西安交通大学

数据库系统上机实验报告

班　级: 计算机 82 班

学　号: 2160901020

姓　名: 杨雪念

2021 年 5 月 16 日

目　　录

1 OpenGauss

1.1 安装

为了简化操作，本次实验使用云和恩墨提供的docker 镜像搭建 openGauss.

1.1.1 docker 镜像

安装 docker 后，执行下面的命令拉取并安装镜像。

```
docker run --name gaussdb --privileged=true -d -e \
    GS_PASSWORD=A_COMPLEX_PASSWORD -p 15432:5432 \
    enmotech/opengauss:latest
```

这将在后台运行 openGauss，端口号为 15432.

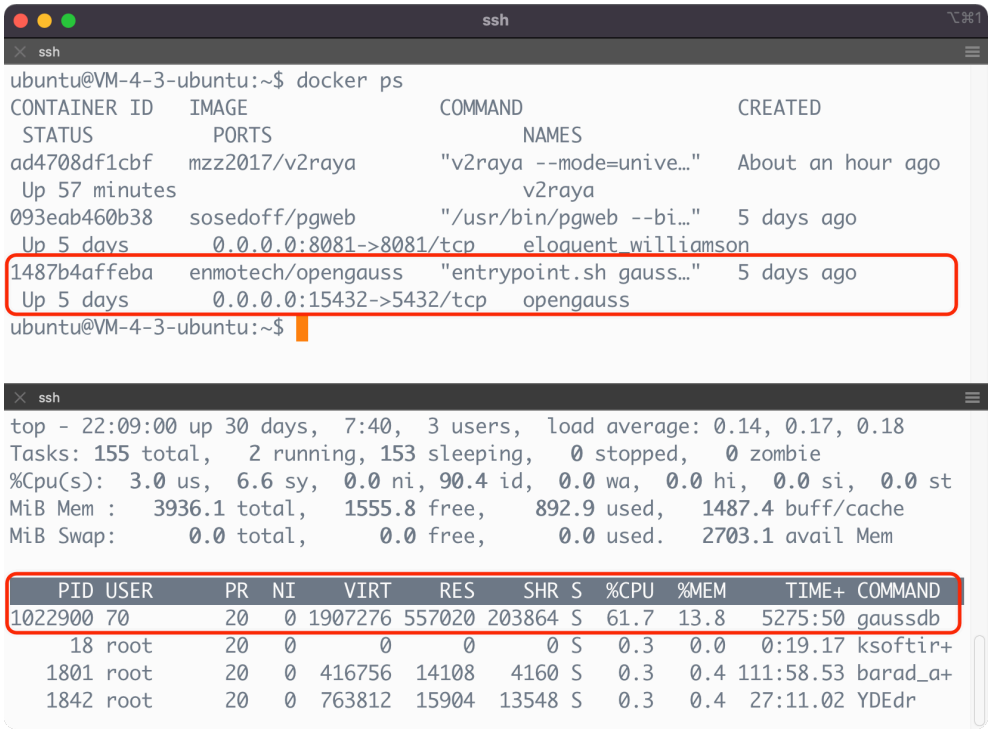


图 1.1 openGauss 运行情况

在 ubuntu 下安装 postgresql-client，方便使用 psql 和 pg_dump.

```
sudo apt-get install postgresql-client libpq-dev
```

1.1.2 建立用户和数据库

进入 openGauss 的 docker 镜像，以超级用户 omm 的身份连接

```
ubuntu@VM-4-3-ubuntu ~> docker exec -it opengauss bash
omm@1487b4affeba:/$ gsql
```

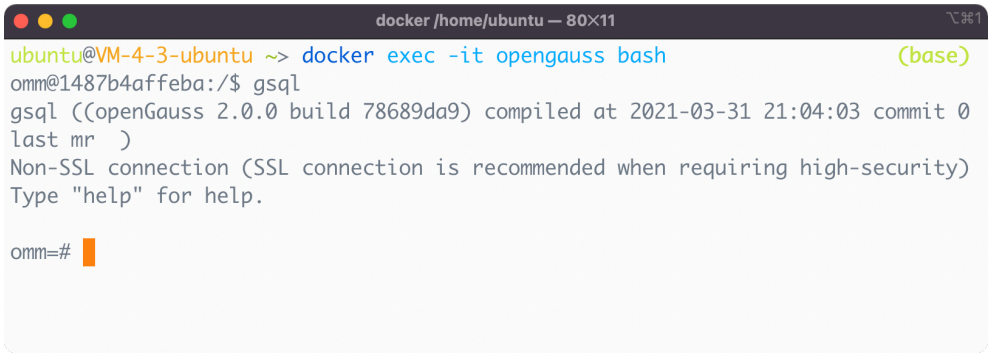


图 1.2 通过超级用户连接数据库

建立新用户并给予相应权限，这里以 XJTU020 为例（后续实验使用 gaussdb 作为用户名）

```
// 新建用户
omm=# create user XJTU020 with password 'a_COMPLEX_password';
NOTICE:  The encrypted password contains MD5 ciphertext, which is
    not secure.
CREATE ROLE

// 新建数据库并设置所有者
omm=# create database mySecDB owner XJTU020;
CREATE DATABASE

// 给予用户对该数据库所有操作权限
omm=# GRANT ALL PRIVILEGES ON DATABASE mySecDB to XJTU020;
GRANT

// 给予用户创建数据库权限
omm=# ALTER ROLE XJTU020 CREATEDB;
ALTER ROLE

```

```
// 给予用户所有权限
omm=# GRANT ALL PRIVILEGES TO XJTU020;
ALTER ROLE
```

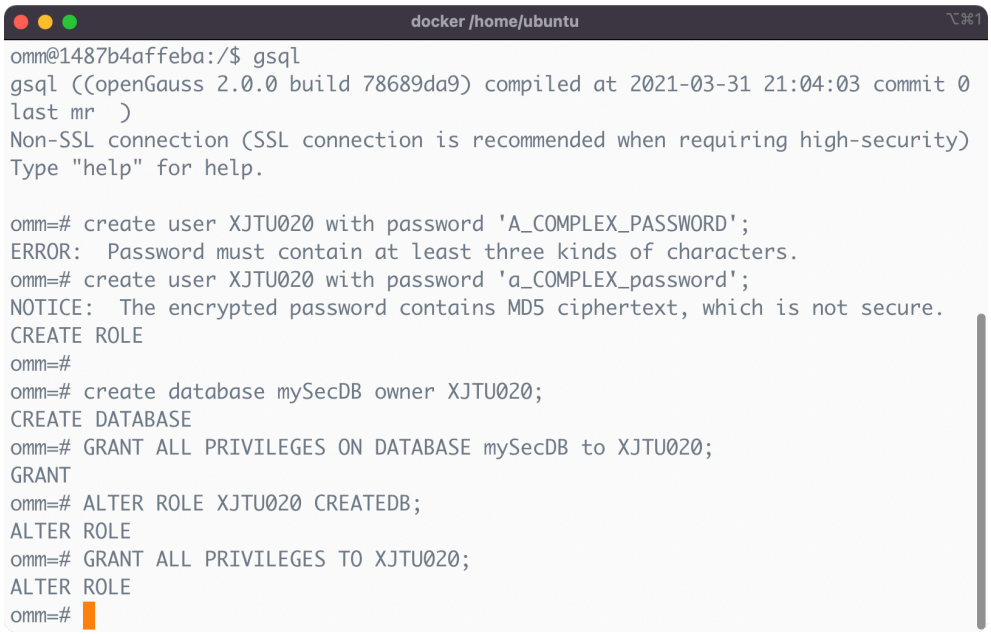


图 1.3 新建用户并给予权限

1.2 数据库操作

1.2.1 通过 psql 在终端上操作

在原主机上使用 psql 连接 openGauss 数据库

```
ubuntu@VM-4-3-ubuntu ~> psql -U gaussdb -d mydb -h 127.0.0.1 -p
    15432
```

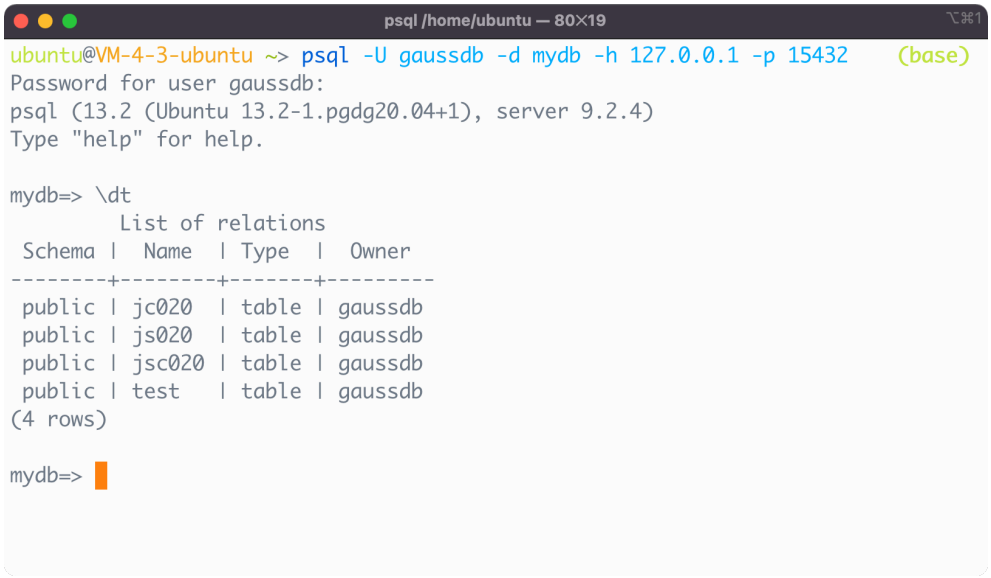


图 1.4　通过 psql 在终端上操作

1.2.2　通过 pgweb 在 web 上操作

使用 docker 部署 pgweb 镜像

```
1 docker run -p 8081:8081 sosedoff/pgweb
```

然后在http://localhost:8081查看 pgweb 的 UI.

