阿里云
交互式分析

用户指南

文档版本：20191119

法律声明

阿里云提醒您在阅读或使用本文档之前仔细阅读、充分理解本法律声明各条款的内容。如果您阅读或使用本文档，您的阅读或使用行为将被视为对本声明全部内容的认可。

1. **您应当通过阿里云网站或阿里云提供的其他授权通道下载、获取本文档，且仅能用于自身的合法合规的业务活动。本文档的内容视为阿里云的保密信息，您应当严格遵守保密义务；未经阿里云事先书面同意，您不得向任何第三方披露本手册内容或提供给任何第三方使用。**
2. **未经阿里云事先书面许可，任何单位、公司或个人不得擅自摘抄、翻译、复制本文档内容的部分或全部，不得以任何方式或途径进行传播和宣传。**
3. **由于产品版本升级、调整或其他原因，本文档内容有可能变更。阿里云保留在没有任何通知或者提示下对本文档的内容进行修改的权利，并在阿里云授权通道中不时发布更新后的用户文档。您应当实时关注用户文档的版本变更并通过阿里云授权渠道下载、获取最新版的用户文档。**
4. **本文档仅作为用户使用阿里云产品及服务的参考性指引，阿里云以产品及服务的“现状”、“有缺陷”和“当前功能”的状态提供本文档。阿里云在现有技术的基础上尽最大努力提供相应的介绍及操作指引，但阿里云在此明确声明对本文档内容的准确性、完整性、适用性、可靠性等不作任何明示或暗示的保证。任何单位、公司或个人因为下载、使用或信赖本文档而发生任何差错或经济损失的，阿里云不承担任何法律责任。在任何情况下，阿里云均不对任何间接性、后果性、惩戒性、偶然性、特殊性或刑罚性的损害，包括用户使用或信赖本文档而遭受的利润损失，承担责任（即使阿里云已被告知该等损失的可能性）。**
5. **阿里云文档中所有内容，包括但不限于图片、架构设计、页面布局、文字描述，均由阿里云和/或其关联公司依法拥有其知识产权，包括但不限于商标权、专利权、著作权、商业秘密等。非经阿里云和/或其关联公司书面同意，任何人不得擅自使用、修改、复制、公开传播、改变、散布、发行或公开发表阿里云网站、产品程序或内容。此外，未经阿里云事先书面同意，任何人不得为了任何营销、广告、促销或其他目的使用、公布或复制阿里云的名称（包括但不限于单独为或以组合形式包含“阿里云”、“Aliyun”、“万网”等阿里云和/或其关联公司品牌，上述品牌的附属标志及图案或任何类似公司名称、商号、商标、产品或服务名称、域名、图案标示、标志、标识或通过特定描述使第三方能够识别阿里云和/或其关联公司）。**
6. **如若发现本文档存在任何错误，请与阿里云取得直接联系。**

通用约定

格式	说明	样例
	该类警示信息将导致系统重大变更甚至故障，或者导致人身伤害等结果。	 **禁止:** 重置操作将丢失用户配置数据。
	该类警示信息可能会导致系统重大变更甚至故障，或者导致人身伤害等结果。	 **警告:** 重启操作将导致业务中断，恢复业务时间约十分钟。
	用于警示信息、补充说明等，是用户必须了解的内容。	 **注意:** 权重设置为0，该服务器不会再接受新请求。
	用于补充说明、最佳实践、窍门等，不是用户必须了解的内容。	 **说明:** 您也可以通过按Ctrl + A选中全部文件。
>	多级菜单递进。	单击设置 > 网络 > 设置网络类型。
粗体	表示按键、菜单、页面名称等UI元素。	在结果确认页面，单击**确定**。
`Courier字体`	命令。	执行`cd /d C:/window`命令，进入Windows系统文件夹。
*## *	表示参数、变量。	`bae log list --instanceid` *`Instance_ID`*
[]或者[a|b]	表示可选项，至多选择一个。	`ipconfig [-all|-t]`
{}或者{a|b}	表示必选项，至多选择一个。	`switch {active|stand}`

目录

1 访问域名

交互式分析（Interactive Analytics）提供了单独的域名访问地址（Endpoint），通过该地址您可以访问阿里云不同区域的交互式分析服务。

交互式分析（Interactive Analytics）在公共云的不同Region及网络环境下的服务连接对照表如下。

表 1-1: 公网网络下Region和服务连接对照表

Region	所在城市	开服状态	Endpoint
华东1	杭州	公测中	demo-hangzhou.hologres.aliyuncs.com
华北2	北京	公测中	demo-beijing.hologres.aliyuncs.com
华东2	上海	公测中	demo-shanghai.hologres.aliyuncs.com
其他区域	无	暂未开服	无

表 1-2: 经典网络下Region和服务连接对照表

Region	所在城市	开服状态	经典网络Endpoint
华东1	杭州	公测中	demo-hangzhou-internal.hologres.aliyuncs.com
华北2	北京	公测中	demo-beijing-internal.hologres.aliyuncs.com
华东2	上海	公测中	demo-shanghai-internal.hologres.aliyuncs.com
其他区域	无	暂未开服	无

说明:

：绑定HoloStudio只能用经典网络。

表 1-3: VPC网络下Region和服务连接对照表

Region	所在城市	开服状态	VPC网络Endpoint
华东1	杭州	公测中	demo-hangzhou-vpc.hologres.aliyuncs.com
华北2	北京	公测中	demo-beijing-vpc.hologres.aliyuncs.com
华东2	上海	公测中	demo-shanghai-vpc.hologres.aliyuncs.com
其他区域	无	暂未开服	无

表中`demo`**均为**`instancename`**，即自己的实例名称。**

说明:

：例如用postgres这个实例连接VPC网络，其Endpoint为：`postgres-hangzhou-vpc.hologres.aliyuncs.com`。

2 实例变更配置

本小节将会为您介绍如何实例扩/缩容以及延长到期时间。

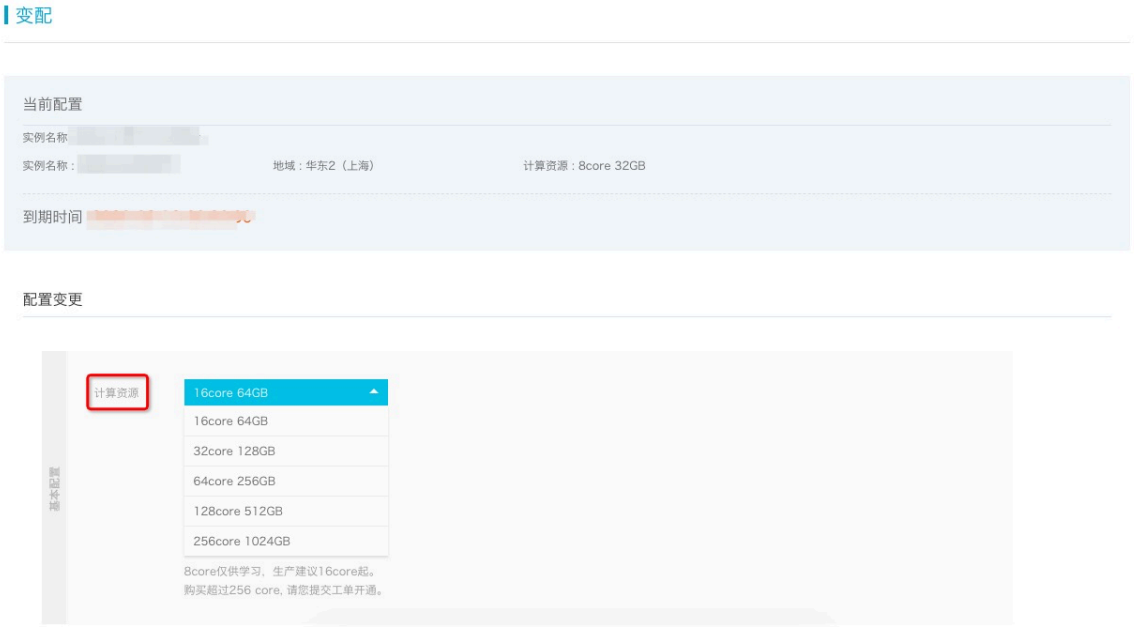
> **说明:**
> **当前只允许主账号对实例进行扩容、延长时间等操作。**

实例扩/缩容

当前计算资源不足/过量时，可以根据项目情况按需对实例进行扩/缩容，操作步骤如下：

1. **阿里云账号登录管控台，选择左上角实例所在的region，单击左侧菜单栏的交互式分析，找到需要扩/缩容的实例，单击右侧更变配置。**

2. **在新跳出的页面选择新的计算资源规模，并单击支付即可。**



延长实例时间

当提示您当前实例即将过期时，可根据项目情况延长实例到期时间，操作步骤如下：

说明:

