PostgreSQL模式(schema)

[November 5, 2013](http://openwares.net/database/postgresql_schema.html)[Database](http://openwares.net/category/database)[PostgresQL](http://openwares.net/tag/postgresql)[openwares](http://openwares.net/author/openwares)

schema就是用于逻辑隔离不同数据库对象的名字空间，schema隶属于特定的数据库。

一个数据库可以包含一个或多个命名schema,然后schema可以包含表等其他对象。

schema也可以包含多种命名对象，包括数据类型(data types),函数(functions)和操作符(operators),相同的对象名可以存在与不同的schema里而不会冲突，不像数据库，schema是逻辑隔离，一个用户可以存取其所连接到数据库的任何schema里的对象，只要用户有相应的权限。

使用schema可能的原因有以下几个：

* 允许多个用户使用同一个数据库而互不干扰
* 组织数据库对象到逻辑组，使其更容易管理
* 第三方应用程序可以放进单独的schema，隔离其名字空间，不与其他对象名字冲突

schema是操作系统目录的模拟，但是schema是不能嵌套的。schema就是namespace。

**创建schema**

*语法*

|  |  |
| --- | --- |
| 1  2  3  4 | CREATE SCHEMA schema\_name [ AUTHORIZATION user\_name ] [ schema\_element [ ... ] ]  CREATE SCHEMA AUTHORIZATION user\_name [ schema\_element [ ... ] ]  CREATE SCHEMA IF NOT EXISTS schema\_name [ AUTHORIZATION user\_name ]  CREATE SCHEMA IF NOT EXISTS AUTHORIZATION user\_name |

CREATE SCHEMA语句在当前数据库中创建名字为schema\_name的schema。

*参数*

* schema\_name

将要创建的schema的名字，在当前数据库中schema的名字不能冲突。如果忽略此参数，则以当前数据库用户的名字命名新创建的schema。schema的名字不能以pg\_开头，这是系统保留的名字。

* user\_name

拥有新创建schema的角色名字，如果不指定则为执行当前命令的数据库用户。如果为其他角色创建此schema，那么当前用户必须是那个角色的直接或间接角色成员，或者当前用户为超级用户。

* schema\_element

创建schema名字空间下其他对象的SQL语句。与创建schema完毕后执行单独的SQL来创建对象是一样的，除了如果指定AUTHORIZATION,那么新创建的对象都有指定的角色拥有。建议分开创建schema包含的对象。

* IF NOT EXISTS

如果同名的schema已经存在，除了提示什么也不做。当指定此选项时，不能包含schema\_element子命令。

*注意*  
要创建schema，调用该命令的用户必需在当前数据库上有 CREATE 权限。当然，超级用户可以绕开这个检查。

在新建的schema下建表

|  |  |
| --- | --- |
| 1  2  3 | CREATE TABLE myschema.mytable (   ...  ); |

**查询schema**  
SQL语句

|  |  |
| --- | --- |
| 1 | SELECT \* FROM pg\_namespace; |

pg\_namespace不只包含schema，还包含其他的名字空间。

或者psql命令

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8 | =# \dn     List of schemas      Name    |  Owner  ------------+----------   public     | postgres   testschema | postgres  (2 rows) |

**删除schema**

如果schema的子对象都已删除，schema已为空对象，可以用以下语句删除schema

|  |  |
| --- | --- |
| 1 | DROP SCHEMA myschema; |

也可以这样删除schema及其子对象

|  |  |
| --- | --- |
| 1 | DROP SCHEMA myschema CASCADE; |

**public schema**

每一个新建的数据库包含一个默认的public模式，数据库中所有没有显式或隐式指定schema的对象，都归属于public名字空间。

|  |  |
| --- | --- |
| 1 | CREATE TABLE products ( ... ); |

与

|  |  |
| --- | --- |
| 1 | CREATE TABLE public.products ( ... ); |

是一样的。

**schema搜索路径**

全限定的名字写起来冗长乏味，因此表经常通过一个没有名字空间限定的名字来使用，仅仅就是表的名字。系统通过一个搜索路径来决定到底使用的是哪一张表，搜索路径是schema的一个列表。搜索路径中第一个匹配的表即是要访问的表。如果搜索路径中没有匹配，会报告一个错误，即使在数据库的其他schema中有相匹配的表。

搜索路径中的第一个schema成为当前schema,也是CREATE命名没有指定schema时的默认schema。

查看当前的搜索路径：

|  |  |
| --- | --- |
| 1  2  3  4  5  6 | => SHOW search\_path;     search\_path  ----------------   "$user",public  (1 row) |

默认情况下，搜索路径的第一个schema是与当前用户同名的schema,第二个则是public schema。

可以这样设置搜索路径：

|  |  |
| --- | --- |
| 1 | SET search\_path TO myschema,public; |

搜索路径同样适用于data types,function names和operator names。

**schema与权限**

默认情况下，用户不能访问不属于他的schema中的任何对象。要允许访问，schema的拥有者必须授予用户在这个schema上的USAGE权限。不同访问权限需要不同的授权。

一个用户也可以在其他schema里创建对象，这需要schema拥有者授予用户CREATE权限。默认情况下，每个人都有public schema上的CREATE和USAGE权限。这允许所有可以连接到指定数据库的用户，在public模式里创建对象。如果不允许这样，可以撤销默认的权限设置：

|  |  |
| --- | --- |
| 1 | REVOKE CREATE ON SCHEMA public FROM PUBLIC; |

第一个小写的public是指 public schema,而第二个大写的PUBLIC是指所有可以连接到数据库的用户。第一个小写的public是个标示符，而第二个大写的PUBLIC则是关键字，所以用大小写予以区分。

**System Catalog Schema**

除了public和用户创建的schema，每一个数据库还包含一个pg\_catalog schema,它包含了系统表所有的内建数据类型，函数和操作符。pg\_catalog一直是schema搜索路径的一部分。虽然没有在搜索路径中显示指定，但它在搜索路径中的schema之前被隐式搜索。这保证了内建的名字总是可以被发现。只要不使用pg\_开头的名字就不会与系统发生名字冲突。

**schema使用范式**

schema用来组织和管理数据库对象，有几个推荐的schema使用范式：

* 如果不创建任何schema，则所有用户隐式的使用public schema。这等同于根本不使用schema。这在数据库中只有一个或者极少的用户时推荐使用。
* 可以为每一个用户创建一个与其用户名相同的schema。因为缺省的搜索路径的第一个schema是$user,与当前用户的名字相同。因此，如果每个用户有一个与其名字相同的单独的schema，则默认他们只能访问自己所属的schema。使用这种schema范式，可以撤销掉对public schema的访问许可，甚至把public schema直接移除，这样每个用户就真正的限定在了他们自己的schema里

参考：  
[Schemas](http://www.postgresql.org/docs/9.3/static/ddl-schemas.html)  
[CREATE SCHEMA](http://www.postgresql.org/docs/9.3/static/sql-createschema.html)

**===  
管得最少，就是最好的政府。 —— 大卫·梭罗**