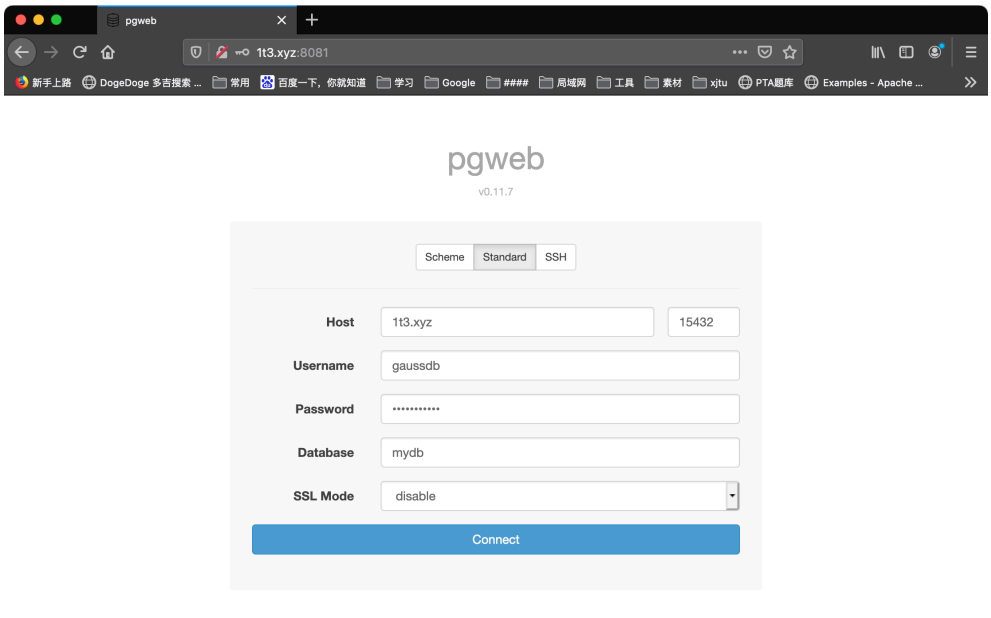


图 1.5　通过 pgweb 操作

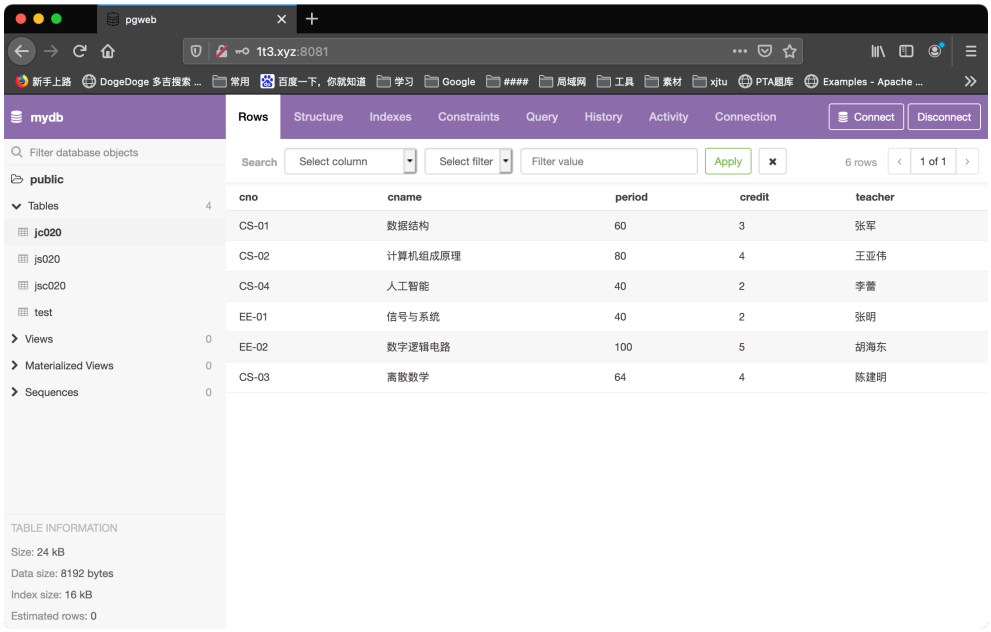


图 1.6　通过 pgweb 操作

1.2.3　通过 ORM 工具和 python 操作

python 可以使用 sqlalchemy 对多种类型的数据库进行操作，同样也适用于 openGauss。

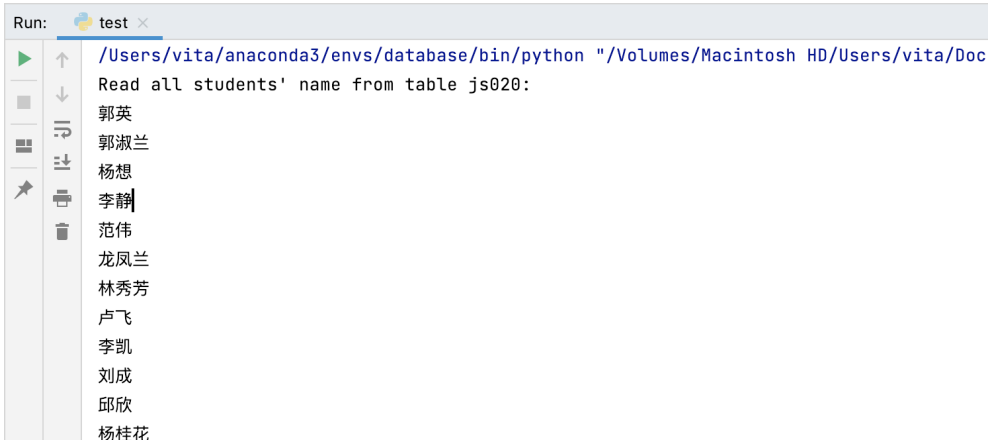


图 1.7　通过 python 操作

```
1 from extents import db, base, JS020
2
3 Session = sessionmaker(db)
4 session = Session()
5 base.metadata.create_all(db)
```

```
6
7 # 增加
8 test = {"sno": "1234567", "sname": "peter"}
9 stu = JS020(**test)
10 session.add(stu)
11 session.commit()
12
13 # 读取
14 print("Read all students' name from table js020:")
15 tests = session.query(JS020)
16 for s in tests:
17     print(s.sname)
18
19 # 修改
20 stu.sname = "张三"
21 session.commit()
22
23 # 删除
24 session.delete(stu)
25 session.commit()
```

includefile/CRUD.py

2 数据库基本实验

2.1 创建数据库，并创建学生、课程、选课表

```
gaussdb=# create database mydb;
CREATE DATABASE
```

依次执行以下 sql 命令创建学生、课程、选课表。

```
CREATE TABLE JS020(
  Sno CHAR(10) NOT NULL,
  Sname CHAR(8) NOT NULL,
  Sex CHAR(3) NOT NULL DEFAULT '男',
  Bdate date NOT NULL DEFAULT '1970-01-01',
  Height DECIMAL(3, 2) NOT NULL DEFAULT 0,
  Dorm CHAR(15),
  PRIMARY KEY (Sno)
);

CREATE TABLE JC020(
  Cno CHAR(12) NOT NULL,
  Cname CHAR(30) NOT NULL,
  Period DECIMAL(4, 1) NOT NULL DEFAULT 0,
  Credit DECIMAL(2, 1) NOT NULL DEFAULT 0,
  Teacher CHAR(10) NOT NULL,
  PRIMARY KEY (Cno)
);

CREATE TABLE jSC020(
  Sno CHAR(10) NOT NULL,
  Cno CHAR(12) NOT NULL,
  Grade DECIMAL(4, 1) DEFAULT NULL,
  PRIMARY KEY (Sno, Cno),
  FOREIGN KEY(Sno) REFERENCES jS020 ON DELETE CASCADE,
  FOREIGN KEY(Cno) REFERENCES jc020 ON DELETE RESTRICT,
  CHECK(
```

```
28      (Grade IS NULL)
29      OR (
30        Grade BETWEEN 0 AND 100
31      )
32    )
33 );
```

2.2 数据录入

具体命令见附件A.1 插入预设数据;

表属性情况如下:

