数据库原理实验报告

题目：面向教学的程序设计类课程在线评测系统

学号：19071125

姓名：卢雨轩

指导教师：杜金莲

日期：2021.12.21

评语：

1：设计部分：

2：上机部分：

3：其它部分：

总分：

目录

相关说明

实验环境:

PostgreSQL 版本：`psql (PostgreSQL) 12.9 (Ubuntu 12.9-0ubuntu0.20.04.1)`
openGauss 版本：`gsql (openGauss 2.0.0 build 78689da9)`

数据库设计

本实验涉及的数据库设计源自EduOJ后端数据库的用户、权限、班级管理部分。EduOJ的数据库设计由卢雨轩（本实验报告的作者）和孙天天共同完成。复用以往项目作为实验设计的行为已经经过指导老师授权。

数据库设计

一、 数据需求描述

0.1.1 管理员:

- 用户增删改查
- 班级增删改查
- 作业增删改查
- 成绩查询、修改

用户相关

- 姓名，学号，密码，邮箱
- 权限相关：
  - 『全局权限』（如：某个用户有权限创建用户、修改用户、创建班级、创建题目）
  - 『针对权限』（如：某个用户有针对 id 为 5 的班级的添加学生权限）
  - 希望能够『批量管理』：把权限授予给『角色』，让『用户』拥有『角色』。
- 统计信息，加快查询速度

班级相关

- 课程名称，课堂名称，管理老师，课程描述
- 用户、作业
  - 和用户是多对多关系
  - 和作业是多对多关系

作业相关

- 标题，起止日期
- 多道题目
- 统计成绩

成绩查询、修改

- 用户、班级、作业、成绩
- 根据用户做题记录生成，用于加快计算，**有数据冗余**。

二、 数据库设计

0.2.1 ER 图

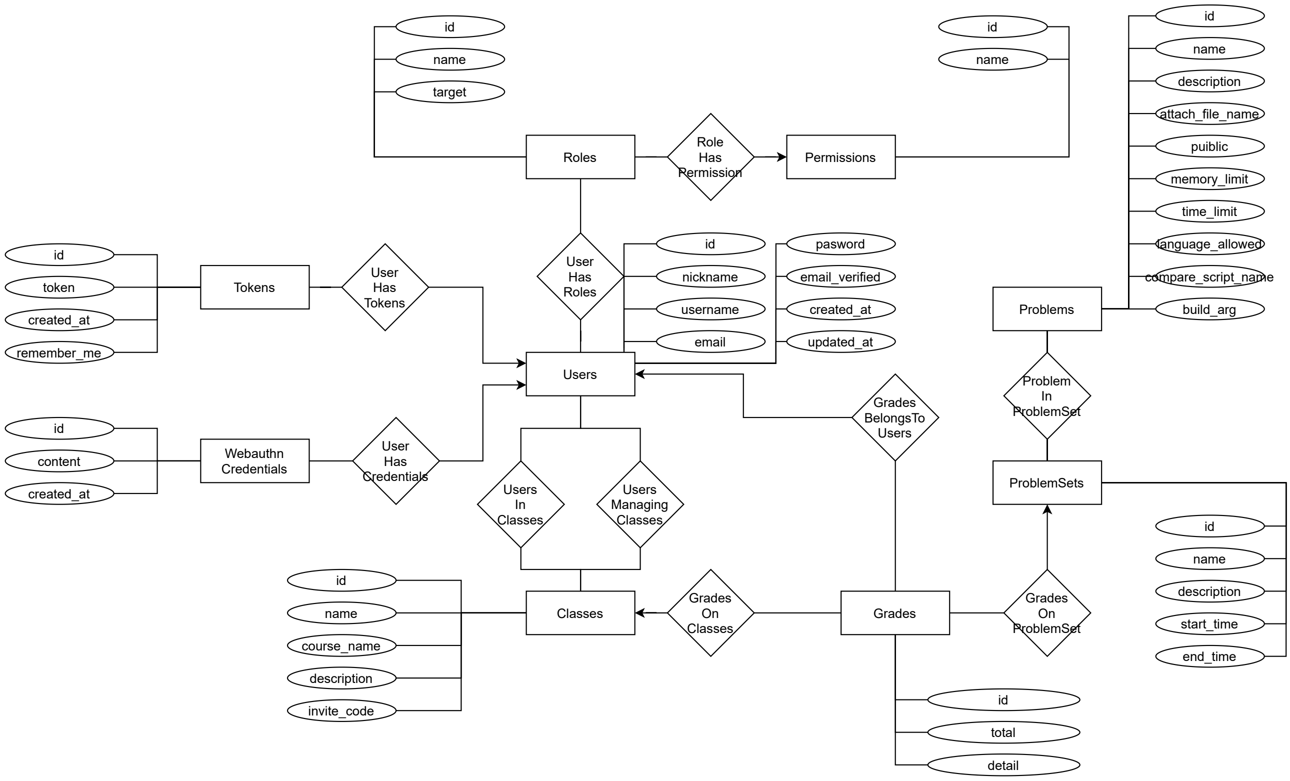

图 1: EduOJ 用户、班级、作业、权限管理部分 ER 图

0.2.2 关系模式

- user(id, nickname, username, email, password, created_at, updated_at)
- roles(id, name, target)
- user_has_roles(id, user_id,)
- permissions(id,role_id, name)
- tokens(id,user_id,token, created_at,remember_me)
- webauthn_credentials(id, user_id, content, created_at)
- classes(id, name, course_name, description, invite_code)
- user_in_classes(id, user_id, class_id)
- user_managing_classes(id, user_id, class_id)
- grades(id, total, detail, user_id, class_id, problem_set_id)
- problem_sets(id, name, description, start_time,end_time)
- problem_in_problem_sets(id, problem_id, problem_set_id)
- problems(id, name, description, attach_file_name, public, memory_limit, time_limit, compare_script_name, build_arg)

0.2.3 范式判断

1NF

所有关系模式中，属性均是原子的，符合范式。

2NF

除了 grades 的所有关系模式中均依赖主键id，符合范式。
grades 中，class_id 依赖 problem_set_id、detail 和 total 依赖评测结果（未给出），但是为了加速查询，保留数据冗余。

3NF

表中除了主键之外所有属性均不互相依赖，符合范式。

BCNF

所有关系模式均只有一个主属性，不存在其他键码，同时非主属性也依赖与键码，所以符合范式。

4NF

所有关系模式均不存在平凡多值依赖，故符合 4NF。

三、 数据表设计

0.3.1 users

字段名称	类型	索引	外键
id	bigint	primary	
username	varchar(30)	index	
nickname	varchar(30)	index	
email	varchar(320)	index	
password	varchar(60)		
created_at	timestamp		
updated_at	timestamp		

roles

字段名称	类型	索引	外键
id	bigint	primary	
name	varchar(255)		
target	varchar(255)	index	

user_has_roles

字段名称	类型	索引	外键
id	bigint	primary	
user_id	bigint	index	users(id)

role_id	bigint	index	roles(id)
target_id	bigint	index	

0.3.2 permissions

字段名称	类型	索引	外键
id	bigint	primary	
role_id	bigint	index	roles(id)
name	varchar(255)	index	

0.3.3 tokens

字段名称	类型	索引	外键
id	bigint	primary	
user_id	bigint	index	users(id)
token	varchar(32)	index	
created_at	timestamp		
remember_me	boolean		

0.3.4 webauthn_credentials

字段名称	类型	索引	外键
id	bigint	primary	
user_id	bigint	index	users(id)
content	varchar(32)		
created_at	timestamp		

0.3.5 classes

字段名称	类型	索引	外键
id	bigint	primary	
name	varchar(255)		
course_name	varchar(255)		
description	text		
invite_code	varchar(255)	index	

0.3.6 user_in_classes

字段名称	类型	索引	外键
id	bigint	primary	
user_id	bigint	index	users(id)
class_id	bigint	index	classes(id)

0.3.7 user_managing_classes

字段名称	类型	索引	外键
id	bigint	primary	
user_id	bigint	index	users(id)
class_id	bigint	index	classes(id)

0.3.8 grades

字段名称	类型	索引	外键
id	bigint	primary	
total	bigint		
detail	JSON		
user_id	bigint	index	users(id)
class_id	bigint	index	classes(id)

0.3.9 problem_sets

字段名称	类型	索引	外键
id	bigint	primary	
class_id	bigint	index	classes(id)
name	varchar(255)		
description	text		
start_time	timestamp		
end_time	timestamp		
created_at	timestamp		
updated_at	timestamp		

0.3.10 problem_in_problem_sets

字段名称	类型	索引	外键
id	bigint	primary	
problem_id	bigint	index	problems(id)
problem_set_id	bigint	index	problem_sets(id)

0.3.11 problems

字段名称	类型	索引	外键
id	bigint	primary	
name	varchar(255)		
description	text		
attach_file_name	varchar(255)		
public	boolean		

memory_limit	bigint	
time_limit	bigint	
build_arg	varchar(2047)	
compare_script_name	text	

实验一：　创建和删除数据库

一、　实验目的

本实验要求使用这两种方法 SQL 语句创建和删除数据库，实验目的在于：

1. 学习使用 SQL 语句建立与管理数据库。
2. 学会 SQL 语句的排错技术。
3. 了解数据文件、日志文件等相关概念。
4. 建立案例数据库以及自己设计的数据库，为以后的实验做准备。
5. 对常见错误操作，进行测试，加深对数据库管理相关语句以及操作的理解。

二、　实验步骤

1.2.1　新建数据库

查看当前数据库情况

使用\l命令查看当前数据库情况，运行结果如下所示：

SQL code

```
\l
```

Result on PostgreSQL

```
                                       List of databases
       Name        |  Owner   | Encoding |   Collate   |    Ctype    |   Access privileges
-------------------+----------+----------+-------------+-------------+-----------------------
 db_exp            | leo      | UTF8     | zh_CN.UTF-8 | zh_CN.UTF-8 |
 dbexp             | leo      | UTF8     | zh_CN.UTF-8 | zh_CN.UTF-8 |
 dbexp123          | leo      | UTF8     | zh_CN.UTF-8 | zh_CN.UTF-8 |
 eduoj             | postgres | UTF8     | zh_CN.UTF-8 | zh_CN.UTF-8 |
 eduoj_2020_2021_2 | leo      | UTF8     | zh_CN.UTF-8 | zh_CN.UTF-8 |
 postgres          | postgres | UTF8     | zh_CN.UTF-8 | zh_CN.UTF-8 |
 solar             | solar    | UTF8     | zh_CN.UTF-8 | zh_CN.UTF-8 | =Tc/solar            +
                   |          |          |             |             | solar=CTc/solar
 template0         | postgres | UTF8     | zh_CN.UTF-8 | zh_CN.UTF-8 | =c/postgres          +
                   |          |          |             |             | postgres=CTc/postgres
 template1         | postgres | UTF8     | zh_CN.UTF-8 | zh_CN.UTF-8 | =c/postgres          +
                   |          |          |             |             | postgres=CTc/postgres
 test              | leo      | UTF8     | zh_CN.UTF-8 | zh_CN.UTF-8 |
(10 rows)
```

Result on OpenGauss

```
                          List of databases
   Name    | Owner  | Encoding | Collate | Ctype | Access privileges
-----------+--------+----------+---------+-------+-------------------
 db_exp    | dbtest | UTF8     | C       | C     |
 dbexp     | dbtest | UTF8     | C       | C     |
 postgres  | omm    | UTF8     | C       | C     | =Tc/omm          +
           |        |          |         |       | omm=CTc/omm      +
           |        |          |         |       | leo=CTc/omm      +
           |        |          |         |       | leo=APm/omm      +
           |        |          |         |       | dbtest=CTc/omm   +
           |        |          |         |       | dbtest=APm/omm
 template0 | omm    | UTF8     | C       | C     | =c/omm           +
           |        |          |         |       | omm=CTc/omm
 template1 | omm    | UTF8     | C       | C     | =c/omm           +
           |        |          |         |       | omm=CTc/omm
(5 rows)
```

