined earlier
```

```
# and runs the kw_map MAP function which returns kw_id, term, [term_position]
```

```
- TASK:
```

```
NAME:kw_prep
```

```
SOURCE:kw
```

```
MAP:kw_map
```

```
# One advantage of MPP MapReduce is that MapReduce tasks can be
```

```
# used as input to SQL operations and SQL can be used to process a MapReduce task.
```

```
# This INPUT defines a SQL query that joins the output of the 'doc_prep'
```

```
# TASK to that of the 'kw_prep' TASK. Matching terms are output to the 'candidate'
```

```
# list (any keyword that shares at least one term with the document).
```

```
- INPUT:
```

```
NAME: term_join
```

```
QUERY: |
```

```
    SELECT doc.doc_id, kw.keyword_id, kw.term, kw.nterms,
```

```
           doc.positions as doc_positions,
```

```
           kw.positions as kw_positions
```

```
    FROM doc_prep doc INNER JOIN kw_prep kw ON (doc.term = kw.term)
```

```
# In MPP MapReduce, a REDUCE function is comprised of one or more functions.
```

```
# A REDUCE has an initial 'state' variable defined for each grouping key. that is
```

```

# A TRANSITION function adjusts the state for every value in a key grouping.
# If present, an optional CONSOLIDATE function combines multiple
# 'state' variables. This allows the TRANSITION function to be executed locally at
# the segment-level and only redistribute the accumulated 'state' over
# the network. If present, an optional FINALIZE function can be used to perform
# final computation on a state and emit one or more rows of output from the state.
#
# This REDUCE function is called 'term_reducer' with a TRANSITION function
# called 'term_transition' and a FINALIZE function called 'term_finalizer'
- REDUCE:
NAME:term_reducer
TRANSITION:term_transition
FINALIZE:term_finalizer

- TRANSITION:
NAME:term_transition
LANGUAGE:python
PARAMETERS:
- state text
  - term text
  - nterms integer
  - doc_positions text
  - kw_positions text
FUNCTION: |

# 'state' has an initial value of "" and is a colon delimited set
# of keyword positions. keyword positions are comma delimited sets of
# integers. For example, '1,3,2:4:'
# If there is an existing state, split it into the set of keyword positions
# otherwise construct a set of 'nterms' keyword positions - all empty
if state:
    kw_split = state.split(':')
else:
    kw_split = []
    for i in range(0,nterms):
        kw_split.append("")

    # 'kw_positions' is a comma delimited field of integers indicating what
    # position a single term occurs within a given keyword.
    # Splitting based on ',' converts the string into a python list.
    # add doc_positions for the current term
for kw_p in kw_positions.split(','):
    kw_split[int(kw_p)-1] = doc_positions

```

```

# This section takes each element in the 'kw_split' array and strings
# them together placing a ':' in between each element from the array.
# For example: for the keyword "computer software computer hardware",
# the 'kw_split' array matched up to the document data of
# "in the business of computer software software engineers"
# would look like: ['5', '6,7', '5', '']
# and the outstate would look like: 5:6,7:5:
outstate = kw_split[0]
for s in kw_split[1:]:
    outstate = outstate + ':' + s
return outstate

```

- FINALIZE:

NAME: term_finalizer

LANGUAGE: python

RETURNS:

- count integer

MODE: MULTI

FUNCTION: |

if not state:

return 0

kw_split = state.split(':')

This function does the following:

1) Splits 'kw_split' on ':'

for example, 1,5,7:2,8 creates '1,5,7' and '2,8'

2) For each group of positions in 'kw_split', splits the set on ','

to create ['1','5','7'] from Set 0: 1,5,7 and

eventually ['2', '8'] from Set 1: 2,8

3) Checks for empty strings

4) Adjusts the split sets by subtracting the position of the set

in the 'kw_split' array

['1','5','7'] - 0 from each element = ['1','5','7']

['2', '8'] - 1 from each element = ['1', '7']

5) Resulting arrays after subtracting the offset in step 4 are

intersected and their overlapping values kept:

['1','5','7'].intersect['1', '7'] = [1,7]

6) Determines the length of the intersection, which is the number of

times that an entire keyword (with all its pieces) matches in the

document data.

previous = None

for i in range(0, len(kw_split)):

```

    for i in range(0, len(kw_split)):
        isplit = kw_split[i].split(',')
        if any(map(lambda(x): x == "", isplit)):
            return 0
        adjusted = set(map(lambda(x): int(x)-i, isplit))
        if (previous):
            previous = adjusted.intersection(previous)
        else:
            previous = adjusted

    # return the final count
    if previous:
        return len(previous)

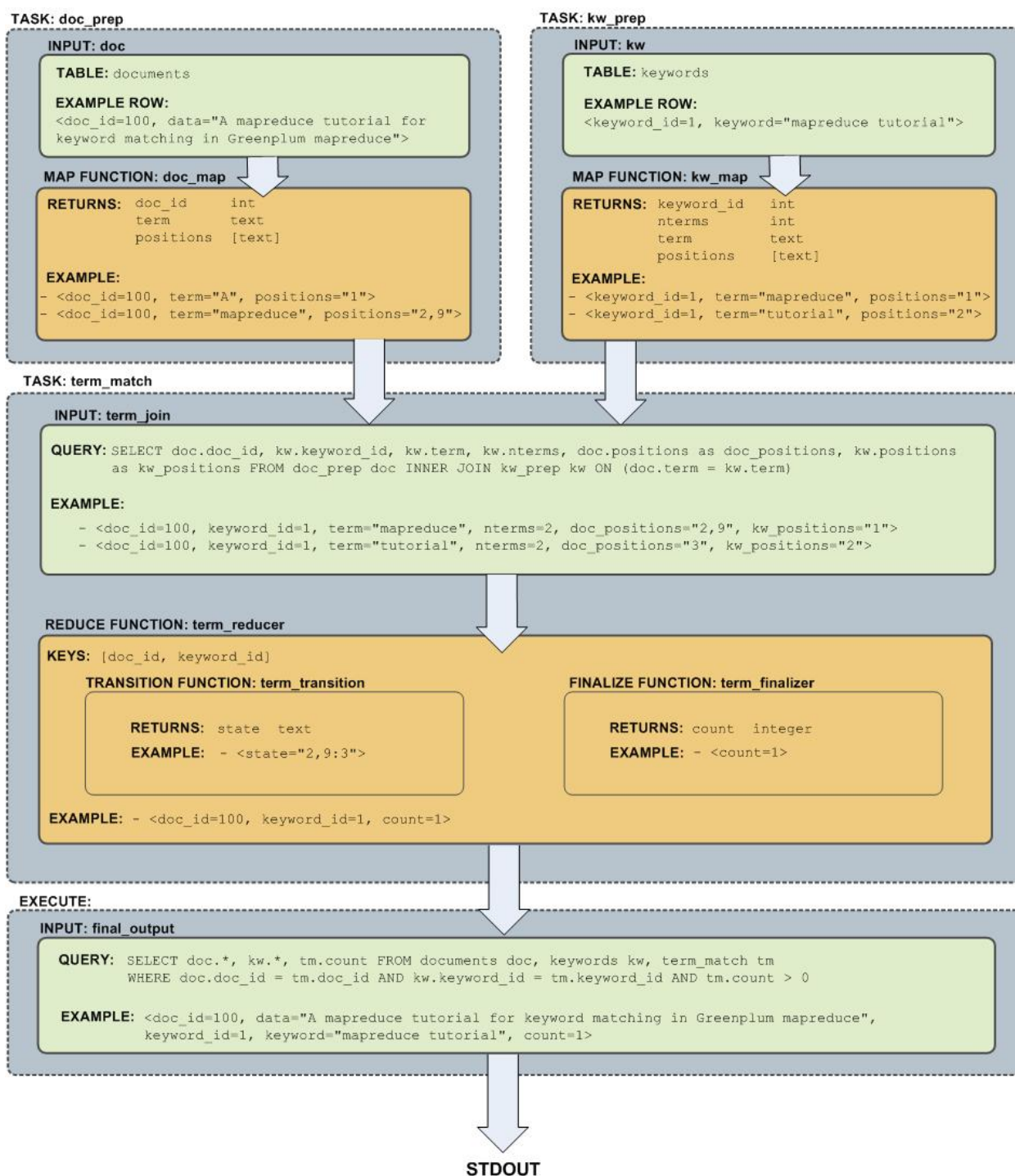
    # Define the 'term_match' task which is then executed as part
    # of the 'final_output' query. It takes the INPUT 'term_join' defined
    # earlier and uses the REDUCE function 'term_reducer' defined earlier
    - TASK:
NAME:term_match
SOURCE:term_join
REDUCE:term_reducer
- INPUT:
NAME:final_output
QUERY:|
    SELECT doc.*, kw.*, tm.count
    FROM documents doc, keywords kw, term_match tm
    WHERE doc.doc_id = tm.doc_id
        AND kw.keyword_id = tm.keyword_id
        AND tm.count > 0

    # Execute this MapReduce job and send output to STDOUT
EXECUTE:
    - RUN:
SOURCE:final_output
TARGET:STDOUT

```

MapReduce示例的流程图

下图显示了示例中定义的MapReduce作业的作业流程：



MPP PL/pgSQL过程语言

本节概述了MPP数据库PL/pgSQL语言。

- [关于MPP数据库PL/pgSQL](#)
- [PL/pgSQL计划缓存](#)
- [PL/pgSQL示例](#)
- [参考](#)

Parent topic: [MPP数据库参考手册](#)

关于MPP数据库PL/pgSQL

MPP数据库PL/pgSQL是一种可加载的过程语言，默认情况下使用MPP数据库进行安装和注册。 您可以使用SQL语句，函数和运算符创建用户定义的函数。

使用PL/pgSQL，您可以在数据库服务器内对计算块和一系列SQL查询进行分组，从而具有过程语言的强大功能和SQL的易用性。 此外，使用PL/pgSQL，您可以使用MPP数据库SQL的所有数据类型，运算符和函数。

PL/pgSQL语言是Oracle PL/SQL的子集。 MPP数据库PL/pgSQL基于Postgres PL/pgSQL。 Postgres PL/pgSQL文档位于

<https://www.postgresql.org/docs/9.4/plpgsql.html>

使用PL/pgSQL函数时，函数属性会影响MPP数据库创建查询计划的方式。 您可以将属性IMMUTABLE，STABLE或VOLATILE指定为LANGUAGE子句的一部分，以对函数类型进行分类。 有关创建函数和函数属性的信息，请参阅*MPP数据库参考手册*中的[CREATE FUNCTION](#)命令。

您可以将PL/SQL代码块作为匿名代码块运行。 请参阅*MPP数据库参考手册*中的[DO](#)命令。

MPP数据库SQL限制

使用MPP数据库PL/pgSQL时，限制包括

- 不支持触发器
- 游标只向前移动（不可滚动）
- 不支持可更新游标（UPDATE...WHERE CURRENT OF和DELETE...WHERE CURRENT OF）。

有关MPP数据库SQL一致性的信息，请参阅*MPP数据库参考手册*中的[MPP 特点概要](#)。

PL/pgSQL语言

PL/pgSQL是一种块结构语言。函数定义的完整文本必须是块。块定义为：

```
[ label ]  
[ DECLARE  
  declarations ]  
BEGIN  
  statements  
END [ label ];
```

块中的每个声明和每个语句都以分号 (;) 结束。出现在另一个块中的块在END之后必须有分号，如上一个块所示。结束函数体的END不需要分号。

仅当您要识别要在EXIT语句中使用的块或限定块中声明的变量的名称时，才需要label。如果在END之后提供label，则它必须与块开头的label匹配。

Important: 对于使用用于事务控制的数据库命令PL/pgSQL分组语句，不要混淆使用BEGIN和END关键字。PL/pgSQL BEGIN和END关键字仅用于分组; 他们不会开始或结束事务。函数总是在外部查询建立的事务中执行 - 它们无法启动

或提交该事务，因为它们没有上下文可供执行。但是，包含EXCEPTION子句的PL/pgSQL块有效地形成了一个子事务，可以在不影响外部事务的情况下回滚。有关EXCEPTION子句的更多信息，请参阅

<https://www.postgresql.org/docs/9.4/plpgsql-control-structures.html#PLPGSQL-ERROR-TRAPPING>。

关键字不区分大小写。除非双引号，否则标识符将隐式转换为小写，就像在普通SQL命令中一样。

PL/pgSQL代码中的注释与普通SQL中的注释相同：

- 双短划线(--)开始一条延伸到该行末尾的注释。
- /* 启动一个块注释，扩展到匹配的 */。
阻止评论嵌套。

块的语句部分中的任何语句都可以是子块。子块可用于逻辑分组或将变量本地化为了一组语句。

在子块中声明的变量在子块的持续时间内屏蔽外部块的任何类似命名的变量。如果使用块的标签限定其名称，则可以访问外部变量。例如，此函数多次声明一个名为quantity的变量：

```
CREATE FUNCTION testfunc() RETURNS integer AS $$
<< outerblock >>
DECLARE
    quantity integer := 30;
BEGIN
    RAISE NOTICE 'Quantity here is %', quantity; -- Prints 30
    quantity := 50;
    --
    -- Create a subblock
    --
    DECLARE
        quantity integer := 80;
    BEGIN
        RAISE NOTICE 'Quantity here is %', quantity; -- Prints 80
        RAISE NOTICE 'Outer quantity here is %', outerblock.quantity; -- Prints 50
    END;
    RAISE NOTICE 'Quantity here is %', quantity; -- Prints 50
    RETURN quantity;
END;
$$ LANGUAGE plpgsql;
```

执行SQL命令

您可以使用PL/pgSQL语句执行SQL命令，例如EXECUTE，PERFORM和SELECT ... INTO。有关PL/pgSQL语句的信息，请参阅 <https://www.postgresql.org/docs/9.4/plpgsql-statements.html>。

Note: EXECUTE语句不支持PL/pgSQL语句SELECT INTO。

PL/pgSQL计划缓存

PL/pgSQL函数的波动率分类对MPP数据库如何缓存引用该函数的计划有影响。有关MPP数据库函数波动率类别的计划缓存注意事项的信息，请参阅*MPP数据*

库管理员手册中的[函数波动率和计划缓存](#)。

当PL/pgSQL函数在数据库会话中第一次执行时，PL/pgSQL解释器会解析函数的SQL表达式和命令。解释器创建一个准备好的执行计划，因为每个表达式和SQL命令首先在函数中执行。PL/pgSQL解释器在数据库连接的生命周期中重用特定表达式和SQL命令的执行计划。虽然这种重用大大减少了解析和生成计划所需的总时间，但是直到执行该部分函数的运行时才能检测到特定表达式或命令中的错误。

如果对查询中使用的任何关系进行任何架构更改，或者重新定义查询中使用的任何用户定义函数，MPP数据库将自动重新规划已保存的查询计划。这使得在大多数情况下重复使用准备好的计划是透明的。

您在PL/pgSQL函数中使用的SQL命令必须在每次执行时引用相同的表和列。您不能将参数用作SQL命令中的表或列的名称。

PL/pgSQL为实际参数类型的每个组合缓存一个单独的查询计划，您可以在其中调用多态函数以确保数据类型差异不会导致意外故障。

有关PL/pgSQL语言中计划缓存注意事项的详细讨论，请参阅PostgreSQL[计划缓存](#)文档。

PL/pgSQL示例

以下是PL/pgSQL用户定义函数的示例。

示例：函数参数的别名

传递给函数的参数使用诸如\$1，\$2之类的标识符命名。（可选）可以为\$n参数名称声明别名，以提高可读性。然后可以使用别名或数字标识符来引用参数

值。

有两种方法可以创建别名。首选方法是在CREATE FUNCTION命令中为参数指定名称，例如：

```
CREATE FUNCTION sales_tax(subtotal real) RETURNS real AS $$  
BEGIN  
    RETURN subtotal * 0.06;  
END;  
$$ LANGUAGE plpgsql;
```

您还可以使用声明语法显式声明别名：

```
name ALIAS FOR $n;
```

此示例使用DECLARE语法创建相同的函数。

```
CREATE FUNCTION sales_tax(real) RETURNS real AS $$  
DECLARE  
    subtotal ALIAS FOR $1;  
BEGIN  
    RETURN subtotal * 0.06;  
END;  
$$ LANGUAGE plpgsql;
```

示例：使用表列的数据类型

声明变量时，可以使用%TYPE构造指定变量或表列的数据类型。这是声明其类型是表列的数据类型的变量的语法：

```
name table.column_name%TYPE;
```

您可以使用%TYPE构造来声明将保存数据库值的变量。例如，假设您的users表中有一个名为user_id的列。声明名为my_userid的变量，其数据类型与users.user_id列相同：

```
my_userid users.user_id%TYPE;
```

%TYPE在多态函数中特别有价值，因为内部变量所需的数据类型可能会从一个调用更改为下一个调用。通过将%TYPE应用于函数的参数或结果占位符，可以创建适当的变量。

示例：基于表行的复合类型

复合类型的变量称为行变量。以下语法基于表行声明复合变量：

```
name table_name%ROWTYPE;
```

这样的行变量可以保存SELECT或FOR查询结果的整行，只要该查询的列集与变量的声明类型匹配即可。使用通常的点表示法访问行值的各个字段，例如rowvar.column。

函数的参数可以是复合类型（完整的表行）。在这种情况下，相应的标识符\$n将是行变量，并且可以选择字段，例如\$1.user_id。

只能在行类型变量中访问表行的用户定义列，而不能访问OID或其他系统列。行类型的字段为数据类型（如char(n)）继承表的字段大小或精度。

下一个示例函数使用行变量复合类型。在创建函数之前，使用此命令创建函数使用的表。


```
CREATE TABLE table1 (  
    f1 text,  
    f2 numeric,  
    f3 integer  
) distributed by (f1);
```

此INSERT命令将数据添加到表中。

```
INSERT INTO table1 values  
( 'test1', 14.1, 3),  
( 'test2', 52.5, 2),  
( 'test3', 32.22, 6),  
( 'test4', 12.1, 4) ;
```

此函数使用基于table1的列%TYPE变量和%ROWTYPE复合变量。

```
CREATE OR REPLACE FUNCTION t1_calc( name text) RETURNS integer  
AS $$  
DECLARE  
    t1_row  table1%ROWTYPE;  
    calc_int table1.f3%TYPE;  
BEGIN  
    SELECT * INTO t1_row FROM table1 WHERE table1.f1 = $1 ;  
    calc_int = (t1_row.f2 * t1_row.f3)::integer ;  
    RETURN calc_int ;  
END;  
$$ LANGUAGE plpgsql VOLATILE;
```

Note: 前一个函数被归类为VOLATILE函数，因为函数值可能在单个表扫描中发生变化。

以下SELECT命令使用该函数。

```
select t1_calc( 'test1' );
```

Note: 示例PL/pgSQL函数将SELECT与INTO子句一起使用。它与SQL命

令SELECT INTO不同。如果要从PL/pgSQL函数内的SELECT结果创建表，请使用SQL命令CREATE TABLE AS。

示例：使用可变数量的参数

只要所有可选参数都具有相同的数据类型，就可以声明PL/pgSQL函数接受可变数量的参数。必须将函数的最后一个参数标记为VARIADIC，并使用数组类型声明参数。您可以将包含VARIADIC参数的函数称为可变参数函数。

例如，这个可变参数函数返回数值变量数组的最小值：

```
CREATE FUNCTION mleast (VARIADIC numeric[])
  RETURNS numeric AS $$
  DECLARE minval numeric;
  BEGIN
    SELECT min($1[i]) FROM generate_subscripts( $1, 1) g(i) INTO minval;
    RETURN minval;
  END;
$$ LANGUAGE plpgsql;

CREATE FUNCTION

SELECT mleast(10, -1, 5, 4.4);
 mleast
-----
      -1
(1 row)
```

实际上，VARIADIC位置或其以外的所有实际参数都被收集到一维数组中。

您可以将已构造的数组传递给可变参数函数。当您想要在可变参数函数之间传递数组时，这尤其有用。在函数调用中指定VARIADIC，如下所示：

```
SELECT mleast(VARIADIC ARRAY[10, -1, 5, 4.4]);
```

这可以防止PL/pgSQL将函数的可变参数扩展为其元素类型。

示例：使用默认参数值

您可以使用某些或所有输入参数的默认值声明PL/pgSQL函数。只要调用的函数少于声明的参数个数，就会插入默认值。因为参数只能从实际参数列表的末尾省略，所以必须在使用默认值定义的参数之后为所有参数提供默认值。

示例：

```
CREATE FUNCTION use_default_args(a int, b int DEFAULT 2, c int DEFAULT 3)
  RETURNS int AS $$
DECLARE
  sum int;
BEGIN
  sum := $1 + $2 + $3;
  RETURN sum;
END;
$$ LANGUAGE plpgsql;
```

```
SELECT use_default_args(10, 20, 30);
 use_default_args
-----
          60
(1 row)
```

```
SELECT use_default_args(10, 20);
 use_default_args
-----
          33
(1 row)
```

```
SELECT use_default_args(10);
 use_default_args
-----
          15
(1 row)
```

您也可以使用=符号代替关键字DEFAULT。

示例：使用多态数据类型

PL/pgSQL支持多态`anyelement`，`anyarray`，`anyenum`和`anynonarray`类型。使用这些类型，您可以创建一个可在多种数据类型上运行的PL/pgSQL函数。有关MPP数据库中多态类型支持的其他信息，请参阅[MPP数据库数据类型](#)。

当PL/pgSQL函数的返回类型声明为多态类型时，会创建一个名为\$0的特殊参

数。\$0的数据类型标识从实际输入类型推导出的函数的返回类型。

在此示例中，您将创建一个多态函数，该函数返回两个值的总和：

```
CREATE FUNCTION add_two_values(v1 anyelement,v2 anyelement)
  RETURNS anyelement AS $$
DECLARE
  sum ALIAS FOR $0;
BEGIN
  sum := v1 + v2;
  RETURN sum;
END;
$$ LANGUAGE plpgsql;
```

执行add_two_values()，提供整数输入值：

```
SELECT add_two_values(1, 2);
 add_two_values
-----
              3
(1 row)
```

add_two_values()的返回类型是整数，即输入参数的类型。现在执行add_two_values()，提供浮点输入值：

```
SELECT add_two_values (1.1, 2.2);
 add_two_values
-----
              3.3
(1 row)
```

在这种情况下，add_two_values()的返回类型是float。

您还可以在多态函数中指定VARIADIC参数。

示例：匿名块

此示例使用DO命令将前一个示例中的t1_calc()函数中的语句作为匿名块执行。在该示例中，匿名块从临时表中检索输入值。

```
CREATE TEMP TABLE list AS VALUES ('test1') DISTRIBUTED RANDOMLY;

DO $$
DECLARE
    t1_row table1%ROWTYPE;
    calc_int table1.f3%TYPE;
BEGIN
    SELECT * INTO t1_row FROM table1, list WHERE table1.f1 = list.column1 ;
    calc_int = (t1_row.f2 * t1_row.f3)::integer ;
    RAISE NOTICE 'calculated value is %', calc_int ;
END $$ LANGUAGE plpgsql ;
```

参考

关于PL/pgSQL的PostgreSQL文档是

在<https://www.postgresql.org/docs/9.4/plpgsql.html>

另请参阅MPP数据库参考手册中的[CREATE FUNCTION](#)命令。

有关内置MPP数据库函数的摘要，请参阅MPP数据库参考手册中的[内置函数摘要](#)。有关使用MPP数据库函数的信息，请参阅MPP数据库管理员手册中的“查询数据”

有关移植Oracle函数的信息，请参

阅<https://www.postgresql.org/docs/9.4/plpgsql-porting.html>。有关在MPP数据库中安装和使用Oracle兼容性函数的信息，请参阅MPP数据库实用程序手册中的“Oracle兼容性函数”。

MPP PL/R 语言扩展

本章包含以下信息：

- [关于MPP数据库PL/R](#)
- [启用PL/R语言支持](#)
- [示例](#)
- [下载和安装R包](#)
- [显示R库信息](#)
- [在启动时加载R模块](#)
- [参考](#)

Parent topic: [MPP数据库参考手册](#)

关于MPP数据库PL/R

PL/R是一种过程式语言。使用MPP数据库PL/R扩展，您可以使用R编程语言编写数据库函数，并使用包含R函数和数据集的R包。

有关支持的PL/R版本的信息，请参阅*MPP数据库发行说明*。

启用PL/R语言支持

对于需要使用它的每个数据库，使用SQL命令CREATE EXTENSION注册PL/R语言。由于PL/R是不可信的语言，因此只有超级用户才能将PL/R注册到数据库。例如，以gpadmin用户身份运行此命令，以使用名为testdb的数据库注册语言：

```
$ psql -d testdb -c 'CREATE EXTENSION plr;'
```

PL/R注册为不受信任的语言。

示例

以下是简单的PL/R示例。

示例1：将PL/R用于单行运算符

此函数使用R函数rnorm()生成具有正态分布的数字数组。

```
CREATE OR REPLACE FUNCTION r_norm(n integer, mean float8,  
    std_dev float8) RETURNS float8[ ] AS  
$$  
    x<-rnorm(n,mean,std_dev)  
    return(x)  
$$  
LANGUAGE 'plr';
```

以下CREATE TABLE命令使用r_norm函数填充表。r_norm函数创建一个包含10个数字的数组。


```
CREATE TABLE test_norm_var
AS SELECT id, r_norm(10,0,1) as x
FROM (SELECT generate_series(1,30:: bigint) AS ID) foo
DISTRIBUTED BY (id);
```

示例2：以表格形式返回PL/R data.frames

假设您的PL/R函数返回R data.frame作为其输出，除非您想使用数组数组，否则需要做一些工作才能将PL/R中的data.frame看作一个简单的SQL表：

- 在MPP数据库中创建一个与R data.frame具有相同尺寸的TYPE：

```
CREATE TYPE t1 AS ...
```

- 定义PL/R函数时使用此TYPE

```
... RETURNS SET OF t1 AS ...
```

下一个示例给出了示例SQL。

示例3：使用PL/R的分层回归

下面的SQL定义了一个TYPE并使用PL/R运行层次回归：

```
--Create TYPE to store model results
DROP TYPE IF EXISTS wj_model_results CASCADE;
CREATE TYPE wj_model_results AS (
  cs text, coefest float, ci_95_lower float, ci_95_upper float,
  ci_90_lower float, ci_90_upper float, ci_80_lower float,
  ci_80_upper float);

--Create PL/R function to run model in R
DROP FUNCTION IF EXISTS wj_plr_RE(float [ ], text [ ]);
CREATE FUNCTION wj_plr_RE(response float [ ], cs text [ ])
RETURNS SETOF wj_model_results AS
$$
  library(arm)
  y<- log(response)
  cs<- cs
  d_temp<- data.frame(y,cs)
  m0 <- lmer (y ~ 1 + (1 | cs), data=d_temp)
  cs_unique<- sort(unique(cs))
  n_cs_unique<- length(cs_unique)
  temp_m0<- data.frame(matrix(0,n_cs_unique, 7))
  for (i in 1:n_cs_unique){temp_m0[i,<-
    c(exp(coef(m0)$cs[i,1] + c(0,-1.96,1.96,-1.65,1.65,
      -1.28,1.28)*se.ranef(m0)$cs[i]))}
  names(temp_m0)<- c("Coefest", "CI_95_Lower",
    "CI_95_Upper", "CI_90_Lower", "CI_90_Upper",
    "CI_80_Lower", "CI_80_Upper")
  temp_m0_v2<- data.frames(cs_unique, temp_m0)
  return(temp_m0_v2)
$$
LANGUAGE 'plr';

--Run modeling plr function and store model results in a
--table
DROP TABLE IF EXISTS wj_model_results_roi;
CREATE TABLE wj_model_results_roi AS SELECT *
FROM wj_plr_RE((SELECT wj_droi2_array),
  (SELECT cs FROM wj_droi2_array));
```

1. 软件包

R包是包含R函数和数据集模块。您可以安装R软件包以扩展MPP数据库中的R和PL/R功能。

Note: 如果扩容MPP数据库并添加segment主机，则必须在新主机的R安装中安装R软件包。

1. 对于R包，标识所有相关R包和每个包Web URL。通过从以下导航页面中选择给定的包，可以找到该信息：

https://cran.r-project.org/web/packages/available_packages_by_name.html

例如，R包arm的页面表明该包需要以下R库：

Matrix，lattice，lme4，R2WinBUGS，coda，abind，foreign和MASS。

您还可以尝试使用R CMD INSTALL命令安装软件包以确定依赖软件包。

对于MPP数据库PL/R扩展中包含的R安装，所需的R软件包随PL/R扩展一起安装。但是，Matrix包需要更新的版本。

2. 从命令行，使用wget实用程序将arm包的tar.gz文件下载到MPP数据库master主机：

```
wget https://cran.r-project.org/src/contrib/Archive/arm/arm_1.5-03.tar.gz
```

```
wget https://cran.r-project.org/src/contrib/Archive/Matrix/Matrix_0.9996875-1.tar.gz
```

3. 使用gpscp实用程序和hosts_all文件将tar.gz文件复制到MPP数据库集群的所有节点上的同一目录。hosts_all文件包含所有MPP数据库segment主机的列表。您可能需要root访问权限才能执行此操作。

```
gpscp -f hosts_all Matrix_0.9996875-1.tar.gz =:/home/gpadmin
```

```
gpscp -f /hosts_all arm_1.5-03.tar.gz =:/home/gpadmin
```

4. 在交互模式下使用gpssh实用程序登录每个MPP数据库segment主机（gpssh

-f all_hosts) 。 使用R CMD INSTALL命令从命令提示符安装软件包。 请注意，这可能需要root访问权限。 例如，此R install命令安装arm包的包。

```
$R_HOME/bin/R CMD INSTALL Matrix_0.9996875-1.tar.gz arm_1.5-03.tar.gz
```

5. 确保软件包安装在所有segment的\$R_HOME/library目录中（gpssh可用于安装软件包）。 例如，此gpssh命令列出R库目录的内容。

```
gpssh -s -f all_hosts "ls $R_HOME/library"
```

gpssh选项-s在远程主机上运行命令之前获取mpp_path.sh文件。

6. 测试是否可以加载R包。

如果可以加载R包，则此函数执行简单测试：

```
CREATE OR REPLACE FUNCTION R_test_require(fname text)
RETURNS boolean AS
$BODY$
    return(require(fname,character.only=T))
$BODY$
LANGUAGE 'plr';
```

此SQL命令检查是否可以加载R包arm：

```
SELECT R_test_require('arm');
```

显示R库信息

您可以使用R命令行显示有关MPP数据库主机上已安装的库和函数的信息。您还可以在R安装中添加和删除库。 要在主机上启动R命令行，请以gadmin用户身份登录主机，并从\$GPHOME/ext/R-3.3.3/bin目录运行脚本R。

此R函数列出R命令行中的可用R包：

```
> library()
```

显示特定R包的文档

```
> library(help="package_name")  
> help(package="package_name")
```

显示R函数的帮助文件：

```
> help("function_name")  
> ?function_name
```

要查看已安装的软件包，请使用R命令`installed.packages()`。这将返回一个矩阵，其中包含已安装的每个包的行。下面，我们来看看这个矩阵的前5行。

```
> installed.packages()
```

必须先安装并加载任何未出现在已安装的软件包矩阵中的软件包，然后才能使用其函数。

可以使用`install.packages()`安装R包：

```
> install.packages("package_name")  
> install.packages("mypkg", dependencies = TRUE, type="source")
```

从R命令行加载包。

```
> library(" package_name ")
```

可以使用remove.packages删除R包

```
> remove.packages("package_name")
```

您可以使用R命令-e选项从命令行运行函数。例如，此命令在R包MASS上显示帮助。

```
$ R -e 'help("MASS")'
```

在启动时加载R模块

PL/R可以在解释器初始化期间自动加载保存的R代码。要使用此功能，请创建plr_modules数据库表，然后将要自动加载的R模块插入表中。如果表存在，PL/R会将其包含的代码加载到解释器中。

在MPP数据库系统中，表行通常是分布式的，因此每行只存在于一个segment实例中。但是，每个segment实例的R解释器需要加载所有模块，因此正态分布的表将不起作用。必须将plr_modules表创建为默认架构中的复制表，以便表中的所有行都出现在每个segment实例中。例如：

```
CREATE TABLE public.plr_modules {  
  modseq int4,  
  modsrc text  
} DISTRIBUTED REPLICATED;
```

有关使用PL/R自动加载功能的更多信息，请参阅
<https://www.joeconway.com/plr/doc/plr-module-funcs.html>。

参考

<https://www.r-project.org/> - R Project主页

<https://cran.r-project.org/web/packages/R/> - R的主页，提供了一个R接口，用于对MPP数据库表和视图进行操作，类似于R data.frame。R还支持直接从R使用机器学习包[MADlib](#)。

R文档随MPP R软件包一起安装：

\$GPHOME/ext/R-3.3.3/doc

R函数和参数

- 参考<https://www.joeconway.com/doc/plr-funcs.html>

在R中传递数据值

- 参考<https://www.joeconway.com/doc/plr-data.html>

R中的聚合函数

- 参考<https://www.joeconway.com/doc/plr-aggregate-funcs.html>

MPP PL/Python语言扩展

本节概述了MPP数据库PL/Python语言。

- [关于MPP PL/Python](#)
- [启用和删除PL/Python支持](#)
- [用PL/Python开发函数](#)
- [安装Python模块](#)
- [示例](#)
- [参考](#)

Parent topic: [MPP数据库参考手册](#)

关于MPP PL/Python

PL/Python是一种可加载的过程语言。使用MPP数据库PL/Python扩展，您可以在Python中编写MPP数据库用户定义函数，利用Python功能和模块快速构建强大的数据库应用程序。

您可以将PL/Python代码块作为匿名代码块运行。请参阅[MPP数据库参考手册](#)中的[DO](#)命令。

MPP数据库PL/Python扩展默认安装在MPP数据库中。MPP数据库安装了Python和PL/Python的一个版本。这是MPP数据库使用的Python安装的位置：

