# 实验2 基于openGauss数据库定义与操作实验

[实验1.1 数据库定义 2](#_Toc88740451)

[1.1.1数据库 2](#_Toc88740452)

[1.1.2模式 4](#_Toc88740453)

[1.1.3管理表空间 6](#_Toc88740454)

[1.1.3创建和管理表 8](#_Toc88740455)

[实验1.2 数据基本查询 10](#_Toc88740456)

[1.2.1前期准备 11](#_Toc88740457)

[1.2.2简单查询 13](#_Toc88740458)

[1.2.3分组统计查询 14](#_Toc88740459)

[1.2.4连接查询 15](#_Toc88740460)

[实验1.3 数据高级查询 18](#_Toc88740461)

[1.3.1 嵌套查询 18](#_Toc88740462)

[1.3.2 集合查询 19](#_Toc88740463)

[实验1.4 数据更新 20](#_Toc88740464)

[1.4.1向表中插入数据 21](#_Toc88740465)

[1.4.2更新表中数据 22](#_Toc88740466)

[1.4.3查看数据 23](#_Toc88740467)

[1.4.4删除表中数据 23](#_Toc88740468)

[实验1.5 视图 24](#_Toc88740469)

[1.5.1创建视图 24](#_Toc88740470)

[1.5.2管理视图 25](#_Toc88740471)

[1.5.3删除视图 28](#_Toc88740472)

[实验1.6 索引 28](#_Toc88740473)

[1.6.1准备工作 28](#_Toc88740474)

[1.6.2建索引 29](#_Toc88740475)

[1.6.3管理索引 30](#_Toc88740476)

[1.6.4索引创建举例 31](#_Toc88740477)

# 实验2 基于openGauss数据库定义与操作实验

## 实验1.1 数据库定义

（1）实验目的

理解和掌握数据库DDL语言，在OpenGauss数据库上能够熟练地使用SQL DDL语句创建、修改和删除数据库、模式和基本表。

（2）实验内容和要求

理解和掌握SQL DDL语句的语法，特别是各种参数的具体含义和使用方法；使用SQL语句创建、修改和删除数据库、模式和基本表。掌握SQL语句常见语法错误的调试方法。

（3）实验重点和难点

实验重点：创建数据库、基本表。

实验难点：创建基本表时，为不同的列选择合适的数据类型，正确创建表级和列级完整性约束，如列值是否允许为空、主码和外码等。

注意：数据完整性约束，可以在创建基本表时定义，也可以先创建表然后定义完整性约束；由于完整性约束的限制，被引用的表要先创建。

### 1.1.1数据库

#### 创建数据库

使用如下命令创建一个新的数据库db\_tpcc。

openGauss=# CREATE DATABASE db\_tpcc;

CREATE DATABASE

**说明**

● 数据库名称遵循SQL标识符的一般规则。当前角色自动成为此新数据库的所有者。

● 如果一个数据库系统用于承载相互独立的用户和项目，建议把它们放在不同的数据库里。

● 如果项目或者用户是相互关联的，并且可以相互使用对方资源，则应该把它们放在同一个

数据库里，但可以规划在不同的模式中。模式只是一个纯粹的逻辑结构，某个模式的访问权

限由权限系统模块控制。

● 创建数据库时，若数据库名称长度超过63字节，server端会对数据库名称进行截断，保留前63个字节，因此建议数据库名称长度不要超过63个字节。

#### 查看数据库。

● 使用\l元命令查看数据库系统的数据库列表。

openGauss=# \l

● 使用如下命令通过系统表pg\_database查询数据库列表。

openGauss=# SELECT datname FROM pg\_database;

#### 修改数据库

用户可以使用如下命令修改数据库属性（比如：owner、名称和默认的配置属性）。

●使用以下命令为数据库设置默认的模式搜索路径。

openGauss=# ALTER DATABASE db\_tpcc SET search\_path TO pa\_catalog,public;

ALTER DATABASE

● 使用如下命令为数据库重新命名。

openGauss=# ALTER DATABASE db\_tpcc RENAME TO human\_tpcds;

ALTER DATABASE

#### 删除数据库

用户可以使用**DROP DATABASE**命令删除数据库。这个命令删除了数据库中的系统目

录，并且删除了磁盘上带有数据的数据库目录。用户必须是数据库的owner或者系统

管理员才能删除数据库。当有人连接数据库时，删除操作会失败。删除数据库时请先

连接到其他的数据库。

使用如下命令删除数据库。

openGauss=# DROP DATABASE db\_tpcc ;

DROP DATABASE

### 1.1.2模式

schema又称作模式。通过管理schema，允许多个用户使用同一数据库而不相互干

扰，可以将数据库对象组织成易于管理的逻辑组，同时便于将第三方应用添加到相应

的schema下而不引起冲突。 管理schema包括：创建schema、使用schema、删除

schema、设置schema的搜索路径以及schema的权限控制。

* openGauss包含一个或多个已命名数据库。用户和用户组在openGauss范围内是共享的，但是其数据并不共享。任何与服务器连接的用户都只能访问连接请求里声明的那个数据库。
* 一个数据库可以包含一个或多个已命名的schema，schema又包含表及其他数据库对象，包括数据类型、函数、操作符等。同一对象名可以在不同的schema中使用而不会引起冲突。例如，schema1和schema2都可以包含一个名为mytable的表。
* 和数据库不同，schema不是严格分离的。用户根据其对schema的权限，可以访问所连接数据库的schema中的对象。进行schema权限管理首先需要对数据库的权限控制进行了解。
* 不能创建以PG\_为前缀的schema名，该类schema为数据库系统预留的。
* 在每次创建新用户时，系统会在当前登录的数据库中为新用户创建一个同名Schema。对于其他数据库，若需要同名Schema，则需要用户手动创建。
* 通过未修饰的表名（名称中只含有表名，没有“schema名”）引用表时，系统会通过search\_path（搜索路径）来判断该表是哪个schema下的表。pg\_temp和pg\_catalog始终会作为搜索路径顺序中的前两位，无论二者是否出现在search\_path中，或者出现在search\_path中的任何位置。search\_path（搜索路径）是一个schema名列表，在其中找到的第一个表就是目标表，如果没有找到则报错。（某个表即使存在，如果它的schema不在search\_path中，依然会查找失败）在搜索路径中的第一个schema叫做"当前schema"。它是搜索时查询的第一个schema，同时在没有声明schema名时，新创建的数据库对象会默认存放在该schema下。
* 每个数据库都包含一个pg\_catalog schema，它包含系统表和所有内置数据类型、函数、操作符。pg\_catalog是搜索路径中的一部分，始终在临时表所属的模式后面，并在search\_path中所有模式的前面，即具有第二搜索优先级。这样确保可以搜索到数据库内置对象。如果用户需要使用和系统内置对象重名的自定义对象时，可以在操作自定义对象时带上自己的模式。

#### 创建管理用户及权限schema

– 执行如下命令来创建一个schema。

openGauss=# CREATE SCHEMA myschema;