使用 SQL 语句创建数据库

使用 `create database` 命令创建数据库，运行结果如下所示：

SQL code

```
create database dbexp;
```

Result on PostgreSQL

```
CREATE DATABASE
```

Result on OpenGauss

```
CREATE DATABASE
```

观察数据库变化

使用`\l`命令查看当前数据库情况，运行结果如下所示：

SQL code

```
\l
```

Result on PostgreSQL

```
                                      List of databases
       Name        |  Owner   | Encoding |   Collate   |    Ctype    |   Access privileges
-------------------+----------+----------+-------------+-------------+-----------------------
 db_exp            | leo      | UTF8     | zh_CN.UTF-8 | zh_CN.UTF-8 |
 dbexp             | leo      | UTF8     | zh_CN.UTF-8 | zh_CN.UTF-8 |
 dbexp123          | leo      | UTF8     | zh_CN.UTF-8 | zh_CN.UTF-8 |
 eduoj             | postgres | UTF8     | zh_CN.UTF-8 | zh_CN.UTF-8 |
 eduoj_2020_2021_2 | leo      | UTF8     | zh_CN.UTF-8 | zh_CN.UTF-8 |
 postgres           | postgres | UTF8     | zh_CN.UTF-8 | zh_CN.UTF-8 |
 solar              | solar    | UTF8     | zh_CN.UTF-8 | zh_CN.UTF-8 | =Tc/solar           +
                    |          |          |             |             | solar=CTc/solar
 template0          | postgres | UTF8     | zh_CN.UTF-8 | zh_CN.UTF-8 | =c/postgres         +
                    |          |          |             |             | postgres=CTc/postgres
 template1          | postgres | UTF8     | zh_CN.UTF-8 | zh_CN.UTF-8 | =c/postgres         +
                    |          |          |             |             | postgres=CTc/postgres
 test               | leo      | UTF8     | zh_CN.UTF-8 | zh_CN.UTF-8 |
(10 rows)
```

Result on OpenGauss

```
                          List of databases
   Name    | Owner  | Encoding | Collate | Ctype  | Access privileges
-----------+--------+----------+---------+--------+-------------------
 db_exp    | dbtest | UTF8     | C       | C      |
 dbexp     | dbtest | UTF8     | C       | C      |
 postgres  | omm    | UTF8     | C       | C      | =Tc/omm          +
           |        |          |         |        | omm=CTc/omm      +
           |        |          |         |        | leo=CTc/omm      +
           |        |          |         |        | leo=APm/omm      +
           |        |          |         |        | dbtest=CTc/omm   +
           |        |          |         |        | dbtest=APm/omm
 template0 | omm    | UTF8     | C       | C      | =c/omm           +
           |        |          |         |        | omm=CTc/omm
 template1 | omm    | UTF8     | C       | C      | =c/omm           +
           |        |          |         |        | omm=CTc/omm
(5 rows)
```

可以看到，增加了 dbexp 数据库。

1.2.2　删除数据库

使用 SQL 语句删除数据库

使用 drop database 命令删除数据库，运行结果如下所示：

SQL code

```sql
drop database dbexp;
```

Result on PostgreSQL

```
DROP DATABASE
```

Result on OpenGauss

```
DROP DATABASE
```

观察数据库变化

使用\l命令查看当前数据库情况，运行结果如下所示：

SQL code

```
\l
```

Result on PostgreSQL

```
                                      List of databases
       Name        |  Owner   | Encoding |   Collate   |    Ctype    |   Access privileges
-------------------+----------+----------+-------------+-------------+-----------------------
 db_exp            | leo      | UTF8     | zh_CN.UTF-8 | zh_CN.UTF-8 |
 dbexp123          | leo      | UTF8     | zh_CN.UTF-8 | zh_CN.UTF-8 |
 eduoj             | postgres | UTF8     | zh_CN.UTF-8 | zh_CN.UTF-8 |
 eduoj_2020_2021_2 | leo      | UTF8     | zh_CN.UTF-8 | zh_CN.UTF-8 |
 postgres          | postgres | UTF8     | zh_CN.UTF-8 | zh_CN.UTF-8 |
 solar             | solar    | UTF8     | zh_CN.UTF-8 | zh_CN.UTF-8 | =Tc/solar            +
                   |          |          |             |             | solar=CTc/solar
 template0         | postgres | UTF8     | zh_CN.UTF-8 | zh_CN.UTF-8 | =c/postgres          +
                   |          |          |             |             | postgres=CTc/postgres
 template1         | postgres | UTF8     | zh_CN.UTF-8 | zh_CN.UTF-8 | =c/postgres          +
                   |          |          |             |             | postgres=CTc/postgres
 test              | leo      | UTF8     | zh_CN.UTF-8 | zh_CN.UTF-8 |
(9 rows)
```

Result on OpenGauss

```
                          List of databases
   Name    | Owner  | Encoding | Collate | Ctype | Access privileges
-----------+--------+----------+---------+-------+-------------------
 db_exp    | dbtest | UTF8     | C       | C     |
 postgres  | omm    | UTF8     | C       | C     | =Tc/omm          +
           |        |          |         |       | omm=CTc/omm      +
           |        |          |         |       | leo=CTc/omm      +
           |        |          |         |       | leo=APm/omm      +
           |        |          |         |       | dbtest=CTc/omm   +
           |        |          |         |       | dbtest=APm/omm
 template0 | omm    | UTF8     | C       | C     | =c/omm           +
           |        |          |         |       | omm=CTc/omm
 template1 | omm    | UTF8     | C       | C     | =c/omm           +
           |        |          |         |       | omm=CTc/omm
(4 rows)
```

可以看到，删除了 dbexp 数据库。

三、 思考题

数据库文件有哪些增长方式?

1. 按百分比增长（例如：每次增长 10%）。
2. 按固定长度增长（例如：每次增长 1MiB）。

日志文件的作用是什么?

记录数据库执行过的所有命令。可以根据日志文件诊断数据库或恢复数据库（如：当服务器意外断电时，可能数据库文件被破坏，此时可以用 binlog 来恢复数据库文件。）

四、 心得体会

实验过程中，碰到的主要问题就是实验用的数据库用户（dbtest）默认没有建立数据库的权限。需要执行 `alter user dbtest CREATEDB;` 来授予权限。

实验二：创建和删除基本表

一、实验目的

本实验的学习目标在于熟练掌握数据库基本表的创建、修改和删除的方法，具体实验目的如下：

1. 学会使用 SQL 语句创建、修改和删除表。
2. 学会使用 SQL 语句设置常用的数据完整性约束，含主键约束、外键约束、空值约束、UNIQUE 约束、默认值以及 CHECK 约束等。
3. 学会使用系统存储过程查看基本表信息。
4. 熟悉 SQL 的常用数据类型。
5. 理解相关概念：基本表与三级结构、实体完整性、参照完整性、用户定义完整性、主键、外键、空值、默认值等。
6. 建立案例数据库以及自己设计的数据库的相关基本表，为后面的实验做准备。
7. 测试各种异常、错误情况，加深对表管理操作以及相关知识点的理解。

二、实验步骤

2.2.1 创建表

查询当前数据库情况

使用 \dt 命令查看当前数据库情况，运行结果如下所示：

SQL code

```
\dt
```

Result on PostgreSQL

```
psql:temp.sql:1: error: Did not find any relations.
```

Result on OpenGauss

```
psql:temp.sql:1: error: Did not find any relations.
```

创建表

使用 CREATE TABLE 命令在默认的 public schema 中创建一个数据表，并增加主键约束：

SQL code

```
CREATE TABLE "users" (
    "id" bigserial,
    "username" varchar(30) NOT NULL,
    "nickname" varchar(30) NOT NULL,
    "email" varchar(320) NOT NULL,
    "password" varchar(60) NOT NULL,
    "created_at" timestamptz NOT NULL,
    "updated_at" timestamptz NOT NULL,
    "deleted_at" timestamptz,
    PRIMARY KEY ("id"),
    UNIQUE ("username"),
    UNIQUE ("email")
)
```

Result on PostgreSQL

```
CREATE TABLE
```

Result on OpenGauss

```
psql:temp.sql:13: NOTICE:  CREATE TABLE will create implicit sequence "users_id_seq" for serial
↪   column "users.id"
psql:temp.sql:13: NOTICE:  CREATE TABLE / PRIMARY KEY will create implicit index "users_pkey"
↪   for table "users"
psql:temp.sql:13: NOTICE:  CREATE TABLE / UNIQUE will create implicit index
↪   "users_username_key" for table "users"
psql:temp.sql:13: NOTICE:  CREATE TABLE / UNIQUE will create implicit index "users_email_key"
↪   for table "users"
CREATE TABLE
```

查询当前数据库情况

使用 `\dt` 命令查看当前数据库情况，运行结果如下所示：

SQL code

```
\dt
```

Result on PostgreSQL

```
       List of relations
 Schema | Name  | Type  | Owner
--------+-------+-------+-------
 public | users | table | leo
(1 row)
```

Result on OpenGauss

```
        List of relations
 Schema | Name  | Type  | Owner
```

```
--------+-------+-------+--------
 public | users | table | dbtest
(1 row)
```

可以看到，新增了 users 表。

2.2.2 修改表

查询当前数据表情况

使用 select 命令查看当前数据表情况，运行结果如下所示：

SQL code

```
SELECT column_name as Name, data_type as Type, is_nullable as Nullable, column_default as
↪  Default FROM information_schema.columns WHERE table_schema = 'public' AND table_name =
↪  'users';
```

Result on PostgreSQL

```
    name    |           type           | nullable |              default
------------+--------------------------+----------+-----------------------------------
 id         | bigint                   | NO       | nextval('users_id_seq'::regclass)
 username   | character varying        | NO       |
 nickname   | character varying        | NO       |
 email      | character varying        | NO       |
 password   | character varying        | NO       |
 created_at | timestamp with time zone | NO       |
 updated_at | timestamp with time zone | NO       |
 deleted_at | timestamp with time zone | YES      |
(8 rows)
```

Result on OpenGauss

```
    name    |           type           | nullable |              default
------------+--------------------------+----------+-----------------------------------
 deleted_at | timestamp with time zone | YES      |
 updated_at | timestamp with time zone | NO       |
 created_at | timestamp with time zone | NO       |
 id         | bigint                   | NO       | nextval('users_id_seq'::regclass)
 password   | character varying        | NO       |
 email      | character varying        | NO       |
 nickname   | character varying        | NO       |
 username   | character varying        | NO       |
(8 rows)
```

修改数据表

使用 alter table 命令修改数据表，运行结果如下所示：

SQL code

```
alter table users add column "age" integer;
```

Result on PostgreSQL

```
ALTER TABLE
```

Result on OpenGauss

```
ALTER TABLE
```

查询修改后数据表情况

使用 `select` 命令查看修改后数据表情况，运行结果如下所示：

SQL code

```sql
SELECT column_name as Name, data_type as Type, is_nullable as Nullable, column_default as
↪  Default FROM information_schema.columns WHERE table_schema = 'public' AND table_name =
↪  'users';
```

Result on PostgreSQL

```
    name    |           type           | nullable |              default
------------+--------------------------+----------+-----------------------------------
 id         | bigint                   | NO       | nextval('users_id_seq'::regclass)
 username   | character varying        | NO       |
 nickname   | character varying        | NO       |
 email      | character varying        | NO       |
 password   | character varying        | NO       |
 created_at | timestamp with time zone | NO       |
 updated_at | timestamp with time zone | NO       |
 deleted_at | timestamp with time zone | YES      |
 age        | integer                  | YES      |
(9 rows)
```

Result on OpenGauss

```
    name    |           type           | nullable |              default
------------+--------------------------+----------+-----------------------------------
 age        | integer                  | YES      |
 deleted_at | timestamp with time zone | YES      |
 updated_at | timestamp with time zone | NO       |
 created_at | timestamp with time zone | NO       |
 id         | bigint                   | NO       | nextval('users_id_seq'::regclass)
 password   | character varying        | NO       |
 email      | character varying        | NO       |
 nickname   | character varying        | NO       |
 username   | character varying        | NO       |
(9 rows)
```

