

PostgreSQL 常用命令速查表

原创

版权

分类专栏：

PostgreSQL

文章标签：

postgresql

常用命令

常用语句

速查表

 PostgreSQL 专栏收录该内

24 订阅

56 篇文章

订阅专栏

| PostgreSQL 常用命令速查表 | | |
|---|---|---|
| 不剪发的 Tony 老师 | | |
| PostgreSQL 是世界上最先进的开源对象-关系型数据库，遵循类似 BSD/MIT 的开源协议，以其强大功能、高性能、可靠性以及可扩展性在企业中被广泛应用。PostgreSQL 支持各种主流的平台，例如 Linux、BSD、AIX、HP-UX、Mac OS X 以及 Windows 等。PostgreSQL 遵循事务的 ACID 原则，高度兼容 SQL 标准；同时支持自定义数据类型、索引类型、过程语言扩展（PL/pgSQL、PL/Python、PL/Java 等）。 | | |
| 连接服务器 | 角色、用户和组 | 数据库和模式 |
| psql -h <i>hostname</i> -p <i>port</i> -U <i>username</i> <i>dbname</i> postgres=# \? [<i>topic</i>] -- 获取 psql 工具帮助 postgres=# \h <i>command</i> -- 获取 SQL 命令帮助 postgres=# \conninfo -- 显示当前连接信息 postgres=# \l -- 列出所有的数据库 postgres=# \q -- 退出 psql 客户端 SELECT version(); -- 查看服务器版本 SHOW (ALL <i>name</i>) -- 查看运行时参数 SET [<i>SESSION</i> <i>LOCAL</i>] <i>name</i> TO { <i>value</i> <i>value</i> DEFAULT } -- 设置运行时参数 SELECT pg_reload_conf(); -- 重新加载配置文件 SELECT * FROM pg_stat_activity; -- 查询所有连接 SELECT pg_cancel_backend(<i>pid</i>); -- 取消运行中的查询 SELECT pg_terminate_backend(<i>pid</i>); -- 终止指定连接 | CREATE USER <i>name</i> PASSWORD ' <i>password</i> '; -- 创建用户 SELECT * FROM pg_user; -- 查看所有用户 SELECT user, current_user; -- 查看当前用户 ALTER USER <i>name</i> PASSWORD ' <i>password</i> '; -- 修改密码 ALTER USER <i>name</i> VALID UNTIL ' <i>timestamp</i> '; -- 设置密码失效时间 DROP USER [IF EXISTS] <i>name</i> ; -- 删除用户 GRANT <i>privilege list</i> ALL ON <i>object_type</i> TO <i>user</i> ; -- 授予权限 REVOKE <i>privilege list</i> ALL ON <i>object_type</i> FROM <i>user</i> ; -- 撤销权限 SELECT * FROM role, table, grants; -- 查看表的授权 备注：PostgreSQL 角色又包含了两种概念，具有登录权限的角色称为用户，包含其他成员的角色称为组（group）。因此，以上语句中 USER 关键字可以替换为 ROLE 或者 GROUP。 | CREATE DATABASE <i>name</i> [WITH OWNER = <i>user</i>]; -- 创建数据库 SELECT * FROM pg_database; -- 查看数据库 ALTER DATABASE <i>name</i> ...; -- 修改数据库 DROP DATABASE [IF EXISTS] <i>name</i> ; -- 删除数据库 postgres=# \dn -- 查看当前数据库中的模式 SELECT * FROM pg_namespaces; -- 查看所有模式 CREATE SCHEMA [IF NOT EXISTS] <i>name</i> [AUTHORIZATION <i>user</i>]; -- 创建模式 ALTER SCHEMA <i>name</i> RENAME TO <i>new_name</i> ; -- 修改模式 ALTER SCHEMA <i>name</i> OWNER TO <i>new_owner</i> ; -- 删除模式 SHOW search_path; -- 查看模式搜索路径 SET search_path TO <i>schemat1</i> , <i>schemat2</i> ...; -- 设置模式搜索路径 DROP SCHEMA [IF EXISTS] <i>name</i> ; -- 删除模式 |

文章目录

连接服务器

- 查看帮助
- 查看连接
- 查看版本
- 配置参数
- 退出客户端

角色、用户和组

- 创建角色
- 查看角色
- 修改密码
- 设置密码失效时间
- 用户授权
- 查看权限
- 撤销权限
- 设置当前角色
- 删除角色

数据库和模式

- 查看数据库
- 创建数据库
- 修改数据库
- 删除数据库
- 查看模式
- 创建模式
- 修改模式
- 模式搜索路径
- 删除模式

管理数据表

- 创建表
- 查看所有表
- 查看表结构
- 增加字段
- 修改字段
- 重命名字段
- 删除字段
- 增加约束
- 删除约束
- 重命名表
- 删除表

索引

- 创建索引
- 查看索引
- 维护索引
- 删除索引

管理表空间

- 创建表空间
- 查看表空间
- 修改表空间
- 删除表空间

备份与还原

- 使用 pg_dump 执行逻辑备份
- 使用 psql/pg_restore 执行还原
- 备份/还原整个数据库集群
- 使用 COPY 导入/导出表数据

查询语句

- 单表查询
- 查询条件
- 排序显示
- 限定数量
- 分组操作
- 多表连接
- 子查询
- 集合运算
- 通用表表达式

DML 语句

- 插入数据
- 更新数据
- 删除数据
- 合并数据

事务控制

- 开始事务
- 提交事务
- 回滚事务
- 事务保存点
- 隔离级别

视图

[创建视图](#)[查看所有视图](#)[查看视图定义](#)[修改视图](#)[删除视图](#)

存储过程/函数

[创建存储过程/函数](#)[调用存储过程/函数](#)[重命名存储过程/函数](#)[删除存储过程/函数](#)

触发器

[创建触发器](#)[查看触发器](#)[修改触发器](#)[启用/禁用触发器](#)[删除触发器](#)

大家好，我是只谈技术不剪发的 Tony 老师。本文为大家精心整理了 [PostgreSQL](#) 数据库中最常用的语句和命令，点击收藏以备不时之需！如果需要 PDF 版本，可以[点此下载](#)。

如果你点击了收藏，也欢迎关注❤️、评论📝、点赞👍

连接服务器

使用 `psql` 客户端工具连接 PostgreSQL 服务器的命令行如下：

```
1 | psql -h hostname -p port -U username -W dbname
```

其中，hostname 表示服务器主机名，默认为本机；port 表示 PostgreSQL 实例服务端口，默认为 5432；username 表示用户名，默认为当前操作系统用户；-W 提示输入密码，默认为行；dbname 表示要连接的数据库，默认和用户名相同。例如：

```
1 | [tony@sqlhost ~]> psql -h 192.168.56.104 -p 5432 -U tony hrdb
2 | Password for user tony:
3 | psql (12.4)
4 | Type "help" for help.
5 |
6 | hrdb=>
```

以上命令使用 tony 用户通过 5432 端口连接到服务器 192.168.56.104 上的 hrdb 数据库。

查看帮助

在 psql 提示符中输入 `help` 获取使用帮助。

```
1 | hrdb=> help
2 | You are using psql, the command-line interface to PostgreSQL.
3 | Type: \copyright for distribution terms
4 |       \h for help with SQL commands
5 |       \? for help with psql commands
6 |       \g or terminate with semicolon to execute query
7 |       \q to quit
```

使用 `\?` 命令可以获取 psql 命令相关的帮助信息：

```

1 | hrdb=> \?
2 | General
3 | \copyright      show PostgreSQL usage and distribution terms
4 | \crosstabview [COLUMNS] execute query and display results in crosstab
5 | \errverbose     show most recent error message at maximum verbosity
6 | \g [FILE] or ;  execute query (and send results to file or |pipe)
7 | \gdesc         describe result of query, without executing it
8 | \gexec         execute query, then execute each value in its result
9 | \gset [PREFIX] execute query and store results in psql variables
10 | \gx [FILE]      as \g, but forces expanded output mode
11 | \q             quit psql
12 | \watch [SEC]   execute query every SEC seconds
13 |
14 | Help
15 | \? [commands]  show help on backslash commands
16 | \? options     show help on psql command-line options
17 | \? variables   show help on special variables
18 | \h [NAME]      help on syntax of SQL commands, * for all commands
19 | ...