当结果显示为如下信息，则表示成功创建一个名为myschema的schema。

CREATE SCHEMA

如果需要在模式中创建或者访问对象，其完整的对象名称由模式名称和具体

的对象名称组成。中间由符号“.”隔开。例如：myschema.table。

– 执行如下命令在创建schema时指定owner。

openGauss=# CREATE SCHEMA myschema AUTHORIZATION omm;

当结果显示为如下信息，则表示成功创建一个属于omm用户，名为

myschema的schema。

CREATE SCHEMA

#### 使用schema

在特定schema下创建对象或者访问特定schema下的对象，需要使用有schema修

饰的对象名。该名称包含schema名以及对象名，他们之间用“.”号分开。

– 执行如下命令在myschema下创建mytable表。

openGauss=# CREATE TABLE myschema.mytable(id int, name varchar(20));

CREATE TABLE

如果在数据库中指定对象的位置，就需要使用有schema修饰的对象名称。

– 执行如下命令查询myschema下mytable表的所有数据。

openGauss=# SELECT \* FROM myschema.mytable;

id | name

----+------

(0 rows)

#### 删除schema

– 当schema为空时，即该schema下没有数据库对象，使用DROP SCHEMA命

令进行删除。例如删除名为nullschema的空schema。

openGauss=# DROP SCHEMA IF EXISTS nullschema;

DROP SCHEMA

– 当schema非空时，如果要删除一个schema及其包含的所有对象，需要使用

CASCADE关键字。例如删除myschema及该schema下的所有对象。

openGauss=# DROP SCHEMA myschema CASCADE;

DROP SCHEMA

– 执行如下命令删除用户jack。

openGauss=# DROP USER jack;

DROP ROLE

### 1.1.3管理表空间

**通过使用表空间，管理员可以控制一个数据库安装的磁盘布局。这样有以下优点：**

● 如果初始化数据库所在的分区或者卷空间已满，又不能逻辑上扩展更多空间，可

以在不同的分区上创建和使用表空间，直到系统重新配置空间。

● 表空间允许管理员根据数据库对象的使用模式安排数据位置，从而提高性能。

– 一个频繁使用的索引可以放在性能稳定且运算速度较快的磁盘上，比如一种

固态设备。

– 一个存储归档的数据，很少使用的或者对性能要求不高的表可以存储在一个

运算速度较慢的磁盘上。

● 管理员通过表空间可以设置占用的磁盘空间。用以在和其他数据共用分区的时

候，防止表空间占用相同分区上的其他空间。

● 表空间对应于一个文件系统目录，假定数据库节点数据目录/pg\_location/

mount1/path1是用户拥有读写权限的空目录。

使用表空间配额管理会使性能有30%左右的影响，MAXSIZE指定每个数据库节点

的配额大小，误差范围在500MB以内。请根据实际的情况确认是否需要设置表空

间的最大值。

**openGauss自带了两个表空间：pg\_default和pg\_global。**

● 默认表空间pg\_default：用来存储非共享系统表、用户表、用户表index、临时

表、临时表index、内部临时表的默认表空间。对应存储目录为实例数据目录下的

base目录。

● 共享表空间pg\_global：用来存放共享系统表的表空间。对应存储目录为实例数据

目录下的global目录。

#### 创建表空间

a. 执行如下命令创建用户jack。

openGauss=# CREATE USER jack IDENTIFIED BY 'xxxxxxxxx';

当结果显示为如下信息，则表示创建成功。

CREATE ROLE

b. 执行如下命令创建表空间。

openGauss=# CREATE TABLESPACE fastspace RELATIVE LOCATION 'tablespace/tablespace\_1';

当结果显示为如下信息，则表示创建成功。

CREATE TABLESPACE

其中“fastspace”为新创建的表空间，“数据库节点数据目录/pg\_location/

tablespace/tablespace\_1”是用户拥有读写权限的空目录。

c. 数据库系统管理员执行如下命令将“fastspace”表空间的访问权限赋予数据

用户jack。

openGauss=# GRANT CREATE ON TABLESPACE fastspace TO jack;

当结果显示为如下信息，则表示赋予成功。

GRANT

#### 查询表空间

– 方式1：检查pg\_tablespace系统表。如下命令可查到系统和用户定义的全部

表空间。

openGauss=# SELECT spcname FROM pg\_tablespace;

– 方式2：使用gsql程序的元命令查询表空间。

openGauss=# \db

#### 修改表空间

执行如下命令对表空间fastspace重命名为fspace。

openGauss=# ALTER TABLESPACE fastspace RENAME TO fspace;

ALTER TABLESPACE

#### 删除表空间

– 执行如下命令删除用户jack。

openGauss=# DROP USER jack CASCADE;

DROP ROLE

– 执行如下命令删除表foo和foo2。

openGauss=# DROP TABLE foo;

openGauss=# DROP TABLE foo2;

当结果显示为如下信息，则表示删除成功。

DROP TABLE

– 执行如下命令删除表空间fspace。

openGauss=# DROP TABLESPACE fspace;

DROP TABLESPACE

### 1.1.3创建和管理表

表是建立在数据库中的，在不同的数据库中可以存放相同的表。甚至可以通过使用模

式在同一个数据库中创建相同名称的表。

#### 创建表

执行如下命令创建表

openGauss=# CREATE TABLE customer\_t1

(

customer\_sk integer,

customer\_id char(5),

first\_name char(6),

last\_name char(8)

);

当结果显示为如下信息，则表示创建成功。

CREATE TABLE

其中c\_customer\_sk 、c\_customer\_id、c\_first\_name和c\_last\_name是表的字段名，

integer、char(5)、char(6)和char(8)分别是这四字段名称的类型。

#### 修改表

修改表，包括修改表的定义、重命名表、重命名表中指定的列、重命名表的约束、设置表的所属模式、添加/更新多个列、打开/关闭行访问控制开关。

**语法格式**

ALTER TABLE [ IF EXISTS ] { table\_name [\*] | ONLY table\_name | ONLY ( table\_name ) }

action [, ... ];

--定义一个表。

openGauss=# CREATE TABLE tpcds.warehouse\_t19

(

W\_WAREHOUSE\_SK INTEGER PRIMARY KEY CHECK (W\_WAREHOUSE\_SK > 0),

W\_WAREHOUSE\_ID CHAR(16) NOT NULL,

W\_WAREHOUSE\_NAME VARCHAR(20)

W\_WAREHOUSE\_SQ\_FT INTEGER ,

W\_STREET\_NUMBER CHAR(10) ,

W\_STREET\_NAME VARCHAR(60) ,

W\_STREET\_TYPE CHAR(15) ,

W\_SUITE\_NUMBER CHAR(10) ,

W\_CITY VARCHAR(60) ,

W\_COUNTY VARCHAR(30) ,

W\_STATE CHAR(2) ,

W\_ZIP CHAR(10) ,

W\_COUNTRY VARCHAR(20) ,

W\_GMT\_OFFSET DECIMAL(5,2)