可以看到，增加了 age 字段。

2.2.3 删除表

查询当前数据库情况

使用 \dt 命令查看当前数据库情况，运行结果如下所示：

SQL code

```
\dt
```

Result on PostgreSQL

```
       List of relations
 Schema | Name  | Type  | Owner
--------+-------+-------+-------
 public | users | table | leo
(1 row)
```

Result on OpenGauss

```
        List of relations
 Schema | Name  | Type  | Owner
--------+-------+-------+--------
 public | users | table | dbtest
(1 row)
```

删除数据表

使用 drop 命令删除数据表，运行结果如下所示：

SQL code

```
drop table users;
```

Result on PostgreSQL

```
DROP TABLE
```

Result on OpenGauss

```
DROP TABLE
```

查询操作后数据库情况

使用 \dt 命令查看操作后数据库情况，运行结果如下所示：

SQL code

```
\dt
```

Result on PostgreSQL

```
psql:temp.sql:1: error: Did not find any relations.
```

Result on OpenGauss

```
psql:temp.sql:1: error: Did not find any relations.
```

可以看到，不存在任何数据表，删除成功。

2.2.4 创建外键约束

创建班级表，并创建外键约束：

SQL code

```sql
CREATE TABLE "classes" (
    "id" bigserial,
    "name" varchar(255) NOT NULL,
    "course_name" varchar(255) NOT NULL,
    "description" text DEFAULT '',
    "invite_code" varchar(255) NOT NULL DEFAULT '',
    "created_at" timestamptz,
    "updated_at" timestamptz,
    "deleted_at" timestamptz,
    PRIMARY KEY ("id")
);

CREATE TABLE "user_in_classes" (
    "class_id" bigint not null,
    "user_id" bigint not null,
    PRIMARY KEY ("class_id", "user_id"),
    CONSTRAINT "fk_user_in_classes_class" FOREIGN KEY ("class_id") REFERENCES "classes"("id"),
    CONSTRAINT "fk_user_in_classes_user" FOREIGN KEY ("user_id") REFERENCES "users"("id")
);
```

Result on PostgreSQL

```
CREATE TABLE
CREATE TABLE
```

Result on OpenGauss

```
psql:temp.sql:11: NOTICE:  CREATE TABLE will create implicit sequence "classes_id_seq" for
↪  serial column "classes.id"
psql:temp.sql:11: NOTICE:  CREATE TABLE / PRIMARY KEY will create implicit index "classes_pkey"
↪  for table "classes"
CREATE TABLE
psql:temp.sql:19: NOTICE:  CREATE TABLE / PRIMARY KEY will create implicit index
↪  "user_in_classes_pkey" for table "user_in_classes"
CREATE TABLE
```

测试外键是否创建成功

尝试插入一条违反外键约束的数据：

SQL code

```
insert into user_in_classes (class_id, user_id) values (2, 2);
```

Result on PostgreSQL

```
psql:temp.sql:1: ERROR:  insert or update on table "user_in_classes" violates foreign key
↪  constraint "fk_user_in_classes_class"
DETAIL:  Key (class_id)=(2) is not present in table "classes".
```

Result on OpenGauss

```
psql:temp.sql:1: ERROR:  insert or update on table "user_in_classes" violates foreign key
↪  constraint "fk_user_in_classes_class"
DETAIL:  Key (class_id)=(2) is not present in table "classes".
```

可以看到，系统阻止了非法数据插入，外键创建成功。

创建非空和唯一约束

已经在第一步中创建了非空和唯一约束。下面验证是否成功：

SQL code

```
insert into users (username, nickname, email, password, created_at, updated_at, deleted_at)
↪  values (null, 'test', 'test@test.com', 'password', '2021-12-14 00:00:00', '2021-12-14
↪  00:00:00', null);
insert into users (username, nickname, email, password, created_at, updated_at, deleted_at)
↪  values ('test', 'test', 'test@test.com', 'password', '2021-12-14 00:00:00', '2021-12-14
↪  00:00:00', null);
insert into users (username, nickname, email, password, created_at, updated_at, deleted_at)
↪  values ('test', 'test', 'test@test.com', 'password', '2021-12-14 00:00:00', '2021-12-14
↪  00:00:00', null);
insert into users (username, nickname, email, password, created_at, updated_at, deleted_at)
↪  values ('test1', 'test', 'test@test.com', 'password', '2021-12-14 00:00:00', '2021-12-14
↪  00:00:00', null);
```

Result on PostgreSQL

```
psql:temp.sql:1: ERROR:  null value in column "username" violates not-null constraint
DETAIL:  Failing row contains (1, null, test, test@test.com, password, 2021-12-14 00:00:00+08,
↪  2021-12-14 00:00:00+08, null).
INSERT 0 1
psql:temp.sql:3: ERROR:  duplicate key value violates unique constraint "users_username_key"
DETAIL:  Key (username)=(test) already exists.
psql:temp.sql:4: ERROR:  duplicate key value violates unique constraint "users_email_key"
DETAIL:  Key (email)=(test@test.com) already exists.
```

Result on OpenGauss

```
psql:temp.sql:1: ERROR:  null value in column "username" violates not-null constraint
DETAIL:  Failing row contains (1, null, test, test@test.com, password, 2021-12-14 00:00:00+08,
↪  2021-12-14 00:00:00+08, null).
```

```
INSERT 0 1
psql:temp.sql:3: ERROR:  duplicate key value violates unique constraint "users_username_key"
DETAIL:  Key (username)=(test) already exists.
psql:temp.sql:4: ERROR:  duplicate key value violates unique constraint "users_email_key"
DETAIL:  Key (email)=(test@test.com) already exists.
```

可以看到，系统阻止了非法数据插入，非空和唯一约束创建成功。

2.2.5 默认值

使用 `select` 命令查看当前数据表结构，运行结果如下所示：

SQL code

```sql
SELECT column_name as Name, data_type as Type, is_nullable as Nullable, column_default as
↪  Default FROM information_schema.columns WHERE table_schema = 'public' AND table_name =
↪  'users';
```

Result on PostgreSQL

```
    name    |           type           | nullable |             default
------------+--------------------------+----------+----------------------------------
 id         | bigint                   | NO       | nextval('users_id_seq'::regclass)
 username   | character varying        | NO       |
 nickname   | character varying        | NO       |
 email      | character varying        | NO       |
 password   | character varying        | NO       |
 created_at | timestamp with time zone | NO       |
 updated_at | timestamp with time zone | NO       |
 deleted_at | timestamp with time zone | YES      |
(8 rows)
```

Result on OpenGauss

```
    name    |           type           | nullable |             default
------------+--------------------------+----------+----------------------------------
 deleted_at | timestamp with time zone | YES      |
 updated_at | timestamp with time zone | NO       |
 created_at | timestamp with time zone | NO       |
 id         | bigint                   | NO       | nextval('users_id_seq'::regclass)
 password   | character varying        | NO       |
 email      | character varying        | NO       |
 nickname   | character varying        | NO       |
 username   | character varying        | NO       |
(8 rows)
```

可以看到 `id` 列的默认值为 `users_id_seq` 序列的下一个值。

2.2.6 check 约束

SQL code

```
ALTER TABLE users ADD CONSTRAINT check_username_length CHECK (LENGTH(username) > 6);
```

Result on PostgreSQL

```
ALTER TABLE
```

Result on OpenGauss

```
ALTER TABLE
```

检查 check 约束

尝试插入违反 check 约束的数据:

SQL code

```
insert into users (username, nickname, email, password, created_at, updated_at, deleted_at)
↪ values ('testtest', 'testtest', 'test1@test.com', 'password', '2021-12-14 00:00:00',
↪ '2021-12-14 00:00:00', null);insert into users (username, nickname, email, password,
↪ created_at, updated_at, deleted_at) values ('test', 'test', 'test2@test.com', 'password',
↪ '2021-12-14 00:00:00', '2021-12-14 00:00:00', null);
```

Result on PostgreSQL

```
INSERT 0 1
psql:temp.sql:1: ERROR:  new row for relation "users" violates check constraint
↪ "check_username_length"
DETAIL:  Failing row contains (6, test, test, test2@test.com, password, 2021-12-14 00:00:00+08,
↪ 2021-12-14 00:00:00+08, null).
```

Result on OpenGauss

```
INSERT 0 1
psql:temp.sql:1: ERROR:  new row for relation "users" violates check constraint
↪ "check_username_length"
DETAIL:  Failing row contains (6, test, test, test2@test.com, password, 2021-12-14 00:00:00+08,
↪ 2021-12-14 00:00:00+08, null).
```

可以看到运行失败，系统阻止了非法数据插入。

2.2.7 创建后续实验所需要的其他数据表

SQL code

```
CREATE TABLE "user_manage_classes" (
    "class_id" bigint,
    "user_id" bigint,
    PRIMARY KEY ("class_id", "user_id"),
    CONSTRAINT "fk_user_manage_classes_class" FOREIGN KEY ("class_id") REFERENCES
    ↪ "classes"("id"),
    CONSTRAINT "fk_user_manage_classes_user" FOREIGN KEY ("user_id") REFERENCES "users"("id")
);

CREATE TABLE "roles" (
    "id" bigserial,
    "name" text,
    "target" text,
    PRIMARY KEY ("id")
);

CREATE TABLE "permissions" (
    "id" bigserial,
    "role_id" bigint,
    "name" text,
    PRIMARY KEY ("id"),
    CONSTRAINT "fk_roles_permissions" FOREIGN KEY ("role_id") REFERENCES "roles"("id")
);

CREATE TABLE "tokens" (
    "id" bigserial,
    "token" text,
    "user_id" bigint,
    "remember_me" boolean,
    "created_at" timestamptz,
    "updated_at" timestamptz,
    PRIMARY KEY ("id"),
    CONSTRAINT "fk_tokens_user" FOREIGN KEY ("user_id") REFERENCES "users"("id")
);

CREATE TABLE "webauthn_credentials" (
    "id" bigserial,
    "user_id" bigint,
    "content" text,
    "created_at" timestamptz,
    PRIMARY KEY ("id"),
    CONSTRAINT "fk_users_credentials" FOREIGN KEY ("user_id") REFERENCES "users"("id")
);

CREATE TABLE "problem_sets" (
    "id" bigserial,
    "class_id" bigint NOT NULL,
    "name" varchar(255) NOT NULL,
    "description" text,
    "start_time" timestamptz,
    "end_time" timestamptz,
    "created_at" timestamptz,
    "updated_at" timestamptz,
    "deleted_at" timestamptz,
    PRIMARY KEY ("id"),
    CONSTRAINT "fk_classes_problem_sets" FOREIGN KEY ("class_id") REFERENCES "classes"("id"),
    CONSTRAINT "fk_problem_sets_class" FOREIGN KEY ("class_id") REFERENCES "classes"("id")
);
```

```
CREATE TABLE "grades" (
    "id" bigserial,
    "user_id" bigint,
    "problem_set_id" bigint,
    "class_id" bigint,
    "detail" JSON,
    "total" bigint,
    "created_at" timestamptz,
    "updated_at" timestamptz,
    PRIMARY KEY ("id"),
    CONSTRAINT "fk_grades_user" FOREIGN KEY ("user_id") REFERENCES "users"("id"),
    CONSTRAINT "fk_grades_problem_set" FOREIGN KEY ("problem_set_id") REFERENCES
    ↪ "problem_sets"("id"),
    CONSTRAINT "fk_grades_class" FOREIGN KEY ("class_id") REFERENCES "classes"("id"),
    CONSTRAINT "fk_problem_sets_grades" FOREIGN KEY ("problem_set_id") REFERENCES
    ↪ "problem_sets"("id"),
    CONSTRAINT "fk_users_grades" FOREIGN KEY ("user_id") REFERENCES "users"("id")
);

CREATE TABLE "scripts" (
    "name" text,
    "filename" text,
    "created_at" timestamptz,
    "updated_at" timestamptz,
    PRIMARY KEY ("name")
);

CREATE TABLE "problems" (
    "id" bigserial,
    "name" varchar(255) NOT NULL DEFAULT '',
    "description" text,
    "attachment_file_name" varchar(255) NOT NULL DEFAULT '',
    "public" boolean NOT NULL DEFAULT false,
    "privacy" boolean NOT NULL DEFAULT false,
    "memory_limit" bigint NOT NULL DEFAULT 0,
    "time_limit" bigint NOT NULL DEFAULT 0,
    "language_allowed" varchar(255) NOT NULL DEFAULT '',
    "build_arg" varchar(2047) NOT NULL DEFAULT '',
    "compare_script_name" text NOT NULL DEFAULT '0',
    "created_at" timestamptz,
    "updated_at" timestamptz,
    "deleted_at" timestamptz,
    PRIMARY KEY ("id"),
    CONSTRAINT "fk_problems_compare_script" FOREIGN KEY ("compare_script_name") REFERENCES
    ↪ "scripts"("name")
);

CREATE TABLE "problems_in_problem_sets" (
    "problem_set_id" bigint,
    "problem_id" bigint,
    PRIMARY KEY ("problem_set_id",
    "problem_id"),
    CONSTRAINT "fk_problems_in_problem_sets_problem_set" FOREIGN KEY ("problem_set_id")
    ↪  REFERENCES "problem_sets"("id"),
    CONSTRAINT "fk_problems_in_problem_sets_problem" FOREIGN KEY ("problem_id") REFERENCES
    ↪  "problems"("id")
);

CREATE TABLE "user_has_roles" (
    "id" bigserial,
    "user_id" bigint NOT NULL,
    "role_id" bigint NOT NULL,
    "target_id" bigint,
    CONSTRAINT "fk_user_has_roles_user" FOREIGN KEY ("user_id") REFERENCES "users"("id"),
    CONSTRAINT "fk_user_has_roles_role" FOREIGN KEY ("role_id") REFERENCES "roles"("id")
);
```

