实　　验　　报　　告

学　　号	2312001 4029	姓　　名	张心顺	专业班级	计算机 23 级
课程名称	**数据库系统**			学期	**2025 年秋季学期**
任课教师	刘艳艳	完成日期	251210	上机课时间	251210
实 验 名 称	openGauss 数据库开发指导手册				

一、实验要求（10%）

熟悉并掌握 openGauss 数据库的基本操作，并通过案例强化学习。

二、实验内容及步骤（80%）

1.0 先启动数据库，执行：

```
gs_om -t start
gsql -d postgres -p 26000 -r
```

```
[omm@ecs-ac18 ~]$ gs_om -t start
Starting cluster.
=========================================
=========================================
Successfully started.
[omm@ecs-ac18 ~]$ gsql -d postgres -p 26000 -r
gsql ((openGauss 1.1.0 build 392c0438) compiled at 2020-12-31 20:08:06 commit 0 last mr
)
Non-SSL connection (SSL connection is recommended when requiring high-security)
Type "help" for help.

postgres=#
```

1.1 创建和管理用户、表空间和数据库

1.1.1 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。

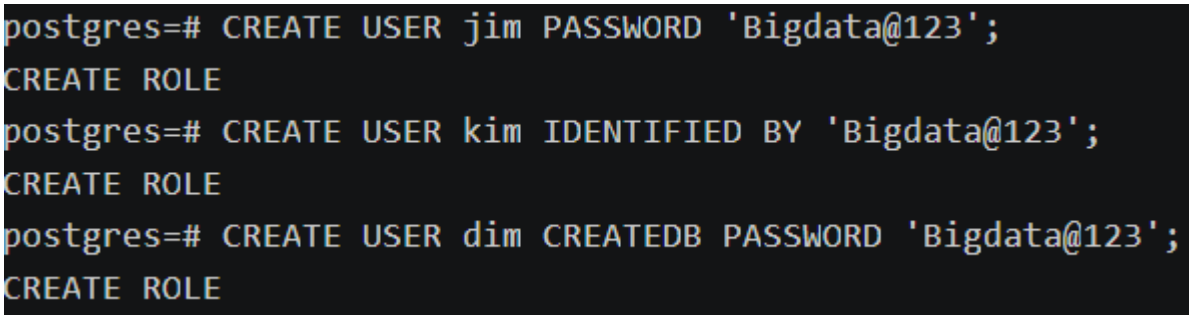
```
postgres=# CREATE USER jim PASSWORD 'Bigdata@123';
CREATE ROLE
```

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
```

```
postgres=# ALTER USER jim IDENTIFIED BY 'Abcd@123' REPLACE 'Bigdata@123';
ALTER ROLE
postgres=# ALTER USER jim CREATEROLE;
ALTER ROLE
postgres=# ALTER USER jim SET enable_seqscan TO on;
ALTER ROLE
postgres=# ALTER USER jim ACCOUNT LOCK;
ALTER ROLE
postgres=# DROP USER kim CASCADE;
DROP ROLE
postgres=# DROP USER jim CASCADE;
DROP ROLE
postgres=# DROP USER dim CASCADE;
DROP ROLE
```

1.1.2 创建和管理表空间

创建表空间

步骤 1　执行如下命令创建用户 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
```

步骤 2　执行如下命令创建表空间。

```
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 。

步骤 3　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。

```
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
```

```
postgres=# CREATE USER jack IDENTIFIED BY 'Bigdata@123';
CREATE ROLE
postgres=# CREATE TABLESPACE fastspace RELATIVE LOCATION 'tablespace/tablespace_1';
CREATE TABLESPACE
postgres=# GRANT CREATE ON TABLESPACE fastspace TO jack;
GRANT
```

管理表空间

1.1.2.1.2 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)
```

```
postgres=# SELECT spcname FROM pg_tablespace;
  spcname
------------
 pg_default
 pg_global
 fastspace
(3 rows)

postgres=# \db
              List of tablespaces
    Name    | Owner |         Location
------------+-------+--------------------------
 fastspace  | omm   | tablespace/tablespace_1
 pg_default | omm   |
 pg_global  | omm   |
(3 rows)
```

1.1.2.1.3 查询表空间使用率

步骤 1　查询表空间的当前使用情况。

```
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。

```
postgres=# SELECT PG_TABLESPACE_SIZE('fastspace');
 pg_tablespace_size
--------------------
               4096
(1 row)
```

步骤 2　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。

按照公式就可以计算出来了。这里没有磁盘大小数据，故没有计算结果。

1.1.2.1.4 修改表空间

执行如下命令对表空间 fastspace 重命名为 fspace。

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

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

1.1.2.1.5 删除表空间

执行如下命令删除用户 jack。

```
postgres=# DROP USER jack CASCADE;
DROP ROLE
```

执行如下命令删除表空间 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。

```
postgres=# DROP USER jack CASCADE;
DROP ROLE
postgres=# DROP TABLESPACE fspace;
DROP TABLESPACE
```

1.1.3 创建和管理数据库

创建数据库

步骤 1　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
```

