《数据库》

openGauss数据库开发指导手册



华为技术有限公司

目录

[前 言 3](#_Toc81318117)

[简介 3](#_Toc81318118)

[内容描述 3](#_Toc81318119)

[前置条件 3](#_Toc81318120)

[实验环境说明 3](#_Toc81318121)

[实验概览 4](#_Toc81318122)

[1 数据库开发实验 5](#_Toc81318123)

[1.1 创建和管理用户、表空间和数据库 5](#_Toc81318124)

[1.1.1 创建和管理用户 5](#_Toc81318125)

[1.1.2 创建和管理表空间 6](#_Toc81318126)

[1.1.3 创建和管理数据库 8](#_Toc81318127)

[1.2 创建和管理表 9](#_Toc81318128)

[1.2.1 创建表 9](#_Toc81318129)

[1.2.2 向表中插入数据 9](#_Toc81318130)

[1.2.3 更新表中数据 10](#_Toc81318131)

[1.2.4 查看数据 11](#_Toc81318132)

[1.2.5 删除表中数据 12](#_Toc81318133)

[1.3 创建和管理其他数据库对象操作 12](#_Toc81318134)

[1.3.1 创建和管理schema 12](#_Toc81318135)

[1.3.2 创建和管理分区表 14](#_Toc81318136)

[1.3.3 创建和管理索引 16](#_Toc81318137)

[1.3.4 创建和管理视图 20](#_Toc81318138)

[1.3.5 创建和管理序列 23](#_Toc81318139)

[1.3.6 创建和管理存储过程 24](#_Toc81318140)

[1.3.7 创建和管理全局临时表 25](#_Toc81318141)

[1.4 学校数据模型 27](#_Toc81318142)

[1.4.1 关于本实验 27](#_Toc81318143)

[1.4.2 关系模型 27](#_Toc81318144)

[1.4.3 E-R图 28](#_Toc81318145)

[1.5 学校数据模型表操作 28](#_Toc81318146)

[1.5.1 创建表 28](#_Toc81318147)

[1.5.2 插入表数据 30](#_Toc81318148)

[1.5.3 查询数据 35](#_Toc81318149)

[1.5.4 修改和删除数据 36](#_Toc81318150)

[附录一：openGauss数据库基本操作 38](#_Toc81318151)

# 前 言

## 简介

本实验指导书主要目的是为了让读者熟悉并掌握openGauss数据库的基本操作，并通过案例强化学习。

## 内容描述

本实验指导书先介绍创建管理用户、表及其它数据库对象等openGauss数据库的基本操作，再从学校数据模型的案例入手，进行模拟演练。

## 前置条件

openGauss数据库开发需要掌握openGauss数据库的基本操作和SQL语法，openGauss数据库支持SQL2003标准语法，数据库基本操作参见[**附录一**](#_附录一：openGauss数据库基本操作)。

## 实验环境说明

* 组网说明

本实验环境为华为云ECS服务器和openGauss数据库。

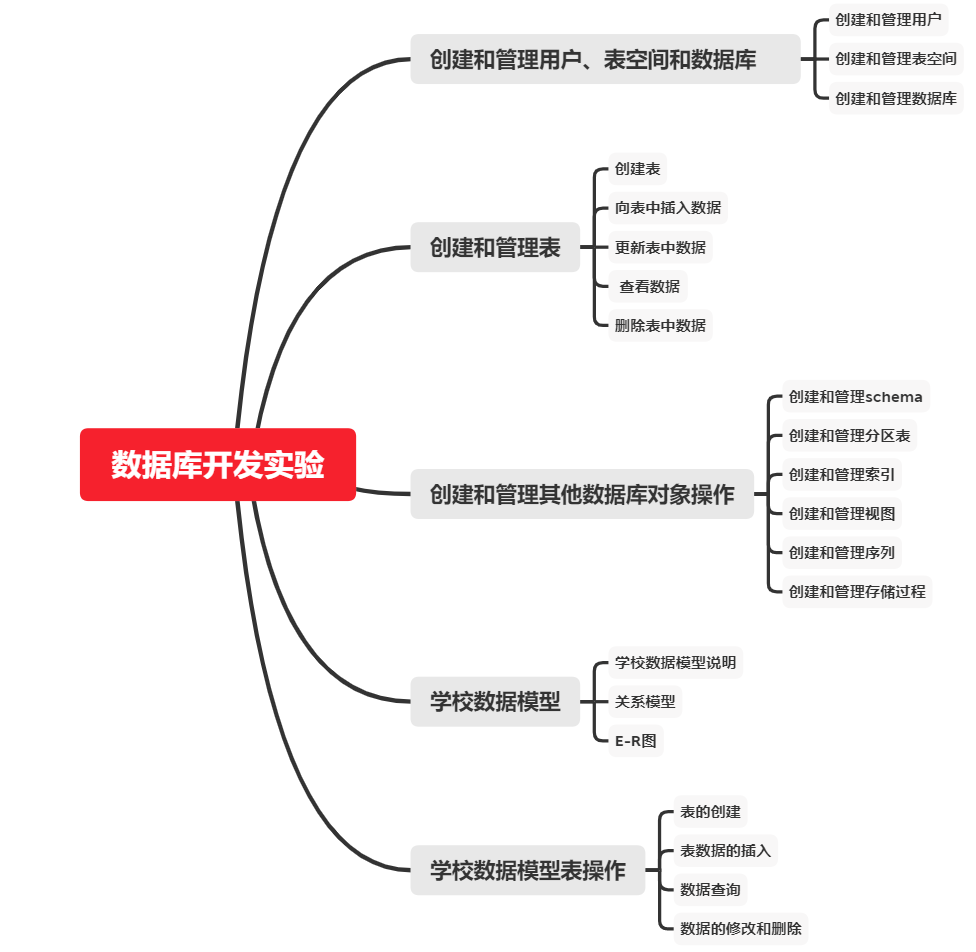
* 设备介绍

为了满足本实验需要，建议实验环境采用以下配置。

配置明细表

|  |  |
| --- | --- |
| 软件名称 | 软件版本 |
| 数据库 | openGauss 2.0.0 |
| 操作系统 | openEuler 20.3LTS |