);

--向tpcds.warehouse\_t19表中增加一个varchar列。

openGauss=# ALTER TABLE tpcds.warehouse\_t19 ADD W\_GOODS\_CATEGORY varchar(30);

--在一个操作中改变两个现存字段的类型。

openGauss=# ALTER TABLE tpcds.warehouse\_t19

ALTER COLUMN W\_GOODS\_CATEGORY TYPE varchar(80),

ALTER COLUMN W\_STREET\_NAME TYPE varchar(100);

--此语句与上面语句等效。

openGauss=# ALTER TABLE tpcds.warehouse\_t19 MODIFY (W\_GOODS\_CATEGORY varchar(30), W\_STREET\_NAME varchar(60));

--给一个已存在字段添加非空约束。

openGauss=# ALTER TABLE tpcds.warehouse\_t19 ALTER COLUMN W\_GOODS\_CATEGORY SET NOT NULL;

--移除已存在字段的非空约束。

openGauss=# ALTER TABLE tpcds.warehouse\_t19 ALTER COLUMN W\_GOODS\_CATEGORY DROP NOT NULL;

--从tpcds.warehouse\_t19表中删除一个字段。

openGauss=# ALTER TABLE tpcds.warehouse\_t19 DROP W\_COUNTRY;

#### 删除表

删除指定的表。

**注意事项**

•DROP TABLE会强制删除指定的表，删除表后，依赖该表的索引会被删除，而使用到该表的函数和存储过程将无法执行。删除分区表，会同时删除分区表中的所有分区。

•只有表的所有者或者被授予了表的DROP权限的用户才能执行DROP TABLE，系统管理员默认拥有该权限。

**语法格式**

DROP TABLE [ IF EXISTS ]

{ [schema.]table\_name } [, ...] [ CASCADE | RESTRICT ];

•IF EXISTS

如果指定的表不存在，则发出一个notice而不是抛出一个错误。

•schema

模式名称。

•table\_name

表名称。

•CASCADE | RESTRICT

◾CASCADE：级联删除依赖于表的对象（比如视图）。

◾RESTRICT（缺省项）：如果存在依赖对象，则拒绝删除该表。这个是缺省。

openGauss=# DROP TABLE tpcds.warehouse\_t19

## 实验1.2 数据基本查询

（1）实验目的

掌握SQL程序设计基本规范，熟练运用SQL语言实现数据基本查询，包括单表查询、分组统计查询和连接查询。

（2）实验内容和要求

针对openGauss数据库设计各种单表查询SQL语句、分组统计查询语句；设计单个表针对自身的连接查询，设计多个表的连接查询。理解和掌握SQL查询语句各个子句的特点和作用，按照SQL程序设计规范写出具体的SQL查询语句，并调试通过。

（3）实验重点和难点

实验重点：分组统计查询、单表自身连接查询、多表连接查询。

实验难点：区分元组过滤条件和分组过滤条件；确定连接属性，正确设计连接条件。

### 1.2.1前期准备

教师（教师编号，教师姓名，职称，性别，年龄，入职日期，所在部门编号）

 班级（班级编号，班级名称，班主任）

 课程（课程编号，课程名称，课程类型，学分）

**上述属性对应的编号为：**

 teacher（tec\_id，tec\_name，tec\_job，tec\_sex，tec\_age，tec\_in，depart\_id）

 school\_department（depart\_id，depart\_name，depart\_teacher）

 course（cor\_id，cor\_name，cor\_type，credit）

**创建表**

DROP TABLE IF EXISTS teacher;

CREATE TABLE teacher

(

tec\_id INT PRIMARY KEY,

tec\_name CHAR(20) NOT NULL,

tec\_job CHAR(15),

tec\_sex CHAR(6),

tec\_age INT,

tec\_in DATE NOT NULL,

depart\_id int

);

DROP TABLE IF EXISTS school\_department;

CREATE TABLE school\_department

(

depart\_id INT PRIMARY KEY,

depart\_name NCHAR(30) NOT NULL,

depart\_teacher INT NOT NULL

);

DROP TABLE IF EXISTS course;

CREATE TABLE course

(

cor\_id INT PRIMARY KEY,

cor\_name NCHAR(30) NOT NULL,

cor\_type NCHAR(20),

credit numeric

);

**插入数据**

INSERT INTO teacher(tec\_id,tec\_name,tec\_job,tec\_sex,tec\_age,tec\_in,depart\_id) VALUES (1,'张一','讲师','男',35,'2009-07-01',1);

INSERT INTO teacher(tec\_id,tec\_name,tec\_job,tec\_sex,tec\_age,tec\_in,depart\_id) VALUES (2,'张二','教授','女',35,'2009-07-01',1);

INSERT INTO teacher(tec\_id,tec\_name,tec\_job,tec\_sex,tec\_age,tec\_in,depart\_id) VALUES (3,'张三','讲师','男',35,'2009-07-01',1);

INSERT INTO teacher(tec\_id,tec\_name,tec\_job,tec\_sex,tec\_age,tec\_in,depart\_id) VALUES (4,'张四','教授','女',35,'2009-07-01',2);

INSERT INTO teacher(tec\_id,tec\_name,tec\_job,tec\_sex,tec\_age,tec\_in,depart\_id) VALUES (5,'张五','讲师','男',35,'2009-07-01',2);

INSERT INTO teacher(tec\_id,tec\_name,tec\_job,tec\_sex,tec\_age,tec\_in,depart\_id) VALUES (6,'张六','教授','男',35,'2009-07-01',3);

INSERT INTO teacher(tec\_id,tec\_name,tec\_job,tec\_sex,tec\_age,tec\_in,depart\_id) VALUES (7,'张七','讲师','女',35,'2009-07-01',3);

INSERT INTO teacher(tec\_id,tec\_name,tec\_job,tec\_sex,tec\_age,tec\_in,depart\_id) VALUES (8,'张八','教授','男',35,'2009-07-01',4);

INSERT INTO teacher(tec\_id,tec\_name,tec\_job,tec\_sex,tec\_age,tec\_in,depart\_id) VALUES (9,'张九','讲师','男',35,'2009-07-01',4);

INSERT INTO teacher(tec\_id,tec\_name,tec\_job,tec\_sex,tec\_age,tec\_in,depart\_id) VALUES (10,'李一','讲师','男',35,'2009-07-01',4);

INSERT INTO teacher(tec\_id,tec\_name,tec\_job,tec\_sex,tec\_age,tec\_in,depart\_id) VALUES (11,'李二','教授','女',35,'2009-07-01',5);

INSERT INTO teacher(tec\_id,tec\_name,tec\_job,tec\_sex,tec\_age,tec\_in,depart\_id) VALUES (12,'李三','讲师','男',35,'2009-07-01',5);

INSERT INTO teacher(tec\_id,tec\_name,tec\_job,tec\_sex,tec\_age,tec\_in,depart\_id) VALUES (13,'李四','教授','男',35,'2009-07-01',6);