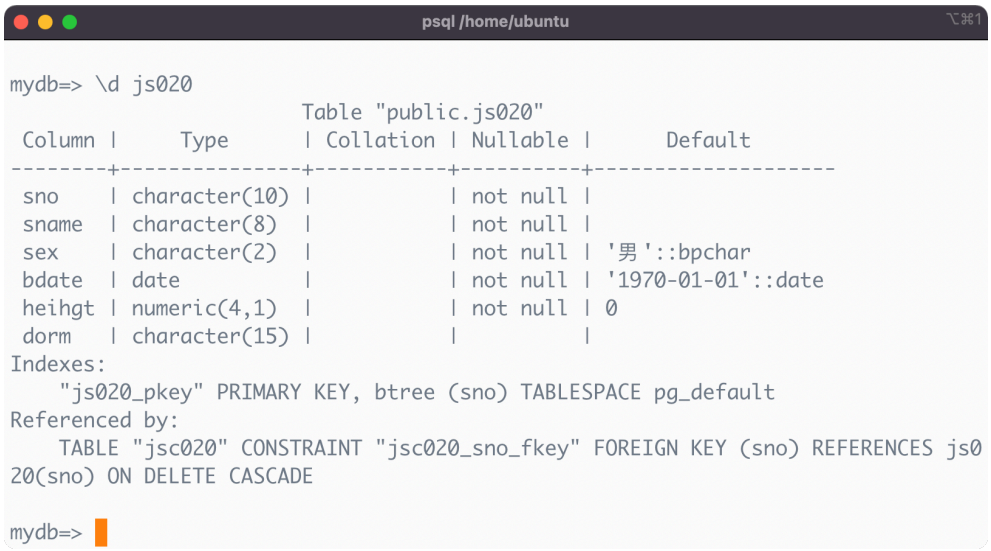


图 2.1 JS020 属性

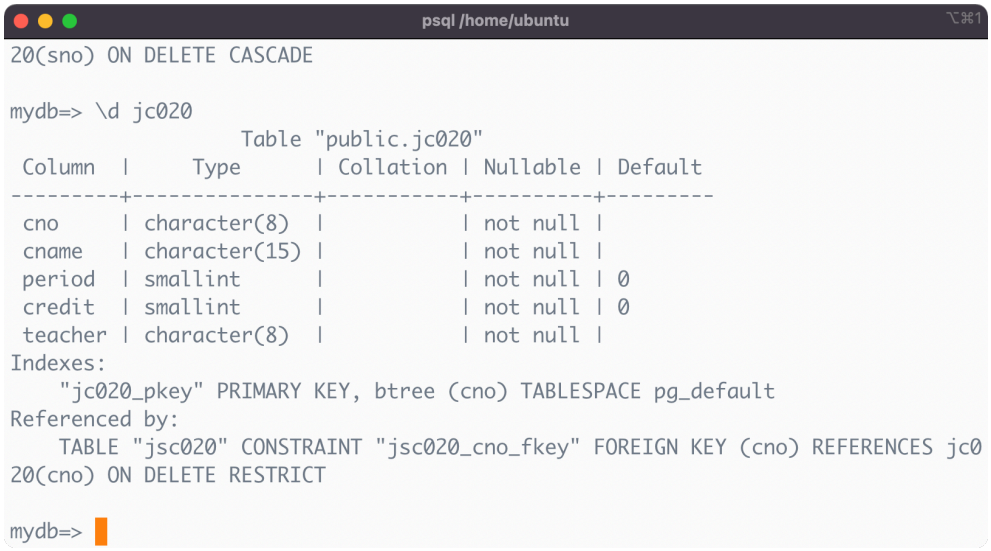


图 2.2 JC020 属性

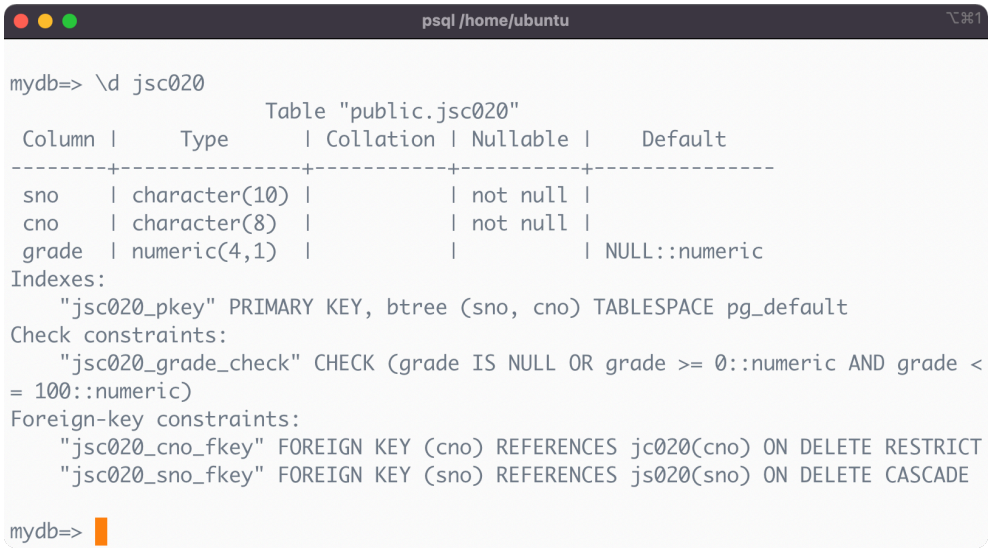


图 2.3 JSC020 属性

2.3 完成 SQL 语句

2.3.1 查询记录

(1) 查询电子工程系（EE）所开课程的课程编号、课程名称及学分数

```
1 SELECT cno,
2   cname,
3   credit
```

```
4 FROM jc020
5 WHERE cno LIKE 'EE-%';
```

图 2.4 查询结果

(2) 查询未选修课程“CS-01”的女生学号及其已选各课程编号、成绩

```
1  SELECT *
2  FROM jsc020
3  WHERE sno NOT IN (
4      SELECT sno
5      FROM jsc020
6      WHERE cno = 'CS-01'
7    )
8    AND sno IN (
9      SELECT sno
10     FROM js020
11     WHERE sex = '女'
12   );
```

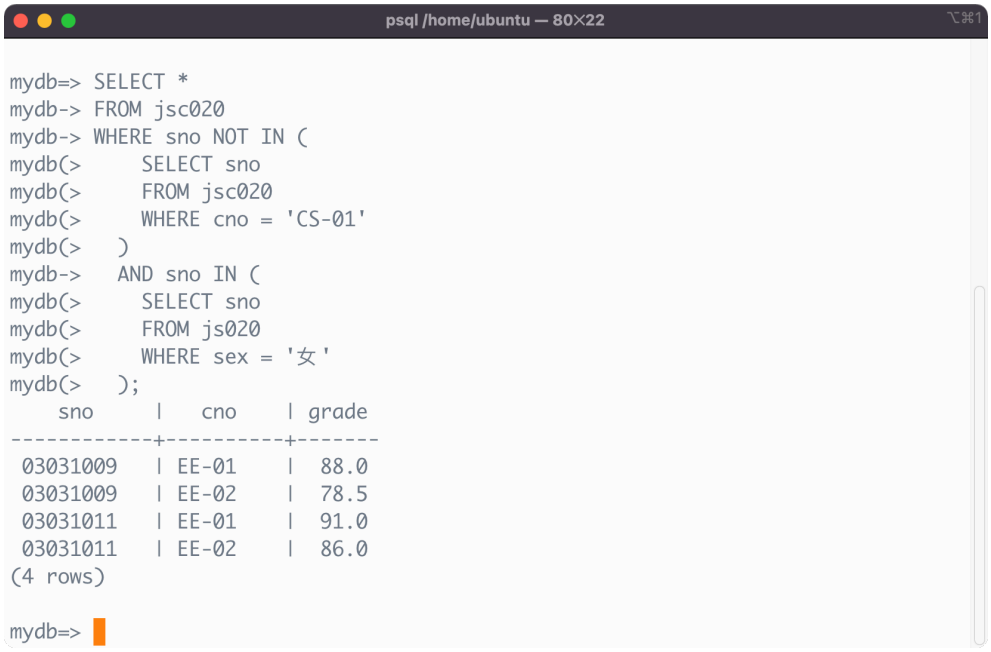


图 2.5　查询结果

(3) 查询 2000 年 ~ 2001 年出生的学生的基本信息

```sql
SELECT *
FROM js020
WHERE to_char(bdate, 'YYYY') IN ('2000', '2001');
```

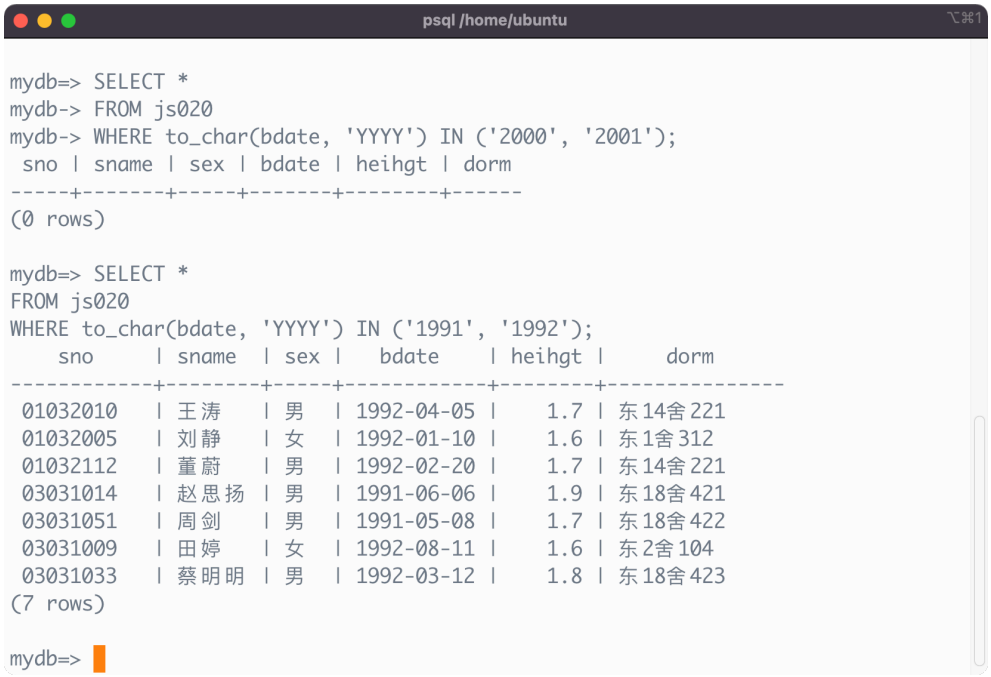


图 2.6　查询结果

由于表中没有 2000 年和 2001 年出生的学生，所以没有结果。修改为 1991 年和 1992 年查询结果正常。

(4) 查询每位学生的学号、学生姓名及其已选修课程的学分总数

```
SELECT stu.sno,
  stu.sname,
  sum(cus.credit)
FROM js020 AS stu
  INNER JOIN jsc020 AS sc ON stu.sno = sc.sno
  INNER JOIN jc020 AS cus ON sc.cno = cus.cno
GROUP BY stu.sno
ORDER BY stu.sno DESC;
```