```

使用 `\h` 命令可以获取 SQL 命令相关的帮助信息：

```

1 | hrdb=> \h
2 | Available help:
3 | ABORT          CREATE FOREIGN DATA WRAPPER  DROP ROUTINE
4 | ALTER AGGREGATE CREATE FOREIGN TABLE      DROP RULE
5 | ALTER COLLATION CREATE FUNCTION           DROP SCHEMA
6 | ALTER CONVERSION CREATE GROUP             DROP SEQUENCE
7 | ALTER DATABASE  CREATE INDEX             DROP SERVER
8 | ALTER DEFAULT PRIVILEGES CREATE LANGUAGE   DROP STATISTICS
9 | ...
10 |
11 | hrdb=> \h insert
12 | Command:  INSERT
13 | Description: create new rows in a table
14 | Syntax:
15 | [ WITH [ RECURSIVE ] with_query [, ...] ]
16 | INSERT INTO table_name [ AS alias ] [ ( column_name [, ...] ) ]
17 |   [ OVERRIDING { SYSTEM | USER } VALUE ]
18 |   { DEFAULT VALUES | VALUES ( { expression | DEFAULT } [, ...] ) [, ...] | query }
19 |   [ ON CONFLICT [ conflict_target ] conflict_action ]
20 |   [ RETURNING * | output_expression [ [ AS ] output_name ] [, ...] ]
21 |
22 | where conflict_target can be one of:
23 |
24 | ( { index_column_name | ( index_expression ) } [ COLLATE collation ] [ opclass ] |
25 |   ON CONSTRAINT constraint_name
26 |
27 | and conflict_action is one of:
28 |
29 | DO NOTHING
30 | DO UPDATE SET { column_name = { expression | DEFAULT } |
31 |   ( column_name [, ...] ) = [ ROW ] ( { expression | DEFAULT } [, ...] ) |
32 |   ( column_name [, ...] ) = ( sub-SELECT )
33 |   } [, ...]
34 |   [ WHERE condition ]
35 |
36 | URL: https://www.postgresql.org/docs/12/sql-insert.html

```

查看连接

使用 psql 中的 `\conninfo` 命令可以查看当前连接信息。

```

1 | hrdb=> \conninfo
2 | You are connected to database "hrdb" as user "tony" on host "192.168.56.104" at

```

查询系统视图 `pg_catalog.pg_stat_activity` 可以列出所有的服务器进程。

```

1 | hrdb=> SELECT datname, pid, username, client_addr, wait_event, state, backend_
2 | hrdb-> FROM pg_catalog.pg_stat_activity;
3 | datname | pid | username | client_addr | wait_event | state | backend_t
4 | -----+-----+-----+-----+-----+-----+-----
5 | | 1734 | | | AutoVacuumMain | | autovacuum launcher
6 | | 1736 | postgres | | LogicalLauncherMain | | logical replication laun
7 | postgres | 27031 | postgres | | active | client backend
8 | hrdb | 27898 | postgres | 192.168.56.1 | ClientRead | idle | client backend
9 | hrdb | 27900 | postgres | 192.168.56.1 | ClientRead | idle | client backend
10 | hrdb | 27902 | postgres | 192.168.56.1 | ClientRead | idle | client backend
11 | | 1732 | | | BgWriterHibernate | | background writer
12 | | 1731 | | | CheckpointerMain | | checkpointer
13 | | 1733 | | | WalWriterMain | | walwriter
14 | (9 rows)

```

psql 中的 SQL 命令以 ; 或者 \g 结束。其中, client backend 是客户端的连接进程。

如果想要取消正在运行中的查询, 可以执行以下语句:

```
1 | SELECT pg_cancel_backend(pid);
```

其中, pid 是执行语句的后台进程 id, 通过上文中的查询可以获得。

如果想要强制终止某个后台连接进程, 可以执行以下语句:

```
1 | SELECT pg_terminate_backend(pid);
```

其中, pid 是执行语句的后台进程 id, 通过上文中的查询可以获得。

 关于如何查看和终止 PostgreSQL 连接会话的详细内容, 可以参考[这篇文章](#)。

查看版本

例如, 以下语句可以查看 PostgreSQL 服务器的版本:

```

1 | hrdb=> SELECT version();
2 | version
3 | -----
4 | PostgreSQL 12.4 on x86_64-pc-linux-gnu, compiled by gcc (GCC) 4.8.5 20150623 (
5 | (1 row)

```

配置参数

PostgreSQL 运行时的参数设置可以通过 SHOW 语句进行查看:

```

1 | SHOW name
2 | SHOW ALL

```

例如:

```

1 | hrdb=> show shared_buffers;
2 | shared_buffers
3 | -----
4 | 256MB
5 | (1 row)

```

运行时的参数可以通过 SET 语句、修改 postgresql.conf 配置文件、设置 PGOPTIONS 环境变量（使用 libpq 或者基于 libpq 应用连接）或者启动服务时的命令行参数进行设置。

使用 SET 语句设置参数的命令如下:

```
1 | SET [ SESSION | LOCAL ] configuration_parameter { TO | = } { value | 'value' | DEFAL
```

如果通过配置文件修改了参数，可以使用以下语句重新加载配置，而不需要重启服务：

```
1 | SELECT pg_reload_conf();
```

 关于 PostgreSQL 服务器参数的配置和优化，可以参考[这篇文章](#)。

退出客户端

使用 `exit`、`quit` 或者 `\q` 命令退出 `psql` 客户端：

```
1 | postgres=# \q
2 | [root@sqlhost ~]#
```

 关于 PostgreSQL 客户端工具 `psql` 的详细使用，可以参考[这篇文章](#)。

角色、用户和组

PostgreSQL 通过角色的概念来控制数据库的访问权限。角色又包含了两种概念，具有登录权限的角色称为用户，包含其他成员（也是角色）的角色称为组（group）。因此，一个角色可以是一个用户，也可以是一个组，或者两者都是。

 PostgreSQL 角色与用户管理的详细内容可以参考[这篇文章](#)。

创建角色

使用 `CREATE ROLE` 语句创建一个角色：

```
1 | CREATE ROLE role_name [[ WITH ] option [ ... ]]
2 |
3 | option:
4 |     SUPERUSER | NOSUPERUSER
5 | | CREATEDB | NOCREATEDB
6 | | CREATEROLE | NOCREATEROLE
7 | | INHERIT | NOINHERIT
8 | | LOGIN | NOLOGIN
9 | | REPLICATION | NOREPLICATION
10 | | BYPASSRLS | NOBYPASSRLS
11 | | CONNECTION LIMIT connlimit
12 | | [ ENCRYPTED ] PASSWORD 'password' | PASSWORD NULL
13 | | VALID UNTIL 'timestamp'
14 | | IN ROLE role_name [ , ... ]
15 | | IN GROUP role_name [ , ... ]
16 | | ROLE role_name [ , ... ]
17 | | ADMIN role_name [ , ... ]
18 | | USER role_name [ , ... ]
19 | | SYSID uid
```