步骤 2　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
```

```
postgres=# CREATE TABLESPACE tpcds_local RELATIVE LOCATION 'tablespace/tablespace_2';
CREATE TABLESPACE
postgres=# CREATE DATABASE db_tpcc WITH TABLESPACE = tpcds_local;
CREATE DATABASE
```

管理数据库

1.1.3.1.2 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)
```

```
postgres=# \l
                        List of databases
   Name    | Owner | Encoding | Collate | Ctype | Access privileges
-----------+-------+----------+---------+-------+-------------------
 db_tpcc   | omm   | UTF8     | C       | C     |
 db_tpcc02 | omm   | UTF8     | C       | C     |
 postgres  | omm   | UTF8     | C       | C     |
 template0 | omm   | UTF8     | C       | C     | =c/omm            +
           |       |          |         |       | omm=CTc/omm
 template1 | omm   | UTF8     | C       | C     | =c/omm            +
           |       |          |         |       | omm=CTc/omm
(5 rows)

postgres=# SELECT datname FROM pg_database;
  datname
-----------
 template1
 template0
 db_tpcc02
 db_tpcc
 postgres
(5 rows)
```

1.1.3.1.3 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
```

```
postgres=# ALTER DATABASE db_tpcc SET search_path TO pa_catalog,public;
ALTER DATABASE
postgres=# ALTER DATABASE db_tpcc RENAME TO human_tpcds;
ALTER DATABASE
```

1.1.3.1.4 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
```

```
postgres=# DROP DATABASE human_tpcds;
DROP DATABASE
```

1.2 创建和管理表

1.2.1 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。

执行如下命令创建表。

```
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。

```
postgres=# CREATE TABLE customer_t1
postgres-# (
postgres(# c_customer_sk             integer,
postgres(# c_customer_id             char(5),
postgres(# c_first_name              char(6),
postgres(# c_last_name               char(8)
postgres(# );
CREATE TABLE
```

1.2.2 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;
```

```
postgres=# INSERT INTO customer_t1(c_customer_sk, c_customer_id, c_first_name) VALUES (37
69, 'hello', 'Grace');
INSERT 0 1
postgres=# INSERT INTO customer_t1 VALUES (3769, 'hello', 'Grace');
INSERT 0 1
postgres=# INSERT INTO customer_t1 (c_customer_sk, c_first_name) VALUES (3769, 'Grace');
INSERT 0 1
postgres=# INSERT INTO customer_t1 VALUES (3769, 'hello');
INSERT 0 1
postgres=# INSERT INTO customer_t1 (c_customer_sk, c_customer_id, c_first_name) VALUES (3
769, 'hello', DEFAULT);
INSERT 0 1
postgres=# INSERT INTO customer_t1 DEFAULT VALUES;
INSERT 0 1
```

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。

```
postgres=# INSERT INTO customer_t1 (c_customer_sk, c_customer_id, c_first_name) VALUES
postgres-# (6885, 'maps', 'Joes'),
postgres-# (4321, 'tpcds', 'Lily'),
postgres-# (9527, 'world', 'James');
INSERT 0 3
```

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
```

```
postgres=# CREATE TABLE customer_t2
postgres-# (
postgres(# c_customer_sk             integer,
postgres(# c_customer_id             char(5),
postgres(# c_first_name              char(6),
postgres(# c_last_name               char(8)
postgres(# );
CREATE TABLE
postgres=# INSERT INTO customer_t2 SELECT * FROM customer_t1;
INSERT 0 9
postgres=# DROP TABLE customer_t2 CASCADE;
DROP TABLE
```

1.2.3 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;
```

```
postgres=# UPDATE customer_t1 SET c_customer_sk = 9876 WHERE c_customer_sk = 9527;
UPDATE 1
postgres=# UPDATE customer_t1 SET c_customer_sk = c_customer_sk + 100;
UPDATE 9
postgres=# UPDATE customer_t1 SET  c_customer_id = 'Admin', c_first_name = 'Local' WHERE
c_customer_sk = 4421;
UPDATE 1
```

1.2.4 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;
```

```
postgres=# SELECT * FROM pg_tables;
     schemaname     |           tablename           | tableowner | tablespace | hasindexe
s | hasrules | hastriggers | tablecreator |            created            |         last_
ddl_time
--------------------+-------------------------------+------------+------------+----------
--+----------+-------------+--------------+-------------------------------+--------------
----------------
 pg_catalog         | pg_statistic                  | omm        |            | t
  | f        | f           |              |                               |
 pg_catalog         | pg_type                       | omm        |            | t
  | f        | f           |              |                               |
 public             | a                             | omm        |            | f
  | f        | f           | omm          | 2025-12-09 20:18:58.080509+08 | 2025-12-09 20
```

使用 gsql 的\d+命令查询表的结构。

```
postgres=# \d+ customer_t1;
```

```
postgres=# \d+ customer_t1;
                         Table "public.customer_t1"
    Column     |     Type     | Modifiers | Storage  | Stats target | Description
---------------+--------------+-----------+----------+--------------+-------------
 c_customer_sk | integer      |           | plain    |              |
 c_customer_id | character(5) |           | extended |              |
 c_first_name  | character(6) |           | extended |              |
 c_last_name   | character(8) |           | extended |              |
Has OIDs: no
Options: orientation=row, compression=no
```

执行如下命令查询表 customer_t1 的数据量。

```
postgres=# SELECT count(*) FROM customer_t1;
```

```
postgres=# SELECT count(*) FROM customer_t1;
 count
-------
     9
(1 row)
```

执行如下命令查询表 customer_t1 的所有数据。

```
postgres=# SELECT * FROM customer_t1;
```

```
postgres=# SELECT * FROM customer_t1;
 c_customer_sk | c_customer_id | c_first_name | c_last_name
---------------+---------------+--------------+-------------
          3869 | hello         | Grace        |
          3869 | hello         | Grace        |
          3869 |               | Grace        |
          3869 | hello         |              |
          3869 | hello         |              |
               |               |              |
          6985 | maps          | Joes         |
          9976 | world         | James        |
          4421 | Admin         | Local        |
(9 rows)
```

执行如下命令只查询字段 c_customer_sk 的数据。