实例到期后，会显示停止服务，需要及时续费才能继续使用，若您没有及时续费，该实例会被删除。

1. **阿里云账号登录管控台，选择左上角实例所在的region，单击左侧菜单栏的交互式分析，找到延长时间的实例，单击右侧续费。**

2. **在新跳出的页面选择新的续费时长，并单击支付即可。**

3 用户授权及角色管理

3.1 角色概述

交互式分析（Interactive Analytics）是和Postgres完全兼容的大数据引擎，所以其用户和权限体系都和Postgres完全兼容。交互式分析（Interactive Analytics）的账号走阿里云账号认证体系，其他逻辑全部同Postgres。

交互式分析（Interactive Analytics）使用角色（Role）的概念管理数据库访问权限。一个角色可以被看成是一个数据库用户或者是一个数据库用户组，这取决于角色被怎样设置。角色可以拥有数据库对象（例如，表和函数）并且能够把那些对象上的权限赋予给其他角色来控制谁能访问哪些对象。

此外，还可以把一个角色中的成员资格授予给另一个角色，这样允许成员角色使用被赋予给另一个角色的权限。

以下为交互式分析（Interactive Analytics）中常用的角色名称定义。

- Cluster Admin：管理整个集群。可创建、销毁instance。阿里云内部管理人员，权限不对外。
- superuser：某个项目内的实例（instance）内的管理员，系统默认将实例申请账号的拥有者设定为superuser。拥有整个实例的权限，可创建、销毁DB，也可创建角色以及为角色授权等。
- DB owner：某个DB的owner。系统默认superuser是DB的Owner，但superuser可授权给某个用户，让其成为DB Owner。
- 普通用户：经过授权后，可在某个DB里执行普通的SQL。需要更多的权限需要向superuser申请。

更多关于角色的介绍可以直接移步*Postgres* **数据库角色**。

3.2 角色管理

本小节将会介绍交互式分析（Interactive Analytics）角色管理的操作命令。

创建用户

要在交互式分析（Interactive Analytics）中创建一个用户，可使用以下两个SQL命令之一。

```
CREATE ROLE name;--不可登录
```

```
CREATE USER name;
```

name遵循SQL标识符的规则：不包含特殊字符，或包含特殊字符但用双引号包围，例如“ALIYUN$xx”。

supperuser可以为用户创建角色，并且在创建角色的同时可以赋予一定的角色权限，例如LOGIN。

相关角色权限语法如下。

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

where option can be:

      SUPERUSER | NOSUPERUSER
    | CREATEDB | NOCREATEDB
    | CREATEROLE | NOCREATEROLE
    | INHERIT | NOINHERIT
    | LOGIN | NOLOGIN
    | REPLICATION | NOREPLICATION
    | BYPASSRLS | NOBYPASSRLS
    | CONNECTION LIMIT connlimit
    | IN ROLE role_name [, ...]
    | IN GROUP role_name [, ...]
    | ROLE role_name [, ...]
    | ADMIN role_name [, ...]
    | USER role_name [, ...]
    | SYSID uid
```

示例如下

```
-- 创建一个可登录的角色，但不设置密码
CREATE USER "RAM$root@example.com:subuser1";

-- 创建一个可以管理数据库、管理其他角色的不可登录角色
CREATE ROLE admin WITH CREATEDB CREATEROLE;
```

说明:

:

- **当前版本暂不支持为角色自定义密码，只支持角色的阿里云账号的AccessKey作为密码。**
- **CREATE USER和CREATE ROLE的区别在于CREATE USR含有LOGIN功能。详细的账号规范可参加阿里云账号认证。**

更多关于创建角色的细节，可参见*PostgreSQL*。

删除角色

想要删除一个角色，可以使用DROP命令，示例如下。

```
DROP ROLE name;
```

说明:

：执行DROP命令后，任何在该组角色中的成员关系会被自动撤销（但是成员角色不会受到影响）。

授权角色

在交互式分析（Interactive Analytics）中，对某个角色授权可以参见以下表格。

权限	语法	必须
创建可登录子账号	`create user "xx@aliyun.com";`或`create user "ALIYUN$xx";`	是
赋予所有子账号对于交互式分析模式下的所有表访问权限	`GRANT USAGE ON SCHEMA hologres TO PUBLIC;`	默认已执行
赋予子账号交互式分析属性表的查增改权限，确保能够访问表属性和建表	`GRANT SELECT, INSERT,UPDATE ON table hologres.holo_table_properties TO PUBLIC;`	默认已执行
赋予所有子账号对模式public下的所有普通表进行查增改的权限	`GRANT SELECT, INSERT,UPDATE ON ALL TABLES IN SCHEMA public to PUBLIC;`	可选
授予子账号superuser的权限	`create user "p4_账号id" SUPERUSER ;`	可选

给大批量用户统一授予权限	`GRANT SELECT, INSERT,UPDATE ON ALL TABLES IN SCHEMA public TO public; //` **用户表的增删改查**	**可选**	

更多关于权限的授予请参见：*GRANT*。

撤销授权

若想撤销子账号的某一个权限，使用REVOKE语句，示例如下：

```
REVOKE superuser from "p4_账号id" ;//撤销子账号superuser的权限
```

更多关于权限的撤销操作请参见：*REVOKE*。

权限查看

您可使用以下命令查看当前账号的权限。

```
select rolname from pg_roles;
select user_display_name(rolname) from pg_roles;
```

若您是使用psql客户端连接交互式分析（Interactive Analytics），您可使用`\du`查看当前账号的所有子账号以及对应的权限。

```
postgres=# select rolname from pg_roles;
          rolname
----------------------------
 [redacted]
 pg_monitor
 pg_read_all_settings
 pg_read_all_stats
 pg_stat_scan_tables
 pg_read_server_files
 pg_write_server_files
 pg_execute_server_program
 pg_signal_backend
 [redacted]
(11 rows)

postgres=# select user_display_name(rolname) from pg_roles;
      user_display_name
------------------------------
[redacted]
 pg_monitor
 pg_read_all_settings
 pg_read_all_stats
 pg_stat_scan_tables
 pg_read_server_files
 pg_write_server_files
 pg_execute_server_program
 pg_signal_backend
 [redacted]t
(11 rows)

postgres=# \du
                                   List of roles
       Role name       |                         Attributes                         | Member of
-----------------------+------------------------------------------------------------+-----------
 [redacted]            | Superuser                                                  | {}
 [redacted]            | Superuser, Create role, Create DB, Replication, Bypass RLS | {}
 [redacted]            |                                                            | {}
```

3.3 角色属性

一个数据库角色必须具有角色属性才能在交互式分析（Interactive Analytics）里进行操作。本小节将会介绍交互式分析（Interactive Analytics）的角色属性。

- **login privilege**

  只有具有LOGIN属性的角色才能被用于一个数据库连接的初始角色名称。一个带有LOGIN属性的角色可以被认为和一个“数据库用户”相同。要创建一个带有登录权限的角色，可使用下述命令中的任何一条：

  ```
  CREATE ROLE name LOGIN;
  CREATE USER name;
  ```

  说明：

  ：CREATE USER和CREATE ROLE等效，但CREATE USER默认有LOGIN权限，更常用于交互式分析（Interactive Analytics）。

- **superuser status**

  一个数据库超级用户会绕开所有权限检查，除了登入的权利。这是一个危险的权限并且应该小心使用，最好用一个不是超级用户的角色来完成您的大部分工作。要创建一个新数据库超级用户，使用以下命令：

  ```
  CREATE ROLE name SUPERUSER;
  ```

  说明:

  : 上述操作需要超级用户角色才能完成。

- **database creation**

  一个角色必须被显式给予权限才能创建数据库（除了超级用户，因为它们会绕开所有权限检查）。要创建这样一个角色，使用以下命令：

  ```
  CREATE ROLE name CREATEDB;
  ```

- **role creation**

  一个角色必须被显式给予权限才能创建更多角色（除了超级用户，因为它们会绕开所有权限检查）。要创建这样一个角色，使用`CREATE ROLE name CREATEROLE;`。一个带有`CREATEROLE`权限的角色也可以修改和删除其他角色，还可以授予或回收角色中的成员关系。然而，要创建、修改、删除或修改一个超级用户角色的成员关系，需要以超级用户的身份操作。`CREATEROLE`不足以完成这一切。

  要移除一个已有的角色，使用相似的命令：

  ```
  DROP ROLE name;
  ```

  要决定现有角色的集合，检查`pg_roles`系统目录，例如：

  ```
  SELECT rolname FROM pg_roles;
  ```

  psql程序的`\du`元命令也可以用来列出现有角色。

  一旦组角色存在，superuser可以使用*GRANT*和*REVOKE*增加或者删除成员