## 实验概览



本实验概览图

# 数据库开发实验

## 创建和管理用户、表空间和数据库

### 创建和管理用户

#### 创建用户

通过CREATE USER创建的用户，默认具有LOGIN权限。

通过CREATE USER创建用户的同时系统会在执行该命令的数据库中，为该用户创建一个同名的SCHEMA；其他数据库中，则不自动创建同名的SCHEMA；用户可使用CREATE SCHEMA命令，分别在其他数据库中，为该用户创建同名SCHEMA。

系统管理员在普通用户同名schema下创建的对象，所有者为schema的同名用户（非系统管理员）。

创建用户jim，登录密码为Bigdata@123。

postgres=# **CREATE USER jim PASSWORD 'Bigdata@123';**

CREATE ROLE

同样的下面语句也可以创建用户。

postgres=# **CREATE USER kim IDENTIFIED BY 'Bigdata@123';**

CREATE ROLE

如果创建有＂创建数据库＂权限的用户，则需要加CREATEDB关键字。

postgres=# **CREATE USER dim CREATEDB PASSWORD 'Bigdata@123';**

CREATE ROLE

#### 管理用户

将用户jim的登录密码由Bigdata@123修改为Abcd@123。

postgres=# **ALTER USER jim IDENTIFIED BY 'Abcd@123' REPLACE 'Bigdata@123';**

ALTER ROLE

为用户jim追加CREATEROLE权限。

postgres=# **ALTER USER jim CREATEROLE;**

ALTER ROLE

将enable\_seqscan的值设置为on，设置成功后，在下一会话中生效。

postgres=# **ALTER USER jim SET enable\_seqscan TO on;**

ALTER ROLE

锁定jim帐户。

postgres=# **ALTER USER jim ACCOUNT LOCK;**

ALTER ROLE

删除用户。

postgres=# **DROP USER kim CASCADE;**

DROP ROLE

postgres=# **DROP USER jim CASCADE;**

DROP ROLE

postgres=# **DROP USER dim CASCADE;**

DROP ROLE

### 创建和管理表空间

#### 创建表空间

创建用户jack。

postgres=# **CREATE USER jack IDENTIFIED BY 'Bigdata@123';**

当结果显示为如下信息，则表示创建成功。

CREATE ROLE

创建表空间。

postgres=# **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，如 /gaussdb/data/db1/pg\_location/tablespace/tablespace\_1 。

将＂fastspace＂表空间的访问权限赋予数据用户jack。

postgres=# **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

#### 管理表空间

##### 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。

postgres=# **SELECT spcname FROM pg\_tablespace;**

spcname

------------

pg\_default

pg\_global

fastspace

(3 rows)

方式2：使用gsql程序的元命令查询表空间。

postgres=# **\db**

List of tablespaces

Name | Owner | Location

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

fastspace | omm | tablespace/tablespace\_1

pg\_default | omm |

pg\_global | omm |

(3 rows)

##### 查询表空间使用率

查询表空间的当前使用情况。（截图）

postgres=# **SELECT PG\_TABLESPACE\_SIZE('fastspace');**

返回如下信息：

pg\_tablespace\_size

--------------------

4096

(1 row)

其中4096表示表空间的大小，单位为字节。

计算表空间使用率。

表空间使用率=PG\_TABLESPACE\_SIZE/表空间所在目录的磁盘大小。

##### 修改表空间

对表空间fastspace重命名为fspace。

postgres=# **ALTER TABLESPACE fastspace RENAME TO fspace;**

ALTER TABLESPACE

##### 删除表空间

删除用户jack。

postgres=# **DROP USER jack CASCADE;**

DROP ROLE

删除表空间fspace。

postgres=# **DROP TABLESPACE fspace;**

DROP TABLESPACE

说明：用户必须是表空间的owner或者系统管理员才能删除表空间。

### 创建和管理数据库

#### 创建数据库

使用如下命令创建一个新的表空间tpcds\_local。

postgres=# **CREATE TABLESPACE tpcds\_local RELATIVE LOCATION 'tablespace/tablespace\_2';**

CREATE TABLESPACE

使用如下命令创建一个新的数据库db\_tpcc。

postgres=# **CREATE DATABASE db\_tpcc WITH TABLESPACE = tpcds\_local;**

CREATE DATABASE

#### 管理数据库

##### 查看数据库

使用\l元命令查看数据库系统的数据库列表（l表示list）。

postgres=# **\l**

List of databases

Name | Owner | Encoding | Collate | Ctype | Access privileges

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

db\_tpcc | omm | SQL\_ASCII | C | C |

postgres | omm | SQL\_ASCII | C | C |

template0 | omm | SQL\_ASCII | C | C | =c/omm +

| | | | | omm=CTc/omm

template1 | omm | SQL\_ASCII | C | C | =c/omm +

| | | | | omm=CTc/omm

(4 rows)

使用如下命令通过系统表pg\_database查询数据库列表。

postgres=# **SELECT datname FROM pg\_database;**

datname

-----------

template1

db\_tpcc

template0

postgres

(4 rows)

##### 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。

postgres=# **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。

postgres=# **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。此命令删除了数据库中的系统目录，并且删除了带有数据的磁盘上的数据库目录。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：

postgres=# **DROP DATABASE human\_tpcds;**

DROP DATABASE

## 创建和管理表

### 创建表

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。 （截图） （说明，请将红色代码一次性拷贝执行）

postgres=# **CREATE TABLE customer\_t1**

**(**

**c\_customer\_sk integer,**

**c\_customer\_id char(5),**

**c\_first\_name char(6),**

**c\_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。

### 向表中插入数据

#### 向表customer\_t1中插入一行数据

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。

postgres=# **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。

postgres=# **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：

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

或

postgres=# **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：

postgres=# **INSERT INTO customer\_t1 (c\_customer\_sk, c\_customer\_id, c\_first\_name) VALUES (3769, 'hello', DEFAULT);**

或

postgres=# **INSERT INTO customer\_t1 DEFAULT VALUES;**

#### 向表中插入多行数据

命令如下：

postgres=# **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

如果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。

postgres=# **CREATE TABLE customer\_t2**

**(**

**c\_customer\_sk integer,**

**c\_customer\_id char(5),**

**c\_first\_name char(6),**

**c\_last\_name char(8)**

**);**

插入数据：

**INSERT INTO customer\_t2 SELECT \* FROM customer\_t1;**

删除备份表：

postgres=# **DROP TABLE customer\_t2 CASCADE;**

DROP TABLE

### 更新表中数据

修改已经存储在数据库中数据的行为叫做更新。用户可以更新单独一行，所有行或者指定的部分行。还可以独立更新每个字段，而其他字段则不受影响。

需要将表customer\_t1中c\_customer\_sk为9527的字段重新定义为9876：

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

UPDATE 1

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：

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

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：

postgres=# **UPDATE customer\_t1 SET c\_customer\_id = 'Admin', c\_first\_name = 'Local' WHERE c\_customer\_sk = 4421;**