INSERT INTO teacher(tec\_id,tec\_name,tec\_job,tec\_sex,tec\_age,tec\_in,depart\_id) VALUES (14,'李五','讲师','女',35,'2009-07-01',6);

INSERT INTO teacher(tec\_id,tec\_name,tec\_job,tec\_sex,tec\_age,tec\_in,depart\_id) VALUES (15,'李六','教授','女',35,'2009-07-01',7);

INSERT INTO teacher(tec\_id,tec\_name,tec\_job,tec\_sex,tec\_age,tec\_in,depart\_id) VALUES (16,'李七','讲师','男',35,'2009-07-01',7);

INSERT INTO teacher(tec\_id,tec\_name,tec\_job,tec\_sex,tec\_age,tec\_in,depart\_id) VALUES (17,'李八','教授','女',35,'2009-07-01',7);

INSERT INTO teacher(tec\_id,tec\_name,tec\_job,tec\_sex,tec\_age,tec\_in,depart\_id) VALUES (18,'李九','讲师','女',35,'2009-07-01',7);

INSERT INTO teacher(tec\_id,tec\_name,tec\_job,tec\_sex,tec\_age,tec\_in,depart\_id) VALUES (19,'王一','讲师','男',35,'2009-07-01',7);

INSERT INTO teacher(tec\_id,tec\_name,tec\_job,tec\_sex,tec\_age,tec\_in,depart\_id) VALUES (20,'王二','讲师','女',35,'2009-07-01',7);

INSERT INTO school\_department(depart\_id,depart\_name,depart\_teacher) VALUES (1,'计算机学院',2);

INSERT INTO school\_department(depart\_id,depart\_name,depart\_teacher) VALUES (2,'自动化学院',4);

INSERT INTO school\_department(depart\_id,depart\_name,depart\_teacher) VALUES (3,'航空宇航学院',6);

INSERT INTO school\_department(depart\_id,depart\_name,depart\_teacher) VALUES (4,'艺术学院',8);

INSERT INTO school\_department(depart\_id,depart\_name,depart\_teacher) VALUES (5,'理学院',11);

INSERT INTO school\_department(depart\_id,depart\_name,depart\_teacher) VALUES (6,'人工智能学院',13);

INSERT INTO school\_department(depart\_id,depart\_name,depart\_teacher) VALUES (7,'工学院',15);

INSERT INTO school\_department(depart\_id,depart\_name,depart\_teacher) VALUES (8,'管理学院',17);

INSERT INTO course(cor\_id,cor\_name,cor\_type,credit) VALUES (1,'数据库系统概论','必修',3);

INSERT INTO course(cor\_id,cor\_name,cor\_type,credit) VALUES (2,'艺术设计概论','选修',1);

INSERT INTO course(cor\_id,cor\_name,cor\_type,credit) VALUES (3,'力学制图','必修',4);

INSERT INTO course(cor\_id,cor\_name,cor\_type,credit) VALUES (4,'飞行器设计历史','选修',1);

INSERT INTO course(cor\_id,cor\_name,cor\_type,credit) VALUES (5,'马克思主义','必修',2);

INSERT INTO course(cor\_id,cor\_name,cor\_type,credit) VALUES (6,'大学历史','必修',2);

INSERT INTO course(cor\_id,cor\_name,cor\_type,credit) VALUES (7,'人力资源管理理论','必修',2.5);

INSERT INTO course(cor\_id,cor\_name,cor\_type,credit) VALUES (8,'线性代数','必修',4);

INSERT INTO course(cor\_id,cor\_name,cor\_type,credit) VALUES (9,'JAVA程序设计','必修',3);

INSERT INTO course(cor\_id,cor\_name,cor\_type,credit) VALUES (10,'操作系统','必修',4);

INSERT INTO course(cor\_id,cor\_name,cor\_type,credit) VALUES (11,'计算机组成原理','必修',3);

INSERT INTO course(cor\_id,cor\_name,cor\_type,credit) VALUES (12,'自动化设计理论','必修',2);

INSERT INTO course(cor\_id,cor\_name,cor\_type,credit) VALUES (13,'情绪表演','必修',2.5);

INSERT INTO course(cor\_id,cor\_name,cor\_type,credit) VALUES (14,'茶学历史','选修',1);

INSERT INTO course(cor\_id,cor\_name,cor\_type,credit) VALUES (15,'艺术论','必修',1.5);

INSERT INTO course(cor\_id,cor\_name,cor\_type,credit) VALUES (16,'机器学习','必修',3);

INSERT INTO course(cor\_id,cor\_name,cor\_type,credit) VALUES (17,'数据挖掘','选修',2);

INSERT INTO course(cor\_id,cor\_name,cor\_type,credit) VALUES (18,'图像识别','必修',3);

INSERT INTO course(cor\_id,cor\_name,cor\_type,credit) VALUES (19,'解剖学','必修',4);

INSERT INTO course(cor\_id,cor\_name,cor\_type,credit) VALUES (20,'3D max','选修',2);

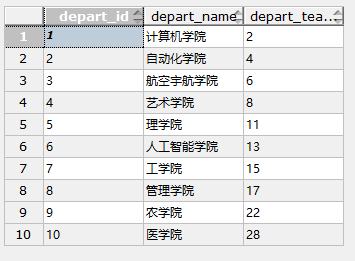
### 1.2.2简单查询

**通过Data Studio进行查询**

**示例1：**查看school\_department表的所有列。

**SELECT** **\*** **FROM** school\_department

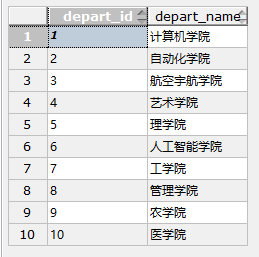
结果如下：



**示例2：**查看school\_department表中的班级编号，班级名称。

**SELECT** depart\_id,depart\_name **FROM** school\_department

结果如下：



### 1.2.3分组统计查询

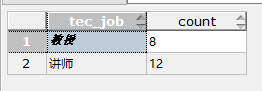
**示例1：**查询员工中的教授和讲师的数量，按tec\_job分组

**SELECT** tec\_job,COUNT(tec\_id)

**FROM** teacher

**GROUP** **BY** tec\_job **ORDER** **BY** tec\_job;

执行结果如下：



**示例2：**统计course表中credit的总和

SELECT SUM(salary) FROM hr.staffs;