  ```
  GRANT group_role TO role1;
  REVOKE group_role FROM role1;
  ```

- 关于角色的指令具体参数可参见PostgreSQL的相关命令。

3.4 阿里云账号认证

交互式分析（Interactive Analytics）和阿里云账号体系深度集成，其数据库角色可以是一个阿里云账号（例如abc@aliyun.com）或者子账号（例如RAM$abc@aliyun.com:subuser1），也可以是一个自己命名的用户组（例如，developer）。本小节将会为您介绍在交互式分析（Interactive Analytics）中阿里云账号体系的一些常见概念。

- 用户显示名称

  例如abc@aliyun.com或者RAM$abcaliyun.com:subuser1。

  > 说明:
  >
  > :
  >
  > 1. 如果是阿里云主账号，则其格式为"abc@aliyun.com"或者"ALIYUN$abc@aliyun.com"（两者等价）。在某些情况下，阿里云账号系统允许创建非email格式的主账号，例如user1，但是必须加上ALIYUN$前缀标识，即ALIYUN$user1。
  > 2. 如果是阿里云子账户，子账户的显示名称格式为以下两种之一：
  >
  >   - RAM$主账号显示名称：子账号名称。这里主账号显示名称的格式为前面所述，但不带ALIYUN$前缀。例如，RAM$abc@aliyun.com:subuser1就是一个子账号显示名称，其中主账号是abc@aliyun.com，子账号是subuser1。
  >   - RAM$主账号Account ID：子账号名称。例如RAM$1898137152164646:subuser1就是一个子账号显示名称，其中主账号的Account ID是1898137152164646，子账号是subuser1。

  在CREATE/ALTER/DROP ROLE/USER或者GRANT/REVOKE等操作要用到阿里云账号时，必须以用户的显示名称来指定。

- 用户Account ID

  格式为一串数字，例如1898137152164646。这个是每个用户的唯一标识，一旦确定就不可修改。

  交互式分析（Interactive Analytics）使用阿里云用户的Account ID作为角色名称。但在对各种角色的操作时，命令中可使用用户显示名称来表示角色。交互式分析（Interactive Analytics）会自动将显示名称转换成用户Account ID。

  例如，为一个阿里云子账号创建同名的交互式分析（Interactive Analytics）角色，需要执行以下命令：

```
CREATE USER "RAM$abc@aliyun.com:subuser1";
```

   说明:

  :

  1. 因为阿里云账号名字包含$@等符号，必须用双引号包围。
  2. 这个阿里云账号必须是阿里云账号体系中已存在的账号。对于一个用户来说，要访问交互式分析（Interactive Analytics），必须在阿里云账号体系中为其创建一个阿里云账号，然后在交互式分析（Interactive Analytics）中为其CREATE USER。
  3. 交互式分析（Interactive Analytics）依赖于阿里云账号体系进行用户认证，所以在交互式分析（Interactive Analytics）中不需要为角色设置密码，认证由阿里云账号体系完成。
  4. 如果在交互式分析（Interactive Analytics）中创建一个可登录（LOGIN）的角色，但角色名并不是一个合法的阿里云用户名，那么这个角色是无法登录的（因为该用户无法通过阿里云认证）。
  5. 其他不可登录的交互式分析（Interactive Analytics）角色并不需要是阿里云账号。

- **Access ID/Access Key**

  这是阿里云颁发的访问凭证，会定期变化。

  对于一个用户来说，访问交互式分析（Interactive Analytics）时，必须要先获得其阿里云账号的：Access ID和Access Key

  在获得Access ID和Access Key后，将Access ID作为用户名，将Access Key作为密码来连接交互式分析（Interactive Analytics）。例如，使用psql访问时：

```
PGPASSWORD=<access key> psql -U <access id> ...
```

  在成功连接交互式分析后，交互式分析会自动从Access ID/Access Key中识别出用户及其Account ID。输入SQL命令：

```
SELECT current_user;
```

  返回结果为一串数字，例如`1898137152164646`，即为该用户的ID。

  要获得此用户的阿里云显示名称，需要使用函数`user_display_name`。执行SQL命令如下：

```
SELECT user_display_name(current_user);
```

  返回结果为当前用户名称，例如：`RAM$abc@aliyun.com:subuser1`。

  获得当前交互式分析（Interactive Analytics）中的所有角色，执行SQL命令如下：

```
SELECT user_display_name(rolname) FROM pg_roles;
```

4 SQL参考

4.1 数据类型

交互式分析（Interactive Analytics） 数据类型与 PostgreSQL 数据类型兼容，但支持的数据类型是 PostgreSQL 的一个子集。

数据类型

交互式分析当前版本支持的数据类型如下表：

数据类型	别名	是否支持	存储大小	范围	说明
integer	int, int4	支持	4 字节	-2147483648 到 +2147483647	常用的整数
BIGINT	int8	支持	8 字节	-9223372036854775808 到 +9223372036854775807	大范围整数
boolean	bool	支持	1 字节	True / False	布尔类型
float	float8	支持	8 字节	15 位十进制数字精度	可变精度，不精确
double precision	无	支持	8 字节	15 位十进制数字精度	可变精度，不精确
text	VarChar	支持	可变长	无	可变长度字符串
timestamp with time zone	timestamptz	支持	8 字节	4713 BC 到 294276 AD	时间戳，包含时区，解析度为1微秒/14位数。示例：`'2004-10-19 10:23:54+02'`
date	date	支持	4 字节	4713 BC 到 5874897 AD	单位是一天

decimal	numeric	支持	可变长	小数点前 38 位；小数点后 38 位	需要指定 precision, scale信息

对于timestamp with time zone类型，SQL标准通过判断timestamptz类型数据的“+”或者“-”符号以及符号后面的时区偏移来识别时区，如果未表明时区偏移，将会有一个默认时区添加到数据上。

说明:

交互式分析当前版本支持的decimal和PostgreSQL的不同之处如下：

- 交互式分析支持precision的范围从公0到38，scale范围支持从0到precision。
- 交互式分析的decimal需要明确指定precision和scale信息，不能使用省略的方式。

关于timestamptz、date、decimal的用法示例SQL如下：

```
CREATE TABLE test_data_type (
 tswtz_column timestamp WITH TIME ZONE,
 date_column date,
 decimal_column decimal(38, 10)
);

INSERT INTO test_data_type
VALUES ('2004-10-19 08:08:08', '2004-10-19', 123.456);

SELECT * FROM test_data_type;
```

数组类型

交互式分析当前版本只支持一维数组，且仅支持如下数组类型：

- int4[]
- int8[]
- float4[]
- float8[]
- boolean[]
- text[]

示例用法如下：

- 数组的声明

```
create table array_example(
int4_array int4[],
int8_array int8[],
float4_array float4[],
float8_array float8[],
boolean_array boolean[],
```

```
text_array text[]);
```

- **数组的插入**
  - **使用ARRAY关键字**

```
insert into array_example(
int4_array,
int8_array,
float4_array,
float8_array,
boolean_array,
text_array)
values (ARRAY[1, 2, 3, 4],
ARRAY[1, 2, 3, 4],
ARRAY[1.0, 2.0],
ARRAY[1.0, 2.0, 3.0],
ARRAY[true, true, false],
ARRAY['foo1', 'foo2', 'foo3']);
```

  - **使用{}表达式**

```
insert into array_example(
int4_array,
int8_array,
float4_array,
float8_array,
boolean_array,
text_array)
values ('{1, 2, 3, 4}',
'{1, 2, 3, 4}',
'{1.0, 2.0}',
'{1.0, 2.0, 3.0}',
'{true, true, false}',
'{"foo1", "foo2", "foo3"}');
```

- **数组的查询**
  - **查询数组中单个元素**

```
SELECT int4_array[3] FROM array_example;
```

  - **查询数组中部分元素**

```
SELECT int4_array[1:2] FROM array_example;
```

4.2 数据库对象和管理

4.2.1 创建和删除数据库

本小节将会为您介绍在交互式分析（Interactive Analytics）中对数据库的命令操作。

普通用户获取数据库命令权限请参考角色管理。并向superuser提出权限申请。

创建数据库

交互式分析（Interactive Analytics）语法是PostgreSQL的一个子集。目前支持的创建数据库语法如下：