```
$GPHOME/ext/python/
```

MPP数据库PL/Python限制

- MPP数据库不支持PL/Python触发器。
 - PL/Python仅作为MPP数据库不可信语言提供。
-

- 不支持可更新游标（UPDATE ... WHERE CURRENT OF和DELETE ... WHERE CURRENT OF）。

启用和删除PL/Python支持

PL/Python语言随MPP数据库一起安装。要在数据库中创建和运行PL/Python用户定义函数（UDF），必须在数据库中注册PL/Python语言。

启用PL/Python支持

对于需要使用它的每个数据库，使用SQL命令CREATE EXTENSION注册PL/Python语言。由于PL/Python是一种不受信任的语言，因此只有超级用户才能将PL/Python注册到数据库中。例如，当gpadmin用户使用名为testdb的数据库注册PL/Python时，运行此命令：

```
$ psql -d testdb -c 'CREATE EXTENSION plpythonu;'
```

PL/Python注册为不受信任的语言。

删除PL/Python支持

对于不再需要PL/Python语言的数据库，请使用SQL命令DROP EXTENSION删除对PL/Python的支持。由于PL/Python是一种不受信任的语言，因此只有超级用户才能从数据库中删除对PL/Python语言的支持。例如，以gpadmin用户身份运行此命令将从名为testdb的数据库中删除对PL/Python的支持：

```
$ psql -d testdb -c 'DROP EXTENSION plpythonu;'
```

如果任何现有对象（如函数）依赖于语言，则默认命令将失败。指定CASCADE选项也可以删除所有依赖对象，包括使用PL/Python创建的函数。

用PL/Python开发函数

PL/Python用户定义函数的主体是Python脚本。调用该函数时，其参数将作为数组args[]的元素传递。命名参数也作为普通变量传递给Python脚本。结果从带有return语句的PL/Python函数返回，或者在结果集语句的情况下从yield语句返回。

数组和列表

使用Python列表将SQL数组值传递到PL/Python函数中。类似地，PL/Python函数将SQL数组值作为Python列表返回。在典型的PL/Python使用模式中，您将使用[]指定数组。

以下示例创建一个返回整数数组的PL/Python函数：

```
CREATE FUNCTION return_py_int_array()
  RETURNS int[]
AS $$
  return [1, 11, 21, 31]
$$ LANGUAGE plpythonu;

SELECT return_py_int_array();
 return_py_int_array
-----
{1,11,21,31}
(1 row)
```

PL/Python将多维数组视为列表列表。使用嵌套的Python列表将多维数组传递给

PL/Python函数。当PL/Python函数返回一个多维数组时，每个级别的内部列表必须都具有相同的大小。

以下示例创建一个PL/Python函数，该函数将多维数组的整数作为输入。该函数显示提供的参数的类型，并返回多维数组：

```
CREATE FUNCTION return_multidim_py_array(x int4[])
  RETURNS int4[]
AS $$
  plpy.info(x, type(x))
  return x
$$ LANGUAGE plpythonu;

SELECT * FROM return_multidim_py_array(ARRAY[[1,2,3], [4,5,6]]);
INFO: ([[1, 2, 3], [4, 5, 6]], <type 'list'>)
CONTEXT: PL/Python function "return_multidim_py_type"
return_multidim_py_array
-----
{{1,2,3},{4,5,6}}
(1 row)
```

PL/Python还接受其他Python序列（如元组）作为函数参数，以便向后兼容不支持多维数组的MPP版本。在这种情况下，Python序列始终被视为一维数组，因为它们与复合类型不一致。

复合类型

使用Python映射将复合类型参数传递给PL/Python函数。映射的元素名称是复合类型的属性名称。如果属性具有空值，则其映射值为None。

您可以将复合类型结果作为序列类型（元组或列表）返回。在多维数组中使用复合类型时，必须将复合类型指定为元组，而不是列表。您不能将复合类型数组作为列表返回，因为确定列表是表示复合类型还是其他数组维度是不明确的。在典型的使用模式中，您将使用()指定复合类型元组。

在以下示例中，您将创建一个复合类型和一个返回复合类型数组的PL/Python函数：

```
CREATE TYPE type_record AS (  
    first text,  
    second int4  
);  
  
CREATE FUNCTION composite_type_as_list()  
    RETURNS type_record[]  
AS $$  
    return [('first', 1), ('second', 1)], [('first', 2), ('second', 2)], [('first', 3), ('second', 3)];  
$$ LANGUAGE plpythonu;  
  
SELECT * FROM composite_type_as_list();  
               composite_type_as_list  
-----  
{{('first,1'),('second,1')},{('first,2'),('second,2')},{('first,3'),('second,3')}}  
(1 row)
```

有关PL/Python处理数组和复合类型的其他信息，请参阅[PostgreSQL数组，列表文档](#)。

执行和准备SQL查询

PL/Python plpy模块提供了两个Python函数来执行SQL查询并为查询plpy.execute和plpy.prepare准备执行计划。如果从多个Python函数运行查询，则为查询准备执行计划很有用。

PL/Python还支持plpy.subtransaction()函数，以帮助在显式子事务中管理plpy.execute调用。有关plpy.subtransaction()的其他信息，请参阅PostgreSQL文档中的[显式子事务](#)。

plpy.execute

使用查询字符串和可选的限制参数调用plpy.execute会导致运行查询并在Python结果对象中返回结果。结果对象模拟列表或字典对象。可以通过行号和列名访问结果对象中返回的行。结果集行编号从0（零）开始。可以修改结果对象。

结果对象具有以下附加方法：

- `nrows` 返回查询返回的行数。
- `status` 这是SPI_execute()返回值。

例如，PL/Python用户定义函数中的此Python语句执行查询。

```
rv = plpy.execute("SELECT * FROM my_table", 5)
```

plpy.execute函数最多可从my_table返回5行。结果集存储在rv对象中。如果my_table有一个列my_column，它将被访问为：

```
my_col_data = rv[i]["my_column"]
```

由于函数最多返回5行，因此索引*i*可以是0到4之间的整数。

plpy.prepare

函数plpy.prepare为查询准备执行计划。如果在查询中有参数引用，则使用查询字符串和参数类型列表调用它。例如，此语句可以在PL/Python用户定义的函数中：

```
plan = plpy.prepare("SELECT last_name FROM my_users WHERE  
first_name = $1", [ "text" ])
```

字符串text是为变量\$1传递的变量的数据类型。准备好语句后，使用plpy.execute函数运行它：

```
rv = plpy.execute(plan, [ "Fred" ], 5)
```

第三个参数是返回的行数限制，是可选的。

使用PL/Python模块准备执行计划时，计划会自动保存。有关执行计划的信息，请参阅Postgres服务器编程接口（SPI）文档<https://www.postgresql.org/docs/9.4/spi.html>。

要在函数调用中有效使用已保存的计划，请使用Python持久存储字典SD或GD之一。

全局字典SD可用于在函数调用之间存储数据。此变量是私有静态数据。全局字典GD是公共数据，可用于会话中的所有Python函数。小心使用GD。

每个函数在Python解释器中都有自己的执行环境，因此myfunc的全局数据和函数参数不可用于myfunc2。如前所述，GD词典中的数据是例外。

此示例使用SD字典：

```
CREATE FUNCTION usesavedplan() RETURNS trigger AS $$
  if SD.has_key("plan"):
    plan = SD["plan"]
  else:
    plan = plpy.prepare("SELECT 1")
    SD["plan"] = plan

  # rest of function

$$ LANGUAGE plpythonu;
```

处理Python错误和消息

Python模块plpy实现了这些函数来管理错误和消息：

- `plpy.debug`
- `plpy.log`
- `plpy.info`
- `plpy.notice`
- `plpy.warning`
- `plpy.error`
- `plpy.fatal`
- `plpy.debug`

消息函数`plpy.error`和`plpy.fatal`引发一个Python异常，如果未被捕获，则传播到调用查询，导致当前事务或子事务中止。函数`raise plpy.ERROR(msg)`和`raise plpy.FATAL(msg)`分别与`plpy.error`和`plpy.fatal`等价。其他消息函数仅生成不同优先级的消息。

是否向客户端报告特定优先级的消息，写入服务器日志，还是两者都由MPP数据库服务器配置参数`log_min_messages`和`client_min_messages`控制。有关参数的信息，请参阅*MPP数据库参考手册*。

使用字典GD提高PL/Python性能

在性能方面，导入Python模块是一项昂贵的操作，可能会影响性能。如果要经常导入同一模块，则可以使用Python全局变量在第一次调用时加载模块，而不需要在后续调用中导入模块。以下PL/Python函数使用GD持久存储字典来避免导入模块（如果模块已经导入并且在GD中）。

```
psql=#  
CREATE FUNCTION pytest() returns text as $$  
  if 'mymodule' not in GD:  
    import mymodule  
    GD['mymodule'] = mymodule  
  return GD['mymodule'].sumd([1,2,3])  
$$;
```

安装Python模块

在MPP数据库上安装Python模块时，MPP数据库Python环境必须在群集中的所有segment主机和mirror主机上添加模块。扩容MPP数据库时，必须将Python模块添加到新的segment主机中。您可以在MPP数据库主机上使用MPP数据库实用程序gpssh和gpscp运行命令，并将文件复制到主机。有关实用程序的信息，请参阅*MPP数据库实用程序手册*。

作为MPP数据库安装的一部分，gpadmin用户环境配置为使用随MPP数据库一起安装的Python。

要检查Python环境，可以使用which命令：

```
which python
```

该命令返回Python安装的位置。与MPP数据库一起安装的Python位于MPP数据库ext/python目录中。

```
/path_to_mpp-db/ext/python/bin/python
```

使用gpssh在远程主机上运行shell命令时，可以指定-s选项。指定该选项后，gpssh会在远程主机上运行命令之前加载mpp_path.sh文件。例如，此命令应显示在每台主机上使用MPP数据库安装的Python。


```
gpssh -f gpdb_hosts which python
```

如果没有，则可以在运行命令之前在远程主机上添加-s到源mpp_path.sh。

```
gpssh -s -f gpdb_hosts which python
```

要显示当前安装的Python模块列表，请运行此命令。

```
python -c "help('modules')"
```

以交互模式运行gpssh以在远程主机上显示Python模块。此示例以交互模式启动gpssh，并在MPP数据库主机sdw1上列出Python模块。

```
$ gpssh -s -h sdw1
=> python -c "help('modules')"
. . .
=> exit
$
```

这些部分描述了安装和测试Python模块：

- [安装Python pip](#)
- [使用pip安装Python包](#)
- [在本地构建和安装Python模块](#)
- [测试已安装的Python模块](#)

安装Python pip

Python实用程序pip安装Python包，其中包含Python模块和来自版本化归档文件的其他资源文件。

运行此命令以安装pip。

```
python -m ensurepip --default-pip
```

该命令运行ensurepip模块以从本地Python安装引导（安装和配置）pip实用程序。

您可以运行此命令以确保pip，setuptools和wheel项目是最新的。当前的Python项目确保您可以从源代码分发或预构建的分发（wheels）安装Python包。

```
python -m pip install --upgrade pip setuptools wheel
```

您可以使用gpssh在MPP数据库主机上运行命令。此示例以交互模式运行gpssh，以在文件gpdb_hosts中列出的主机上安装pip。

```
$ gpssh -s -f gpdb_hosts
=> python -m ensurepip --default-pip
[centos6-mdw1] Ignoring indexes: https://pypi.python.org/simple
[centos6-mdw1] Collecting setuptools
[centos6-mdw1] Collecting pip
[centos6-mdw1] Installing collected packages: setuptools, pip
[centos6-mdw1] Successfully installed pip-8.1.1 setuptools-20.10.1
[centos6-sdw1] Ignoring indexes: https://pypi.python.org/simple
[centos6-sdw1] Collecting setuptools
[centos6-sdw1] Collecting pip
[centos6-sdw1] Installing collected packages: setuptools, pip
[centos6-sdw1] Successfully installed pip-8.1.1 setuptools-20.10.1
=> exit
$
```

=>是gpssh的非活动提示。该实用程序显示每个主机的输出。exit命令退出gpssh交互模式。

此gpssh命令在文件gpdb_hosts中列出的所有主机上运行单个命令。

```
gpssh -s -f gpdb_hosts python -m pip install --upgrade pip setuptools wheel
```

该实用程序显示每个主机的输出。

有关安装Python包的更多信息，参考<https://packaging.python.org/tutorials/installing-packages/>。

使用pip安装Python包

安装pip后，您可以安装Python包。此命令安装numpy和scipy包。

```
python -m pip install --user numpy scipy
```

--user选项尝试在安装Python包时避免冲突。

您可以使用gpssh在MPP数据库主机上运行该命令。

有关这些和其他Python包的信息，请参阅[参考](#)。

在本地构建和安装Python模块

如果要构建Python模块，则必须确保构建创建正确的可执行文件。例如，在Linux系统上，构建应该创建一个64位可执行文件。

在构建要安装的Python模块之前，请确保已安装并正确配置构建模块的相应软件。只有在构建模块的主机上才需要构建环境。

您可以使用MPP数据库实用程序gpssh和gpscp在MPP数据库主机上运行命令并将文件复制到主机。

测试已安装的Python模块

您可以创建一个简单的PL/Python用户定义函数（UDF）来验证Python模块在MPP数据库中是否可用。此示例测试NumPy模块。

此PL/Python UDF导入NumPy模块。如果导入模块成功，则函数返回SUCCESS;如果发生导入错误，则返回FAILURE。

```
CREATE OR REPLACE FUNCTION plpy_test(x int)
returns text
as $$
try:
    from numpy import *
    return 'SUCCESS'
except ImportError, e:
    return 'FAILURE'
$$ language plpythonu;
```

创建一个包含每个MPP数据库segment实例上的数据的表。根据MPP数据库安装的大小，您可能需要生成更多数据以确保将数据分发到所有segment实例。

```
CREATE TABLE DIST AS (SELECT x FROM generate_series(1,50) x ) DISTRIBUTED RANDOMLY;
```

此SELECT命令在segment主机上运行UDF，其中数据存储在primary实例中。

```
SELECT gp_segment_id, plpy_test(x) AS status
FROM dist
GROUP BY gp_segment_id, status
ORDER BY gp_segment_id, status;
```

如果UDF在MPP数据库segment实例上成功导入Python模块，则SELECT命令返回SUCCESS。如果SELECT命令返回FAILURE，则可以找到segment实例主机的segment主机。MPP数据库系统表gp_segment_configuration包含有关mirror和segment配置的信息。此命令返回segment ID的主机名。

```
SELECT hostname, content AS seg_ID FROM gp_segment_configuration
WHERE content = seg_id ;
```

如果返回FAILURE，则可能是以下原因：

- 访问所需库时出现问题。对于NumPy示例，MPP数据库可能在访问OpenBLAS库或segment主机上的Python库时遇到问题。在segment主机上以gpadmin用户身份运行命令时，请确保没有错误。此gpssh命令测试在segment主机mdw1上导入numpy模块。

```
gpssh -s -h mdw1 python -c "import numpy"
```

- 如果Python import命令未返回错误，则可能未在MPP数据库环境中配置环境变量。例如，在主机系统上安装Python包之后，可能尚未重新启动MPP数据库。

示例

此PL/Python UDF返回最多两个整数：

```
CREATE FUNCTION pymax (a integer, b integer)
RETURNS integer
AS $$
    if (a is None) or (b is None):
        return None
    if a > b:
        return a
    return b
$$ LANGUAGE plpythonu;
```

您可以使用STRICT属性执行空值处理，而不是使用两个条件语句。

```
CREATE FUNCTION pymax (a integer, b integer)
  RETURNS integer AS $$
return max(a,b)
$$ LANGUAGE plpythonu STRICT ;
```

您可以使用SELECT命令运行用户定义的函数pymax。此示例运行UDF并显示输出。

```
SELECT ( pymax(123, 43));
column1
-----
    123
(1 row)
```

此示例返回针对表运行的SQL查询中的数据。这两个命令创建一个简单的表并向表中添加数据。

```
CREATE TABLE sales (id int, year int, qtr int, day int, region text)
  DISTRIBUTED BY (id) ;

INSERT INTO sales VALUES
(1, 2014, 1,1, 'usa'),
(2, 2002, 2,2, 'europe'),
(3, 2014, 3,3, 'asia'),
(4, 2014, 4,4, 'usa'),
(5, 2014, 1,5, 'europe'),
(6, 2014, 2,6, 'asia'),
(7, 2002, 3,7, 'usa') ;
```

此PL/Python UDF执行SELECT命令，该命令从表中返回5行。Python函数从输入值指定的行返回REGION值。在Python函数中，行编号从0开始。函数的有效输入是0到4之间的整数。

```
CREATE OR REPLACE FUNCTION mypytest(a integer)
  RETURNS setof text
AS $$
  rv = plpy.execute("SELECT * FROM sales ORDER BY id", 5)
  region = []
  region.append(rv[a]["region"])
  return region
$$ language plpythonu EXECUTE ON MASTER;
```

运行此SELECT语句将从结果集的第三行返回REGION列值。

```
SELECT mypytest(2) ;
```

此命令从数据库中删除UDF。

```
DROP FUNCTION mypytest(integer) ;
```

此示例使用DO命令将前一示例中的PL/Python函数作为匿名块执行。在该示例中，匿名块从临时表中检索输入值。

```
CREATE TEMP TABLE mytemp AS VALUES (2) DISTRIBUTED RANDOMLY;

DO $$
  temprow = plpy.execute("SELECT * FROM mytemp", 1)
  myval = temprow[0]["column1"]
  rv = plpy.execute("SELECT * FROM sales ORDER BY id", 5)
  region = rv[myval]["region"]
  plpy.notice("region is %s" % region)
$$ language plpythonu;
```

参考

技术参考

有关Python语言的信息，参考<https://www.python.org/>。

有关PL/Python的信息，请参阅PostgreSQL文档<https://www.postgresql.org/docs/9.4/plpython.html>。

有关Python包索引（PyPI）的信息，参考<https://pypi.python.org/pypi>。

这些是可以安装的一些Python模块：

- SciPy库提供用户友好且高效的数值例程，例如用于数值积分和优化的例程。SciPy站点包括其他类似的Python库 <http://www.scipy.org/index.html>。
- Natural Language Toolkit (nltk) 是一个用于构建Python程序以处理人类语言数据的平台。 <http://www.nltk.org/>。有关安装工具包的信息，请参阅<http://www.nltk.org/install.html>。

MPP PL/Container语言扩展

本节包括有关PL/Container 1.1及更高版本的以下信息：

- [关于PL/Container语言扩展](#)
 - [关于PL/Container资源管理](#)
 - [PL/Container Docker镜像](#)
 - [#topic_qbl_dkq_vcb](#)
 - [#topic_upg110](#)
 - [卸载PL/Container](#)
 - [使用PL/Container函数](#)
-

- [关于PL/Container运行PL/Python](#)
- [关于PL/Container运行PL/R](#)
- [配置PL/Container](#)
- [安装Docker](#)
- [参考](#)

Warning: PL/Container与MPP Database 5.2.0及更高版本兼容。 尚未测试PL/Container与MPP Database 5.1.0或5.0.0的兼容性。

Parent topic: [MPP数据库参考手册](#)

关于PL/Container语言扩展

MPP数据库PL/Container语言扩展（PL/Container）是一个接口，允许MPP数据库与Docker容器交互以在容器中执行用户定义的函数（UDF）。 Docker容器确保用户代码无法访问源主机的文件系统。 此外，容器在没有网络访问或网络访问受限的情况下启动，无法连接回MPP数据库或打开任何其他外部连接。 有关可用UDF语言的信息，请参阅[PL/Container Docker镜像](#)。

一般来说，Docker容器是一个Linux进程，它使用Linux内核功能（如cgroup，命名空间和联合文件系统）以托管方式运行。 Docker镜像是容器的基础。 Docker容器是Docker镜像的运行实例。 启动Docker容器时，指定Docker镜像。 Docker镜像是在主机系统上运行Docker容器时使用的根文件系统更改和执行参数的集合。 镜像没有状态，永远不会改变。 有关Docker的信息，请访问Docker网站<https://www.docker.com/>。

MPP数据库仅在第一次调用该容器中的函数时启动容器。 例如，考虑使用所有可用segment选择表数据的查询，并使用PL/Container函数对数据应用转换。 在这种情况下，MPP数据库将在每个segment上仅启动Docker容器一次，然后联系正在运行的容器以获取结果。

在开始查询执行的完整循环之后，执行程序向容器发送调用。容器可能会响应容器执行的SPI-SQL查询，以从数据库中获取一些数据，并将结果返回给查询执行器。

当与其连接关闭时，容器将关闭。当您关闭启动容器的MPP数据库会话时会发生这种情况。在待机模式下运行的容器几乎不消耗CPU资源，因为它在套接字上等待。PL/Container内存消耗取决于您在全局词典中缓存的数据量。

Warning: 当MPP数据库在Docker容器中运行时，不支持PL/Container。

PL/Container语言扩展可用作开源模块。有关该模块的信息，请参阅位于<https://github.com/mpp-db/plcontainer>的GitHub存储库中的README文件。

关于PL/Container资源管理

MPP数据库在Docker容器中运行PL/Container用户定义的函数。Docker容器和MPP数据库服务器共享相同主机上的CPU和内存资源。在默认情况下，MPP数据库不知道运行PL/Container实例所消耗的资源。在PL/Container 1.2及更高版本中，您可以使用MPP数据库资源组来控制运行PL/Container实例的总体CPU和内存资源使用情况，如以下部分所述。

PL/Container管理两个级别的资源使用 - 容器级别和runtime级别。您可以使用为PL/Container runtime配置的memory_mb和cpu_share设置来控制容器级CPU和内存资源。memory_mb控制每个容器实例可用的内存资源。cpu_share设置标识容器与其他容器相比的CPU使用率的相对权重。有关PL/Container配置信息，请参阅[PL/Container配置文件](#)。

默认情况下，您不能限制执行PL/Container容器实例的数量，也不能限制它们使用的内存或CPU资源总量。

使用资源组管理PL/Container资源

在PL/Container 1.2.0及更高版本中，您可以使用MPP数据库资源组来管理和限制PL/Container runtime中容器的总CPU和内存资源。有关启用，配置和使用MPP数据库资源组的更多信息，请参阅[MPP数据库管理员手册](#)中的[使用资源组](#)。

Note: 如果未显式配置PL/Container runtime的资源组，则其容器实例仅受系统资源的限制。容器可能会以MPP数据库服务器为代价消耗资源。

外部组件（如PL/Container）的资源组使用Linux控制组（cgroup）来管理组件级内存和CPU资源的使用。使用资源组管理PL/Container资源时，可以配置MPP数据库应用于共享相同PL/Container runtime配置的所有容器实例的内存限制和CPU限制。

创建资源组以管理PL/Container runtime的资源时，除了所需的CPU和内存限制外，还必须指定MEMORY_AUDITOR=cgroup和CONCURRENCY=0。例如，以下命令为PL/Container runtime创建名为plpy_run1_rg的资源组：

```
CREATE RESOURCE GROUP plpy_run1_rg WITH (MEMORY_AUDITOR=cgroup, CONCURRENCY=0, CPU_RATE_LIMIT=10, MEMORY_LIMIT=10);
```

PL/Container不使

用MEMORY_SHARED_QUOTA和MEMORY_SPILL_RATIO资源组内存限制。有关此SQL命令的详细信息，请参阅[CREATE RESOURCE GROUP](#)参考页。

您可以创建一个或多个资源组来管理正在运行的PL/Container实例。为PL/Container创建资源组后，将资源组分配给一个或多个PL/Container runtime。您使用资源组的groupid进行此分配。您可以从gp_resgroup_config gp_toolkit视图确定给定资源组名称的groupid。例如，以下查询显示名为plpy_run1_rg的资源组的groupid：

```
SELECT groupname, groupid FROM gp_toolkit.gp_resgroup_config
WHERE groupname='plpy_run1_rg';
```

```
groupname | groupid
-----+-----
plpy_run1_rg | 16391
(1 row)
```

通过为plcontainer runtime-add（新runtime）或plcontainer runtime-replace（现有runtime）命令指定-s resource_group_id=rg_groupid选项，可以将资源组分配给PL/Container runtime配置。例如，要将plpy_run1_rg资源组分配给名为python_run1的新PL/Container runtime：

```
plcontainer runtime-add -r python_run1 -i pivotaldata/plcontainer_python_shared:devel -l python
```

您还可以使用plcontainer runtime-edit命令将资源组分配给PL/Container runtime。有关plcontainer命令的信息，请参阅[plcontainer实用程序](#)。

将资源组分配给PL/Container runtime后，共享相同runtime配置的所有容器实例都受为组配置的内存和CPU的限制。如果减小PL/Container资源组的内存限制，则在组内运行容器中执行的查询可能会因内存不足而失败。如果在运行容器实例时删除PL/Container资源组，MPP数据库会终止正在运行的容器。

为PL/Container配置资源组

要使用MPP数据库资源组来管理PL/Container资源，必须显式配置资源组和PL/Container。

Note: PL/Container版本1.2利用MPP数据库版本5.8.0中引入的新资源组功能。如果您降级到使用PL/Container版本1.1或者更早的MPP数据库系统，您必须使用plcontainer runtime-edit从PL/Container runtime配置中删除任何resource_group_id设置。

过程

执行以下过程以配置PL/Container以使用MPP数据库资源组进行CPU和内存资源管理：

1. 如果尚未在MPP数据库部署中配置和启用资源组，请按照*MPP数据库管理员手册*中的[使用资源组](#)中的说明配置cgroup并启用MPP数据库资源组。
Note: 如果先前已在部署中配置并启用了资源组，请确保MPP数据库资源组gpdb.conf cgroups配置文件包含内存{}块，如上一个链接中所述。
2. 分析MPP数据库部署的资源使用情况。确定要分配给PL/Container Docker容器的资源组CPU和内存资源的百分比。
3. 确定如何在PL/Container runtime中分配您在上一步中确定的总PL/Container CPU和内存资源。确认：
 - 您需要的PL/Container资源组的数量。
 - 分配给每个资源组的内存和CPU资源的百分比。
 - 资源组到PL/Container runtime分配。
4. 创建您在上面的步骤中标识的PL/Container资源组。例如，假设您选择将25 % MPP数据库的内存和CPU资源分配给PL/Container。如果进一步将这些资源拆分为2个资源组60/40，则以下SQL命令将创建资源组：

```
CREATE RESOURCE GROUP plr_run1_rg WITH (MEMORY_AUDITOR=cgroup, CONCUP
CPU_RATE_LIMIT=15, MEMORY_LIMIT=15);
CREATE RESOURCE GROUP plpy_run1_rg WITH (MEMORY_AUDITOR=cgroup, CONCL
CPU_RATE_LIMIT=10, MEMORY_LIMIT=10);
```

5. 查找并记下与您创建的每个资源组关联的groupid。例如：

```
SELECT groupname, groupid FROM gp_toolkit.gp_resgroup_config
WHERE groupname IN ('plpy_run1_rg', 'plr_run1_rg');
```

```
groupname | groupid
-----+-----
plpy_run1_rg | 16391
plr_run1_rg | 16393
(1 row)
```

6. 将您创建的每个资源组分配给所需的PL/Container runtime配置。如果尚未创建runtime配置，请使用plcontainer runtime-add命令。如果runtime已存在，请使用plcontainer runtime-replace或plcontainer runtime-edit命令将资源组分配添加到runtime配置。例如：

```
plcontainer runtime-add -r python_run1 -i pivotaldata/plcontainer_python_shared:devel -l py
plcontainer runtime-replace -r r_run1 -i pivotaldata/plcontainer_r_shared:devel -l r -s resourc
```

有关plcontainer命令的信息，请参阅[plcontainer实用程序](#)。

PL/Container Docker镜像

PL/Python镜像和PL/R镜像可从 Network的MPP数据库产品下载站点获得，网址为<https://network.pivotal.io/>。

- 用于Python的PL/Container - 安装了Python 2.7.12的Docker镜像。
还安装了Python数据科学模块。该模块包含一组与数据科学相关的python库。
- 用于R的PL/Container - 安装了R-3.3.3的容器的Docker镜像。
还安装了R Data Science软件包。该软件包包含一组与数据科学相关的R库。

Docker镜像tag表示PL/Container版本（例如，1.0.0）。例如，用于Python

Docker镜像的PL/Container的完整Docker镜像名称类似于pivotaldata/plc_python_shared:1.0.0。这是默认PL/Container配置中引用的名称。此外，您可以创建自定义Docker镜像，安装镜像并将图像添加到PL/Container配置。

前提条件

确保您的MPP数据库系统满足以下前提条件：

- Red Hat Enterprise Linux (RHEL) 7.x (或更高版本) 和CentOS 7.x (或更高版本) 上的MPP数据库5.2.x支持PL/Container。
Note: RHEL/CentOS 6.x系统不支持PL/Container，因为这些平台不官方支持Docker。
- 这些是Docker主机操作系统的前提条件。
RHEL或CentOS 7.x - 支持的最低Linux操作系统内核版本为3.10。 RHEL 7.x和CentOS 7.x使用此内核版本。
您可以使用命令uname -r检查内核版本
Note: Red Hat提供，维护和支持的Docker版本仅在RHEL 7上可用。 Docker功能开发与RHEL7.x基础架构组件相关，用于内核，设备映射（精简配置，直接lvm），sVirt和systemd。
- Docker安装在MPP数据库主机（master，primary和所有standby主机）上
 - 对于RHEL或CentOS 7.x - Docker 17.05

请见[安装Docker](#)。
- 在每个MPP数据库主机上，gpadmin用户应该是docker组的一部分，以使用户能够管理Docker镜像和容器。

安装PL/Container语言扩展

要使用PL/Container，请安装PL/Container语言扩展，安装Docker镜像，并配置PL/Container以使用镜像。

1. 确保MPP数据库主机满足前提条件，参考[前提条件](#)。
2. 从源代码构建和安装PL/Container扩展，参考 [构建和安装PL/Container语言扩展](#)。
3. 安装Docker镜像并配置PL/Container，参考[安装PL/Container Docker镜像](#)。

构建和安装PL/Container语言扩展

PL/Container语言扩展可用作开源模块。有关构建和安装作为MPP数据库的一部分的模块信息，请参阅位于<https://github.com/mpp-db/plcontainer>的GitHub存储库中的README文件。

安装PL/Container Docker镜像

PL/Container语言扩展包括plcontainer实用程序，该实用程序在MPP数据库主机上安装Docker镜像，并将配置信息添加到PL/Container配置文件。配置信息允许PL/Container使用Docker镜像创建Docker容器。有关plcontainer的信息，请参阅[plcontainer实用程序](#)。

PL/Container开源模块包含dockerfiles，用于构建可与PL/Container一起使用的Docker镜像。您可以构建一个Docker镜像来运行PL/Python UDF和一个Docker镜像来运行PL/R UDF。请参阅GitHub存储库中的dockerfiles，网址为<https://github.com/mpp-db/plcontainer>。

在MPP数据库主机上安装Docker镜像。此示例使用plcontainer实用程序为Python安装Docker镜像并更新PL/Container配置。该示例假定要安装的Docker

镜像位于/home/gpadmin中的文件中。

此plcontainer命令从Docker镜像文件安装PL/Python的Docker镜像。

```
plcontainer image-add -f /home/gpadmin/plcontainer-python-images-1.0.0.tar.gz
```

该实用程序在MPP数据库主机上安装Docker镜像时显示进度信息。

使用plcontainer image-list命令在本地主机上显示已安装的Docker镜像。

此命令将信息添加到PL/Container配置文件，以便PL/Container可以访问Docker镜像以创建Docker容器。

```
plcontainer runtime-add -r plc_py -i pivotaldata/plcontainer_python_shared:devel -l python
```

该实用程序在更新MPP数据库实例上的PL/Container配置文件时显示进度信息。

您可以使用plcontainer runtime-show -r plc_py命令查看PL/Container配置信息。您可以使用plcontainer runtime-edit命令查看PL/Container配置XML文件。

卸载PL/Container

要卸载PL/Container，请删除Docker容器和镜像，然后从MPP数据库中删除PL/Container支持。

删除对PL/Container的支持后，您在数据库中创建的plcontainer用户定义函数将不再起作用。

卸载Docker容器和镜像

在MPP数据库主机上，卸载不再需要的Docker容器和镜像。

`plcontainer image-list`命令列出了安装在本地MPP数据库主机上的Docker镜像。

`plcontainer image-delete`命令从所有MPP数据库主机中删除指定的Docker镜像。

如果容器不是由PL/Container管理，则某些Docker容器可能存在于主机上。您可能需要使用Docker命令删除容器。这些docker命令管理本地主机上的Docker容器和镜像。

- 命令`docker ps -a`列出主机上的所有容器。命令`docker stop`关闭容器。
- 命令`docker images`列出主机上的镜像。
- 命令`docker rmi`删除镜像。
- 命令`docker rm`删除容器。

删除数据库的PL/Container支持

对于不再需要PL/Container的数据库，请删除对PL/Container的支持。

PL/Container 1.1及更高版本

对于PL/Container 1.1及更高版本，请从数据库中删除扩展。此`psql`命令运行`DROP EXTENSION`命令以删除数据库`mytest`中的PL/Container。

```
psql -d mytest -c 'DROP EXTENSION plcontainer cascade;'
```

该命令将删除`plcontainer`扩展并从数据库中删除PL/Container特定的函数和视

图。

PL/Container 1.0

以gpadmin用户身份运行plcontainer_uninstall.sql脚本。例如，此命令删除mytest数据库中的plcontainer语言。

```
psql -d mytest -f $GPHOME/share/postgresql/plcontainer/plcontainer_uninstall.sql
```

该脚本使用CASCADE删除plcontainer语言，以从数据库中删除PL/Container特定的函数和视图。

使用PL/Container函数

在MPP数据库系统的数据库中启用PL/Container时，语言plcontainer将在数据库中注册。当您将plcontainer指定为UDF定义中的语言时，可以使用PL/Container Docker镜像支持的过程语言创建和运行用户定义的函数。

使用PL/Container的UDF定义必须包含这些项。

- UDF的第一行必须是# container: *ID*
- LANGUAGE属性必须是plcontainer

*ID*是PL/Container用于标识Docker镜像的名称。当MPP数据库在主机上执行UDF时，主机上的Docker镜像用于启动运行UDF的Docker容器。在XML配置文件plcontainer_configuration.xml中，有一个runtime XML元素，它包含指定Docker容器启动信息的相应id XML元素。有关PL/Container如何将*ID*映射到Docker镜像的信息，请参阅[配置PL/Container](#)。有关示例UDF定义，请参阅[示例](#)。

PL/Container配置文件仅在运行PL/Container函数的每个MPP数据库会话中第一次调用PL/Container函数时读取。您可以通过在会话期间对视图plcontainer_refresh_config执行SELECT命令来强制重新读取配置文件。例如，此SELECT命令强制读取配置文件。

```
SELECT * FROM plcontainer_refresh_config;
```

运行该命令会执行PL/Container函数，该函数会更新master实例和segment实例上的配置并返回刷新的状态。

```
gp_segment_id | plcontainer_refresh_local_config
-----+-----
          1 | ok
          0 | ok
         -1 | ok
(3 rows)
```

此外，您可以通过在视图plcontainer_show_config上执行SELECT命令来显示会话中的所有配置。例如，此SELECT命令返回PL/Container配置。

```
SELECT * FROM plcontainer_show_config;
```

运行该命令将执行PL/Container函数，该函数显示master实例和segment实例的配置信息。这是视图输出的开始和结束的示例。

```
INFO: plcontainer: Container 'plc_py_test' configuration
INFO: plcontainer: image = 'pivotaldata/plcontainer_python_shared:devel'
INFO: plcontainer: memory_mb = '1024'
INFO: plcontainer: use container network = 'no'
INFO: plcontainer: use container logging = 'no'
INFO: plcontainer: shared directory from host '/usr/local/mpp-db/./bin/plcontainer_clients' to
INFO: plcontainer: access = readonly
```

...