结果如下图所示：



**示例3：**查询课程学分大于2的课程的课程名字和学分

**SELECT** cor\_name,credit

**FROM** course

**GROUP** **BY** cor\_name,credit **HAVING** credit**>**2;

执行结果如下图所示：



### 1.2.4连接查询

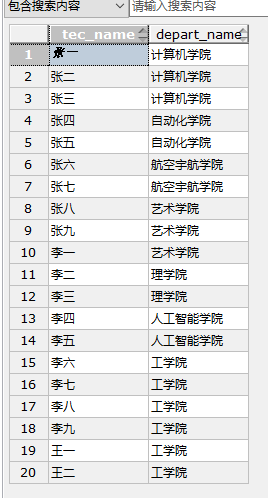
**示例1：**查询教师姓名和所属学院

**SELECT** tec\_name,depart\_name

**FROM** school\_department

**JOIN** teacher **ON** (teacher.depart\_id **=**school\_department.depart\_id);

执行结果如下图：



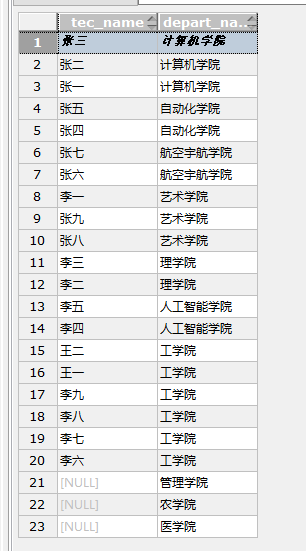
**示例2**：示例1的左外连接查询

**SELECT** tec\_name,depart\_name

**FROM** school\_department

**LEFT** **JOIN** teacher **ON** (teacher.depart\_id**=**school\_department.depart\_id);

执行结果如下图：



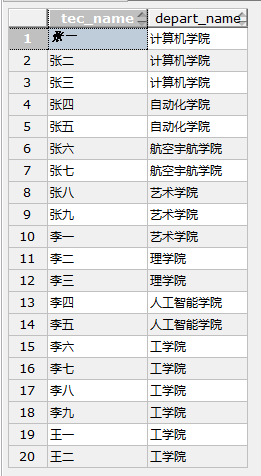
**示例3：**示例1的右外连接查询

**SELECT** tec\_name,depart\_name

**FROM** school\_department

**RIGHT** **JOIN** teacher **ON**(teacher.depart\_id**=**school\_department.depart\_id);

执行结果如下：



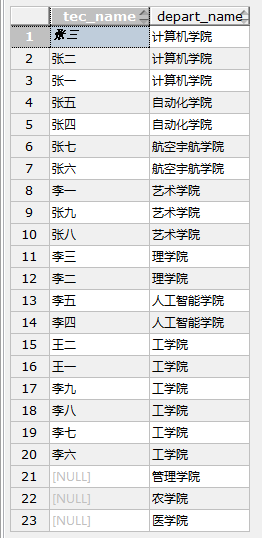
**示例4：**全外链接。

**SELECT** tec\_name,depart\_name

**FROM** school\_department

**FULL** **JOIN** teacher **ON** (teacher.depart\_id**=**school\_department.depart\_id);

执行结果如下图所示：



## 实验1.3 数据高级查询

（1）实验目的

掌握SQL嵌套查询和集合查询等各种高级查询的设计方法等。

（2）实验内容和要求

针对openGauss数据库，正确分析用户查询要求，设计各种嵌套查询和集合查询。

（3）实验重点和难点

实验重点：嵌套查询。

实验难点：相关子查询、多层EXIST嵌套查询

### 1.3.1 嵌套查询

**定义：**

1 .指在一个外层查询中包含有另一个内层查询。其中外层查询称为主查询，内层查询称为子查询。

2 .SQL允许多层嵌套，由内而外地进行分析，子查询的结果作为主查询的查询条件

3 .子查询中一般不使用order by子句，只能对最终查询结果进行排序

**相关数据**

雇员（编号，姓名，工资，所属部门编号）

**上述属性对应的编号为：**

 employee( emp\_id ， emp\_name, salary, deptid);

DROP TABLE IF EXISTS employee;

create table employee(

emp\_id int primary key ,

emp\_name varchar(50),

salary bigint,

deptid int);

insert into employee values(1,'zs',1000,1);

insert into employee values(2,'ls',800,1);

insert into employee values(3,'ww',950,1);

insert into employee values(4,'z2',900,1);

insert into employee values(5,'z3',1000,2);

insert into employee values(6,'z4',900,2);

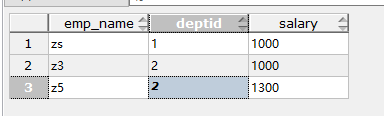
insert into employee values(8,'z5',1300,2);

**示例**：查找每个部门中高出部门平均工资的人员

**SELECT** emp\_name,deptid, salary

**FROM** employee **WHERE** salary **>=**(**SELECT** avg(salary) **FROM** employee )

执行结果如下：



### 1.3.2 集合查询

UNION:并  
INTERSECT:交  
EXCEPT:差  
都是用于两个查询之间。

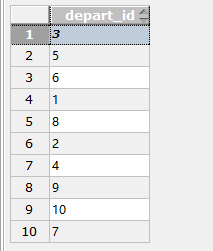
**例子：**

**示例1：**查询school\_department表和teacher表中的depart\_id并且做并集

**select** depart\_id **from** school\_department

**union** **select** depart\_id **from** teacher;

执行结果如下：

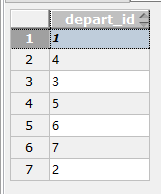


**示例2**：查询school\_department表和teacher表中的depart\_id并且做交集

**select** depart\_id **from** school\_department

**INTERSECT** **select** depart\_id **from** teacher;

执行结果如下：

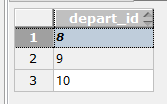


例子3：查询school\_department表和teacher表中的depart\_id并且做差集

**select** depart\_id **from** school\_department

**EXCEPT** **select** depart\_id **from** teacher;

执行结果：



**事务控制 COMMIT**

存储过程本身就处于一个事务中，开始调用最外围存储过程时会自动开启一个事务，在调用结束时自动提交或者发生异常时回滚。除了系统自动的事务控制外，也可以使用COMMIT/ROLLBACK来控制存储过程中的事务。在存储过程中调用COMMIT/ROLLBACK命令，将提交/回滚当前事务并自动开启一个新的事务，后续的所有操作都会在此新事务中运行。

保存点SAVEPOINT是事务中的一个特殊记号，它允许将那些在它建立后执行的命令全部回滚，把事务的状态恢复到保存点所在的时刻。存储过程中允许使用保存点来进行事务管理，当前支持保存点的创建、回滚和释放操作。存储过程中使用回滚保存点只是回退当前事务的修改，而不会改变存储过程的执行流程，也不会回退存储过程中的局部变量值等。

## 实验1.4 数据更新

（1）实验目的

熟悉数据库的数据更新操作，能够使用SQL语句对数据库进行数据的插入、修改、删除操作。

（2）实验内容和要求

针对openGauss数据库设计单元组插入、批量数据插入、修改数据和删除数据等SQL语句。理解和掌握INSERT、UPDATE和DELETE语法结构的各个组成成分，结合嵌套SQL子查询，分别设计几种不同形式的插入、修改和删除数据的语句，并调试成功。

### 1.4.1向表中插入数据

在创建一个表后，表中并没有数据，在使用这个表之前，需要向表中插入数据。本小

节介绍如何使用**INSERT**命令插入一行或多行数据，及从指定表插入数据。

执行如下命令建立示例中需要使用的表table1、table2。

openGauss=# CREATE TABLE table1(id int, a char(6), b varchar(6),c varchar(6));

openGauss=# CREATE TABLE table2(id int, a char(20), b varchar(20),c varchar(20));