```
CREATE DATABASE db_name
    [ [ WITH ] [ OWNER [=] user_name ];
```

说明：

：

1. **其中name遵循SQL标识符的一般规则。**
2. **当前角色自动成为该新数据库的拥有者，以后删除这个数据库也是该拥有者的特权（同时还会删除其中的所有对象，即使那些对象有不同的拥有者）。**
3. **作为superuser可为其他人创建数据库，并使其成为新的数据库owner，这样owner就能自行管理和配置该数据库。**

删除数据库

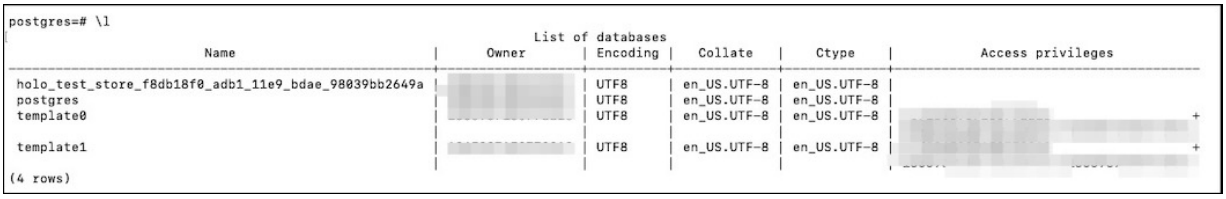
```
DROP DATABASE [ IF EXISTS ] db_name;
```

说明：

：只有该数据库的superuser，或者被superuser设置成该数据库的owner，才能删除该数据库。删除数据库会移除其中包括的所有对象。删除的数据库不能被销毁。不能在与目标数据库连接时执行DROP命令，但可以连接到其他任意数据库。

查看数据库

在psql客户端，输入\l命令，即可查看当前项目中的数据库，包括系统内置的数据库。



4.2.2 创建和管理表

本小节将会为您介绍在交互式分析（Interactive Analytics）中对表的命令操作。

关于分区表详细命令操作，请见创建和管理分区表。

关于外部表详细命令操作，请见创建和管理外部表。

创建表

目前交互式分析（Interactive Analytics） 语法是PostgreSQL的一个子集，支持的建表语法有如下三种。

```
-- 典型的create table语句
create table [if not exists] [schema_name.]table_name ([
  {
   column_name column_type [column_constraints, [...]]
   | table_constraints
   [, ...]
  }
]);

-- 典型的create partition table语句
create table [if not exists] [schema_name.]table_name partition by
list (column_name) ([
  {
   column_name column_type [column_constraints, [...]]
   | table_constraints
   [, ...]
  }
]);

-- 典型的create child partition table语句
create table [if not exists] [schema_name.]table_name partition of
parent_table
  for values in (string_literal);
```

 **说明:**

:

1. **column_type支持**`int, integer, int4, int8, bigint, bool, boolean, float, double precision, float8, text, varchar, timestamp`**，详见数据类型。**
2. **列约束**`column_constraints`**和表约束**`table_constraints`**的支持情况如下。**

column_constraints	table_constraints
primary key	支持
not null	支持
null	支持
unique	不支持
check	不支持
default	不支持

3. `column_onstraints`**不能有多列为**`primary key`**，但**`tableConstraints`**允许设置多列为表**`primary key`**。**

4. 表名和列名均对大小写不敏感，如需定义大写表名、大写列名、特殊字符表名或列名，可使用双引号“进行转义。例如：

```
create table "TBL"(a int);
select relname from pg_class where relname = 'TBL';
insert into "TBL" values (-1977);
select * from "TBL";
-----------------------------------------------------------------------
begin;
create table tbl ("C1" int not null);
call set_table_property('tbl', 'clustering_key', '"C1"');
commit;
-----------------------------------------------------------------------
begin;
create table tbl ("C1" int not null, c2 text);
call set_table_property('tbl', 'clustering_key', '"C1,c2:desc"');
 -- set_table_property 见下文
call set_table_property('tbl', 'segment_key', '"C1",c2:desc');
commit;
-----------------------------------------------------------------------
create table "Tab_$A%*" (a int);
select relname from pg_class where relname = 'Tab_$A%*';
insert into "Tab_$A%*" values (-1977);
select * from "Tab_$A%*";
```

5. 在创建表时，如果不存在同名表且语义正确，表创建都会返回成功。如果不指定`IF NOT EXISTS`选项而存在同名表，则返回异常。如果指定`IF NOT EXISTS`选项，交互式分析（Interactive Analytics）会提示信息，跳过表创建步骤，返回成功。直观的规则如下。

指定`IF NOT EXISTS`	不指定`IF NOT EXISTS`
存在同名表	NOTICE：relation “xx “already exists，skippingSUCCEED
不存在同名表	SUCCEED

6. 如未做双引号“转义，则表名、列名中不能有特殊字符，只能用英文的a-z、A-Z及数字和下划线（），且必须以字母开头。由于大小写不敏感，A-Z统一会被看成小写。例如。

```
begin;
create table TABLE_one (a int not null);
call set_table_property('table_one', 'clustering_key', 'a');
insert into table_one values (12);
commit;
```

7. 表名的长度不超过64字节，超过64字节将被截断。

8. `partition by`类型仅支持list，切分partition list只能有一个值，且类型只能为string。

设置表属性

在交互式分析（Interactive Analytics）中，可以通过set_table_property为表设置多种属性，合理的表属性设置可以有助于系统高效地组织和查询数据。执行命令如下。

```
call set_table_property('tbl', property, value)
```

说明:

: set_table_property的调用需要与create table在同一事务中执行。

主要语法支持且仅支持以下几种。

```
call set_table_property('table_name', 'orientation', '[column | row
]');
call set_table_property('table_name', 'clustering_key', '[columnName{:
[desc|asc]} [,...]]');
call set_table_property('table_name', 'segment_key', '[columnName
 [,...]]');
call set_table_property('table_name', 'bitmap_columns', '[columnName
 [,...]]');
call set_table_property('table_name', 'dictionary_encoding_columns',
 '[columnName [,...]]');
call set_table_property('table_name', 'time_to_live_in_seconds', '<
non_negative_literal>');
call set_table_property('table_name', 'distribution_key', '[columnName
[,...]]');
```

下面分别对每种做详细定义。

1. orientation

```
call set_table_property('tbl', 'orientation', '[column | row]');
```

- **orientation属性指定了数据库表在交互式分析（Interactive Analytics）中是列存还是行存。**
- **在交互式分析（Interactive Analytics）中，数据库表默认为列存（column store）形式。列存对于OLAP场景较为友好，适合各种复杂查询，行存对于kv场景比较友好，适合基于primary key的点查和扫描scan。**
- **举例**

```
begin;
create table tbl (a int not null, b text not null);
call set_table_property('tbl', 'orientation', 'row');
```

```
commit;
```

2. clustering key

```
call set_table_property('tbl', 'clustering_key', '[columnName{:[desc
|asc]} [,...]]');
```

- clustering_key指定一些列作为聚簇索引：交互式分析（Interactive Analytics）在指定的列上将建立聚簇索引。交互式分析（Interactive Analytics）会在聚簇索引上对数据进行排序，建立聚簇索引能够加速用户在索引列上的range和filter查询。
- clustering_key指定的列必须满足非空约束（not null）。
- clustering_key指定列时，可在列名后添加：desc或者asc来表明构建索引时的排序方式。排序方式默认为asc，即升序。
- 数据类型为float/double的列，不能设置为clustering_key。
- 举例

```
begin;
create table tbl (a int not null, b text not null);
call set_table_property('tbl', 'clustering_key', 'a,b');
commit;
----------------------------------------------------------------
begin;
create table tbl (a int not null, b text not null);
call set_table_property('tbl', 'clustering_key', 'a:desc,b:asc');
commit;
```

3. bitmap columns

```
call set_table_property('tbl', 'bitmap_columns', '[columnName
 [,...]]');
```

- bitmap_columns指定比特编码列，交互式分析（Interactive Analytics）在这些列上构建比特编码。bitmap可以对segment内部的数据进行快速过滤，所以建议用户把filter条件的数据建成比特编码。
- 设置bitmap_columns要求orientation为 column，即列存表。
- bitmap_columns适合无序且取值不多的列，对于每个取值构造一个二进制串，表示取值所在位置的bitmap。
- bitmap_columns指定的列可以为null。
- 默认所有text列都会被隐式地设置到bitmap_columns中。
- 举例

```
begin;
create table tbl (a int not null, b text not null);
call set_table_property('tbl', 'bitmap_columns', 'a,b');
```

```
commit;
```

4. dictionary encoding columns

```
call set_table_property('tbl', 'dictionary_encoding_columns', '[
columnName [,...]]');
```

- dictionary_encoding_columns指定字典编码列，交互式分析（Interactive Analytics）为指定列的值构建字典映射。字典编码可以将字符串的比较转成数字的比较，加速group by、filter等查询。
- 设置dictionary_encoding_columns要求orientation为column，即列存表。
- dictionary_encoding_columns指定的列可以为null。
- 无序但取值较少的列适合设置dictionary_encoding_columns，可以压缩存储。
- 默认所有text列都会被隐式地设置到dictionary_encoding_columns中。
- 举例