```
postgres=# SELECT c_customer_sk FROM customer_t1;
```

```
postgres=# SELECT c_customer_sk FROM customer_t1;
 c_customer_sk
---------------
          3869
          3869
          3869
          3869
          3869

          6985
          9976
          4421
(9 rows)
```

执行如下命令过滤字段 c_customer_sk 的重复数据。

```
postgres=# SELECT DISTINCT( c_customer_sk ) FROM customer_t1;
```

```
postgres=# SELECT DISTINCT( c_customer_sk ) FROM customer_t1;
 c_customer_sk
---------------
          9976
          6985

          3869
          4421
(5 rows)
```

执行如下命令查询字段 c_customer_sk 为 3869 的所有数据。

```
postgres=# SELECT * FROM customer_t1 WHERE c_customer_sk = 3869;
```

```
postgres=# SELECT * FROM customer_t1 WHERE c_customer_sk = 3869;
 c_customer_sk | c_customer_id | c_first_name | c_last_name
---------------+---------------+--------------+-------------
          3869 | hello         | Grace        |
          3869 | hello         | Grace        |
          3869 |               | Grace        |
          3869 | hello         |              |
          3869 | hello         |              |
(5 rows)
```

执行如下命令按照字段 c_customer_sk 进行排序。

```
postgres=# SELECT * FROM customer_t1 ORDER BY c_customer_sk;
```

```
postgres=# SELECT * FROM customer_t1 ORDER BY c_customer_sk;
 c_customer_sk | c_customer_id | c_first_name | c_last_name
---------------+---------------+--------------+-------------
          3869 | hello         | Grace        |
          3869 | hello         | Grace        |
          3869 |               | Grace        |
          3869 | hello         |              |
          3869 | hello         |              |
          4421 | Admin         | Local        |
          6985 | maps          | Joes         |
          9976 | world         | James        |
               |               |              |
(9 rows)
```

执行如下命令查询 ROWNUM 伪列。

```
postgres=# SELECT rownum,c_customer_sk,c_customer_id FROM customer_t1;
```

```
postgres=# SELECT rownum,c_customer_sk,c_customer_id FROM customer_t1;
 rownum | c_customer_sk | c_customer_id
--------+---------------+---------------
      1 |          3869 | hello
      2 |          3869 | hello
      3 |          3869 |
      4 |          3869 | hello
      5 |          3869 | hello
      6 |               |
      7 |          6985 | maps
      8 |          9976 | world
      9 |          4421 | Admin
(9 rows)
```

执行如下命令使用别名进行查询(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;
```

```
postgres=# SELECT rownum CNB,T.c_customer_sk CSK,T.c_customer_id CID FROM customer_t1 T;
 cnb | csk  |  cid
-----+------+-------
   1 | 3869 | hello
   2 | 3869 | hello
   3 | 3869 |
   4 | 3869 | hello
   5 | 3869 | hello
   6 |      |
   7 | 6985 | maps
   8 | 9976 | world
   9 | 4421 | Admin
(9 rows)
```

1.2.5 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;
```

```
postgres=# DELETE FROM customer_t1 WHERE c_customer_sk = 3869;
DELETE 5
postgres=# TRUNCATE TABLE customer_t1;
TRUNCATE TABLE
postgres=# DROP TABLE customer_t1;
DROP TABLE
```

1.3 创建和管理其他数据库对象操作

1.3.1 创建和管理 schema

创建 schema

执行如下命令来创建一个 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
```

```
postgres=# CREATE SCHEMA myschema;
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。

```
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
```

```
postgres=# CREATE SCHEMA myschema AUTHORIZATION omm;
CREATE SCHEMA
```

管理 schema

1.3.1.1.1 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。

```
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。

执行如下命令查询 myschema 下 mytable 表的所有数据。

```
postgres=# SELECT * FROM myschema.mytable;
id | name
----+------
(0 rows)
```

```
postgres=# CREATE TABLE myschema.mytable(id int, name varchar(20));
CREATE TABLE
postgres=# SELECT * FROM myschema.mytable;
 id | name
----+------
(0 rows)
```

1.3.1.1.2 schema 的搜索路径

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。

执行如下命令查看搜索路径。

```
postgres=# SHOW SEARCH_PATH;
  search_path
----------------
  "$user",public
(1 row)
```

执行如下命令将搜索路径设置为 myschema、public，首先搜索 myschema。

```
postgres=# SET SEARCH_PATH TO myschema,public;
SET
```

```
postgres=# SHOW SEARCH_PATH;
  search_path
----------------
 "$user",public
(1 row)

postgres=# SET SEARCH_PATH TO myschema,public;
SET
```

1.3.1.1.3 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)
```

执行如下命令创建用户 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
```


```
postgres=# REVOKE CREATE ON SCHEMA public FROM PUBLIC;
REVOKE
postgres=# SELECT current_schema();
 current_schema
----------------
 myschema
(1 row)

postgres=# CREATE USER jack IDENTIFIED BY 'Bigdata@123';
CREATE ROLE
postgres=# GRANT USAGE ON schema myschema TO jack;
GRANT
postgres=# REVOKE USAGE ON schema myschema FROM jack;
REVOKE
```

1.3.1.1.4 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
```

执行如下命令删除用户 jack。

```
postgres=# DROP USER jack;
DROP ROLE
```

```
postgres=# DROP SCHEMA IF EXISTS nullschema;
NOTICE:  schema "nullschema" does not exist, skipping
DROP SCHEMA
postgres=# DROP SCHEMA myschema CASCADE;
NOTICE:  drop cascades to table mytable
DROP SCHEMA
postgres=# DROP USER jack;
DROP ROLE
```

1.3.2 创建和管理分区表

创建分区表

步骤 1　创建 schema。