### 查看数据

使用系统表pg\_tables查询数据库所有表的信息。

postgres=# **SELECT \* FROM pg\_tables;**

使用gsql的\d+命令查询表的结构。

postgres=# **\d+ customer\_t1;**

查询表customer\_t1的数据量。

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

查询表customer\_t1的所有数据。

postgres=# **SELECT \* FROM customer\_t1;**

只查询字段c\_customer\_sk的数据。

postgres=# **SELECT c\_customer\_sk FROM customer\_t1;**

过滤字段c\_customer\_sk的重复数据。

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

查询字段c\_customer\_sk为3869的所有数据。

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

按照字段c\_customer\_sk进行排序。

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

查询ROWNUM伪列。

postgres=# **SELECT rownum,c\_customer\_sk,c\_customer\_id FROM customer\_t1;**

使用别名进行查询(CNB、CSK、CID为列别名，T 为表别名)。

postgres=# **SELECT rownum CNB,T.c\_customer\_sk CSK,T.c\_customer\_id CID FROM customer\_t1 T;**

### 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。

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：

postgres=# **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。

postgres=# **DELETE FROM customer\_t1;**

或：

postgres=# **TRUNCATE TABLE customer\_t1;**

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：

postgres=# **DROP TABLE customer\_t1;**

## 创建和管理其他数据库对象操作

### 创建和管理schema

#### 创建schema

创建一个schema。

postgres=# **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。

创建schema时指定owner。（注意，因默认用户为omm，因此再次执行此命令会出现重复schema）

postgres=# **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

#### 管理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。

在myschema下创建mytable表。

postgres=# **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。

查询myschema下mytable表的所有数据。

postgres=# **SELECT \* FROM myschema.mytable;**

id | name

----+------

(0 rows)

##### 设置schema搜索路径

可以设置search\_path配置参数指定寻找对象可用schema的顺序。在搜索路径列出的第一个schema会变成默认的schema。如果在创建对象时不指定schema，则会创建在默认的schema中。

查看搜索路径。

postgres=# **SHOW SEARCH\_PATH;**

search\_path

----------------

＂$user＂,public

(1 row)

将搜索路径设置为myschema、public，首先搜索myschema。

postgres=# **SET SEARCH\_PATH TO myschema,public;**

SET

##### schema的权限控制

默认情况下，用户只能访问属于自己的schema中的数据库对象。如果需要访问其他schema的对象，则该schema的所有者应该赋予他对该schema的usage权限。

通过将模式的CREATE权限授予某用户，被授权用户就可以在此模式中创建对象。注意默认情况下，所有角色都拥有在public模式上的USAGE权限，但是普通用户没有在public模式上的CREATE权限。普通用户能够连接到一个指定数据库并在它的public模式中创建对象是不安全的，如果普通用户具有在public模式上的CREATE权限，则建议通过如下语句撤销该权限。

撤销PUBLIC在public模式下创建对象的权限，下面语句中第一个＂public＂是模式，第二个＂PUBLIC＂指的是所有角色。

postgres=# **REVOKE CREATE ON SCHEMA public FROM PUBLIC;**

REVOKE

使用以下命令查看现有的schema：

postgres=# **SELECT current\_schema();**

current\_schema

----------------

myschema

(1 row)

创建用户jack，并将myschema的usage权限赋给用户jack。

postgres=# **CREATE USER jack IDENTIFIED BY 'Bigdata@123';**

CREATE ROLE

postgres=# **GRANT USAGE ON schema myschema TO jack;**

GRANT

将用户jack对于myschema的usage权限收回。

postgres=# **REVOKE USAGE ON schema myschema FROM jack;**

REVOKE

##### 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。

postgres=# **DROP SCHEMA IF EXISTS nullschema;**

NOTICE: schema ＂nullschema＂ does not exist, skipping

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。

postgres=# **DROP SCHEMA myschema CASCADE;**

DROP SCHEMA

删除用户jack。

postgres=# **DROP USER jack;**

DROP ROLE

### 创建和管理分区表

#### 创建分区表

创建schema。

**CREATE SCHEMA tpcds;**

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

创建分区表并插入数据。

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');**

插入数据返回如下。（查看表中数据并截图，命令未提供，请自行完成）

INSERT 0 4

#### 管理分区表

##### 修改分区表行迁移属性

命令如下：

postgres=# **ALTER TABLE tpcds.web\_returns\_p2 DISABLE ROW MOVEMENT;**

ALTER TABLE

##### 删除分区

删除分区P8。

postgres=# **ALTER TABLE tpcds.web\_returns\_p2 DROP PARTITION P8;**

ALTER TABLE

##### 增加分区

增加分区P8。

postgres=# **ALTER TABLE tpcds.web\_returns\_p2 ADD PARTITION P8 VALUES LESS THAN (MAXVALUE);**

ALTER TABLE

##### 重命名分区

重命名分区P8为P\_9。

postgres=# **ALTER TABLE tpcds.web\_returns\_p2 RENAME PARTITION P8 TO P\_9;**

ALTER TABLE

重命名分区P\_9为P8。

postgres=# **ALTER TABLE tpcds.web\_returns\_p2 RENAME PARTITION FOR (40000) TO P8;**

ALTER TABLE

##### 修改分区的表空间

修改分区P6的表空间为example3。

postgres=# **ALTER TABLE tpcds.web\_returns\_p2 MOVE PARTITION P6 TABLESPACE example3;**

ALTER TABLE

修改分区P4的表空间为example4。

postgres=# **ALTER TABLE tpcds.web\_returns\_p2 MOVE PARTITION P4 TABLESPACE example4;**

