-- 创建数据库

-- create table test(

--

-- id int PRIMARY key not NULL,

-- name text not NULL,

-- age int not NULL,

-- address char(50),

-- salary REAL

-- );

-- 插入语句

-- INSERT INTO test(id, name, age,address,salary) values (1,'paul',24,'californial',2000.00)

-- INSERT INTO test (id,name,age,address) VALUES (2,'ruso',25,'china')

-- INSERT INTO test(id,name,age,address,salary) VALUES (3,'Ami',26,DEFAULT,200.00)

-- 插入多行数据

-- INSERT INTO test(id,name,age,address,salary) VALUES (4,'lucy',24,'India',600.00),(5,'lina',22,'China',900.3)

-- 查询语句

-- SELECT \* from test

-- SELECT id,name,age from test

-- SELECT name,age from test where id=2

-- SELECT name from test where age!=24 and salary>500

-- select \* from test where age=24 or salary<500

-- SELECT \* from test where address IS not null

-- SELECT name,age, address from test as addition

-- select avg(age) from test

-- SELECT sum(id) from test where salary>500

-- SELECT sum(id),count(\*) from test

-- SELECT CURRENT\_DATE,CURRENT\_TIME,CURRENT\_ROLE,CURRENT\_USER,CURRENT\_SCHEMA

-- 查询姓名以l开头的用户信息

-- select \* from test where name like 'l%'

-- 查询年龄为22或25的用户信息

-- SELECT \* from test where age in(22,25)

-- SELECT \* from test where age not in(22,24)

-- 查询年龄在22--26之间的用户信息

-- SELECT \* from test where age BETWEEN 22 and 26

-- test 表中存在salary 大于500的，则查询所有用户的id

-- SELECT id from test where EXISTS (SELECT \* from test where salary>500)

-- select \* from test where age >(SELECT age from test where salary=600)

-- 更新字段值

-- UPDATE test set salary =500 where id=3

-- UPDATE test set age=19,salary=500,address='california' where id=3

-- SELECT \* from test where id=3

-- insert into test (id,name,age,address,salary) VALUES (6,'sa',20,DEFAULT,DEFAULT);

-- select \* from test where id=6;

-- DELETE from test where id=6;

-- select \* from test where id=6

-- 查询name中包含字母a的用户信息

-- SELECT \* from test where name like '%a%'

-- select \* from test where name like '%na';

-- like 只能对用于对字符数据进行比较

-- SELECT \* from test where name like '\_u%'

-- 偏移两个，从第三个开始，读取两条记录

-- SELECT \* from test limit 2 offset 3

-- ORDER BY 用于对一列或多列数据进行生序（ASC）或降序（DESC）排列

-- SELECT \* from test ORDER BY id asc；

-- 对name和salary进行降序排列

-- SELECT \* from test ORDER BY name,salary desc

-- GROUP BY和select 一起使用，用来对相同的数据进行分组，必须放在order by 子句之前

-- insert into test (id,name,age,address,salary) VALUES (6,'lina',22,DEFAULT,100),(7,'ruso',25,DEFAULT,250),(8,'Ami',19,DEFAULT,20);

-- SELECT \* FROM test

-- 通过name进行分组，查询相同name用户的salary总和

-- SELECT name,sum(salary) from test GROUP BY name

-- SELECT name,sum(salary) from test GROUP BY name ORDER BY sum(salary) asc --生序asc

-- WITH cte as (SELECT id,name,age,address from test) SELECT \* from cte

-- with RECURSIVE t(n) as (VALUES(0) union all

-- SELECT salary from test where salary>500

-- ) SELECT sum(n) from t

-- 创建和test拥有一致表结构的test1

-- CREATE TABLE test1(

-- id int primary key not null,

-- name text not null,

-- age int not null,

-- address char(50),

-- salary real

-- );

-- with 语句实现将删除记录存放在move\_rows中，并插入到test1 中。实现将test 中的数据选择性删除并转移到test1 中

-- with move\_rows as (

-- DELETE from test

-- where

-- salary>1000

-- returning \*

-- )