```
CREATE SCHEMA tpcds;
```

步骤 2　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
```

```
postgres=# CREATE SCHEMA tpcds;
CREATE SCHEMA
postgres=# CREATE TABLESPACE example1 RELATIVE LOCATION 'tablespace1/tablespace_1';
CREATE TABLESPACE
postgres=# CREATE TABLESPACE example2 RELATIVE LOCATION 'tablespace2/tablespace_2';
CREATE TABLESPACE
postgres=# CREATE TABLESPACE example3 RELATIVE LOCATION 'tablespace3/tablespace_3';
CREATE TABLESPACE
postgres=# CREATE TABLESPACE example4 RELATIVE LOCATION 'tablespace4/tablespace_4';
CREATE TABLESPACE
```

步骤 3　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
```

```
postgres=# CREATE TABLE tpcds.web_returns_p2
postgres-# (
postgres(#     ca_address_sk       integer                 NOT NULL   ,
postgres(#     ca_address_id       character(16)           NOT NULL   ,
postgres(#     ca_street_number    character(10)                      ,
postgres(#     ca_street_name      character varying(60)              ,
postgres(#     ca_street_type      character(15)                      ,
postgres(#     ca_suite_number     character(10)                      ,
postgres(#     ca_city             character varying(60)              ,
postgres(#     ca_county           character varying(30)              ,
postgres(#     ca_state            character(2)                       ,
postgres(#     ca_zip              character(10)                      ,
postgres(#     ca_country           character varying(20)              ,
postgres(#     ca_gmt_offset       numeric(5,2)                       ,
postgres(#     ca_location_type    character(20)
postgres(# )
postgres-# TABLESPACE example1
postgres-# PARTITION BY RANGE (ca_address_sk)
postgres-# (
postgres(#         PARTITION P1 VALUES LESS THAN(5000),
postgres(#         PARTITION P2 VALUES LESS THAN(10000),
postgres(#         PARTITION P3 VALUES LESS THAN(15000),
postgres(#         PARTITION P4 VALUES LESS THAN(20000),
postgres(#         PARTITION P5 VALUES LESS THAN(25000),
postgres(#         PARTITION P6 VALUES LESS THAN(30000),
postgres(#         PARTITION P7 VALUES LESS THAN(40000),
postgres(#         PARTITION P8 VALUES LESS THAN(MAXVALUE) TABLESPACE example2
postgres(# )
postgres-# ENABLE ROW MOVEMENT;
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：

```
INSERT 0 4
```

```
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');
INSERT 0 4
```

管理分区表

1.3.2.1.2 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
```

```
postgres=# ALTER TABLE tpcds.web_returns_p2 DISABLE ROW MOVEMENT;
ALTER TABLE
```

1.3.2.1.3 删除分区

删除分区 P8。

```
postgres=# ALTER TABLE tpcds.web_returns_p2 DROP PARTITION P8;
ALTER TABLE
```

```
postgres=# ALTER TABLE tpcds.web_returns_p2 DROP PARTITION P8;
ALTER TABLE
```

1.3.2.1.4 增加分区

增加分区 P8，范围为 40000<= P8<=MAXVALUE。

```
postgres=# ALTER TABLE tpcds.web_returns_p2 ADD PARTITION P8 VALUES LESS THAN (MAXVALUE);
ALTER TABLE
```

```
postgres=# ALTER TABLE tpcds.web_returns_p2 ADD PARTITION P8 VALUES LESS THAN (MAXVALUE);

ALTER TABLE
```

1.3.2.1.5 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
```

```
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
```

```
postgres=# ALTER TABLE tpcds.web_returns_p2 RENAME PARTITION FOR (40000) TO P8;
ALTER TABLE
```

1.3.2.1.6 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
```

```
postgres=# ALTER TABLE tpcds.web_returns_p2 MOVE PARTITION P6 TABLESPACE example3;
ALTER TABLE
postgres=# ALTER TABLE tpcds.web_returns_p2 MOVE PARTITION P4 TABLESPACE example4;
ALTER TABLE
```

1.3.2.1.7 查询分区

查询分区 P1。

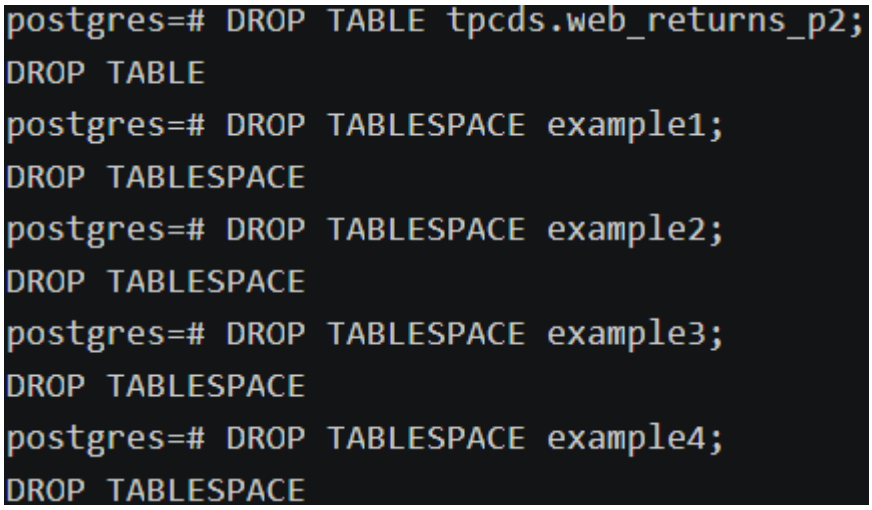
```
postgres=# SELECT * FROM tpcds.web_returns_p2 PARTITION (P1);
```