其中，`role_name` 表示角色名、用户名或者组名；如果指定了 `LOGIN` 属性表示创建用户，`PASSWORD` 属性用于指定用户密码。例如：

```
1 | CREATE ROLE tony LOGIN PASSWORD 'Pswd#123';
2 |
3 | CREATE GROUP tony LOGIN PASSWORD 'Pswd#123';
4 |
5 | CREATE USER tony PASSWORD 'Pswd#123';
```

以上语句创建了一个角色 `tony`，并且设置了登录密码。

 `CREATE USER` 和 `CREATE GROUP` 都是 `CREATE ROLE` 的同义词，只是 `CREATE USER` 默认指定了 `LOGIN` 选项。

一个角色被创建之后，可以通过 **ALTER ROLE** 语句修改它的属性。例如，以下命令为用户 tony 设置了创建数据库的权限：

```
1 | ALTER ROLE tony createdb;
2 |
3 | ALTER USER tony createdb;
```

查看角色

PostgreSQL 中的角色信息存储在系统视图 pg_catalog.pg_roles 中：

```
1 | SELECT rolname, rolsuper, rolcanlogin
2 | FROM pg_catalog.pg_roles;
3 |
4 | rolname          | rolsuper | rolcanlogin |
5 | -----/-----/-----/
6 | pg_monitor       | false   | false      |
7 | pg_read_all_settings | false   | false      |
8 | pg_read_all_stats  | false   | false      |
9 | pg_stat_scan_tables | false   | false      |
10 | pg_read_server_files | false   | false      |
11 | pg_write_server_files | false   | false      |
12 | pg_execute_server_program | false   | false      |
13 | pg_signal_backend  | false   | false      |
14 | postgres         | true    | true       |
15 | monitor_system_stats | false   | false      |
16 | tony              | false   | true       |
```

查询结果中大部分都是系统创建的角色。另外，也可以通过 pg_catalog.pg_user 查看用户信息，或者使用 psql 中的 \du 命令列出角色。

user 或者 current_user 函数可以用于查看当前的用户：

```
1 | SELECT current_user;
2 |
3 | current_user |
4 | -----/
5 | postgres    |
```

修改密码

使用 **ALTER ROLE** 语句修改角色的密码：

```
1 | ALTER ROLE tony PASSWORD 'Pswd123@';
2 |
3 | ALTER USER tony PASSWORD 'Pswd123@';
```

另外，也可以使用 psql 中的 \password [USERNAME] 命令修改用户的密码。

设置密码失效时间

使用 **ALTER ROLE** 语句设置密码的失效时间：

```
1 | ALTER ROLE tony VALID UNTIL '2020-12-12 00:00:00';
2 |
3 | ALTER USER tony VALID UNTIL 'infinity';
```

第一个语句将 tony 的密码失效时间设置为 2020 年 12 月 12 日零点；第二个语句取消 tony 的密码失效时间，意味着永远有效。

用户授权

PostgreSQL 使用 **GRANT** 语句进行数据库对象的授权操作。以表为例，基本的授权语法如下：

```
1 | GRANT privilege_list | ALL
2 | ON [ TABLE ] table_name
```

```
3 | TO role_name;
```

其中, privilege_list 权限列表可以是 SELECT、INSERT、UPDATE、DELETE、TRUNCATE 等, ALL 表示表上的所有权限。例如, 以下语句将 employees、departments 和 jobs 表上的增删改查权限授予了 tony 用户:

```
1 | GRANT SELECT, INSERT, UPDATE, DELETE
2 | ON employees, departments, jobs
3 | TO tony;
```

另一种授权方式是将某个组角色的成员资格授予其他用户, 使得这些用户可以自动获得该角色的权限。例如:

```
1 | GRANT monitor_system_stats TO tony;
2 |
3 | ALTER GROUP monitor_system_stats ADD USER tony;
```

以上语句将用户 tony 添加为角色 monitor_system_stats 的成员。

查看权限

系统视图 information_schema.role_table_grants 或者 information_schema.table_privileges 包含了授予用户的表和视图权限:

```
1 | SELECT table_catalog, table_schema, table_name, privilege_type
2 | FROM information_schema.role_table_grants
3 | WHERE grantee = 'tony';
4 |
5 | table_catalog|table_schema|table_name |privilege_type|
6 | -----/-----/-----/-----/
7 | hrdb        |public    |employees |INSERT  |
8 | hrdb        |public    |employees |SELECT  |
9 | hrdb        |public    |employees |UPDATE  |
10 | hrdb        |public    |employees |DELETE  |
11 | ...
```

其他对象的权限可以通过 information_schema 视图 role_column_grants (column_privileges)、role_routine_grants (routine_privileges)、role_udt_grants (udt_privileges)、role_usage_grants (usage_privileges) 等进行查看。

撤销权限

PostgreSQL 使用 REVOKE 语句撤销数据库对象上的权限。同样以表为例, 基本的撤销授权语句如下:

```
1 | REVOKE privilege_list | ALL
2 | ON TABLE table_name
3 | FROM role_name;
```

其中的参数与 GRANT 语句相同。以下语句撤销了用户 tony 对 employees、departments 和 jobs 表的增删改查权限:

```
1 | REVOKE SELECT, INSERT, UPDATE, DELETE
2 | ON employees, departments, jobs
3 | FROM tony;
```

同样, 可以使用 REVOKE 语句撤销某个用的成员资格。例如以下语句撤销了用户 tony 的 monitor_system_stats 成员资格:

```
1 | REVOKE monitor_system_stats
2 | FROM tony;
3 |
4 | ALTER GROUP monitor_system_stats DROP USER tony;
```


设置当前角色

SET ROLE 语句可以设置当前会话的用户 ID。例如：

```
1 | SELECT session_user, current_user;
2 | session_user|current_user|
3 | -----/-----/
4 | postgres |postgres |
5 |
6 | SET ROLE tony;
7 |
8 | SELECT session_user, current_user;
9 | session_user|current_user|
10| -----/-----/
11| postgres |tony |
```

以上语句将当前用户设置为 tony。此时，该会话只拥有 tony 用户的权限，不再拥有超级用户权限。可以使用以下语句要恢复初始状态的会话权限：

```
1 | SET ROLE NONE;
2 |
3 | RESET ROLE;
```

删除角色

删除角色可以使用 DROP ROLE 语句。例如以下语句删除了用户 tony：

```
1 | DROP ROLE IF EXISTS tony;
```

另外，DROP USER 和 DROP GROUP 语句都是 DROP ROLE 的同义词。

数据库和模式

PostgreSQL 中的数据库（Database）由一组相关的对象组成，例如表、索引、视图、存储过程 等。数据库中的对象使用模式（Schema）进行组织。因此，一个数据库由多个模式组成，一个模式由许多对象组成。