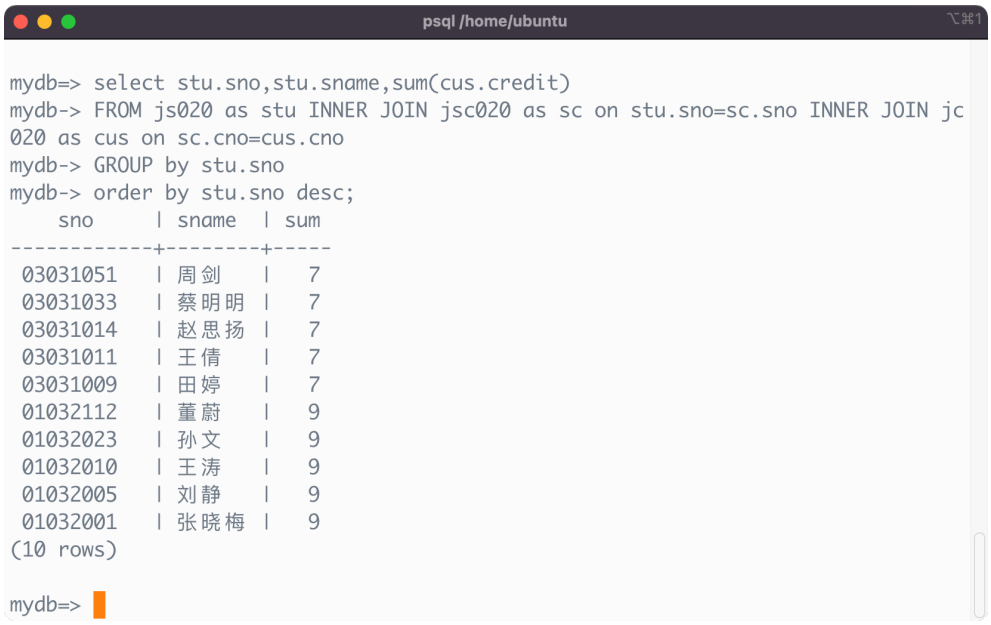


图 2.7　查询结果

(5) 查询选修课程“CS-02”的学生中成绩第二高的学生学号

```
SELECT Sno
FROM jsc020
WHERE Grade =(
    SELECT max(Grade)
    FROM jsc020
```

```
    WHERE cno = 'CS-02'
      AND Grade <(
        SELECT max(Grade)
        FROM jsc020
        WHERE cno = 'CS-02'
      )
)
AND cno = 'CS-02';
```

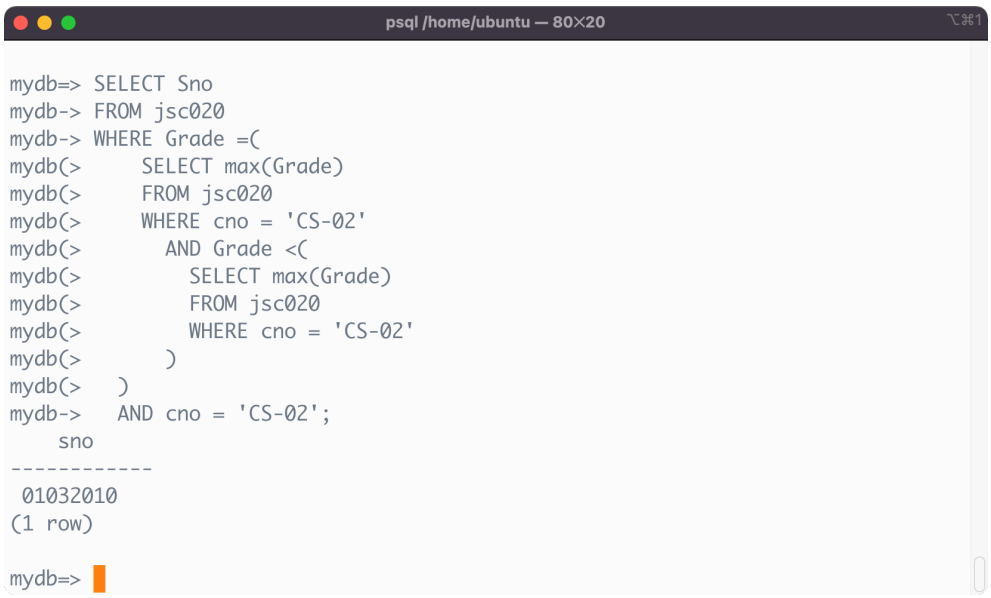


图 2.8　查询结果

(6) 查询平均成绩超过“王涛“同学的学生学号、姓名和平均成绩，并按学号进行降序排列

```
SELECT js020.sno,
  js020.sname,
  avg.avg_grade
FROM js020
  INNER JOIN (
    SELECT sno,
      avg(grade) AS avg_grade
    FROM jsc020
    GROUP BY sno
  ) AS avg ON js020.sno = avg.sno
```

```
11 WHERE avg.avg_grade >(
12     SELECT avg_grade --改为min(avg_grade) 查找小于某个王涛平均成绩的
       人
13     FROM (
14         SELECT sno,
15           avg(grade) AS avg_grade
16         FROM jsc020
17         GROUP BY sno
18       )
19     WHERE sno IN (
20         SELECT sno
21         FROM js020
22         WHERE sname = '王涛'
23       )
24   )
25 ORDER BY js020.sno DESC;
```