ALTER TABLE

##### 查询分区

查询分区P1。

postgres=# **SELECT \* FROM tpcds.web\_returns\_p2 PARTITION (P1);**

##### 删除分区表和表空间

命令如下

postgres=# **DROP TABLE tpcds.web\_returns\_p2;**

DROP TABLE

postgres=# **DROP TABLESPACE example1;**

postgres=# **DROP TABLESPACE example2;**

postgres=# **DROP TABLESPACE example3;**

postgres=# **DROP TABLESPACE example4;**

DROP TABLESPACE

### 创建和管理索引

#### 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');**

#### 创建索引

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。（截图）

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

#### 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

#### 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

### 创建和管理视图

#### 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。

#### 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。 （截图）

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

**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)

##### 删除视图

删除视图。

postgres=# **DROP VIEW MyView;**

DROP VIEW

postgres=# **DROP MATERIALIZED VIEW MV\_MyView;**

DROP MATERIALIZED VIEW

### 创建和管理序列

#### 方法一：声明字段类型为序列整型(serial)来定义标识符字段

例如：

postgres=# **CREATE TABLE T1**

**(id serial,**

**name text**

**);**

当结果显示为如下信息，则表示创建成功。

CREATE TABLE

#### 方法二：创建序列，并通过nextval('sequence\_name')函数指定为某一字段的默认值

创建序列。

postgres=# **CREATE SEQUENCE seq1 cache 100;**

结果显示为如下信息，则表示创建成功。

CREATE SEQUENCE

指定为某一字段的默认值，使该字段具有唯一标识属性。

postgres=# **CREATE TABLE T2**

**(**

**id int not null default nextval('seq1'),**

**name text**

**);**

当结果显示为如下信息，则表示默认值指定成功。

CREATE TABLE

指定序列与列的归属关系。

将序列和一个表的指定字段进行关联。删除此字段或其所在表的时候会自动删除已关联的序列。

postgres=# **ALTER SEQUENCE seq1 OWNED BY T2.id;**

当结果显示为如下信息，则表示指定成功。

ALTER SEQUENCE

#### 删除序列

**DROP SEQUENCE seq1 CASCADE;**

### 创建和管理存储过程

#### 创建存储过程

创建表t\_test。

postgres=# **create table t\_test(c1 int, c2 int);**

创建存储过程insert\_data。

postgres=# **create or replace procedure insert\_data**

**is**

**a int;**

**b int;**

**begin**

**a=1;**

**b=2;**

**insert into t\_test values(a,b);**

**insert into t\_test values(b,a);**

**end;**

**/**

调用存储过程。

**call insert\_data();**

查询表内容。

postgres=# **select \* from t\_test;**

c1 | c2

----+----

1 | 2

2 | 1

(2 rows)

#### 管理存储过程

管理存储过程，命令如下： （截图）

postgres=# **\sf insert\_data**

结果如下：

CREATE OR REPLACE FUNCTION public.insert\_data()

RETURNS void

LANGUAGE plpgsql

NOT FENCED NOT SHIPPABLE

AS $function$ DECLARE

a int;

b int;

begin

a=1;

b=2;

insert into t\_test values(a,b);

insert into t\_test values(b,a);

end$function$

删除存储过程，命令如下：

**drop procedure insert\_data;**

### 创建和管理全局临时表

#### 会话级全局临时表

数据会话级可见，其他会话看不到数据，但表结构可见。

创建临时表t\_test2。

建表语句，使用**ON** **COMMIT** **PRESERVE** **ROWS**

postgres=# **CREATE GLOBAL TEMPORARY TABLE t\_test2(**

**id integer,**

**lbl text**

**) ON COMMIT PRESERVE ROWS;**

成功返回如下：

CREATE TABLE

在当前会话插入数据并查询。

postgres=# **insert into t\_test2 values(1,'data1');**

INSERT 0 1

postgres=# **insert into t\_test2 values(2,'data2');**

INSERT 0 1

postgres=# **select \* from t\_test2;**

id | lbl

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

1 | data1

2 | data2

(2 rows)

退出会话再查看。

postgres=# **\q**

[omm@ecs-32de ~]$ **gsql -d postgres -p 26000 -r**

gsql ((openGauss 1.0.1 build 13b34b53) compiled at 2020-10-12 02:01:33 commit 0 last mr )

Non-SSL connection (SSL connection is recommended when requiring high-security)

Type ＂help＂ for help.

postgres=# **\dt t\_test2**

List of relations

Schema | Name | Type | Owner | Storage

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

public | t\_test2 | table | omm | {orientation=row,compression=no,on\_commit\_delete\_rows=false}

(1 row)

查询表内容。

postgres=# **select \* from t\_test2;**

id | lbl

----+-----

(0 rows)

此时可以发现，在其它会话中表结构可以看到，但是表数据看不到。

删除临时表。

postgres=# **drop table t\_test2;**

DROP TABLE

#### 事务级全局临时表

数据事务级可见，事务提交后数据删除。

创建临时表t\_test3。

建表语句，使用**ON** **COMMIT** **DELETE** **ROWS**

postgres=# **CREATE GLOBAL TEMPORARY TABLE t\_test3(**

**id integer,**

**lbl text**

**) ON COMMIT DELETE ROWS;**

CREATE TABLE

插入数据并查询。

先用begin开始一个事务，接着给表插入数据，此时再对表进行查询，可以查出相应数据。

postgres=# **begin;**

BEGIN

postgres=# **insert into t\_test3 values(1,'data1');**

INSERT 0 1

postgres=# **select \* from t\_test3;**

id | lbl

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

1 | data1

(1 row)

结束事务再查询。

先用commit提交来结束事务，此时再对表进行查询，可以发现已经查询不出数据了。

postgres=# **commit;**

COMMIT

postgres=# **select \* from t\_test3;**

id | lbl

----+-----

(0 rows) (1 row)

删除临时表。

postgres=# **drop table t\_test3;**

DROP TABLE

## 学校数据模型

### 关于本实验

以学校数据库模型为例，介绍openGauss数据库数据库、表、表空间、用户及其它对象，以及SQL语法使用。

假设A市B学校为了加强对学校的管理，引入了华为openGauss数据库。在B学校里，主要涉及的对象有学生、教师、班级、院系和课程。本实验假设在B学校数据库中，教师会教授课程，学生会选修课程，院系会聘请教师，班级会组成院系，学生会组成班级。因此，根据此关系，本实验给出了相应的关系模式和ER图，并对其进行基本的数据库操作。