PostgreSQL 数据库和模式管理的详细内容可以参考[这篇文章](#)。

查看数据库

使用 psql 工具的 \l 命令可以查看当前集群中的数据库：

```
1 | postgres=# \l
2 |          List of databases
3 |  Name | Owner | Encoding | Collate | Ctype | Access privileges
4 | -----+-----+-----+-----+-----+-----
5 | ds2   | postgres | UTF8 | en_US.UTF-8 | en_US.UTF-8 |
6 | hrdb  | tony   | UTF8 | en_US.UTF-8 | en_US.UTF-8 |
7 | pagila | postgres | UTF8 | en_US.UTF-8 | en_US.UTF-8 |
8 | postgres | postgres | UTF8 | en_US.UTF-8 | en_US.UTF-8 |
9 | template0 | postgres | UTF8 | en_US.UTF-8 | en_US.UTF-8 | =c/postgres +
10|      |      |      |      |      | postgres=Ct/postgres
11| template1 | postgres | UTF8 | en_US.UTF-8 | en_US.UTF-8 | =c/postgres +
12|      |      |      |      |      | postgres=Ct/postgres
13| (6 rows)
```

也可以通过 pg_catalog.pg_database 查看所有的数据库：

```
1 | SELECT datname
2 | FROM pg_catalog.pg_database;
3 | datname |
4 | -----/
5 | postgres |
6 | template1|
7 | template0|
```

```
8 | pagila |
9 | ds2 |
10 | hrdb |
```

创建数据库

PostgreSQL 使用 **CREATE DATABASE** 语句创建数据库：

```
1 | CREATE DATABASE db_name
2 |   [[ WITH ] [ OWNER [=] user_name ]
3 |     [ TEMPLATE [=] template ]
4 |     [ ENCODING [=] encoding ]
5 |     [ LC_COLLATE [=] lc_collate ]
6 |     [ LC_CTYPE [=] lc_ctype ]
7 |     [ TABLESPACE [=] tablespace_name ]
8 |     [ ALLOW_CONNECTIONS [=] allowconn ]
9 |     [ CONNECTION LIMIT [=] connlimit ]
10 |     [ IS_TEMPLATE [=] istemplate ] ]
```

其中，db_name 是数据库的名称，OWNER 用于指定数据库的拥有者，TEMPLATE 是创建数据库时使用的模板（默认使用 template1 数据库），ENCODING 用于设置数据库的字符集编码。

例如，以下语句创建了一个名为 testdb 的数据库：

```
1 | CREATE DATABASE testdb;
```

修改数据库

PostgreSQL 使用 **ALTER DATABASE** 语句修改数据库的属性和配置：

```
1 | ALTER DATABASE db_name RENAME TO new_name
2 |
3 | ALTER DATABASE db_name OWNER TO { new_owner | CURRENT_USER | SESSION_
4 |
5 | ALTER DATABASE db_name SET TABLESPACE new_tablespace
6 |
7 | ALTER DATABASE db_name [[ WITH ] option [ ... ] ]
8 |
9 | option:
10 |   ALLOW_CONNECTIONS allowconn
11 |   CONNECTION LIMIT connlimit
12 |   IS_TEMPLATE istemplate
```

例如，以下语句将 testdb 的名称修改为 newdb：

```
1 | ALTER DATABASE testdb RENAME TO newdb;
```

除了修改常见的属性之外，ALTER DATABASE 语句还可以用于修改运行时配置变量的会话默认值：

```
1 | ALTER DATABASE db_name SET configuration_parameter { TO | = } { value | DEFAU
2 | ALTER DATABASE db_name RESET configuration_parameter
```

例如，以下语句将会默认禁用数据库 newdb 中的索引扫描：

```
1 | ALTER DATABASE newdb SET enable_indexscan TO off;
```

删除数据库

PostgreSQL 使用 **DROP DATABASE** 语句删除数据库，该数据库中的所有对象以及与该数据库相关的数据目录也会被删除：

```
1 | DROP DATABASE [IF EXISTS] db_name;
```

我们将 newdb 数据库和相关的数据库删除：

```
1 | DROP DATABASE newdb;
```

如果存在任何目标为该数据库的连接，无法执行删除操作。

查看模式

创建了数据库之后，还需要创建模式才能够存储数据库对象。通常在创建一个新的数据库时，默认会创建一个公共模式 public。我们再次创建一个数据库 testdb：

```
1 | CREATE DATABASE testdb;
2 |
3 | postgres=# \c testdb;
4 | You are now connected to database "testdb" as user "postgres".
```

其中，\c 用于连接数据库。然后使用 \dn 查看当前数据库中的模式：

```
1 | testdb=# \dn
2 | List of schemas
3 | Name | Owner
4 | -----+-----
5 | public | postgres
6 | (1 row)
```

另外，也可以系统视图 pg_catalog.pg_namespace 查询所有的模式：

```
1 | SELECT *
2 | FROM pg_catalog.pg_namespace;
3 | oid |nspname |nspowner|nspacl |
4 | ----+-----+-----+-----+
5 | 99 |pg_toast | 10|NULL |
6 | 12314 |pg_temp_1 | 10|NULL |
7 | 12315 |pg_toast_temp_1 | 10|NULL |
8 | 11 |pg_catalog | 10|{postgres=UC/postgres,=U/postgres} |
9 | 2200 |public | 10|{postgres=UC/postgres,=UC/postgres} |
10 | 13887 |information_schema | 10|{postgres=UC/postgres,=U/postgres} |
```

查询结果还返回了系统提供的其他模式。

创建模式

PostgreSQL 使用 CREATE SCHEMA 语句创建一个新的模式：

```
1 | CREATE SCHEMA IF NOT EXISTS schema_name [ AUTHORIZATION role_name ]
```

其中，schema_name 是模式名，role_name 是模式的拥有者，默认为执行该语句的用户。例如：

```
1 | CREATE SCHEMA app AUTHORIZATION tony;
```

该语句创建了一个新的模式 app，拥有者为 tony。

修改模式

如果需要修改已有模式的属性，可以使用 ALTER SCHEMA 语句：

```
1 | ALTER SCHEMA schema_name RENAME TO new_name
2 | ALTER SCHEMA schema_name OWNER TO new_owner
```

例如，以下语句将模式 app 重命名为 sale：

```
1 | ALTER SCHEMA app RENAME TO sale;
```

模式搜索路径

当我们访问数据表时，完整的对象名应该是 database.schema.table。例如：

```
1 | SELECT count(*) FROM hrdb.public.employees;
```

为了方便书写，常常可以直接使用表名进行访问，此时 PostgreSQL 按照预定义的搜索路径查找不同模式下的对象。搜索路径由不同的模式名称组成，可以使用 SHOW 语句查看：

```
1 | show search_path;  
2 | search_path  
3 | -----  
4 | "$user", public  
5 | (1 row)
```

默认的搜索路径是与当前用户同名的模式，加上 public 模式。不仅仅是对象查找，其他语句也会使用搜索路径。例如 CREATE TABLE 默认会在 “\$user” 模式下创建表，如果该模式不存在，则在 public 模式下创建。

可以使用 SET 命令修改搜索路径，例如：

```
1 | SET search_path TO sale,public;
```

删除模式

使用 DROP SCHEMA 语句删除模式：

```
1 | DROP SCHEMA [ IF EXISTS ] schema_name [ CASCADE | RESTRICT ]
```

如果模式中存在对象，需要使用 CASCADE 级联删除；否则，无法删除模式。例如，以下语句将会删除模式 sale：

```
1 | DROP SCHEMA sale;
```

管理数据表

 关于 PostgreSQL 表管理的详细内容可以参考[这篇文章](#)。

创建表

PostgreSQL 使用 CREATE TABLE 语句创建表：

```
1 | CREATE TABLE [ IF NOT EXISTS ] table_name  
2 | (  
3 | column_name data_type column_constraint,  
4 | column_name data_type,  
5 | ...,  
6 | table_constraint  
7 | );
```

