数据库实验报告

一、实验环境准备

本次数据库实验使用本地虚拟机的 OpenGauss：

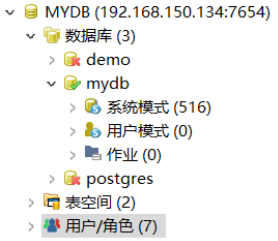
1. 在 VMWare 上安装 OpenEuler23，勾选安装 OpenGauss。

2. DataStudio 连接，不能使用原始的管理员用户，先要创建用户 CREATE USER testuser IDENTIFIED BY '密码'; ALTER USER testuser SYSADMIN;。

创建数据库：

1. 创建 MYDB 数据库：create database demo ENCODING 'UTF8' template = template0;

2. 使用 DataStudio 连接，使用 IP 192.168.150.134 设置端口为 7654。



二、基本表的创建

1. 创建 Student 表：

使用如下命令进行表创建，其中，考虑到学号 Sno 一般为定长，所以使用 Char 不使用 Varchar，升高的单位为米，使用小数点前一位，小数点后两位进行表示。

```
CREATE TABLE IF NOT EXISTS S049(
Sno Char(10) NOT NULL,
Sname VARCHAR(40) NOT NULL,
Sex Char(4) NOT NULL,
BDATE Date NOT NULL,
Height DEC(3,2) NOT NULL,
Dorm VARCHAR(32) NOT NULL,
PRIMARY KEY (Sno)
);
```

运行指令后返回的信息如下：

```
[2024-05-19 11:01:41.739 CST] : [INFO] 操作影响的记录行数 : 0
[2024-05-19 11:01:41.739 CST] : [INFO] 执行时间 : 64 ms
[2024-05-19 11:01:41.739 CST] : [INFO] 执行成功...
[2024-05-19 11:01:41.903 CST] : [NOTICE] CREATE TABLE / PRIMARY KEY will create implicit index "s049_pkey" for table "s049"
```

2. 创建 Course 表

使用如下命令进行 Course 表的创建，考虑到学时一般不会很大，使用 Smallint 类型，一门课的学分最多只有十分左右，使用 DEC(3,1)类型。

```
CREATE TABLE IF NOT EXISTS C049(
Cno Char(10) NOT NULL,
Cname VARCHAR(100) NOT NULL,
Period Smallint NOT NULL,
```

```
Credit DEC(3,1) NOT NULL,
Teacher VARCHAR(40) NOT NULL,
PRIMARY KEY (Cno)
);
```

运行指令后返回的信息如下：

```
[2024-05-19 11:10:30.193 CST] : [INFO] 操作影响的记录行数 : 0
[2024-05-19 11:10:30.193 CST] : [INFO] 执行时间 : 11 ms
[2024-05-19 11:10:30.193 CST] : [INFO] 执行成功...
[2024-05-19 11:10:30.431 CST] : [NOTICE] CREATE TABLE / PRIMARY KEY will create implicit index "c049_pkey" for table "c049"
```

3. 创建 SC 表

使用以下命令创建 SC 表，其中的 Grade 字段考虑会出现一位小数，最大会有 100 分，设置为 DEC(4,1)类型，表中的 Sno 和 Cno 为外键。

```
CREATE TABLE IF NOT EXISTS SC049(
Sno Char(10) NOT NULL,
Cno Char(10) NOT NULL,
Grade DEC(4,1) DEFAULT NULL,
PRIMARY KEY(Sno, Cno),
Foreign Key(Sno)
references "s049"
on delete cascade,
Foreign Key(Cno)
references "c049"
on delete restrict,
check((Grade is NULL) or (Grade Between 0 and 100))
);
```

运行指令后返回的信息如下：

```
[2024-05-19 11:26:20.084 CST] : [INFO] 操作影响的记录行数 : 0
[2024-05-19 11:26:20.084 CST] : [INFO] 执行时间 : 27 ms
[2024-05-19 11:26:20.084 CST] : [INFO] 执行成功...
[2024-05-19 11:26:20.225 CST] : [NOTICE] CREATE TABLE / PRIMARY KEY will create implicit index "sc049_pkey" for table "sc049"
```

4. 基本数据的插入

（1）使用如下语句进行 S049 表基本数据的插入

```
INSERT INTO S049 VALUES
('01032010','王涛','男','2003-4-5',1.72,'东 6 舍 221'),
('01032023','孙文','男','2004-6-10',1.8,'东 6 舍 221'),
('01032001','张晓梅','女','2004-11-17',1.58,'东 1 舍 312'),
('01032005','刘静','女','2003-1-10',1.63,'东 1 舍 312'),
('01032112','董蔚','男','2003-2-20',1.71,'东 6 舍 221'),
('03031011','王倩','女','2004-12-20',1.66,'东 2 舍 104'),
('03031014','赵思扬','男','2002-6-6',1.85,'东 18 舍 421'),
('03031051','周剑','男','2002-5-8',1.68,'东 18 舍 422'),
('03031009','田菲','女','2003-8-11',1.6,'东 2 舍 104'),
('03031033','蔡明明','男','2003-3-12',1.75,'东 18 舍 423'),
('03031056','曹子衿','女','2004-12-15',1.65,'东 2 舍 305');
```

返回结果：

```
[2024-05-19 11:40:37.004 CST] : [INFO] 操作影响的记录行数 : 11
[2024-05-19 11:40:37.004 CST] : [INFO] 执行时间 : 10 ms
[2024-05-19 11:40:37.004 CST] : [INFO] 执行成功...
```