```
begin;
create table tbl (a int not null, b text not null);
call set_table_property('tbl', 'dictionary_encoding_columns', 'a,b
');
commit;
```

5. time to live in seconds

```
call set_table_property('tbl', 'time_to_live_in_seconds', '<
non_negative_literal>');
```

- time_to_live_in_seconds指定了表的生存时间，单位为秒，必须是非负数字类型，整数或浮点数均可。
- 举例

```
begin;
create table tbl (a int not null, b text not null);
call set_table_property('tbl', 'time_to_live_in_seconds', '3.14159
');
commit;
----------------------------------------------------------------
begin;
create table tbl (a int not null, b time not null);
call set_table_property('tbl', 'time_to_live_in_seconds', '86400
');
```

```
commit;
```

6. distribution_key

```
call set_table_property('table_name', 'distribution_key', '[
columnName[,...]]');
```

- distribution_key属性指定了数据库表分布策略。
- columnName部分如设置单值，不要有多余空格。如设置多值，则以逗号分隔，同样不要有多余的空格。
- distribution_key指定的列可以为null。
- 交互式分析中，数据库表默认为随机分布形式。数据将被随机分配到各个shard上。如果制定了分布列，数据将按照指定列，将数据shuffle到各个shard，同样的数值肯定会在同样的shard中。当以分布列做过滤条件时，交互式分析可以直接筛选出数据相关的shard进行扫描。当以分布列做join条件时，交互式分析不需要再次将数据shuffle到其他计算节点，直接在本节点join本节点数据即可，可以大大提高执行效率。
- 举例

```
begin;
create table tbl (a int not null, b text not null);
call set_table_property('tbl', 'distribution_key', 'a');
commit;

begin;
create table tbl (a int not null, b text not null);
call set_table_property('tbl', 'distribution_key', 'a,b');
commit;
```

删除表

交互式分析（Interactive Analytics）的删除表（DROP TABLE）语法如下。

```
DROP TABLE [ IF EXISTS ] table_name [, ...];
```

说明:

:

1. DROP TABLE支持一次DROP多个表。
2. 如果指定IF EXISTS，无论表存在与否，都会返回成功。如果不指定IF EXISTS选项而表不存在，则返回异常：ERROR: table “non_exist_table“does not exist。

修改表

交互式分析当前版本对表的修改（ALTER TABLE）仅支持重命名表（RENAME TABLE）和增加列（ADD COLUMN）。对于外部表（foreign table）没有限制。

针对交互式分析的分区表，还支持ATTACH PARTITION和DETACH PARTITION两种修改。详见创建和管理分区表一节。

- 重命名表

  交互式分析的重命名普通表语法如下：

```
ALTER TABLE table_name RENAME to new_table_name;
ALTER FOREIGN TABLE my_foreign_table_name to my_new_foreign_table
_name;
```

说明：

：如果RENAME不存在的表，或者将表重命名为已存在的表名，都会返回异常。

- 增加列

  交互式分析给普通表增加列的语法如下：

```
ALTER TABLE IF EXISTS table_name ADD COLUMN col_add_1, ADD COLUMN IF
 NOT EXISTS col_add_2;
```

增加注释

交互式分析支持给表、外表、列等增加注释的功能， 使用方法可以参见*PostgreSQL*。

示例建表语句如下：

```
-- 给表增加注释
COMMENT ON TABLE table_name IS 'my comments on table table_name.';

-- 给列增加注释
COMMENT ON COLUMN table_name.col1 IS 'This my first col1';

-- 给外部表增加注释
COMMENT ON FOREIGN TABLE foreign_table IS ' comments on my foreign
table';
```

4.2.3 创建和管理分区表

本小节将会为您介绍在交互式分析（Interactive Analytics）中对分区表的命令操作。

创建分区表

交互式分析创建分区表的父表和子表的语法如下。

```
-- 典型的create partition table语句
create table [if not exists] [schema_name.]table_name partition by
list (column_name) ([
  {
   column_name column_type [column_constraints, [...]]
   | table_constraints
   [, ...]
  }
]);
```

```
-- 典型的create child partition table语句
create table [if not exists] [schema_name.]table_name partition of
parent_table
  for values in (partition_value);
```

示例语句如下：

```
begin;
create table hologres_parent(a text primary key, b int, c timestamp, d
 text) partition by list(a);
call set_table_property('hologres_parent', 'orientation', 'column');
call set_table_property('hologres_parent', 'clustering_key', 'a,b');
call set_table_property('hologres_parent', 'segment_key', 'c');
call set_table_property('hologres_parent', 'bitmap_columns', 'a,d');
call set_table_property('hologres_parent', 'dictionary_encoding_
columns', 'a,d');
call set_table_property('hologres_parent', 'time_to_live_in_seconds',
 '86400');
create table hologres_child1 partition of hologres_parent for values
in('a');
create table hologres_child2 partition of hologres_parent for values
in('b');
create table hologres_child3 partition of hologres_parent for values
in('c');
commit;
```

1. **分区父表和分区子表的定义需要包含在事务中且需要在同一个事务中。**
2. **不能向父表插入任何数据。**
3. `DROP`**父表将默认同时**`DROP`**子表。**
4. **只有text/varchar类型才能作为分区键（partition key）。**
5. **一个分区规则只能创建一个分区表。**
6. **分区值（partition value）必须是单个字符串类型的值，如‘a‘，‘abc‘等。**

修改分区表

交互式分析支持三种更改分区表的操作：

```
ALTER TABLE [IF EXISTS] table_name RENAME to new_table_name;
ALTER TABLE [IF EXISTS] table_name ATTACH PARTITION new_partition_name
 FOR VALUES in (<string_literal>);
ALTER TABLE [IF EXISTS] table_name DETACH PARTITION paritition_name;
```

1. `ATTACH PARTITION partition_name FOR VALUES in (<string_literal>)`

   这种形式使用与`CREATE TABLE`**相同的** `partition_bound_spec`**语法，将现有的表（可能本身是分区）作为目标表的分区。分区绑定规范必须对应于目标表的分区策略和分区键。要绑定的表必须与目标表具有相同的列，而不能多或少；此外，列类型也必须匹配。此外，它必须具**

有目标表的所有NOT NULL约束。如果附加一个不接受NULL值的列表分区，除非它是一个表达式， 否则向分区键列添加NOT NULL约束。

2. DETACH PARTITION partition_name

该形式分离目标表的指定分区。分离的分区作为独立表继续存在，但不再与分离的表相关联。

删除分区表

删除分区表（DROP PARTITION TABLE）的操作与删除常规表的操作一致，参见创建和管理表。

示例如下：

```
DROP PARTITION TABLE test;
```

4.2.4 创建和管理外部表

本小节将会为您介绍在交互式分析（Interactive Analytics）中对外部表的命令操作。

说明：

：本版本的交互式分析（Interactive Analytics）仅支持对MaxCompute表操作。

创建外部表

创建外部表使用CREATE FOREIGN TABLE命令，示例如下：

```
CREATE FOREIGN TABLE [ IF NOT EXISTS ] table_name ( [
  { column_name data_type }
  [, ... ]
  ] )
SERVER odps_server
[ OPTIONS ( option 'value' [, ... ] ) ]
```

举例说明：

```
create foreign table test(key text)
server odps_server options(project_name 'odps_project', table_name '
test');
```

说明：

：options必须要填写project_name和table_name，其中project_name的值是MaxCompute中project的名字，table_name 是project中表的名字。

查询外部表

查询外部表使用SELECT FOREIGN TABLE，相关命令语法请参考DML的数据查询。

典型的查询MaxCompute外部表的示例：

```
create foreign table src_pt(id text, pt text) server test_server
options(project_name 'projectname', table_name 'tablename');
select * from src_pt;
```

典型的将MaxCompute外部表数据导入交互式分析（Interactive Analytics）表的示例：

```
create foreign table src_pt_odps(id text, pt text) server test_server
options(project_name 'projecname', table_name 'tablename');
begin;
create table src_pt(id text, pt text);
commit;
insert into src_pt select * from src_pt_odps;
```

说明:

- **project_name为要查询的MaxCompute表所在的项目名称。**
- **table_name为要查询的MaxCompute表所在的表名称。**

修改外部表

交互式分析（Interactive Analytics） 修改外表的语义当前只支持RENAME操作，语法如下：

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

举例说明：

```
alter foreign table test rename to new_test_table
```

删除外部表

交互式分析（Interactive Analytics） 删除外表的语义如下：

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

举例说明：