Result on PostgreSQL

```
CREATE TABLE
CREATE TABLE
CREATE TABLE
CREATE TABLE
CREATE TABLE
CREATE TABLE
CREATE TABLE
CREATE TABLE
CREATE TABLE
CREATE TABLE
CREATE TABLE
```

Result on OpenGauss

```
psql:temp.sql:7: NOTICE:  CREATE TABLE / PRIMARY KEY will create implicit index
↪  "user_manage_classes_pkey" for table "user_manage_classes"
CREATE TABLE
psql:temp.sql:14: NOTICE:  CREATE TABLE will create implicit sequence "roles_id_seq" for serial
↪  column "roles.id"
psql:temp.sql:14: NOTICE:  CREATE TABLE / PRIMARY KEY will create implicit index "roles_pkey"
↪  for table "roles"
CREATE TABLE
psql:temp.sql:22: NOTICE:  CREATE TABLE will create implicit sequence "permissions_id_seq" for
↪  serial column "permissions.id"
psql:temp.sql:22: NOTICE:  CREATE TABLE / PRIMARY KEY will create implicit index
↪  "permissions_pkey" for table "permissions"
CREATE TABLE
psql:temp.sql:33: NOTICE:  CREATE TABLE will create implicit sequence "tokens_id_seq" for
↪  serial column "tokens.id"
psql:temp.sql:33: NOTICE:  CREATE TABLE / PRIMARY KEY will create implicit index "tokens_pkey"
↪  for table "tokens"
CREATE TABLE
psql:temp.sql:42: NOTICE:  CREATE TABLE will create implicit sequence
↪  "webauthn_credentials_id_seq" for serial column "webauthn_credentials.id"
```

```
psql:temp.sql:42: NOTICE:  CREATE TABLE / PRIMARY KEY will create implicit index
↪  "webauthn_credentials_pkey" for table "webauthn_credentials"
CREATE TABLE
psql:temp.sql:57: NOTICE:  CREATE TABLE will create implicit sequence "problem_sets_id_seq" for
↪  serial column "problem_sets.id"
psql:temp.sql:57: NOTICE:  CREATE TABLE / PRIMARY KEY will create implicit index
↪  "problem_sets_pkey" for table "problem_sets"
CREATE TABLE
psql:temp.sql:74: NOTICE:  CREATE TABLE will create implicit sequence "grades_id_seq" for
↪  serial column "grades.id"
psql:temp.sql:74: NOTICE:  CREATE TABLE / PRIMARY KEY will create implicit index "grades_pkey"
↪  for table "grades"
CREATE TABLE
psql:temp.sql:82: NOTICE:  CREATE TABLE / PRIMARY KEY will create implicit index "scripts_pkey"
↪  for table "scripts"
CREATE TABLE
psql:temp.sql:101: NOTICE:  CREATE TABLE will create implicit sequence "problems_id_seq" for
↪  serial column "problems.id"
psql:temp.sql:101: NOTICE:  CREATE TABLE / PRIMARY KEY will create implicit index
↪  "problems_pkey" for table "problems"
CREATE TABLE
psql:temp.sql:110: NOTICE:  CREATE TABLE / PRIMARY KEY will create implicit index
↪  "problems_in_problem_sets_pkey" for table "problems_in_problem_sets"
CREATE TABLE
psql:temp.sql:119: NOTICE:  CREATE TABLE will create implicit sequence "user_has_roles_id_seq"
↪  for serial column "user_has_roles.id"
CREATE TABLE
```

三、 思考题

什么叫做外键?

如果公共关键字在一个关系中是主关键字，那么这个公共关键字被称为另一个关系的外键。

外键的作用是什么?

当两个表中数据存在依赖关系时，保证数据一致性和完整性，并提供跨表查询的索引。

四、 心得体会

在本次实验过程中，我了解了创建、删除、修改表的方法，并掌握了 SQL 的常用数据类型，了解了外键、唯一等约束的存在意义。在实际生产过程中，除了在应用端对数据进行检查之外，还应该尽可能把约束写入数据库中，保证数据一致性。
实验过程中碰到的主要问题就是 OpenGauss 2.0 版本不支持 JSONB 数据类型，只得使用 JSON。

实验三：　数据的增删改

一、　实验目的

有关数据库中表的更新操作的实验，主要目的是:

1. 学会使用 SQL 语句进行数据的增删改。
2. 掌握数据增删改对数据约束的影响，深入理解主键约束、外键约束、check 约束以及空值、默认值等相关概念。
3. 熟练掌握各种数据类型的使用。
4. 对于案例数据库以及自己设计的数据库中的基本表，插入数据，作为后面查询实验的基础

二、　实验步骤

3.2.1　插入数据

使用 `insert` 指令插入数据

SQL code

```sql
insert into users (username, nickname, email, password, created_at, updated_at, deleted_at)
↪   values
('test1', 'test1', 'test1@test1.com', 'password', '2021-12-14 00:00:00', '2021-12-14 00:00:00',
↪   null),
('test2', 'test2', 'test2@test2.com', 'password', '2021-12-14 00:00:00', '2021-12-14 00:00:00',
↪   null),
('test3', 'test3', 'test3@test3.com', 'password', '2021-12-14 00:00:00', '2021-12-14 00:00:00',
↪   null),
('test4', 'test4', 'test4@test4.com', 'password', '2021-12-14 00:00:00', '2021-12-14 00:00:00',
↪   null);
```

Result on PostgreSQL

```
INSERT 0 4
```

Result on OpenGauss

```
INSERT 0 4
```

查看插入的结果

SQL code

```sql
select * from users;
```

Result on PostgreSQL

```
 id | username | nickname |   email    | password |  created_at   |  updated_at   | deleted_at
----+----------+----------+------------+----------+---------------+---------------+------------
  7 | test1    | test1    | test1@test.| password | 2021-12-14 00.| 2021-12-14 00.|
    |          |          |.1.com      |          |.:00:00+08     |.:00:00+08     |
  8 | test2    | test2    | test2@test.| password | 2021-12-14 00.| 2021-12-14 00.|
    |          |          |.2.com      |          |.:00:00+08     |.:00:00+08     |
  9 | test3    | test3    | test3@test.| password | 2021-12-14 00.| 2021-12-14 00.|
    |          |          |.3.com      |          |.:00:00+08     |.:00:00+08     |
 10 | test4    | test4    | test4@test.| password | 2021-12-14 00.| 2021-12-14 00.|
    |          |          |.4.com      |          |.:00:00+08     |.:00:00+08     |
(4 rows)
```

Result on OpenGauss

```
 id | username | nickname |   email    | password |  created_at   |  updated_at   | deleted_at
----+----------+----------+------------+----------+---------------+---------------+------------
  7 | test1    | test1    | test1@test.| password | 2021-12-14 00.| 2021-12-14 00.|
    |          |          |.1.com      |          |.:00:00+08     |.:00:00+08     |
  8 | test2    | test2    | test2@test.| password | 2021-12-14 00.| 2021-12-14 00.|
    |          |          |.2.com      |          |.:00:00+08     |.:00:00+08     |
  9 | test3    | test3    | test3@test.| password | 2021-12-14 00.| 2021-12-14 00.|
    |          |          |.3.com      |          |.:00:00+08     |.:00:00+08     |
 10 | test4    | test4    | test4@test.| password | 2021-12-14 00.| 2021-12-14 00.|
    |          |          |.4.com      |          |.:00:00+08     |.:00:00+08     |
(4 rows)
```

3.2.2 删除数据

使用 `delete` 指令插入数据

SQL code

```sql
delete from users where username = 'test2';
```

Result on PostgreSQL

```
DELETE 1
```

Result on OpenGauss

```
DELETE 1
```

查看删除的结果

SQL code

```sql
select * from users;
```

Result on PostgreSQL

```
 id | username | nickname |   email    | password |  created_at    |  updated_at    | deleted_at
----+----------+----------+------------+----------+---------------+---------------+------------
  7 | test1    | test1    | test1@test.| password | 2021-12-14 00.| 2021-12-14 00.|
    |          |          |.1.com      |          |.:00:00+08     |.:00:00+08     |
  9 | test3    | test3    | test3@test.| password | 2021-12-14 00.| 2021-12-14 00.|
    |          |          |.3.com      |          |.:00:00+08     |.:00:00+08     |
 10 | test4    | test4    | test4@test.| password | 2021-12-14 00.| 2021-12-14 00.|
    |          |          |.4.com      |          |.:00:00+08     |.:00:00+08     |
(3 rows)
```

Result on OpenGauss

```
 id | username | nickname |   email    | password |  created_at    |  updated_at    | deleted_at
----+----------+----------+------------+----------+---------------+---------------+------------
  7 | test1    | test1    | test1@test.| password | 2021-12-14 00.| 2021-12-14 00.|
    |          |          |.1.com      |          |.:00:00+08     |.:00:00+08     |
  9 | test3    | test3    | test3@test.| password | 2021-12-14 00.| 2021-12-14 00.|
    |          |          |.3.com      |          |.:00:00+08     |.:00:00+08     |
 10 | test4    | test4    | test4@test.| password | 2021-12-14 00.| 2021-12-14 00.|
    |          |          |.4.com      |          |.:00:00+08     |.:00:00+08     |
(3 rows)
```

可以看到，第二个用户被删除了。

3.2.3 修改数据

使用 `update` 指令修改数据

SQL code

```sql
update users set email = CONCAT('changed_', email) where true;
update users set nickname = CONCAT(nickname, '_changed') where id = 10;
```

Result on PostgreSQL

```
UPDATE 3
UPDATE 1
```

Result on OpenGauss

```
UPDATE 3
UPDATE 1
```

查看修改的结果

SQL code

```sql
select * from users;
```

Result on PostgreSQL

```
 id | username | nickname |    email     | password |  created_at  |  updated_at  | deleted_at
----+----------+----------+--------------+----------+--------------+--------------+------------
  7 | test1    | test1    | changed_test.| password | 2021-12-14 0.| 2021-12-14 0.|
    |          |          |.1@test1.com  |          |.0:00:00+08   |.0:00:00+08   |
  9 | test3    | test3    | changed_test.| password | 2021-12-14 0.| 2021-12-14 0.|
    |          |          |.3@test3.com  |          |.0:00:00+08   |.0:00:00+08   |
 10 | test4    | test4_ch.| changed_test.| password | 2021-12-14 0.| 2021-12-14 0.|
    |          |.anged    |.4@test4.com  |          |.0:00:00+08   |.0:00:00+08   |
(3 rows)
```

Result on OpenGauss

```
 id | username | nickname |    email     | password |  created_at  |  updated_at  | deleted_at
----+----------+----------+--------------+----------+--------------+--------------+------------
  7 | test1    | test1    | changed_test.| password | 2021-12-14 0.| 2021-12-14 0.|
    |          |          |.1@test1.com  |          |.0:00:00+08   |.0:00:00+08   |
  9 | test3    | test3    | changed_test.| password | 2021-12-14 0.| 2021-12-14 0.|
    |          |          |.3@test3.com  |          |.0:00:00+08   |.0:00:00+08   |
 10 | test4    | test4_ch.| changed_test.| password | 2021-12-14 0.| 2021-12-14 0.|
    |          |.anged    |.4@test4.com  |          |.0:00:00+08   |.0:00:00+08   |
(3 rows)
```

可以看到，邮箱和昵称字段的数据被修改了。

三、 思考题

PostgreSQL 和 OpenGauss 提供了哪些类型的约束?