其中，table_name 指定了新表的名称；括号内是字段的定义，column_name 是字段的名称，data_type 是它的类型，column_constraint 是可选的字段约束；多个字段使用逗号进行分隔；最后，table_constraint 是可选的表级约束。

以下语句用于创建部门表和员工表：

```
1 | CREATE TABLE departments  
2 | ( department_id INTEGER NOT NULL PRIMARY KEY  
3 | , department_name CHARACTER VARYING(30) NOT NULL  
4 | );  
5 |  
6 | CREATE TABLE employees  
7 | ( employee_id INTEGER NOT NULL  
8 | , first_name CHARACTER VARYING(20)  
9 | , last_name CHARACTER VARYING(25) NOT NULL
```

```
10 | ,email CHARACTER VARYING(25) NOT NULL
```



除了自定义表的结构之外，PostgreSQL 还提供了另一个创建表的方法，就是通过一个查询的结果创建新表：

```
1 | CREATE TABLE [ IF NOT EXISTS ] table_name
2 | AS query
3 | [ WITH [ NO ] DATA ];
```

例如，我们可以基于 employees 复制出一个新的表：

```
1 | CREATE TABLE emp1
2 | AS
3 | SELECT *
4 | FROM employees;
```

emp1 只会复制 employees 的字段类型和名称，以及查询的结果，不会复制任何约束和索引。

查看所有表

在 psql 中使用 \d 命令可以查看当前数据库中的所有表：

```
1 | testdb=# \d
2 |      List of relations
3 | Schema | Name      | Type | Owner
4 | -----+-----+-----+-----
5 | public | departments | table | postgres
6 | public | emp1       | table | postgres
7 | public | employees  | table | postgres
8 | (3 rows)
```

也可以通过 information_schema.tables 查看表的信息：

```
1 | SELECT table_schema, table_name
2 | FROM information_schema.tables
3 | WHERE table_type = 'BASE TABLE';
```

查看表结构

PostgreSQL 可以使用 psql 工具的 \d 命令查看表结构：

```
1 | testdb=# \d employees
2 |      Table "public.employees"
3 | Column | Type          | Collation | Nullable | Default
4 | -----+-----+-----+-----+-----
5 | employee_id | integer      |           | not null |
6 | first_name  | character varying(20) |           |          |
7 | last_name   | character varying(25) |           | not null |
8 | email       | character varying(25) |           | not null |
9 | phone_number | character varying(20) |           |          |
10 | hire_date   | date          |           | not null |
11 | salary      | numeric(8,2)   |           |          |
```



增加字段

增加字段使用 ALTER TABLE ... ADD COLUMN 语句：

```
1 | ALTER TABLE table_name
2 | ADD COLUMN column_name data_type column_constraint;
```

添加字段与创建表时的字段选项相同，包含字段名称、字段类型以及可选的约束。以下语句为 `employees` 表增加一个字段：

```
1 | ALTER TABLE employees ADD COLUMN description varchar(100);
```

修改字段

修改字段使用 `ALTER TABLE ... ALTER COLUMN` 语句。以下语句为字段 `description` 设置了一个默认值：

```
1 | ALTER TABLE employees ALTER COLUMN description SET DEFAULT 'No description';
```

以下语句将字段 `description` 的数据类型修改为 `text`：

```
1 | ALTER TABLE employees ALTER COLUMN description TYPE text;
```

重命名字段

使用 `ALTER TABLE ... RENAME COLUMN` 语句为字段指定一个新的名称，以下语句将 `description` 字段重命名为 `notes`：

```
1 | ALTER TABLE employees
2 | RENAME COLUMN description TO notes;
```

删除字段

删除字段使用 `ALTER TABLE ... DROP COLUMN` 语句，以下语句用于删除 `notes` 字段：

```
1 | ALTER TABLE employees DROP COLUMN IF EXISTS notes;
```

增加约束

为表增加约束可以使用 `ALTER TABLE` 语句：

```
1 | ALTER TABLE table_name ADD table_constraint;
2 |
3 | ALTER TABLE table_name ALTER COLUMN column_name SET DEFAULT expression;
4 |
5 | ALTER TABLE table_name ALTER COLUMN column_name SET NOT NULL;
```

删除约束

删除约束同样可以使用 `ALTER TABLE` 语句：

```
1 | ALTER TABLE table_name DROP CONSTRAINT IF EXISTS constraint_name [ RESTRICT ];
2 |
3 | ALTER TABLE table_name ALTER COLUMN column_name DROP DEFAULT;
4 |
5 | ALTER TABLE table_name ALTER COLUMN column_name DROP NOT NULL;
```

重命名表

重命名表可以使用 `ALTER TABLE ... DROP COLUMN` 语句：

```
1 | ALTER TABLE [ IF EXISTS ] table_name
2 | RENAME TO new_name;
```

删除表

删除表可以使用 `DROP TABLE` 语句：

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

如果存在依赖于 table_name 的视图或外键约束，需要指定 CASCADE 选项，执行级联删除。

索引

索引（Index）可以用于提高数据库的查询性能；但是索引也需要进行读写，同时还会占用更多的存储空间；因此了解并适当利用索引对于数据库的优化至关重要。

 关于 PostgreSQL 索引和优化的详细内容可以参考[这篇文章](#)。

创建索引

PostgreSQL 为主键和唯一约束自动创建相应的索引，另外我们也可以手动创建索引。创建索引的命令如下：

```
1 | CREATE INDEX index_name ON table_name
2 | [USING method]
3 | (column_name [ASC | DESC] [NULLS FIRST | NULLS LAST], ...);
```

其中 index_name 是索引的名称，table_name 是表的名称；method 表示索引的类型，例如 btree、hash、gist、spgist、gin 或者 brin。默认为 btree；column_name 是字段名，ASC 表示升序排序（默认值），DESC 表示降序索引；NULLS FIRST 和 NULLS LAST 表示索引中空值的排列顺序，升序索引时默认为 NULLS LAST，降序索引时默认为 NULLS FIRST。

查看索引

PostgreSQL 提供了一个关于索引的视图 pg_catalog.pg_indexes，可以用于查看索引的信息：

```
1 | SELECT * FROM pg_catalog.pg_indexes;
```

psql 工具的 \di 命令也可以用于查看数据库中的索引列表。

维护索引

PostgreSQL 提供了一些修改和重建索引的方法：

```
1 | ALTER INDEX index_name RENAME TO new_name;
2 | ALTER INDEX index_name SET TABLESPACE tablespace_name;
3 |
4 | REINDEX [ ( { VERBOSE } ) ] { INDEX | TABLE | SCHEMA | DATABASE | SYSTEM } index_n
```

其中，两个 ALTER INDEX 语句分别用于重命名索引和移动索引到其他 **表空间**；REINDEX 语句用于重建索引数据，支持不同级别的索引重建。

删除索引

删除索引使用以下命令：

```
1 | DROP INDEX IF EXISTS index_name [ CASCADE | RESTRICT ];
```

CASCADE 表示级联删除其他依赖该索引的对象；RESTRICT（默认值）表示如果存在依赖于该索引的对象，将会拒绝删除操作。

管理表空间

在 PostgreSQL 中，表空间（tablespace）表示数据文件的存放目录，这些数据文件代表了数据库的对象，例如表或索引。当我们访问表时，系统通过它所在的表空间定位到对应数据文件所在的位置。