```
drop foreign table test;
```

MaxCompute与交互式分析的数据类型映射

MaxCompute数据类型与交互式分析数据类型对照表如下：

MaxCompute 类型	交互式分析（Interactive Analytics） 类型	直接查询 / 导入到交互式分析表	说明
SMALLINT	int2	支持/转成BIGINT可支持	无

INT	int4，int，integer	支持/转成BIGINT可支持	无
BIGINT	int8，BIGINT	支持/转成BIGINT可支持	无
BOOLEAN	boolean，bool	支持/支持	无
DOUBLE	float8，double precision	支持/转换成float支持	无
FLOAT	float4	支持/转换成float支持	无
DECIMAL	numeric	支持decimal1.0/不支持持	无
VarChar	text，VarChar	支持/支持	无
STRING	text，VarChar	支持/支持	无
DATETIME	timestamp with time zone	支持/转成timestamptz支持	交互式分析仅支持 `timstamp with time zone`

说明:

:

1. MaxCompute DATETIME是日期时间类型，使用东八区时间作为系统标准时间。范围从0000年1月1日到9999年12月31日，精确到毫秒 。
2. 交互式分析（Interactive Analytics）支持的是PostgreSQL的timestamptz，带时区，范围从 4713 BC 到 294276 AD ，且精确到微秒。
3. 当MaxCompute表中含有交互式分析（Interactive Analytics）不支持的类型的时候， 如果不访问交互式分析（Interactive Analytics）不支持字段，其它字段可以正常查询

4.3 数据查询和操作

4.3.1 数据写入

本小节将会为您介绍在交互式分析（Interactive Analytics）中数据写入INSERT的语法格式。

语法

```
INSERT INTO table [( column [, ...] )]
   VALUES ( {expression}  [, ...] )
```

```
    [, ...] | query}
```

INSERT将新行插入到一个表中。我们可以插入一个或者更多由值表达式指定的行，或者插入来自一个查询的零行或者更多行。

目标列的名称可以以任意顺序列出。如果没有给出列名列表，则有两种确定目标列的可能性。第一种是以被声明的顺序列出该表的所有列。另一种可能性是，如果VALUES子句或者query只提供N个列，则以被声明的顺序列出该表的前N列。VALUES子句或者query提供的值会被从左至右关联到这些显式或者隐式给出的目标列。

每一个没有出现在显式或者隐式列列表中的列都将被默认填充，如果为该列声明过默认值则用默认值填充，否则用空值填充。如果任意列的表达式不是正确的数据类型，将会尝试自动类型转换。

参数

table_name：一个已有表的名称（可以被模式限定）。

alias：table_name的别名。当提供了一个别名时，它会完全隐藏掉表的实际名称。

column_name：名为table_name的表中的一个列的名称。如有必要，列名可以用一个子域名或者数组下标限定（指向一个组合列的某些列中插入会让其他域为空）。

expression：要赋予给相应列的表达式或者值。

query：提供要被插入行的查询（SELECT语句）。 其语法描述请参考SELECT语句。

示例

在交互式分析（Interactive Analytics）中，目前INSERT支持以下数据写入方式：

1. insert into values

```
INSERT INTO rh_holo2mysqltest (cate_id, cate_name) VALUES
    (3, 'true'),
    (3, 'fale'),
    (3, 'trxxue'),
    (3, 'x'),
    (4, 'The Dinner Game');
```

2. insert into select

```
insert into test2
select 2,'two';
```

4.3.2 数据查询

本小节将会为您介绍在交互式分析（Interactive Analytics）中数据查询SELECT的语法格式。

语法

```
SELECT [ALL | DISTINCT [ON (expression [, ...])]]
  * | expression [[AS] output_name] [, ...]
```

```
[FROM from_item [, ...]]
[WHERE condition]
[GROUP BY grouping_element [, ...]]
[HAVING condition [, ...]]
[{UNION | INTERSECT | EXCEPT} [ALL] select]
[ORDER BY expression [ASC | DESC | USING operator] [, ...]]
[LIMIT {count | ALL}]
```

其中：

- **grouping_element包含**

```
expression
```

- **from_item包含**

```
table_name [[AS] alias [( column_alias [, ...] )]]
(select) [AS] alias [( column_alias [, ...] )]
from_item [NATURAL] join_type from_item
          [ON join_condition | USING ( join_column [, ...] )]
```

描述

数据查询SELECT可以进行多种形式的查询。 SELECT的通常用法如下：

1. **FROM列表中的所有元素都会被计算（ FROM中的每一个元素都是一个真实表或者虚拟表）。如果在FROM列表中指定了多于一个元素，得到的结果做并集。**
2. **如果指定了WHERE子句，所有不满足该条件的行都会被从输出中消除。**
3. **如果指定了`GROUP BY`子句或者如果有聚集函数，输出会被组合成由在一个或者多个值上匹配的行构成的分组，并且在其上计算聚集函数的结果。如果出现了HAVING子句，它会消除不满足给定条件的分组。**
4. **对于每一个被选中的行或者行组，会使用SELECT输出表达式计算实际的输出行。**
5. **`SELECT DISTINCT`从结果中消除重复的行。 `SELECT DISTINCT ON`消除在所有指定表达式上匹配的行。`SELECT ALL`（默认）将返回所有候选行， 包括重复的行。**
6. **通过使用操作符UNION、INTERSECT和EXCEPT，多于一个SELECT语句的输出可以被整合形成一个结果集。UNION操作符返回位于一个或者两个结果集中的全部行。INTERSECT操作符返回同时位于两个结果集中的所有行。EXCEPT操作符返回位于第一个结果集但不在第二个结果集中的行。在所有三种情况下，重复行都会被消除（除非指定ALL）。可以增加DISTINCT来显式的消除重复行。注意虽然ALL是SELECT自身的默认行为，但这里DISTINCT是默认行为。**
7. **如果指定了`ORDER BY`子句，被返回的行会以指定的顺序排序。如果没有给定`ORDER BY`，系统会以能最快产生行的顺序返回它们。**
8. **如果指定了LIMIT（或FETCH FIRST） 或者OFFSET子句，SELECT语句只返回结果行的一个子集。**

参数

1. WITH列表

   with列表是位于SELECT之前或者作为SELECT子句的subquery存在，通常位于SELECT之前，用于定义子查询，并给这些子查询声明一个名字和返回的列名，定义每一个with子句为一个CTE（Common Table Expression），定义语法如下：

   ```
   with_query_name [ ( column_name [, ...] ) ] AS ( select )
   ```

   `with_query_name`指定当前CTE的名字，可以是任意有效的标识符，`column_name`列表对应着子查询返回值的列名，类似于SELECT子句中的AS的语义，子查询可以是一个常规的SELECT查询。

   CTE之间通过逗号分隔，后面出现的CTE定义可以引用前面定义的CTE，但是目前暂时不支持递归的CTE调用，在之后的查询中，可以直接将`with_query_name`作为一个视图（view）出现在查询中，如果没有指定`column_name`列表，该view的`column_name`为对应子查询返回列的列名，如果指定`column_name`列表，则需要使用定义`column_name`，并且`column_name`列表需要和SELECT返回值列表个数相同。

2. SELECT列表

   SELECT列表（位于关键词 SELECT和FROM之间）指定构成 SELECT语句输出行的表达式。这些表达式可以（并且通常确实会）引用FROM子句中计算得到的列。

   正如在表中一样，SELECT的每一个输出列都有一个名称。 在一个简单的SELECT中，这个名称只是被用来标记要显示的列，但是当SELECT是一个大型查询的一个子查询时，大型查询会把该名称看做子查询产生的虚表的列名。要指定用于输出列的名称，在该列的表达式 后面写上`AS output_name`（可以省略AS，但只能在期望的输出名称不匹配任何 PostgreSQL关键词时省略。为了避免和未来增加的关键词冲突， 推荐总是写上AS或者用双引号引用输出名称）。如果不指定列名， PostgreSQL会自动选择一个名称。如果列的表达式是一个简单的列引用，那么被选择的名称就和该列的名称相同。在使用函数或者类型名称的更复杂的情况中，系统可能会生成诸如`column`undefined之类的名称。

   一个输出列的名称可以被用来在`ORDER BY`以及`GROUP BY`子句中引用该列的值，但是不能用于WHERE和HAVING子句（在其中必须写出表达式）。

   可以在输出列表中写*来取代表达式，它是被选中行的所有列的一种简写方式。还可以写`%table_name*;`，它是只来自那个表的所有列的简写形式。在这些情况中无法用AS指定新的名称，输出行的名称将和表列的名称相同。

3. FROM子句

FROM子句为SELECT 指定一个或者更多源表。如果指定了多个源表，结果将是所有源表的笛卡尔积（交叉连接）。但是通常会增加限定条件（通过 WHERE）来把返回的行限制为该笛卡尔积的一个小子集。

FROM子句可以包含下列元素：

table_name：一个现有表或视图的名称（可以限定表的schma模式）。

alias：一个包含别名的FROM项的替代名称。别名被用于让书写简洁或者消除自连接中的混淆（其中同一个表会被扫描多次）。当提供一个别名时，表或者函数的实际名称会被隐藏。例如，给定`FROM foo AS f`，SELECT的剩余部分就必须以 f而不是foo来引用这个 FROM项。

select：一个子-SELECT可以出现在 FROM子句中。这就好像把它的输出创建为一个存在于该SELECT命令期间的临时表。注意子-SELECT必须用圆括号包围，并且必须为它提供一个别名。

function_name：函数调用可以出现在FROM子句中（对于返回结果集合的函数特别有用，但是可以使用任何函数）。这就好像把该函数的输出创建为一个存在于该SELECT命令期间的临时表。

可以用和表一样的方式提供一个别名。如果写了一个别名，还可以写一个列别名列表来为该函数的组合返回类型的一个或者多个属性提供替代名称， 包括由ORDINALITY（如果有）增加的新列。

通过把多个函数调用包围在`ROWS FROM（。）`中可以把它们整合在单个FROM-子句项中。这样一个项的输出是把每一个函数的第一行串接起来，然后是每个函数的第二行，以此类推。如果有些函数产生的行比其他函数少，则在缺失数据的地方放上 NULL，这样被返回的总行数总是和产生最多行的函数一样。