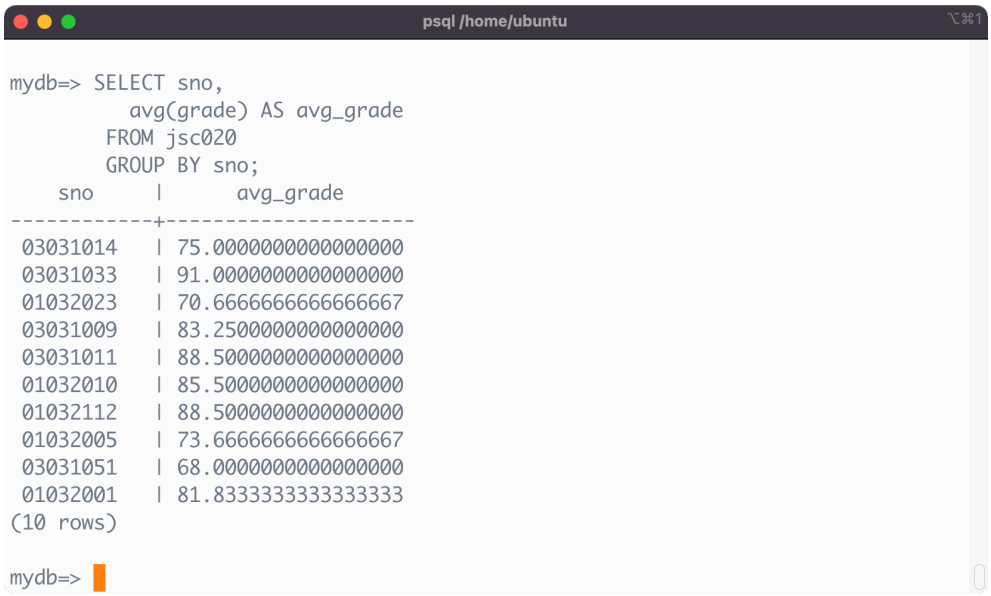


图 2.9 所有同学的平均成绩

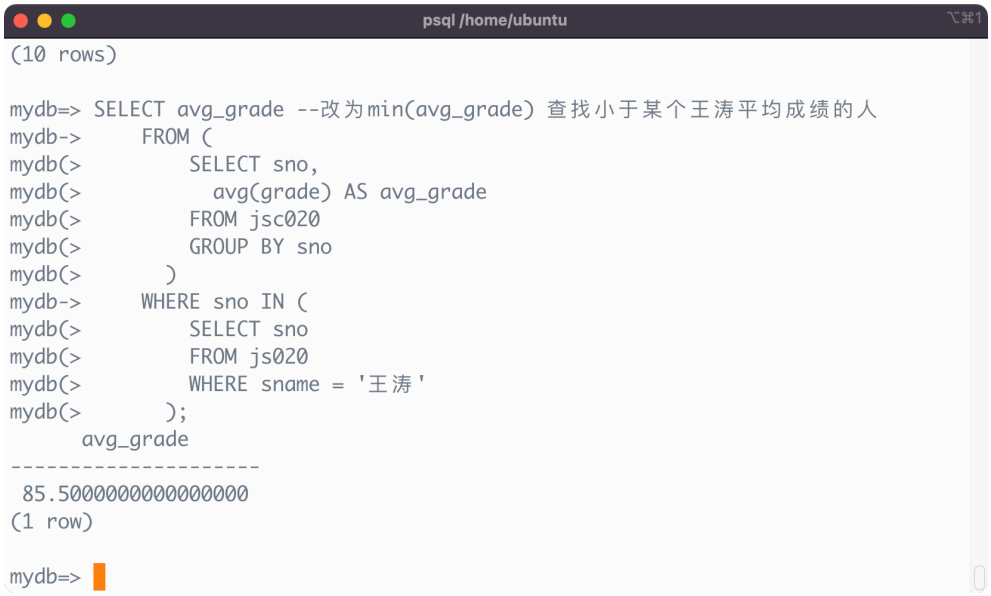


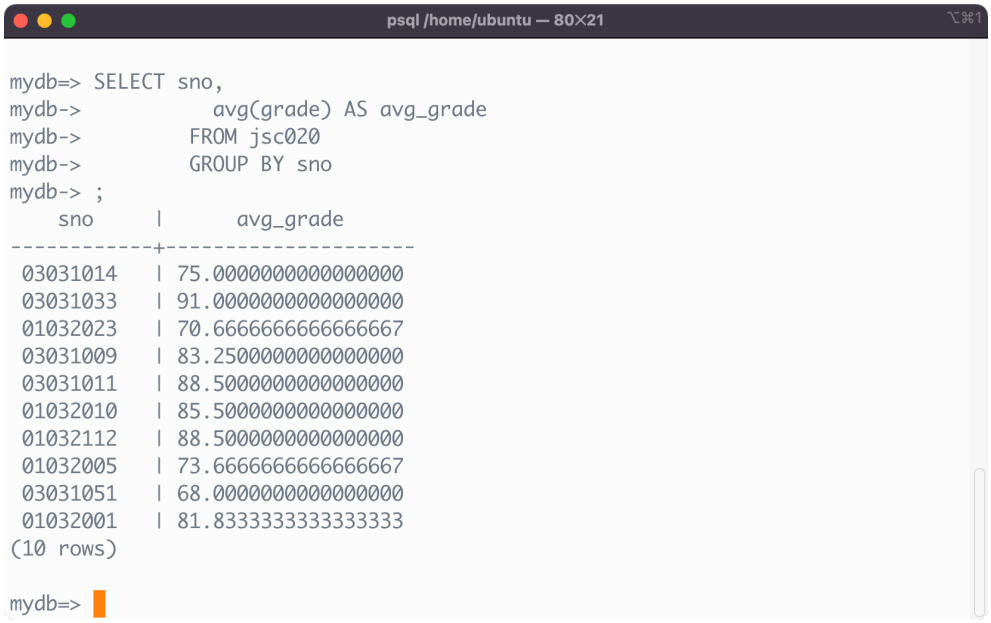
图 2.10　王涛的平均成绩

图 2.11　查询结果

(7) 查询选修了 3 门以上课程(包括 3 门)的学生中平均成绩最高的同学学号及姓名

```
1 SELECT sno,
2   sname
3 FROM js020
4 WHERE sno IN(
5     SELECT sno
```

```
6     FROM (
7         SELECT sno,
8           avg(grade) AS avg_grade
9         FROM jsc020
10        GROUP BY sno
11      )
12    WHERE avg_grade =(
13        SELECT max(avg_grade)
14        FROM (
15            SELECT sno,
16              avg(grade) AS avg_grade,
17              COUNT(cno) AS c_amount
18            FROM jsc020
19            GROUP BY sno
20          )
21        WHERE c_amount > 2
22      )
23  );
```


psql /home/ubuntu — 80×21
mydb=> SELECT sno,
mydb->   avg(grade) AS avg_grade
mydb->   FROM jsc020
mydb->   GROUP BY sno
mydb-> ;
    sno | avg_grade
------------+---------------------
 03031014 | 75.0000000000000000
 03031033 | 91.0000000000000000
 01032023 | 70.6666666666666667
 03031009 | 83.2500000000000000
 03031011 | 88.5000000000000000
 01032010 | 85.5000000000000000
 01032112 | 88.5000000000000000
 01032005 | 73.6666666666666667
 03031051 | 68.0000000000000000
 01032001 | 81.8333333333333333
(10 rows)

mydb=>
```

图 2.12　选修了 3 门以上课程（包括 3 门）的学生的平均成绩
```

图 2.13　查询结果

2.3.2　新增记录

分别在 JSXXX 和 JCXXX 表中加入记录。

```
('01032005', '刘竞', '男', '1993-12-10', '1.75', '东14舍312')
('CS-03', '离散数学', '64', '4', '陈建明')
```

分别执行新增记录，过程如下：

```
INSERT INTO js020 (Sno, SNAME, SEX, BDATE, Heihgt, DORM)
VALUES(
    '01032005',
    '刘竞',
    '男',
    '1993-12-10',
    '1.75',
    '东14舍312'
  )
-- ERROR:  duplicate key value violates unique constraint "
   js020_pkey"
-- DETAIL:  Key (sno)=(01032005) already exists.
```

执行出错，因为学号 01032005 已经有记录了，原记录的姓名是刘静。

```
INSERT INTO jc020 (Cno, CNAME, PERIOD, CREDIT, TEACHER)
VALUES ('CS-03', '离散数学', '64', '4', '陈建明');
```

执行成功。

2.3.3 删除记录

将 JS××× 表中已修学分数大于 60 的学生记录删除。

```
DELETE FROM js020
WHERE sno IN (
    SELECT sno
    FROM(
        SELECT sno,
          sum(credit) AS all_credit
        FROM jsc020
          LEFT OUTER JOIN jc020 ON jsc020.cno = jc020.cno
        WHERE grade > 59
        GROUP BY sno
      )
    WHERE all_credit > 60
  );
```

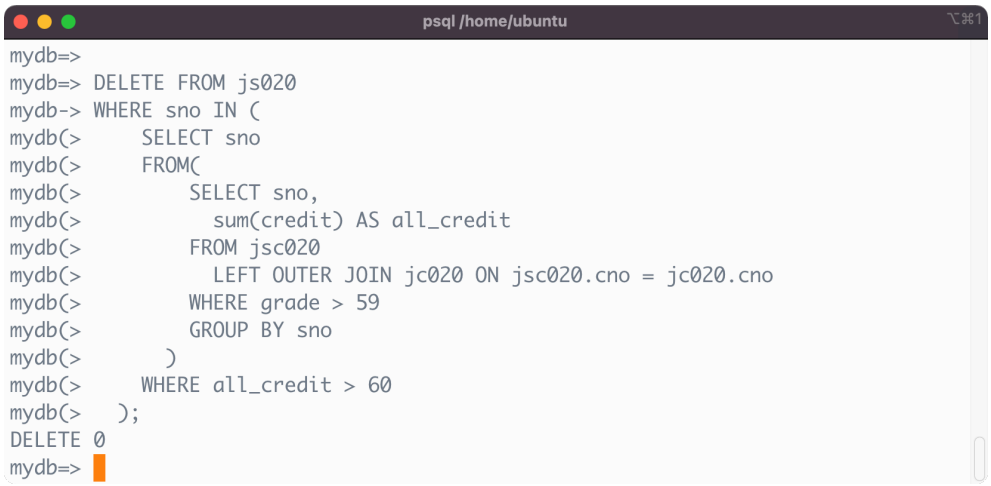


图 2.14 删除结果

由于没有学生已修分数大于 60，所以没有字段被删除。

2.3.4 修改记录

将“张明”老师负责的“信号与系统”课程的学时数调整为 64，同时增加一个学分。

```
UPDATE course
SET period_ = 64,
  credit = credit + 1
WHERE cname = '信号与系统'
  AND teacher = '张明';
```

执行成功。

2.3.5 建立视图

(1) 居住在“东 18 舍”的男生视图，包括学号、姓名、出生日期、身高等属性

建立视图，SQL 语句如下

```
CREATE VIEW e18m AS(
  SELECT snum,
    sname,
    bdate,
    heihgt
  FROM student
  WHERE sex = '男'
    AND dorm LIKE '东18%'
);
```

(2) “张明”老师所开设课程情况的视图，包括课程编号、课程名称、平均成绩等属性

建立视图，SQL 语句如下

```
CREATE VIEW zmtech AS(
  SELECT course.cnum,
    course.cname,
    tmp.avg_grade
  FROM course,
    (
      SELECT cnum,
        avg(grade) AS avg_grade
```

```
9         FROM sc
10        GROUP BY cnum
11      ) AS tmp
12    WHERE course.cnum = tmp.cnum
13      AND course.teacher = '张明'
14 );
```

(3) 所有选修了“人工智能”课程的学生视图，包括学号、姓名、成绩等属性建立视图，SQL 语句如下

```
1  CREATE VIEW rgzn AS(
2    SELECT student.snum,
3      student.sname,
4      sc.grade
5    FROM student,
6      sc
7    WHERE student.snum = sc.snum
8      AND sc.cnum IN (
9        SELECT cnum
10       FROM course
11       WHERE cname = '人工智能'
12     )
13 );
```

3 数据库触发器实验

3.1 实验要求

创建一个触发器，当 jc 表中新插入一条记录后，自动在 jsc 表中插入一条记录。其中 sno 为该生学号，cno 为课程的 cno，成绩为 100.

3.2 Solution

创建函数和触发器如下：

```
1  CREATE OR REPLACE FUNCTION give_100()
2  RETURNS TRIGGER AS $$
3  DECLARE stu_num char(10);
4  BEGIN
5    SELECT sno INTO stu_num FROM js020 LIMIT 1;
6    INSERT INTO jsc020 (Sno, Cno, GRADE)
7    VALUES (stu_num, new.cno, '100.0');
8  RETURN new;
9  END;
10 $$
11 language plpgsql;
12
13
14 CREATE TRIGGER give_100
15 AFTER INSERT ON jc020
16 FOR EACH ROW EXECUTE PROCEDURE give_100();
```

4 数据扩充和查询优化

4.1 数据扩充方法

4.1.1 学生假数据

在 Faker 库的基础上，根据需要进一步封装得到 MyFaker() 类。

```
class MyFaker():
    Faker.seed(114)
    fake = Faker('zh_CN')

    def student(self):
        fake_stu = {"sno": self.fake.unique.bothify(text='0#0#####')
    }  # 学号为不重复的8位字符串，其中第1、3位为0
        if self.fake.boolean():
            fake_stu["sex"] = '男'  # sex
            fake_stu["sname"] = self.fake.name_male()  # sname
        else:
            fake_stu["sex"] = '女'  # sex
            fake_stu["sname"] = self.fake.name_female()  # sname
        fake_stu["bdate"] = self.fake.date_between("-23y", "-19y").
    strftime("%Y-%m-%d")  # bdate 年龄在19-23之间
        fake_stu["height"] = "{:.2f}".format(self.fake.random_int(
    min=150, max=210) / 100) # height 1.50-2.10m
        fake_stu["dorm"] = self.fake.bothify(letters='东西南北', text
    ="?") + "{}舍{}{:0>2d}".format(
            self.fake.random_int(min=1, max=20),
            self.fake.random_int(min=1, max=7),
            self.fake.random_int(min=1, max=30))  # dorm
        return fake_stu
```