```
INFO: plcontainer: Container 'plc_r_example' configuration (seg0 slice3 192.168.180.45:40000)
INFO: plcontainer: image = 'pivotaldata/plcontainer_r_without_clients:0.2' (seg0 slice3 192.168.180.45:40000)
INFO: plcontainer: memory_mb = '1024' (seg0 slice3 192.168.180.45:40000 pid=3304)
INFO: plcontainer: use container network = 'no' (seg0 slice3 192.168.180.45:40000 pid=3304)
INFO: plcontainer: use container logging = 'yes' (seg0 slice3 192.168.180.45:40000 pid=3304)
INFO: plcontainer: shared directory from host '/usr/local/mpp-db/bin/plcontainer_clients' to container
INFO: plcontainer: access = readonly (seg0 slice3 192.168.180.45:40000 pid=3304)
gp_segment_id | plcontainer_show_local_config
```

```
-----+-----
```

```
0 | ok
```

```
-1 | ok
```

```
1 | ok
```

PL/Container函数`plcontainer_containers_summary()`显示有关当前运行的Docker容器的信息。

```
SELECT * FROM plcontainer_containers_summary();
```

如果普通（非超级用户）MPP数据库用户运行该函数，则该函数仅显示用户创建的容器的信息。如果MPP数据库超级用户运行该函数，则会显示MPP数据库用户创建的所有容器的信息。这是2个容器运行时的示例输出。

SEGMENT_ID	CONTAINER_ID	UP_TIME	OWNER
1	693a6cb691f1d2881ec0160a44dae2547a0d5b799875d4ec106c09c97da422ea	Up 8 s	
1	bc9a0c04019c266f6d8269ffe35769d118bfb96ec634549b2b1bd2401ea20158	Up 2 m	

(2 rows)

示例

示例的# container行中的值plc_python_shared和plc_r_shared是plcontainer_config.xml文件中定义 id XML元素。 id元素映射到指定要启动的 Docker 镜像的 image 元素。 如果您使用不同的 ID 配置 PL/Container，请更改# container 行的值。 有关配置 PL/Container 和查看配置设置的信息，请参阅[配置 PL/Container](#)。

这是使用 plc_python_shared 容器运行的 PL/Python 函数的示例：

```
CREATE OR REPLACE FUNCTION pylog100() RETURNS double precision AS $$
# container: plc_python_shared
import math
return math.log10(100)
$$ LANGUAGE plcontainer;
```

这是使用 plc_r_shared 容器的类似函数的示例：

```
CREATE OR REPLACE FUNCTION rlog100() RETURNS text AS $$
# container: plc_r_shared
return(log10(100))
$$ LANGUAGE plcontainer;
```

如果 UDF 中的# container 行指定了不在 PL/Container 配置文件中的 ID，则当您尝试执行 UDF 时，MPP 数据库会返回错误。

关于PL/Container运行PL/Python

在Python语言容器中，实现了模块plpy。该模块包含以下方法：

- `plpy.execute(stmt)` - 执行查询字符串`stmt`并将查询结果返回到字典对象列表中。为了能够访问结果字段，请确保查询返回命名字段。
- `plpy.prepare(stmt[, argtypes])` - 准备查询的执行计划。如果在查询中有参数引用，则使用查询字符串和参数类型列表调用它。
- `plpy.execute(plan[, argtypes])` - 执行准备好的计划。
- `plpy.debug(msg)` - 将DEBUG2消息发送到MPP数据库日志。
- `plpy.log(msg)` - 将LOG消息发送到MPP数据库日志。
- `plpy.info(msg)` - 将INFO消息发送到MPP数据库日志。
- `plpy.notice(msg)` - 将NOTICE消息发送到MPP数据库日志。
- `plpy.warning(msg)` - 将WARNING消息发送到MPP数据库日志。
- `plpy.error(msg)` - 将ERROR消息发送到MPP数据库日志。在MPP数据库中引发的ERROR消息导致查询停止并回滚事务。
- `plpy.fatal(msg)` - 将FATAL消息发送到MPP数据库日志。FATAL消息导致关闭MPP数据库会话并回滚事务。
- `plpy.subtransaction()` - 在显式子事务中管理`plpy.execute`调用。有关`plpy.subtransaction()`的其他信息，请参阅PostgreSQL文档中的[显式子事务](#)。

如果在嵌套的Python函数调用中引发级别ERROR或FATAL的错误，则该消息包括封闭函数的列表。

Python语言容器支持这些在构造ad-hoc查询时很有用的字符串引用函数。

- `plpy.quote_literal(string)` - 返回引用的字符串，用作SQL语句字符串中的字符串文字。内嵌的单引号和反斜杠正常加倍。`quote_literal()`在null输入（空输入）上返回null。如果参数可能为null，则`quote_nullable()`可能更合适。
-

- `plpy.quote_nullable(string)` - 返回引用的字符串，用作SQL语句字符串中的字符串文字。如果参数为null，则返回NULL。内嵌的单引号和反斜杠正常加倍。
- `plpy.quote_ident(string)` - 返回引用的字符串，用作SQL语句字符串中的标识符。仅在必要时添加引号（例如，如果字符串包含非标识符字符或将进行大小写折叠）。内嵌的引号正常加倍。

从PL/Python函数返回文本时，PL/Container将Python unicode对象转换为数据库编码中的文本。如果无法执行转换，则返回错误。

PL/Container不支持此MPP数据库PL/Python特性：

- 多维数组。

此外，Python模块有两个全局字典对象，它们在函数调用之间保留数据。它们被命名为GD和SD。GD用于在同一容器内运行的所有函数之间共享数据，而SD用于在每个单独函数的多个调用之间共享数据。请注意，当容器进程位于segment或master上时，只能在同一会话中访问数据。请注意，对于空闲会话，MPP数据库会终止segment进程，这意味着相关容器将被关闭，GD和SD中的数据将丢失。

有关PL/Python的信息，参考[MPP PL/Python语言扩展](#)。

有关plpy方法的信息，参考<https://www.postgresql.org/docs/8.4/plpython-database.htm>。

关于PL/Container运行PL/R

在R语言容器中，实现了模块pg.spi。该模块包含以下方法：

- `pg.spi.exec(stmt)` - 执行查询字符串stmt并在R data.frame中返回查询结果。为了能够访问结果字段，请确保您的查询返回命名字段。
 - `pg.spi.prepare(stmt[, argtypes])` - 准备查询的执行计划。如果在查询中有参数
-

引用，则使用查询字符串和参数类型列表调用它。

- `pg.spi.execp(plan[, argtypes])` - 执行准备好的计划。
- `pg.spi.debug(msg)` - 将DEBUG2消息发送到MPP数据库日志。
- `pg.spi.log(msg)` - 将LOG消息发送到MPP数据库日志。
- `pg.spi.info(msg)` - 将INFO消息发送到MPP数据库日志。
- `pg.spi.notice(msg)` - 将NOTICE消息发送到MPP数据库日志。
- `pg.spi.warning(msg)` - 将WARNING消息发送到MPP数据库日志。
- `pg.spi.error(msg)` - 将ERROR消息发送到MPP数据库日志。在MPP数据库中引发的ERROR消息导致查询停止并回滚事务。
- `pg.spi.fatal(msg)` - 将FATAL消息发送到MPP数据库日志。FATAL消息导致关闭MPP数据库会话并回滚事务。

PL/Container不支持此PL/R特性：

- 多维数组。

关于PL/R的信息，参考[MPP PL/R 语言扩展](#)。

关于pg.spi方法的信息，参考<http://www.joeconway.com/plr/doc/plr-spi-rsupport-funcs-normal.html>

配置PL/Container

MPP数据库实用程序plcontainer管理MPP数据库系统中的PL/Container配置文件。该实用程序可确保配置文件在MPP数据库master和segment实例之间保持一致。

Warning: 在不使用该实用程序的情况下修改segment实例上的配置文件，可能会在不同的MPP数据库segment上创建导致不可预期行为的不同的、不兼容的配置。

使用该实用程序进行的配置更改将应用于所有MPP数据库segment上的XML文件。但是，当前运行的会话的PL/Container配置使用会话启动期间存在的配置。要在正在运行的会话中更新PL/Container配置，请在会话中执行此命令。

```
SELECT * FROM plcontainer_refresh_config;
```

运行该命令会执行PL/Container函数，该函数会更新master和segment实例上的会话配置。

plcontainer实用程序

plcontainer实用程序安装Docker镜像并管理PL/Container配置。该实用程序包含两组命令。

- image-*命令管理MPP数据库系统主机上的Docker镜像。
- runtime-*命令管理MPP数据库实例上的PL/Container配置文件。您可以将Docker镜像信息添加到PL/Container配置文件，包括镜像名称，位置和共享文件夹信息。您还可以编辑配置文件。

要将PL/Container配置为使用Docker镜像，请在所有MPP数据库主机上安装Docker镜像，然后将配置信息添加到PL/Container配置中。

PL/Container配置值（例如镜像名称，runtime ID，参数值和名称）区分大小写。

plcontainer语法

```
plcontainer [command] [-h | --help] [--verbose]
```

其中`command`是下面的其中一个。

```
image-add {{-f | --file} image_file} | {{-u | --URL} image_URL}
image-delete {-i | --image} image_name
image-list

runtime-add {-r | --runtime} runtime_id
  {-i | --image} image_name {-l | --language} {python | r}
  [{-v | --volume} shared_volume [{-v | --volume} shared_volume...]]
  [{-s | --setting} param=value [{-s | --setting} param=value ...]]
runtime-replace {-r | --runtime} runtime_id
  {-i | --image} image_name -l {r | python}
  [{-v | --volume} shared_volume [{-v | --volume} shared_volume...]]
  [{-s | --setting} param=value [{-s | --setting} param=value ...]]
runtime-show {-r | --runtime} runtime_id
runtime-delete {-r | --runtime} runtime_id
runtime-edit [{-e | --editor} editor]
runtime-backup {-f | --file} config_file
runtime-restore {-f | --file} config_file
runtime-verify
```

plcontainer命令和选项

image-add *location*

在MPP数据库主机上安装Docker镜像。指定主机上Docker镜像文件的位置或Docker镜像的URL。这些是受支持的位置选项。

- **{-f | --file}** *image_file*指定包含Docker镜像的主机上的tar存档文件。此示例指向gpadmin主目录中的镜像文件
/home/gpadmin/test_image.tar.gz
- **{-u | --URL}** *image_URL*指定Docker存储库和镜像的URL。此示例URL指向本地Docker存储库 192.168.0.1:5000/images/mytest_plc_r:devel

安装Docker镜像后，使用[runtime-add](#)命令配置PL/Container以使用Docker镜像。

```
image-delete {-i | --image} image_name
```

从所有MPP数据库主机中删除已安装的Docker镜像。例如，指定包括tag的完整Docker镜像名称 `pivotaldata/plcontainer_python_shared:1.0.0`

image-list

列出主机上安装的Docker镜像。该命令仅列出本地主机上的镜像，而不是远程主机。该命令列出了所有已安装的Docker镜像，包括使用Docker命令安装的镜像。

runtime-add *options*

将配置信息添加到所有MPP数据库主机上的PL/Container配置文件中。如果指定的*runtime_id*存在，则该实用程序将返回错误，并且不会添加配置信息。

关于PL/Container配置信息，参考[PL/Container配置文件](#)。

这些是支持的选项：

{-i | --image} *docker-image*

必须。指定MPP数据库主机上安装的完整Docker镜像名称，包括tag。例如 `pivotaldata/plcontainer_python:1.0.0`。

如果未安装指定的Docker镜像，该实用程序将返回警告。

`plcontainer image-list`命令显示已安装的镜像信息，包括名称和tag（存储库和tag列）。

{-l | --language} *python | r*

必须。指定PL/Container语言类型，支持的值是python（PL/Python）和r（PL/R）。为runtime添加配置信息时，该实用程序会根据您指定的语言向配置添加启动命令。

Python语言的启动命令。

```
/clientdir/pyclient.sh
```

R语言的启动命令。

```
/clientdir/rclient.sh
```

{-r | --runtime} *runtime_id*

需要。添加runtime ID。在PL/Container配置文件中添加runtime元素

时，这是PL/Container配置文件中id元素的值。最大长度为63字节。

在MPP数据库UDF# container行指定名称。参考[示例](#)。

{-s | --setting} param=value

可选的。指定要添加到runtime配置信息的设置。您可以多次指定此选项。该设置适用于runtime_id指定的runtime配置。该参数是PL/Container配置文件中[settings](#)元素的XML属性。这些是有效的参数。

- **cpu_share** - 在runtime配置中为每个容器设置CPU限制。默认值为1024。该值是CPU使用率与其他容器的相对权重。
- **memory_mb** - 在runtime配置中为每个容器设置内存限制。默认值为1024。该值是一个整数，指定以MB为单位的内存量。
- **resource_group_id** - 将指定的资源组分配给runtime配置。资源组限制共享此runtime配置的所有容器的CPU和内存资源总使用量。您必须指定资源组的groupid。有关管理PL/Container资源的信息，请参阅[关于PL/Container资源管理](#)。
- **roles** - 指定允许为runtime配置运行容器的MPP数据库角色。您可以指定单个角色名称或逗号分隔的角色名称列表。默认值没有限制。
- **use_container_logging** - 启用或禁用容器的Docker日志记录。值为yes（启用日志记录）或no（禁用日志记录，默认值）。MPP数据库服务器配置参数[log_min_messages](#)控制日志级别。默认日志级别为warning。有关PL/Container日志信息的信息，请参阅[备注](#)。

{-v | --volume} shared-volume

可选的。指定要绑定装载的Docker卷。您可以多次指定此选项以定义多个卷。

共享卷的格式：*host-dir.container-dir.[rw|ro]*。该信息作为属性存储在PL/Container配置文件中的runtime元素的shared_directory元素中。

- **host-dir** - 主机系统上目录的绝对路径。MPP数据库管理员用户

(gpadmin) 必须具有对目录的适当访问权限。

- *container-dir* - Docker容器中目录的绝对路径。
- [rw|ro] - 从容器访问主机目录的只读或读写权限。

为新runtime添加配置信息时，该实用程序会添加此只读共享卷信息。

```
mpp-home/bin/plcontainer_clients:/clientdir:ro
```

如果需要，您可以指定其他共享目录。如果指定的*container-dir*与实用程序添加的相同，或者如果指定具有相同*container-dir*的多个共享卷，则该实用程序将返回错误。

Warning: 允许对主机目录进行读写访问需要特殊注意事项。

- 指定对主机目录的读写访问权限时，请确保指定的主机目录具有正确的权限。
- 运行PL/Container用户定义的函数时，主机上运行的多个并发Docker容器可能会更改主机目录中的数据。确保这些函数支持对主机目录中的数据进行多个并发访问。

`runtime-backup {-f | --file} config_file`

将PL/Container配置文件复制到本地主机上的指定文件。

`runtime-delete {-r | --runtime} runtime_id`

删除所有MPP数据库实例上PL/Container配置文件中的runtime配置信息。如果文件中不存在指定的*runtime_id*，则实用程序将返回一条消息。

`runtime-edit [{-e | --editor} editor]`

使用指定的编辑器编辑XML文件plcontainer_configuration.xml。默认编辑器是vi。

保存文件会更新所有MPP数据库主机上的配置文件。如果更新的文件中存在错误，该实用程序将返回错误并且不会更新该文件。

runtime-replace *options*

替换所有MPP数据库实例上的PL/Container配置文件中的runtime配置信息。如果`runtime_id`不存在，则将信息添加到配置文件中。该实用程序将启动命令和共享目录添加到配置中。

有关命令选项和添加到配置的信息，请参阅[runtime-add](#)。

runtime-restore **{-f | --file}** *config_file*

使用本地主机上指定文件的信息替换所有MPP数据库实例上的PL/Container配置文件`plcontainer_configuration.xml`中的信息。

runtime-show **[-r | --runtime]** *runtime_id*

显示格式化的PL/Container runtime配置信息。如果未指定`runtime_id`，则显示所有runtime标识的配置。

runtime-verify

使用master服务器上的配置信息检查MPP数据库实例上的PL/Container配置信息。如果该实用程序发现不一致，系统将提示您使用本地副本替换远程副本。该实用程序还执行XML验证。

-h | --help

显示帮助文本。如果在没有命令的情况下指定，则显示所有plcontainer命令的帮助。如果使用命令指定，则显示该命令的帮助。

--verbose

为命令启用详细日志记录。

示例

这些是管理PL/Container的常用命令的示例：

- 在所有MPP数据库主机上安装Docker镜像。此示例从文件加载Docker镜像。该实用程序在命令行上显示进度信息，因为该实用程序在所有主机上安装Docker镜像。

```
plcontainer image-add -f plc_newr.tar.gz
```

安装Docker镜像后，您可以在PL/Container配置文件中添加或更新runtime条目，以便PL/Container访问Docker镜像以启动Docker容器。

- 将容器条目添加到PL/Container配置文件中。此示例添加PL/R runtime的配置信息，并指定内存和日志记录的共享卷和设置。

```
plcontainer runtime-add -r runtime2 -i test_image2:0.1 -l r \  
-v /host_dir2/shared2:/container_dir2/shared2:ro \  
-s memory_mb=512 -s use_container_logging=yes
```

该实用程序在命令行上显示进度信息，因为它将runtime配置添加到配置文件，并将更新的配置分发到所有实例。

- 在配置文件中显示给定runtime标识的特定runtime

```
plcontainer runtime-show -r plc_python_shared
```

该实用程序显示与此输出类似的配置信息。

```
PL/Container Runtime Configuration:
```

```
-----  
Runtime ID: plc_python_shared  
Linked Docker Image: test1:latest  
Runtime Setting(s):  
Shared Directory:  
---- Shared Directory From HOST '/usr/local/mpp-db/bin/plcontainer_clients' to Container '/c  
---- Shared Directory From HOST '/home/gpadmin/share/' to Container '/opt/share', access  
-----
```

- 在您选择的交互式编辑器中编辑配置。此示例使用vim编辑器编辑配置文件。


```
plcontainer runtime-edit -e vim
```

保存文件时，该实用程序会在将文件分发到MPP数据库主机时在命令行上显示进度信息。

- 将当前的PL/Container配置保存到文件中。此示例将文件保存到本地文件/home/gpadmin/saved_plc_config.xml

```
plcontainer runtime-backup -f /home/gpadmin/saved_plc_config.xml
```

- 使用XML文件覆盖PL/Container配置文件。此示例使用/home/gpadmin目录中文件中的信息替换配置文件中的信息。

```
plcontainer runtime-restore -f /home/gpadmin/new_plcontainer_configuration.xml
```

该实用程序在将更新的文件分发到MPP数据库实例时，在命令行上显示进度信息。

PL/Container配置文件

PL/Container在所有MPP数据库segment的数据目录中维护配置文件plcontainer_configuration.xml。PL/Container配置文件是XML文件。在XML文件中，根元素configuration包含一个或多个runtime元素。您可以在PL/Container函数定义的# container:行中指定runtime元素的id。

在XML文件中，名称（如元素和属性名称）和值区分大小写。

这是一个示例文件。

```
<?xml version="1.0" ?>
<configuration>
  <runtime>
    <id>plc_python_example1</id>
    <image>pivotaldata/plcontainer_python_with_clients:0.1</image>
    <command>./pyclient</command>
  </runtime>
  <runtime>
    <id>plc_python_example2</id>
    <image>pivotaldata/plcontainer_python_without_clients:0.1</image>
    <command>/clientdir/pyclient.sh</command>
    <shared_directory access="ro" container="/clientdir" host="/usr/local/mpp-db/bin/plcontai
    <setting memory_mb="512"/>
    <setting use_container_logging="yes"/>
    <setting cpu_share="1024"/>
    <setting resource_group_id="pl_container.html16391"/>
  </runtime>
  <runtime>
    <id>plc_r_example</id>
    <image>pivotaldata/plcontainer_r_without_clients:0.2</image>
    <command>/clientdir/rclient.sh</command>
    <shared_directory access="ro" container="/clientdir" host="/usr/local/mpp-db/bin/plcontai
    <setting use_container_logging="yes"/>
    <setting roles="gpadmin,user1"/>
  </runtime>
  <runtime>
  </runtime>
</configuration>
```

这些是PL/Container配置文件中的XML元素和属性。

configuration

XML文件的根元素。

runtime

系统中可用的每个特定容器的一个元素。 这些是configuration元素的子元素。

id

必须。该值用于从PL/Container用户定义的函数引用Docker容器。

id值在配置中必须是唯一的。id必须以字符或数字（a-z，A-Z或0-9）开头，并且可以包含字符_（下划线），.（期间），或-（破折号）。最大长度为63字节。

id指定当PL/Container创建Docker容器以执行用户定义的函数时要使用的Docker镜像。

image

必须。该值是完整的Docker镜像名称，包括镜像tag。与在Docker中启动此容器指定它们的方式相同。配置允许有许多容器对象引用相同的镜像名称，这种方式在Docker中它们将由相同的容器表示。

例如，您可能有两个runtime元素，具有不同的id元素，plc_python_128和plc_python_256，两者都引用Docker镜像pivotaldata/plcontainer_python:1.0.0。第一个runtime指定128MB RAM限制，第二个指定由setting元素的memory_mb属性指定的256MB限制。

command

必须。该值是要在容器内部运行以在容器内启动客户端进程的命令。在创建runtime元素时，plcontainer实用程序会根据语言（-l 选项）添加command元素。
python语言的command元素。

```
<command>/clientdir/pyclient.sh</command>
```

R语言的command元素。

```
<command>/clientdir/rclient.sh</command>
```

只有在构建自定义容器并希望在容器启动之前实现一些其他初始化逻辑时，才应修改该值。

Note: 无法使用plcontainer实用程序设置此元素。您可以使

用plcontainer runtime-edit命令更新配置文件。

shared_directory

可选的。此元素为具有访问信息的容器指定共享Docker共享卷。允许多个shared_directory元素。每个shared_directory元素指定一个共享卷。shared_directory元素的XML属性：

- host - 主机系统上的目录位置。
- container - 容器内的目录位置。
- access - 主机目录的访问级别，可以是ro（只读）或rw（读写）。

创建runtime元素时，plcontainer实用程序会添加shared_directory元素。

```
<shared_directory access="ro" container="/clientdir" host="/usr/local/mpp-db/bin/
```

对于每个runtime元素，shared_directory元素的container属性必须是唯一的。例如，runtime元素不能有两个带有属性container="/clientdir"的shared_directory元素。

Warning: 允许对主机目录进行读写访问需要特别考虑。

- 指定对主机目录的读写访问权限时，请确保指定的主机目录具有正确的权限。
- 运行PL/Container用户定义的函数时，主机上运行的多个并发Docker容器可能会更改主机目录中的数据。确保这些函数支持对主机目录中的数据进行多个并发访问。

settings

可选的。此元素指定Docker容器配置信息。每个setting元素都包含一个属性。element属性指定日志记录，内存或网络信息。例如，此元素启用日志记录。

```
<setting use_container_logging="yes"/>
```

这些是合法属性。

`cpu_share`

可选的。在runtime中指定每个PL/Container容器的CPU使用率。元素的值是正整数。默认值为1024。该值是CPU使用率与其他容器的相对权重。

例如，与具有默认值1024的容器相比，`cpu_share`为2048的容器被分配了两倍的CPU切片时间。

`memory_mb="size"`

可选的。该值指定允许每个容器使用的内存量（MB）。每个容器都以这个RAM量和两倍的交换空间开始。容器内存消耗受主机系统cgroups配置的限制，这意味着在内存过量使用的情况下，容器由系统终止。

`resource_group_id="pl_container.htmlrg_groupid"`

可选的。该值指定要分配给PL/Container runtime的资源组的groupid。资源组限制共享此runtime配置的所有正在运行的容器的CPU和内存资源总使用量。您必须指定资源组的groupid。如果未将资源组分配给PL/Container runtime配置，则其容器实例仅受系统资源的限制。有关管理PL/Container资源的信息，请参阅[关于PL/Container资源管理](#)。

`roles="list_of_roles"`

可选的。该值是MPP数据库角色名称或以逗号分隔的角色列表。PL/Container运行仅对列出的角色使用PL/Container runtime配置的容器。如果未指定该属性，则任何MPP数据库角色都可以运行此容器runtime配置的实例。例如，您创建一个指定plcontainer语言的UDF，并标识具有roles属性集的#container: runtime配置。当角色（用户）运行UDF时，PL/Container会检查角色列表，并仅在角色位于列表中时运行容器。

`use_container_logging="{yes | no}"`

可选的。启用或禁用容器的Docker日志记录。属性值yes启用日志记录。属性值no禁用日志记录（默认值）。

MPP数据库服务器配置参数[log_min_messages](#)控制PL/Container日志级别。默认日志级别为warning。有关PL/Container日志信息的信息，请参阅[备注](#)。

默认情况下，PL/Container日志信息将发送到系统服务。在Red Hat 7或CentOS 7系统上，日志信息将发送到journald服务。在Red Hat 6或CentOS 6系统上，日志将发送到syslogd服务。

更新PL/Container配置

您可以使用plcontainer runtime-add命令将runtime元素添加到PL/Container配置文件中。命令选项指定runtime ID，Docker镜像和语言等信息。您可以使用plcontainer runtime-replace命令更新现有的runtime元素。该实用程序更新master和所有segment实例上的配置文件。

PL/Container配置文件可以包含多个runtime元素，这些元素引用由XML元素image指定的相同Docker镜像。在示例配置文件中，runtime元素包含名为plc_python_128和plc_python_256的id元素，两者都引用Docker容器pivotaldata/plcontainer_python:1.0.0。第一个runtime元素定义为128MB RAM限制，第二个定义为256MB RAM限制。

```

<configuration>
  <runtime>
    <id>plc_python_128</id>
    <image>pivotaldata/plcontainer_python:1.0.0</image>
    <command>./client</command>
    <shared_directory access="ro" container="/clientdir" host="/usr/local/gpdb/bin/plcontainer_c
    <setting memory_mb="128"/>
  </runtime>
  <runtime>
    <id>plc_python_256</id>
    <image>pivotaldata/plcontainer_python:1.0.0</image>
    <command>./client</command>
    <shared_directory access="ro" container="/clientdir" host="/usr/local/gpdb/bin/plcontainer_c
    <setting memory_mb="256"/>
    <setting resource_group_id="pl_container.html16391"/>
  </runtime>
</configuration>

```

备注

- PL/Container不支持MPP数据库域对象。
- PL/Container在所有MPP数据库segment实例的数据目录中维护配置文件plcontainer_configuration.xml：master，standby master，primary和mirror。此查询列出了MPP数据库系统数据目录：

```
SELECT hostname, datadir FROM gp_segment_configuration;
```

PL/Container配置文件示例位于\$GPHOME/share/postgresql/plcontainer中。

- 当MPP数据库执行PL/Container UDF时，Query Executer（QE）进程启动Docker容器并根据需要重用它们。经过一定的空闲时间后，QE进程退出并销毁其Docker容器。您可以使用MPP数据库服务器配置参数[gp_vmem_idle_resource_timeout](#)控制空闲时间。控制空闲时间可能有助于Docker容器重用，并避免创建和启动Docker容器的开销。

Warning: 更改gp_vmem_idle_resource_timeout值可能会因资源问题而影响性能。该参数还控制释放除Docker容器之外的MPP数据库资源。

- 如果PL/Container Docker容器超过允许的最大内存量，则会终止该容器并显示内存不足警告。在配置了Docker 1.7.1版的Red Hat 6或CentOS 6系统上，如果PL/Container Docker容器主程序（PID 1）终止，也会显示内存不足警告。
- 在某些情况下，当PL/Container在高并发环境中运行时，Docker守护程序会挂起并指示内存不足的日志条目。即使系统似乎有足够的可用内存，也可能发生这种情况。

这个问题似乎是由两个因素共同触发的，即PL/Container使用的Go语言（golang）运行时的激进虚拟内存需求，以及overcommit_memory的MPP数据库Linux服务器内核参数设置。该参数设置为2，不允许内存过量使用。可能有用的解决方法是增加交换空间量并增加Linux服务器内核参数overcommit_ratio。如果更改后问题仍然存在，则可能存在内存不足问题。您应检查系统上的可用内存，并在需要时添加更多RAM。您还可以降低群集负载。

- PL/Container不限制Docker基本设备大小，Docker容器的大小。在某些情况下，Docker守护程序控制基本设备大小。例如，如果Docker存储驱动程序是devicemapper，则Docker守护程序--storage-opt选项标志dm.basesize控制基本设备大小。devicemapper的默认基本设备大小为10GB。Docker命令docker info显示Docker系统信息，包括存储驱动程序。基本设备大小显示在Docker 1.12及更高版本中。有关Docker存储驱动程序的信息，请参阅Docker信息[守护存储驱动程序](#)。

设置Docker基本设备大小时，必须在所有MPP数据库主机上设置大小。

- 启用PL/Container日志记录后，可以使用MPP数据库服务器配置参数[log_min_messages](#)设置日志级别。默认日志级别为warning。该参数控制PL/Container日志级别，还控制MPP数据库日志级别。
 - 使用setting属性use_container_logging为每个runtime ID启用或禁用PL/Container日志记录。默认为无记录。
 - PL/Container日志信息是来自Docker容器中运行的UDF的信息。默认情况下，PL/Container日志信息将发送到系统服务。在Red Hat 7或CentOS 7

系统上，日志信息将发送到journald服务。在Red Hat 6或CentOS 6系统上，日志信息将发送到syslogd服务。PL/Container日志信息将发送到Docker容器运行的主机的日志文件中。

- MPP数据库日志信息将发送到MPP数据库master上的日志文件。

在测试或排除PL/Container UDF故障时，可以使用SET命令更改MPP数据库日志级别。您可以在运行PL/Container UDF之前在会话中设置参数。此示例将日志级别设置为debug1。