```
join_type
[ 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`NATURAL`、`ON join_condition`或者 `USING （join_column [，...]）`之一（只能有一种）。其含义见下文。对于`CROSS JOIN`，上述子句不能出现。一个JOIN子句联合两个FROM项（为了方便我们称之为“表“，尽管实际上它们可以是任何类型的FROM项）。如有必要可以使用圆括号确定嵌

套的顺序。 在没有圆括号时，JOIN会从左至右嵌套。在任何情况下，JOIN的联合比用逗号分隔FROM-列表项更强。

`CROSS JOIN`和`INNER JOIN` 会产生简单的笛卡尔积，也就是与在FROM列出两个表得到的结果相同，但是要用连接条件（如果有）约束该结果。`CROSS JOIN`与`INNER JOIN ON (TRUE)` 等效，也就是说条件不会移除任何行。这些连接类型只是一种记号上的方便，因为没有什么是用纯粹的FROM和 WHERE能做而它们不能做的。

`LEFT OUTER JOIN`返回被限制过的笛卡尔积中的所有行（即所有通过了其连接条件的组合行），以及左表中每行的一个副本，因为没有右行通过了连接条件。。通过在右手列中插入空值，这种左手行会被扩展为连接表的完整行。注意在决定哪些行匹配时，只考虑JOIN子句自身的条件。之后才应用外条件。

相反，`RIGHT OUTER JOIN`返回所有连接行，外加每一个没有匹配上的右手行（在左端用空值扩展）。这只是为了记号上的方便，因为可以通过交换左右表把它转换成一个`LEFT OUTER JOIN`。

`FULL OUTER JOIN`返回所有连接行，外加每一个没有匹配上的左手行（在右端用空值扩展），再外加每一个没有匹配上的右手行（在左端用空值扩展）

`join_condition`是一个会得到boolean类型值的表达式（类似于一个WHERE子句），它说明一次连接中哪些行被认为相匹配。

形式`USING (a, b...)` 的子句是`ON left_table.a = right_table.a AND left_table.b = right_table.b ...`的简写。还有， USING表示每一对相等列中只有一个会被包括在连接输出中。

NATURAL：是列出在两个表中所有具有相同名称的列的USING的简写。

4. WHERE子句

可选的WHERE子句的形式

```
WHERE condition
```

其中condition是任一计算得到布尔类型结果的表达式。任何不满足这个条件的行都会从输出中被消除。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。

5. GROUP BY子句

可选的GROUP BY子句的形式

```
GROUP BY grouping_element [, ...]
```

GROUP BY将会把所有被选择的行中共享相同分组表达式值的那些行压缩成一个行。grouping_element中使用的expression可以是输入列名、输出列（SELECT列表项） 的名称或序号或者由输入列值构成的任意表达式。在出现歧义时， GROUP BY名称将被解释为输入列名而不是输出列名。

如果任何GROUPING SETS、ROLLUP或者CUBE作为分组元素存在，则GROUP BY子句整体上定义了数个独立的分组集。其效果等效于在具有独立分组集作为它们的GROUP BY子句的子查询间构建一个UNION ALL。

聚集函数（如果使用）会在组成每一个分组的所有行上进行计算，从而为每一个分组产生一个单独的值（如果有聚集函数但是没有GROUP BY子句，则查询会被当成是由所有选中行构成的一个单一分组）。传递给每一个聚集函数的行集合可以通过在聚集函数调用附加一个FILTER子句来进一步过滤。当存在一个FILTER子句时，只有那些匹配它的行才会被包括在该聚集函数的输入中。

当存在GROUP BY子句或者任何聚集函数时， SELECT列表表达式不能引用非分组列（除非它出现在聚集函数中或者它函数依赖于分组列），因为这样做会导致返回非分组列的值时会有多种可能的值。如果分组列是包含非分组列的表的主键（ 或者主键的子集），则存在函数依赖。

记住所有的聚集函数都是在HAVING子句或者 SELECT列表中的任何“标量“表达式之前被计算。 这意味着一个CASE表达式不能被用来跳过一个聚集表达式的计算。

6. DISTINCT 子句

如果指定了SELECT DISTINCT，所有重复的行会被从结果集中移除（为每一组重复的行保留一行）。SELECT ALL则指定相反的行为：所有行都会被保留，这也是默认情况。

SELECT DISTINCT ON （expression [, ...]）只保留在给定表达式上计算相等的行集合中的第一行。 DISTINCT ON表达式使用和ORDER BY相同的规则（见上文）解释。注意，除非用ORDER BY来确保所期望的行出现在第一位，每一个集合的“第一行“是不可预测的。例如：

```
SELECT DISTINCT ON (location) location, time, report
    FROM weather_reports
```

```
    ORDER BY location, time DESC;
```

为每个地点检索最近的天气报告。但是如果我们不使用ORDER BY来强制对每个地点的时间值进行降序排序， 我们为每个地点得到的报告的时间可能是无法预测的。

DISTINCT ON表达式必须匹配最左边的ORDER BY表达式。ORDER BY子句通常将包含额外的表达式，这些额外的表达式用于决定在每一个DISTINCT ON分组内行的优先级。

7. COUNT DISTINCT子句

COUNT DISTINCT支持计算去重之后的某一个column的个数，对于该列中出现多次的值只会被计算一次，和COUNT的计算类似，如果该列包含NULL值，它将不会计算在内。

精确计算的语法示例如下：

```
SELECT c1, COUNT(DISTINCT c2) FROM table GROUP BY c1
```

由于精确计算的COUNT DISTINCT需要消耗较大的资源，因此交互式分析还支持非精确的COUNT DISTINCT计算，语法示例如下：

```
SELECT c1, approx_count_distinct(c2) FROM table GROUP BY c1
```

8. UNION 子句

```
select_statement UNION [ ALL | DISTINCT ] select_statement
```

select_statement是任何没有ORDER BY、LIMIT、 FOR NO KEY UPDATE、FOR UPDATE、 FOR SHARE和FOR KEY SHARE子句的SELECT语句（如果子表达式被包围在圆括号内， ORDER BY和LIMIT可以被附着到其上。如果没有圆括号，这些子句将被应用到UNION的结果而不是右手边的表达式上）。

UNION操作符计算所涉及的SELECT语句所返回的行的并集。如果一至少出现在两个结果集中的一个内，它就会在并集中。作为UNION两个操作数的SELECT语句必须产生相同数量的列并且对应位置上的列必须具有兼容的数据类型。

UNION的结果不会包含重复行，除非指定了ALL选项。ALL会阻止消除重复（因此，UNION ALL通常显著地快于UNION，尽量使用ALL）。可以写DISTINCT来显式地指定消除重复行的行为。

除非用圆括号指定计算顺序，同一个SELECT语句中的多个UNION操作符会从左至右计算。

9. INTERSECT 子句

INTERSECT子句具有下面的形式：

```
select_statement INTERSECT [ ALL | DISTINCT ] select_statement
```

select_statement是任何没有ORDER BY LIMIT子句的SELECT语句。

INTERSECT操作符计算所涉及的 SELECT语句返回的行的交集。如果一行同时出现在两个结果集中，它就在交集中。

INTERSECT的结果不会包含重复行，除非指定了 ALL选项。如果有ALL，一个在左表中有m次重复并且在右表中有n 次重复的行将会在结果中出现min（m，n）次。 DISTINCT可以写DISTINCT来显式地指定消除重复行的行为。