```
postgres=# SELECT * FROM tpcds.web_returns_p2 PARTITION (P1);
 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_gmt_offs
et |   ca_location_type
---------------+------------------+------------------+----------------+-----------------+
----------------+---------+-----------+----------+------------+------------+------------
---+----------------------
             1 | a                | 1                | a              | a               |
```

1.3.2.1.8 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
```



1.3.3 创建和管理索引

准备工作

步骤 1　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
```

```
postgres=# CREATE TABLESPACE example1 RELATIVE LOCATION 'tablespace1/tablespace_1';
CREATE TABLESPACE
postgres=# CREATE TABLESPACE example2 RELATIVE LOCATION 'tablespace2/tablespace_2';
CREATE TABLESPACE
postgres=# CREATE TABLESPACE example3 RELATIVE LOCATION 'tablespace3/tablespace_3';
CREATE TABLESPACE
postgres=# CREATE TABLESPACE example4 RELATIVE LOCATION 'tablespace4/tablespace_4';
CREATE TABLESPACE
```

步骤 2　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
```

```
postgres=# CREATE TABLE tpcds.web_returns_p2
postgres-# (
postgres(#     ca_address_sk       integer                 NOT NULL    ,
postgres(#     ca_address_id       character(16)           NOT NULL    ,
postgres(#     ca_street_number    character(10)                       ,
postgres(#     ca_street_name      character varying(60)               ,
postgres(#     ca_street_type      character(15)                       ,
postgres(#     ca_suite_number     character(10)                       ,
postgres(#     ca_city             character varying(60)               ,
postgres(#     ca_county           character varying(30)               ,
postgres(#     ca_state            character(2)                        ,
postgres(#     ca_zip              character(10)                       ,
postgres(#     ca_country           character varying(20)               ,
postgres(#     ca_gmt_offset       numeric(5,2)                        ,
postgres(#     ca_location_type    character(20)
postgres(# )
postgres-# TABLESPACE example1
postgres-# PARTITION BY RANGE (ca_address_sk)
postgres-# (
postgres(#         PARTITION P1 VALUES LESS THAN(5000),
postgres(#         PARTITION P2 VALUES LESS THAN(10000),
postgres(#         PARTITION P3 VALUES LESS THAN(15000),
postgres(#         PARTITION P4 VALUES LESS THAN(20000),
postgres(#         PARTITION P5 VALUES LESS THAN(25000),
postgres(#         PARTITION P6 VALUES LESS THAN(30000),
postgres(#         PARTITION P7 VALUES LESS THAN(40000),
postgres(#         PARTITION P8 VALUES LESS THAN(MAXVALUE) TABLESPACE example2
postgres(# )
postgres-# ENABLE ROW MOVEMENT;
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。

```
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
```

```
postgres=# CREATE INDEX tpcds_web_returns_p2_index1 ON tpcds.web_returns_p2 (ca_address_i
d) LOCAL;
CREATE INDEX
postgres=# CREATE INDEX tpcds_web_returns_p2_index2 ON tpcds.web_returns_p2 (ca_address_s
k) LOCAL
postgres-# (
postgres(#     PARTITION web_returns_p2_P1_index,
postgres(#     PARTITION web_returns_p2_P2_index TABLESPACE example3,
postgres(#     PARTITION web_returns_p2_P3_index TABLESPACE example4,
postgres(#     PARTITION web_returns_p2_P4_index,
postgres(#     PARTITION web_returns_p2_P5_index,
postgres(#     PARTITION web_returns_p2_P6_index,
postgres(#     PARTITION web_returns_p2_P7_index,
postgres(#     PARTITION web_returns_p2_P8_index
postgres(# ) TABLESPACE example2;
CREATE INDEX
postgres=#
```

管理索引

1.3.3.1.2 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
```

```
postgres=# ALTER INDEX tpcds.tpcds_web_returns_p2_index2 MOVE PARTITION web_returns_p2_P2
_index TABLESPACE example1;
ALTER INDEX
postgres=# ALTER INDEX tpcds.tpcds_web_returns_p2_index2 MOVE PARTITION web_returns_p2_P3
_index TABLESPACE example2;
ALTER INDEX
```

1.3.3.1.3 重命名索引分区

执行如下命令对索引分区 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
```

```
postgres=# ALTER INDEX tpcds.tpcds_web_returns_p2_index2 RENAME PARTITION web_returns_p2_P8_index TO web_returns_p2_P8_index_new;
ALTER INDEX
postgres=#
```

1.3.3.1.4 查询索引

执行如下命令查询系统和用户定义的所有索引。

```
postgres=# SELECT RELNAME FROM PG_CLASS WHERE RELKIND='i';
```

执行如下命令查询指定索引的信息。

```
postgres=# \di+ tpcds.tpcds_web_returns_p2_index1
```

```
postgres=# \di+ tpcds.tpcds_web_returns_p2_index1
                                        List of relations
 Schema |            Name             | Type  | Owner |     Table      | Size  | Storage | Description
--------+-----------------------------+-------+-------+----------------+-------+---------+-------------
 tpcds  | tpcds_web_returns_p2_index1 | index | omm   | web_returns_p2 | 88 kB |         |
(1 row)
```

1.3.3.1.5 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
```

```
postgres=# DROP INDEX tpcds.tpcds_web_returns_p2_index1;
DROP INDEX
postgres=# DROP INDEX tpcds.tpcds_web_returns_p2_index2;
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。

步骤 1　创建一个普通表。