#### 向表中插入一行：

数据值是按照这些字段在表中出现的顺序列出的，并且用逗号分隔。通常数据值

是文本（常量），但也允许使用标量表达式。

openGauss=# INSERT INTO customer\_t1(c\_customer\_sk, c\_customer\_id, c\_first\_name) VALUES (3769,

'hello', 'Grace');

如果用户已经知道表中字段的顺序，也可无需列出表中的字段。例如以下命令与

上面的命令效果相同。

openGauss=# INSERT INTO customer\_t1 VALUES (3769, 'hello', 'Grace');

如果用户不知道所有字段的数值，可以忽略其中的一些。没有数值的字段将被填

充为字段的缺省值。例如：

openGauss=# INSERT INTO customer\_t1 (c\_customer\_sk, c\_first\_name) VALUES (3769, 'Grace');

openGauss=# INSERT INTO customer\_t1 VALUES (3769, 'hello');

用户也可以对独立的字段或者整个行明确缺省值：

openGauss=# INSERT INTO customer\_t1 (c\_customer\_sk, c\_customer\_id, c\_first\_name) VALUES

(3769, 'hello', DEFAULT);

openGauss=# INSERT INTO customer\_t1 DEFAULT VALUES;

#### 在表中插入多行

openGauss=# INSERT INTO customer\_t1 (c\_customer\_sk, c\_customer\_id, c\_first\_name) VALUES

(6885, 'maps', 'Joes'),

(4321, 'tpcds', 'Lily'),

(9527, 'world', 'James');

如果需要向表中插入多条数据，除此命令外，也可以多次执行插入一行数据命令

实现。但是建议使用此命令可以提升效率。

#### 从指定表插入数据到当前表

例如在数据库中创建了一个表customer\_t1的备

份表customer\_t2，现在需要将表customer\_t1中的数据插入到表customer\_t2

中，则可以执行如下命令。

openGauss=# CREATE TABLE customer\_t2

(

c\_customer\_sk integer,

c\_customer\_id char(5),

c\_first\_name char(6),

c\_last\_name char(8)

);

openGauss=# INSERT INTO customer\_t2 SELECT \* FROM customer\_t1;

**从指定表插入数据到当前表时，若指定表与当前表对应的字段数据类型之间不存在隐式转**

**换，则这两种数据类型必须相同。**

#### 删除备份表

openGauss=# DROP TABLE customer\_t2 CASCADE;

**在删除表的时候，若当前需删除的表与其他表有依赖关系，需先删除关联的表，然后再删**

**除当前表。**

### 1.4.2更新表中数据

指定的部分行。还可以独立更新每个字段，而其他字段则不受影响。

使用**UPDATE**命令更新现有行，需要提供以下三种信息：

1. 表的名称和要更新的字段名
2. 字段的新值
3. 要更新哪些行

SQL通常不会为数据行提供唯一标识，因此无法直接声明需要更新哪一行。但是可以

通过声明一个被更新的行必须满足的条件。只有在表里存在主键的时候，才可以通过

主键指定一个独立的行。

需要将表customer\_t1中c\_customer\_sk为9527的地域重新定义为9876：

openGauss=# UPDATE customer\_t1 SET c\_customer\_sk = 9876 WHERE c\_customer\_sk = 9527;

这里的表名称也可以使用模式名修饰，否则会从默认的模式路径找到这个表。SET后面

紧跟字段和新的字段值。新的字段值不仅可以是常量，也可以是变量表达式。

比如，把所有c\_customer\_sk的值增加100：

openGauss=# UPDATE customer\_t1 SET c\_customer\_sk = c\_customer\_sk + 100;

在这里省略了WHERE子句，表示表中的所有行都要被更新。如果出现了WHERE子

句，那么只有匹配其条件的行才会被更新。

在SET子句中的等号是一个赋值，而在WHERE子句中的等号是比较。WHERE条件不一

定是相等测试，许多其他的操作符也可以使用。

用户可以在一个UPDATE命令中更新更多的字段，方法是在SET子句中列出更多赋值，

比如：

openGauss=# UPDATE customer\_t1 SET c\_customer\_id = 'Admin', c\_first\_name = 'Local' WHERE

c\_customer\_sk = 4421;

批量更新或删除数据后，会在数据文件中产生大量的删除标记，查询过程中标记删除

的数据也是需要扫描的。故多次批量更新/删除后，标记删除的数据量过大会严重影响

查询的性能。建议在批量更新/删除业务会反复执行的场景下，定期执行VACUUM

FULL以保持查询性能。

### 1.4.3查看数据

● 使用系统表pg\_tables查询数据库所有表的信息。

openGauss=# SELECT \* FROM pg\_tables;

● 使用gsql的\d+命令查询表的属性。

openGauss=# \d+ customer\_t1;

● 执行如下命令查询表customer\_t1的数据量。

openGauss=# SELECT count(\*) FROM customer\_t1;

● 执行如下命令查询表customer\_t1的所有数据。

openGauss=# SELECT \* FROM customer\_t1;

● 执行如下命令只查询字段c\_customer\_sk的数据。

openGauss=# SELECT c\_customer\_sk FROM customer\_t1;

● 执行如下命令过滤字段c\_customer\_sk的重复数据。

openGauss=# SELECT DISTINCT( c\_customer\_sk ) FROM customer\_t1;

● 执行如下命令查询字段c\_customer\_sk为3869的所有数据。

openGauss=# SELECT \* FROM customer\_t1 WHERE c\_customer\_sk = 3869;

● 执行如下命令按照字段c\_customer\_sk进行排序。

openGauss=# SELECT \* FROM customer\_t1 ORDER BY c\_customer\_sk;

### 1.4.4删除表中数据

在使用表的过程中，可能会需要删除已过期的数据，删除数据必须从表中整行的删除。

SQL不能直接访问独立的行，只能通过声明被删除行匹配的条件进行。如果表中有一个主键，用户可以指定准确的行。用户可以删除匹配条件的一组行或者一次删除表中的所有行。

使用DELETE命令删除行，如果删除表customer\_t1中所有c\_customer\_sk为3869的记录：

openGauss=# DELETE FROM customer\_t1 WHERE c\_customer\_sk = 3869;

如果执行如下命令之一，会删除表中所有的行。

openGauss=# DELETE FROM customer\_t1;

或

openGauss=# TRUNCATE TABLE customer\_t1;

说明

全表删除的场景下，建议使用truncate，不建议使用delete。

删除创建的表：

openGauss=# DROP TABLE customer\_t1;