### 关系模型

对于B校中的5个对象，分别建立属于每个对象的属性集合，具体属性描述如下：

* 学生（学号，姓名，性别，出生日期，入学日期，家庭住址）
* 教师（教师编号，教师姓名，职称，性别，年龄，入职日期）
* 班级（班级编号，班级名称，班主任）
* 院系（系编号，系名称，系主任）
* 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：

* student（std\_id，std\_name，std\_sex，std\_birth，std\_in，std\_address）
* teacher（tec\_id，tec\_name，tec\_job，tec\_sex，tec\_age，tec\_in）
* class（cla\_id，cla\_name，cla\_teacher）
* school\_department（depart\_id，depart\_name，depart\_teacher）
* course（cor\_id，cor\_name，cor\_type，credit）

对象之间的关系：

* 一位学生可以选择多门课程，一门课程可被多名学生选择
* 一位老师可以教授多门课程，一门课程可被多名老师教授
* 一个院系可由多个班级组成
* 一个院系可聘请多名老师
* 一个班级可由多名学生组成

**说明：请先自行绘制ER图，再和给出的ER图进行参考对照。请将自行绘制的ER图填写在实验报告中。**

### E-R图



M

E-R图

## 学校数据模型表操作

### 创建表

根据B学校的场景描述，本实验分别针对学生（student），教师（teacher），班级（class），院系（school\_department）和课程（course）创建相应的表。具体的实验步骤如下所示：

创建学生信息表。

**DROP TABLE IF EXISTS student;**

**CREATE TABLE student**

**(**

**std\_id INT PRIMARY KEY,**

**std\_name NCHAR(20) NOT NULL,**

**std\_sex NCHAR(6),**

**std\_birth DATE,**

**std\_in DATE NOT NULL,**

**std\_address VARCHAR(100)**

**);**

创建教师信息表。

**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**

**);**

创建班级信息表。

**DROP TABLE IF EXISTS class;**

**CREATE TABLE class**

**(**

**cla\_id INT PRIMARY KEY,**

**cla\_name CHAR(20) NOT NULL,**

**cla\_teacher INT NOT NULL**

**);**

创建院系信息表。

**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**

**);**

创建课程信息表。

**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**

**);**

### 插入表数据

向student表中插入数据。

INSERT INTO student(std\_id,std\_name,std\_sex,std\_birth,std\_in,std\_address) VALUES (1,'张一','男','1993-01-01','2011-09-01','江苏省南京市雨花台区');

INSERT INTO student(std\_id,std\_name,std\_sex,std\_birth,std\_in,std\_address) VALUES (2,'张二','男','1993-01-02','2011-09-01','江苏省南京市雨花台区');

INSERT INTO student(std\_id,std\_name,std\_sex,std\_birth,std\_in,std\_address) VALUES (3,'张三','男','1993-01-03','2011-09-01','江苏省南京市雨花台区');

INSERT INTO student(std\_id,std\_name,std\_sex,std\_birth,std\_in,std\_address) VALUES (4,'张四','男','1993-01-04','2011-09-01','江苏省南京市雨花台区');

INSERT INTO student(std\_id,std\_name,std\_sex,std\_birth,std\_in,std\_address) VALUES (5,'张五','男','1993-01-05','2011-09-01','江苏省南京市雨花台区');

INSERT INTO student(std\_id,std\_name,std\_sex,std\_birth,std\_in,std\_address) VALUES (6,'张六','男','1993-01-06','2011-09-01','江苏省南京市雨花台区');

INSERT INTO student(std\_id,std\_name,std\_sex,std\_birth,std\_in,std\_address) VALUES (7,'张七','男','1993-01-07','2011-09-01','江苏省南京市雨花台区');

INSERT INTO student(std\_id,std\_name,std\_sex,std\_birth,std\_in,std\_address) VALUES (8,'张八','男','1993-01-08','2011-09-01','江苏省南京市雨花台区');

INSERT INTO student(std\_id,std\_name,std\_sex,std\_birth,std\_in,std\_address) VALUES (9,'张九','男','1993-01-09','2011-09-01','江苏省南京市雨花台区');

INSERT INTO student(std\_id,std\_name,std\_sex,std\_birth,std\_in,std\_address) VALUES (10,'李一','男','1993-01-10','2011-09-01','江苏省南京市雨花台区');

INSERT INTO student(std\_id,std\_name,std\_sex,std\_birth,std\_in,std\_address) VALUES (11,'李二','男','1993-01-11','2011-09-01','江苏省南京市雨花台区');

INSERT INTO student(std\_id,std\_name,std\_sex,std\_birth,std\_in,std\_address) VALUES (12,'李三','男','1993-01-12','2011-09-01','江苏省南京市雨花台区');

INSERT INTO student(std\_id,std\_name,std\_sex,std\_birth,std\_in,std\_address) VALUES (13,'李四','男','1993-01-13','2011-09-01','江苏省南京市雨花台区');

INSERT INTO student(std\_id,std\_name,std\_sex,std\_birth,std\_in,std\_address) VALUES (14,'李五','男','1993-01-14','2011-09-01','江苏省南京市雨花台区');

INSERT INTO student(std\_id,std\_name,std\_sex,std\_birth,std\_in,std\_address) VALUES (15,'李六','男','1993-01-15','2011-09-01','江苏省南京市雨花台区');

INSERT INTO student(std\_id,std\_name,std\_sex,std\_birth,std\_in,std\_address) VALUES (16,'李七','男','1993-01-16','2011-09-01','江苏省南京市雨花台区');

INSERT INTO student(std\_id,std\_name,std\_sex,std\_birth,std\_in,std\_address) VALUES (17,'李八','男','1993-01-17','2011-09-01','江苏省南京市雨花台区');

INSERT INTO student(std\_id,std\_name,std\_sex,std\_birth,std\_in,std\_address) VALUES (18,'李九','男','1993-01-18','2011-09-01','江苏省南京市雨花台区');

INSERT INTO student(std\_id,std\_name,std\_sex,std\_birth,std\_in,std\_address) VALUES (19,'王一','男','1993-01-19','2011-09-01','江苏省南京市雨花台区');

INSERT INTO student(std\_id,std\_name,std\_sex,std\_birth,std\_in,std\_address) VALUES (20,'王二','男','1993-01-20','2011-09-01','江苏省南京市雨花台区');