（2）使用如下语句进行 C049 表基本数据的插入

```
INSERT INTO C049 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'),
('CS-05','深度学习',40,2,'崔昀'),
('EE-01','信号与系统',60,3,'张明'),
('EE-02','数字逻辑电路',100,5,'胡海东'),
('EE-03','光电子学与光子学',40,2,'石韬');
```

返回结果：

```
[2024-05-19 11:48:32.330 CST] : [INFO] 操作影响的记录行数 : 7
[2024-05-19 11:48:32.330 CST] : [INFO] 执行时间 : 6 ms
[2024-05-19 11:48:32.330 CST] : [INFO] 执行成功...
```

（3）使用如下语句进行 SC049 表基本数据的插入

```
INSERT INTO SC049 VALUES
('01032010','CS-01',82), ('01032010','CS-02',91), ('01032010','CS-04',83.5),
('01032001','CS-01',77.5), ('01032001','CS-02',85), ('01032001','CS-04',83),
('01032005','CS-01',62), ('01032005','CS-02',77), ('01032005','CS-04',82),
('01032023','CS-01',55), ('01032023','CS-02',81), ('01032023','CS-04',76),
('01032112','CS-01',88), ('01032112','CS-02',91.5), ('01032112','CS-04',86),
('01032112','CS-05',NULL), ('03031033','EE-01',93), ('03031033','EE-02',89),
('03031009','EE-01',88), ('03031009','EE-02',78.5), ('03031011','EE-01',91),
('03031011','EE-02',86), ('03031051','EE-01',78), ('03031051','EE-02',58),
('03031014','EE-01',79), ('03031014','EE-02',71);
```

返回结果：

```
[2024-05-19 11:52:59.409 CST] : [INFO] 操作影响的记录行数 : 7
[2024-05-19 11:52:59.409 CST] : [INFO] 执行时间 : 5 ms
[2024-05-19 11:52:59.409 CST] : [INFO] 执行成功...
[2024-05-19 11:56:59.387 CST] : [INFO] 操作影响的记录行数 : 26
[2024-05-19 11:56:59.387 CST] : [INFO] 执行时间 : 8 ms
[2024-05-19 11:56:59.387 CST] : [INFO] 执行成功...
```

三、数据操作

1. 查询操作

（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。

```
SELECT Cno,Cname,Credit
FROM c049
WHERE CNO LIKE 'EE-%';
```

	cno	cname	credit
1	***EE-01***	信号与系统	3.0
2	EE-02	数字逻辑电路	5.0
3	EE-03	光电子学与光子学	2.0

（2）查询未选修课程“CS-02”的女生学号及其已选各课程编号、成绩。

```
SELECT s049.Sno, sc049.Cno, sc049.Grade
FROM sc049, s049
WHERE sc049.Sno = s049.Sno And s049.Sno NOT IN(
SELECT Sno
FROM sc049
WHERE Cno = 'CS-02')
AND Sex='女';
```

	sno	cno	grade
1	***03031011***	EE-01	91.0
2	03031011	EE-02	86.0
3	03031009	EE-01	88.0
4	03031009	EE-02	78.5

（3）查询 2002 年～2003 年出生学生的基本信息。

```
SELECT *
FROM s049
WHERE BDATE BETWEEN '2002-01-01' AND '2003-12-31';
```

	sno	sname	sex	bdate	height	dorm
1	***01032010***	王涛	男	2003-04-05 00:00:00	1.72	东6舍221
2	01032005	刘静	女	2003-01-10 00:00:00	1.63	东1舍312
3	01032112	董蔚	男	2003-02-20 00:00:00	1.71	东6舍221
4	03031014	赵思扬	男	2002-06-06 00:00:00	1.85	东18舍421
5	03031051	周剑	男	2002-05-08 00:00:00	1.68	东18舍422
6	03031009	田菲	女	2003-08-11 00:00:00	1.60	东2舍104
7	03031033	蔡明明	男	2003-03-12 00:00:00	1.75	东18舍423

（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 s049.Sno,s049.Sname,SUM(Credit) as "total credits"
FROM sc049,s049,c049
WHERE sc049.Sno = s049.Sno AND sc049.Cno = c049.Cno
GROUP BY s049.Sno;
```

	sno	sname	total credits
1	***01032001***	张晓梅	9.0
2	01032005	刘静	9.0
3	01032010	王涛	9.0
4	01032023	孙文	9.0
5	01032112	董蔚	11.0
6	03031009	田菲	8.0
7	03031011	王倩	8.0
8	03031014	赵思扬	8.0
9	03031033	蔡明明	8.0
10	03031051	周剑	8.0

（5）查询选修课程“CS-01”的学生中成绩第二高的学生学号。

```
SELECT sc1.Sno
FROM sc049 sc1
```

```
WHERE sc1.Cno = 'CS-01'
AND sc1.Grade = (
SELECT MAX(Grade)
FROM sc049 sc2
WHERE sc2.Cno = 'CS-01'
AND sc2.grade < (
   SELECT MAX ( GRADE )
   FROM sc049 sc3
   WHERE sc3.Cno = 'CS-01')
);
```

	sno
1	01032010

（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 s049.Sno,s049.Sname, AVG(Grade) AS "Average Grade"
FROM sc049, s049
WHERE sc049.Sno=s049.Sno And GRADE IS NOT NULL
GROUP BY s049.Sno
HAVING AVG(Grade) > ALL(
   SELECT AVG(Grade)
   FROM sc049 JOIN s049 USING(Sno)
   WHERE Sname='王涛' AND Grade IS NOT NULL
)
ORDER BY Sno DESC;
```

	sno	sname	Average Grade
1	03031033	蔡明明	91.0000000000000000
2	03031011	王倩	88.5000000000000000
3	01032112	董蔚	88.5000000000000000