```
postgres=# CREATE TABLE tpcds.customer_address_bak AS TABLE tpcds.web_returns_p2;
```

```
postgres=# CREATE TABLE tpcds.customer_address_bak AS TABLE tpcds.web_returns_p2;
INSERT 0 4
```

步骤 2　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;
```

```
postgres=# SELECT ca_address_sk FROM tpcds.customer_address_bak WHERE ca_address_sk=14888
;
 ca_address_sk
---------------
         14888
(1 row)
```

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
```

```
postgres=# CREATE INDEX index_wr_returned_date_sk ON tpcds.customer_address_bak (ca_addre
ss_sk);
CREATE INDEX
```

步骤 3　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;
```

```
postgres=# SELECT ca_address_sk,ca_address_id FROM tpcds.customer_address_bak WHERE ca_ad
dress_sk = 5050 AND ca_street_number < 1000;
 ca_address_sk |  ca_address_id
---------------+------------------
          5050 | c
(1 row)
```

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
```

```
postgres=# CREATE INDEX more_column_index ON tpcds.customer_address_bak(ca_address_sk ,c
_street_number);
CREATE INDEX
```

步骤 4　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
```

```
postgres=# CREATE INDEX part_index ON tpcds.customer_address_bak(ca_address_sk) WHERE ca_
address_sk = 5050;
CREATE INDEX
```

步骤 5　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;
```

```
postgres=#  SELECT * FROM tpcds.customer_address_bak WHERE trunc(ca_street_number) < 1000
;
 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_gmt_offs
et |   ca_location_type
---------------+------------------+------------------+----------------+-----------------+
-----------------+---------+-----------+----------+------------+------------+------------
---+---------------------
             1 | a                | 1                | a              | a               |
 a               | a       | a         | a        | a          | a          |         1.
```

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
```

```
postgres=# CREATE INDEX para_index ON tpcds.customer_address_bak (trunc(ca_street_number)
);
CREATE INDEX
```

步骤 6　删除 tpcds.customer_address_bak 表。

```
postgres=# DROP TABLE tpcds.customer_address_bak;
DROP TABLE
```

```
postgres=# DROP TABLE tpcds.customer_address_bak;
DROP TABLE
```

1.3.4 创建和管理视图

创建视图

执行如下命令创建普通视图 MyView。

```
postgres=# CREATE OR REPLACE VIEW MyView AS SELECT * FROM tpcds.web_returns_p2 WHERE
trunc(ca_address_sk) > 10000;
CREATE VIEW
```

执行如下命令创建物化视图 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。

```
postgres=# CREATE OR REPLACE VIEW MyView AS SELECT * FROM tpcds.web_returns_p2 WHERE trun
c(ca_address_sk) > 10000;
CREATE VIEW
postgres=# CREATE MATERIALIZED VIEW MV_MyView AS SELECT * FROM tpcds.web_returns_p2 WHERE
 trunc(ca_address_sk) > 5000;
CREATE MATERIALIZED VIEW
```

管理视图

1.3.4.1.1 查询普通视图

执行如下命令查询 MyView 视图。

```
postgres=# SELECT * FROM MyView;
```

```
postgres=#  SELECT * FROM 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_gmt_offs
et |   ca_location_type
---------------+------------------+------------------+----------------+-----------------+
----------------+---------+-----------+----------+------------+------------+------------
---+----------------------
        14888 | d                | 400              | d              | d               |
 d               | d       | d         | d        | d          | d          |          1.
50 | d
(1 row)
```

1.3.4.1.2 查看普通图的具体信息

执行如下命令查询 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;
```

```
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 precision;
```

1.3.4.1.3 查询物化视图

执行如下命令查询 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)
```

```
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_gmt_offs
et |   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)
```

1.3.4.1.4 查看物化图的具体信息

执行如下命令查询 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
```

给基表新增二条记录，然后刷新物化视图。

```
postgres=# \d+ MV_MyView
                          Materialized view "public.mv_myview"
      Column      |         Type          | Modifiers | Storage  | Stats target | Descrip
tion
------------------+-----------------------+-----------+----------+--------------+--------
-----
 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 |              |
```

```
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
```

```
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',
5, 'd');
INSERT 0 2
```

1.3.4.1.5 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，发现多了二条记录。

```
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)
```

```
postgres=# REFRESH MATERIALIZED VIEW MV_MyView;
REFRESH MATERIALIZED VIEW
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_gmt_offs
et |   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)
```

1.3.4.1.6 删除视图

执行如下命令删除视图。

```
postgres=# DROP VIEW MyView;
DROP VIEW
postgres=# DROP MATERIALIZED VIEW MV_MyView;
DROP MATERIALIZED VIEW
```

```
postgres=# DROP VIEW MyView;
DROP VIEW
postgres=# DROP MATERIALIZED VIEW MV_MyView;
DROP MATERIALIZED VIEW
```

1.3.5 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
```

```
postgres=#  CREATE TABLE T1
postgres-# (id    serial,
postgres(# name  text
postgres(# );
NOTICE:  CREATE TABLE will create implicit sequence "t1_id_seq" for serial column "t1.id"
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。

步骤 1　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
```

步骤 2　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
```

步骤 3　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
```

```
postgres=# CREATE SEQUENCE seq1 cache 100;
CREATE SEQUENCE
postgres=# CREATE TABLE T2
postgres-# (
postgres(#     id    int not null default nextval('seq1'),
postgres(#     name text
postgres(# );
CREATE TABLE
postgres=# ALTER SEQUENCE seq1 OWNED BY T2.id;
ALTER SEQUENCE
```