INSERT INTO student(std\_id,std\_name,std\_sex,std\_birth,std\_in,std\_address) VALUES (21,'王三','男','1993-01-21','2011-09-01','江苏省南京市雨花台区');

INSERT INTO student(std\_id,std\_name,std\_sex,std\_birth,std\_in,std\_address) VALUES (22,'王四','男','1993-01-22','2011-09-01','江苏省南京市雨花台区');

INSERT INTO student(std\_id,std\_name,std\_sex,std\_birth,std\_in,std\_address) VALUES (23,'王五','男','1993-01-23','2011-09-01','江苏省南京市雨花台区');

INSERT INTO student(std\_id,std\_name,std\_sex,std\_birth,std\_in,std\_address) VALUES (24,'王六','男','1993-01-24','2011-09-01','江苏省南京市雨花台区');

INSERT INTO student(std\_id,std\_name,std\_sex,std\_birth,std\_in,std\_address) VALUES (25,'王七','男','1993-01-25','2011-09-01','江苏省南京市雨花台区');

INSERT INTO student(std\_id,std\_name,std\_sex,std\_birth,std\_in,std\_address) VALUES (26,'王八','男','1993-01-26','2011-09-01','江苏省南京市雨花台区');

INSERT INTO student(std\_id,std\_name,std\_sex,std\_birth,std\_in,std\_address) VALUES (27,'王九','男','1993-01-27','2011-09-01','江苏省南京市雨花台区');

INSERT INTO student(std\_id,std\_name,std\_sex,std\_birth,std\_in,std\_address) VALUES (28,'钱一','男','1993-01-28','2011-09-01','江苏省南京市雨花台区');

INSERT INTO student(std\_id,std\_name,std\_sex,std\_birth,std\_in,std\_address) VALUES (29,'钱二','男','1993-01-29','2011-09-01','江苏省南京市雨花台区');

INSERT INTO student(std\_id,std\_name,std\_sex,std\_birth,std\_in,std\_address) VALUES (30,'钱三','男','1993-01-30','2011-09-01','江苏省南京市雨花台区');

INSERT INTO student(std\_id,std\_name,std\_sex,std\_birth,std\_in,std\_address) VALUES (31,'钱四','男','1993-02-01','2011-09-01','江苏省南京市雨花台区');

INSERT INTO student(std\_id,std\_name,std\_sex,std\_birth,std\_in,std\_address) VALUES (32,'钱五','男','1993-02-02','2011-09-01','江苏省南京市雨花台区');

INSERT INTO student(std\_id,std\_name,std\_sex,std\_birth,std\_in,std\_address) VALUES (33,'钱六','男','1993-02-03','2011-09-01','江苏省南京市雨花台区');

INSERT INTO student(std\_id,std\_name,std\_sex,std\_birth,std\_in,std\_address) VALUES (34,'钱七','男','1993-02-04','2011-09-01','江苏省南京市雨花台区');

INSERT INTO student(std\_id,std\_name,std\_sex,std\_birth,std\_in,std\_address) VALUES (35,'钱八','男','1993-02-05','2011-09-01','江苏省南京市雨花台区');

INSERT INTO student(std\_id,std\_name,std\_sex,std\_birth,std\_in,std\_address) VALUES (36,'钱九','男','1993-02-06','2011-09-01','江苏省南京市雨花台区');

INSERT INTO student(std\_id,std\_name,std\_sex,std\_birth,std\_in,std\_address) VALUES (37,'吴一','男','1993-02-07','2011-09-01','江苏省南京市雨花台区');

INSERT INTO student(std\_id,std\_name,std\_sex,std\_birth,std\_in,std\_address) VALUES (38,'吴二','男','1993-02-08','2011-09-01','江苏省南京市雨花台区');

INSERT INTO student(std\_id,std\_name,std\_sex,std\_birth,std\_in,std\_address) VALUES (39,'吴三','男','1993-02-09','2011-09-01','江苏省南京市雨花台区');

INSERT INTO student(std\_id,std\_name,std\_sex,std\_birth,std\_in,std\_address) VALUES (40,'吴四','男','1993-02-10','2011-09-01','江苏省南京市雨花台区');

INSERT INTO student(std\_id,std\_name,std\_sex,std\_birth,std\_in,std\_address) VALUES (41,'吴五','男','1993-02-11','2011-09-01','江苏省南京市雨花台区');

INSERT INTO student(std\_id,std\_name,std\_sex,std\_birth,std\_in,std\_address) VALUES (42,'吴六','男','1993-02-12','2011-09-01','江苏省南京市雨花台区');

INSERT INTO student(std\_id,std\_name,std\_sex,std\_birth,std\_in,std\_address) VALUES (43,'吴七','男','1993-02-13','2011-09-01','江苏省南京市雨花台区');

INSERT INTO student(std\_id,std\_name,std\_sex,std\_birth,std\_in,std\_address) VALUES (44,'吴八','男','1993-02-14','2011-09-01','江苏省南京市雨花台区');

INSERT INTO student(std\_id,std\_name,std\_sex,std\_birth,std\_in,std\_address) VALUES (45,'吴九','男','1993-02-15','2011-09-01','江苏省南京市雨花台区');

INSERT INTO student(std\_id,std\_name,std\_sex,std\_birth,std\_in,std\_address) VALUES (46,'柳一','男','1993-02-16','2011-09-01','江苏省南京市雨花台区');

INSERT INTO student(std\_id,std\_name,std\_sex,std\_birth,std\_in,std\_address) VALUES (47,'柳二','男','1993-02-17','2011-09-01','江苏省南京市雨花台区');

INSERT INTO student(std\_id,std\_name,std\_sex,std\_birth,std\_in,std\_address) VALUES (48,'柳三','男','1993-02-18','2011-09-01','江苏省南京市雨花台区');

INSERT INTO student(std\_id,std\_name,std\_sex,std\_birth,std\_in,std\_address) VALUES (49,'柳四','男','1993-02-19','2011-09-01','江苏省南京市雨花台区');

INSERT INTO student(std\_id,std\_name,std\_sex,std\_birth,std\_in,std\_address) VALUES (50,'柳五','男','1993-02-20','2011-09-01','江苏省南京市雨花台区');

向teacher表中插入数据。

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

INSERT INTO teacher(tec\_id,tec\_name,tec\_job,tec\_sex,tec\_age,tec\_in) VALUES (21,'王三','讲师','男',35,'2009-07-01');