（7）查询选修计算机专业全部课程（课程编号为“CS-××”）的学生姓名及已获得学分。

```
SELECT Sname,SUM(CREDIT) as "total credits"
FROM s049,sc049,c049
WHERE c049.Cno LIKE 'CS-%' AND
c049.Cno = sc049.Cno AND s049.Sno = sc049.Sno AND Grade BETWEEN 60 AND 100
GROUP BY s049.Sno
HAVING COUNT(*) = (
   SELECT COUNT(Cno)
   FROM c049
   WHERE Cno LIKE 'CS-%'
);
```

	sname	total credits

这里查询结果为空，这题考虑分数大于 60 分才能获得学分。除了上面的方法，还可以在

SUM 函数中用 CASE 语句来判断是否获得学分，后面将会对这两种方法进行查询性能比较。

（8）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。

```
SELECT s049.Sno,s049.Sname
FROM s049,sc049
WHERE s049.Sno=sc049.Sno AND GRADE IS NOT NULL
GROUP BY s049.Sno
HAVING AVG(GRADE) = (
   SELECT AVG(GRADE)
   FROM s049,sc049
   WHERE s049.sno = sc049.sno AND GRADE IS NOT NULL
   GROUP BY s049.SNO
   HAVING COUNT(*) >= 3
   ORDER BY AVG(GRADE) DESC
   LIMIT 1
);
```

	sno	sname
1	***01032112***	董蔚
2	03031011	王倩

注：可以使用 LIMIT 1 取出排序的第一行

limit n //获得前 n 行

limit m,n //跳过第 m 行，从第 m+1 行开始获取 n 行

2. 数据插入

分别在 S 和 C 表中加入记录（‘01032005’，‘刘竞’，‘男’，‘2003-12-10’,1.75,‘东 14 舍 312’）及(‘CS-03’，“离散数学”,64,4,‘陈建明’)。使用如下语句进行实现。

```
INSERT INTO s049 VALUES
('01032005','刘竞','男','2003-12-10',1.75,'东 14 舍 312');
```

运行结果如下，可以发现由于主键的冲突，插入并没有成功。

```
[2024-05-19 19:05:03.462 CST] : [ERROR] 执行失败
错误代码：  [0]SQL错误码: = 23505
ERROR: duplicate key value violates unique constraint "s049_pkey"
  Detail: Key (sno)=(01032005  ) already exists.
Line Number: 74
```

```
INSERT INTO c049 VALUES
('CS-03','离散数学',64,4,'陈建明');
```

运行结果如下：

```
[2024-05-19 19:05:38.818 CST] : [INFO] 操作影响的记录行数 : 1
[2024-05-19 19:05:38.818 CST] : [INFO] 执行时间 : 3 ms
[2024-05-19 19:05:38.818 CST] : [INFO] 执行成功...
```

3. 数据删除

将 S049 表中已修学分数大于 60 的学生记录删除。使用如下语句进行实现。

```
DELETE FROM s049
WHERE Sno IN(
```

```
    SELECT Sno
    FROM sc049,c049
    WHERE sc049.Cno = c049.Cno AND Grade BETWEEN 60 AND 100
    GROUP BY Sno
    HAVING SUM(CREDIT) > 60
);
```

运行结果如下，没有发现满足要求的元祖，所以没有删除记录。

```
[2024-05-19 19:10:28.395 CST] : [INFO] 操作影响的记录行数 : 0
[2024-05-19 19:10:28.395 CST] : [INFO] 执行时间 : 6 ms
[2024-05-19 19:10:28.395 CST] : [INFO] 执行成功...
```

4. 数据修改

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。使用如下语句进行实现。

```
UPDATE c049
SET CREDIT = CREDIT+1, PERIOD = 64
WHERE CNAME='信号与系统' AND teacher='张明';
```

运行结果如下：

```
[2024-05-19 19:12:00.641 CST] : [INFO] 操作影响的记录行数 : 1
[2024-05-19 19:12:00.641 CST] : [INFO] 执行时间 : 7 ms
[2024-05-19 19:12:00.641 CST] : [INFO] 执行成功...
```

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。

```
CREATE VIEW EAST_18_MAN
AS SELECT Sno,Sname,Bdate,Height
FROM s049
WHERE SEX='男' AND DORM LIKE '东 18 舍%';
```

运行结果如下：

```
[2024-05-19 19:19:25.991 CST] : [INFO] 操作影响的记录行数 : 0
[2024-05-19 19:19:25.991 CST] : [INFO] 执行时间 : 8 ms
[2024-05-19 19:19:25.991 CST] : [INFO] 执行成功...
```

	sno	sname	bdate	height
1	***03031014***	赵思扬	2002-06-06 00:00:00	1.85
2	03031051	周剑	2002-05-08 00:00:00	1.68
3	03031033	蔡明明	2003-03-12 00:00:00	1.75

（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。

```
CREATE VIEW COURSES_OF_ZHANGMING
AS SELECT c049.Cno,Cname,AVG(GRADE) AS "Average Grade"
FROM c049,sc049
WHERE TEACHER='张明' AND c049.Cno = sc049.Cno
GROUP BY c049.Cno;
```