运行样例如下：

```
>>> from extents import MyFaker
>>> fake=MyFaker()
>>> print(fake.student())
```

```
{'sno': '03091539', 'sex': '男', 'sname': '郭英', 'bdate': '
    2000-10-16', 'height': '1.54', 'dorm': '南11舍604'}
>>> print(fake.student())
{'sno': '01076198', 'sex': '男', 'sname': '郭淑兰', 'bdate': '
    2000-12-18', 'height': '1.79', 'dorm': '西16舍626'}
```

4.1.2 课程假数据

由于课程数据具有一定语义，不便使用随机的方法生成。这里通过http://ehall.xjtu.edu.cn/的全校课表查询接口获取了电子与信息学部、电气工程学院和马克思主义学院的课程，处理后作为实验的假数据。

分析接口返回的 json 数据，容易得知`json_text["datas"]["qxfbkccx"]["rows"]`中的每一行为一门课程的信息，而对于其中的每个 row 有

1. `row['KCH']` - 课程号，如“AUTO200127”
2. `row['KCM']` - 课程名，如“计算机科学与人工智能的数学基础 I”
3. `row['XS']` - 学时，如 64.0
4. `row['XF']` - 学分，如 4.0
5. `row['SKJS']` - 授课教师，如“汪建基”

获取 JSON 中的课程信息后，发现部分字段存在冗余，需要删除无用内容，所以分几步进行处理。

部分课程名带有英文后缀,如`"KCM": "信息论基础The Elements of Information Theory"`，使用正则表达式删除无用内容

```
cus_name = re.sub("\b?[A-Za-z].*$", "", row['KCM'])
```

存在课程号相同，但授课老师不同的课程。由于主键不能重复，这与数据库设计相悖，所以每个课程号只保留一门课程。

部分课程由多位老师共同教授，老师之间使用逗号隔开。由于数据库未设计这种情况，故只保留第一位老师，使用正则表达式处理

```
fake_cus["teacher"] = re.sub("[, ].*$", "", row['SKJS'])
```

运行样例如下

```
>>> from extents import MyFaker
>>> fake=MyFaker()
>>> print(fake.course('src/course.json')[0][0])
```

```
{'cno': 'AUTO200127', 'cname': '计算机科学与人工智能的数学基础', 'period': 64.0, 'credit': 4.0, 'teacher': '汪建基'}
>>> print(fake.course('src/course.json')[0][10])
{'cno': 'AUTO402127', 'cname': '人工智能工具、平台与系统', 'period': 40.0, 'credit': 2.0, 'teacher': '刘剑毅'}
```

4.1.3 选课假数据

选课表包含学号、课程号和成绩，先遍历已生成的学号，然后随机选取若干个课程号作为学生的选课，并为之生成一个成绩即可。

假设每名学生可以选择 0-12 门课，课程的成绩在 [0,100]，步长为 0.5 的离散区间，或 None。可以按条件生成选课数据。

部分样例如下

```
{'sno': '04033475', 'cno': 'ELEC526404', 'grade': '100.0'}
{'sno': '03055355', 'cno': 'EELC511205', 'grade': '35.0'}
{'sno': '09011948', 'cno': 'AUTO401327'}  # default to be null
```

4.1.4 数据插入

数据的插入借助了SQLAlchemy 库，SQLAlchemy 是 python 下一个非常方便的 ORM 工具。

首先定义数据库表的结构和属性，例如附件A.2.2.

然后初始化数据库对象，进行增加操作

```
DB_LINK='postgresql://xjtu020:a_COMPLEX_password@127.0.0.1:15432/mysecdb'
db_link = DB_LINK
db = create_engine(db_link)
base = declarative_base()

s_info={'sno': '03091539', 'sex': '男', 'sname': '郭英', 'bdate': '2000-10-16', 'height': '1.54', 'dorm': '南11舍604'}
student = JS020(**s_info)
session.add(student)
session.commit()  # 插入对象成功
```

4.1.5 第一次扩充

在 JS020 表中补充数据至约 1000 行，在 JC020 表中补充数据至约 100 行，在 JSC020 表中补充数据至约 5000 行，为（4）-（7）查询编写不同的 SQL 语句实现，并分析其运行效率。

通过上述程序生成假数据，其中学生 1000 名，课程 229 个，选课记录 5832 条。

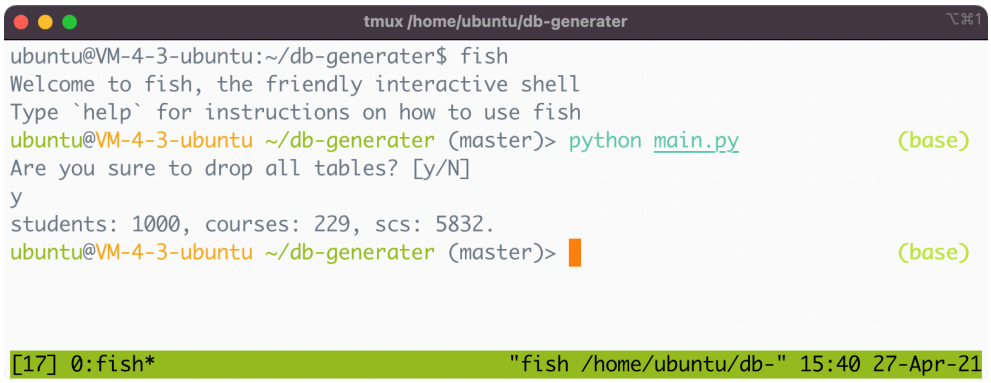


图 4.1 生成假数据

4.1.6 第二次扩充

在 JS020 表中补充数据至约 5000 行，在 JC020 表中补充数据至约 1000 行，在 JSC020 表中补充数据至约 20000 行，重复 SQL 语句运行，分析其运行效率，并给出可提高查询效率的改进方法。

通过上述程序生成假数据，其中学生 2000 名，课程 1334 个，选课记录 5832 条。

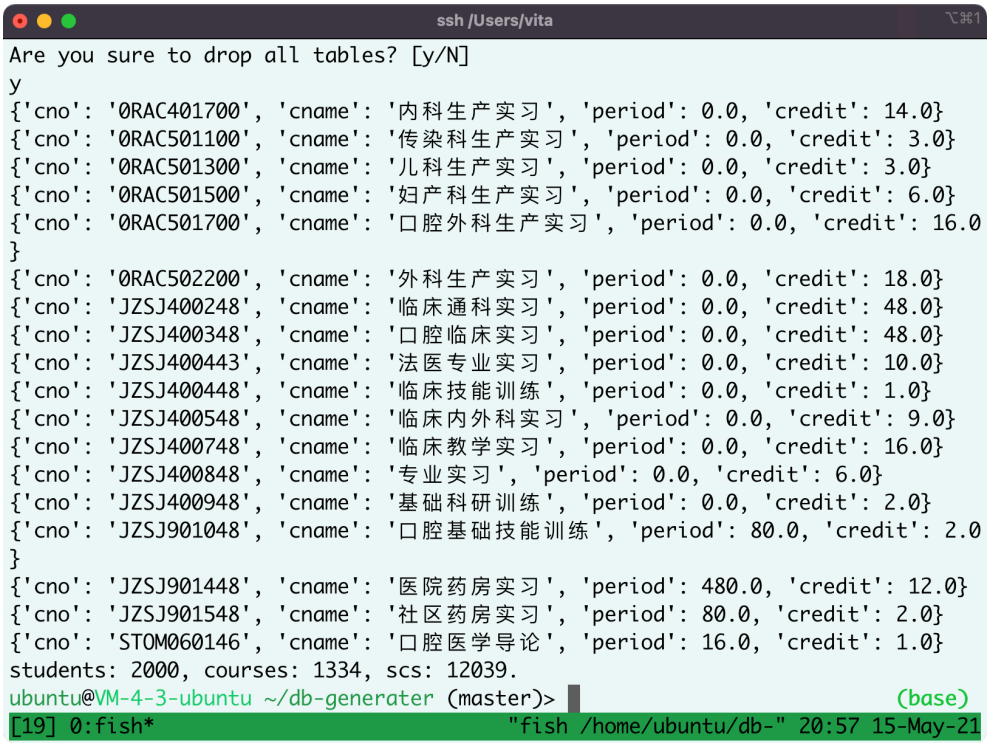
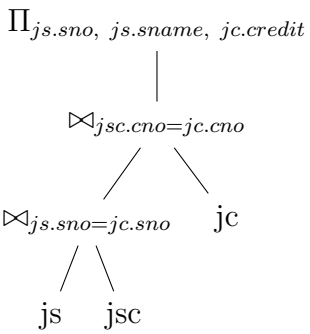
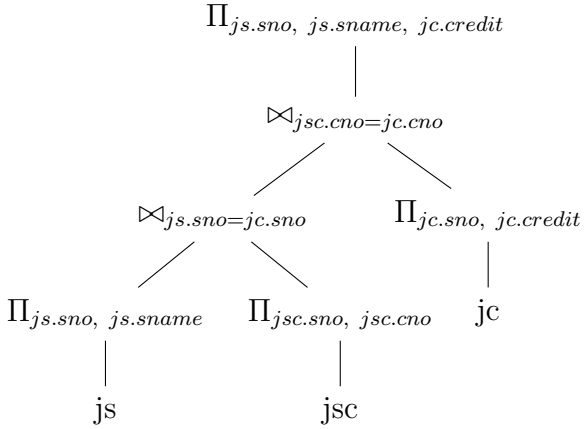


图 4.2 生成假数据

4.2 查询优化和性能比较

(4) 查询每位学生的学号、学生姓名及其已选修课程的学分总数

表 4.1 语法树

上文写法的语法树	优化后语法树
	

上文写法，通过两次内连接：

```sql
SELECT js.sno,
  js.sname,
  sum(jc.credit)
FROM js020 AS js
  INNER JOIN jsc020 AS jsc ON js.sno = jsc.sno
  INNER JOIN jc020 AS jc ON jsc.cno = jc.cno
GROUP BY js.sno, js.sname
ORDER BY js.sno DESC;
```