CHECK、NOT NULL、UNIQUE、PRIMARY KEY、FOREIGN KEY、EXCLUDE。

`delete` 语句和 `drop table` 语句有何不同?

前者只删除表中部分或全部数据，后者在删除全部数据的同时会删掉表结构。

四、 心得体会

本次实验过程中我熟悉了如何在数据表中进行增删改操作。

实验四： 数据的检索

单表查询

一、 实验目的

单表查询的实验是使用 SELECT 语句从单一基本表查询数据，主要目的是：

1. 学会 SELECT 子句各种基本用法。
2. 熟悉单表查询中各种 WHERE 条件的使用方法。
3. 掌握常用的聚合函数的用法。
4. 掌握分组统计的概念，熟悉 GROUP BY 子句以及 HAVING 子句的基本用法。
5. 掌握结果集输出时的各种排序方法，ORDER BY 子句的常用方法。

二、 实验步骤

4.2.1 数据准备

首先插入一些示例数据，用于以后查询。

SQL code

```sql
insert into roles (name, target) values
    ('admin', null),
    ('creator', 'problem'),
    ('creator', 'class'),
    ('manager', 'problem'),
    ('student', 'class')
;
insert into permissions (role_id, name) values
    (1, 'all'),
    (2, 'all'),
    (3, 'all'),
    (4, 'read'),
    (4, 'change'),
    (4, 'update')
```

```
;
update users set deleted_at = '2021-12-14 00:00:00' where id = 10;
insert into user_has_roles (user_id, role_id, target_id) values
    (7, 1, null),
    (7, 2, 3),
    (7, 4, 4),
    (9, 2, 4),
    (9, 5, null)
;
```

Result on PostgreSQL

```
INSERT 0 5
INSERT 0 6
UPDATE 1
INSERT 0 5
```

Result on OpenGauss

```
INSERT 0 5
INSERT 0 6
UPDATE 1
INSERT 0 5
```

4.2.2 查询语句的使用

下面，通过 EduOJ 的真实使用场景来展示不同查询语句的使用。

都有哪些 role 具有 permission?

SQL code

```
select role_id from permissions group by role_id;
select distinct role_id from permissions;
```

Result on PostgreSQL

```
 role_id
---------
       3
       4
       2
       1
(4 rows)

 role_id
---------
       3
       4
       2
       1
```

```
(4 rows)
```

Result on OpenGauss

```
 role_id
---------
       1
       4
       3
       2
(4 rows)

 role_id
---------
       1
       4
       3
       2
(4 rows)
```

有哪些 role 具有 2 个以上的 permission?

SQL code

```
select role_id from permissions group by role_id having count(*) >= 2;
```

Result on PostgreSQL

```
 role_id
---------
       4
(1 row)
```

Result on OpenGauss

```
 role_id
---------
       4
(1 row)
```

按照 username 排序，第 3 个用户是哪个用户?

SQL code

```
select * from users order by username asc offset 2 limit 1;
```

Result on PostgreSQL

```
 id | username | nickname |    email     | password | created_at  | updated_at  |  deleted_at
----+----------+----------+--------------+----------+-------------+-------------+--------------
 10 | test4    | test4_ch.| changed_test.| password | 2021-12-14 .| 2021-12-14 .| 2021-12-14 0.
    |          |.anged    |.4@test4.com  |          |.00:00:00+08 |.00:00:00+08 |.0:00:00+08
(1 row)
```

Result on OpenGauss

```
 id | username | nickname |    email     | password | created_at  | updated_at  |  deleted_at
----+----------+----------+--------------+----------+-------------+-------------+--------------
 10 | test4    | test4_ch.| changed_test.| password | 2021-12-14 .| 2021-12-14 .| 2021-12-14 0.
    |          |.anged    |.4@test4.com  |          |.00:00:00+08 |.00:00:00+08 |.0:00:00+08
(1 row)
```

没被删除的用户有哪些?

SQL code

```sql
select * from users where deleted_at is null;
```

Result on PostgreSQL

```
 id | username | nickname |    email     | password |  created_at  |  updated_at  | deleted_at
----+----------+----------+--------------+----------+--------------+--------------+------------
  7 | test1    | test1    | changed_test.| password | 2021-12-14 0.| 2021-12-14 0.|
    |          |          |.1@test1.com  |          |.0:00:00+08   |.0:00:00+08   |
  9 | test3    | test3    | changed_test.| password | 2021-12-14 0.| 2021-12-14 0.|
    |          |          |.3@test3.com  |          |.0:00:00+08   |.0:00:00+08   |
(2 rows)
```

Result on OpenGauss

```
 id | username | nickname |    email     | password |  created_at  |  updated_at  | deleted_at
----+----------+----------+--------------+----------+--------------+--------------+------------
  7 | test1    | test1    | changed_test.| password | 2021-12-14 0.| 2021-12-14 0.|
    |          |          |.1@test1.com  |          |.0:00:00+08   |.0:00:00+08   |
  9 | test3    | test3    | changed_test.| password | 2021-12-14 0.| 2021-12-14 0.|
    |          |          |.3@test3.com  |          |.0:00:00+08   |.0:00:00+08   |
(2 rows)
```

被删除了的用户有哪些?

SQL code

```sql
select * from users where deleted_at is not null;
```

Result on PostgreSQL

```
 id | username | nickname |     email      | password | created_at   | updated_at   |  deleted_at
----+----------+----------+--------------+----------+-------------+-------------+--------------
 10 | test4    | test4_ch.| changed_test.| password | 2021-12-14 .| 2021-12-14 .| 2021-12-14 0.
    |          |.anged    |.4@test4.com  |          |.00:00:00+08 |.00:00:00+08 |.0:00:00+08
(1 row)
```

Result on OpenGauss

```
 id | username | nickname |     email      | password | created_at   | updated_at   |  deleted_at
----+----------+----------+--------------+----------+-------------+-------------+--------------
 10 | test4    | test4_ch.| changed_test.| password | 2021-12-14 .| 2021-12-14 .| 2021-12-14 0.
    |          |.anged    |.4@test4.com  |          |.00:00:00+08 |.00:00:00+08 |.0:00:00+08
(1 row)
```

三、 思考题

什么是空值?

空值是表示该行没有值的一种特殊状态，而不是值为空（如空字符串）。

为什么空值不用等号判定?

因为空值不是值，而是一种特殊状态。如：有`UNIQUE`约束的表中可以有多个 NULL。

聚合函数可以出现在什么字句中?

`SELECT`和`HAVING`。

什么情况下使用`HAVING`?

当需要对 `group by` 后的数据进行进一步筛选时。
筛选顺序：`where -> group by -> having`

多表查询

一、 实验目的

多表查询的实验是使用查询语句从多个基本表或视图查询数据，包含连接查询（内连接）、集合查询以及子查询 3 种查询方法，本实验主要目的是：

1. 学会内连接查询的表示方法（标准表示法或简约表示法均可），以及自连接的表示法。
2. 学会集合查询的达，包括 UNION、INTERSECT 和 EXCEPT 的表达，集合运算的“并兼容”问题。
3. 学会子查询即嵌套查询的使用方法，包括 3 种形式引入子查询的方法：[NOT] IN、比较运算符与 ALL|ANY 和 EXISTS；理解相关子查询和独立子查询的概念，学会相关子查询的表达方法。
4. 学会上述 3 种多表查询方法的综合应用。
5. 学会上述 3 种多表查询与 GROUP BY 子句以及 ORDER BY 子句的联合使用。
6. 深入理解主键、外键的概念。
7. 深入理解实体完整性约束与参照完整性约束的概念。

学习使用 SELECT 语句在多张基本表中查询各类信息。熟悉 WHERE 条件的表达、DISTINCT 的使用、连接条件与选择条件的表达。理解连接运算。

二、 实验步骤

4.2.1 多表查询语句的使用

下面，结合 EduOJ 的真实使用场景，展示多表查询语句的使用。

具有针对 id 为 3 的 problem 的 all 权限的用户有哪些?

SQL code

```sql
select * from users where id in (
    select user_id from user_has_roles where role_id in (
        select r.id from roles r
        inner join permissions p on r.id = p.role_id
        where p.name = 'all' and r.target = 'problem'
    ) and target_id = 3
);
```

Result on PostgreSQL

```
 id | username | nickname |    email     | password |  created_at  |  updated_at  | deleted_at
----+----------+----------+--------------+----------+--------------+--------------+------------
  7 | test1    | test1    | changed_test.| password | 2021-12-14 0.| 2021-12-14 0.|
    |          |          |.1@test1.com  |          |.0:00:00+08   |.0:00:00+08   |
(1 row)
```

Result on OpenGauss

```
 id | username | nickname |    email     | password |  created_at  |  updated_at  | deleted_at
----+----------+----------+--------------+----------+--------------+--------------+------------
  7 | test1    | test1    | changed_test.| password | 2021-12-14 0.| 2021-12-14 0.|
    |          |          |.1@test1.com  |          |.0:00:00+08   |.0:00:00+08   |
(1 row)
```

具有针对 id 为 4 的 problem 的 read 或 all 权限的用户有哪些?

SQL code

```
select * from users where id in (
    select user_id from user_has_roles where role_id in (
        select r.id from roles r
        inner join permissions p on r.id = p.role_id
        where p.name in ('all', 'read') and r.target = 'problem'
    ) and target_id = 4
);
```

Result on PostgreSQL

```
 id | username | nickname |    email     | password |  created_at  |  updated_at  | deleted_at
----+----------+----------+--------------+----------+--------------+--------------+------------
  9 | test3    | test3    | changed_test.| password | 2021-12-14 0.| 2021-12-14 0.|
    |          |          |.3@test3.com  |          |.0:00:00+08   |.0:00:00+08   |
  7 | test1    | test1    | changed_test.| password | 2021-12-14 0.| 2021-12-14 0.|
    |          |          |.1@test1.com  |          |.0:00:00+08   |.0:00:00+08   |
(2 rows)
```

Result on OpenGauss

```
 id | username | nickname |    email     | password |  created_at  |  updated_at  | deleted_at
----+----------+----------+--------------+----------+--------------+--------------+------------
  9 | test3    | test3    | changed_test.| password | 2021-12-14 0.| 2021-12-14 0.|
    |          |          |.3@test3.com  |          |.0:00:00+08   |.0:00:00+08   |
  7 | test1    | test1    | changed_test.| password | 2021-12-14 0.| 2021-12-14 0.|
    |          |          |.1@test1.com  |          |.0:00:00+08   |.0:00:00+08   |
(2 rows)
```

SQL code

```
select * from users where id in (
    select user_id from user_has_roles where role_id in (
        select r.id from roles r
        inner join permissions p on r.id = p.role_id
        where p.name = 'all' and r.target = 'problem'
        union
        select r.id from roles r
        inner join permissions p on r.id = p.role_id
        where p.name = 'read' and r.target = 'problem'
    ) and target_id = 4
);
```

Result on PostgreSQL

```
 id | username | nickname |     email      | password |  created_at   |  updated_at   | deleted_at
----+----------+----------+--------------+----------+--------------+--------------+------------
  9 | test3    | test3    | changed_test.| password | 2021-12-14 0.| 2021-12-14 0.|
    |          |          |.3@test3.com  |          |.0:00:00+08   |.0:00:00+08   |
  7 | test1    | test1    | changed_test.| password | 2021-12-14 0.| 2021-12-14 0.|
    |          |          |.1@test1.com  |          |.0:00:00+08   |.0:00:00+08   |
(2 rows)
```

Result on OpenGauss

```
 id | username | nickname |     email      | password |  created_at   |  updated_at   | deleted_at
----+----------+----------+--------------+----------+--------------+--------------+------------
  9 | test3    | test3    | changed_test.| password | 2021-12-14 0.| 2021-12-14 0.|
    |          |          |.3@test3.com  |          |.0:00:00+08   |.0:00:00+08   |
  7 | test1    | test1    | changed_test.| password | 2021-12-14 0.| 2021-12-14 0.|
    |          |          |.1@test1.com  |          |.0:00:00+08   |.0:00:00+08   |
(2 rows)
```

具有针对 id 为 4 的 problem 的 read 或 all 权限的第二个用户是哪个?