运行结果如下：

```
[2024-05-19 19:24:10.068 CST] : [INFO] 操作影响的记录行数 : 0
[2024-05-19 19:24:10.068 CST] : [INFO] 执行时间 : 7 ms
[2024-05-19 19:24:10.068 CST] : [INFO] 执行成功...
```

	cno	cname	Average Grade
1	EE-01	信号与系统	85.8000000000000000

（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。

```
CREATE VIEW AI_STUDENT
AS SELECT s049.Sno,Sname,GRADE
FROM sc049,s049,c049
WHERE Cname='人工智能' AND s049.Sno = sc049.Sno
AND c049.Cno = sc049.Cno;
```

运行结果如下：

```
[2024-05-19 19:32:05.863 CST] : [INFO] 操作影响的记录行数 : 0
[2024-05-19 19:32:05.863 CST] : [INFO] 执行时间 : 5 ms
[2024-05-19 19:32:05.863 CST] : [INFO] 执行成功...
```

	sno	sname	grade
1	01032010	王涛	83.5
2	01032001	张晓梅	83.0
3	01032005	刘静	82.0
4	01032023	孙文	76.0
5	01032112	董蔚	86.0

四、数据扩充

1. JDBC 的使用

这部分实验使用 JDBC 进行数据操作，向 IDEA 中添加 gsjdbc4.jar 和 gsjdbc200.jar 即可使用 JDBC 连接数据库，比如使用 JDBC 取出 SC049 表的数据的结果如下所示。（本实验中所有用到的 JAVA 代码见附录）

```
"C:\Program Files\Java\jdk1.8.0_351\bin\java.exe" ...
连接数据库...
 实例化Statement对象...
Sno: 01032010  , Cno: CS-01     , Grade: 82.0
Sno: 01032010  , Cno: CS-02     , Grade: 91.0
Sno: 01032010  , Cno: CS-04     , Grade: 83.5
Sno: 01032001  , Cno: CS-01     , Grade: 77.5
Sno: 01032001  , Cno: CS-02     , Grade: 85.0
Sno: 01032001  , Cno: CS-04     , Grade: 83.0
Sno: 01032005  , Cno: CS-01     , Grade: 62.0
Sno: 01032005  , Cno: CS-02     , Grade: 77.0
Sno: 01032005  , Cno: CS-04     , Grade: 82.0
Sno: 01032023  , Cno: CS-01     , Grade: 55.0
```

2. 向三个基本表中添加数据

（1）数据获取：使用 Python 爬虫在姓名大全中获取姓名，在中国大学 MOOC 中爬取数据作为课程名。其他数据，比如课程号或者学号使用随机生成。（Python 代码在附录中）

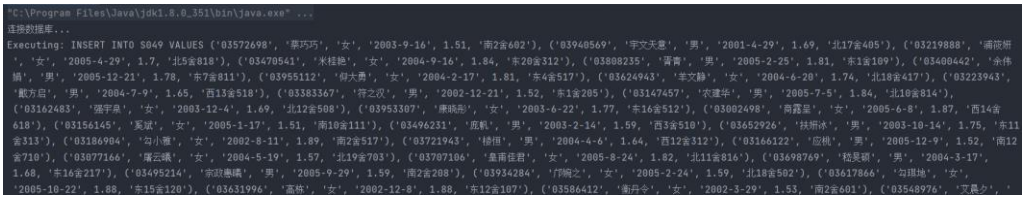
（2）生成需要执行的 SQL 语句：根据获取的数据，生成 C049_1，C049_2，S049_1，S049_2，SC049_1，SC049_2 和 delete_less_than_60 这 7 个文本文件，其中存储了插入和删除的 SQL 语句，XXX_1 表示第一次向表 XXX 中插入的数据，用于这一部分的实验，XXX_2 表示进一步添加的数据，用于后续优化 SQL 语句的实验，delete_less_than_60 中存储了删除语句，可以删除 SC049 中分数小于 60 的数据。实验中需要注意的是 SC049_1 中的需要插入的数据的 Sno 和 Cno 必定要再 C049 和 S049 出现，以满足完整性约束。SC049_1 和 delete_less_than_60 的部分数据如下所示，SC049_1 中的数据会分成 200 次进行插入，在

插入的同时进行删除操作。（JAVA 代码在附录中）

```
INSERT INTO SC049 VALUES
('03088101', 'BI-49', 43),
('03971845', 'BI-82', 65.5),
('03818298', 'CS-52', 67),
('03750043', 'AI-63', 60),
('03122174', 'MA-29', 67.5),
('03904953', 'MT-77', 41.5),
('03466746', 'SE-11', 94.5),
('03902335', 'MA-43', 53),
('03853441', 'PH-76', 46),
('03103731', 'CH-12', 46.5),
```

```
DELETE FROM SC049 WHERE (Sno, Cno, GRADE) IN (
('01032023', 'CS-01', 55.0),
('03031051', 'EE-02', 58.0),
('03088101', 'BI-49', 43),
('03853441', 'PH-76', 46),
('03103731', 'CH-12', 46.5),
('03664684', 'MT-77', 40.5),
('03818147', 'MU-70', 49),
('03210443', 'AI-62', 59.5),
('03684535', 'AI-63', 50.5),
('03935794', 'AI-07', 40),
```