优化后写法：

```sql
SELECT js.sno, js.sname, sum(jc.credit)
FROM (
    SELECT sno, sname
    FROM js020
  ) AS js
  INNER JOIN (
    SELECT sno, cno
    FROM jsc020
  ) AS jsc ON js.sno = jsc.sno
  INNER JOIN (
    SELECT cno, credit
    FROM jc020
  ) AS jc ON jsc.cno = jc.cno
GROUP BY js.sno, js.sname
ORDER BY js.sno DESC;
```

通过在命令前添加EXPLIAN前缀分析两种方法的性能，发现两者没有区别。

```
# (4) 查询每位学生的学号、学生姓名及其已选修课程的学分总数。
# ------ new ------
Sort  (cost=405.92..408.42 rows=1000 width=54)
  Sort Key: js020.sno DESC
  ->  HashAggregate  (cost=346.09..356.09 rows=1000 width=54)
        Group By Key: js020.sno, js020.sname
        ->  Hash Join  (cost=40.65..302.35 rows=5832 width=22)
              Hash Cond: ((jsc020.cno)::text = (jc020.cno)::text)
              ->  Hash Join  (cost=32.50..214.01 rows=5832 width=28)
                    Hash Cond: ((jsc020.sno)::text = (js020.sno)::te
                    ->  Seq Scan on jsc020  (cost=0.00..101.32 rows=
                    ->  Hash  (cost=20.00..20.00 rows=1000 width=17)
                          ->  Seq Scan on js020  (cost=0.00..20.00 T
              ->  Hash  (cost=5.29..5.29 rows=229 width=16)
                    ->  Seq Scan on jc020  (cost=0.00..5.29 rows=229
(13 rows)
```

```
# (4) 查询每位学生的学号、学生姓名及其已选修课程的学分总数。
# ------ old ------
Sort  (cost=405.92..408.42 rows=1000 width=54)
  Sort Key: js.sno DESC
  ->  HashAggregate  (cost=346.09..356.09 rows=1000 width=54)
        Group By Key: js.sno, js.sname
        ->  Hash Join  (cost=40.65..302.35 rows=5832 width=22)
              Hash Cond: ((jsc.cno)::text = (jc.cno)::text)
              ->  Hash Join  (cost=32.50..214.01 rows=5832 width=28)
                    Hash Cond: ((jsc.sno)::text = (js.sno)::text)
                    ->  Seq Scan on jsc020 jsc  (cost=0.00..101.32 rows=5832 w
                    ->  Hash  (cost=20.00..20.00 rows=1000 width=17)
                          ->  Seq Scan on js020 js  (cost=0.00..20.00 rows=1000
              ->  Hash  (cost=5.29..5.29 rows=229 width=16)
                    ->  Seq Scan on jc020 jc  (cost=0.00..5.29 rows=229 width=
(13 rows)
```

图 4.3 性能分析结果

(5) 查询选修课程'CS-02' 的学生中成绩第二高的学生学号

上文写法，先找到选修该课程学生的最高成绩 MG，然后找到最大成绩成绩小于 MG 的最大成绩，最后在 jsc020 中找到取得该成绩对应的学号：

```
SELECT Sno
FROM jsc020
WHERE Grade in(
    SELECT max(Grade)
    FROM jsc020
    WHERE cno = 'AUTO200127'
      AND Grade <(
        SELECT max(Grade)
        FROM jsc020
        WHERE cno = 'AUTO200127'
      )
  )
  AND cno = 'AUTO200127';
```

第二种写法，对所有成绩排序并去重，找到最高的前 2 个成绩，然后取这两个成绩的最小值，最后在 jsc020 中找到取得该成绩对应的学号：

```
SELECT Sno
FROM jsc020
WHERE Grade IN (
    SELECT min(grade)
    FROM(
        SELECT DISTINCT grade
        FROM jsc020
        WHERE cno = 'AUTO200127'
          AND grade IS NOT NULL
        ORDER BY grade DESC
        LIMIT 2
      )
  )
  AND cno = 'AUTO200127';
```

通过在命令前添加EXPLIAN前缀分析两种方法的性能，结果说明第二种性能更好。

由于第二种方法（成绩去重并选择最高的 2 个成绩）的中间结果，相比于第一种方法（小于最大成绩的剩余成绩）要少很多记录，所以性能更好。

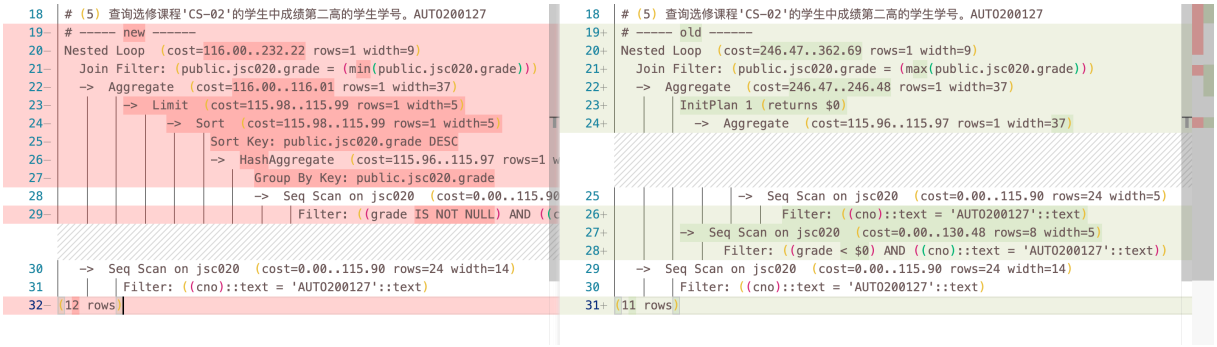


图 4.4 性能分析结果

(6) 查询平均成绩超过’王涛’同学的学生学号、姓名和平均成绩，并按学号进行降序排列

由于数据库中没有名为’王涛’的同学，此处改为’李凯’。

第一种方法，先计算 jsc020 表中每个人的平均成绩，然后找出姓名为’李凯’同学的平均成绩，通过 HAVING 保留大于该成绩的记录。然后将记录与 js020 内连接，排序。

```
SELECT js020.sno,
  js020.sname,
  avg.avg_grade
FROM js020
  INNER JOIN (
    SELECT jsc020.sno,
      avg(jsc020.grade) AS avg_grade
    FROM jsc020
    GROUP BY jsc020.sno
    HAVING (
        avg_grade > (
          SELECT avg_grade
          FROM (
              SELECT jsc020.sno,
                avg(jsc020.grade) AS avg_grade
              FROM jsc020
              GROUP BY jsc020.sno
```

```
            )
          WHERE sno IN (
              SELECT js020.sno
              FROM js020
              WHERE js020.sname = '李凯'
            )
        )
      )
  ) AS avg ON js020.sno = avg.sno
ORDER BY js020.sno DESC;
```

第二种方法，先将表 jsc020 和表 js020 内连接，并求出每名学生的平均成绩，再选择平均成绩大于’ 李凯’ 记录，排序。

```
SELECT *
FROM (
    SELECT js020.sno AS stu_sno,
      js020.sname AS stu_name,
      avg(jsc020.grade) AS avg_grade
    FROM js020
      INNER JOIN jsc020 ON js020.sno = jsc020.sno
    GROUP BY js020.sno,
      js020.sname
  ) AS js_avg
WHERE avg_grade > (
    SELECT avg_grade
    FROM (
        SELECT js020.sname AS stu_name,
          avg(jsc020.grade) AS avg_grade
        FROM js020
          INNER JOIN jsc020 ON js020.sno = jsc020.sno
        GROUP BY js020.sname
      )
    WHERE stu_name = '李凯'
  )
ORDER BY stu_sno DESC;
```

通过在命令前添加EXPLIAN前缀分析两种方法的性能，两者差距不大，后者较前者性能稍好。两种方法都有两次内连接，第一种方法的内连接相当耗时，因为其中有 Having 子句；后者有两次相同内连接，有一次被优化掉了，可能是重复查询存在缓存。

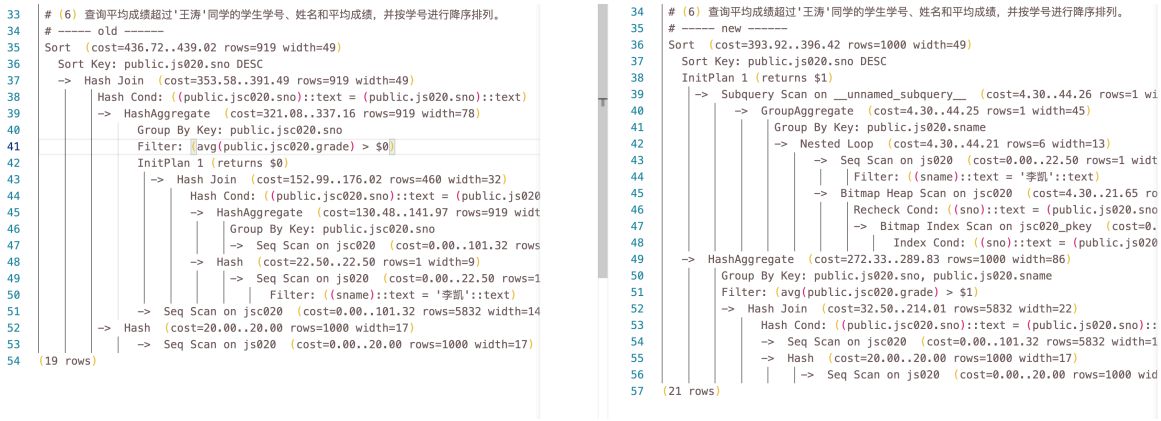


图 4.5 性能分析结果

(7) 查询平均成绩超过’王涛’同学的学生学号、姓名和平均成绩，并按学号进行降序排列

由于数据库中没有名为’王涛’的同学，此处改为’李凯’。

第一种方法，先计算 jsc020 表中每个人的平均成绩，然后找出姓名为’李凯’同学的平均成绩，通过 HAVING 保留大于该成绩的记录。然后将记录与 js020 内连接，排序。

```
SELECT js020.sno,
  js020.sname,
  avg.avg_grade
FROM js020
  INNER JOIN (
    SELECT jsc020.sno,
      avg(jsc020.grade) AS avg_grade
    FROM jsc020
    GROUP BY jsc020.sno
    HAVING (
        avg_grade > (
          SELECT avg_grade
          FROM (
```

```
14                SELECT jsc020.sno,
15                  avg(jsc020.grade) AS avg_grade
16                FROM jsc020
17                GROUP BY jsc020.sno
18              )
19            WHERE sno IN (
20                SELECT js020.sno
21                FROM js020
22                WHERE js020.sname = '李凯'
23              )
24          )
25        )
26    ) AS avg ON js020.sno = avg.sno
27 ORDER BY js020.sno DESC;
```