SQL code

```
select * from users where id in (
    select user_id from user_has_roles where role_id in (
        select r.id from roles r
        inner join permissions p on r.id = p.role_id
        where p.name in ('all', 'read') and r.target = 'problem'
    ) and target_id = 4
) order by id asc offset 1 limit 1;
```

Result on PostgreSQL

```
 id | username | nickname |     email      | password |  created_at   |  updated_at   | deleted_at
----+----------+----------+--------------+----------+--------------+--------------+------------
  9 | test3    | test3    | changed_test.| password | 2021-12-14 0.| 2021-12-14 0.|
    |          |          |.3@test3.com  |          |.0:00:00+08   |.0:00:00+08   |
(1 row)
```

Result on OpenGauss

```
 id | username | nickname |    email     | password |  created_at   |  updated_at   | deleted_at
----+----------+----------+--------------+----------+--------------+--------------+------------
  9 | test3    | test3    | changed_test.| password | 2021-12-14 0.| 2021-12-14 0.|
    |          |          |.3@test3.com  |          |.0:00:00+08   |.0:00:00+08   |
(1 row)
```

同时具有针对 id 为 3 的 problem 的 all 或 read 权限以及 id 为 4 的 problem 的 all 或 read 权限的用户有哪些?

SQL code

```sql
select * from users where id in (
    select user_id from user_has_roles where role_id in (
        select r.id from roles r
        inner join permissions p on r.id = p.role_id
        where p.name in ('all', 'read') and r.target = 'problem'
    ) and target_id = 4
    intersect
    select user_id from user_has_roles where role_id in (
        select r.id from roles r
        inner join permissions p on r.id = p.role_id
        where p.name in ('all', 'read') and r.target = 'problem'
    ) and target_id = 3
);
```

Result on PostgreSQL

```
 id | username | nickname |    email     | password |  created_at   |  updated_at   | deleted_at
----+----------+----------+--------------+----------+--------------+--------------+------------
  7 | test1    | test1    | changed_test.| password | 2021-12-14 0.| 2021-12-14 0.|
    |          |          |.1@test1.com  |          |.0:00:00+08   |.0:00:00+08   |
(1 row)
```

Result on OpenGauss

```
 id | username | nickname |    email     | password |  created_at   |  updated_at   | deleted_at
----+----------+----------+--------------+----------+--------------+--------------+------------
  7 | test1    | test1    | changed_test.| password | 2021-12-14 0.| 2021-12-14 0.|
    |          |          |.1@test1.com  |          |.0:00:00+08   |.0:00:00+08   |
(1 row)
```

具有针对 id 为 4 的 problem 的 all 或 read 权限但没有 id 为 3 的 problem 的 all 或 read 权限的用户有哪些?

SQL code

```
select * from users where id in (
    select user_id from user_has_roles where role_id in (
        select r.id from roles r
        inner join permissions p on r.id = p.role_id
        where p.name in ('all', 'read') and r.target = 'problem'
    ) and target_id = 4
    except
    select user_id from user_has_roles where role_id in (
        select r.id from roles r
        inner join permissions p on r.id = p.role_id
        where p.name in ('all', 'read') and r.target = 'problem'
    ) and target_id = 3
);
```

Result on PostgreSQL

```
 id | username | nickname |     email     | password |  created_at   |  updated_at   | deleted_at
----+----------+----------+---------------+----------+---------------+---------------+------------
  9 | test3    | test3    | changed_test.| password | 2021-12-14 0.| 2021-12-14 0.|
    |          |          |.3@test3.com   |          |.0:00:00+08    |.0:00:00+08    |
(1 row)
```

Result on OpenGauss

```
 id | username | nickname |     email     | password |  created_at   |  updated_at   | deleted_at
----+----------+----------+---------------+----------+---------------+---------------+------------
  9 | test3    | test3    | changed_test.| password | 2021-12-14 0.| 2021-12-14 0.|
    |          |          |.3@test3.com   |          |.0:00:00+08    |.0:00:00+08    |
(1 row)
```

具有针对 id 为 4 的 problem 的 read、change、update 这 3 个权限中至少 2 个的用户有哪些?

SQL code

```
select users.* from users inner join (
    select ur.user_id from user_has_roles ur
    inner join permissions p on ur.role_id = p.role_id
    where ur.target_id = 4 group by user_id having count(*) >= 2
) as rr on users.id = rr.user_id
```

Result on PostgreSQL

```
 id | username | nickname |     email     | password |  created_at   |  updated_at   | deleted_at
----+----------+----------+---------------+----------+---------------+---------------+------------
  7 | test1    | test1    | changed_test.| password | 2021-12-14 0.| 2021-12-14 0.|
    |          |          |.1@test1.com   |          |.0:00:00+08    |.0:00:00+08    |
(1 row)
```

Result on OpenGauss

```
 id | username | nickname |     email      | password |  created_at   |  updated_at   | deleted_at
----+----------+----------+--------------+----------+--------------+--------------+------------
  7 | test1    | test1    | changed_test.| password | 2021-12-14 0.| 2021-12-14 0.|
    |          |          |.1@test1.com  |          |.0:00:00+08   |.0:00:00+08   |
(1 row)
```

三、　思考题

连接条件一定是对应属性相等吗?

不一定，可以是 ≥ 等，甚至是 `true`。

所有的查询都可以使用多表连接和子查询两种方法吗?

不一定。`join` 的条件写 `true` 可以做笛卡尔乘积，但是无法用子查询达到一样的效果。
在绝大部分情况下，二者可以互相替换。二者的区别在于，子查询是『逻辑上』更合理的方式，而连接则可以更有效的运用表间外键的索引，达到更高的效率。

实验五：　创建和删除视图

一、　实验目的

本实验主要是通过学习视图的相关知识，了解数据库对象——视图的作用，创建、修改、删除视图及视图加密等相关技术。具体要求如下：

1. 掌握视图的基本概念，了解视图在数据库系统中的作用及原理。
2. 掌握使用-SL 进行视图的创建、修改和删除操作。
3. 了解基于视图进行表数据的修改及其注意事项。
4. 了解视图加密的方法。

二、　实验步骤

5.2.1　视图的创建

SQL code

```sql
create view undeleted_users as select * from users where deleted_at is not null;
```

Result on PostgreSQL

```
CREATE VIEW
```

Result on OpenGauss

```
CREATE VIEW
```

5.2.2　视图减少一列

SQL code

```sql
create view undeleted_users_no_deleted_at as select id, username, nickname, email, password,
↪   created_at, updated_at from undeleted_users where true;
```

Result on PostgreSQL

```
CREATE VIEW
```

Result on OpenGauss

```
CREATE VIEW
```

5.2.3 插入一条记录

SQL code

```
insert into undeleted_users(username, nickname, email, password,  created_at, updated_at)
values ('test0', 'test0', 'test0@test0.com', 'password',  '2021-12-14 00:00:00', '2021-12-14
↪  00:00:00');
```

Result on PostgreSQL

```
INSERT 0 1
```

Result on OpenGauss

```
psql:temp.sql:2: ERROR:  cannot insert into view "undeleted_users"
HINT:  You need an unconditional ON INSERT DO INSTEAD rule or an INSTEAD OF INSERT trigger.
```

注意，OpenGauss 不支持对于没有 Trigger 的视图的插入操作。

5.2.4 删除一条记录

SQL code

```
delete from undeleted_users where username = 'test4';
```

Result on PostgreSQL

```
DELETE 1
```

Result on OpenGauss

```
psql:temp.sql:1: ERROR:  cannot delete from view "undeleted_users"
HINT:  You need an unconditional ON DELETE DO INSTEAD rule or an INSTEAD OF DELETE trigger.
```

5.2.5 修改一条记录

SQL code

```
insert into undeleted_users(username, nickname, email, password,  created_at, updated_at,
↪  deleted_at)
values ('test4', 'test4', 'test4@test4.com', 'password',  '2021-12-14 00:00:00', '2021-12-14
↪  00:00:00', '2021-12-14 00:00:00');
update undeleted_users set username = 'undeleted_users' where undeleted_users = 'test4';
```

Result on PostgreSQL

```
INSERT 0 1
psql:temp.sql:3: ERROR:  input of anonymous composite types is not implemented
LINE 1: ...sername = 'undeleted_users' where undeleted_users = 'test4';
                                                               ^
```

Result on OpenGauss

```
psql:temp.sql:2: ERROR:  cannot insert into view "undeleted_users"
HINT:  You need an unconditional ON INSERT DO INSTEAD rule or an INSTEAD OF INSERT trigger.
psql:temp.sql:3: ERROR:  input of anonymous composite types is not implemented
LINE 1: ...sername = 'undeleted_users' where undeleted_users = 'test4';
                                                               ^
```

5.2.6 限制引用表的删除

SQL code

```
CREATE RULE do_not_delete_user AS ON DELETE TO users DO INSTEAD NOTHING;
```

Result on PostgreSQL

```
CREATE RULE
```

Result on OpenGauss

```
CREATE RULE
```

测试删除：

SQL code

```
delete from undeleted_users where username = 'test4';
```

Result on PostgreSQL

```
DELETE 0
```

Result on OpenGauss

```
psql:temp.sql:1: ERROR:  cannot delete from view "undeleted_users"
HINT:  You need an unconditional ON DELETE DO INSTEAD rule or an INSTEAD OF DELETE trigger.
```

重新查询：

SQL code

```
select * from undeleted_users;
```

Result on PostgreSQL

```
 id | username | nickname |   email   | password |  created_at  |  updated_at  |  deleted_at
----+----------+----------+-----------+----------+--------------+--------------+---------------
 12 | test4    | test4    | test4@tes.| password | 2021-12-14 0.| 2021-12-14 0.| 2021-12-14 00.
    |          |          |.t4.com    |          |.0:00:00+08   |.0:00:00+08   |.:00:00+08
(1 row)
```

Result on OpenGauss

```
 id | username | nickname |    email     | password | created_at  | updated_at  |  deleted_at
----+----------+----------+--------------+----------+-------------+-------------+--------------
 10 | test4    | test4_ch.| changed_test.| password | 2021-12-14 .| 2021-12-14 .| 2021-12-14 0.
    |          |.anged    |.4@test4.com  |          |.00:00:00+08 |.00:00:00+08 |.0:00:00+08
(1 row)
```

可以看到，删除操作没有成功。

三、 思考题

视图和基本表有何不同?