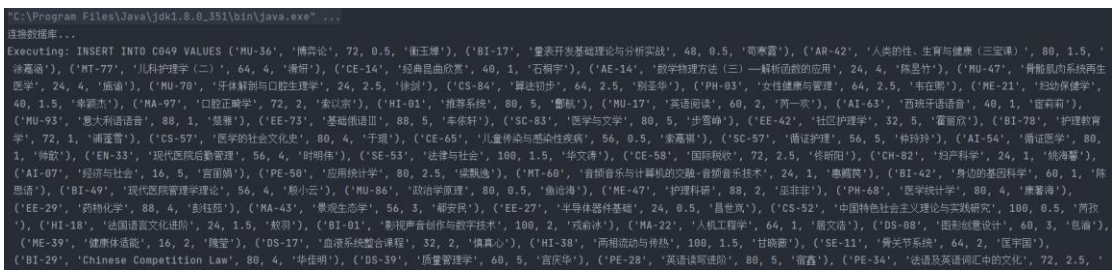
（3）S 和 C 表的数据添加：使用 JDBC 向 S 和 C 表中加入数据。向 S049 表中加入数据的程序输出如下。



从 DataStudio 中可以看到 S049 数据已经有了 1011 行。同时可以发现，DataStudio 以一千行为单位进行数据的装载，并非直接导入完整的数据。

993	03896125	利剑龙	女	2003-07-19 00:00:00	1.85	西17舍510
994	03034136	屠真	女	2004-10-31 00:00:00	1.65	北1舍511
995	03737229	龚泉	女	2005-07-25 00:00:00	1.60	南8舍314
996	03948239	罗一纾	女	2004-10-01 00:00:00	1.68	北18舍214
997	03584661	仝花琪	女	2002-12-15 00:00:00	1.79	南2舍813
998	03962920	贺文杰	女	2002-09-08 00:00:00	1.88	东5舍707
999	03650684	厍饿	男	2004-01-31 00:00:00	1.78	东20舍308
1000	03262957	冯骊华	男	2003-04-02 00:00:00	1.67	北5舍110
1001	03340349	凌睿翱	女	2001-09-13 00:00:00	1.70	南1舍506
1002	03904953	司空瑞铮	男	2003-06-24 00:00:00	1.76	南9舍603
1003	03887367	宓波	男	2001-01-20 00:00:00	1.65	东18舍218
1004	03367251	古晓媛	男	2005-07-22 00:00:00	1.76	东16舍707
1005	03410599	殷晨萱	女	2005-10-24 00:00:00	1.50	东5舍418
1006	03312545	巫慧妮	女	2003-04-02 00:00:00	1.90	东14舍614
1007	03954916	俞怡沁	女	2001-03-31 00:00:00	1.72	南3舍802
1008	03457562	鄢妍淇	男	2002-07-26 00:00:00	1.56	北19舍802
1009	03853235	郑允浩	女	2004-08-19 00:00:00	1.53	北17舍714
1010	03707078	冀雪冬	男	2003-07-22 00:00:00	1.89	西12舍607
1011	03272137	隋静璇	男	2005-04-13 00:00:00	1.76	北7舍618

同理，向 C049 中加入数据的程序输出如下。



从 DataStudio 中可以看到 C049 数据已经有了 108 行。

92	SC-38	核医学	80	0.5	闻明
93	EE-41	电力系统继电保护	88	1.5	齐凌菲
94	CS-62	大学物理-热学、电学	32	1.0	蔡春花
95	MA-32	出版学基础	56	1.5	松善善
96	ME-71	大地测量学基础	100	2.0	羿茗迪
97	AR-08	地球科学概论	88	1.0	牟钰琳
98	PE-88	损伤与反应	60	4.0	终结者
99	MT-30	档案学基础	40	5.0	元哲贤
100	AE-62	印刷材料	16	4.0	练新瑞
101	PH-49	大学生物学（通识版）	100	1.0	易菲
102	CH-12	恢复生态学	60	1.5	充建伟
103	EN-25	运动损伤诊疗	80	1.0	苑霞薇
104	MA-02	临床研究方法学	24	2.5	钱淋西
105	DS-74	临床变态反应学	64	2.5	辛精洗
106	SC-17	癌症的预防与筛查	80	2.5	溥弋
107	CH-95	工匠书缘：古籍的修复与文化传承	56	4.0	武嘉琦
108	PH-76	收益管理	80	4.0	付家可

（4）向 SC 表中加入数据，同时进行数据删除的执行

这里使用多线程在数据插入的同时进行数据的删除操作，主要代码如下，其中的 ExecSQL_Times 表示每读入 100 行数据的时候执行一次插入语句，这么做是因为如果一次读入全部数据插入，同时又执行删除操作，两个操作看似由于多线程在同时进行，实则是执行有先后，数据库会优先执行其中的某一条语句，就没有在插入的同时进行删除的效果了。ExexSQL 用于执行删除操作。

```
Connection finalConn = conn;
Thread t1 = new Thread(new Runnable() {
    @Override
    public void run() {
        try {
            ExecSQL_Times(finalConn, filename: "./scripts/SC049_1.txt", lines: 100);
        } catch (IOException | InterruptedException e) {
            e.printStackTrace();
        }
    }
});

Thread t2 = new Thread(new Runnable() {
    @Override
    public void run() {
        try {
            TimeUnit.SECONDS.sleep( timeout: 2);
            ExecSQL(finalConn, filename: "./scripts/delete_less_than_60.txt");
        } catch (IOException | InterruptedException e) {
            e.printStackTrace();
        }
    }
});

t1.start();
t2.start();

t1.join();
t2.join();
```