```
SET log_min_messages='debug1';
```

Note: 参数log_min_messages控制MPP数据库和PL/Container日志记录，即使PL/Container UDF未运行，增加日志级别也可能影响MPP数据库性能。

安装Docker

要使用PL/Container，必须在所有MPP数据库主机系统上安装Docker。这些说明显示了如何在CentOS 7上设置Docker服务。在RHEL 7上安装是一个类似的过程。

在执行Docker安装之前，请确保满足这些要求。

- 可以访问CentOS extras存储库。
- 用户具有sudo权限或是root权限。

另请参阅CentOS的Docker站点安装说

明<https://docs.docker.com/engine/installation/linux/centos/>。有关Docker命令的列表，请参阅Docker引擎运行参

考<https://docs.docker.com/engine/reference/run/>。

在CentOS 7上安装Docker

这些步骤安装docker软件包并以具有sudo权限的用户身份启动docker服务。

1. 安装Docker所需的依赖项

```
sudo yum install -y yum-utils device-mapper-persistent-data lvm2
```

2. 添加Docker仓库

```
sudo yum-config-manager --add-repo https://download.docker.com/linux/centos/docker-ce.r
```

3. 更新yum cache

```
sudo yum makecache fast
```

4. 安装Docker

```
sudo yum -y install docker-ce
```

5. 启动Docker守护进程。

```
sudo systemctl start docker
```

6. 要授予对Docker守护程序和docker命令的访问权限，请将MPP数据库管理员（gpadmin）分配给组docker。

```
sudo usermod -aG docker gpadmin
```

7. 退出会话并再次登录以更新权限。

8. 运行Docker命令以测试Docker安装。 此命令列出当前运行的Docker容器。

```
docker ps
```

此命令将Docker配置为在主机系统启动时启动。

```
sudo systemctl start docker.service
```

在所有MPP数据库主机上安装Docker后，重新启动MPP数据库系统以允许MPP数据库访问Docker。

```
gpstop -ra
```

在CentOS 6上安装Docker

这些步骤安装Docker软件包并以具有sudo权限的用户身份启动docker服务。

1. 安装EPEL包

```
sudo yum -y install epel-release
```

2. 安装Docker

```
sudo yum -y install docker-io
```

3. 创建docker组

```
sudo groupadd docker
```

4. 启动Docker

```
sudo service docker start
```

5. 要授予对Docker守护程序和docker命令的访问权限，请将MPP数据库管理员（gpadmin）分配给组docker。

```
sudo usermod -aG docker gpadmin
```

6. 退出会话并再次登录以更新权限。

7. 运行Docker命令以测试Docker安装。此命令列出当前运行的Docker容器。

```
docker ps
```

此命令将Docker配置为在主机系统启动时启动。

```
sudo chkconfig docker on
```

在所有MPP数据库主机上安装Docker后，重新启动MPP数据库系统以允许MPP数据库访问Docker。

```
gpstop -ra
```

参考

Docker主页<https://www.docker.com/>

Docker命令行接口<https://docs.docker.com/engine/reference/commandline/cli/>

Dockerfile参考 <https://docs.docker.com/engine/reference/builder/>

在Linux系统上安装

Docker<https://docs.docker.com/engine/installation/linux/centos/>

通过systemd控制和配置Docker<https://docs.docker.com/engine/admin/systemd/>

MPP的PL/Java语言扩展

本节包含MPP数据库的PL/Java语言的概述。

- [关于PL/Java](#)
- [关于MPP数据库PL/Java](#)
- [编写PL/Java函数](#)
- [使用JDBC](#)
- [异常处理](#)
- [保存点](#)
- [日志](#)
- [安全](#)
- [一些PL/Java问题和解决方案](#)
- [示例](#)
- [参考](#)

Parent topic: [MPP数据库参考手册](#)

关于PL/Java

使用MPP数据库的PL/Java扩展，用户可以使用自己喜欢的Java IDE编写Java方法，并将包含这些方法的JAR文件安装到MPP数据库中。

MPP数据库的PL/Java包基于开源PL/Java 1.5.0。MPP数据库的PL/Java提供以下功能。

- 能够使用Java 8或Java 11执行PL/Java函数。
- 能够指定Java运行时。
- 在数据库中安装和维护Java代码的标准化实用程序（在SQL 2003提案之后设计）
- 参数和结果的标准化映射。支持复杂类型和集合。
- 使用MPP数据库内部SPI例程的嵌入式高性能JDBC驱动程序。
- 元数据支持JDBC驱动程序。 包括DatabaseMetaData和ResultSetMetaData。
- 能够从查询中返回ResultSet，作为逐行构建ResultSet的替代方法。
- 完全支持保存点和异常处理。
- 能够使用IN，INPUT和OUT参数。
- 两种独立的MPP数据库语言：
 - pljava, TRUSTED PL/Java language
 - pljavau, UNTRUSTED PL/Java language
- 当一个事务或者保存点提交或者回滚时，事务和保存点监听器能够被编译执行。
- 在所选平台上与GNU GCJ集成。

SQL中的一个函数将在Java类中指定一个静态方法。为了使函数能够执行，所指定的类必须能够通过MPP数据库服务器上的pljava_classpath配置参数来指定类路径。PL/Java扩展添加了一组有助于安装和维护java类的函数。类存储在普通的Java档案——JAR文件中。JAR文件可以选择性地包含部署描述符，该描

述符又包含在部署或取消部署JAR时要执行的SQL命令。这些函数是按照SQL 2003提出的标准进行设计的。

PL/Java实现了传递参数和返回值的标准化方法。使用标准JDBC ResultSet类传递复杂类型和集合。

PL/Java中包含JDBC驱动程序。此驱动程序调用MPP数据库内部SPI例程。驱动程序是必不可少的，因为函数通常将调用数据库以获取数据。当PL/Java函数提取数据时，它们必须使用与输入PL/Java执行上下文的主函数使用的相同的事务边界。

PL/Java针对性能进行了优化。Java虚拟机在与后端相同的进程中执行，以最小化调用开销。PL/Java的设计目的是为了使数据库本身能够实现Java的强大功能，以便数据库密集型业务逻辑可以尽可能靠近实际数据执行。

当后端和Java VM之间的桥梁被调用时，将使用标准Java本机接口（JNI）。

关于MPP数据库PL/Java

在标准PostgreSQL和MPP数据库中实现PL/Java有一些关键的区别。

函数

以下函数在MPP数据库中不被支持。在分布式的MPP数据库环境中，classpath的处理方式与PostgreSQL环境下不同。

- sqlj.install_jar
 - sqlj.replace_jar
 - sqlj.remove_jar
 - sqlj.get_classpath
-

- `sqlj.set_classpath`

MPP数据库使用`pljava_classpath`服务器配置参数代替`sqlj.set_classpath`函数。

服务器配置参数

以下服务器配置参数由PL/Java在MPP数据库中使用。 这些参数取代了标准PostgreSQL PL/Java实现中使用的`pljava.*`参数：

- `pljava_classpath`

冒号(:)分隔的包含任何PL/Java函数中使用的Java类的jar文件列表。 所有的jar文件必须安装在所有的MPP数据库主机的相同位置。 使用可信的PL/Java语言处理程序，jar文件路径必须相对于`$GPHOME/lib/postgresql/java/`目录。 使用不受信任的语言处理程序（javaU语言标记），路径可以相对于`$GPHOME/lib/postgresql/java/`或使用绝对路径。

服务器配置参数`pljava_classpath_insecure`控制服务器配置参

数`pljava_classpath`是否可以由用户设置， 无需MPP数据库超级用户权限。

当启用`pljava_classpath_insecure`时， 正在开发PL/Java函数的MPP数据库开发人员不必是数据库超级用户身份才能来更改`pljava_classpath`。

Warning: 启用`pljava_classpath_insecure`通过为非管理员数据库用户提供能够运行未经授权的Java方法暴露了安全风险。

- `pljava_statement_cache_size`

为准备语句设置最近使用（MRU）缓存的大小（KB）。

- `pljava_release_lingering_savepoints`

如果为TRUE，在函数退出后，长期持续的保存点将会释放。 如果为FALSE，它们将被回滚。

- `pljava_vmoptions`

定义MPP数据库Java VM的启动选项。

参阅MPP数据库参考手册有关MPP数据库服务器配置参数的信息。

启用PL/Java并安装JAR文件

以MPP数据库管理员gpadmin的身份执行以下步骤。

1. 通过执行CREATE EXTENSION命令注册语言，以在数据库中启用PL/Java。例如，此命令在testdb数据库中启用PL/Java：

```
$ psql -d testdb -c 'CREATE EXTENSION pljava;'
```

Note: 不推荐使用PL/Java install.sql脚本（在以前的版本中用于注册该语言）。

2. 将Java归档（JAR文件）复制到所有MPP数据库主机上的同一目录。本示例使用MPP数据库gpscp程序将文件myclasses.jar复制到目录\$GPHOME/lib/postgresql/java/:

```
$ gpscp -f gphosts_file myclasses.jar  
=:/usr/local/mpp-db/lib/postgresql/java/
```

文件gphosts_file包含一个MPP数据库主机的列表。

3. 设置pljava_classpath服务器配置参数在master的postgresql.conf文件中。对于此示例，参数值是冒号(:)分隔的JAR文件列表。例如：

```
$ gpconfig -c pljava_classpath -v 'examples.jar:myclasses.jar'
```

当用户使用gppkg实用程序安装PL/Java扩展包时，将安装examples.jar文件。

Note: 如果将JAR文件安装在除\$GPHOME/lib/postgresql/java/之外的目录中，则必须指定JAR文件的绝对路径。所有的MPP数据库主机上的每个JAR文件必须位于相同的位置。有关指定JAR文件位置的更多信息，参阅MPP数据库参考手册中的pljava_classpath服务器配置参数。

4. 重新加载postgresql.conf文件。

```
$ gpstop -u
```

5. (可选)MPP提供了一个包含可用于测试的示例PL/Java函数的examples.sql文件。运行此文件中的命令来创建测试函数 (它使用examples.jar中的Java类)。

```
$ psql -f $GPHOME/share/postgresql/pljava/examples.sql
```

编写PL/Java函数

有关使用PL/Java编写函数的信息。

- [SQL声明](#)
- [类型映射](#)
- [NULL处理](#)
- [复杂类型](#)
- [返回复杂类型](#)
- [返回复杂类型](#)
- [返回集的函数](#)
- [返回SETOF<标量类型>](#)
- [返回SETOF<复杂类型>](#)

SQL声明

一个Java函数被声明为该类的一个类名称和静态方法。 该类将用于为该函数声明的schema定义的classpath进行解析。 如果没有为该schema定义classpath，则使用public schema。 如果没有找到classpath，则使用系统类加载器解析该类。

可以声明以下函数来访问java.lang.System类上的静态方法getProperty：

```
CREATE FUNCTION getsysprop(VARCHAR)
  RETURNS VARCHAR
  AS 'java.lang.System.getProperty'
  LANGUAGE java;
```

运行以下命令返回Java user.home属性：

```
SELECT getsysprop('user.home');
```

类型映射

标量类型以简单的方式映射。 此表列出了当前的映射。

Table 1. PL/Java数据类型映射

PostgreSQL	Java
bool	boolean
char	byte
int2	short
int4	int
int8	long

PostgreSQL	Java
varchar	java.lang.String
text	java.lang.String
bytea	byte[]
date	java.sql.Date
time	java.sql.Time (stored value treated as local time)
timetz	java.sql.Time
timestamp	java.sql.Timestamp (stored value treated as local time)
timestampz	java.sql.Timestamp
complex	java.sql.ResultSet
setof complex	java.sql.ResultSet

所有其他类型都映射到java.lang.String，并将使用为各自类型注册的标准textin/textout例程。

NULL处理

映射到java基元的标量类型不能作为NULL值传递。要传递NULL值, 这些类型可以有一个替代映射。用户可以通过在方法引用中明确的指定该映射来启用映射。

```
CREATE FUNCTION truelyEvenOrNull(integer)
  RETURNS bool
  AS 'foo.fee.Fum.truelyEvenOrNull(java.lang.Integer)'
  LANGUAGE java;
```

Java代码将类似于：

```
package foo.fee;
public class Fum
{
    static boolean truelyEvenOrNull(Integer value)
    {
        return (value == null)
            ? true
            : (value.intValue() % 2) == 0;
    }
}
```

以下两个语句都产生true：

```
SELECT truelyEvenOrNull(NULL);
SELECT truelyEvenOrNull(4);
```

为了从Java方法返回NULL值, 可以使用与原始对象相对应的对象类型 (例如, 返回java.lang.Integer而不是int)。 PL/Java解析机制无论如何都会找到该方法。 由于Java对于具有相同名称的方法不能具有不同的返回类型, 因此不会引入任何歧义。

复杂类型

复杂类型将始终作为只读的java.sql.ResultSet传递, 只有一行。 ResultSet位于其行上, 因此不应该调用next()。 使用ResultSet的标准getter方法检索复杂类型的值。

例如:

```
CREATE TYPE complexTest
  AS(base integer, incbase integer, ctime timestamptz);
CREATE FUNCTION useComplexTest(complexTest)
  RETURNS VARCHAR
  AS 'foo.fee.Fum.useComplexTest'
  IMMUTABLE LANGUAGE java;
```

在java类Fum中，我们添加以下静态方法：

```
public static String useComplexTest(ResultSet complexTest)
  throws SQLException
{
  int base = complexTest.getInt(1);
  int incbase = complexTest.getInt(2);
  Timestamp ctime = complexTest.getTimestamp(3);
  return "Base = \"" + base +
    "\", incbase = \"" + incbase +
    "\", ctime = \"" + ctime + "\"";
}
```

返回复杂类型

Java没有规定任何创建ResultSet的方法。因此，返回ResultSet不是一个选项。SQL-2003草案建议将复杂的返回值作为IN/OUT参数处理。PL/Java以这种方式实现了一个ResultSet。如果用户声明一个返回复杂类型的函数，则需要使用带有最后一个参数类型为java.sql.ResultSet的布尔返回类型的Java方法。该参数将被初始化为一个空的可更新结果集，它只包含一行。

假设已经创建了上一节中的complexTest类型。

```
CREATE FUNCTION createComplexTest(int, int)
RETURNS complexTest
AS 'foo.fee.Fum.createComplexTest'
IMMUTABLE LANGUAGE java;
```

PL/Java方法解析现在将在Fum类中找到以下方法：

```
public static boolean complexReturn(int base, int increment,
    ResultSet receiver)
throws SQLException
{
    receiver.updateInt(1, base);
    receiver.updateInt(2, base + increment);
    receiver.updateTimestamp(3, new
        Timestamp(System.currentTimeMillis()));
    return true;
}
```

返回值表示接收方是否应被视为有效的元组（true）或NULL（false）。

返回集的函数

返回结果集时，不要在返回结果集之前构建结果集，因为构建大型结果集将消耗大量资源。最好一次产生一行。顺便提一句，那就是MPP数据库后端期望一个使用SETOF返回的函数。那用户就可以返回SETOF的一个标量类型，如int, float或varchar或者可以返回一个复合类型的SETOF。

返回SETOF<标量类型>

为了返回一组标量类型，用户需要创建一个实现java.util.Iterator接口的Java方法。这是一个返回一个SETOF的varchar的方法的例子：

```
CREATE FUNCTION javatest.getSystemProperties()  
RETURNS SETOF varchar  
AS 'foo.fee.Bar.getNames'  
IMMUTABLE LANGUAGE java;
```

这个简单的Java方法返回一个迭代器：

```
package foo.fee;  
import java.util.Iterator;  
  
public class Bar  
{  
    public static Iterator getNames()  
    {  
        ArrayList names = new ArrayList();  
        names.add("Lisa");  
        names.add("Bob");  
        names.add("Bill");  
        names.add("Sally");  
        return names.iterator();  
    }  
}
```

返回SETOF<复杂类型>

返回SETOF<复杂类型>的方法必须使用接

口org.postgresql.pljava.ResultSetProvider或org.postgresql.pljava.ResultSetHandler。具有两个接口的原因是它们满足两种不同用例的最佳处理。前者适用于要动态创建要从SETOF函数返回的每一行的情况。在用户要返回执行查询的结果的情况下，后者将生成。

使用ResultSetProvider接口

该接口有两种方法。布尔型assignRowValues(java.sql.ResultSet tupleBuilder, int rowNumber)和void close()方法。MPP数据库的查询执行器将重复调用assignRowValues直到它返回假或者直到执行器决定不需要更多行为止。然后它会调用close。

用户可以通过以下方式使用此接口：

```
CREATE FUNCTION javatest.listComplexTests(int, int)
  RETURNS SETOF complexTest
  AS 'foo.fee.Fum.listComplexTest'
  IMMUTABLE LANGUAGE java;
```

该函数映射到一个返回实现ResultSetProvider接口实例的静态java方法。

```
public class Fum implements ResultSetProvider
{
    private final int m_base;
    private final int m_increment;
    public Fum(int base, int increment)
    {
        m_base = base;
        m_increment = increment;
    }
    public boolean assignRowValues(ResultSet receiver, int
currentRow)
    throws SQLException
    {
        // Stop when we reach 12 rows.
        //
        if(currentRow >= 12)
            return false;
        receiver.updateInt(1, m_base);
        receiver.updateInt(2, m_base + m_increment * currentRow);
        receiver.updateTimestamp(3, new
Timestamp(System.currentTimeMillis()));
        return true;
    }
    public void close()
    {
        // Nothing needed in this example
    }
    public static ResultSetProvider listComplexTests(int base,
int increment)
    throws SQLException
    {
        return new Fum(base, increment);
    }
}
```

listComplexTests方法被调用一次。如果没有可用结果或ResultSetProvider实例，将返回NULL。这里的Java类Fum实现了这个接口，所以它返回一个自己的实例。然后将重复调用assignRowValues方法，直到返回false。到那时候，将会调用close。

使用ResultSetHandle接口

该接口类似于ResultSetProvider接口因为它也有将在最后调用的close()方法。但是，不是让evaluator调用一次构建一行的方法，而是返回一个ResultSet的方法。查询evaluator将遍历该集合，并将ResultSet内容一次一个元组传递给调用者，直到对next()的调用返回false或者evaluator决定不需要更多行。

这是一个使用默认连接获取的语句执行查询的示例。适用于部署描述符的SQL看起来像这样：

```
CREATE FUNCTION javatest.listSupers()  
  RETURNS SETOF pg_user  
  AS 'org.postgresql.pljava.example.Users.listSupers'  
  LANGUAGE java;  
CREATE FUNCTION javatest.listNonSupers()  
  RETURNS SETOF pg_user  
  AS 'org.postgresql.pljava.example.Users.listNonSupers'  
  LANGUAGE java;
```

并且在Java包org.postgresql.pljava.example中加入了一个Users类：

```
public class Users implements ResultSetHandle
{
    private final String m_filter;
    private Statement m_statement;
    public Users(String filter)
    {
        m_filter = filter;
    }
    public ResultSet getResultSet()
    throws SQLException
    {
        m_statement =
            DriverManager.getConnection("jdbc:default:connection").createStatement();
        return m_statement.executeQuery("SELECT * FROM pg_user
            WHERE " + m_filter);
    }

    public void close()
    throws SQLException
    {
        m_statement.close();
    }

    public static ResultSetHandle listSupers()
    {
        return new Users("usesuper = true");
    }

    public static ResultSetHandle listNonSupers()
    {
        return new Users("usesuper = false");
    }
}
```

使用JDBC

PL/Java包含映射到PostgreSQL SPI函数的JDBC驱动程序。可以使用以下语句获取映射到当前事务的连接：

```
Connection conn =  
    DriverManager.getConnection("jdbc:default:connection");
```

获取连接后，可以准备和执行类似于其他JDBC连接的语句。这些是PL/Java JDBC驱动程序的限制：

- 事务无法以任何方式进行管理。因此，连接后用户不能用如下方法：
 - `commit()`
 - `rollback()`
 - `setAutoCommit()`
 - `setTransactionIsolation()`
- 在保存点上也有一些限制。保存点不能超过其设置的功能，并且必须由同一功能回滚或释放。
- 从`executeQuery()`返回的结果集始终为`FETCH_FORWARD`和`CONCUR_READ_ONLY`。
- 元数据仅在PL/Java 1.1或更高版本中可用。
- `CallableStatement`（用于存储过程）没有实现。
- `Clob`和`Blob`类型未完全实现，需要更多工作。`byte[]`和`String`可分别用于`bytea`和`text`。

异常处理

您可以像其他任何异常一样捕获并处理MPP数据库后端中的异常。后端的`ErrorData`结构作为一个名为`org.postgresql.pljava.ServerException`(从`java.sql.SQLException`中派生)的类

中的属性公开，并且Java try/catch机制与后端机制同步。

Important: 在函数返回之前，用户将无法继续执行后端函数，并且在后端生成异常时传播错误，除非用户使用了保存点。当回滚保存点时，异常条件被重置，用户可以继续执行。

保存点

MPP数据库保存点使用java.sql.Connection接口公开。有两个限制。

- 必须在设置的函数中回滚或释放保存点。
- 保存点不能超过其设置的功能。

日志

PL/Java使用标准的Java Logger。因此，用户可以按如下写：

```
Logger.getAnonymousLogger().info( "Time is " + new  
Date(System.currentTimeMillis()));
```

目前，记录器使用一个处理程序来映射MPP数据库配置设置的当前状态log_min_messages到有效的Logger级别，并使用MPP数据库后端功能输出所有消息elog()。

Note: 第一次执行会话中的PL/Java函数时，将从数据库中读取log_min_messages设置。在Java端，在特定会话中执行第一个PL/Java函数之后，设置不会更改，直到重新启动使用PL/Java的MPP数据库会话。

Logger级别和MPP数据库后端级别之间适用以下映射。

Table 2. PL/Java日志级别

java.util.logging.Level	MPP数据库级别
SEVERE ERROR	ERROR
WARNING	WARNING
CONFIG	LOG
INFO	INFO
FINE	DEBUG1
FINER	DEBUG2
FINEST	DEBUG3

安全

- [安装](#)
- [可信语言](#)

安装

只有数据库超级用户才可以安装PL/Java。 使用SECURITY DEFINER安装PL/Java实用程序函数，以便它们执行以授予函数创建者的访问权限。

可信语言

PL/Java是一种可信语言。 可信的PL/Java语言无法访问PostgreSQL定义可信语言所规定的文件系统。 任何数据库用户都可以创建和访问受信任的语言的函

数。

PL/Java还为语言javau安装语言处理程序。此版本不受信任，只有超级用户可以创建使用它的新函数。任何用户都可以调用这些函数。

一些PL/Java问题和解决方案

当编写PL/Java时，将JVM映射到与MPP数据库后端代码相同的进程空间中，对于多个线程，异常处理和内存管理，已经出现了一些问题。这里是简要说明如何解决这些问题。

- [多线程](#)
- [异常处理](#)
- [Java垃圾收集器与palloc\(\)和堆栈分配](#)

多线程

Java本身就是多线程的。MPP数据库后端不是。没有什么可以阻止开发人员在Java代码中使用多个Threads类。调用后端的终结器可能是从背景垃圾回收线程中产生的。可能使用的几个第三方Java包使用多个线程。该模式在同一过程中如何与MPP数据库后端共存？

解决方案

解决方案很简单。PL/Java定义了一个特殊对象Backend.THREADLOCK。当初始化PL/Java时，后端立即抓取该对象监视器（即它将在此对象上同步）。当后端调用Java函数时，监视器将被释放，然后在调用返回时立即恢复。来自Java的所有调用到后端代码都在同一个锁上同步。这确保一次只能有一个线程

可以从Java调用后端，并且只能在后端正在等待返回Java函数调用的时候调用。

异常处理

Java经常使用try/catch/finally块。MPP数据库有时会使用一个异常机制来调用longjmp来将控件转移到已知状态。这样的跳转通常会有效地绕过JVM。

解决方案

后端现在允许使用宏PG_TRY/PG_CATCH/PG_END_TRY捕获错误，并且在catch块中，可以使用ErrorData结构检查错误。PL/Java实现了一个名为org.postgresql.pljava.ServerException的java.sql.SQLException子类。可以从该异常中检索和检查ErrorData。允许捕获处理程序发送回滚到保存点。回滚成功后，执行可以继续。

Java垃圾收集器与palloc()和堆栈分配

原始类型始终按值传递。包括String类型(这是必需的，因为Java使用双字节字符)。复杂类型通常包含在Java对象中并通过引用传递。例如，Java对象可以包含指向palloc'ed或stack分配的内存的指针，并使用原生的JNI调用来提取和操作数据。一旦调用结束，这些数据将变得陈旧。进一步尝试访问这些数据最多只会产生非常不可预知的结果，但更有可能导致内存错误和崩溃。

解决方案

PL/Java包含的代码可以确保当MemoryContext或堆栈分配超出范围时，陈旧的

指针被清除。Java包装器对象可能会生效，但是使用它们的任何尝试将导致陈旧的原生处理异常。

示例

以下简单的Java示例创建一个包含单个方法并运行该方法的JAR文件。

Note: 该示例需要Java SDK来编译Java文件。

以下方法返回一个子字符串。

```
{
public static String substring(String text, int beginIndex,
int endIndex)
{
return text.substring(beginIndex, endIndex);
}
}
```

在文本文件example.class中输入这些Java代码。

manifest.txt文件的内容：

```
Manifest-Version: 1.0
Main-Class: Example
Specification-Title: "Example"
Specification-Version: "1.0"
Created-By: 1.6.0_35-b10-428-11M3811
Build-Date: 01/20/2013 10:09 AM
```

编译java代码：

```
javac *.java
```

创建名为analytics.jar的JAR存档，其中包含JAR中的类文件和清单文件MANIFEST文件。

```
jar cfm analytics.jar manifest.txt *.class
```

将jar文件上传到MPP master主机。

运行gpscp实用程序将jar文件复制到MPP Java目录。使用-f选项指定包含master节点和segment节点主机列表的文件。

```
gpscp -f gphosts_file analytics.jar  
=:/usr/local/mpp-db/lib/postgresql/java/
```

使用gpconfig程序设置MPPpljava_classpath服务器配置参数。该参数列出已安装的jar文件。

```
gpconfig -c pljava_classpath -v 'analytics.jar'
```

运行gpstop实用程序-u选项重新加载配置文件。

```
gpstop -u
```

来自psql命令行，运行以下命令显示已安装的jar文件。

```
show pljava_classpath
```

以下SQL命令创建一个表并定义一个Java函数来测试jar文件中的方法：

```
create table temp (a varchar) distributed randomly;  
insert into temp values ('my string');  
--Example function  
create or replace function java_substring(varchar, int, int)  
returns varchar as 'Example.substring' language java;  
--Example execution  
select java_substring(a, 1, 5) from temp;
```

用户可以将内容放在一个文件mysample.sql中，并从psql命令行运行该命令：

```
> \i mysample.sql
```

输出类似于：

```
java_substring  
-----  
y st  
(1 row)
```

参考

The PL/Java Github wiki page - <https://github.com/tada/pljava/wiki>.

PL/Java 1.5.0 release - https://github.com/tada/pljava/tree/REL1_5_STABLE.

MPP的PL/Perl语言扩展

本章包含了如下信息：

- [关于MPP的PL/Perl](#)
- [MPP数据库中PL/Perl限制](#)
- [可信/不可信语言](#)
- [用PL/Perl开发函数](#)

Parent topic: [MPP数据库参考手册](#)

关于MPP的PL/Perl

通过MPP数据库的PL/Perl扩展，用户可以用Perl写用户定义的函数，可以利用Perl先进的字符串处理操作和函数。PL/Perl同时提供了可信和不可信的语言变体。

PL/Perl内嵌在用户的MPP数据库发布中。MPP数据库的PL/Perl要求系统中每台数据库主机都要事先装好Perl。

参考[PostgreSQL PL/Perl文档](#)获取额外信息。

MPP数据库中PL/Perl限制

MPP数据库PL/Perl的限制包括：

- MPP数据库不支持PL/Perl触发器。
- PL/Perl函数不能直接相互调用。
- SPI还没有完全的实现。
- 如果用户使用spi_exec_query()获取非常大的数据集，用户应该要意识到它们都将进入内存中。用户可以通过使用spi_query()/spi_fetchrow()来防止这个问题。

题的发生。一个相似的问题也可能发生，当集合返回函数通过一个return语句传递一个大量行的数据集到MPP数据库中。对每一返回行使用return_next能够避免该问题的出现。

- 当一个会话正常结束而不是由于一个致命错误结束时，PL/Perl会执行用户定义好的任何END块。当前没有其它的动作执行（文件处理不会被自动地刷写并且对象也不会被自动销毁）。

可信/不可信语言

PL/Perl 包含了可信和不可信两种语言的变体。

PL/Perl可信语言被命名为plperl。可信PL/Perl语言限制文件系统操作，和require、use以及其它可能同操作系统或者数据库服务进程进行交互的语句。有了这些限制，任何MPP用户能够创建和执行可信的plperl语言的函数。

PL/Perl不可信语言被命名为plperlu。您不能使用plperlu不受信任的语言限制您创建的函数的操作。只有数据库的超级用户才有权限创建有不可信PL/Perl语言的用户定义函数。同时只有数据库的超级用户以及其他被显式授予特权的用户能够执行不可信PL/Perl的用户定义函数。

在解释器以及运行在单个进程中的多个解释器之间的通信方面，PL/Perl有限制。请参考 PostgreSQL的[可信与不可信PL/Perl](#)PL/Perl文档获取更多的信息。

启用和移除PL/Perl支持

在用户能在数据库中创建以及执行一个PL/Perl的用户定义函数前，必须要在该数据库上注册PL/Perl语言。要实现移除PL/Perl，用户必须显式的从每一个它注册的数据库中移除扩展。您必须是数据库超级用户或所有者才能能在MPP数据库中注册或删除受信任的语言。

Note: 只用数据库超级用户可能注册或者移除对不可信PL/Perl语言plperlu的支持。

在用户在数据库中开启或者移除PL/Perl支持之前，确保：

- 用户的MPP数据库正在运行。
- 用户已经执行（source）了mpp_path.sh。
- 用户已经设置了\$MASTER_DATA_DIRECTORY和\$GPHOME环境变量。

开启PL/Perl支持

对于要在其中启用PL/Perl的每个数据库，请使用SQL [CREATE EXTENSION](#) 命令注册该语言。例如，以gpadmin用户身份运行以下命令，为名为testdb的数据库注册可信PL/Perl语言：

```
$ psql -d testdb -c 'CREATE EXTENSION plperl;'
```

移除PL/Perl

要从数据库中删除对PL/Perl的支持，请运行SQL [DROP EXTENSION](#) 命令。例如，以gpadmin用户身份运行以下命令，从名为testdb的数据库中删除对可信PL/Perl语言的支持：

```
$ psql -d testdb -c 'DROP EXTENSION plperl;'
```

如果任何现有对象（如函数）依赖于语言，则默认命令将失败。指定CASCADE选项也可以删除所有依赖对象，包括使用PL/Perl创建的函数。

用PL/Perl开发函数

您可以使用标准SQL `CREATE FUNCTION`语法定义PL/Perl函数。PL/Perl用户定义函数的主体是普通的Perl代码。PL/Perl解释器将此代码包装在Perl子例程中。

您还可以使用PL/Perl创建匿名代码块。使用SQL `DO`命令调用的匿名代码块不接收任何参数，并且丢弃它可能返回的任何值。否则，PL/Perl匿名代码块的行为就像一个函数。只有数据库超级用户使用不受信任的plperl语言创建匿名代码块。

`CREATE FUNCTION`命令的语法要求您将PL/Perl函数体写为字符串常量。虽然使用美元引用更方便，但您可以选择使用转义字符串语法（E"），前提是您将函数体中使用的任何单引号和反斜杠加倍。

PL/Perl参数和结果在Perl中处理。传递给PL/Perl函数的参数可通过`@_`数组访问。使用`return`语句返回结果值，或者作为函数中计算的最后一个表达式返回结果值。PL/Perl函数不能直接返回非标量类型，因为您在标量上下文中调用它。您可以通过返回引用来返回PL/Perl函数中的非标量类型（如数组，记录和集）。

PL/Perl将null参数值视为“未定义”。将`STRICT`关键字添加到`LANGUAGE`子句指示MPP数据库在任何输入参数为null时立即返回null。当创建为`STRICT`时，函数本身不需要执行null检查。

以下PL/Perl函数使用`STRICT`关键字返回两个整数中的较大者，如果任何输入为null，则返回null：


```
CREATE FUNCTION perl_max (integer, integer) RETURNS integer AS $$
    if ($_[0] > $_[1]) { return $_[0]; }
    return $_[1];
$$ LANGUAGE plperl STRICT;
```

```
SELECT perl_max( 1, 3 );
perl_max
-----
      3
(1 row)
```

```
SELECT perl_max( 1, null );
perl_max
-----

(1 row)
```

PL/Perl认为函数参数中的任何内容都不是对字符串的引用，即标准的MPP数据库外部文本表示。提供给PL/Perl函数的参数值只是转换为文本形式的输入参数（就像它们已经被SELECT语句显示一样）。如果函数参数不是普通的数字或文本类型，则必须将MPP数据库类型转换为Perl更易使用的形式。相反，return和return_next语句接受任何字符串，该字符串是函数声明的返回类型的可接受输入格式。

有关其他信息，请参阅PostgreSQL [PL/Perl函数和参数](#)文档，包括复合类型和结果集操作。

内置PL/Perl函数

PL/Perl包含用于访问数据库的内置函数，包括用于准备和执行查询以及操作查询结果的函数。该语言还包括用于错误记录和字符串操作的实用程序函数。

以下示例创建一个包含整数和文本列的简单表。它创建一个PL/Perl用户定义函数，该函数接受输入字符串参数并调用spi_exec_query()内置函数来选择表的所

有列和行。该函数返回查询结果中的所有行，其中v列包含函数输入字符串。