第二种方法，相比第一种在最后获取学号和姓名时直接左连接 js020 表。

```
1  SELECT final.sno,
2    js020.sname
3  FROM (
4      SELECT sno_avg.sno
5      FROM (
6          SELECT sno,
7            avg(grade) AS avg_grade
8          FROM jsc020
9          GROUP BY sno
10         HAVING COUNT (*) > 2
11         ORDER BY avg_grade
12       ) AS sno_avg
13     WHERE sno_avg.avg_grade = (
14         SELECT max(cno_count.avg_grade)
15         FROM (
16             SELECT avg(grade) AS avg_grade,
17               sno
18             FROM jsc020
19             GROUP BY sno
20             HAVING COUNT (*) > 2
21           ) AS cno_count
```

```
22          )
23    ) AS final
24    LEFT JOIN js020 ON js020.sno = final.sno;
```

通过在命令前添加EXPLIAN前缀分析两种方法的性能，两者差距不大，后者较前者性能稍好。

```
56  # (7) 查询选修了3门以上课程（包括3门）的学生中平均成绩最高的同学学号及姓名。
57  # ------ old ------
58  Hash Join  (cost=393.51..418.01 rows=500 width=17)
59    Hash Cond: ((js020.sno)::text = (__unnamed_subquery__.sno)::text)
60    InitPlan 1 (returns $0)
61      ->  Aggregate  (cost=184.91..184.92 rows=1 width=64)
62            ->  HashAggregate  (cost=159.64..173.43 rows=919 width=73)
63                  Group By Key: public.jsc020.sno
64                  Filter: (count(public.jsc020.cno) > 2)
65                  ->  Seq Scan on jsc020  (cost=0.00..101.32 rows=5832 width=25
66    ->  Seq Scan on js020  (cost=0.00..20.00 rows=1000 width=17)
67    ->  Hash  (cost=206.09..206.09 rows=200 width=9)
68          ->  HashAggregate  (cost=204.09..206.09 rows=200 width=9)
69                Group By Key: (__unnamed_subquery__.sno)::text
70                ->  Subquery Scan on __unnamed_subquery__  (cost=174.22..201.79
71                      ->  HashAggregate  (cost=174.22..192.60 rows=919 width=10
72                            Group By Key: public.jsc020.sno
73                            Filter: ((count(public.jsc020.cno) > 2) AND (avg(pu
74                            ->  Seq Scan on jsc020  (cost=0.00..101.32 rows=583
75  (17 rows)
76
```

```
59  # (7) 查询选修了3门以上课程（包括3门）的学生中平均成绩最高的同学学号及姓名。
60  # ------ new ------
61  Hash Left Join  (cost=362.48..402.69 rows=919 width=17)
62    Hash Cond: ((public.jsc020.sno)::text = (js020.sno)::text)
63    InitPlan 1 (returns $0)
64      ->  Aggregate  (cost=170.33..170.34 rows=1 width=64)
65            ->  HashAggregate  (cost=145.06..158.85 rows=919 width=54)
66                  Group By Key: public.jsc020.sno
67                  Filter: (count(*) > 2)
68                  ->  Seq Scan on jsc020  (cost=0.00..101.32 rows=5832 width=14)
69    ->  HashAggregate  (cost=159.64..178.02 rows=919 width=86)
70          Group By Key: public.jsc020.sno
71          Filter: ((count(*) > 2) AND (avg(public.jsc020.grade) = $0))
72          ->  Seq Scan on jsc020  (cost=0.00..101.32 rows=5832 width=14)
73    ->  Hash  (cost=20.00..20.00 rows=1000 width=17)
74          ->  Seq Scan on js020  (cost=0.00..20.00 rows=1000 width=17)
75  (14 rows)
76
```

图 4.6　性能分析结果

5 实验总结

本学期我们学习了《数据库系统》课程, 通过具体实验更细致的了解了 DBMS 的基本原理，对 SQL 也掌握地更加熟练。

首先，我们学习了 openGauss 数据库的搭建。我本身有一些 openEuler 系统的使用经验，在上学期因课外项目需要，学习过另一款华为开发的虚拟化容器引擎 iSulad。该引擎亦为 openEuler 独占，幸好最后通过学习官方跨平台编译文档，将引擎编译到了 ubuntu 上运行，实验才得以继续。我想 openGauss 应该也是这么回事，最后发现云和墨恩已经做好了这件事，并写成了 Dockerfile。于是本次实验我就使用了 Docker 容器管理 openGauss。

随后我学习了基本的 SQL 操作，包括建立数据库、建立用户、用户权限管理、建表、数据导入、查询操作，以及触发器的编写。这部分内容相对简单，按部就班就好。

最后我扩充了实验数据，测试了同一项查询，不同 SQL 语句的性能差异。在这里我学到了使用 python 库中的 ORM 工具，对数据库做到了增删改查控制。我将全校课程列入了爬取目录，生成了一批较为真实的假数据。最后也学到了一些 python 单元测试的知识。

实验还算比较顺利的完成。在第二次验收上，我还抢先回答了一个关于触发器的问题，为实验赢得了加分。通过这次实验，我发现在书本上学习过数据库的基本知识, 原以为差不多熟练掌握了, 但事实却不是这样。在实际实验时，一项 SQL 查询要想做到尽善尽美往往需要比较多的实验思考。

实验的末尾想感谢以下教授本课程和实验的何亮老师，真正做到了身为世范，为人师表。

附录 A　相关代码

A.1　插入预设数据

```
INSERT INTO js020 (Sno, SNAME, SEX, BDATE, Heihgt, DORM)
VALUES (
    '01032010',
    '王涛',
    '男',
    '1992-4-5',
    '1.72',
    '东14舍221'
  ),
  (
    '01032023',
    '孙文',
    '男',
    '1993-6-10',
    '1.80',
    '东14舍221'
  ),
  (
    '01032001',
    '张晓梅',
    '女',
    '1993-11-17',
    '1.58',
    '东1舍312'
  ),
  (
    '01032005',
    '刘静',
    '女',
    '1992-1-10',
```

```
    '1.63',
    '东1舍312'
  ),
  (
    '01032112',
    '董蔚',
    '男',
    '1992-2-20',
    '1.71',
    '东14舍221'
  ),
  (
    '03031011',
    '王倩',
    '女',
    '1993-12-20',
    '1.66',
    '东2舍104'
  ),
  (
    '03031014',
    '赵思扬',
    '男',
    '1991-6-6',
    '1.85',
    '东18舍421'
  ),
  (
    '03031051',
    '周剑',
    '男',
    '1991-5-8',
    '1.68',
    '东18舍422'
  ),
  (
```

```
    '03031009',
    '田婷',
    '女',
    '1992-8-11',
    '1.60',
    '东2舍104'
  ),
  (
    '03031033',
    '蔡明明',
    '男',
    '1992-3-12',
    '1.75',
    '东18舍423'
  );


  INSERT INTO jc020 (Cno, CNAME, PERIOD, CREDIT, TEACHER)
VALUES ('CS-01', '数据结构', '60', '3', '张军'),
  ('CS-02', '计算机组成原理', '80', '4', '王亚伟'),
  ('CS-04', '人工智能', '40', '2', '李蕾'),
  ('EE-01', '信号与系统', '40', '2', '张明'),
  ('EE-02', '数字逻辑电路', '100', '5', '胡海东');


  INSERT INTO jsc020 (Sno, Cno, GRADE)
VALUES ('01032010', 'CS-01', '82.0'),
  ('01032010', 'CS-02', '91.0'),
  ('01032010', 'CS-04', '83.5'),
  ('01032001', 'CS-01', '77.5'),
  ('01032001', 'CS-02', '85.0'),
  ('01032001', 'CS-04', '83.0'),
  ('01032005', 'CS-01', '62.0'),
  ('01032005', 'CS-02', '77.0'),
  ('01032005', 'CS-04', '82.0'),
  ('01032023', 'CS-01', '55.0'),
```

```
103   ('01032023', 'CS-02', '81.0'),
104   ('01032023', 'CS-04', '76.0'),
105   ('01032112', 'CS-01', '88.0'),
106   ('01032112', 'CS-02', '91.5'),
107   ('01032112', 'CS-04', '86.0'),
108   ('03031033', 'EE-01', '93.0'),
109   ('03031033', 'EE-02', '89.0'),
110   ('03031009', 'EE-01', '88.0'),
111   ('03031009', 'EE-02', '78.5'),
112   ('03031011', 'EE-01', '91.0'),
113   ('03031011', 'EE-02', '86.0'),
114   ('03031051', 'EE-01', '78.0'),
115   ('03031051', 'EE-02', '58.0'),
116   ('03031014', 'EE-01', '79.0'),
117   ('03031014', 'EE-02', '71.0');
```

A.2　生成假数据并插入到数据库

A.2.1　程序入口

https://github.com/magicwenli/db-generater/blob/master/main.py

A.2.2　数据库模型

https://github.com/magicwenli/db-generater/blob/master/models.py

A.2.3　其他扩展

https://github.com/magicwenli/db-generater/blob/master/extents.py