运行代码后程序输出如下，发现程序在进行数据添加的过程中执行了数据的删除操作。

```
Executing: INSERT INTO SC049 VALUES('03466746', 'PE-88', 97), ('03893224', 'EE-42', 41), ('03334894', 'AE-51', 79), ('03459096', 'IE-62',
Executing: INSERT INTO SC049 VALUES('03023343', 'SC-83', 72), ('03542435', 'EE-27', 42.5), ('03703858', 'EE-51', 99.5), ('03891118', 'MU-5
Executing: INSERT INTO SC049 VALUES('03980293', 'CS-57', 60.5), ('03602970', 'CE-58', 59.5), ('03881088', 'AE-51', 44.5), ('03192981', 'BI
Executing: INSERT INTO SC049 VALUES('03733542', 'MT-30', 52.5), ('03046117', 'MA-29', 43.5), ('03534919', 'AI-54', 89.5), ('03021777', 'EE
SQL: DELETE
Executing: INSERT INTO SC049 VALUES('03052842', 'MT-13', 81.5), ('03023343', 'SE-94', 73), ('03390044', 'HI-18', 99), ('03632746', 'CH-12'
Executing: INSERT INTO SC049 VALUES('03950264', 'BI-82', 74.5), ('03596987', 'ME-71', 74.5), ('03721943', 'CS-52', 57.5), ('03139610', 'MA
Executing: INSERT INTO SC049 VALUES('03410599', 'PH-99', 80.5), ('03602229', 'CS-62', 79), ('03451006', 'ME-23', 60.5), ('03690431', 'MT-7
Executing: INSERT INTO SC049 VALUES('03782669', 'PH-49', 77.5), ('03080648', 'AI-63', 96), ('03181870', 'MT-15', 58), ('03186904', 'MA-43'
Executing: INSERT INTO SC049 VALUES('03533750', 'MU-55', 96), ('03286000', 'SC-36', 94.5), ('03728810', 'BI-82', 96), ('03325179', 'HI-01'
Executing: INSERT INTO SC049 VALUES('03820822', 'CH-82', 98), ('03527118', 'MA-43', 92), ('03123995', 'EE-42', 83.5), ('03173885', 'HI-38'
Executing: INSERT INTO SC049 VALUES('03048517', 'CS-62', 63), ('03383367', 'ME-71', 50.5), ('03464873', 'ME-55', 95), ('03496231', 'AI-07'
Executing: INSERT INTO SC049 VALUES('03765863', 'CH-12', 75), ('03863286', 'AR-08', 89), ('03471418', 'CS-62', 71.5), ('03555951', 'BI-42'
```

执行之后，此时 SC 表中数据有 19826 条。

```
Sno: 03624797  , Cno: CS-84     , Grade: 63.0
Sno: 03017426  , Cno: MU-93     , Grade: 43.5
Sno: 03756125  , Cno: MA-43     , Grade: 84.0
Sno: 03880789  , Cno: MA-22     , Grade: 76.0
Total: 19826
```

（5）插入剩余数据

使用 JDBC 运行 S049_2，C049_2 和 SC049_2 中的 SQL 语句，程序输出如下。

```
连接数据库...
SQL: INSERT
SQL: INSERT
SQL: INSERT

进程已结束,退出代码0
```

使用程序统计此时各个表中的数据，S049 表中目前有 5011 条记录。

```
Sno: 03362053  , Sname: 佟琳怡
Sno: 03823327  , Sname: 尚鑫钥
Sno: 03298815  , Sname: 太叔惠曦
Sno: 03553477  , Sname: 庞国棉
Sno: 03400100  , Sname: 栗铭
Sno: 03092141  , Sname: 冉起鹦
Sno: 03869382  , Sname: 付婷帛
Total: 5011
```

统计 C049 表，目前表中有 1007 条数据。

```
Cno: MU-91     , Cname: 英语教学理论与实践, Period: 16, Credit: 3.0
Cno: HI-53     , Cname: 中国神话, Period: 16, Credit: 1.0
Cno: EE-54     , Cname: 文化翻译说·具象篇, Period: 60, Credit: 0.5
Cno: MT-44     , Cname: 意大利语会话（基础篇）, Period: 60, Credit: 4.0
Cno: PH-61     , Cname: 俄汉翻译, Period: 100, Credit: 1.0
Cno: AR-29     , Cname: 基础瑞典语, Period: 72, Credit: 2.0
Cno: EN-60     , Cname: 职场英语写作, Period: 24, Credit: 2.5
Cno: EE-67     , Cname: 语言跨学科研究方法, Period: 88, Credit: 1.0
Total: 1007
```

统计 SC049 表，目前表中有 199826 条数据。

```
Sno: 03515418  , Cno: AI-83     , Grade: 45.5
Sno: 03515418  , Cno: AR-48     , Grade: 93.0
Sno: 03515418  , Cno: AR-34     , Grade: 99.0
Sno: 03515418  , Cno: SC-43     , Grade: 84.0
Sno: 03515418  , Cno: AI-90     , Grade: 85.0
Sno: 03515418  , Cno: DS-07     , Grade: 54.0
Sno: 03515418  , Cno: PE-51     , Grade: 41.5
Sno: 03515418  , Cno: DS-86     , Grade: 92.0
Total: 199826
```