删除序列

```
DROP SEQUENCE seq1 CASCADE;
```

```
postgres=# DROP SEQUENCE seq1 CASCADE;
NOTICE:  drop cascades to default for table t2 column id
DROP SEQUENCE
```

1.3.6 创建和管理存储过程

创建存储过程

步骤 1　创建表 t_test。

```
postgres=# create table t_test(c1 int, c2 int);
```

步骤 2　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;
/
```

步骤 3　调用存储过程。

```
call insert_data();
```

步骤 4　查询表内容。

```
postgres=# select * from t_test;
 c1 | c2
----+----
  1 |  2
  2 |  1
(2 rows)
```

```
postgres=# create table t_test(c1 int, c2 int);
CREATE TABLE
postgres=# create or replace procedure insert_data
postgres-# is
postgres$# a int;
postgres$# b int;
postgres$# begin
postgres$# a=1;
postgres$# b=2;
postgres$# insert into t_test values(a,b);
postgres$# insert into t_test values(b,a);
postgres$# end;
postgres$# /
CREATE PROCEDURE
postgres=# call insert_data();
 insert_data
-------------

(1 row)

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：

```
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;
```

```
postgres=# \sf insert_data
CREATE OR REPLACE PROCEDURE public.insert_data()
AS  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;
/
postgres=# drop procedure insert_data;
DROP PROCEDURE
```

1.3.7 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。

步骤 1　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
```

步骤 2　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)
```

```
postgres=# CREATE GLOBAL TEMPORARY TABLE  t_test2(
postgres(# id integer,
postgres(# lbl text
postgres(# ) ON COMMIT PRESERVE ROWS;
CREATE TABLE
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)
```

步骤 3　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)
```

```
postgres=# \q
[omm@ecs-ac18 ~]$ gsql -d postgres -p 26000 -r
gsql ((openGauss 1.1.0 build 392c0438) compiled at 2020-12-31 20:08:06 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)
```

步骤 4　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。

```
postgres=# select * from t_test2;
 id | lbl
----+-----
(0 rows)
```

步骤 5　删除临时表。

```
postgres=# drop table t_test2;
DROP TABLE
```

```
postgres=# drop table t_test2;
DROP TABLE
```

1.3.8 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
```

```
postgres=# CREATE GLOBAL TEMPORARY TABLE  t_test3(
postgres(# id integer,
postgres(# lbl text
postgres(# ) ON COMMIT DELETE ROWS;
CREATE TABLE
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)

postgres=# commit;
COMMIT
postgres=# select * from t_test3;
 id | lbl
----+-----
(0 rows)

postgres=# drop table t_test3;
DROP TABLE
```

1.4 学校数据模型

1.4.1 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 语法使用的介绍。

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。

1.4.2 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
- 一位老师可以选择多门课程，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