除非用圆括号指定计算顺序， 同一个SELECT语句中的多个 INTERSECT操作符会从左至右计算。 INTERSECT的优先级比UNION更高。也即， A UNION B INTERSECT C将被读成A UNION (B INTERSECT C) 。

10.EXCEPT 子句

EXCEPT子句具有下面的形式：

```
select_statement EXCEPT [ ALL | DISTINCT ] select_statement
```

select_statement是任何没有ORDER BY、LIMIT子句的 SELECT语句。

EXCEPT操作符计算位于左SELECT语句的结果中但不在右SELECT语句结果中的行集合。

EXCEPT的结果不会包含重复行，除非指定了 ALL选项。如果有ALL，一个在左表中有 m次重复并且在右表中有n次重复的行将会在结果集中出现max（m-n，0）次。 DISTINCT可以写DISTINCT来显式地指定消除重复行的行为。

除非用圆括号指定计算顺序， 同一个SELECT语句中的多个 EXCEPT操作符会从左至右计算。EXCEPT的优先级与 UNION相同。

当前，FOR NO KEY UPDATE、FOR UPDATE、 FOR SHARE和FOR KEY SHARE不能用于EXCEPT结果或者 EXCEPT的任何输入。

11.ORDER BY 子句

可选的`ORDER BY`子句的形式如下：

```
ORDER BY expression [ ASC | DESC | USING operator ] [ NULLS { FIRST
 | LAST } ] [, ...]
```

`ORDER BY`子句导致结果行被按照指定的表达式排序。 如果两行按照最左边的表达式是相等的，则会根据下一个表达式比较它们， 以此类推。如果按照所有指定的表达式它们都是相等的，则它们被返回的顺序取决于实现。

每一个expression 可以是输出列（SELECT列表项）的名称或者序号，它也可以是由输入列值构成的任意表达式。

序号指的是输出列的顺序（从左至右）位置。这种特性可以为不具有唯一名称的列定义一个顺序。但这不是必要操作，因为可以使用 AS子句为输出列赋予一个名称。

也可以在ORDER BY子句中使用任意表达式，包括没有出现在SELECT输出列表中的列。因此， 下面的语句是合法的：

```
SELECT name FROM distributors ORDER BY code;
```

这种特性的一个限制是一个应用在UNION、 INTERSECT或EXCEPT子句结果上的 `ORDER BY`只能指定输出列名称或序号，但不能指定表达式。

如果一个ORDER BY表达式是一个既匹配输出列名称又匹配输入列名称的简单名称，ORDER BY将把它解读成输出列名称。这与在同样情况下GROUP BY会做出的选择相反。这种 不一致是为了与 SQL 标准兼容。

可以为ORDER BY子句中的任何表达式之后增加关键词 ASC（上升）或者DESC（下降）。如果没有指定， ASC被假定为默认值。或者，可以在USING 子句中指定一个特定的排序操作符名称。一个排序操作符必须是某个 B-树操作符族的小于或者大于成员。ASC通常等价于 USING <而DESC通常等价于 USING >（但是一种用户定义数据类型的创建者可以准确地定义默认排序顺序是什么，并且它可能会对应于其他名称的操作符）。

如果指定`NULLS LAST`，空值会排在非空值之后；如果指定`NULLS FIRST`，空值会排在非空值之前。如果都没有指定， 在指定或者隐含ASC时的默认行为是`NULLS LAST`， 而指定或

者隐含DESC时的默认行为是NULLS FIRST（因此，默认行为是空值大于非空值）。 当指定USING时，默认的空值顺序取决于该操作符是否为小于或者大于操作符。

注意顺序选项只应用到它们所跟随的表达式上。例如ORDER BY x, y DESC和ORDER BY x DESC, y DESC是不同的。

字符串数据会被根据引用到被排序列上的排序规则排序。根据需要可以通过在 expression中包括一个 COLLATE子句来覆盖，例如ORDER BY mycolumn COLLATE “en_US“。

12.LIMIT 子句

LIMIT子句由两个独立的子句构成：

```
LIMIT { count | ALL }
OFFSET start
```

count指定要返回的最大行数，而start指定在返回行之前要跳过的行数。在两者都被指定时，在开始计算要返回的count行之前会跳过 start行。

如果count表达式计算为NULL，它会被当成LIMIT ALL，即没有限制。如果 start计算为NULL，它会被当作OFFSET 0。

在使用LIMIT时，用一个ORDER BY子句把结果行约束到一个唯一顺序是个好办法。否则讲得到该查询结果行的一个不可预测的子集 — 可能要求从第 10 到第 20 行，但是在什么顺序下的第10到第20呢？除非指定ORDER BY，否则是不知道顺序的。

查询规划器在生成一个查询计划时会考虑LIMIT，因此根据使用的LIMIT和OFFSET，很可能得到不同的计划（得到不同的行序）。所以，使用不同的LIMIT/OFFSET值来选择一个查询结果的不同子集将会给出不一致的结果，除非用ORDER BY强制一种可预测的结果顺序。这不是一个缺陷，它是SQL不承诺以任何特定顺序（除非使用ORDER BY来约束顺序）给出一个查询结果这一事实造成的必然后果。

如果没有一个ORDER BY来强制选择一个确定的子集， 重复执行同样的LIMIT查询甚至可能会返回一个表中行的不同子集。同样，这也不是一种缺陷，再这样一种情况下也无法保证结果的确定性。

示例

- 两表join

```
SELECT f.title, f.did, d.name, f.date_prod, f.kind FROM
```

```
distributors d, films f WHERE f.did = d.did
```

- **GROUP BY分组**

```
SELECT kind, sum(length) AS total FROM films GROUP BY kind;
```

- **HAVING过滤**

```
SELECT kind, sum(length) AS total FROM films GROUP BY kind
HAVING sum(length) < interval '5 hours';
```

- **ORDER BY**

```
SELECT * FROM distributors ORDER BY name;
```

4.3.3 其他命令支持

本小节将会为您介绍交互式分析当前版本除INSERT、 SELECT外还支持的命令操作。

交互式分析（Interactive Analytics）是Postgres的子集，目前能支持的Postgres功能如下：

SQL command	是否支持	说明
ALTER ROLE	是	无
ANALYZE	是	无
BEGIN	是	目前仅对DDL有效
COMMIT	是	目前仅对DDL有效
CREATE DATABASE	是	无
CREATE EXTENSION	是	无
CREATE FOREIGN DATA WRAPPER	是	无
CREATE FOREIGN TABLE	是	只支持MaxCompute
CREATE GROUP	是	无
CREATE SERVER	是	无

CREATE TABLE	是	仅支持PostggreSQL的子集（例如，partition类型是能是list，且partition list只能有一个值，且类型只能为string）。 UNLOGGED不支持 TEMP不支持 IF NOT EXISTS不支持 LIKE不支持 CHECK不支持 DEFAULT不支持 GENERATED不支持 UNIQUE不支持 EXCLUDE不支持 FOREIGN KEY不支持 DEFERRABLE不支持 WITH OIDS不支持 GLOBAL/LOCAL不支持
CREATE USER	是	无
CREATE USER MAPPING	是	无
DROP DATABASE	是	无
DROP FOREIGN DATA WRAPPER	是	无
DROP FOREIGN TABLE	是	无
DROP GROUP	是	无
DROP OWNED	是	无
DROP POLICY	是	无
DROP ROLE	是	无
DROP SERVER	是	无
DROP TABLE	是	无

DROP USER	是	无
DROP USER MAPPING	是	无
END	是	只能与DDL语句结合使用
EXPLAIN	是	无
INSERT	是	无
ROLLBACK	是	无
SELECT	是	部分功能支持。 CUBE/GROUPING SET/ ROLL UP不支持 递归查询不支持 NULL FIRST/LAST不支持 INTERSECT/EXCEPT不支持 TABLESAMPLE不支持 Locking不支持 ONLY不支持
SET	是	部分pg的属性设置了也会出现没效果的现象
SET ROLE	是	无
START TRANSACTION	是	无

4.4 内建函数

SQL有很多可用于计数和计算的内建函数，函数的使用，可以方便计算，节约大量时间。

交互式分析（Interactive Analytics）兼容PostgreSQL接口，因此，常规的SQL函数都可以用于交互式分析（Interactive Analytics）开发。关于函数的使用，请参见*PostgreSQL*官网。