五、不同语句的效率分析

实验的本部分为第三节中的数据查询部分的（2）（5）（6）（7）（8）查询编写了不同的 SQL 查询语句并进行了效率分析。

（2）查询未选修课程“CS-02”的女生学号及其已选各课程编号、成绩。

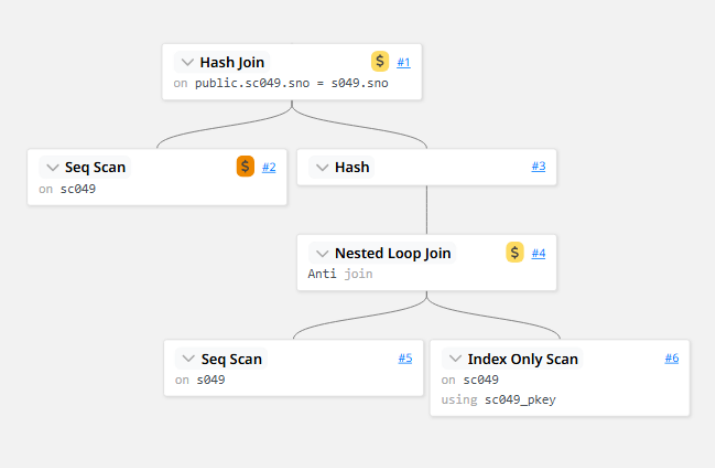
实现 1：

```
SELECT s049.Sno, sc049.Cno, sc049.Grade
FROM sc049, s049
WHERE sc049.Sno = s049.Sno And s049.Sno NOT IN(
SELECT Sno
FROM sc049
WHERE Cno = 'CS-02')
AND Sex='女';
```

运行时间:0.253s

上述 SQL 的执行过程如下：

```
Hash Join  (cost=1479.83..8035.16 rows=232576 width=27)
  Hash Cond: (public.sc049.sno = s049.sno)
  ->  Seq Scan on sc049  (cost=0.00..3480.26 rows=199826 width=27)
  ->  Hash  (cost=1451.15..1451.15 rows=2294 width=11)
        ->  Nested Loop Anti Join  (cost=0.00..1451.15 rows=2294 width=11)
              ->  Seq Scan on s049  (cost=0.00..119.64 rows=2505 width=11)
                    Filter: (sex = '女'::bpchar)
              ->  Index Only Scan using sc049_pkey on sc049  (cost=0.00..4.51 rows=182 width=11)
                    Index Cond: ((sno = s049.sno) AND (cno = 'CS-02'::bpchar))
```



实现 2：

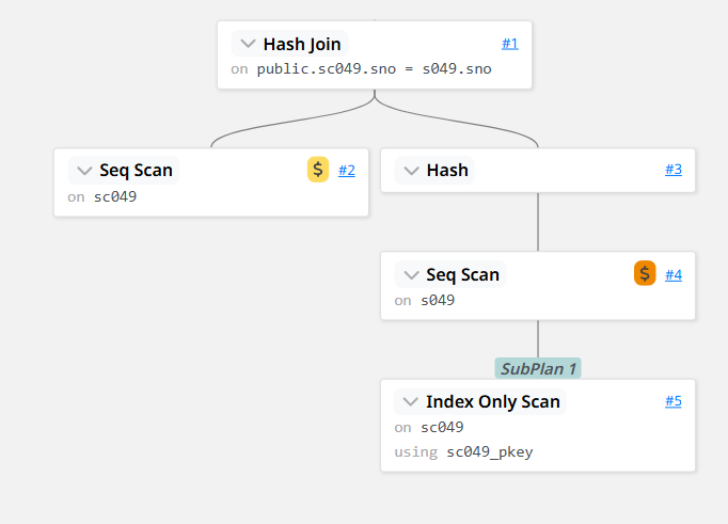
```
SELECT SC049.Sno, SC049.Cno, SC049.Grade
FROM SC049 JOIN (SELECT S049.SNO FROM S049 WHERE Sex='女') AS sc ON SC049.Sno=sc.Sno
WHERE sc.Sno NOT IN (SELECT Sno FROM SC049 WHERE Cno = 'CS-02' AND sc.Sno = sc049.Sno);
```

运行时间：0.205s

上述 SQL 的执行过程如下：

```
Hash Join  (cost=20874.58..26374.51 rows=127033 width=27)
  Hash Cond: (public.sc049.sno = s049.sno)
  ->  Seq Scan on sc049  (cost=0.00..3480.26 rows=199826 width=27)
  ->  Hash  (cost=20858.91..20858.91 rows=1253 width=11)
```

```
->  Seq Scan on s049  (cost=0.00..20858.91 rows=1253 width=11)
      Filter: ((sex = '女'::bpchar) AND (NOT (SubPlan 1)))
      SubPlan 1
        ->  Index Only Scan using sc049_pkey on sc049  (cost=0.00..8.27 rows=1 width=11)
              Index Cond: ((sno = s049.sno) AND (cno = 'CS-02'::bpchar))
```



运行过程分析：

实现 1 和实现 2 实现总体思路类似，均使用了 Hash Join 操作来连接 sc049 和 s049 表，在 s049 和 sc049 表上使用了 Seq Scan 操作。但是实现 2 中多了一层子查询,用于先找出 s049 表中 Sex='女' 的学生,然后在 sc049 表中进行过滤和连接，这样使得连接的开销有所减小，使得实现 2 的计算较快。