INSERT INTO teacher(tec\_id,tec\_name,tec\_job,tec\_sex,tec\_age,tec\_in) VALUES (22,'王四','讲师','男',35,'2009-07-01');

INSERT INTO teacher(tec\_id,tec\_name,tec\_job,tec\_sex,tec\_age,tec\_in) VALUES (23,'王五','讲师','男',35,'2009-07-01');

INSERT INTO teacher(tec\_id,tec\_name,tec\_job,tec\_sex,tec\_age,tec\_in) VALUES (24,'王六','讲师','男',35,'2009-07-01');

INSERT INTO teacher(tec\_id,tec\_name,tec\_job,tec\_sex,tec\_age,tec\_in) VALUES (25,'王七','讲师','男',35,'2009-07-01');

INSERT INTO teacher(tec\_id,tec\_name,tec\_job,tec\_sex,tec\_age,tec\_in) VALUES (26,'王八','讲师','男',35,'2009-07-01');

INSERT INTO teacher(tec\_id,tec\_name,tec\_job,tec\_sex,tec\_age,tec\_in) VALUES (27,'王九','讲师','男',35,'2009-07-01');

INSERT INTO teacher(tec\_id,tec\_name,tec\_job,tec\_sex,tec\_age,tec\_in) VALUES (28,'钱一','讲师','男',35,'2009-07-01');

INSERT INTO teacher(tec\_id,tec\_name,tec\_job,tec\_sex,tec\_age,tec\_in) VALUES (29,'钱二','讲师','男',35,'2009-07-01');

INSERT INTO teacher(tec\_id,tec\_name,tec\_job,tec\_sex,tec\_age,tec\_in) VALUES (30,'钱三','讲师','男',35,'2009-07-01');

INSERT INTO teacher(tec\_id,tec\_name,tec\_job,tec\_sex,tec\_age,tec\_in) VALUES (31,'钱四','讲师','男',35,'2009-07-01');

INSERT INTO teacher(tec\_id,tec\_name,tec\_job,tec\_sex,tec\_age,tec\_in) VALUES (32,'钱五','讲师','男',35,'2009-07-01');

INSERT INTO teacher(tec\_id,tec\_name,tec\_job,tec\_sex,tec\_age,tec\_in) VALUES (33,'钱六','讲师','男',35,'2009-07-01');

INSERT INTO teacher(tec\_id,tec\_name,tec\_job,tec\_sex,tec\_age,tec\_in) VALUES (34,'钱七','讲师','男',35,'2009-07-01');

INSERT INTO teacher(tec\_id,tec\_name,tec\_job,tec\_sex,tec\_age,tec\_in) VALUES (35,'钱八','讲师','男',35,'2009-07-01');

INSERT INTO teacher(tec\_id,tec\_name,tec\_job,tec\_sex,tec\_age,tec\_in) VALUES (36,'钱九','讲师','男',35,'2009-07-01');

INSERT INTO teacher(tec\_id,tec\_name,tec\_job,tec\_sex,tec\_age,tec\_in) VALUES (37,'吴一','讲师','男',35,'2009-07-01');

INSERT INTO teacher(tec\_id,tec\_name,tec\_job,tec\_sex,tec\_age,tec\_in) VALUES (38,'吴二','讲师','男',35,'2009-07-01');

INSERT INTO teacher(tec\_id,tec\_name,tec\_job,tec\_sex,tec\_age,tec\_in) VALUES (39,'吴三','讲师','男',35,'2009-07-01');

INSERT INTO teacher(tec\_id,tec\_name,tec\_job,tec\_sex,tec\_age,tec\_in) VALUES (40,'吴四','讲师','男',35,'2009-07-01');

INSERT INTO teacher(tec\_id,tec\_name,tec\_job,tec\_sex,tec\_age,tec\_in) VALUES (41,'吴五','讲师','男',35,'2009-07-01');

INSERT INTO teacher(tec\_id,tec\_name,tec\_job,tec\_sex,tec\_age,tec\_in) VALUES (42,'吴六','讲师','男',35,'2009-07-01');

INSERT INTO teacher(tec\_id,tec\_name,tec\_job,tec\_sex,tec\_age,tec\_in) VALUES (43,'吴七','讲师','男',35,'2009-07-01');

INSERT INTO teacher(tec\_id,tec\_name,tec\_job,tec\_sex,tec\_age,tec\_in) VALUES (44,'吴八','讲师','男',35,'2009-07-01');

INSERT INTO teacher(tec\_id,tec\_name,tec\_job,tec\_sex,tec\_age,tec\_in) VALUES (45,'吴九','讲师','男',35,'2009-07-01');

INSERT INTO teacher(tec\_id,tec\_name,tec\_job,tec\_sex,tec\_age,tec\_in) VALUES (46,'柳一','讲师','男',35,'2009-07-01');

INSERT INTO teacher(tec\_id,tec\_name,tec\_job,tec\_sex,tec\_age,tec\_in) VALUES (47,'柳二','讲师','男',35,'2009-07-01');

INSERT INTO teacher(tec\_id,tec\_name,tec\_job,tec\_sex,tec\_age,tec\_in) VALUES (48,'柳三','讲师','男',35,'2009-07-01');

INSERT INTO teacher(tec\_id,tec\_name,tec\_job,tec\_sex,tec\_age,tec\_in) VALUES (49,'柳四','讲师','男',35,'2009-07-01');

INSERT INTO teacher(tec\_id,tec\_name,tec\_job,tec\_sex,tec\_age,tec\_in) VALUES (50,'柳五','讲师','男',35,'2009-07-01');

向class表插入数据。

INSERT INTO class(cla\_id,cla\_name,cla\_teacher) VALUES (1,'计算机',1);

INSERT INTO class(cla\_id,cla\_name,cla\_teacher) VALUES (2,'自动化',3);

INSERT INTO class(cla\_id,cla\_name,cla\_teacher) VALUES (3,'飞行器设计',5);

INSERT INTO class(cla\_id,cla\_name,cla\_teacher) VALUES (4,'大学物理',7);

INSERT INTO class(cla\_id,cla\_name,cla\_teacher) VALUES (5,'高等数学',9);

INSERT INTO class(cla\_id,cla\_name,cla\_teacher) VALUES (6,'大学化学',12);