 PostgreSQL 表空间管理的详细内容可以参考[这篇文章](#)。

创建表空间

创建新的表空间使用 `CREATE TABLESPACE` 语句：

```
1 | CREATE TABLESPACE tablespace_name
2 | OWNER user_name
3 | LOCATION 'directory';
```

表空间的名称不能以 ‘pg_’ 开头，它们是系统表空间的保留名称；LOCATION 参数必须指定绝对路径名，指定的目录必须是一个已经存在的空目录，PostgreSQL 操作系统用户（postgres）必须是该目录的拥有者，以便能够进行文件的读写。

 PostgreSQL 支持在 CREATE DATABASE、CREATE TABLE、CREATE INDEX 以及 ADD CONSTRAINT 语句中指定 tablespace_name 选项，覆盖默认的表空间（pg_default）；也可以使用相应的 ALTER ... 语句将对象从一个表空间移到另一个表空间。

查看表空间

系统视图 pg_catalog.pg_tablespace 中包含了所有表空间的信息：

```
1 | SELECT * FROM pg_catalog.pg_tablespace;
```

另外，psql 工具的 \db 命令也可以列出所有表空间。

修改表空间

如果需要修改表空间的定义，可以使用 `ALTER TABLESPACE` 语句：

```
1 | ALTER TABLESPACE tablespace_name RENAME TO new_name;
2 |
3 | ALTER TABLESPACE tablespace_name OWNER TO new_owner;
4 |
5 | ALTER TABLESPACE tablespace_name SET ( tablespace_option = value [, ... ] );
6 | ALTER TABLESPACE tablespace_name RESET ( tablespace_option [, ... ] );
```

第一个语句用于表空间的重命名；第二个语句用于修改表空间的拥有者；最后两个语句用于设置表空间的参数。PostgreSQL 支持设置的表空间参数包括 seq_page_cost、random_page_cost 以及 effective_io_concurrency，它们用于查询计划器选择执行计划时的代价评估。

删除表空间

对于不再需要的表空间，可以使用 DROP TABLESPACE 语句进行删除：

```
1 | DROP TABLESPACE [ IF EXISTS ] tablespace_name;
```

IF EXISTS 可以避免删除不存在的表空间时产生错误信息。删除表空间时，同时会删除文件系统中对应的表空间子目录。

备份与还原

 关于 PostgreSQL 备份与恢复的详细介绍可以参考这篇文章。

使用 pg_dump 执行逻辑备份

pg_dump 是 PostgreSQL 逻辑备份工具，用于导出创建数据库（CREATE DATABASE）和插入数据的文本文件或者其他格式文件。

使用 pg_dump 文本格式备份指定数据库的命令如下：

```
1 | pg_dump db_name > file_name.sql
```


其中，db_name 表示数据库名；file_name 表示备份文件名。

pg_dump 也可以选择导出指定的表：

```
1 | pg_dump -t 'table_name*' db_name > file_name.sql
```

以上命令表示导出数据库 db_name 中名字以 table_name 开头的表。

使用其他格式备份数据库的语法如下：

```
1 | pg_dump -Fc db_name -f file_name.dmp
2 |
3 | pg_dump -Fd db_name -f file_dir
4 |
5 | pg_dump -Ft db_name -f file_name.tar
```

使用 psql/pg_restore 执行还原

对于 sql 文件格式的备份，可以使用 psql 还原数据库：

```
1 | psql newdb -f file_name.sql
```

pg_dump 和 psql 支持的读写管道功能使得我们可以直接将数据库从一个服务器导出到另一个服务器，例如：

```
1 | pg_dump -h host1 db_name | psql -h host2 db_name
```

其他格式的备份需要pg_restore 工具进行还原：

```
1 | pg_restore -d newdb file_name.dmp
2 |
3 | pg_restore -d newdb file_dir
4 |
5 | pg_restore -d newdb file_name.tar
```

备份/还原整个数据库集群

PostgreSQL 提供了导出数据库集群的 pg_dumpall 工具。

```
1 | pg_dumpall -f cluster.sql
```

pg_dumpall 导出 sql 文件格式的备份，还原时直接使用 psql 导入相关文件即可。

```
1 | psql -f cluster.sql postgres
```

使用 COPY 导入/导出表数据

使用 COPY 命令可以导出单个表中的数据或查询结果集：

```
1 | COPY table_name
2 | FROM { 'filename' | PROGRAM 'command' | STDIN }
3 | [[ WITH ] ( option [, ...] ) ]
4 | [ WHERE condition ]
5 |
6 | COPY { table_name | ( query ) }
7 | TO { 'filename' | PROGRAM 'command' | STDOUT }
8 | [[ WITH ] ( option [, ...] ) ]
```

查询语句

单表查询

 PostgreSQL 简单查询可以参考[这篇文章](#)。

查询单个表中的字段：

```
1 | SELECT column1, column2, ...
2 | FROM table_name;
```

查询所有字段：

```
1 | SELECT * FROM table_name;
```

排除查询结果中的重复数据：

```
1 | SELECT DISTINCT column1, column2, ...
2 | FROM table_name;
```

查询条件

 关于 PostgreSQL 查询条件的详细介绍可以参考[这篇文章](#)。

使用 **WHERE** 指定查询条件：

```
1 | SELECT column1, column2, ...
2 | FROM table
3 | WHERE conditions;
```

常用的查询条件包括：=、!=、<>、<、<=、>、>=、BETWEEN、IN、EXISTS、LIKE、AND、OR、NOT、IS [NOT] NULL 等。

模糊匹配

使用 **LIKE** 运算符进行简单的字符串模式匹配：

```
1 | expr LIKE pattern [ESCAPE escape_character]
```

其中，pattern 用于指定一个匹配模式，百分号 (%) 匹配任意多个字符，下划线 (_) 匹配任意单个字符；escape_character 指定转义字符。例如：

```
1 | SELECT first_name,
2 |    last_name
3 | FROM employees
4 | WHERE last_name LIKE 'Kin%';
5 | first_name | last_name
6 | -----+-----
7 | Steven    | King
8 | Janette   | King
9 | (2 rows)
```

另外，**NOT LIKE** 运算符匹配与 LIKE 相反的结果。PostgreSQL 同时还提供了不区分大小写的 **[NOT] ILIKE** 运算符。

复杂条件

WHERE 子句可以包含多个条件，使用逻辑运算符（AND、OR、NOT）将它们进行组合，并根据最终的逻辑值进行过滤。对于 **AND** 运算符，只有当它两边的结果都为真时，最终结果才为真；否则最终结果为假，不返回结果。**OR** 逻辑运算符只要有一个条件为真，结果就为真。当我们组合 AND 和 OR 运算符时，AND 运算符优先级更高。

排序显示

 关于 PostgreSQL 的排序操作可以参考[这篇文章](#)。

指定排序字段的方式如下：

```
1 | SELECT column1, column2, ...
2 | FROM table_name
3 | ORDER BY column1 ASC, column2 DESC, ...;
```

PostgreSQL 支持使用 NULLS FIRST（空值排在最前）和 NULLS LAST（空值排在最后）指定空值的排序位置；升序排序时默认为 NULLS LAST，降序排序时默认为 NULLS FIRST。