```
CREATE TABLE test (
    i int,
    v varchar
);
INSERT INTO test (i, v) VALUES (1, 'first line');
INSERT INTO test (i, v) VALUES (2, 'line2');
INSERT INTO test (i, v) VALUES (3, '3rd line');
INSERT INTO test (i, v) VALUES (4, 'different');

CREATE OR REPLACE FUNCTION return_match(vchar) RETURNS SETOF test AS $$
    # store the input argument
    $ss = $_[0];

    # run the query
    my $rv = spi_exec_query('select i, v from test;');

    # retrieve the query status
    my $status = $rv->{status};

    # retrieve the number of rows returned in the query
    my $nrows = $rv->{processed};

    # loop through all rows, comparing column v value with input argument
    foreach my $rn (0 .. $nrows - 1) {
        my $row = $rv->{rows}[$rn];
        my $textstr = $row->{v};
        if( index($textstr, $ss) != -1 ) {
            # match! return the row.
            return_next($row);
        }
    }
    return undef;
$$ LANGUAGE plperl EXECUTE ON MASTER ;

SELECT return_match( 'iff' );
return_match
-----
(4,different)
(1 row)
```

有关可用函数的详细讨论，请参阅PostgreSQL PL/Perl[内置函数文档](#)。

PL/Perl中的全局值

您可以使用全局哈希映射%_SHARED在当前会话的生命周期内在PL/Perl函数调用之间共享数据，包括代码引用。

以下示例使用%_SHARED在用户定义的set_var()和get_var() PL/Perl函数之间共享数据：

```
CREATE OR REPLACE FUNCTION set_var(name text, val text) RETURNS text AS $$
    if ($_SHARED[$_0] = $_1) {
        return 'ok';
    } else {
        return "cannot set shared variable $_0 to $_1";
    }
$$ LANGUAGE plperl;

CREATE OR REPLACE FUNCTION get_var(name text) RETURNS text AS $$
    return $_SHARED[$_0];
$$ LANGUAGE plperl;

SELECT set_var('key1', 'value1');
set_var
-----
ok
(1 row)

SELECT get_var('key1');
get_var
-----
value1
(1 row)
```

出于安全原因，PL/Perl为每个角色创建一个单独的Perl解释器。这可以防止一个用户对另一个用户的PL/Perl函数的行为进行意外或恶意干扰。每个这样的解

释器都保留其自己的%_SHARED变量值和其他全局状态。当且仅当它们由相同的SQL角色执行时，两个PL/Perl函数共享相同的%_SHARED值。

在某些情况下，您必须采取明确的步骤来确保PL/Perl函数可以在%_SHARED中共享数据。例如，如果应用程序在单个会话中在多个SQL角色（通过SECURITY DEFINER函数，使用SET ROLE等）下执行，请确保需要通信的函数由同一用户拥有，并将这些函数标记为SECURITY DEFINER。

备注

在开发PL/Perl函数其它要考虑的：

- PL/Perl内部使用UTF-8编码。它将其他编码中提供的任何参数转换为UTF-8，并将UTF-8的返回值转换回原始编码。
- 嵌套命名的PL/Perl子例程保留了与Perl相同的危险。
- 只有不受信任的PL/Perl语言变体支持模块导入。小心使用plperl。
- 您在plperl函数中使用的任何模块都必须在所有MPP数据库主机上的相同位置可用。

用于分析的MPP MADlib扩展

本章节包含了以下信息：

- [关于 MADlib](#)
.....
 - [示例](#)
.....
 - [参考](#)
.....
-

Parent topic: [MPP数据库参考手册](#)

关于 MADlib

MADlib是一个可扩展数据库分析的开源库。通过MPP的MADlib扩展，用户可以在MPP数据库中使用MADlib功能。

MADlib为结构化数据以及非结构化数据提供了数学、统计学以及机器学习方法的数据并行的实现。它提供了一整套基于SQL的机器学习、数据挖掘以及统计学算法，只需要运行在数据库引擎中，而不需要在MPP和其它工具之间进行数据的传递。

MADlib需要m4宏处理器版本1.4.13或更高版本。

MADlib可以与R一同使用，一个R包允许用户使用R客户端同MPP的数据进行交互。见[关于MADlib、R、R](#)。

示例

下面是使用MPPMADlib扩展的示例：

- [线性回归](#)
- [关联规则](#)
- [朴素贝叶斯分类](#)

见MADlib文档获取更多的示例。

线性回归

该示例是在表regr_example执行一个线性回归。因变量数据在y列中，独立变量数据在x1和x2列中。

下面的语句创建regr_example表同时加载样本数据：

```
DROP TABLE IF EXISTS regr_example;
CREATE TABLE regr_example (
  id int,
  y int,
  x1 int,
  x2 int
);
INSERT INTO regr_example VALUES
(1, 5, 2, 3),
(2, 10, 7, 2),
(3, 6, 4, 1),
(4, 8, 3, 4);
```

MADlib的linregr_train()函数产生一个根据一个输入表包含的训练数据产生一个回归模型。下面的SELECT语句在表regr_example执行一个简单的多元回归同时保存模型在表reg_example_model中。

```
SELECT madlib.linregr_train (
  'regr_example',      -- source table
  'regr_example_model', -- output model table
  'y',                -- dependent variable
  'ARRAY[1, x1, x2]'   -- independent variables
);
```

madlib.linregr_train()函数能够添加参数来设置分组的列以及计算模型的异方差性。

Note: 截距通过将一个独立变量设置为常数1来计算，如前一个例子所示。

在表regr_example上执行该查询并创建带有一行数据的regr_example_model表：

```
SELECT * FROM regr_example_model;
-[ RECORD 1 ]-----+-----
coef              | {0.1111111111111127,1.14814814814815,1.01851851851852}
r2                | 0.968612680477111
std_err           | {1.49587911309236,0.207043331249903,0.346449758034495}
t_stats           | {0.0742781352708591,5.54544858420156,2.93987366103776}
p_values          | {0.952799748147436,0.113579771006374,0.208730790695278}
condition_no      | 22.650203241881
num_rows_processed | 4
num_missing_rows_skipped | 0
variance_covariance | {{2.23765432098598,-0.257201646090342,-0.437242798353582},
                        {-0.257201646090342,0.042866941015057,0.0342935528120456},
                        {-0.437242798353582,0.0342935528120457,0.12002743484216}}
```

被保存到regr_example_model表中的模型能够同MADlib线性回归预测函数使用，madlib.linregr_predict()来查看残差：

```
SELECT regr_example.*,
       madlib.linregr_predict ( ARRAY[1, x1, x2], m.coef ) as predict,
       y - madlib.linregr_predict ( ARRAY[1, x1, x2], m.coef ) as residual
FROM regr_example, regr_example_model m;
id | y | x1 | x2 | predict | residual
----+-----+-----+-----+-----+-----
1 | 5 | 2 | 3 | 5.46296296296297 | -0.462962962962971
3 | 6 | 4 | 1 | 5.72222222222224 | 0.2777777777777762
2 | 10 | 7 | 2 | 10.1851851851852 | -0.185185185185201
4 | 8 | 3 | 4 | 7.62962962962964 | 0.370370370370364
(4 rows)
```

关联规则

这个例子说明了关联规则的数据挖掘技术在交易数据集。关联规则挖掘是发现

大数据集中的变量之间关系的技术。这个例子将考虑那些在商店中通常一起购买的物品。除了购物篮分析，关联规则也应用在生物信息学中，网络分析，和其他领域。

这个例子分析利用MADlib函数MADlib.assoc_rules分析存储在表中的关于七个交易的购买信息。函数假定数据存储在两列中，每行有一个物品和交易ID。多个物品的交易，包括多个行，每行一个物品。

这些命令创建表。

```
DROP TABLE IF EXISTS test_data;
CREATE TABLE test_data (
  trans_id INT,
  product text
);
```

该INSERT命令向表中添加数据。

```
INSERT INTO test_data VALUES
  (1, 'beer'),
  (1, 'diapers'),
  (1, 'chips'),
  (2, 'beer'),
  (2, 'diapers'),
  (3, 'beer'),
  (3, 'diapers'),
  (4, 'beer'),
  (4, 'chips'),
  (5, 'beer'),
  (6, 'beer'),
  (6, 'diapers'),
  (6, 'chips'),
  (7, 'beer'),
  (7, 'diapers');
```

MADlib函数madlib.assoc_rules()分析数据同时确定具有以下特征的关联规则。

- 一个值至少为.40的支持率。支持率表示包含X的交易与所有交易的比。
- 一个值至少为.75的置信率。置信率表示包含X的交易与包含Y的交易的比。可以将该度量看做给定Y下X的条件概率。

该SELECT命令确定关联规则，创建表assoc_rules同时添加统计信息到表中。

```
SELECT * FROM madlib.assoc_rules (  
  .40,      -- support  
  .75,      -- confidence  
  'trans_id', -- transaction column  
  'product', -- product purchased column  
  'test_data', -- table name  
  'public',  -- schema name  
  false);    -- display processing details
```

这是SELECT命令的输出。这有两条符合特征的规则。

```
output_schema | output_table | total_rules | total_time  
-----+-----+-----+-----  
public      | assoc_rules |          2 | 00:00:01.153283  
(1 row)
```

为了查看关联规则，用户可以使用该SELECT命令。

```
SELECT pre, post, support FROM assoc_rules  
ORDER BY support DESC;
```

这是输出。pre和post列分别是关联规则左右两边的项集。

```
pre  | post | support  
-----+-----+-----  
{diapers} | {beer} | 0.714285714285714  
{chips}  | {beer} | 0.428571428571429  
(2 rows)
```

基于数据，啤酒和尿布经常一起购买。为了增加销售额，用户可以考虑将啤酒和尿布放在一个架子上。

朴素贝叶斯分类

朴素贝叶斯分析，基于一个或多个独立变量或属性，预测一类变量或类结果的可能性。类变量是非数值类型变量，一个变量可以有一个数量有限的值或类别。类变量表示的整数，每个整数表示一个类别。例如，如果类别可以是一个“真”、“假”，或“未知”的值，那么变量可以表示为整数1, 2或3。

属性可以是数值类型、非数值类型以及类类型。训练函数有两个签名 - 一个用于所有属性为数值 另外一个用于混合数值和类类型的情况。后者的附加参数标识那些应该被当做数字值处理的属性。属性以数组的形式提交给训练函数。

MADlib朴素贝叶斯训练函数产生一个特征概率表和一个类的先验表，该表可以同预测函数使用为属性集提供一个类别的概率。

朴素贝叶斯示例 1 - 简单的所有都是数值属性

在第一个示例中，类变量取值为1或者2，同时这里有三个整型属性。

1. 下面的命令创建输入表以及加载样本数据。

```
DROP TABLE IF EXISTS class_example CASCADE;
CREATE TABLE class_example (
  id int, class int, attributes int[]);
INSERT INTO class_example VALUES
  (1, 1, '{1, 2, 3}'),
  (2, 1, '{1, 4, 3}'),
  (3, 2, '{0, 2, 2}'),
  (4, 1, '{1, 2, 1}'),
  (5, 2, '{1, 2, 2}'),
  (6, 2, '{0, 1, 3}');
```

在生产环境中的实际数据比该示例中的数据量更大，也能获得更好的结果。更大的训练数据集能够显著地提高分类的精确度。

2. 使用create_nb_prepared_data_tables()函数训练模型。

```
SELECT * FROM madlib.create_nb_prepared_data_tables (  
  'class_example',      -- name of the training table  
  'class',              -- name of the class (dependent) column  
  'attributes',          -- name of the attributes column  
  3,                    -- the number of attributes  
  'example_feature_probs', -- name for the feature probabilities output table  
  'example_priors'       -- name for the class priors output table  
);
```

3. 为了使用模型进行分类，创建带有数据的表。

```
DROP TABLE IF EXISTS class_example_topredict;  
CREATE TABLE class_example_topredict (  
  id int, attributes int[]);  
INSERT INTO class_example_topredict VALUES  
  (1, '{1, 3, 2}'),  
  (2, '{4, 2, 2}'),  
  (3, '{2, 1, 1}');
```

4. 用特征概率、类的先验和class_example_topredict表创建一个分类视图。

```
SELECT madlib.create_nb_probs_view (  
  'example_feature_probs', -- feature probabilities output table  
  'example_priors',        -- class priors output table  
  'class_example_topredict', -- table with data to classify  
  'id',                    -- name of the key column  
  'attributes',             -- name of the attributes column  
  3,                        -- number of attributes  
  'example_classified'     -- name of the view to create  
);
```

5. 显示分类结果。

```
SELECT * FROM example_classified;
```

```
key | class | nb_prob
```

```
-----+-----+-----
```

```
1 | 1 | 0.4
```

```
1 | 2 | 0.6
```

```
3 | 1 | 0.5
```

```
3 | 2 | 0.5
```

```
2 | 1 | 0.25
```

```
2 | 2 | 0.75
```

```
(6 rows)
```

朴素贝叶斯示例 2 – 天气和户外运动

该示例计算在给定的天气条件下，用户要进行户外运动，例如高尔夫、网球等的概率。

表weather_example包含了样本值。

表的标识列是day，整型类型。

play列包含了因变量以及两个类别：

- 0 - No
- 1 - Yes

有四个属性：outlook、temperature、humidity、以及wind。他们是类变量。

MADlib create_nb_classify_view()函数希望属性提供的是INTEGER、NUMERIC、或者FLOAT8值类型的数组，所以该示例的属性均用为整型进行编码：

- *outlook* 可能取值为 sunny (1), overcast (2), 或 rain (3).
- *temperature* 可能取值为 hot (1), mild (2), 或 cool (3).
- *humidity* 可能取值为 high (1) 或 normal (2).
- *wind* 可能取值为 strong (1) 或 weak (2).

下表显示了编码变量之前的训练数据。

day	play	outlook	temperature	humidity	wind
2	No	Sunny	Hot	High	Strong
4	Yes	Rain	Mild	High	Weak
6	No	Rain	Cool	Normal	Strong
8	No	Sunny	Mild	High	Weak
10	Yes	Rain	Mild	Normal	Weak
12	Yes	Overcast	Mild	High	Strong
14	No	Rain	Mild	High	Strong
1	No	Sunny	Hot	High	Weak
3	Yes	Overcast	Hot	High	Weak
5	Yes	Rain	Cool	Normal	Weak
7	Yes	Overcast	Cool	Normal	Strong
9	Yes	Sunny	Cool	Normal	Weak
11	Yes	Sunny	Mild	Normal	Strong
13	Yes	Overcast	Hot	Normal	Weak
(14 rows)					

1. 创建一个训练表。

```
DROP TABLE IF EXISTS weather_example;
CREATE TABLE weather_example (
  day int,
  play int,
  attrs int[]
);
INSERT INTO weather_example VALUES
( 2, 0, '{1,1,1,1}'), -- sunny, hot, high, strong
( 4, 1, '{3,2,1,2}'), -- rain, mild, high, weak
( 6, 0, '{3,3,2,1}'), -- rain, cool, normal, strong
( 8, 0, '{1,2,1,2}'), -- sunny, mild, high, weak
(10, 1, '{3,2,2,2}'), -- rain, mild, normal, weak
(12, 1, '{2,2,1,1}'), -- etc.
(14, 0, '{3,2,1,1}'),
( 1, 0, '{1,1,1,2}'),
( 3, 1, '{2,1,1,2}'),
( 5, 1, '{3,3,2,2}'),
( 7, 1, '{2,3,2,1}'),
( 9, 1, '{1,3,2,2}'),
(11, 1, '{1,2,2,1}'),
(13, 1, '{2,1,2,2}');
```

2. 根据训练表创建模型。

```
SELECT madlib.create_nb_prepared_data_tables (
  'weather_example', -- training source table
  'play',            -- dependent class column
  'attrs',           -- attributes column
  4,                 -- number of attributes
  'weather_probs',   -- feature probabilities output table
  'weather_priors'   -- class priors
);
```

3. 查看特征概率：

```
SELECT * FROM weather_probs;
class | attr | value | cnt | attr_cnt
-----+-----+-----+-----+-----
1 | 3 | 2 | 6 | 2
1 | 1 | 2 | 4 | 3
0 | 1 | 1 | 3 | 3
0 | 1 | 3 | 2 | 3
0 | 3 | 1 | 4 | 2
1 | 4 | 1 | 3 | 2
1 | 2 | 3 | 3 | 3
1 | 2 | 1 | 2 | 3
0 | 2 | 2 | 2 | 3
0 | 4 | 2 | 2 | 2
0 | 3 | 2 | 1 | 2
0 | 1 | 2 | 0 | 3
1 | 1 | 1 | 2 | 3
1 | 1 | 3 | 3 | 3
1 | 3 | 1 | 3 | 2
0 | 4 | 1 | 3 | 2
0 | 2 | 3 | 1 | 3
0 | 2 | 1 | 2 | 3
1 | 2 | 2 | 4 | 3
1 | 4 | 2 | 6 | 2
(20 rows)
```

4. 用模型分类一组记录，首先装载数据到一个表中。 在该示例中，表t1有四个行将要分类。

```
DROP TABLE IF EXISTS t1;
CREATE TABLE t1 (
  id integer,
  attributes integer[]);
insert into t1 values
(1, '{1, 2, 1, 1}'),
(2, '{3, 3, 2, 1}'),
(3, '{2, 1, 2, 2}'),
(4, '{3, 1, 1, 2}');
```

5. 使用MADlib create_nb_classify_view()函数对表中的行进行分类。

```
SELECT madlib.create_nb_classify_view (  
  'weather_probs',    -- feature probabilities table  
  'weather_priors',   -- classPriorsName  
  't1',               -- table containing values to classify  
  'id',               -- key column  
  'attributes',       -- attributes column  
  4,                 -- number of attributes  
  't1_out'            -- output table name  
);
```

结果有四行，每行对应表t1中的一条记录。

```
SELECT * FROM t1_out ORDER BY key;  
key | nb_classification  
----+-----  
1 | {0}  
2 | {1}  
3 | {1}  
4 | {0}  
(4 rows)
```

参考

MADlib网站在<http://madlib.apache.org/>。

MADlib文档在<http://madlib.apache.org/documentation.html>。

R是第一类能够让用户使用R客户端对MPP驻留的数据和MADLib进行交互的R包。

关于MADlib、R、R

R语言是一门用于统计计算的开源编程语言。R 是一个能够让用户通过R客户端与常驻MPP数据库的数据进行交互的R语言包。使用R要求MADlib已经安装在了MPP数据库中。

R允许R用户不用离开R命令行就能利用数据库内分析的可扩展性和性能。计算工作在数据库内执行，而终端用户受益于熟悉的R语言接口。与相应的原生R函数相比，在可扩展性上得到提升的同时在执行时间上有降低。此外，R消除了对于非常大的数据集需要花费几个小时完成的数据移动。

R包的关键特征：

- 以R语法的方式探索和操作数据库内的数据。SQL翻译由R来执行。
- 使用熟悉的R语法的预测分析算法，例如线性和逻辑回归。R访问MADlib数据库内分析函数调用。
- 对于广泛关于以标准R格式的示例文档包能够通过R客户端来访问。
- R包也支持MADlib函数的访问。

更多关于R的信息包括支持的MADlib功能的信息，见
<https://cwiki.apache.org/confluence/display/MADLIB/R>。

R的R语言包可以在<https://cran.r-project.org/web/packages/R/index.html>找到。

MPP的PostGIS扩展

该章节包含以下信息：

- [关于MPP的PostGIS](#)
 - [MPP的PostGIS扩展](#)
 - [PostGIS使用](#)
-

上一个话题：[MPP数据库参考手册](#)

关于MPP的PostGIS

PostGIS是MPP数据库的空间数据库扩展。允许将GIS（地理信息系统）对象存储在数据库中。MPP PostGIS扩展包括基于GiST的R-Tree空间索引和用于GIS对象进行分析和处理的函数。

可以去<http://postgis.refrations.net>获取更多关于PostGIS的信息。

MPP的PostGIS扩展

MPP数据库默认安装了PostGIS 2.1.5版本。

PostGIS使用

具体用法请参考官方PostGIS文档：<http://postgis.net/docs/manual-2.1/>。

下面例子中的SQL语句创建带有几何结构的非OpenGIS表。

```
CREATE TABLE geom_test ( gid int4, geom geometry, name varchar(25) );
INSERT INTO geom_test ( gid, geom, name )
VALUES ( 1, 'POLYGON((0 0 0,0 5 0,5 5 0,5 0 0,0 0 0))', '3D Square');
INSERT INTO geom_test ( gid, geom, name )
VALUES ( 2, 'LINESTRING(1 1 1,5 5 5,7 7 5)', '3D Line' );
INSERT INTO geom_test ( gid, geom, name )
VALUES ( 3, 'MULTIPOINT(3 4,8 9)', '2D Aggregate Point' );
SELECT * from geom_test WHERE geom &&
Box3D(ST_GeomFromEWKT('LINESTRING(2 2 0, 3 3 0)));
```

下面例子中的SQL语句创建了一个表，向表中添加一个几何类型字段，并通过一个SRID整型值和SPATIAL_REF_SYS表中的一条记录关联起来。INSERT语句想向表中插入几何类型元组。

```
CREATE TABLE geotest (id INT4, name VARCHAR(32) );
SELECT AddGeometryColumn('geotest','geopoint', 4326,'POINT',2);
INSERT INTO geotest (id, name, geopoint) VALUES (1, 'Olympia', ST_GeometryFromText('POINT(123456 789012)', 4326));
INSERT INTO geotest (id, name, geopoint) VALUES (2, 'Renton', ST_GeometryFromText('POINT(123456 789012)', 4326));
SELECT name,ST_AsText(geopoint) FROM geotest;
```

空间索引

MPP数据库提供对GiST空间索引支持。GiST方案甚至为大对象提供索引。它使用一个有损索引的系统，索引中较小的对象作为较大对象的代理。在PostGIS索引系统中，索引中所有的对象使用他们的边界值作为代理。

可以通过下面的SQL创建一个GiST索引：

```
CREATE INDEX indexname ON tablename USING GIST (geometryfield);
```

MPP的Job_schedule扩展

- [系统概述](#)
- [数据库扩展](#)
- [客户端工具](#)
- [数据库接口](#)

上一个话题：

系统概述

任务调度是指按预定的时间，自动执行设定的动作，完成预期的任务。

整个任务调度系统中最重要的对象是作业（job），而其它的对象都是对作业的支撑。作业是任务调度的基本单元，是一个集合，由步骤（step或称动作action）和调度规则（schedule）组成。

作业步骤是用户定义的对数据库或服务器执行的动作，每个作业必须指定一个步骤。一个步骤可以是一条或多条SQL语句，也可以是操作系统命令如shell脚本。

调度规则是用户定义的一个时间安排，指定一个作业在何、时执行多长时间。在给定的时刻到来时，系统会启动相关的作业，按照作业定义的步骤依次执行。调度可以是一次性的，也可以是周期性的。时间间隔采用日历表示法。如果修改了执行时间或调度规则，会自动重新计算作业下次执行时间。

任务调度功能以扩展的方式提供，用户可通过CREATE EXTENSION pgagent创建出扩展，然后启动客户端工具job_schedule，再通过数据库接口设置定时任务，就可以使用任务调度功能了。

数据库扩展

要在数据库中启用任务调度功能，为每一个数据库运行一次扩展创建语句CREATE EXTENSION pgagent。例如，在数据库testdb中启动任务调度功能，可使用这个命令：

```
psql -p 5432 -d testdb -c "CREATE EXTENSION pgagent;"
```

创建扩展会创建出任务调度功能所需要的模式、表、视图和函数等数据库对象。用户可输入SQL语句或调用函数来定义和查看任务信息。

客户端工具

job_schedule（可执行程序）是MPP数据库用于管理和执行任务调度的客户端工具。job_schedule的执行依赖于数据库中的计划任务。每一个数据库服务器同时只允许一个job_schedule对其提供服务。

概要

```
job_schedule [option...] [connection-option...]  
job_schedule -? | --help  
job_schedule -v
```

描述

启动job_schedule进程后，开始循环扫描数据库的任务配置信息，并开始按照调度规则逐个任务调度执行。进程可以并行调度多个任务。当任务执行失败时，能在有限次数内自动重试。

选项

option为通用选项：

- f
指定是否需要后台运行。默认后台运行。
- t
指定任务轮询时间。单位：秒，默认10，必须大于0。
- r
指定发生异常时程序重试间隔时间。单位：秒，默认30，必须大于10。
- s
指定日志文件。如果没有指定，日志将记录到stdout。
- l
指定记录日志等级。默认：0。ERROR=0，WARNING=1，DEBUG=0。
- v
显示job_schedule版本并退出。
- ? | --help
显示有关job_schedule命令行参数的帮助并退出。

connection-option为控制数据库连接参数的选项：

dbname

指定要连接到的数据库名。

host

指定服务器正在运行的机器的主机名。如果该值开始于一个斜线，它被用作一个Unix域套接字的目录。

port

指定服务器正在监听连接的TCP端口或本地Unix域套接字文件扩展名。默认使用编译在程序中的默认值。

username

指定连接数据库的用户名。

password

指定与数据库用户名匹配的密码。

示例

例1：启用一个运行在终端的job_schedule：

```
job_schedule -f -l 2 "user=test password=***** dbname=testdb host=127.0.0.1 port=5432"
```

例2：启动一个在后端运行的job_schedule（推荐）

```
job_schedule -l 2 "user=test password=***** dbname=testdb host=127.0.0.1 port=5432" &
```

数据库接口

用户可以创建、删除、修改、运行、启用和停用任务。

用户可以设置作业的名称、描述、作业是否使能、下次运行时间等信息。

用户可以设置步骤的名称、描述、动作类型、动作和数据库连接串等信息。

用户可以设置调度规则的名称、描述、开始时间、结束时间和重复执行的时间间隔等信息。

为方便用户管理和查看任务定义和运行信息，新增如下视图：

视图名称	描述
ALL_JOBS	所有任务信息
ALL_ACTIONS	所有步骤信息
ALL_SCHEDULES	所有调度规则信息

视图名称	描述
ALL_JOB_LOG	所有任务运行日志
ALL_JOB_RUN_DETAILS	所有任务运行详细日志

为了提供面向对象的任务管理方法，新增如下函数：

函数名称	描述
broken	启用或停用一个任务。停用的任务不会再被调度。
change	修改任务执行的动作、调度时间和间隔等。
interval	修改任务的时间间隔（日历表示法）。
next_date	修改任务的调度时间。
remove	删除一个任务。
run	运行一个任务。
submit	创建一个任务。
what	修改任务执行的动作。

这组接口函数提供与Oracle的DBMS_JOB包类似的功能。

新增GUC参数控制任务失败重试次数：

参数名称	描述
max_failures	任务执行失败时重试的最大次数

函数 broken

名称

```
pgagent.broken(in_jobname text, in_broken boolean, in_next_date TIMESTAMP)
```

参数

参数	描述
in_jobname	任务名称
in_broken	true停用任务，false启用任务。
in_next_date	任务下次运行时间。

描述

函数pgagent.broken用于启用或停用一个任务。

如果指定的in_jobname不存在则报错退出；否则更新pga_job表中名为in_jobname的任务的jobenabled字段。被启用的任务，如果到期被会调度。停用的任务不会再被调度。

用法

#启用任务job1：

```
SELECT pgagent.broken('job1', false, now()::timestamp);
```

函数 change

名称

```
pgagent.change(  
  in_jobname text,  
  in_what TEXT,  
  in_next_date TIMESTAMP,  
  in_interval TEXT)
```

参数

参数	描述
in_jobname	任务名称
in_what	步骤动作，批处理命令或SQL命令内容。
in_next_date	任务下次运行时间。
in_interval	重复执行的时间间隔

描述

函数pgagent.change用于修改任务的执行动作、调度时间等。

如果指定的in_jobname不存在则报错退出；否则修改名为in_jobname的任务的信息：

如果指定了in_what字段，就更新pga_jobstep表中名为in_jobname的任务的jstcode字段；

如果指定了in_next_date字段，就更新pga_job表中名为in_jobname的任务的jobnextrun字段；

如果指定了in_interval字段，就更新pga_schedule表中名为in_jobname的任务的jscrepeat_interval字段。

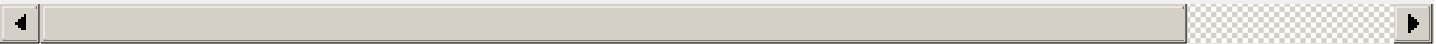
用法

#修改任务job1的调度规则，时间间隔改成每两天：

```
SELECT pgagent.change('job1', NULL, NULL, 'freq=daily;interval=2');
```

#修改任务job1的执行动作：

```
SELECT pgagent.change('job1',  
  'INSERT INTO pgagent.test VALUES ((SELECT max(a)+10 FROM  pgagent.test), now())', N
```



函数 interval

名称

```
pgagent.interval(  
  in_jobname text,  
  in_interval TEXT)
```

参数

参数	描述
in_jobname	任务名称
in_interval	重复执行的时间间隔

描述

函数pgagent.interval用于修改任务的时间间隔。

如果指定的in_jobname不存在则报错退出；否则更新名为in_jobname的任务对应调度规则的jscrepeat_interval字段。

in_interval是'freq=xx;interval=yy'形式的字符串。

支持的freq类型有：yearly、monthly、weekly、daily、hourly、minutely。

支持的interval范围是：数字1-99。

用法

#修改任务job1的时间间隔：

```
SELECT pgagent.interval('job1', 'freq=daily;interval=2');
```

函数 next_data

名称

```
pgagent.next_date(  
    in_jobname text,  
    in_next_date TIMESTAMP)
```

参数

参数	描述
in_jobname	任务名称
in_next_date	任务下次运行时间

描述

函数pgagent.next_date用于修改任务的调度时间。

如果指定的in_jobname不存在则报错退出；否则更新pga_job表中名为in_jobname的任务的jobnextrun字段。

用法

#修改任务job1的调度时间：

```
SELECT pgagent.next_date('job1', '2020-1-2 9:00:00');
```

函数 remove

名称

```
pgagent.remove(in_jobname text)
```

参数

参数 描述 in_jobname 任务名称

描述

函数pgagent.remove用于删除一个定时任务。包括任务包含的运行计划和执行步骤都会被删除。

如果指定的in_jobname不存在，则报错退出；否则删除名为in_jobname的任务，同时删除其关联的步骤和调度规则。

用法

#删除一个定时任务job1：

```
SELECT pgagent.remove('job1');
```

函数 run

名称

```
pgagent.run(in_jobname text)
```

参数

参数 描述 in_jobname 任务名称

描述

函数pgagent.run用于运行一个任务。

如果指定的in_jobname不存在则报错退出；否则更新pga_job表中名为in_jobname的任务的jobenabled字段为true，并更新其jobnextrun字段为当前时间。这样任务会在紧接着的轮询周期被调度运行。

用法

#运行任务job1：

```
SELECT pgagent.run('job1');
```

函数 submit

名称

```
pgagent.submit(  
    in_jobname text,  
    in_next_date timestamp with time zone,  
    in_jstkind char,  
    in_what TEXT,  
    in_jstconnstr text,  
    in_jscstart timestamp with time zone,  
    in_jscend timestamp with time zone,  
    in_interval text)
```

参数

参数	描述
in_job_name	任务名称
in_next_date	任务下次运行时间。
in_jobkind	步骤类型， b :批处理命令， s :SQL命令。
in_what	步骤动作，批处理命令或SQL命令内容。
in_jobconnstr	数据库连接串信息， dbname :要连接到的数据库名， host :服务器正在运行的机器的主机名， port :数据库服务端口， username :连接数据库的用户名， password :数据库用户的密码，形如： 'user=user1 password=***** host=h112 port=5432 dbname=testdb' 。

参数	描述
in_jsc_start	开始时间
in_jsc_end	结束时间
in_interval	重复执行的时间间隔

描述

函数pgagent.submit用于创建一个定时任务。包括任务名称、描述、运行计划和执行步骤等信息。提交任务后，系统会按其指定的调度时间执行该任务指定的动作。

注意：如果任务类型是批处理，如果执行失败，并不会像在数据库中执行SQL那样失败会回滚。这是设置此类任务时应考虑的地方。

如果给定的in_jobname在表pga_job中已存在则报错退出；否则创建名为in_jobname的新任务。大体步骤如下：

获取jobid并创建名为in_jobanme的任务（向pga_job表插入一条记录）；

获取jstid并创建步骤（向pga_jobstep表插入一条记录，其jstjobid为jobid）；

获取jscid并创建调度规则（向pga_schedule表插入一条记录，其jscjobid为jobid）。

in_interval采用优化时间间隔表示法：

in_interval='freq=x;interval=y'，x可以是yearly、monthly、weekly、daily、hourly、minutely其中的一个，y可以是1-99的数字。

用法

#创建一个定时任务job1，每天定时向表插入数据：