INSERT INTO class(cla\_id,cla\_name,cla\_teacher) VALUES (7,'表演',14);

INSERT INTO class(cla\_id,cla\_name,cla\_teacher) VALUES (8,'服装设计',16);

INSERT INTO class(cla\_id,cla\_name,cla\_teacher) VALUES (9,'工业设计',18);

INSERT INTO class(cla\_id,cla\_name,cla\_teacher) VALUES (10,'金融学',21);

INSERT INTO class(cla\_id,cla\_name,cla\_teacher) VALUES (11,'医学',23);

INSERT INTO class(cla\_id,cla\_name,cla\_teacher) VALUES (12,'土木工程',25);

INSERT INTO class(cla\_id,cla\_name,cla\_teacher) VALUES (13,'机械',27);

INSERT INTO class(cla\_id,cla\_name,cla\_teacher) VALUES (14,'建筑学',29);

INSERT INTO class(cla\_id,cla\_name,cla\_teacher) VALUES (15,'经济学',32);

INSERT INTO class(cla\_id,cla\_name,cla\_teacher) VALUES (16,'财务管理',34);

INSERT INTO class(cla\_id,cla\_name,cla\_teacher) VALUES (17,'人力资源',36);

INSERT INTO class(cla\_id,cla\_name,cla\_teacher) VALUES (18,'力学',38);

INSERT INTO class(cla\_id,cla\_name,cla\_teacher) VALUES (19,'人工智能',41);

INSERT INTO class(cla\_id,cla\_name,cla\_teacher) VALUES (20,'会计',45);

向school\_department表插入数据。

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 school\_department(depart\_id,depart\_name,depart\_teacher) VALUES (9,'农学院',22);

INSERT INTO school\_department(depart\_id,depart\_name,depart\_teacher) VALUES (10,'医学院',28);

向course表插入数据。

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);

### 查询数据 （本部分给出参考查询语句，实际实验时，请先自行撰写SQL语句，再对照给出的查询语句）

#### 单表查询

查询B校课程信息表的所有信息。

postgres=# **SELECT \* from course;**

cor\_id | cor\_name | cor\_type | credit

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

1 | C语言程序设计 | 必修 | 3.5

2 | 艺术设计概论 | 选修 | 1

3 | 力学制图 | 必修 | 4

4 | 飞行器设计历史 | 选修 | 1

5 | 马克思主义 | 必修 | 2

6 | 大学历史 | 必修 | 2

7 | 人力资源管理理论 | 必修 | 2.5

8 | 线性代数 | 必修 | 4

9 | JAVA程序设计 | 必修 | 3

10 | 操作系统 | 必修 | 4

11 | 计算机组成原理 | 必修 | 3

12 | 自动化设计理论 | 必修 | 2

13 | 情绪表演 | 必修 | 2.5

14 | 茶学历史 | 选修 | 1

15 | 艺术论 | 必修 | 1.5

16 | 机器学习 | 必修 | 3

17 | 数据挖掘 | 选修 | 2

18 | 图像识别 | 必修 | 3

19 | 解剖学 | 必修 | 4

20 | 3D max | 选修 | 2

(20 rows)

#### 条件查询

在教师信息表中查询教师编号大于45的老师的入学年份。

postgres=# **SELECT tec\_id, tec\_in FROM teacher WHERE tec\_id>45;**

tec\_id | tec\_in

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

46 | 2009-07-01 00:00:00

47 | 2009-07-01 00:00:00

48 | 2009-07-01 00:00:00

49 | 2009-07-01 00:00:00

50 | 2009-07-01 00:00:00

(5 rows)

查询B校中所有选修的课程的信息。

postgres=# **SELECT \* FROM course WHERE cor\_type='选修';**

cor\_id | cor\_name | cor\_type | credit

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

2 | 艺术设计概论 | 选修 | 1

4 | 飞行器设计历史 | 选修 | 1

14 | 茶学历史 | 选修 | 1

17 | 数据挖掘 | 选修 | 2

20 | 3D max | 选修 | 2

(5 rows)

### 修改和删除数据

#### 修改数据

修改/更新课程信息表数据。

postgres=# **UPDATE course SET cor\_name='C语言程序设计',cor\_type='必修',credit=3.5 WHERE cor\_id=1;**

UPDATE 1

postgres=# **SELECT \* FROM course WHERE cor\_id=1;**

cor\_id | cor\_name | cor\_type | credit

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

1 | C语言程序设计 | 必修 | 3.5

(1 row)

#### 删除指定数据

在B校中删除教师编号8和15所管理的院系。

postgres=# **DELETE FROM school\_department WHERE depart\_teacher=8 OR depart\_teacher=15;**

DELETE 0

postgres=# **SELECT \* FROM school\_department; （截图）**

depart\_id | depart\_name | depart\_teacher

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

1 | 计算机学院 | 2

2 | 自动化学院 | 4

3 | 航空宇航学院 | 6

5 | 理学院 | 11

6 | 人工智能学院 | 13

8 | 管理学院 | 17

9 | 农学院 | 22

10 | 医学院 | 28

(8 rows)

本实验结束。

# 附录一：openGauss数据库基本操作

## 查看数据库对象

* 查看帮助信息。

postgres=# \?

* 切换数据库。

postgres=# \c dbname

* 列举数据库。

使用\l元命令查看数据库系统的数据库列表。

postgres=# \l

使用如下命令通过系统表pg\_database查询数据库列表。

postgres=# SELECT datname FROM pg\_database;

* 列举表。

postgres=# \dt

* 列举所有表、视图和索引。

postgres=# \d+

使用gsql的\d+命令查询表的属性。

postgres=# \d+ tablename

* 查看表结构。

*p*ostgres=# \d tablename

* 列举schema。

postgres=# \dn

* 查看索引。

postgres=# \di

* 查询表空间。

使用gsql程序的元命令查询表空间。

postgres=# \db

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。

postgres=# SELECT spcname FROM pg\_tablespace;

* 查看数据库用户列表。

postgres=# SELECT \* FROM pg\_user;

* 查看用户属性。

postgres=# SELECT \* FROM pg\_authid;

* 查看所有角色。

postgres=# SELECT \* FROM PG\_ROLES;

## 其他操作

* 查看openGauss支持的所有SQL语句。

postgres=#\h

* 切换用户。

postgres=# \c – username

* 退出数据库。

postgres=# \q