限定数量

 关于 PostgreSQL Top-N 和分页查询的实现可以参考[这篇文章](#)。

PostgreSQL 支持 SQL 标准的 FETCH 和 OFFSET 子句，以及扩展的 LIMIT 语法限制返回结果的数量：

```
1 | SELECT column1, column2, ...
2 | FROM table_name
3 | [WHERE conditions]
4 | [ORDER BY column1 ASC, column2 DESC, ...]
5 | [OFFSET m {ROW | ROWS}]
6 | [FETCH { FIRST | NEXT } [ num_rows ] { ROW | ROWS } ONLY];
7 |
8 | SELECT column1, column2, ...
9 | FROM table_name
10 | [WHERE conditions]
11 | [ORDER BY column1 ASC, column2 DESC, ...]
12 | [LIMIT { num_rows | ALL } ]
13 | [OFFSET m {ROW | ROWS}];
```

分组操作

 关于 PostgreSQL 分组汇总的具体介绍可以参考[这篇文章](#)。

指定分组和过滤：

```
1 | SELECT column1, column2, agg_func()
2 | FROM table_name
3 | GROUP BY column1, column2
4 | HAVING conditions;
```

常用的聚合函数：AVG、COUNT、MIN、MAX、SUM、STRING_AGG 等。
PostgreSQL 除了支持基本的分组操作之外，还支持 3 种高级的分组选项：GROUPING SETS、ROLLUP 以及 CUBE。

多表连接

 关于 PostgreSQL 连接查询[这篇文章](#)。

连接查询用于从多个表中查询关联数据：

```
1 | SELECT t1.column1, t2.column2, ...
2 | FROM table1 AS t1
3 | [INNER | LEFT | RIGHT | FULL | CROSS] JOIN table2 AS t2
4 | ON conditions;
```

子查询

 关于 PostgreSQL 子查询的介绍可以参考[这篇文章](#)。

FROM 子句中的子查询被称为派生表：

```
1 | SELECT column1, column2, ...
2 | FROM (subquery) AS table_alias;
```

WHERE 条件中的子查询可以与 IN、ALL、ANY 等操作符一起使用。通过增加 LATERAL 关键字，横向子查询可以引用左侧表中的列：

```
1 | SELECT table1.column1, t2.col1, ...
2 | FROM table1
3 | JOIN LATERAL (
4 |     SELECT ...
5 |     FROM table2
6 |     WHERE table1.col1 = table1.column1) t2;
```

EXISTS 操作符与关联子查询：

```
1 | SELECT table1.column1, table1.column12, ...
2 | FROM table1
3 | WHERE EXISTS ( SELECT 1
4 |               FROM table2
5 |               WHERE table2.col1 = table1.col1);
```

集合运算

 关于 PostgreSQL 集合运算符的使用可以参考[这篇文章](#)。

集合运算包括并集、交集和差集：

```
1 | SELECT column1, column2
2 | FROM table1
3 | UNION [DISTINCT | ALL]
4 | SELECT col1, col2
5 | FROM table2;
6 |
7 | SELECT column1, column2
8 | FROM table1
9 | INTERSECT [DISTINCT | ALL]
10 | SELECT col1, col2
11 | FROM table2;
```



其中 ALL 表示保留结果集中的重复记录；DISTINCT（默认值）表示去除查询结果中的重复记录。相同的集合操作符按照从左至右的顺序执行，INTERSECT 的优先级高于 UNION 和 EXCEPT，使用括号可以修改集合操作的执行顺序。

通用表表达式

 关于 PostgreSQL 通用表表达式的详细内容可以参考[这篇文章](#)。

通用表表达式类似于派生表或者语句级别的视图，但是可读性和性能更好：

```
1 | WITH cte_name (col1, col2, ...) AS (
2 |     cte_query_definition
3 | )
4 | sql_statement;
```

递归 CTE 允许在它的定义中进行自引用，以下查询生成 1 到 10 的数字序列：

```
1 | WITH RECURSIVE cte(n) AS (
2 |     SELECT 1 AS n -- 初始查询
3 |     UNION ALL
4 |     cte(n+1) -- 递归查询
```

```
5 | SELECT n+1 FROM cte WHERE n < 10) -- 递归查询
   SELECT * FROM cte;
```

DML 语句

 关于 PostgreSQL 数据修改语句的使用可以参考[这篇文章](#)。

插入数据

插入数据使用 **INSERT** 语句：

```
1 | INSERT INTO table_name(column1, column2, ...)
2 | VALUES (value1, value2, ...);
```

一次插入多条记录：

```
1 | INSERT INTO table_name(column1, column2, ...)
2 | VALUES (val11,val12,...), (val21,val22,...), (val31,val32,...);
```

插入查询语句的结果：

```
1 | INSERT INTO table_name(column1, column2, ...)
2 | SELECT ...;
```

PostgreSQL 对 SQL 进行了扩展，可以在 INSERT 语句之后使用 RETURNING 返回插入的数据值：

```
1 | INSERT INTO table_name(column1, column2, ...)
2 | ...
3 | RETURNING ...;
```

更新数据

PostgreSQL 使用 **UPDATE** 语句更新表中已有的数据：

```
1 | UPDATE table_name
2 | SET column1 = value1,
3 |   column2 = value2,
4 |   ...
5 | WHERE conditions;
```

其中，WHERE 决定了需要更新的数据行，只有满足条件的数据才会更新；如果省略 WHERE 条件，将会更新表中的所有数据。

另外，PostgreSQL 还支持通过关联其他表中的数据进行更新：

```
1 | UPDATE table1
2 | SET column1 = table2.salary,
3 |   column2 = value2,
4 |   ...
5 | FROM table2
6 | WHERE conditions;
```

PostgreSQL 同样对 UPDATE 语句进行了扩展，支持使用 RETURNING 返回更新后的数据值：

```
1 | UPDATE table_name
2 | SET ...
3 | WHERE conditions
4 | RETURNING ...;
```

删除数据

删除数据可以使用 **DELETE** 语句：

```
1 | DELETE FROM table_name
2 | WHERE conditions;
```

同样，只有满足WHERE条件的数据才会被删除；如果省略，将会删除表中所有的数据。

PostgreSQL 同样支持通过关联其他表进行数据删除：

```
1 | DELETE
2 | FROM table1
3 | USING table2
4 | WHERE conditions;
```

DELETE 语句也可以使用 RETURNING 返回被删除的数据：

```
1 | DELETE FROM table_name
2 | WHERE conditions
3 | RETURNING ...;
```

另外， **TRUNCATE TABLE** 语句用于快速清除表中的全部数据：

```
1 | TRUNCATE TABLE table_name;
```

合并数据

PostgreSQL 可以通过 **INSERT INTO ON CONFLICT** 语句实现数据合并（MERGE）的功能：

```
1 | INSERT INTO table_name(column1, column2, ...)
2 | ...
3 | ON CONFLICT [conflict_target]
4 | {DO NOTHING | DO UPDATE SET ... [WHERE contidions]};
```

其中，conflict_target 是判断数据是否存在冲突的条件：

- ({ index_column_name | (index_expression) })，基于某个具有索引的字段或者表达式进行判断；
- ON CONSTRAINT constraint_name，基于某个唯一约束进行判断。

事务控制

 关于 PostgreSQL 数据库事务和事务控制语句可以参考[这篇文章](#)。

开始事务

默认情况下，PostgreSQL 执行任何语句都会自动开始一个事务并提交该事务。如果有需要，可以使用 **BEGIN** 语句手动开始一个事务：

```
1 | BEGIN;
```