视图是编译过的 `select` 语句。如果视图是非持久话的，内存中不会存在一个视图的表结构。如果视图是持久化的，那么会在创建视图的时候创建临时表储存结果，之后只能手动刷新。此时，持久化视图创建的临时表除去可以手动刷新之外，表现和基本表就是一样的。

实验六： 创建和删除索引

一、 实验目的

本实验主要目的在于通过学习数据库索引的相关知识，了解数据库索引的结构、类型，创建方法以及索引的基本维护方法（重新生成索引和重新组织索引）。具体要求如下：

- 掌握数据库索引基本概念，以及索引的基本类型。
- 学会使用 SQL 创建、查看和修改索引。
- 学会使用 SQL 重新生成索引。
- 学会使用 SQL 重新组织索引。

二、 实验步骤

6.2.1 加索引前，分析 SQL 语句执行时间

以下列一句为例，分析加索引前所需要的执行时间：

SQL code

```sql
explain select * from users where id in (
    select user_id from user_has_roles where role_id in (
        select r.id from roles r
        inner join permissions p on r.id = p.role_id
        where p.name = 'all' and r.target = 'problem'
    ) and target_id = 3
);
```

Result on PostgreSQL

```
                                   QUERY PLAN
-----------------------------------------------------------------------------------------------
 Nested Loop  (cost=70.83..75.53 rows=1 width=842)
   ->  HashAggregate  (cost=70.69..70.70 rows=1 width=8)
         Group Key: user_has_roles.user_id
         ->  Hash Semi Join  (cost=43.67..70.68 rows=1 width=8)
               Hash Cond: (user_has_roles.role_id = r.id)
               ->  Seq Scan on user_has_roles  (cost=0.00..27.00 rows=7 width=16)
                     Filter: (target_id = 3)
               ->  Hash  (cost=43.65..43.65 rows=1 width=16)
                     ->  Merge Join  (cost=43.60..43.65 rows=1 width=16)
                           Merge Cond: (r.id = p.role_id)
                           ->  Sort  (cost=20.16..20.18 rows=4 width=8)
```

```
                                 Sort Key: r.id
                                 ->  Seq Scan on roles r  (cost=0.00..20.12 rows=4 width=8)
                                       Filter: (target = 'problem'::text)
                           ->  Sort  (cost=23.43..23.45 rows=5 width=8)
                                 Sort Key: p.role_id
                                 ->  Seq Scan on permissions p  (cost=0.00..23.38 rows=5 width.
.=8)
                                       Filter: (name = 'all'::text)
   ->  Index Scan using users_pkey on users  (cost=0.14..4.73 rows=1 width=842)
         Index Cond: (id = user_has_roles.user_id)
(20 rows)
```

Result on OpenGauss

```
                                         QUERY PLAN
--------------------------------------------------------------------------------------------
 Nested Loop  (cost=69.72..79.44 rows=7 width=842)
   ->  HashAggregate  (cost=69.72..69.74 rows=2 width=8)
         Group By Key: user_has_roles.user_id
         ->  Hash Semi Join  (cost=43.19..69.71 rows=4 width=8)
               Hash Cond: (user_has_roles.role_id = r.id)
               ->  Seq Scan on user_has_roles  (cost=0.00..26.46 rows=7 width=16)
                     Filter: (target_id = 3)
               ->  Hash  (cost=43.17..43.17 rows=1 width=16)
                     ->  Hash Join  (cost=20.06..43.17 rows=1 width=16)
                           Hash Cond: (p.role_id = r.id)
                           ->  Seq Scan on permissions p  (cost=0.00..23.09 rows=5 width=8)
                                 Filter: (name = 'all'::text)
                           ->  Hash  (cost=20.01..20.01 rows=4 width=8)
                                 ->  Seq Scan on roles r  (cost=0.00..20.01 rows=4 width=8)
                                       Filter: (target = 'problem'::text)
   ->  Index Scan using users_pkey on users  (cost=0.00..4.84 rows=1 width=842)
         Index Cond: (id = user_has_roles.user_id)
(17 rows)
```

可以发现，这个查询语句用了 4 层循环，其中 1 层优化为了 join 操作，整体 cost 为 79.44。

6.2.2 添加索引

SQL code

```sql
create index p_role_id on permissions (role_id);
create index p_name on permissions (name);
create index r_target on roles (target);
create index u_user_id on user_has_roles (user_id);
create index u_target_id on user_has_roles (target_id);
create index u_role_id on user_has_roles (role_id);
create index u_nickname on users (nickname);
create index u_email on users (email);
create index u_username on users (username);
create index c_deleted_at on classes(deleted_at);
create index w_user_id on webauthn_credentials(user_id);
create index c_invite_code on classes(invite_code);
create index uc_user_id on user_in_classes(user_id);
create index uc_class_id on user_in_classes(class_id);
create index ucm_user_id on user_manage_classes(user_id);
create index ucm_class_id on user_manage_classes(class_id);
create index g_user_id on grades(user_id);
create index g_class_id on grades(class_id);
create index ps_class_id on problem_sets(class_id);
create index pp_problem_id on problems_in_problem_sets(problem_id);
create index pp_problem_set_id on problems_in_problem_sets(problem_set_id);
reindex database dbexp;
```

Result on PostgreSQL

```
CREATE INDEX
CREATE INDEX
CREATE INDEX
CREATE INDEX
CREATE INDEX
CREATE INDEX
CREATE INDEX
CREATE INDEX
CREATE INDEX
CREATE INDEX
CREATE INDEX
CREATE INDEX
CREATE INDEX
CREATE INDEX
CREATE INDEX
CREATE INDEX
CREATE INDEX
CREATE INDEX
CREATE INDEX
CREATE INDEX
CREATE INDEX
REINDEX
```

Result on OpenGauss

```
CREATE INDEX
CREATE INDEX
CREATE INDEX
CREATE INDEX
CREATE INDEX
CREATE INDEX
CREATE INDEX
CREATE INDEX
CREATE INDEX
CREATE INDEX
CREATE INDEX
CREATE INDEX
CREATE INDEX
CREATE INDEX
CREATE INDEX
CREATE INDEX
CREATE INDEX
CREATE INDEX
CREATE INDEX
CREATE INDEX
CREATE INDEX
psql:temp.sql:22: NOTICE:  table "pg_catalog.pg_class" was reindexed
psql:temp.sql:22: NOTICE:  table "pg_catalog.pg_statistic" was reindexed
psql:temp.sql:22: NOTICE:  table "pg_catalog.pg_type" was reindexed
psql:temp.sql:22: NOTICE:  table "pg_catalog.pg_ts_dict" was reindexed
psql:temp.sql:22: NOTICE:  table "pg_catalog.pg_job_proc" was reindexed
psql:temp.sql:22: NOTICE:  table "public.users" was reindexed
psql:temp.sql:22: NOTICE:  table "public.user_in_classes" was reindexed
psql:temp.sql:22: NOTICE:  table "public.classes" was reindexed
psql:temp.sql:22: NOTICE:  table "public.user_manage_classes" was reindexed
psql:temp.sql:22: NOTICE:  table "public.permissions" was reindexed
psql:temp.sql:22: NOTICE:  table "public.tokens" was reindexed
psql:temp.sql:22: NOTICE:  table "public.webauthn_credentials" was reindexed
psql:temp.sql:22: NOTICE:  table "public.grades" was reindexed
psql:temp.sql:22: NOTICE:  table "pg_catalog.pg_authid" was reindexed
psql:temp.sql:22: NOTICE:  table "pg_catalog.pg_statistic_ext" was reindexed
psql:temp.sql:22: NOTICE:  table "pg_catalog.gs_wlm_instance_history" was reindexed
psql:temp.sql:22: NOTICE:  table "pg_catalog.gs_wlm_session_query_info_all" was reindexed
psql:temp.sql:22: NOTICE:  table "public.scripts" was reindexed
psql:temp.sql:22: NOTICE:  table "pg_catalog.gs_wlm_user_resource_history" was reindexed
psql:temp.sql:22: NOTICE:  table "pg_catalog.pg_user_mapping" was reindexed
psql:temp.sql:22: NOTICE:  table "pg_catalog.gs_wlm_operator_info" was reindexed
psql:temp.sql:22: NOTICE:  table "pg_catalog.gs_wlm_plan_operator_info" was reindexed
psql:temp.sql:22: NOTICE:  table "public.problem_sets" was reindexed
psql:temp.sql:22: NOTICE:  table "pg_catalog.gs_wlm_plan_encoding_table" was reindexed
psql:temp.sql:22: NOTICE:  table "public.problems" was reindexed
psql:temp.sql:22: NOTICE:  table "public.problems_in_problem_sets" was reindexed
psql:temp.sql:22: NOTICE:  table "pg_catalog.statement_history" was reindexed
psql:temp.sql:22: NOTICE:  table "pg_catalog.gs_wlm_ec_operator_info" was reindexed
psql:temp.sql:22: NOTICE:  table "public.roles" was reindexed
psql:temp.sql:22: NOTICE:  table "public.user_has_roles" was reindexed
psql:temp.sql:22: NOTICE:  table "pg_catalog.plan_table_data" was reindexed
psql:temp.sql:22: NOTICE:  table "pg_catalog.pg_largeobject" was reindexed
psql:temp.sql:22: NOTICE:  table "pg_catalog.pg_attribute" was reindexed
psql:temp.sql:22: NOTICE:  table "pg_catalog.pg_proc" was reindexed
psql:temp.sql:22: NOTICE:  table "pg_catalog.pg_partition" was reindexed
psql:temp.sql:22: NOTICE:  table "pg_catalog.pg_attrdef" was reindexed
psql:temp.sql:22: NOTICE:  table "pg_catalog.pg_constraint" was reindexed
psql:temp.sql:22: NOTICE:  table "pg_catalog.pg_inherits" was reindexed
psql:temp.sql:22: NOTICE:  table "pg_catalog.pg_index" was reindexed
psql:temp.sql:22: NOTICE:  table "pg_catalog.pg_operator" was reindexed
psql:temp.sql:22: NOTICE:  table "pg_catalog.pg_opfamily" was reindexed
psql:temp.sql:22: NOTICE:  table "pg_catalog.pg_opclass" was reindexed
psql:temp.sql:22: NOTICE:  table "pg_catalog.pg_am" was reindexed
```

```
psql:temp.sql:22: NOTICE:  table "pg_catalog.pg_amop" was reindexed
psql:temp.sql:22: NOTICE:  table "pg_catalog.pg_amproc" was reindexed
psql:temp.sql:22: NOTICE:  table "pg_catalog.pg_language" was reindexed
psql:temp.sql:22: NOTICE:  table "pg_catalog.pg_largeobject_metadata" was reindexed
psql:temp.sql:22: NOTICE:  table "pg_catalog.pg_aggregate" was reindexed
psql:temp.sql:22: NOTICE:  table "pg_catalog.pg_rewrite" was reindexed
psql:temp.sql:22: NOTICE:  table "pg_catalog.pg_trigger" was reindexed
psql:temp.sql:22: NOTICE:  table "pg_catalog.pg_description" was reindexed
psql:temp.sql:22: NOTICE:  table "pg_catalog.pg_cast" was reindexed
psql:temp.sql:22: NOTICE:  table "pg_catalog.pg_enum" was reindexed
psql:temp.sql:22: NOTICE:  table "pg_catalog.pg_namespace" was reindexed
psql:temp.sql:22: NOTICE:  table "pg_catalog.pg_conversion" was reindexed
psql:temp.sql:22: NOTICE:  table "pg_catalog.pg_depend" was reindexed
psql:temp.sql:22: NOTICE:  table "pg_catalog.pg_database" was reindexed
psql:temp.sql:22: NOTICE:  table "pg_catalog.pg_db_role_setting" was reindexed
psql:temp.sql:22: NOTICE:  table "pg_catalog.pg_tablespace" was reindexed
psql:temp.sql:22: NOTICE:  table "pg_catalog.pg_pltemplate" was reindexed
psql:temp.sql:22: NOTICE:  table "pg_catalog.pg_auth_members" was reindexed
psql:temp.sql:22: NOTICE:  table "pg_catalog.pg_shdepend" was reindexed
psql:temp.sql:22: NOTICE:  table "pg_catalog.pg_shdescription" was reindexed
psql:temp.sql:22: NOTICE:  table "pg_catalog.pg_ts_config" was reindexed
psql:temp.sql:22: NOTICE:  table "pg_catalog.pg_ts_config_map" was reindexed
psql:temp.sql:22: NOTICE:  table "pg_catalog.pg_ts_parser" was reindexed
psql:temp.sql:22: NOTICE:  table "pg_catalog.pg_ts_template" was reindexed
psql:temp.sql:22: NOTICE:  table "pg_catalog.pg_extension" was reindexed
psql:temp.sql:22: NOTICE:  table "pg_catalog.pg_foreign_data_wrapper" was reindexed
psql:temp.sql:22: NOTICE:  table "pg_catalog.pg_foreign_server" was reindexed
psql:temp.sql:22: NOTICE:  table "pg_catalog.pgxc_class" was reindexed
psql:temp.sql:22: NOTICE:  table "pg_catalog.pgxc_node" was reindexed
psql:temp.sql:22: NOTICE:  table "pg_catalog.pgxc_group" was reindexed
psql:temp.sql:22: NOTICE:  table "pg_catalog.pg_resource_pool" was reindexed
psql:temp.sql:22: NOTICE:  table "pg_catalog.pg_workload_group" was reindexed
psql:temp.sql:22: NOTICE:  table "pg_catalog.pg_app_workloadgroup_mapping" was reindexed
psql:temp.sql:22: NOTICE:  table "pg_catalog.pg_foreign_table" was reindexed
psql:temp.sql:22: NOTICE:  table "pg_catalog.pg_rlspolicy" was reindexed
psql:temp.sql:22: NOTICE:  table "pg_catalog.pg_default_acl" was reindexed
psql:temp.sql:22: NOTICE:  table "pg_catalog.pg_seclabel" was reindexed
psql:temp.sql:22: NOTICE:  table "pg_catalog.pg_shseclabel" was reindexed
psql:temp.sql:22: NOTICE:  table "pg_catalog.pg_collation" was reindexed
psql:temp.sql:22: NOTICE:  table "pg_catalog.pg_range" was reindexed
psql:temp.sql:22: NOTICE:  table "pg_catalog.gs_encrypted_columns" was reindexed
psql:temp.sql:22: NOTICE:  table "pg_catalog.gs_column_keys" was reindexed
psql:temp.sql:22: NOTICE:  table "pg_catalog.gs_column_keys_args" was reindexed
psql:temp.sql:22: NOTICE:  table "pg_catalog.gs_client_global_keys" was reindexed
psql:temp.sql:22: NOTICE:  table "pg_catalog.gs_client_global_keys_args" was reindexed
psql:temp.sql:22: NOTICE:  table "pg_catalog.pg_job" was reindexed
psql:temp.sql:22: NOTICE:  table "pg_catalog.gs_asp" was reindexed
psql:temp.sql:22: NOTICE:  table "pg_catalog.pg_object" was reindexed
psql:temp.sql:22: NOTICE:  table "pg_catalog.pg_synonym" was reindexed
psql:temp.sql:22: NOTICE:  table "pg_catalog.pg_directory" was reindexed
psql:temp.sql:22: NOTICE:  table "pg_catalog.pg_hashbucket" was reindexed
psql:temp.sql:22: NOTICE:  table "pg_catalog.streaming_stream" was reindexed
```

```
psql:temp.sql:22: NOTICE:  table "pg_catalog.streaming_cont_query" was reindexed
psql:temp.sql:22: NOTICE:  table "pg_catalog.streaming_reaper_status" was reindexed
psql:temp.sql:22: NOTICE:  table "pg_catalog.gs_matview" was reindexed
psql:temp.sql:22: NOTICE:  table "pg_catalog.gs_matview_dependency" was reindexed
psql:temp.sql:22: NOTICE:  table "pg_catalog.pgxc_slice" was reindexed
psql:temp.sql:22: NOTICE:  table "pg_catalog.gs_opt_model" was reindexed
psql:temp.sql:22: NOTICE:  table "pg_catalog.pg_user_status" was reindexed
psql:temp.sql:22: NOTICE:  table "pg_catalog.pg_auth_history" was reindexed
psql:temp.sql:22: NOTICE:  table "pg_catalog.pg_extension_data_source" was reindexed
psql:temp.sql:22: NOTICE:  table "pg_catalog.gs_auditing_policy" was reindexed
psql:temp.sql:22: NOTICE:  table "pg_catalog.gs_auditing_policy_access" was reindexed
psql:temp.sql:22: NOTICE:  table "pg_catalog.gs_auditing_policy_filters" was reindexed
psql:temp.sql:22: NOTICE:  table "pg_catalog.gs_auditing_policy_privileges" was reindexed
psql:temp.sql:22: NOTICE:  table "pg_catalog.gs_policy_label" was reindexed
psql:temp.sql:22: NOTICE:  table "pg_catalog.gs_masking_policy" was reindexed
psql:temp.sql:22: NOTICE:  table "pg_catalog.gs_masking_policy_actions" was reindexed
psql:temp.sql:22: NOTICE:  table "pg_catalog.gs_masking_policy_filters" was reindexed
psql:temp.sql:22: NOTICE:  table "information_schema.sql_features" was reindexed
psql:temp.sql:22: NOTICE:  table "information_schema.sql_implementation_info" was reindexed
psql:temp.sql:22: NOTICE:  table "information_schema.sql_languages" was reindexed
psql:temp.sql:22: NOTICE:  table "information_schema.sql_packages" was reindexed
psql:temp.sql:22: NOTICE:  table "information_schema.sql_parts" was reindexed
psql:temp.sql:22: NOTICE:  table "information_schema.sql_sizing" was reindexed
psql:temp.sql:22: NOTICE:  table "information_schema.sql_sizing_profiles" was reindexed
REINDEX
```