```
SELECT pgagent.submit('job1', now(), 's',  
'INSERT INTO pgagent.test VALUES ((SELECT max(a)+1 FROM pgagent.test), now())',  
'user=user1 password=***** host=h112 port=5432 dbname=testdb', '2020-1-1 15:30:00',  
'2021-1-1 15:29:59', 'freq=daily;interval=1');
```

#创建一个定时任务job2，如果任务要执行的SQL（in_what字段）包含单引号，要对其转义（因为最终会把这段SQL字符串存储在数据库的一个text类型字段中）：

```
SELECT pgagent.submit('job2', now(), 's',  
'INSERT INTO pgagent.test2 select generate_series(1,10), "aaaa;bbb;ccc;ddd;eee;"',  
'user=user1 password=***** host=h112 port=5432 dbname=testdb', '2020-1-1 15:30:00',  
'2021-1-1 15:29:59', 'freq=daily;interval=1');
```

#创建一个定时任务job3，执行指定的批处理程序，批处理程序在本地执行，数据库连接串为空即可：

```
SELECT pgagent.submit('job3', now(), 'b',  
'sh /tmp/action.sh', '', '2020-1-1 15:30:00', '2021-1-1 15:29:59',  
'freq=minutely;interval=2');
```

函数 what

名称

```
pgagent.what(  
    in_jobname text,  
    in_what TEXT)
```

参数

参数	描述
in_jobname	任务名称
in_what	步骤动作，批处理命令或SQL命令内容。

描述

函数pgagent.what用于修改任务的执行动作。

如果指定的in_jobname不存在则报错退出；否则更新名为in_jobname的任务对应的步骤的jstcode字段。

用法

#修改任务job1的执行动作：

```
SELECT pgagent.what('job1',  
    'INSERT INTO pgagent.test VALUES ((SELECT max(a)+10 FROM  pgagent.test), now())');
```

视图 ALL_JOBS

名称

pgagent.ALL_JOBS

参数

列名	类型	任务名称
JOB_NAME	TEXT	任务名称
ACTION_NAME	TEXT	步骤名称
ACTION_TYPE	CHAR	步骤类型，SQL或BATCH
ACTION_CODE	TEXT	执行指令
ACTION_DESCRIPTION	TEXT	连接信息
SCHEDULE_NAME	TEXT	调度规则名称
START_DATE	TIMESTAMP WITHOUT TIME ZONE	调度开始时间
END_DATE	TIMESTAMP WITHOUT TIME ZONE	调度结束时间
REPEAT_INTERVAL	TEXT	调度规则名称
ENABLED	TEXT	任务是否启用，TRUE启用，FALSE停用
COMMENTS	TEXT	任务备注

描述

视图pgagent.ALL_JOBS用于查看所有任务的详细信息，包括执行步骤和调度规则等。

用法

#查看所有任务信息：

```
postgres=# \x
```

Expanded display is on postgres=# SELECT * FROM pgagent.all_jobs ;-[
RECORD 1]---+----- JOB_NAME |
job1 ACTION_NAME | action1 ACTION_TYPE | SQL ACTION_CODE |
CREATE table tttt(a int, b text) ACTION_DEST | user=test password=*****
host=h112 port=5432 dbname=testdb SCHEDULE_NAME | schd1
START_DATE | 2020-01-01 15:30:00 END_DATE | 2030-01-01 15:29:59
REPEAT_INTERVAL | freq=minutely;interval=2 ENABLED | TRUE
COMMENTS | desc of job1

视图 ALL_ACTIONS

名称

```
pgagent.ALL_ACTIONS
```

参数

列名	类型	任务名称
ACTION_NAME	TEXT	名称
ACTION_TYPE	CHAR	类型，SQL或BATCH
ACTION_CODE	TEXT	执行指令
ACTION_DEST	TEXT	连接信息
ENABLED	BOOLEAN	是否启用，TRUE启用，FALSE停用
MAX_RUNS	INT	失败重试最大次数
COMMENTS	TEXT	备注

描述

视图pgagent.ALL_ACTIONS用于查看所有步骤的详细信息。

用法

#查看所有任务信息：

```
postgres=# SELECT * FROM pgagent.all_actions ;
-[ RECORD 1 ]+-----
ACTION_NAME | action1
ACTION_TYPE | SQL
ACTION_CODE | CREATE table ttttt(a int, b text)
ENABLED      | t
MAX_RUNS     | 100
MAX_FAILURES | 0
COMMENTS     | desc of act1
```

视图 ALL_SCHEDULES

名称

```
pgagent.ALL_SCHEDULES
```

参数

列名	类型	任务名称
SCHEDULE_NAME	TEXT	调度规则名称
SCHEDULE_TYPE	TEXT	只支持一种类型，CALENDAR日历法
START_DATE	TIMESTAMP WITHOUT TIME ZONE	调度开始时间

列名	类型	任务名称
END_DATE	TIMESTAMP WITHOUT TIME ZONE	调度结束时间
REPEAT_INTERVAL	TEXT	调度时间间隔
COMMENTS	TEXT	备注

描述

视图pgagent.ALL_SCHEDULES用于查看所有调度规则的详细信息。

用法

#查看所有任务信息：

```
postgres=# SELECT * FROM pgagent.all_schedules;
-[ RECORD 1 ]---+-----
SCHEDULE_NAME   | schd1
SCHEDULE_TYPE   | CALENDAR
START_DATE      | 2020-01-01 15:30:00
END_DATE        | 2030-01-01 15:29:59
REPEAT_INTERVAL | freq=minutely;interval=2
COMMENTS        | desc of schd 1
```

视图 ALL_JOB_LOG

名称

```
pgagent.ALL_JOB_LOG
```

参数

列名	类型	任务名称
LOG_ID	INT	日志ID
RUN_DATE	TIMESTAMP WITHOUT TIME ZONE	日志时间
JOB_NAME	TEXT	任务名称
STATUS	TEXT	任务执行状态， running 正在运行， success 成功， failed 失败， interval error 内部错误， aborted 中止。

描述

视图pgagent.ALL_JOB_LOG用于查看所有任务的执行日志。

用法

#查看所有任务的执行日志：

```
postgres=# select * from pgagent.all_job_log;
 LOG_ID |      RUN_DATE      | JOB_NAME | STATUS
-----+-----+-----+-----
      1 | 2020-01-03 11:05:38.800717 | job1    | success
      2 | 2020-01-03 11:07:03.848618 | job1    | failed
      3 | 2020-01-03 11:08:08.91924  | job1    | failed
      4 | 2020-01-03 11:10:14.031536 | job1    | failed
      5 | 2020-01-03 11:14:14.256666 | job1    | failed
      6 | 2020-01-03 11:22:14.681884 | job1    | failed
      7 | 2020-01-03 11:38:15.53979  | job1    | failed
      8 | 2020-01-03 12:10:17.2441   | job1    | failed
      9 | 2020-01-03 13:14:20.720161 | job1    | failed
(9 rows)
```

视图 ALL_JOB_RUN_DETAILS

名称

pgagent.ALL_JOB_RUN_DETAILS

参数

列名	类型	任务名称
LOG_ID	INT	日志ID
LOG_DATE	TIMESTAMP WITHO UT TIME ZONE	日志时间
JOB_NAME	TEXT	任务名称
STATUS	TEXT	任务执行状态，同pgagent.ALL_ JOB_LOG.STATUS。
ACTUAL_ST ART_DATE	TIMESTAMP WITHO UT TIME ZONE	任务步骤的实际执行时间
RUN_DURATI ON	TEXT	执行耗时
STORE_OUT PUT	TIMESTAMP WITHO UT TIME ZONE	错误信息

描述

视图pgagent.ALL_JOB_RUN_DETAILS用于查看所有任务的详细执行信息。

用法

#查看所有任务的详细执行信息：


```
postgres=# select * from pgagent.all_job_run_details;
 LOG_ID | LOG_DATE | JOB_NAME | STATUS | ACTUAL_START_DATE
-----+-----+-----+-----+-----
      1 | 2020-01-03 11:05:38.800717 | job1 | success | 2020-01-03 11:05:38.912351 | 00:00:00
      2 | 2020-01-03 11:07:03.848618 | job1 | failed | 2020-01-03 11:07:03.882275 | 00:00:00
      3 | 2020-01-03 11:08:08.91924 | job1 | failed | 2020-01-03 11:08:08.952876 | 00:00:00
      4 | 2020-01-03 11:10:14.031536 | job1 | failed | 2020-01-03 11:10:14.061155 | 00:00:00
      5 | 2020-01-03 11:14:14.256666 | job1 | failed | 2020-01-03 11:14:14.288496 | 00:00:00
      6 | 2020-01-03 11:22:14.681884 | job1 | failed | 2020-01-03 11:22:14.71571 | 00:00:00
      7 | 2020-01-03 11:38:15.53979 | job1 | failed | 2020-01-03 11:38:15.586201 | 00:00:00
      8 | 2020-01-03 12:10:17.2441 | job1 | failed | 2020-01-03 12:10:17.292575 | 00:00:00
      9 | 2020-01-03 13:14:20.720161 | job1 | failed | 2020-01-03 13:14:20.763181 | 00:00:00
(9 rows)
```

附加提供的模块

本节介绍了MPP数据库安装中可用的其他模块。 这些模块可能来自PostgreSQL或MPP。

contrib模块通常打包为扩展。 您可以使用[CREATE EXTENSION](#)命令在数据库中注册模块。 您可以使用[DROP EXTENSION](#)从数据库中删除模块。

安装了以下MPP数据库和PostgreSQL contrib模块； 有关使用说明，请参阅链接的模块文档。

- [citext](#) - 提供不区分大小写，可识别多字节的文本数据类型。
- [dblink](#) - 提供与其他MPP数据库的连接。
- [fuzzystrmatch](#) - 确定字符串之间的相似性和差异。
- [gp_sparse_vector](#) - 实现MPP数据库数据类型，该数据类型使用零压缩存储

来帮助进行浮点计算。

- [hstore](#) - 提供一种数据类型，用于在单个PostgreSQL值中存储键/值对的集合。
- [orafce](#) - 提供MPP数据库特定的Oracle SQL兼容性函数。
- [pageinspect](#) - 低级检查数据库页面内容的函数；仅对超级用户可用。
- [pgcrypto](#) - 为MPP数据库提供加密函数。

Parent topic: [MPP数据库参考手册](#)

citext

citext 模块提供了一个不区分大小写的字符串数据类型, citext。本质上，citext在比较值时在内部调用lower 函数。除此之外，它的操作几乎与text相同

对文本值执行不区分大小写匹配的方法是曾经是在比较值时使用 lower 函数，例如

```
SELECT * FROM tab WHERE lower(col) = LOWER(?);
```

这种方法可行, 但是有缺点：

- 它会使SQL语句变得冗长, 而且总是在列查询值中同时使用lower函数。
- 除非已经使用lower函数创建一个函数索引，否则该函数不适用于索引。

citext数据类型允许您在SQL查询中不调用lower函数，并且可以在citext类型的列上创建不区分大小写的主键。类似于ext类型，citext是本地化识别匹配, 这意

意味着大写和小写字母的匹配依赖于数据库LC_CTYPE设置规则。这和在查询中使用lower函数功能相同，但它由数据类型透明地完成，因此您不必在查询中执行任何特殊操作。

安装 citext

在使用citext类型之前，必须在要使用该类型的每个数据库中注册扩展名：

```
$ psql -d testdb -c "CREATE EXTENSION citext"
```

使用citext类型

以下是一个定义一个citext类型数据列的简单的例子：

```
CREATE TABLE users (  
    id bigint PRIMARY KEY,  
    nick CITEXT NOT NULL,  
    pass TEXT NOT NULL  
) DISTRIBUTED BY (id);  
  
INSERT INTO users VALUES (1, 'larry', md5(random()::text) );  
INSERT INTO users VALUES (2, 'Tom', md5(random()::text) );  
INSERT INTO users VALUES (3, 'Damian', md5(random()::text) );  
INSERT INTO users VALUES (4, 'NEAL', md5(random()::text) );  
INSERT INTO users VALUES (5, 'Bjørn', md5(random()::text) );  
  
SELECT * FROM users WHERE nick = 'Larry';
```

SELECT语句将返回一个元组，尽管nick列被设置为larry并且查询指定为Larry

字符串比较操作

citext通过将每个字符串的每个字符转换为小写（和调用lower函数相似）然后正常比较结果。如果调用lower函数后两个字符串的值一样，则认为两个字符串相等。

为了尽可能地模拟不区分大小写的排序规则,citext提供了许多字符串处理运算符以及运算函数的特殊版本。例如正则表达式~和~*在应用到citext时表现出相同的操作：他们都是大小写不敏感的。相同的特性也表现在!~和!~*上，以及LIKE运算符~~，~~*，!~~和!~~*。如果您需要大小写敏感，可以将运算符的参数强制转换为text。

类似，如果它们的参数为citext，以下函数执行不区分大小写匹配：

- regexp_match()
- regexp_matches()
- regexp_replace()
- regexp_split_to_array()
- regexp_split_to_table()
- replace()
- split_part()
- strpos()
- translate()

对于regexp函数，如果想要大小写敏感，可以声明“c”标志以强制大小写匹配。如果您想要大小写敏感操作，则在使用其中一个函数之前必须强制转换为text类型。

局限性

- `citext`类型的折叠操作依赖于数据库的`LC_CTYPE`设置。因此，在创建数据库时，应确定如何比较值。通过在Unicode标准定义的术语中并不是真正大小写不敏感的。实际上，只要您对您的排序规则感到满意，您应该对`citext`的比较感到满意。但是，如果您的数据库中存储了不同语言的数据，则即使排序规则适用于其他语言，有一种语言的用户也可能会发现其查询结果与预期不符。
- `citext`不如`text`高效，因为运算符函数和使用B树的比较函数必须拷贝一份数据并将它们转换为小写进行比较。但是，它比使用`lower`执行不区分大小写的匹配效率稍高一些。
- 如果你需要数据比较在某些情况中大小写敏感但其他情况中大小写不敏感`citext`不一定是一个最好的选择，建议的做法是使用`text`类型，然后在需要大小写不敏感的情况下手动使用`lower`函数。这种方法适用于较少需要大小写不敏感比较的情况下。如果您需要的大小写不敏感比较的频率明显大于大小写敏感，您可以将数据存储在`citext`类型，当想要进行大小写敏感比较时将数据类型强制转换为`text`类型。如果您希望在两种情况下都能高效的进行，您需要建立两个索引。
- 包含`citext`运算的模式必须在当前的`search_path`中（通常是`public`）如果不是这样的话，请调用大小写敏感的`text`运算。

dblink

`dblink`模块支持从数据库会话中连接到其他MPP数据库数据库。这些数据库可以驻留在相同的MPP数据库系统中，也可以驻留在远程系统中。

MPP数据库支持MPP数据库安装中具有相同主要版本号的数据库之间的`dblink`连接。您还可以使用`dblink`连接到使用兼容`libpq`库的其他MPP数据库安

装。

Note: dblink适用于数据库用户在其他数据库中执行简短的临时查询。dblink不能替代外部表或gpccopy之类的管理工具。

MPP数据库dblink模块是PostgreSQL dblink模块的修改版本。在MPP数据库中使用该模块时，存在一些限制。

安装和注册模块

当您安装MPP数据库时，将安装dblink模块。在使用模块中定义的任何函数之前，必须在要使用这些函数的每个数据库中注册dblink扩展。

MPP数据库注意事项

在此版本的MPP数据库中，修改表数据的语句不能使用命名或隐式dblink连接。而是必须直接在dblink()函数中提供连接字符串。例如：

```
gpadmin=# CREATE TABLE testdbllocal (a int, b text) DISTRIBUTED BY (a);  
CREATE TABLE  
gpadmin=# INSERT INTO testdbllocal select * FROM dblink('dbname=postgres', 'SELECT  
INSERT 0 2
```

dblink的MPP数据库版本禁用以下异步函数：

- dblink_send_query()
 - dblink_is_busy()
 - dblink_get_result()
-

使用dblink

以下过程确定了在MPP数据库中配置和使用dblink的基本步骤。这些示例使用dblink_connect()创建与数据库的连接，并使用dblink()执行SQL查询。

1. 首先创建一个示例表以使用dblink函数进行查询。这些命令在postgres数据库中创建一个表，稍后您将使用dblink从testdb数据库中查询该表：

```
$ psql -d postgres
psql (9.4.20)
Type "help" for help.

postgres=# CREATE TABLE testdblink (a int, b text) DISTRIBUTED BY (a);
CREATE TABLE
postgres=# INSERT INTO testdblink VALUES (1, 'Cheese'), (2, 'Fish');
INSERT 0 2
postgres=# \q
$
```

2. 以超级用户身份登录到另一个数据库。在此示例中，超级用户gpadmin登录到数据库testdb。如果dblink函数尚不可用，请在数据库中注册dblink扩展：

```
$ psql -d testdb
psql (9.4beta1)
Type "help" for help.

testdb=# CREATE EXTENSION dblink;
CREATE EXTENSION
```

3. 使用dblink_connect()函数创建与另一个数据库的隐式或命名连接。您提供的连接字符串应该是libpq样式的关键字/值字符串。本示例创建一个名为mylocalconn的连接到本地MPP数据库系统上的postgres数据库：

```
testdb=# SELECT dblink_connect('mylocalconn', 'dbname=postgres user=gpadmin');
dblink_connect
-----
OK
(1 row)
```

Note: 如果未指定user，则在启动MPP数据库时，dblink_connect()将使用PGUSER环境变量的值。如果未设置PGUSER，则默认值为启动MPP数据库的系统用户。

4. 使用dblink()函数可使用已配置的连接查询数据库。请记住，此函数返回记录类型，因此您必须分配dblink()查询中返回的列。例如，以下命令使用命名连接来查询您先前创建的表：

```
testdb=# SELECT * FROM dblink('mylocalconn', 'SELECT * FROM testdblink') AS dbli
id | product
---+-----
 1 | Cheese
 2 | Fish
(2 rows)
```

要以另一个用户身份连接到本地数据库，请在连接字符串中指定该用户。本示例以用户test_user的身份连接到数据库。使用dblink_connect()，超级用户无需指定密码即可创建与另一个本地数据库的连接。

```
testdb=# SELECT dblink_connect('localconn2', 'dbname=postgres user=test_user');
```

要建立与远程数据库系统的连接，请在连接字符串中包含主机和密码信息。例如，要创建到远程系统的隐式dblink连接：

```
testdb=# SELECT dblink_connect('host=remotehost port=5432 dbname=postgres user=g
```


作为非超级用户使用dblink

要使用dblink_connect()与数据库建立连接，非超级用户必须在连接字符串中包含主机，用户和密码信息。即使连接到本地数据库，也必须包括主机，用户和密码信息。例如，用户test_user可以使用以下命令创建到本地系统mdw的dblink连接：

```
testdb=> SELECT dblink_connect('host=mdw port=5432 dbname=postgres user=test_user');
```

如果非超级用户需要创建不需要密码的dblink连接，则可以使用dblink_connect_u()函数。dblink_connect_u()函数与dblink_connect()相同，区别在于它允许非超级用户创建不需要密码的连接。

最初安装dblink_connect_u()时，它具有从PUBLIC撤消的所有特权，因此除超级用户外，它无法调用。在某些情况下，将dblink_connect_u()的EXECUTE权限授予被认为可信任的特定用户可能是适当的，但是应格外小心。

Warning: 如果MPP数据库系统为用户配置了不涉及密码的身份验证方法，那么当非超级用户执行dblink_connect_u()时，可能会冒充他人，并随后升级特权。dblink连接似乎源自该函数指定的用户。例如，非超级用户可以执行dblink_connect_u()并指定使用trust认证配置的用户。

同样，即使dblink连接需要密码，也可以从服务器环境中提供该密码，例如属于服务器用户的~/.pgpass文件。建议所有属于服务器用户的~/.pgpass文件都不要包含任何指定通配符主机名的记录。

1. 作为超级用户，对用户数据库中的dblink_connect_u()函数授予EXECUTE特权。本示例向具有创建隐式或命名dblink连接的签名的函数的非超级用户test_user授予特权。

```
testdb=# GRANT EXECUTE ON FUNCTION dblink_connect_u(text) TO test_user;  
testdb=# GRANT EXECUTE ON FUNCTION dblink_connect_u(text, text) TO test_user;
```

2. 现在，test_user无需密码即可创建到另一个本地数据库的连接。例如，test_user可以登录到testdb数据库并执行此命令以创建一个名为testconn的连接到本地postgres数据库。

```
testdb=> SELECT dblink_connect_u('testconn', 'dbname=postgres user=test_user');
```

Note: 如果未指定user，则在启动MPP数据库时，dblink_connect_u()将使用PGUSER环境变量的值。如果未设置PGUSER，则默认值为启动MPP数据库的系统用户。

3. test_user可以使用dblink()函数通过dblink连接执行查询。例如，此命令使用在上一步中创建的名为testconn的dblink连接。test_user必须具有对该表的适当访问权限。

```
testdb=> SELECT * FROM dblink('testconn', 'SELECT * FROM testdblink') AS dbltab(i
```

附加模块文档

有关此模块中各个函数的详细信息，请参考PostgreSQL文档中的[dblink](#)。

fuzzystmatch

fuzzystmatch模块提供了根据各种算法确定字符串之间的相似性和距离的函数。

MPP数据库fuzzystrmatch模块等同于PostgreSQL fuzzystrmatch模块。该模块没有MPP数据库或MPP特定的注意事项。

安装和注册模块

当您安装MPP数据库时，将安装fuzzystrmatch模块。在使用模块中定义的任何函数之前，必须在要使用这些函数的每个数据库中注册fuzzystrmatch扩展。

模块文档

有关此模块中各个函数的详细信息，请参见PostgreSQL文档中的[fuzzystrmatch](#)。

hstore

hstore模块实现了一种数据类型，用于在单个MPP数据库数据字段中存储（键，值）对的集合。这在各种情况下很有用，例如具有许多很少检查的属性的行或半结构化数据。

在当前的实现中，键和值字符串的长度都不能超过65535个字节。如果超出此限制，将引发错误。这些最大长度可能会在将来的版本中更改。

安装hstore

在使用hstore数据类型和函数之前，请在要查询其他数据库的每个数据库中运行安装脚本\$GPHOME/share/postgresql/contrib/hstore.sql：

```
$ psql -d testdb -f $GPHOME/share/postgresql/contrib/hstore.sql
```

hstore外部表示

hstore值的文本表示形式包括零个或多个`key=>value`项，以逗号分隔。例如：

```
k => v  
foo => bar, baz => whatever  
"1-a" => "anything at all"
```

项目的顺序不重要（并且可能不会在输出中复制）。项目之间或`=>`符号周围的空格将被忽略。如果键或值包含空格，逗号，`=`或`>`，请使用双引号。要在键或值中包含双引号或反斜杠，请在其前面加上另一个反斜杠。（请记住，根据`standard_conforming_strings`的设置，您可能需要在SQL文字字符串中加双反斜杠。）

值（但不是键）可以是SQL NULL。这表示为

```
key => NULL
```

NULL关键字不区分大小写。同样，如果要将字符串null视为普通数据值，请使用双引号引起来。

当前，即使不是绝对必要，双引号始终用于在输出中包含键和值字符串。

hstore运算符和函数

Table 1. hstore运算符

运算符	描述	示例	结果
hstore -> text	获取键的值（如果不存在，则为null）	'a=>x, b=>y'::hstore -> 'a'	x
text => text	制作单项hstore	'a' => 'b'	"a"=>"b"
hstore hstore	连接	'a=>b, c=>d'::hstore 'c=>x, d=>q'::hstore	"a"=>"b", "c"=>"x", "d"=>"q"
hstore ? text	hstore是否包含键？	'a=>1'::hstore ? 'a'	t
hstore @> hstore	左操作数是否包含右操作数？	'a=>b, b=>1, c=>NULL'::hstore @> 'b=>1'	t
hstore <@ hstore	左操作数是否包含于右操作数？	'a=>c'::hstore <@ 'a=>b, b=>1, c=>NULL'	f

Note: 不推荐使用=>运算符，并且可能在以后的版本中将其删除。请改用hstore(text, text)函数。

Table 2. hstore函数

函数	返回类型	描述	示例	结果
hstore(text, text)	hstore	制作单项hstore	hstore('a', 'b')	"a"=>"b"
akeys(hstore)	text[]	以数组形式获取hstore的键	akeys('a=>1,b=>2')	{a,b}
skeys(hstore)	setof text	以集合形式获取hstore的键	skeys('a=>1,b=>2')	a b

函数	返回类型	描述	示例	结果
avals(hstore)	text[]	以数组形式获取 hstore 的值	avals('a=>1,b=>2')	{1,2}
svals(hstore)	setof text	以集合形式获取 hstore 的值	svals('a=>1,b=>2')	1 2
each(hstore)	setof (key text, value text)	以集合形式获取 hstore 的键值	select * from each('a=>1,b=>2')	key value ----+----- a 1 b 2
exist(hstore,text)	boolean	hstore 是否包含键？	exist('a=>1','a')	t
defined(hstore,text)	boolean	对于键 hstore 是否包含非空值？	defined('a=>NULL','a')	f
delete(hstore,text)	hstore	删除匹配键的所有项	delete('a=>1,b=>2','b')	"a"=>"1"

Indexes

hstore 对 @> 和 ? 运算符有索引支持。 您可以使用 GiST 或 GIN 索引类型。 例如：

```
CREATE INDEX hidx ON testhstore USING GIST(h);  
  
CREATE INDEX hidx ON testhstore USING GIN(h);
```

示例

添加键，或使用新键更新现有键：

```
UPDATE tab SET h = h || ('c' => '3');
```

删除键：

```
UPDATE tab SET h = delete(h, 'k1');
```

统计

hstore类型由于其固有的自由度，可能包含许多不同的键。检查有效键是应用程序的任务。下面的示例演示了几种用于检查键和获取统计信息的技术。

简单示例：

```
SELECT * FROM each('aaa=>bq, b=>NULL, ""=>1');
```

使用表：

```
SELECT (each(h)).key, (each(h)).value INTO stat FROM testhstore;
```

在线统计：

```
SELECT key, count(*) FROM
  (SELECT (each(h)).key FROM testhstore) AS stat
GROUP BY key
ORDER BY count DESC, key;
  key  | count
-----+-----
```

line		883
query		207
pos		203
node		202
space		197
status		195
public		194
title		190
org		189

.....

orafce

描述MPP数据库中的Oracle兼容SQL函数。 这些函数针对PostgreSQL，但也可以在MPP数据库中使用。

关于MPP数据库的Oracle兼容性函数

用于MPP数据库的Oracle兼容性函数基于<https://github.com/orafce/orafce>上的开源Orafce项目。

可以在[MPP数据库开源项目](#)的gpcontrib/orafce目录中找到经过修改的MPP数据

库的Orafce源文件。源文件包括Orafce 3.6.1版本和对[3af70a28f6](#)的其他提交。

以下函数默认情况下在MPP数据库中可用，并且不需要安装Oracle兼容性函数：

- sinh
- tanh
- cosh
- decode (更多信息请参考[Oracle和MPP的实现差异](#)。)

有关使用Oracle兼容性函数的信息，请参阅Orafce项目主页上的文档，网址为<https://github.com/orafce/orafce>。Orafce项目中文档的源文件也包含在MPP数据库开源项目中。

安装Oracle兼容性函数

Note: 始终使用MPP数据库版本随附的Oracle兼容性函数模块。在升级到新的MPP数据库版本之前，请从每个数据库中卸载兼容性函数，然后，在升级完成后，从新的MPP数据库版本中重新安装兼容性函数。有关升级先决条件和步骤，请参阅MPP数据库发行说明。

使用CREATE EXTENSION SQL命令在每个数据库中安装Oracle兼容性函数。

```
$ psql -d db_name -c 'CREATE EXTENSION orafce;'
```

Note: 某些Oracle兼容性函数位于oracle模式中。要访问它们，请为数据库设置搜索路径以包括oracle模式名称。例如，此命令设置数据库的默认搜索路径以包括oracle模式：

```
ALTER DATABASE db_name SET search_path = "$user", public, oracle;
```

要卸载Oracle兼容性函数，请使用DROP EXTENSION SQL命令删除orafce扩展。

```
$ psql -d db_name -c 'DROP EXTENSION orafce;'
```

如果要从MPP数据库5.x发行版中卸载兼容性函数，请改用以下命令：

```
$ psql -d db_name -f $GPHOME/share/postgresql/contrib/uninstall_orafunc.sql
```

Oracle和MPP的实现差异

MPP数据库中兼容性函数的实现与Oracle实现之间存在一些差异。如果使用验证脚本，则输出可能与Oracle中的输出不完全相同。一些区别如下：

- Oracle执行十进制舍入，MPP数据库则不：
 - 2.00在Oracle中变成2
 - MPP数据库中的2.00仍为2.00
- 提供的Oracle兼容性函数以不同的方式处理隐式类型转换。例如，使用decode函数：

```
decode(expression, value, return [,value, return]...  
      [, default])
```

Oracle在比较之前自动将表达式和每个值转换为第一个值的数据类型。

Oracle自动将`return`转换为与第一个结果相同的数据类型。

MPP数据库实现将`return`和`default`限制为相同的数据类型。如果可以将值的数据类型转换为表达式的数据类型，则表达式和值可以是不同的类型。这是隐式完成的。否则，`decode`将失败，并出现invalid input syntax错误。例如：

```

SELECT decode('a','M',true,false);
CASE
-----
f
(1 row)
SELECT decode(1,'M',true,false);
ERROR: Invalid input syntax for integer: "M"
LINE 1: SELECT decode(1,'M',true,false);

```

- bigint格式的数字在Oracle中以科学计数法显示，但在MPP数据库中不是这样：
 - 9223372036854775在Oracle中显示为9.2234E + 15
 - 9223372036854775在MPP数据库中仍为9223372036854775
- Oracle中的默认日期和时间戳格式与MPP数据库中的默认格式不同。如果执行以下代码：

```

CREATE TABLE TEST(date1 date, time1 timestamp, time2
                    timestamp with time zone);
INSERT INTO TEST VALUES ('2001-11-11','2001-12-13
                          01:51:15','2001-12-13 01:51:15 -08:00');
SELECT DECODE(date1, '2001-11-11', '2001-01-01') FROM TEST;

```

MPP数据库返回该行，但是Oracle不返回任何行。

Note: Oracle中返回该行的正确语法是：

```

SELECT DECODE(to_char(date1, 'YYYY-MM-DD'), '2001-11-11',
              '2001-01-01') FROM TEST

```

以下是将Oracle兼容函数与PostgreSQL和MPP数据库结合使用的区别。

- 从MPP数据库Oracle兼容性函数中删除了decode()函数。MPP数据库解析器将decode() - 函数调用转换为case语句。
- Oracle兼容性函数dbms_pipe软件包中的函数仅在MPP数据库master主机上

执行。

- 没有为MPP数据库实现Oracle兼容性函数dbms_alert软件包中的函数。
- Orafce项目中的升级脚本不适用于MPP数据库。

pgcrypto

MPP数据库安装有一个可选的加密/解密函数模块pgcrypto。pgcrypto函数允许数据库管理员以加密形式存储数据的某些列。这为敏感数据增加了一层额外的保护，因为没有加密密钥的任何人都无法读取以加密形式存储在MPP数据库中的数据，也不能直接从磁盘读取该数据。

Note: pgcrypto函数在数据库服务器内部运行，这意味着所有数据和密码都以明文形式在pgcrypto和客户端应用程序之间传输。为了获得最佳安全性，请考虑在客户端和MPP主服务器之间使用SSL连接。

有关此模块中各个函数的更多信息，请参见PostgreSQL文档中的[pgcrypto](#)。

MPP Partner Connector API

使用MPP Partner Connector API (GPPC API)，您可以使用C和C++编程语言编写可移植的MPP数据库用户定义函数 (UDF)。使用GPPC API开发的函数不需要重新编译或修改即可使用较旧或较新的MPP数据库版本。

可以使用MPP数据库中的SQL调用您写入GPPC API的函数。API提供了一组函数和宏，可用于通过服务器编程接口（SPI）发出SQL命令，操作简单和复合数据类型函数参数和返回值，管理内存和处理数据。

您将使用GPPC API开发的C/C++函数编译到共享库中。在MPP数据库群集中安装共享库并将GPPC函数注册为SQL UDF之后，MPP数据库用户可以使用GPPC函数。

Note: MPP Partner Connector支持MPP数据库版本4.3.5.0及更高版本。

本主题包含以下信息：

- [使用GPPC API](#)
 - [要求](#)
 - [头文件和库文件](#)
 - [数据类型](#)
 - [函数声明，参数和结果](#)
 - [内存处理](#)
 - [使用可变长度文本类型](#)
 - [错误报告和记录](#)
 - [SPI函数](#)
 - [关于元组描述符和元组](#)
 - [Set-Returning函数](#)
 - [表函数](#)
 - [限制](#)
 - [样例代码](#)
 - [使用PGXS构建GPPC共享库](#)
 - [使用MPP数据库注册GPPC函数](#)
 - [打包和部署注意事项](#)
 - [GPPC文本函数示例](#)
 - [GPPC Set-Returning函数示例](#)
-

Parent topic: [MPP数据库参考手册](#)

使用GPPC API

GPPC API与PostgreSQL定义的C语言函数共享一些概念。有关开发C语言函数的详细信息，请参阅PostgreSQL文档中的[C语言函数](#)。

GPPC API是一个包装器，它使C/C++函数可以在MPP数据库中调用SQL。这个包装器通过API定义的函数和宏对表和数据操作以及SPI操作进行规范化来屏蔽您从MPP数据库更改中编写的GPPC函数。

GPPC API包括以下函数和宏：

- 对基础和复合数据类型进行操作。
- 处理函数参数和返回值。
- 分配和释放内存。
- 记录并向客户报告错误。
- 发出SPI查询。
- 返回一个表或一组行。
- 将表作为函数输入参数。

要求

使用GPPC API进行开发时：

- 您必须在具有与MPP数据库主机相同的硬件和软件体系结构的系统上开发代码。
 - 您必须使用C或C++编程语言编写GPPC函数。
-

- 函数代码必须使用GPPC API，数据类型和宏。
- 函数代码不得使用PostgreSQL C语言函数API，头文件，函数或宏。
- 函数代码不能#include postgres.h头文件或使用PG_MODULE_MAGIC。
- 您必须仅使用GPPC包装的内存函数来分配和释放内存。 请参阅[内存处理](#)。
- 目标文件中的符号名称不得相互冲突，也不得与MPP数据库服务器中定义的符号冲突。 如果收到此类错误消息，则必须重命名函数或变量。

头文件和库文件

GPPC头文件和库文件安装在\$GPHOME中：

- \$GPHOME/include/gppc.h - 主要的GPPC头文件
- \$GPHOME/include/gppc_config.h - 定义GPPC版本的头文件
- \$GPHOME/lib/libgppc.[a, so, so.1, so.1.2] - GPPC存档和共享库

数据类型

您创建的GPPC函数将对驻留在MPP数据库中的数据进行操作。 GPPC API包含等效MPP数据库SQL数据类型的数据类型定义。 您必须在GPPC函数中使用这些类型。

GPPC API定义了可用于表示任何GPPC类型的通用数据类型。 此数据类型名为GppcDatum，定义如下：