-- insert into test1 (SELECT \* from move\_rows);

-- select \* from test;

-- select \* from test1

-- HAVING 可用于筛选group by 处理后的数据上。

-- HAVING 用于在 GROUP BY 子句创建的分组上设置条件。

-- WHERE 用于在所选列上设置条件

-- SELECT \* from test;

-- 选择姓名出现次数小于两次的用户名. SELECT 后的参数必须和 group by 后的参数一致

-- SELECT name from test GROUP BY name HAVING count(name)<2

-- DISTINCT 去除重复项，只获取唯一一次记录

-- select name from test;

-- select distinct name from test

-- decimal, numeric, money

-- CREATE TABLE test2(

-- id int PRIMARY key not null,

-- dept char(50) not null,

-- emp\_id int not null

-- );

-- insert into test2 VALUES (1,'it',1),(2,'he',2);

-- SELECT \* from test2

-- 交叉连接（CROSS JOIN）把第一个表的每一行与第二个表的每一行进行匹配。如果两个输入表分别有 x 和 y 行，则结果表有 x\*y 行。

--

-- 由于交叉连接（CROSS JOIN）有可能产生非常大的表，使用时必须谨慎，只在适当的时候使用它们。

-- SELECT

-- emp\_id,

-- name,

-- address

-- FROM

-- test

-- CROSS JOIN test2

-- 内连接

-- SELECT \* from test inner join test2 on test.id=test2.emp\_id;

-- SELECT \* from test left join test2 on test.id=test2.emp\_id;

-- select \* from test RIGHT JOIN test2 on test.id=test2.emp\_id;

-- CREATE TABLE test\_audit(

-- id int primary key not null ,

-- emp\_id int not null,

-- entry\_date TEXT not NULL

-- );

-- 在test上创建触发器

--

-- CREATE or replace function auditlogfunc() returns trigger as $example\_table$

-- begin

-- insert into test\_audit(emp\_id, entry\_date) values (new.ID,CURRENT\_TIMESTAMP);

-- return new;

-- end;

-- $example\_table$ language plpgsql;

--

-- CREATE TRIGGER example\_trigger after insert on test for each row execute procedure auditlogfunc();

-- alter table test\_audit drop id;

-- INSERT into test VALUES (9,'doub',26,'america',600),(10,'firs',27,'American',700);

-- select \* from test\_audit

-- 列出数据库中的所有触发器

-- select \* from pg\_trigger

-- 删除test表上的触发器example\_trigger

-- drop trigger example\_trigger on test

-- PostgreSql 索引。（加速查询）。有助于加快select查询和where子句，但会减慢update 和insert 语句的数据输入。

-- 创建单列唯一索引

-- CREATE UNIQUE index test\_index on test(id);

-- create index salary\_index on test(salary)

-- SELECT \* from pg\_indexs ??

-- drop index salary\_index

-- 组合索引

-- create index test\_index on test (name, age);

-- SELECT \* from pg\_index

-- drop index test\_index

-- 使用索引时，需要考虑下列准则：

--

-- 索引不应该使用在较小的表上。

-- 索引不应该使用在有频繁的大批量的更新或插入操作的表上。

-- 索引不应该使用在含有大量的 NULL 值的列上。

-- 索引不应该使用在频繁操作的列上

-- alter table test add gender char(1);

-- select \* from test

-- alter table test drop gender

-- SELECT \* from test

-- drop 彻底从数据库删除表。

-- truncate 只删除数据，保留结构 。不扫描表，故速度快。可立即释放表空间

-- SELECT \* from test1;

-- drop table test1

-- truncate table test1

-- CREATE VIEW test\_view AS SELECT id,name,age from test

-- select \* from test\_view

-- drop view test\_view

-- SELECT \* from test where age = 25;

-- begin transaction;

-- delete from test where age=25;

-- commit;

-- lock语句只在事务模式下工作

-- begin;

-- lock table test in access exclusive mode;

--

-- commit;