6.2.3 加索引后，分析 SQL 语句执行时间

以下列一句为例，分析加索引后所需要的执行时间：

SQL code

```sql
explain select * from users where id in (
    select user_id from user_has_roles where role_id in (
        select r.id from roles r
        inner join permissions p on r.id = p.role_id
        where p.name = 'all' and r.target = 'problem'
    ) and target_id = 3
);
```

Result on PostgreSQL

```
                                     QUERY PLAN
-------------------------------------------------------------------------------------
 Nested Loop Semi Join  (cost=0.00..4.33 rows=1 width=842)
   Join Filter: (users.id = user_has_roles.user_id)
   ->  Seq Scan on users  (cost=0.00..1.04 rows=4 width=842)
   ->  Materialize  (cost=0.00..3.23 rows=1 width=8)
         ->  Nested Loop Semi Join  (cost=0.00..3.23 rows=1 width=8)
               Join Filter: (user_has_roles.role_id = r.id)
               ->  Seq Scan on user_has_roles  (cost=0.00..1.06 rows=1 width=16)
                     Filter: (target_id = 3)
               ->  Nested Loop  (cost=0.00..2.15 rows=1 width=16)
                     Join Filter: (r.id = p.role_id)
                     ->  Seq Scan on roles r  (cost=0.00..1.06 rows=1 width=8)
                           Filter: (target = 'problem'::text)
                     ->  Seq Scan on permissions p  (cost=0.00..1.07 rows=1 width=8)
                           Filter: (name = 'all'::text)
(14 rows)
```

Result on OpenGauss

```
                                   QUERY PLAN
---------------------------------------------------------------------------------
 Hash Right Semi Join  (cost=3.22..4.33 rows=1 width=842)
   Hash Cond: (user_has_roles.user_id = users.id)
   ->  Hash Right Semi Join  (cost=2.15..3.24 rows=1 width=8)
         Hash Cond: (r.id = user_has_roles.role_id)
         ->  Hash Join  (cost=1.07..2.16 rows=1 width=16)
               Hash Cond: (p.role_id = r.id)
               ->  Seq Scan on permissions p  (cost=0.00..1.07 rows=1 width=8)
                     Filter: (name = 'all'::text)
               ->  Hash  (cost=1.06..1.06 rows=1 width=8)
                     ->  Seq Scan on roles r  (cost=0.00..1.06 rows=1 width=8)
                           Filter: (target = 'problem'::text)
         ->  Hash  (cost=1.06..1.06 rows=1 width=16)
               ->  Seq Scan on user_has_roles  (cost=0.00..1.06 rows=1 width=16)
                     Filter: (target_id = 3)
   ->  Hash  (cost=1.03..1.03 rows=3 width=842)
         ->  Seq Scan on users  (cost=0.00..1.03 rows=3 width=842)
(16 rows)
```

可以看到，SQL 语句准备的时间由 70.83 降低到了 0，执行全部结果的时间由 75.53 降低到了 4.33。同时可以发现，PostgreSQL 优化 SQL 查询的能力高于 OpenGauss。

6.2.4 删除索引

SQL code

```sql
drop index p_role_id;
```

Result on PostgreSQL

```
DROP INDEX
```

Result on OpenGauss

```
DROP INDEX
```

6.2.5 删除一条索引后，分析 SQL 执行时间

SQL code

```sql
explain select * from users where id in (
    select user_id from user_has_roles where role_id in (
        select r.id from roles r
        inner join permissions p on r.id = p.role_id
        where p.name = 'all' and r.target = 'problem'
    ) and target_id = 3
);
```

Result on PostgreSQL

```
                                     QUERY PLAN
-------------------------------------------------------------------------------------
 Nested Loop Semi Join  (cost=1.09..4.36 rows=1 width=73)
   Join Filter: (users.id = user_has_roles.user_id)
   ->  Seq Scan on users  (cost=0.00..1.04 rows=4 width=73)
   ->  Materialize  (cost=1.09..3.26 rows=1 width=8)
         ->  Nested Loop Semi Join  (cost=1.09..3.26 rows=1 width=8)
               Join Filter: (user_has_roles.role_id = r.id)
               ->  Seq Scan on user_has_roles  (cost=0.00..1.06 rows=1 width=16)
                     Filter: (target_id = 3)
               ->  Hash Join  (cost=1.09..2.18 rows=1 width=16)
                     Hash Cond: (p.role_id = r.id)
                     ->  Seq Scan on permissions p  (cost=0.00..1.07 rows=3 width=8)
                           Filter: (name = 'all'::text)
                     ->  Hash  (cost=1.06..1.06 rows=2 width=8)
                           ->  Seq Scan on roles r  (cost=0.00..1.06 rows=2 width=8)
                                 Filter: (target = 'problem'::text)
(15 rows)
```

Result on OpenGauss

```
                                   QUERY PLAN
---------------------------------------------------------------------------------
 Hash Right Semi Join  (cost=3.23..4.38 rows=1 width=79)
   Hash Cond: (user_has_roles.user_id = users.id)
   ->  Hash Right Semi Join  (cost=2.16..3.30 rows=1 width=8)
         Hash Cond: (r.id = user_has_roles.role_id)
         ->  Hash Join  (cost=1.09..2.20 rows=3 width=16)
               Hash Cond: (p.role_id = r.id)
               ->  Seq Scan on permissions p  (cost=0.00..1.07 rows=3 width=8)
                     Filter: (name = 'all'::text)
               ->  Hash  (cost=1.06..1.06 rows=2 width=8)
                     ->  Seq Scan on roles r  (cost=0.00..1.06 rows=2 width=8)
                           Filter: (target = 'problem'::text)
         ->  Hash  (cost=1.06..1.06 rows=1 width=16)
               ->  Seq Scan on user_has_roles  (cost=0.00..1.06 rows=1 width=16)
                     Filter: (target_id = 3)
   ->  Hash  (cost=1.03..1.03 rows=3 width=79)
         ->  Seq Scan on users  (cost=0.00..1.03 rows=3 width=79)
(16 rows)
```

可以看到语句执行变慢，索引删除成功。

三、　思考题

索引在数据库中的作用是什么?

加快查询速度，也可以保证数据唯一。

索引有哪几种类型?

B-tree、Hash、GiST 和 GIN。