```
typedef int64_t GppcDatum;
```

下表标识了每个GPPC数据类型以及它映射到的SQL类型。

SQL类型	GPPC类型	GPPC类型的Oid
boolean	GppcBool	GppcOidBool
char (single byte)	GppcChar	GppcOidChar
int2/smallint	GppcInt2	GppcOidInt2
int4/integer	GppcInt4	GppcOidInt4
int8/bigint	GppcInt8	GppcOidInt8
float4/real	GppcFloat4	GppcOidFloat4
float8/double	GppcFloat8	GppcOidFloat8
text	*GppcText	GppcOidText
varchar	*GppcVarChar	GppcOidVarChar
char	*GppcBpChar	GppcOidBpChar
bytea	*GppcBytea	GppcOidBytea
numeric	*GppcNumeric	GppcOidNumeric
date	GppcDate	GppcOidDate
time	GppcTime	GppcOidTime
timetz	*GppcTimeTz	GppcOidTimeTz
timestamp	GppcTimestamp	GppcOidTimestamp
timestampTz	GppcTimestampTz	GppcOidTimestampTz
anytable	GppcAnyTable	GppcOidAnyTable
oid	GppcOid	

GPPC API专门处理文本，数字和时间戳数据类型，提供对这些类型进行操作的函数。

示例GPPC基本数据类型声明：


```
GppcText      message;
GppcInt4      arg1;
GppcNumeric   total_sales;
```

GPPC API定义了通用GppcDatum类型和GPPC特定类型之间进行转换的函数。例如，要从整数转换为datum：

```
GppcInt4 num = 13;
GppcDatum num_dat = GppcInt4GetDatum(num);
```

符合类型

复合数据类型表示行或记录的结构，由字段名称列表及其数据类型组成。该结构信息通常被称为元组描述符。复合类型的实例通常称为元组或行。元组没有固定的布局，可以包含空字段。

GPPC API提供了一个界面，您可以使用该界面定义元组的结构，访问和设置元组。当GPPC函数将表作为输入参数或返回表或记录类型集时，将使用此接口。本主题后面将介绍使用表中的元组和设置返回函数。

函数声明，参数和结果

GPPC API依赖于宏来声明函数并简化函数参数和结果的传递。 这些宏包括：

任务	宏签名	描述
使函数SQL-可调用	GPPC_FUNCTION_INFO(<i>function_name</i>)	粘贴使函数 <i>function_name</i> SQL-可调用。

任务	宏签名	描述
声明一个函数	GppcDatum <i>function_name</i> (GPPC_FUNCTION_ARGS)	声明名为 <i>function_name</i> 的GPPC函数; 每个函数都必须具有相同的签名。
返回参数的数量	GPPC_NARGS()	返回传递给函数的参数数量。
获取参数	GPPC_GETARG_<ARGTYPE>(arg_num)	获取参数编号arg_num的值（从0开始），其中<ARGTYPE>标识参数的数据类型。例如，GPPC_GETARG_FLOAT8(0)。
获取并创建文本类型参数的副本	GPPC_GETARG_<ARGTYPE>_COPY(arg_num)	获取并复制参数号arg_num的值（从0开始）。<ARGTYPE>标识文本类型（text，varchar，bpchar，bytea）。例如，GPPC_GETARG_BYTEA_COPY(1)。
确定参数是否为NULL	GPPC_ARGISNULL(arg_num)	返回参数编号arg_num是否为NULL。
返回结果	GPPC_RETURN_<ARGTYPE>(return_val)	返回值return_val，其中<ARGTYPE>标识返回值的数据类型。例如，GPPC_RETURN_INT4(131)。

定义和实现GPPC函数时，必须使用上面标识的两个声明使用GPPC API声明它。例如，要声明名为add_int4s()的GPPC函数：

```
GPPC_FUNCTION_INFO(add_int4s);
GppcDatum add_int4s(GPPC_FUNCTION_ARGS);

GppcDatum
add_int4s(GPPC_FUNCTION_ARGS)
{
    // code here
}
```

如果`add_int4s()`函数接受两个`int4`类型的输入参数，则使用`GPPC_GETARG_INT4(arg_num)`宏来访问参数值。参数索引从0开始。例如：

```
GppcInt4 first_int = GPPC_GETARG_INT4(0);
GppcInt4 second_int = GPPC_GETARG_INT4(1);
```

如果`add_int4s()`返回两个输入参数的总和，则使用`GPPC_RETURN_INT8(return_val)`宏来返回此总和。例如：

```
GppcInt8 sum = first_int + second_int;
GPPC_RETURN_INT8(sum);
```

完整的GPPC函数：

```
GPPC_FUNCTION_INFO(add_int4s);
GppcDatum add_int4s(GPPC_FUNCTION_ARGS);

GppcDatum
add_int4s(GPPC_FUNCTION_ARGS)
{
    // get input arguments
    GppcInt4 first_int = GPPC_GETARG_INT4(0);
    GppcInt4 second_int = GPPC_GETARG_INT4(1);

    // add the arguments
    GppcInt8 sum = first_int + second_int;

    // return the sum
    GPPC_RETURN_INT8(sum);
}
```

内存处理

GPPC API提供用于分配和释放内存的函数，包括文本内存。必须将这些函数用

于所有内存操作。

函数名	描述
<code>void *GppcAlloc(size_t num)</code>	分配 <code>num</code> 个字节的未初始化内存。
<code>void *GppcAlloc0(size_t num)</code>	分配 <code>num</code> 个字节的初始化为0内存。 。
<code>void *GppcRealloc(void *ptr, size_t num)</code>	调整预分配的内存大小。
<code>void GppcFree(void *ptr)</code>	释放分配的内存。

分配内存后，可以使用`memcpy()`等系统函数来设置数据。

以下示例分配`GppcDatum`数组并将数组设置为函数输入参数的`datum`版本：

```
GppcDatum *values;
int attnum = GPPC_NARGS();

// allocate memory for attnum values
values = GppcAlloc( sizeof(GppcDatum) * attnum );

// set the values
for( int i=0; i<attnum; i++ ) {
    GppcDatum d = GPPC_GETARG_DATUM(i);
    values[i] = d;
}
```

为GPPC函数分配内存时，可以在当前上下文中分配它。 GPPC API包括返回，创建，切换和重置内存上下文的函数。

函数名	描述
<code>GppcMemoryContext GppcGetCurrentMemoryContext(void)</code>	返回当前内存上下文。

函数名	描述
GppcMemoryContext GppcMemoryContextCreate(GppcMemoryContext <i>parent</i>)	在 <i>parent</i> 下创建新的内存上下文。
GppcMemoryContext GppcMemoryContextSwitchTo(GppcMemoryContext <i>context</i>)	切换到内存上下文 <i>context</i> 。
void GppcMemoryContextReset(GppcMemoryContext <i>context</i>)	在内存上下文 <i>context</i> 中重置（释放）内存。

MPP数据库通常在每元组上下文中调用一个SQL调用的函数，它在每次服务器后端处理表行时创建和删除。不要假设在当前内存上下文中分配的内存存在多个函数调用中可用。

使用可变长度文本类型

GPPC API支持可变长度文本，varchar，空白填充和字节数组类型。在对这些数据类型进行操作时，必须使用GPPC API提供的函数。GPPC API中提供的可变文本操作函数包括为其分配内存，确定字符串长度，获取字符串指针以及访问这些类型的函数：

函数名	描述
GppcText GppcAllocText(size_t <i>len</i>) GppcVarChar GppcAllocVarChar(size_t <i>len</i>) GppcBpChar GppcAllocBpChar(size_t <i>len</i>) GppcBytea GppcAllocBytea(size_t <i>len</i>)	为不同长度类型分配 <i>len</i> 个字节的内存。

函数名	描述
size_t GppcGetTextLength(GppcText s) size_t GppcGetVarCharLength(GppcVarChar s) size_t GppcGetBpCharLength(GppcBpChar s) size_t GppcGetByteaLength(GppcBytea b)	返回内存块中的字节数。
char *GppcGetTextPointer(GppcText s) char *GppcGetVarCharPointer(GppcVarChar s) char *GppcGetBpCharPointer(GppcBpChar s) char *GppcGetByteaPointer(GppcBytea b)	返回一个指向内存块头部的字符串指针。该字符串不以空值终止。
char *GppcTextGetCString(GppcText s) char *GppcVarCharGetCString(GppcVarChar s) char *GppcBpCharGetCString(GppcBpChar s)	返回一个指向内存块头部的字符串指针。该字符串以空值终止。

函数名	描述
GppcText *GppcCStringGetText(const char *s)	从字符串构建变长类型。
GppcVarChar *GppcCStringGetVarChar(const char *s)	
GppcBpChar *GppcCStringGetBpChar(const char *s)	

GppcGet<VLEN_ARGTYPE>Pointer()函数返回的内存可能指向实际的数据库内容。请勿修改内存内容。GPPC API提供了在需要时为这些类型分配内存的函数。分配内存后，可以使用memcpy()等系统函数来设置数据。

以下示例处理文本输入参数，并为文本字符串连接操作分配和设置结果内存：

```
GppcText first_textstr = GPPC_GETARG_TEXT(0);
GppcText second_textstr = GPPC_GETARG_TEXT(1);

// determine the size of the concatenated string and allocate
// text memory of this size
size_t arg0_len = GppcGetTextLength(first_textstr);
size_t arg1_len = GppcGetTextLength(second_textstr);
GppcText retstring = GppcAllocText(arg0_len + arg1_len);

// construct the concatenated return string; copying each string
// individually
memcpy(GppcGetTextPointer(retstring), GppcGetTextPointer(first_textstr), arg0_len);
memcpy(GppcGetTextPointer(retstring) + arg0_len, GppcGetTextPointer(second_textstr), arg1
```

错误报告和记录

GPPC API提供错误报告和日志记录函数。API定义的报告级别与MPP数据库中

的级别相同：

```
typedef enum GppcReportLevel
{
    GPPC_DEBUG1           = 10,
    GPPC_DEBUG2           = 11,
    GPPC_DEBUG3           = 12,
    GPPC_DEBUG4           = 13,
    GPPC_DEBUG             = 14,
    GPPC_LOG               = 15,
    GPPC_INFO              = 17,
    GPPC_NOTICE            = 18,
    GPPC_WARNING           = 19,
    GPPC_ERROR             = 20,
} GppcReportLevel;
```

(MPP数据库[client_min_messages](#)服务器配置参数控制当前客户端日志记录级别。[log_min_messages](#)配置参数控制当前日志到日志文件级别。)

GPPC报告包括报告级别，报告消息和可选的报告回调函数。

GPPC API提供的报告和处理函数包括：

函数名	描述
GppcReport()	格式化并打印/记录指定报告级别的字符串。
GppcInstallReportCallback()	注册/安装报告回调函数。
GppcUninstallReportCallback()	卸载报告回调函数。
GppcGetReportLevel()	从错误报告中检索级别。
GppcGetReportMessage()	从错误报告中检索消息。
GppcCheckForInterrupts()	如果中断挂起则出错。

GppcReport()函数签名是：


```
void GppcReport(GppcReportLevel elevel, const char *fmt, ...);
```

GppcReport()采用类似于printf()的格式化字符串输入参数。 以下示例生成格式化GPPC文本参数的错误级别报告消息：

```
GppcText uname = GPPC_GETARG_TEXT(1);
GppcReport(GPPC_ERROR, "Unknown user name: %s", GppcTextGetCString(uname));
```

有关示例报告回调处理程序，请参阅[GPPC示例代码](#)。

SPI函数

MPP数据库服务器编程接口（SPI）为C/C++函数的编写者提供了在GPPC函数中运行SQL命令的能力。 有关SPI函数的其他信息，请参阅PostgreSQL文档中的[服务器编程接口](#)。

GPPC API公开了PostgreSQL SPI函数的子集。 通过该子集，您可以在GPPC函数中发出SPI查询并检索SPI结果值。 GPPC SPI包装器函数是：

SPI函数名	GPPC函数名	描述
SPI_connect()	GppcSPIConnect()	连接到MPP数据库服务器编程接口。
SPI_finish()	GppcSPIFinish()	断开与MPP数据库服务器编程接口的连接。

SPI函数名	GPPC函数名	描述
SPI_exec()	Gppc SPIExec()	执行SQL语句，返回行数。
SPI_getvalue()	Gppc SPIGetValue()	从SQL结果中按编号检索特定属性的值作为字符串。
	Gppc SPIGetDatum()	从SQL结果中按编号检索特定属性的值作为GppcDatum。
	Gppc SPIGetValueByName()	按名称从SQL结果中检索特定属性的值作为字符串。
	Gppc SPIGetDatumByName()	按名称从SQL结果中检索特定属性的值作为GppcDatum。

创建访问服务器编程接口的GPPC函数时，您的函数应符合以下流程：

```
GppcSPIConnect();
GppcSPIExec(...)
// process the results - GppcSPIGetValue(...), GppcSPIGetDatum(...)
GppcSPIFinish()
```

您可以使用GppcSPIExec()在GPPC函数中执行SQL语句。调用此函数时，还可以标识要返回的最大行数。GppcSPIExec()的函数签名是：

```
GppcSPIResult GppcSPIExec(const char *sql_statement, long rcount);
```

GppcSPIExec()返回GppcSPIResult结构。该结构表示SPI结果数据。它包括指向数据的指针，有关处理的行数的信息，计数器和结果代码。GPPC API定义此结构如下：

```
typedef struct GppcSPIResultData
{
    struct GppcSPITupleTableData *tuptable;
    uint32_t processed;
    uint32_t current;
    int rescode;
} GppcSPIResultData;
typedef GppcSPIResultData *GppcSPIResult;
```

您可以设置和使用GppcSPIResult结构中的current字段来检查tuptable结果数据的每一行。

以下代码摘录使用GPPC API连接到SPI，执行简单查询，循环查询结果并完成处理：

```
GppcSPIResult result;
char      *attname = "id";
char      *query = "SELECT i, 'foo' || i AS val FROM generate_series(1, 10)i ORDER BY 1";
bool      isnull = true;

// connect to SPI
if( GppcSPIConnect() < 0 ) {
    GppcReport(GPPC_ERROR, "cannot connect to SPI");
}

// execute the query, returning all rows
result = GppcSPIExec(query, 0);

// process result
while( result->current < result->processed ) {
    // get the value of attname column as a datum, making a copy
    datum = GppcSPIGetDatumByName(result, attname, &isnull, true);

    // do something with value

    // move on to next row
    result->current++;
}

// complete processing
GppcSPIFinish();
```

关于元组描述符和元组

表或一组记录包含一个或多个元组（行）。元组的每个属性的结构由元组描述符定义。元组描述符为元组中的每个属性定义以下内容：

- 属性名称
- 属性数据类型的对象标识符
- 属性数据类型的字节长度
- 属性修饰符的对象标识符

GPPC API定义了一个抽象类型GppcTupleDesc来表示元组/行描述符。 API还提供了可用于创建，访问和设置元组描述符的函数：

函数名称	描述
GppcCreateTemplateTupleDesc()	创建具有指定数量的属性的空元组描述符。
GppcTupleDescInitEntry()	在指定位置向元组描述符添加属性。
GppcTupleDescNattrs()	获取元组描述符中的属性数。
GppcTupleDescAttrName()	获取元组描述符中特定位置（从0开始）的属性名称。
GppcTupleDescAttrType()	获取元组描述符中特定位置（从0开始）的属性的类型对象标识符。
GppcTupleDescAttrLen()	获取元组描述符中特定位置（从0开始）的属性的类型长度。
GppcTupleDescAttrTypmod()	获取元组描述符中特定位置（从0开始）的属性的类型修饰符对象标识符。

要构造元组描述符，首先要创建一个模板，然后为每个属性填写描述符字段。这些函数的签名是：

```
GppcTupleDesc GppcCreateTemplateTupleDesc(int natts);
void GppcTupleDescInitEntry(GppcTupleDesc desc, uint16_t attno,
                             const char *attname, GppcOid typid, int32_t typmod);
```

在某些情况下，您可能希望从现有元组中的属性定义初始化元组描述符条目。以下函数获取元组描述符中的属性数，以及描述符中特定属性（按编号）的定义：

```
int GppcTupleDescNattrs(GppcTupleDesc tupdesc);
const char *GppcTupleDescAttrName(GppcTupleDesc tupdesc, int16_t attno);
GppcOid GppcTupleDescAttrType(GppcTupleDesc tupdesc, int16_t attno);
int16_t GppcTupleDescAttrLen(GppcTupleDesc tupdesc, int16_t attno);
int32_t GppcTupleDescAttrTypmod(GppcTupleDesc tupdesc, int16_t attno);
```

以下示例初始化两个属性元组描述符。 第一个属性使用来自不同描述符的属性定义进行初始化，第二个属性初始化为布尔类型属性：

```
GppcTupleDesc    tdesc;
GppcTupleDesc    indesc = some_input_descriptor;

// initialize the tuple descriptor with 2 attributes
tdesc = GppcCreateTemplateTupleDesc(2);

// use third attribute from the input descriptor
GppcTupleDescInitEntry(tdesc, 1,
    GppcTupleDescAttrName(indesc, 2),
    GppcTupleDescAttrType(indesc, 2),
    GppcTupleDescAttrTypmod(indesc, 2));

// create the boolean attribute
GppcTupleDescInitEntry(tdesc, 2, "is_active", GppcOidBool, 0);
```

GPPC API定义了一个抽象类型GppcHeapTuple来表示元组/记录/行。 元组由其元组描述符定义，每个元组属性的值以及每个值是否为NULL的指示符。

GPPC API提供了可用于设置和访问元组及其属性的函数：

函数名称	描述
GppcHeapFormTuple()	从GppcDatum数组中形成一个元组。
GppcBuildHeapTupleDatum()	从GppcDatum数组中形成一个GppcDatum元组。
GppcGetAttributeByName()	按名称从元组中获取属性。
GppcGetAttributeByNum()	从数字中获取元组的属性（从1开始）。

构建元组GPPC函数的签名是：

```
GppcDatum GppcBuildHeapTupleDatum(GppcTupleDesc tupdesc, GppcDatum *values, bool *nulls)
GppcDatum GppcBuildHeapTupleDatum(GppcTupleDesc tupdesc, GppcDatum *values, bool *nulls)
```

以下代码摘录从上面的代码示例中的元组描述符构造GppcDatum元组，并从函数的整数和布尔输入参数构造：

```
GppcDatum intarg = GPPC_GETARG_INT4(0);
GppcDatum boolarg = GPPC_GETARG_BOOL(1);
GppcDatum result, values[2];
bool nulls[2] = { false, false };

// construct the values array
values[0] = intarg;
values[1] = boolarg;
result = GppcBuildHeapTupleDatum( tdesc, values, nulls );
```

Set-Returning函数

其签名包括RETURNS SETOF RECORD或RETURNS TABLE(...)的MPP数据库UDF是set-returning函数。

GPPC API为GPPC函数返回集（例如，多行/元组）提供支持。MPP数据库为每个行或项目调用一次set-returning函数（SRF）。该函数必须保存足够的状态以记住它正在做什么并返回每次调用的下一行。您在SRF上下文中分配的内存必须在多个函数调用中存活。

GPPC API提供宏和函数，以帮助跟踪和设置此上下文，并分配SRF内存。他们包括：

函数/宏名称	描述
GPPC_SRF_RESULT_DESC()	获取此SRF的输出行元组描述符。结果元组描述符由输出表定义或DESCRIBE函数确定。
GPPC_SRF_IS_FIRSTCALL()	确定这是否是对SRF的第一次调用。
GPPC_SRF_FIRSTCALL_INIT()	初始化SRF上下文。
GPPC_SRF_PERCALL_SETUP()	在每次调用SRF时恢复上下文。
GPPC_SRF_RETURN_NEXT()	从SRF返回值并继续处理。
GPPC_SRF_RETURN_DONE()	SRF处理完成的信号。
GppSRFAlloc()	在此SRF上下文中分配内存。
GppSRFAlloc0()	在此SRF上下文中分配内存并将其初始化为零。
GppSRFSave()	在此SRF上下文中保存用户状态。
GppSRFRestore()	在此SRF上下文中还原用户状态。

GppcFuncCallContext结构提供SRF的上下文。您在第一次调用SRF时创建此上下文。您的set-returning GPPC函数必须在每次调用时检索函数上下文。例如：


```
// set function context
GppcFuncCallContext fctx;
if (GPPC_SRF_IS_FIRSTCALL()) {
    fctx = GPPC_SRF_FIRSTCALL_INIT();
}
fctx = GPPC_SRF_PERCALL_SETUP();
// process the tuple
```

GPPC函数必须在返回元组结果时提供上下文或指示处理已完成。 例如：

```
GPPC_SRF_RETURN_NEXT(fctx, result_tuple);
// or
GPPC_SRF_RETURN_DONE(fctx);
```

使用DESCRIBE函数定义使用RETURNS SETOF RECORD子句的函数的输出元组描述符。 使用GPPC_SRF_RESULT_DESC()宏获取使用RETURNS TABLE(...)子句的函数的输出元组描述符。

有关set-returning函数代码和部署示例，请参阅[GPPC Set-Returning函数示例](#)。

表函数

GPPC API提供GppcAnyTable类型以将表作为输入参数传递给函数，或者将表作为函数结果返回。

GPPC API中提供的与表相关的函数和宏包括：

函数/宏名称	描述
GPPC_GETARG_ANYTABLE()	获取任何表函数参数。
GPPC_RETURN_ANYTABLE()	返回表。

函数/宏名称	描述
GppcAnyTableGetTupleDesc()	获取表的元组描述符。
GppcAnyTableGetNextTuple()	获取表中的下一行。

您可以使用GPPC_GETARG_ANYTABLE()宏来检索表输入参数。 当您有权访问该表时，可以使用GppcAnyTableGetTupleDesc()函数检查该表的元组描述符。 该函数的签名是：

```
GppcTupleDesc GppcAnyTableGetTupleDesc(GppcAnyTable t);
```

例如，要检索作为函数的第一个输入参数的表的元组描述符：

```
GppcAnyTable    intbl;
GppcTupleDesc   in_desc;

intbl = GPPC_GETARG_ANYTABLE(0);
in_desc = GppcAnyTableGetTupleDesc(intbl);
```

GppcAnyTableGetNextTuple()函数从表中获取下一行。 同样，要从上表中检索下一个元组：

```
GppcHeapTuple   ntuple;

ntuple = GppcAnyTableGetNextTuple(intbl);
```

限制

使用MPP数据库版本5.0.x的GPPC API不支持以下运算符：

- integer || integer
- integer = text
- text < integer

样例代码

MPP数据库github存储库中的[gppc test](#)目录包含示例GPPC代码：

- gppc_demo/ - 示例代码，用于执行GPPC SPI函数，错误报告，数据类型参数和返回宏，set-returning函数和编码函数
- tabfunc_gppc_demo/ - 示例代码执行GPPC表和set-returning函数

使用PGXS构建GPPC共享库

您可以将使用GPPC API编写的函数编译到MPP数据库服务器按需加载的一个或多个共享库中。

您可以使用PostgreSQL构建扩展基础结构（PGXS）根据MPP数据库安装为您的GPPC函数构建源代码。该框架自动化简单模块的通用构建规则。如果您有一个更复杂的用例，则需要编写自己的构建系统。

要使用PGXS基础结构为使用GPPC API创建的函数生成共享库，请创建一个设置PGXS特定变量的简单Makefile。

Note: 有关PGXS支持的Makefile变量的信息，请参阅PostgreSQL文档中的[扩展的构建基础设施](#)。

例如，以下Makefile从名为src1.c和src2.c的两个C源文件生成名为sharedlib_name.so的共享库：

```
MODULE_big = sharedlib_name
OBJS = src1.o src2.o
PG_CPPFLAGS = -I$(shell $(PG_CONFIG) --includedir)
SHLIB_LINK = -L$(shell $(PG_CONFIG) --libdir) -lgppc

PG_CONFIG = pg_config
PGXS := $(shell $(PG_CONFIG) --pgxs)
include $(PGXS)
```

MODULE_big标识Makefile生成的共享库的基本名称。

PG_CPPFLAGS将MPP数据库安装包含目录添加到编译器头文件搜索路径中。

SHLIB_LINK将MPP数据库安装库目录添加到链接器搜索路径。此变量还将GPPC库（-lgppc）添加到link命令。

PG_CONFIG和PGXS变量设置和include语句是必需的，通常位于Makefile的最后三行。

使用MPP数据库注册GPPC函数

在用户可以从SQL调用GPPC函数之前，必须使用MPP数据库注册该函数。

注册GPPC函数涉及将GPPC函数签名映射到SQL用户定义的函数。您可以使用CREATE FUNCTION .. AS 命令定义此映射，以指定GPPC共享库名称。您可以选择为GPPC和SQL函数使用相同的名称或不同的名称。

示例CREATE FUNCTION ... AS 语法如下：

```
CREATE FUNCTION sql_function_name(arg[, ...]) RETURNS return_type
AS 'shared_library_path', 'gppc_function_name'
LANGUAGE C STRICT [WITH (DESCRIBE=describe_function)];
```

指定`shared_library_path`时，可以省略共享库`.so`扩展名。

如果GPPC函数在名为`gppc_try.so`的共享库中编译和链接，则以下命令将本主题前面引用的示例`add_int4s()`函数注册到名为`add_two_int4s_gppc()`的SQL UDF：

```
CREATE FUNCTION add_two_int4s_gppc(int4, int4) RETURNS int8
AS 'gppc_try.so', 'add_int4s'
LANGUAGE C STRICT;
```

关于动态加载

您可以在`CREATE FUNCTION ... AS`命令中指定GPPC共享库的名称，以在MPP数据库的共享库中注册GPPC函数。MPP数据库动态加载程序在用户第一次调用在该共享库中链接的用户定义函数时将GPPC共享库文件加载到内存中。如果在`CREATE FUNCTION ... AS`命令中未提供共享库的绝对路径，MPP数据库将尝试使用以下有序步骤找到库：

1. 如果共享库文件路径以字符串`$libdir`开头，则MPP数据库将在PostgreSQL包库目录中查找该文件。运行`pg_config --pkglibdir`命令以确定此目录的位置。
2. 如果指定了没有目录前缀的共享库文件名，则MPP数据库将在`dynamic_library_path`服务器配置参数值标识的目录中搜索该文件。
3. 当前的工作目录。

打包和部署注意事项

您必须以适合MPP集群中MPP数据库管理员部署的形式打包GPPC共享库和SQL函数注册脚本。提供GPPC包的特定部署说明。

构建程序包和部署说明时，请考虑以下事项：

- 考虑提供MPP数据库管理员运行的shell脚本或程序，以便将共享库安装到所需的文件系统位置并注册GPPC函数。
- 必须将GPPC共享库安装到master主机上的相同文件系统位置以及MPP数据库群集中的每个segment主机上。
- gpadmin用户必须具有遍历GPPC共享库文件的完整文件系统路径的权限。
- 安装在MPP数据库部署中后，GPPC共享库的文件系统位置决定了在使用CREATE FUNCTION ... AS命令在库中注册函数时如何引用共享库。
- 创建一个.sql脚本文件，为GPPC共享库中的每个GPPC函数注册一个SQL UDF。您在.sql注册脚本中创建的函数必须引用GPPC共享库的部署位置。在GPPC部署包中包含此脚本。
- 记录运行GPPC包部署脚本的说明（如果提供）。
- 如果未在程序包部署脚本中包含此任务，请记录有关安装GPPC共享库的说明。
- 如果未在程序包部署脚本中包含此任务，请记录有关安装和运行函数注册脚本的说明。

GPPC文本函数示例

在此示例中，您将开发，构建和部署GPPC共享库，并注册并运行名为concat_two_strings的GPPC函数。此函数使用GPPC API连接两个字符串参数并返回结果。

您将在MPP数据库主控主机上开发GPPC函数。部署您在此示例中创建的GPPC共享库需要对MPP数据库集群的管理访问权限。

执行以下过程以运行该示例：

1. 登录MPP数据库主控主机并设置您的环境。例如：

```
$ ssh gadmin@<gpmaster>
gadmin@gpmaster$ . /usr/local/mpp-db/mpp_path.sh
```

2. 创建工作目录并导航到新目录。例如：

```
gadmin@gpmaster$ mkdir gppc_work
gadmin@gpmaster$ cd gppc_work
```

3. 通过在您选择的编辑器中打开文件，为GPPC源代码准备文件。例如，要使用vi打开名为gppc_concat.c的文件：

```
gadmin@gpmaster$ vi gppc_concat.c
```

4. 将以下代码复制/粘贴到文件中：
-

```
#include <stdio.h>
#include <string.h>
#include "gppc.h"

// make the function SQL-invokable
GPPC_FUNCTION_INFO(concat_two_strings);

// declare the function
GppcDatum concat_two_strings(GPPC_FUNCTION_ARGS);

GppcDatum
concat_two_strings(GPPC_FUNCTION_ARGS)
{
    // retrieve the text input arguments
    GppcText arg0 = GPPC_GETARG_TEXT(0);
    GppcText arg1 = GPPC_GETARG_TEXT(1);

    // determine the size of the concatenated string and allocate
    // text memory of this size
    size_t arg0_len = GppcGetTextLength(arg0);
    size_t arg1_len = GppcGetTextLength(arg1);
    GppcText retstring = GppcAllocText(arg0_len + arg1_len);

    // construct the concatenated return string
    memcpy(GppcGetTextPointer(retstring), GppcGetTextPointer(arg0), arg0_len);
    memcpy(GppcGetTextPointer(retstring) + arg0_len, GppcGetTextPointer(arg1), arg1_len);

    GPPC_RETURN_TEXT( retstring );
}
```

代码声明并实现了concat_two_strings()函数。它使用GPPC数据类型，宏和函数来获取函数参数，为连接的字符串分配内存，将参数复制到新字符串中，然后返回结果。

5. 保存文件并退出编辑器。
 6. 在您选择的编辑器中打开名为Makefile的文件。将以下文本复制/粘贴到文件中：
-


```
MODULE_big = gppc_concat
OBJS = gppc_concat.o

PG_CONFIG = pg_config
PGXS := $(shell $(PG_CONFIG) --pgxs)

PG_CPPFLAGS = -I$(shell $(PG_CONFIG) --includedir)
SHLIB_LINK = -L$(shell $(PG_CONFIG) --libdir) -lgppc
include $(PGXS)
```

7. 保存文件并退出编辑器。

8. 为concat_two_strings()函数构建GPPC共享库。例如：

```
gpadmin@gpmaster$ make all
```

make命令在当前工作目录中生成名为gppc_concat.so的共享库文件。

9. 将共享库复制到MPP数据库安装。您必须具有MPP数据库管理权限才能复制该文件。例如：

```
gpadmin@gpmaster$ cp gppc_concat.so /usr/local/mpp-db/lib/postgresql/
```

10. 将共享库复制到MPP数据库安装中的每个主机。例如，如果seghostfile包含MPP数据库集群中segment主机的列表，每个主机行：

```
gpadmin@gpmaster$ gpscp -v -f seghostfile /usr/local/mpp-db/lib/postgresql/gppc_concat.so
```

1. 打开psql会话。例如：

```
gpadmin@gpmaster$ psql -d testdb
```

2. 使用MPP数据库注册名为concat_two_strings()的GPPC函数。例如，将

MPP数据库函数concat_with_gppc()映射到GPPC concat_two_strings()函数：