## 实验1.5 视图

（1）实验目的

熟悉SQL语言有关视图的操作，能够熟练使用SQL语句来创建需要的视图，定义数据库外模式，并能使用所创建的视图实现数据管理。

（2）实验内容和要求

针对给定的openGauss数据库模式，以及相应的应用需求，创建视图。理解和掌握视图消解执行原理。

（3）实验重点和难点

实验重点：创建视图。

实验难点：普通视图和物化视图的区别。

### 1.5.1创建视图

创建普通视图MyView。

postgres=# CREATE OR REPLACE VIEW MyView AS SELECT \* FROM tpcds.web\_returns\_p2 WHERE trunc(ca\_address\_sk) > 10000;

CREATE VIEW

创建物化视图MV\_MyView。

postgres=# CREATE MATERIALIZED VIEW MV\_MyView AS SELECT \* FROM tpcds.web\_returns\_p2 WHERE trunc(ca\_address\_sk) > 5000;

SELECT 2

物化视图使用场景：报表统计、大表统计等，定期固化数据快照，避免对多表重复跑相同的查询。

物化视图使用注意事项：  
不可以在临时表或全局临时表上创建。  
当基表数据发生变化时，需要使用刷新命令保持物化视图与基表同步。

### 1.5.2管理视图

#### 查询普通视图

查询MyView视图。

postgres=# SELECT \* FROM MyView;

#### 查看普通视图的具体信息

查询MyView视图的详细信息。

postgres=# \d+ Myview

View ＂public.myview＂

Column | Type | Modifiers | Storage | Description

------------------+-----------------------+-----------+----------+-------------

ca\_address\_sk | integer | | plain |

ca\_address\_id | character(16) | | extended |

ca\_street\_number | character(10) | | extended |

ca\_street\_name | character varying(60) | | extended |

ca\_street\_type | character(15) | | extended |

ca\_suite\_number | character(10) | | extended |

ca\_city | character varying(60) | | extended |

ca\_county | character varying(30) | | extended |

ca\_state | character(2) | | extended |

ca\_zip | character(10) | | extended |

ca\_country | character varying(20) | | extended |

ca\_gmt\_offset | numeric(5,2) | | main |

ca\_location\_type | character(20) | | extended |

View definition:

SELECT \*

FROM tpcds.web\_returns\_p2

WHERE trunc(web\_returns\_p2.ca\_address\_sk::double precision) > 10000::double prec

ision;

#### 查询物化视图

查询MV\_MyView视图。

postgres=# SELECT \* FROM MV\_MyView;

ca\_address\_sk | ca\_address\_id | ca\_street\_number | ca\_street\_name | ca\_street\_type | ca\_suite\_number | ca\_city | ca\_county | ca\_state | ca\_zip | ca\_country | ca\_gm

t\_offset | ca\_location\_type

---------------+------------------+------------------+----------------+-----------------+-----------------+---------+-----------+----------+------------+------------+------

---------+----------------------

5050 | c | 300 | c | c | c | c | c | c | c | c |

1.20 | c

14888 | d | 400 | d | d | d | d | d | d | d | d |

1.50 | d

(2 rows)

#### 查看物化视图的具体信息

查询MyView视图的详细信息。

postgres=# \d+ MV\_MyView

Materialized view ＂public.mv\_myview＂

Column | Type | Modifiers | Storage | Stats target | Description

------------------+-----------------------+-----------+----------+--------------+-------------

ca\_address\_sk | integer | | plain | |

ca\_address\_id | character(16) | | extended | |

ca\_street\_number | character(10) | | extended | |

ca\_street\_name | character varying(60) | | extended | |

ca\_street\_type | character(15) | | extended | |

ca\_suite\_number | character(10) | | extended | |

ca\_city | character varying(60) | | extended | |

ca\_county | character varying(30) | | extended | |

ca\_state | character(2) | | extended | |

ca\_zip | character(10) | | extended | |

ca\_country | character varying(20) | | extended | |

ca\_gmt\_offset | numeric(5,2) | | main | |

ca\_location\_type | character(20) | | extended | |

View definition:

SELECT \*

FROM tpcds.web\_returns\_p2

WHERE trunc(web\_returns\_p2.ca\_address\_sk::double precision) > 5000::double precision;

Replica Identity: NOTHING

Has OIDs: no

给基表新增两条记录，然后刷新物化视图。

postgres=# insert into tpcds.web\_returns\_p2 values (7050, 'c', 300, 'c', 'c', 'c', 'c', 'c', 'c', 'c', 'c', 1.2, 'c'), (8888, 'd', 400, 'd', 'd', 'd', 'd', 'd', 'd', 'd', 'd', 1.5, 'd');

INSERT 0 2

#### 刷新物化图

由于基表数据变更过，可以执行如下命令刷新物化视图MV\_MyView。

postgres=# REFRESH MATERIALIZED VIEW MV\_MyView;

REFRESH MATERIALIZED VIEW

再查看物化视图MV\_MyView，发现多了两条记录。

postgres=# SELECT \* FROM MV\_MyView;

ca\_address\_sk | ca\_address\_id | ca\_street\_number | ca\_street\_name | ca\_street\_type | ca\_suite\_number | ca\_city | ca\_county | ca\_state | ca\_zip | ca\_country | ca\_gm

t\_offset | ca\_location\_type

---------------+------------------+------------------+----------------+-----------------+-----------------+---------+-----------+----------+------------+------------+------

---------+----------------------

5050 | c | 300 | c | c | c | c | c | c | c | c |

1.20 | c

7050 | c | 300 | c | c | c | c | c | c | c | c |

1.20 | c

8888 | d | 400 | d | d | d | d | d | d | d | d |

1.50 | d

14888 | d | 400 | d | d | d | d | d | d | d | d |

1.50 | d

(4 rows)

### 1.5.3删除视图

删除视图。

postgres=# DROP VIEW MyView;

DROP VIEW

postgres=# DROP MATERIALIZED VIEW MV\_MyView;

DROP MATERIALIZED VIEW

## 实验1.6 索引

（1）实验目的

掌握索引设计原则和技巧，能够创建合适的索引以提高数据库查询、统计分析效率。

（2）实验内容和要求

针对给定的openGauss数据库模式和具体应用需求，创建唯一索引、函数索引、复合索引等；修改索引；删除索引。设计相应的SQL查询验证索引有效性。分析SQL查询是否使用了所创建的索引，并能够分析其原因，执行SQL查询并估算索引提高查询效率的百分比。要求实验数据集达到10万条记录以上的数据量，以便验证索引效果。

（3）实验重点和难点

实验重点：创建索引。

实验难点：设计SQL查询验证索引有效性。

### 1.6.1准备工作

#### 创建表空间

postgres=# CREATE TABLESPACE example1 RELATIVE LOCATION 'tablespace1/tablespace\_1';

postgres=# CREATE TABLESPACE example2 RELATIVE LOCATION 'tablespace2/tablespace\_2';

postgres=# CREATE TABLESPACE example3 RELATIVE LOCATION 'tablespace3/tablespace\_3';

postgres=# CREATE TABLESPACE example4 RELATIVE LOCATION 'tablespace4/tablespace\_4';