1.4.3 E-R 图

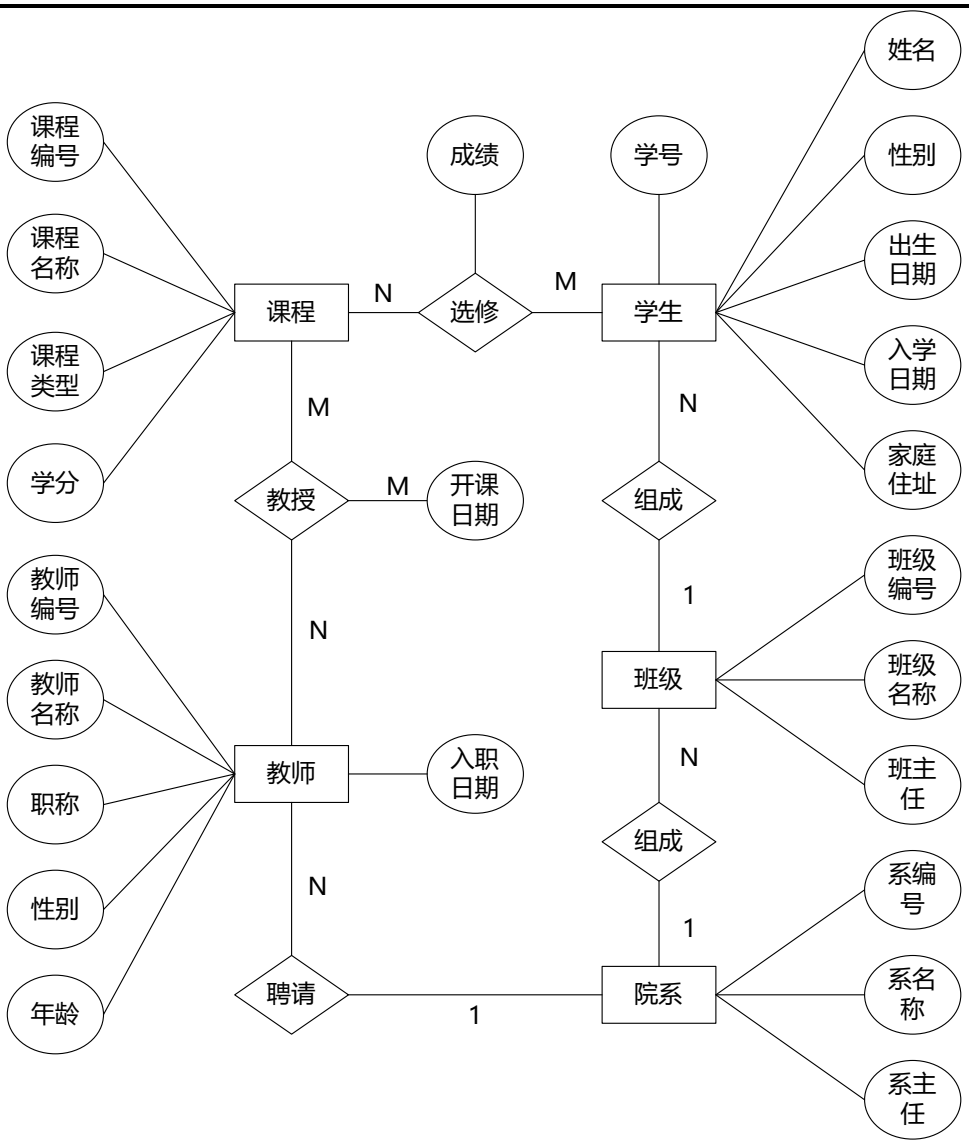


图1-1 E-R 图

1.5 学校数据模型表操作

1.5.1 表的创建

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：

步骤 1　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)
);
```

步骤 2　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
);
```

```
postgres=# DROP TABLE IF EXISTS student;
NOTICE:  table "student" does not exist, skipping
DROP TABLE
postgres=# CREATE TABLE student
postgres-# (
postgres(#         std_id INT PRIMARY KEY,
postgres(#         std_name NCHAR(20) NOT NULL,
postgres(#         std_sex NCHAR(6),
postgres(#         std_birth DATE,
postgres(#         std_in DATE NOT NULL,
postgres(#         std_address VARCHAR(100)
postgres(# );
NOTICE:  CREATE TABLE / PRIMARY KEY will create implicit index "student_pkey" for table "
student"
CREATE TABLE
postgres=# DROP TABLE IF EXISTS teacher;
NOTICE:  table "teacher" does not exist, skipping
DROP TABLE
postgres=# CREATE TABLE teacher
postgres-# (
postgres(#         tec_id INT PRIMARY KEY,
postgres(#         tec_name CHAR(20) NOT NULL,
postgres(#         tec_job CHAR(15),
postgres(#         tec_sex CHAR(6),
postgres(#         tec_age INT,
postgres(#         tec_in DATE NOT NULL
postgres(# );
NOTICE:  CREATE TABLE / PRIMARY KEY will create implicit index "teacher_pkey" for table "
teacher"
CREATE TABLE
```

步骤 3　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
);
```

步骤 4　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
);
```

步骤 5　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
);
```

```
postgres=# CREATE TABLE class
postgres-# (
postgres(#         cla_id INT PRIMARY KEY,
postgres(#         cla_name CHAR(20) NOT NULL,
postgres(#         cla_teacher INT NOT NULL
postgres(# );
NOTICE:  CREATE TABLE / PRIMARY KEY will create implicit index "class_pkey" for table "cl
ass"
CREATE TABLE
postgres=# DROP TABLE IF EXISTS school_department;
NOTICE:  table "school_department" does not exist, skipping
DROP TABLE
postgres=# CREATE TABLE school_department
postgres-# (
postgres(#         depart_id INT PRIMARY KEY,
postgres(#         depart_name NCHAR(30) NOT NULL,
postgres(#         depart_teacher INT NOT NULL
postgres(# );
NOTICE:  CREATE TABLE / PRIMARY KEY will create implicit index "school_department_pkey" f
or table "school_department"
CREATE TABLE
postgres=# DROP TABLE IF EXISTS course;
NOTICE:  table "course" does not exist, skipping
DROP TABLE
postgres=# CREATE TABLE course
postgres-# (
postgres(#         cor_id INT PRIMARY KEY,
postgres(#         cor_name NCHAR(30) NOT NULL,
postgres(#         cor_type NCHAR(20),
postgres(#         credit numeric
postgres(# );
NOTICE:  CREATE TABLE / PRIMARY KEY will create implicit index "course_pkey" for table "c
ourse"
CREATE TABLE
```

1.5.2 表数据的插入

步骤 1　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');
```

步骤 2　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');
```

步骤 3　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);
```

步骤 4　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);
```

步骤 5　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);
```

数据插入过程部分展示：

```
postgres=# INSERT INTO course(cor_id,cor_name,cor_type,credit) VALUES (19,'解剖学','必修'
,4);
INSERT 0 1
postgres=# INSERT INTO course(cor_id,cor_name,cor_type,credit) VALUES (20,'3D max','选修'
,2);
INSERT 0 1
```

1.5.3 数据查询

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
--------+------------------------------------+-------------------------+--------
      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
 1 | C 语言程序设计          | 必修 |  3.5
(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)
```

1.5.4 数据的修改和删除

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)
```

```
postgres=# SELECT * from course;
 cor_id |            cor_name            |      cor_type      | credit
--------+--------------------------------+--------------------+--------
      1 | 数据库系统概论                 | 必修               |      3
      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)

postgres=# SELECT * FROM course WHERE cor_id=1;
 cor_id |        cor_name          |      cor_type      | credit
--------+--------------------------+--------------------+--------
      1 | 数据库系统概论           | 必修               |      3
(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;
```

```
 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。

```
postgres=# DELETE FROM school_department WHERE depart_teacher=8 OR depart_teacher=15;
DELETE 2
postgres=# SELECT * FROM school_department;
 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)
```

三、心得总结（写出自己在完成实验过程中遇到的问题、解决方法，以及体会、收获等）（10%）

遇到的第一个问题：
实验过程中停了一次，后来发现服务器服务关掉了。打开后顺利进行了实验。

第二个问题：
建表或其他 CREATE 的时候，发现先前有同名内容，不知道是什么时候建立的，删除又建解决了问题。