```
testdb=# CREATE FUNCTION concat_with_gppc(text, text) RETURNS text
        AS 'gppc_concat', 'concat_two_strings'
        LANGUAGE C STRICT;
```

3. 运行concat_with_gppc()函数。例如：

```
testdb=# SELECT concat_with_gppc( 'happy', 'monday' );
concat_with_gppc
-----
happymonday
(1 row)
```

GPPC Set-Returning函数示例

在此示例中，您将开发，构建和部署GPPC共享库。您还可以为名为return_tbl()的GPPC函数创建并运行.sql注册脚本。此函数使用GPPC API获取带有整数和文本列的输入表，确定整数列是否大于13，并返回带有输入整数列的结果表和一个标识整数是否是的整数的布尔列return_tbl()使用GPPC API报告和SRF函数和宏。

您将在MPP数据库master主机上开发GPPC函数。部署您在此示例中创建的GPPC共享库需要对MPP数据库集群的管理访问权限。

执行以下过程以运行该示例：

1. 登录MPP数据库master主机并设置您的环境。例如：

```
$ ssh gpadmin@<gpmaster>
gpadmin@gpmaster$ . /usr/local/mpp-db/mpp_path.sh
```

2. 创建工作目录并导航到新目录。例如：

```
gpadmin@gpmaster$ mkdir gppc_work
gpadmin@gpmaster$ cd gppc_work
```

3. 通过在您选择的编辑器中打开文件，为GPPC代码准备源文件。例如，要使用vi打开名为gppc_concat.c的文件：

```
gpadmin@gpmaster$ vi gppc_rettbl.c
```

4. 将以下代码复制/粘贴到文件中：

```
#include <stdio.h>
#include <string.h>
#include "gppc.h"

// initialize the logging level
GppcReportLevel level = GPPC_INFO;

// make the function SQL-invokable and declare the function
GPPC_FUNCTION_INFO(return_tbl);
GppcDatum return_tbl(GPPC_FUNCTION_ARGS);

GppcDatum
return_tbl(GPPC_FUNCTION_ARGS)
{
    GppcFuncCallContext fctx;
    GppcAnyTable intbl;
    GppcHeapTuple intuple;
    GppcTupleDesc in_tupdesc, out_tupdesc;
    GppcBool    resbool = false;
    GppcDatum    result, boolres, values[2];
    bool    nulls[2] = {false, false};

    // single input argument - the table
    intbl = GPPC_GETARG_ANYTABLE(0);

    // set the function context
    if (GPPC_SRF_IS_FIRSTCALL()) {
```

```

    fctx = GPPC_SRF_FIRSTCALL_INIT();
}
fctx = GPPC_SRF_PERCALL_SETUP();

// get the tuple descriptor for the input table
in_tupdesc = GppcAnyTableGetTupleDesc(intbl);

// retrieve the next tuple
intuple = GppcAnyTableGetNextTuple(intbl);
if( intuple == NULL ) {
    // no more tuples, conclude
    GPPC_SRF_RETURN_DONE(fctx);
}

// get the output tuple descriptor and verify that it is
// defined as we expect
out_tupdesc = GPPC_SRF_RESULT_DESC();
if (GppcTupleDescNattrs(out_tupdesc) != 2 ||
    GppcTupleDescAttrType(out_tupdesc, 0) != GppcOidInt4 ||
    GppcTupleDescAttrType(out_tupdesc, 1) != GppcOidBool) {
    GppcReport(GPPC_ERROR, "INVALID out_tupdesc tuple");
}

// log the attribute names of the output tuple descriptor
GppcReport(level, "output tuple descriptor attr0 name: %s", GppcTupleDescAttrName(out_tupdesc, 0));
GppcReport(level, "output tuple descriptor attr1 name: %s", GppcTupleDescAttrName(out_tupdesc, 1));

// retrieve the attribute values by name from the tuple
bool text_isnull, int_isnull;
GppcDatum intdat = GppcGetAttributeByName(intuple, "id", &int_isnull);
GppcDatum textdat = GppcGetAttributeByName(intuple, "msg", &text_isnull);

// convert datum to specific type
GppcInt4 intarg = GppcDatumGetInt4(intdat);
GppcReport(level, "id: %d", intarg);
GppcReport(level, "msg: %s", GppcTextGetCString(GppcDatumGetText(textdat)));

// perform the >13 check on the integer
if( !int_isnull && (intarg > 13) ) {
    // greater than 13?
    resbool = true;
    GppcReport(level, "id is greater than 13!");
}

```

```

}

// values are datums; use integer from the tuple and
// construct the datum for the boolean return
values[0] = intdat;
boolres = GppcBoolGetDatum(resbool);
values[1] = boolres;

// build a datum tuple and return
result = GppcBuildHeapTupleDatum(out_tupdesc, values, nulls);
GPPC_SRF_RETURN_NEXT(fctx, result);

}

```

代码声明并实现了`return_tbl()`函数。它使用GPPC数据类型，宏和函数来获取函数参数，检查元组描述符，构建返回元组，并返回结果。该函数还使用SRF宏来跟踪函数调用之间的元组上下文。

5. 保存文件并退出编辑器。
6. 在您选择的编辑器中打开名为Makefile的文件。将以下文本复制/粘贴到文件中：

```

MODULE_big = gppc_rettbl
OBJS = gppc_rettbl.o

PG_CONFIG = pg_config
PGXS := $(shell $(PG_CONFIG) --pgxs)

PG_CPPFLAGS = -I$(shell $(PG_CONFIG) --includedir)
SHLIB_LINK = -L$(shell $(PG_CONFIG) --libdir) -lgppc
include $(PGXS)

```

7. 保存文件并退出编辑器。
8. 为`return_tbl()`函数构建GPPC共享库。例如：

```
gpadmin@gpmaster$ make all
```

make命令在当前工作目录中生成名为gppc_rettbl.so的共享库文件。

9. 将共享库复制到MPP数据库安装。您必须具有MPP数据库管理权限才能复制该文件。例如：

```
gpadmin@gpmaster$ cp gppc_rettbl.so /usr/local/mpp-db/lib/postgresql/
```

此命令将共享库复制到\$libdir

0. 将共享库复制到MPP数据库安装中的每个主机。例如，如果seghostfile包含MPP数据库集群中segment主机的列表，每个主机一行：

```
gpadmin@gpmaster$ gpscp -v -f seghostfile /usr/local/mpp-db/lib/postgresql/gppc_rettbl.so
```

1. 创建.sql文件以注册GPPC return_tbl()函数。在您选择的编辑器中打开名为gppc_rettbl_reg.sql的文件。
2. 将以下文本复制/粘贴到文件中：

```
CREATE FUNCTION rettbl_gppc(anytable) RETURNS TABLE(id int4, thirteen bool)
  AS 'gppc_rettbl', 'return_tbl'
LANGUAGE C STRICT;
```

3. 通过运行刚刚创建的脚本来注册GPPC功能。例如，要在名为testdb的数据库中注册该函数：

```
gpadmin@gpmaster$ psql -d testdb -f gppc_rettbl_reg.sql
```

4. 打开psql会话。例如：

```
gpadmin@gpmaster$ psql -d testdb
```

5. 创建包含一些测试数据的表。例如：

```
CREATE TABLE gppc_testtbl( id int, msg text );
INSERT INTO gppc_testtbl VALUES (1, 'f1');
INSERT INTO gppc_testtbl VALUES (7, 'f7');
INSERT INTO gppc_testtbl VALUES (10, 'f10');
INSERT INTO gppc_testtbl VALUES (13, 'f13');
INSERT INTO gppc_testtbl VALUES (15, 'f15');
INSERT INTO gppc_testtbl VALUES (17, 'f17');
```

6. 运行rettbl_gppc()函数。例如：

```
testdb=# SELECT * FROM rettbl_gppc(TABLE(SELECT * FROM gppc_testtbl));
 id | thirteen
----+-----
  1 | f
  7 | f
 13 | f
 15 | t
 17 | t
 10 | f
(6 rows)
```

后台工作进程

可以扩展MPP数据库以在单独的进程中运行用户提供的代码。这些进程由postgres启动，停止和监视，这使得它们的生命周期与服务器的状态紧密相关。这些进程可以选择连接到MPP数据库的共享内存区域并在内部连接到数据

库;它们还可以串行运行多个事务，就像常规的客户端连接服务器进程一样。此外，通过链接到libpq，它们可以连接到服务器，并且行为类似于常规客户端应用程序。

Warning: 使用后台工作进程存在相当大的稳健性和安全性风险，因为使用C语言编写时，它们可以不受限制地访问数据。希望启用包含后台工作进程的模块的管理员应该格外小心。只允许经过仔细审核的模块运行后台工作进程。

通过在`shared_preload_libraries`服务器配置参数中包含模块名称，可以在启动MPP数据库时初始化后台工作程序。希望运行后台工作程序的模块可以通过从`_PG_init()`调用`RegisterBackgroundWorker(BackgroundWorker *worker)`来注册它。通过调用函数`RegisterDynamicBackgroundWorker(BackgroundWorker *worker, BackgroundWorkerHandle **handle)`，系统启动并运行后，也可以启动后台工作程序。与只能在postmaster中调用的`RegisterBackgroundWorker`不同，必须从常规后端调用`RegisterDynamicBackgroundWorker`。

因此定义了BackgroundWorker结构：

```
typedef void (*bgworker_main_type)(Datum main_arg);
typedef struct BackgroundWorker
{
    char    bgw_name[BGW_MAXLEN];
    int     bgw_flags;
    BgWorkerStartTime bgw_start_time;
    int     bgw_restart_time; /* in seconds, or BGW_NEVER_RESTART */
    bgworker_main_type bgw_main;
    char    bgw_library_name[BGW_MAXLEN]; /* only if bgw_main is NULL */
    char    bgw_function_name[BGW_MAXLEN]; /* only if bgw_main is NULL */
    Datum   bgw_main_arg;
    int     bgw_notify_pid;
} BackgroundWorker;
```

`bgw_name`是一个在日志消息，进程列表和类似上下文中使用的字符串。

`bgw_flags`是一个按位或位掩码，表示模块想要的功能。可能的值是`BGWORKER_SHMEM_ACCESS`（请求共享内存访问）

和BGWORKER_BACKEND_DATABASE_CONNECTION（请求建立数据库连接的能力，以后可以通过它运行事务和查询）。使用BGWORKER_BACKEND_DATABASE_CONNECTION连接到数据库的后台工作进程还必须使用BGWORKER_SHMEM_ACCESS附加共享内存，否则工作进程将启动失败。

bgw_start_time是postgres应该启动进程的服务器状态;它可以是BgWorkerStart_PostmasterStart之一（一旦postgres完成自己的初始化就开始;请求它的进程不符合数据库连接的条件），BgWorkerStart_ConsistentState（一旦在热备份中达到一致状态就开始，允许进程连接到数据库并运行只读查询）和BgWorkerStart_RecoveryFinished（系统进入正常读写状态后立即启动）。请注意，最后两个值在不是热备份的服务器中是等效的。请注意，此设置仅指示何时启动进程;当达到不同的状态时，它们不会停止。

bgw_restart_time是postgres在重新启动进程之前应该等待的时间间隔（以秒为单位），以防它崩溃。它可以是任何正值，或BGW_NEVER_RESTART，表示在发生崩溃时不重启进程。

bgw_main是指向启动进程时要运行的函数的指针。此函数必须采用Datum类型的单个参数并返回void。bgw_main_arg将作为唯一参数传递给它。请注意，全局变量MyBgworkerEntry指向在注册时传递的BackgroundWorker结构的副本。bgw_main可能为NULL;在这种情况下，将使用bgw_library_name和bgw_function_name来确定入口点。这对postmaster启动后启动的后台工作进程很有用，其中postmaster没有加载必需的库。

bgw_library_name是库的名称，在该库中应该寻找后台工作程序的初始入口点。除非bgw_main为NULL，否则它将被忽略。但是如果bgw_main为NULL，则命名库将由工作进程动态加载，bgw_function_name将用于标识要调用的函数。

bgw_function_name是动态加载库中函数的名称，该库应该用作新后台工作程序的初始入口点。除非bgw_main为NULL，否则它将被忽略。

`bgw_notify_pid`是一个MPP数据库后端进程的PID，`postmaster`应该在进程启动或退出时向其发送SIGUSR1。对于在`postmaster`启动时注册的工作进程，或者注册工作进程的后端不希望等待工作进程启动时，应该为0。否则，它应该初始化为`MyProcPid`。

一旦运行，该进程可以通过调用`BackgroundWorkerInitializeConnection(char *dbname, char *username)`连接到数据库。这允许进程使用SPI接口运行事务和查询。如果`dbname`为NULL，则会话未连接到任何特定数据库，但可以访问共享目录。如果`username`为NULL，则进程将以`initdb`期间创建的超级用户身份运行。`BackgroundWorkerInitializeConnection`只能在每个后台进程中调用一次，无法切换数据库。

当控制到达`bgw_main`函数时，信号最初被阻止，并且必须被它解除阻塞；这是为了允许进程在必要时自定义其信号处理程序。通过调用`BackgroundWorkerUnblockSignals`可以在新进程中取消阻止信号，并通过调用`BackgroundWorkerBlockSignals`来阻止信号。

如果后台工作程序的`bgw_restart_time`配置为`BGW_NEVER_RESTART`，或者退出时退出代码为0或由`TerminateBackgroundWorker`终止，则退出时将由`postmaster`自动取消注册。否则，它将在通过`bgw_restart_time`配置的时间段之后重新启动，或者如果`postmaster`由于后端故障而重新初始化群集时立即重新启动。需要暂时暂停执行的后端应该使用可中断的睡眠而不是退出；这可以通过调用`WaitLatch()`来实现。确保在调用该函数时设置`WL_POSTMASTER_DEATH`标志，并在`postgres`本身已终止的紧急情况下验证返回代码以提示退出。

当使用`RegisterDynamicBackgroundWorker`函数注册后台工作程序时，执行注册的后端可以获得有关工作进程状态的信息。希望这样做的后端应该将`BackgroundWorkerHandle *`的地址作为`RegisterDynamicBackgroundWorker`的第二个参数传递。如果工作程序已成功注册，则此指针将使用不透明句柄进行初始化，该句柄随后可以传递给`GetBackgroundWorkerPid(BackgroundWorkerHandle *, pid_t *)`或`TerminateBackgroundWorker(BackgroundWorkerHandle *)`。`GetBackgroundWorkerPid`可用于轮询工作进程的状态：返回

值BGWH_NOT_YET_STARTED表示工作进程尚未由postmaster启动; BGWH_STOPPED表示它已经启动但不再运行; 和BGWH_STARTED表示它当前正在运行。 在最后一种情况下，PID也将通过第二个参数返回。 TerminateBackgroundWorker使postmaster在运行时将SIGTERM发送给worker，并在它不运行时立即取消注册。

在某些情况下，注册后台工作进程的进程可能希望等待工作进程启动。 这可以通过将bgw_notify_pid初始化为MyProcPid，然后将在注册时获得的BackgroundWorkerHandle *传递给WaitForBackgroundWorkerStartup(BackgroundWorkerHandle *handle,pid_t *)函数来实现。 此函数将阻塞，直到postmaster尝试启动后台工作程序，或直到postmaster死亡。 如果后台运行器正在运行，则返回值将为BGWH_STARTED，并且PID将被写入提供的地址。 否则，返回值将为BGWH_STOPPED或BGWH_POSTMASTER_DIED。

worker_spi模块包含一个工作示例，演示了一些有用的技术。

注册后台工作进程的最大数量受max-worker-processes限制。

Parent topic: [MPP数据库参考手册](#)

MPP 特点概要

该部分提供了MPP数据库的系统要求和功能集合的高级概述。它包含以下主题：

- [MPP SQL标准一致性](#)
- [MPP和PostgreSQL兼容性](#)

Parent topic: [MPP数据库参考手册](#)

MPP SQL标准一致性

SQL语言于1986年被美国国家标准学会（ANSI）第一次作为SQL正式标准化。SQL标准的后续版本已由ANSI和国际标准化组织（ISO）标准发布：SQL 1989，SQL 1992，SQL 1999，SQL 2003，SQL 2006，和最后的 SQL 2008，它就是当前的SQL标准。该标准的正式名称为 ISO/IEC 9075-14:2008。一般来说，每个更新的版本都增加了更多的内容，虽然偶尔也有一些内容被启用或者删除。

重要的是要注意，没有完全遵从SQL标准的商业数据库系统。MPP数据库几乎完全符合SQL 1992的标准，多数功能来源于SQL 1999。几个来源于SQL 2003的功能也被实现了（最著名的是 SQL OLAP 功能）。

该部分针对MPP数据库和SQL标准相关的重要一致性的问题。有关对最新的SQL标准的支持功能列表，请参阅[SQL 2008可选特性兼容性](#)。

核心SQL一致性

在构建并行，无共享架构的数据库系统和查询优化器的过程中，某些常见的SQL结构尚未在MPP数据库中实现。不支持以下的SQL结构：

1. 有些设置在EXISTS或NOT EXISTS子句中返回子查询，MPP的并行优化器不能将之重写为到连接之中。
 2. 向后回滚游标，包括FETCH PRIOR，FETCH FIRST，FETCH ABSOLUTE，和FETCH RELATIVE操作的使用。
 3. 在CREATE TABLE语句上（哈希分布表）：UNIQUE或PRIMARY KEY子句必须包括分布键列的所有值，或者是其超集。因为这个限制，在CREATE TABLE的语句中，仅允许一个UNIQUE子句或者PRIMARY KEY子句。UNIQUE或PRIMARY KEY子句不允许出现在随机分布的表中。
-

4. CREATE UNIQUE INDEX语句不包含分布键的列的所有值或者为其超集。
CREATE UNIQUE INDEX不允许使用在随机分布的表上。
注意UNIQUE INDEXES（但是不是UNIQUE CONSTRAINTS）在分布表的单个部分上执行，它们保证每个部分或者子部分内的键值的唯一性。
5. VOLATILE或STABLE函数不能在segment上执行，因此通常仅限于传递文字值作为其参数的参数。
6. 触发器是不支持的，因为他么通常依赖于VOLATILE函数的使用。
7. 引用完整性约束（外键）不会再MPP数据库中实施。用户可以声明外键，但是这些信息保存在系统catalog中。
8. 序列操纵函数CURRVAL和LASTVAL。

SQL 1992 一致性

以下 SQL 1992 的特性在MPP数据库中不支持：

1. NATIONAL CHARACTER (NCHAR) 和NATIONAL CHARACTER VARYING (NVARCHAR)。用户可以声明NCHAR和NVARCHAR类型，但是它们只是MPP数据库中CHAR和VARCHAR的同义词。
 2. CREATE ASSERTION语句。
 3. INTERVAL文字在MPP数据库中是支持的，但是不符合标准。
 4. GET DIAGNOSTICS语句。
 5. GLOBAL TEMPORARY TABLE和LOCAL TEMPORARY TABLE。MPP TEMPORARY TABLE不符合SQL标准，但许多商业数据库系统以相同的方式实现了临时表。MPP临时表与Teradata中的VOLATILE TABLE相同。
-

6. UNIQUE断言。
7. 引用完整性检查的MATCH PARTIAL（很可能不会在MPP数据库中实现）。

SQL 1999 一致性

以下SQL 1999的功能在MPP数据库中不支持：

1. Large Object数据类型：BLOB, CLOB, NCLOB。但是，Grennplum数据库中该BYTEA和TEXT列可以存储大量的数据（数百兆字节）。
2. MODULE（SQL 客户端模块）。
3. CREATE PROCEDURE (SQL/PSM)。这可通过创建返回值为void的FUNCTION在MPP数据库中进行操作，如下调用函数：

```
SELECT myfunc(args);
```

4. 该PostgreSQL/MPP函数定义语言（PL/PGSQL）是Oracle的PL/SQL的子集，而不是和SQL/PSM函数定义语言的兼容。MPP数据库还支持使用Python，Perl，Java和R定义函数。
5. BIT和BIT VARYING数据类型（故意忽略）。这些在SQL 2003中被弃用，并且在SQL 2008中被替代。
6. MPP支持63个字符长的标识符。该SQL标准要求支持达到128个字符长的标识符。
7. 准备好的事务（PREPARE TRANSACTION，COMMIT PREPARED，ROLLBACK PREPARED）。这也意味着MPP不支持XA事务（数据库事务和外部事务2个阶段的提交的协调）。

8. CHAR()或VARCHAR()列定义的CHARACTER SET选项。
9. 指明CHARACTERS或OCTETS(BYTES)关于CHAR()或VARCHAR()列的长度。例如，VARCHAR(15 CHARACTERS)或VARCHAR(15 OCTETS)或VARCHAR(15 BYTES)。
0. CURRENT_SCHEMA函数。
1. CREATE DISTINCT TYPE语句。CREATE DOMAIN可以用作MPP中的一种解决方案。
2. *explicit table*构造。

SQL 2003 一致性

以下SQL 2003的功能在MPP数据库中不支持：

1. MERGE语句。
 2. IDENTITY列和相关的GENERATED ALWAYS/GENERATED BY DEFAULT子句。该SERIAL或BIGSERIAL数据类型与INT或BIGINT GENERATED BY DEFAULT AS IDENTITY非常相似。
 3. 数据类型的MULTISET修饰符。
 4. ROW数据类型。
 5. MPP使用序列的语法是非标准的。例如，在MPP中使用nextval('seq')来替代标准的NEXT VALUE FOR seq。
 6. GENERATED ALWAYS AS列。视图可以用作解决方法。
 7. SELECT语句上的示例子句（TABLESAMPLE）。该random()函数可以用解
-

决从表中获取随机样本的方法。

8. 该*partitioned join tables*结构（连接中的PARTITION BY）。
9. 对CREATE TABLE x (LIKE(y))语句，MPP不支持[INCLUDING|EXCLUDING][DEFAULTS|CONSTRAINTS|INDEXES]语句。
10. MPP数组数据类型几乎符合SQL标准，但是有一些例外。通常，用户不应该遇到使用它们的问题。

SQL 2008 一致性

以下SQL 2008的特性在MPP数据库中不支持：

1. BINARY和VARBINARY数据类型。在MPP数据库中，BYTEA可以用来替代VARBINARY。
2. FETCH FIRST或FETCH NEXT子句对SELECT，例如：

```
SELECT id, name FROM tab1 ORDER BY id OFFSET 20 ROWS FETCH  
NEXT 10 ROWS ONLY;
```

MPP有LIMIT和LIMIT OFFSET子句可以用来替代。

3. 除非还使用LIMIT子句，否则在视图和子查询中将忽略ORDER BY子句。这是有意的，因为MPP优化器无法确定何时避免排序是安全的，从而对此类ORDER BY子句造成意外的性能影响。要解决这个问题，用户可以指定一个非常大的LIMIT。例如：SELECT * FROM mytable ORDER BY 1 LIMIT 9999999999
 4. *row subquery*结构是不支持的。
-

5. TRUNCATE TABLE不接受CONTINUE IDENTITY和RESTART IDENTITY子句。

MPP和PostgreSQL兼容性

MPP数据库基于PostgreSQL 9.4。 为了支持MPP数据库系统的分布式特性和典型工作负载，添加或修改了一些SQL命令，并且有一些不受支持的PostgreSQL功能。 MPP还添加了PostgreSQL中没有的功能，例如物理数据分发，并行查询优化，外部表，资源队列和增强的表分区。 有关完整的SQL语法和参考，请参阅[SQL Command Reference](#)。

Note: MPP数据库不支持PostgreSQL[大对象工具](#)，用于流式传输存储在大型对象结构中的用户数据。

Table 1. MPP数据库中的SQL支持

SQL命令	MPP是否支持	修改，限制，例外
ALTER AGGREGATE	YES	
ALTER CONVERSION	YES	
ALTER DATABASE	YES	
ALTER DOMAIN	YES	
ALTER EXTENSION	YES	更改MPP数据库扩展的定义 - 基于PostgreSQL 9.6。
ALTER FUNCTION	YES	
ALTER GROUP	YES	ALTER ROLE 的别名
ALTER INDEX	YES	
ALTER LANGUAGE	YES	

SQL命令	MPP是否支持	修改，限制，例外
ALTER OPERATOR	YES	
ALTER OPERATOR CLASS	YES	
ALTER OPERATOR FAMILY	YES	
ALTER PROTOCOL	YES	
ALTER RESOURCE QUEUE	YES	MPP数据库资源管理特性 - 不在PostgreSQL中。
ALTER ROLE	YES	MPP数据库子句： RESOURCE QUEUE <i>queue_name</i> none
ALTER SCHEMA	YES	
ALTER SEQUENCE	YES	
ALTER SYSTEM	NO	

SQL命令	MPP是否支持	修改，限制，例外
ALTER TABLE	YES	不支持的子句/选项： CLUSTER ON ENABLE/DISABLE TRIGGER MPP数据库子句： ADD DROP RENAME SPLIT EXCHANGE PARTITION SET SUBPARTITION TEMPLATE SET WITH (REORGANIZE=true false) SET DISTRIBUTED BY
ALTER TABLESPACE	YES	
ALTER TRIGGER	NO	
ALTER TYPE	YES	MPP数据库子句： SET DEFAULT ENCODING
ALTER USER	YES	ALTER ROLE 的别名
ALTER VIEW	YES	
ANALYZE	YES	
BEGIN	YES	
CHECKPOINT	YES	
CLOSE	YES	
CLUSTER	YES	

SQL命令	MPP是否支持	修改，限制，例外
COMMENT	YES	
COMMIT	YES	
COMMIT PREPARED	NO	
COPY	YES	修改的子句： ESCAPE [AS] '<i>escape</i>' 'OFF' MPP数据库子句： [LOG ERRORS] SEGMENT REJECT LIMIT <i>count</i> [ROWS PERCENT]
CREATE AGGREGATE	YES	不支持的子句/选项： [, SORTOP = <i>sort_operator</i>] MPP数据库子句： [, COMBINEFUNC = <i>combinefunc</i>] 限制： The functions used to implement the aggregate must be IMMUTABLE functions.
CREATE CAST	YES	
CREATE CONSTRAINT TRIGGER	NO	

SQL命令	MPP是否支持	修改，限制，例外
CREATE CONVERSION	YES	
CREATE DATABASE	YES	
CREATE DOMAIN	YES	
CREATE EXTENSION	YES	将新扩展加载到MPP数据库 - 基于PostgreSQL 9.6。
CREATE EXTERNAL TABLE	YES	MPP数据库并行ETL特性 - 不在PostgreSQL 9.4中。
CREATE FUNCTION	YES	限制： 定义为STABLE或VOLATILE的函数可以在MPP数据库中执行，只要它们仅在master上执行即可。 STABLE和VOLATILE函数不能在segment级别执行的语句中使用。
CREATE GROUP	YES	CREATE ROLE 的别名

SQL命令	MPP是否支持	修改，限制，例外
CREATE INDEX	YES	MPP数据库子句： USING bitmap (bitmap indexes) 限制： 仅当UNIQUE索引包含MPP分发键列的所有（或超集）时，才允许使用。在分区表上，仅在单个分区内支持唯一索引 - 而不是跨所有分区。 MPP不支持CONCURRENTLY关键字。
CREATE LANGUAGE	YES	
CREATE MATERIALIZED VIEW	NO	MPP数据库不支持物化视图（PostgreSQL扩展）。
CREATE OPERATOR	YES	限制： 用于实现运算符的函数必须是IMMUTABLE函数。
CREATE OPERATOR CLASS	YES	
CREATE OPERATOR FAMILY	YES	
CREATE PROTOCOL	YES	
CREATE RESOURCE QUEUE	YES	MPP数据库资源管理功能 - 不在PostgreSQL 9.4中。

SQL命令	MPP是否支持	修改，限制，例外
CREATE ROLE	YES	MPP 数据库子句： RESOURCE QUEUE <i>queue_name</i> none
CREATE RULE	YES	
CREATE SCHEMA	YES	
CREATE SEQUENCE	YES	限制： 不支持lastval()和currval()函数。 setval()函数仅在不对分布式数据进行操作的查询中允许。

SQL命令	MPP是否支持	修改，限制，例外
CREATE TABLE	YES	不支持的子句/选项： [GLOBAL LOCAL] REFERENCES FOREIGN KEY [DEFERRABLE NOT DEFERRABLE] 限制子句： UNIQUE或PRIMARY KEY约束仅允许在哈希分布式表（DISTRIBUTED BY）上使用，并且约束列必须与表的分发键列相同或超集，并且必须包括分区键的所有分发键列。 MPP数据库子句： DISTRIBUTED BY (<i>column</i>, [...]) DISTRIBUTED RANDOMLY PARTITION BY <i>type</i> (<i>column</i> [, ...]) (<i>partition_specification</i>, [...]) WITH (appendoptimized=true [,compresslevel=value,blocksize=value])
CREATE TABLE AS	YES	参考 CREATE TABLE

SQL命令	MPP是否支持	修改，限制，例外
CREATE TABLESPACE	YES	MPP数据库子句： 为特定的segment实例指定主机文件系统位置。 WITH (contentID_1='/path/to/dir1...')
CREATE TRIGGER	NO	
CREATE TYPE	YES	MPP数据库子句： COMPRESSTYPE COMPRESSLEVEL BLOCKSIZE 限制： 用于实现新基类型的函数必须是IMMUTABLE函数。
CREATE USER	YES	CREATE ROLE 的别名
CREATE VIEW	YES	
DEALLOCATE	YES	

SQL命令	MPP是否支持	修改，限制，例外
DECLARE	YES	不支持的子句/选项： SCROLL FOR UPDATE [OF column [, ...]] 限制： 游标不能向后滚动。支持向前滚动。 PL/pgSQL不支持可更新游标。
DELETE	YES	
DISCARD	YES	
DO	YES	PostgreSQL 9.0特性
DROP AGGREGATE	YES	
DROP CAST	YES	
DROP CONVERSION	YES	
DROP DATABASE	YES	
DROP DOMAIN	YES	
DROP EXTENSION	YES	从MPP数据库中删除扩展 - 基于PostgreSQL 9.6。
DROP EXTERNAL TABLE	YES	MPP数据库并行ETL功能 - 不在PostgreSQL 9.4中。
DROP FUNCTION	YES	
DROP GROUP	YES	DROP ROLE 的别名
DROP INDEX	YES	

SQL命令	MPP是否支持	修改，限制，例外
DROP LANGUAGE	YES	
DROP OPERATOR	YES	
DROP OPERATOR CLASS	YES	
DROP OPERATOR FAMILY	YES	
DROP OWNED	NO	
DROP PROTOCOL	YES	
DROP RESOURCE QUEUE	YES	MPP数据库资源管理功能 - 不在PostgreSQL 9.4中。
DROP ROLE	YES	
DROP RULE	YES	
DROP SCHEMA	YES	
DROP SEQUENCE	YES	
DROP TABLE	YES	
DROP TABLESPACE	YES	
DROP TRIGGER	NO	
DROP TYPE	YES	
DROP USER	YES	DROP ROLE 的别名
DROP VIEW	YES	
END	YES	
EXECUTE	YES	
EXPLAIN	YES	

SQL命令	MPP是否支持	修改，限制，例外
FETCH	YES	不支持的子句/选项： LAST PRIOR BACKWARD BACKWARD ALL 限制： 无法以非顺序方式获取行;不支持向后扫描。
GRANT	YES	
INSERT	YES	
LATERAL Join Type	NO	
LISTEN	NO	
LOAD	YES	
LOCK	YES	
MOVE	YES	参考 FETCH
NOTIFY	NO	
PREPARE	YES	
PREPARE TRANSACTION	NO	
REASSIGN OWNED	YES	

SQL命令	MPP是否支持	修改，限制，例外
REFRESH MATERIALIZED VIEW	NO	MPP数据库不支持物化视图（PostgreSQL扩展）。
REINDEX	YES	
RELEASE SAVEPOINT	YES	
RESET	YES	
REVOKE	YES	
ROLLBACK	YES	
ROLLBACK PREPARED	NO	
ROLLBACK TO SAVEPOINT	YES	
SAVEPOINT	YES	

SQL命令	MPP是否支持	修改，限制，例外
SELECT	YES	限制： 在FROM或WHERE子句中有限使用VOLATILE和STABLE函数 不支持文本搜索（Tsearch2） FETCH FIRST或FETCH NEXT子句不受支持 MPP数据库子句(OLAP): [GROUP BY <i>grouping_element</i> [, ...]] [WINDOW <i>window_name</i> AS (<i>window_specification</i>)] [FILTER (WHERE <i>condition</i>)]应用于SELECT列表中的聚合函数
SELECT INTO	YES	参考SELECT
SET	YES	
SET CONSTRAINTS	NO	在PostgreSQL中，这仅适用于外键约束，这些约束目前未在MPP数据库中实施。
SET ROLE	YES	
SET SESSION AUTHORIZATION	YES	从PostgreSQL 8.1开始不推荐 - 请参阅SET ROLE

SQL命令	MPP是否支持	修改，限制，例外
SET TRANSACTION	YES	限制： DEFERRABLE子句无效。 SET TRANSACTION SNAPSHOT命令不支持。
SHOW	YES	
START TRANSACTION	YES	
TRUNCATE	YES	
UNLISTEN	NO	
UPDATE	YES	限制： 不允许SET为MPP分发键列。
VACUUM	YES	限制： MPP数据库不推荐使用VACUUM FULL。
VALUES	YES	