当结果显示为如下信息，则表示创建成功。

CREATE TABLESPACE

#### 创建表并在表中插入数据。

postgres=# CREATE TABLE tpcds.web\_returns\_p2

(

ca\_address\_sk integer NOT NULL ,

ca\_address\_id character(16) NOT NULL ,

ca\_street\_number character(10) ,

ca\_street\_name character varying(60) ,

ca\_street\_type character(15) ,

ca\_suite\_number character(10) ,

ca\_city character varying(60) ,

ca\_county character varying(30) ,

ca\_state character(2) ,

ca\_zip character(10) ,

ca\_country character varying(20) ,

ca\_gmt\_offset numeric(5,2) ,

ca\_location\_type character(20)

)

TABLESPACE example1

PARTITION BY RANGE (ca\_address\_sk)

(

PARTITION P1 VALUES LESS THAN(5000),

PARTITION P2 VALUES LESS THAN(10000),

PARTITION P3 VALUES LESS THAN(15000),

PARTITION P4 VALUES LESS THAN(20000),

PARTITION P5 VALUES LESS THAN(25000),

PARTITION P6 VALUES LESS THAN(30000),

PARTITION P7 VALUES LESS THAN(40000),

PARTITION P8 VALUES LESS THAN(MAXVALUE) TABLESPACE example2

)

ENABLE ROW MOVEMENT;

显示如下，表示创建成功。

CREATE TABLE

插入数据。

postgres=# insert into tpcds.web\_returns\_p2 values(1, 'a', 1, 'a', 'a', 'a', 'a', 'a', 'a', 'a', 'a', 1.0, 'a'), (2, 'b', 2, 'b', 'b', 'b', 'b', 'b', 'b', 'b', 'b', 1.1, 'b'), (5050, 'c', 300, 'c', 'c', 'c', 'c', 'c', 'c', 'c', 'c', 1.2, 'c'), (14888, 'd', 400, 'd', 'd', 'd', 'd', 'd', 'd', 'd', 'd', 1.5, 'd');

### 1.6.2建索引

创建分区表索引tpcds\_web\_returns\_p2\_index1，不指定索引分区的名称。

postgres=# CREATE INDEX tpcds\_web\_returns\_p2\_index1 ON tpcds.web\_returns\_p2 (ca\_address\_id) LOCAL;

当结果显示为如下信息，则表示创建成功。

CREATE INDEX

创建分区索引tpcds\_web\_returns\_p2\_index2，并指定索引分区的名称。

postgres=# CREATE INDEX tpcds\_web\_returns\_p2\_index2 ON tpcds.web\_returns\_p2 (ca\_address\_sk) LOCAL

(

PARTITION web\_returns\_p2\_P1\_index,

PARTITION web\_returns\_p2\_P2\_index TABLESPACE example3,

PARTITION web\_returns\_p2\_P3\_index TABLESPACE example4,

PARTITION web\_returns\_p2\_P4\_index,

PARTITION web\_returns\_p2\_P5\_index,

PARTITION web\_returns\_p2\_P6\_index,

PARTITION web\_returns\_p2\_P7\_index,

PARTITION web\_returns\_p2\_P8\_index

) TABLESPACE example2;

当结果显示为如下信息，则表示创建成功。

CREATE INDEX

### 1.6.3管理索引

#### 修改索引分区的表空间

修改索引分区web\_returns\_p2\_P2\_index的表空间为example1。

postgres=# ALTER INDEX tpcds.tpcds\_web\_returns\_p2\_index2 MOVE PARTITION web\_returns\_p2\_P2\_index TABLESPACE example1;

当结果显示为如下信息，则表示修改成功。

ALTER INDEX

修改索引分区web\_returns\_p2\_P3\_index的表空间为example2。

postgres=# ALTER INDEX tpcds.tpcds\_web\_returns\_p2\_index2 MOVE PARTITION web\_returns\_p2\_P3\_index TABLESPACE example2;

当结果显示为如下信息，则表示修改成功。

ALTER INDEX

#### 重命名索引分区

对索引分区web\_returns\_p2\_P8\_index重命名web\_returns\_p2\_P8\_index\_new。

postgres=# ALTER INDEX tpcds.tpcds\_web\_returns\_p2\_index2 RENAME PARTITION web\_returns\_p2\_P8\_index TO web\_returns\_p2\_P8\_index\_new;

当结果显示为如下信息，则表示重命名成功。

ALTER INDEX

#### 查询索引

查询系统和用户定义的所有索引。

postgres=# SELECT RELNAME FROM PG\_CLASS WHERE RELKIND='i';

#### 删除索引

postgres=# DROP INDEX tpcds.tpcds\_web\_returns\_p2\_index1;

postgres=# DROP INDEX tpcds.tpcds\_web\_returns\_p2\_index2;

当结果显示为如下信息，则表示删除成功。

DROP INDEX

### 1.6.4索引创建举例

openGauss支持4种创建索引的方式：唯一索引、多字段索引、部分索引、表达式索引。

**创建一个普通表。**

postgres=# CREATE TABLE tpcds.customer\_address\_bak AS TABLE tpcds.web\_returns\_p2;

**创建普通索引。**

如果对于tpcds.customer\_address\_bak表，需要经常进行以下查询。

postgres=# SELECT ca\_address\_sk FROM tpcds.customer\_address\_bak WHERE ca\_address\_sk=14888;

**使用以下命令创建索引。**

postgres=# CREATE INDEX index\_wr\_returned\_date\_sk ON tpcds.customer\_address\_bak (ca\_address\_sk);

CREATE INDEX

**创建多字段索引**

假如用户需要经常查询表tpcds.customer\_address\_bak中ca\_address\_sk是5050，且ca\_street\_number小于1000的记录，使用以下命令进行查询。

postgres=# SELECT ca\_address\_sk,ca\_address\_id FROM tpcds.customer\_address\_bak WHERE ca\_address\_sk = 5050 AND ca\_street\_number < 1000;

使用以下命令在字段ca\_address\_sk和ca\_street\_number上定义一个多字段索引。

postgres=# CREATE INDEX more\_column\_index ON tpcds.customer\_address\_bak(ca\_address\_sk ,ca\_street\_number);

CREATE INDEX

**创建部分索引**

如果只需要查询ca\_address\_sk为5050的记录，可以创建部分索引来提升查询效率。

postgres=# CREATE INDEX part\_index ON tpcds.customer\_address\_bak(ca\_address\_sk) WHERE ca\_address\_sk = 5050;

CREATE INDEX

**创建表达式索引**

假如经常需要查询ca\_street\_number小于1000的信息，执行如下命令进行查询。

postgres=# SELECT \* FROM tpcds.customer\_address\_bak WHERE trunc(ca\_street\_number) < 1000;

可以为上面的查询创建表达式索引：

postgres=# CREATE INDEX para\_index ON tpcds.customer\_address\_bak (trunc(ca\_street\_number));

CREATE INDEX

**删除tpcds.customer\_address\_bak表。**

postgres=# DROP TABLE tpcds.customer\_address\_bak;

DROP TABLE