另外，也可以使用 **BEGIN WORK** 或者 **BEGIN TRANSACTION** 开始事务。

提交事务

PostgreSQL 使用 **COMMIT** 语句提交已经打开的事务：

```
1 | COMMIT;
```

另外，也可以使用 **COMMIT WORK** 或者 **COMMIT TRANSACTION** 提交事务。

回滚事务

ROLLBACK 用于回滚当前事务：

```
1 | ROLLBACK;
```

另外，也可以使用 `ROLLBACK WORK` 或者 `ROLLBACK TRANSACTION` 回滚事务。

事务保存点

事务保存点可以用于回滚部分事务，`SAVEPOINT` 语句用于在事务中定义保存点：

```
1 | BEGIN;
2 | ...
3 | SAVEPOINT savepoint_name;
4 | ...
5 | ROLLBACK TO SAVEPOINT savepoint_name;
6 | ...
7 | COMMIT;
```

其中，`ROLLBACK TO` 用于回退到保存点时的状态。另外，`RELEASE SAVEPOINT savepoint_name` 可以释放保存点。

隔离级别

使用 `SHOW` 命令可以查看当前的隔离级别：

```
1 | SHOW transaction_isolation;
```

如果需要修改当前事务的隔离级别，可以在事务的最开始执行 `SET TRANSACTION` 命令：

```
1 | BEGIN;
2 | SET TRANSACTION ISOLATION LEVEL { SERIALIZABLE | REPEATABLE READ | READ (
3 | ...
```

视图

 关于 PostgreSQL 视图的概念和使用，可以参考[这篇文章](#)。

创建视图

PostgreSQL 使用 `CREATE VIEW` 语句创建视图：

```
1 | CREATE [OR REPLACE] VIEW view_name
2 | AS
3 | select-statement
4 | WITH CHECK OPTION;
```

`WITH CHECK OPTION` 选项可以阻止通过视图修改或者插入视图范围之外的基础表数据。

查看所有视图

PostgreSQL 系统表 `information_schema.views` 中存储了关于视图的所有信息：

```
1 | SELECT *
2 | FROM information_schema.views;
```

`psql` 命令 `\dv` 也可以列出当前数据库中的所有视图。

查看视图定义

查看视图的定义：

```
1 | SELECT view_definition
2 | FROM information_schema.views
3 | WHERE table_schema = 'public'
4 | AND table_name = 'employees_it';
```

```
5 |  
6 | SELECT employees.employee_id,  
7 |     employees.first_name,  
8 |     employees.last_name,  
9 |     employees.email,  
10 |    employees.phone_number,  
11 |    employees.hire_date,  
12 |    employees.job_id,  
13 |    employees.manager_id,  
14 |    employees.department_id  
15 | FROM employees  
16 | WHERE (employees.department_id = 60);
```

修改视图

PostgreSQL 使用 **ALTER VIEW** 语句修改视图的信息。例如重命名视图：

```
1 | ALTER VIEW IF EXISTS view_name RENAME TO new_name
```

删除视图

PostgreSQL 使用 **DROP VIEW** 语句删除视图：

```
1 | DROP VIEW [IF EXISTS] view_name;
```

存储过程/函数

 关于 PostgreSQL 存储过程/函数的定义和使用，可以参考[这篇文章](#)。

创建存储过程/函数

使用 **CREATE PROCEDURE** 语句创建PL/pgSQL 存储过程：

```
1 | CREATE [ OR REPLACE ] PROCEDURE  
2 |   procedure_name ( [ [ argmode ] [ argname ] argtype [ { DEFAULT | = } default_exp  
3 | AS $$  
4 | DECLARE  
5 |   declarations  
6 | BEGIN  
7 |   statements;  
8 |   ...  
9 | END; $$  
10 | LANGUAGE plpgsql;  
11 |
```

使用 **CREATE FUNCTION** 语句创建PL/pgSQL 函数：

```
1 | CREATE [ OR REPLACE ] FUNCTION  
2 |   function_name ( [ [ argmode ] [ argname ] argtype [ { DEFAULT | = } default_expr ]  
3 | RETURNS rettype  
4 | AS $$  
5 | DECLARE  
6 |   declarations  
7 | BEGIN  
8 |   statements;  
9 |   ...  
10 | END; $$  
11 | LANGUAGE plpgsql;
```

调用存储过程/函数

调用存储过程使用 **CALL** 语句：

```
1 | CALL procedure_name( argument1, ... );
```


存储函数可以像可以像内置函数一样在 SQL 语句中进行调用：

```
1 | SELECT function_name( argument1, ... );
```

重命名存储过程/函数

使用 **ALTER PROCEDURE** 和 **ALTER FUNCTION** 语句修改存储过程/函数的属性，例如修改名称：

```
1 | ALTER PROCEDURE procedure_name [ ( [ argmode ] [ argname ] argtype [, ...] ) ]  
2 |     RENAME TO new_name;  
3 |  
4 | ALTER FUNCTION function_name [ ( [ argmode ] [ argname ] argtype [, ...] ) ]  
5 |     RENAME TO new_name;
```

删除存储过程/函数

删除存储过程/函数使用 **DROP** 语句：

```
1 | DROP PROCEDURE IF EXISTS procedure_name [ CASCADE | RESTRICT ];  
2 |  
3 | DROP FUNCTION IF EXISTS function_name [ CASCADE | RESTRICT ];
```

触发器

 关于 PostgreSQL 触发器的详细介绍，可以参考[这篇文章](#)。

创建触发器

PostgreSQL 触发器的创建分为两步：首先，使用 **CREATE FUNCTION** 语句创建一个触发器函数；然后，使用 **CREATE TRIGGER** 语句将该函数与表进行关联。

```
1 | CREATE [ OR REPLACE ] FUNCTION trigger_function ()  
2 |     RETURNS trigger  
3 | AS $$  
4 | DECLARE  
5 |     declarations  
6 | BEGIN  
7 |     statements;  
8 |     ...  
9 | END; $$  
10 | LANGUAGE plpgsql;  
11 |  
12 | CREATE TRIGGER trigger_name  
13 | { BEFORE | AFTER | INSTEAD OF } { event [ OR ... ] }  
14 | ON table_name  
15 | [ FOR [ EACH ] { ROW | STATEMENT } ]  
16 | [ WHEN ( condition ) ]  
17 | EXECUTE FUNCTION trigger_function;
```

其中，event 可以是 INSERT、UPDATE、DELETE 或者 TRUNCATE，UPDATE 支持特定字段（UPDATE OF col1, col2）的更新操作；触发器可以在事件之前（BEFORE）或者之后（AFTER）触发，INSTEAD OF 只能用于替代视图上的 INSERT、UPDATE 或者 DELETE 操作；FOR EACH ROW 表示行级触发器，FOR EACH STATEMENT 表示语句级触发器；WHEN 用于指定一个额外的触发条件，满足条件才会真正支持触发器函数。

查看触发器

视图 information_schema.triggers 中存储了关于触发器的信息：

```
1 | SELECT *  
2 | FROM information_schema.triggers;
```

修改触发器

PostgreSQL 使用 **ALTER TRIGGER** 语句修改触发器，目前只支持修改触发器的名称：

```
1 | ALTER TRIGGER trigger_name ON table_name RENAME TO new_name;
```

启用/禁用触发器

PostgreSQL 支持触发器的禁用和启用：


```
1 | ALTER TABLE table_name
2 | {ENABLE | DISABLE} TRIGGER {trigger_name | ALL | USER};
```

删除触发器

删除触发器的语句如下：


```
1 | DROP TRIGGER [IF EXISTS] trigger_name
2 | ON table_name [RESTRICT | CASCADE];
```

显示推荐内容





不剪发的Tony老师


关注

 54

 13

 397





专栏目录