（5）查询选修课程“CS-01”的学生中成绩第二高的学生学号。

实现 1：

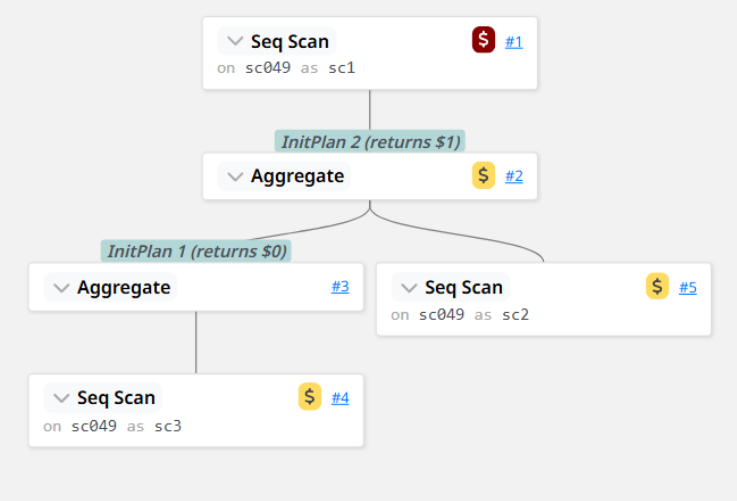
```
SELECT sc1.Sno
FROM sc049 sc1
WHERE sc1.Cno = 'CS-01'
AND sc1.Grade = (
   SELECT MAX(Grade)
   FROM sc049 sc2
   WHERE sc2.Cno = 'CS-01'
   AND sc2.grade < (
      SELECT MAX ( GRADE )
      FROM sc049 sc3
      WHERE sc3.Cno = 'CS-01'
   )
);
```

运行时间：0.156s

上述 SQL 的执行过程如下：

```
Seq Scan on sc049 sc1  (cost=8459.84..12939.23 rows=2 width=11)
  Filter: ((cno = 'CS-01'::bpchar) AND (grade = $1))
  InitPlan 2 (returns $1)
```

```
->  Aggregate  (cost=8459.83..8459.84 rows=1 width=37)
      InitPlan 1 (returns $0)
        ->  Aggregate  (cost=3980.28..3980.29 rows=1 width=37)
              ->  Seq Scan on sc049 sc3  (cost=0.00..3979.83 rows=182 width=5)
                    Filter: (cno = 'CS-01'::bpchar)
      ->  Seq Scan on sc049 sc2  (cost=0.00..4479.39 rows=61 width=5)
            Filter: ((grade < $0) AND (cno = 'CS-01'::bpchar))
```



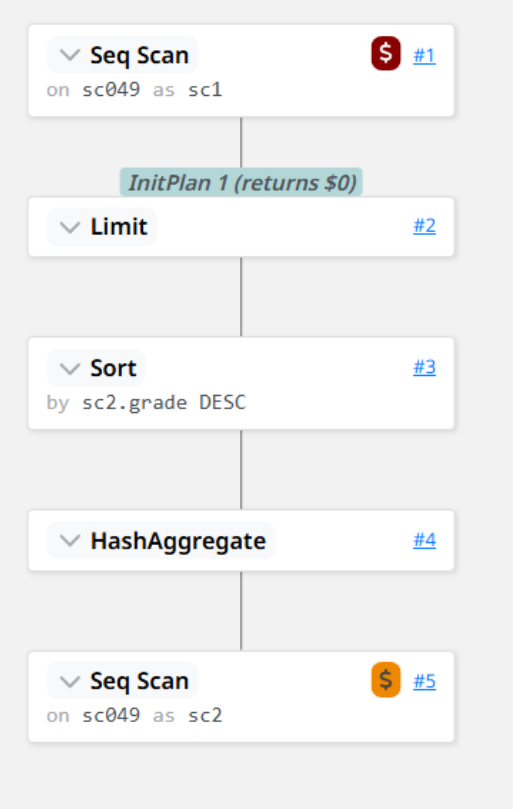
实现 2：

```
SELECT Sno
FROM sc049 sc1
WHERE sc1.cno = 'CS-01'
and sc1.grade = (
   SELECT DISTINCT grade
   FROM sc049 sc2
   WHERE sc2.cno = 'CS-01'
   ORDER BY grade DESC
   LIMIT 1,1
);
```

运行时间：0.08s

上述 SQL 的执行过程如下：

```
Seq Scan on sc049 sc1  (cost=3980.31..8459.69 rows=2 width=11)
  Filter: ((cno = 'CS-01'::bpchar) AND (grade = $0))
  InitPlan 1 (returns $0)
    ->  Limit  (cost=3980.31..3980.31 rows=1 width=5)
          ->  Sort  (cost=3980.30..3980.31 rows=1 width=5)
                Sort Key: sc2.grade DESC
                ->  HashAggregate  (cost=3980.28..3980.29 rows=1 width=5)
                      Group By Key: sc2.grade
                      ->  Seq Scan on sc049 sc2  (cost=0.00..3979.83 rows=182 width=5)
                            Filter: (cno = 'CS-01'::bpchar)
```



运行过程分析：

实现 1 使用了三层嵌套查询，主要使用了 Seq Scan 操作,没有用到任何索引。实现 2 使用了一个子查询来找到第二高的成绩,而不是像实现 1 那样嵌套多个子查询，执行计划同样主要使用了 Seq Scan 操作,没有用到任何索引，但是用到了排序 sort。总体来说实现 2 的查询嵌套更少，效率更高

（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。

实现 1：

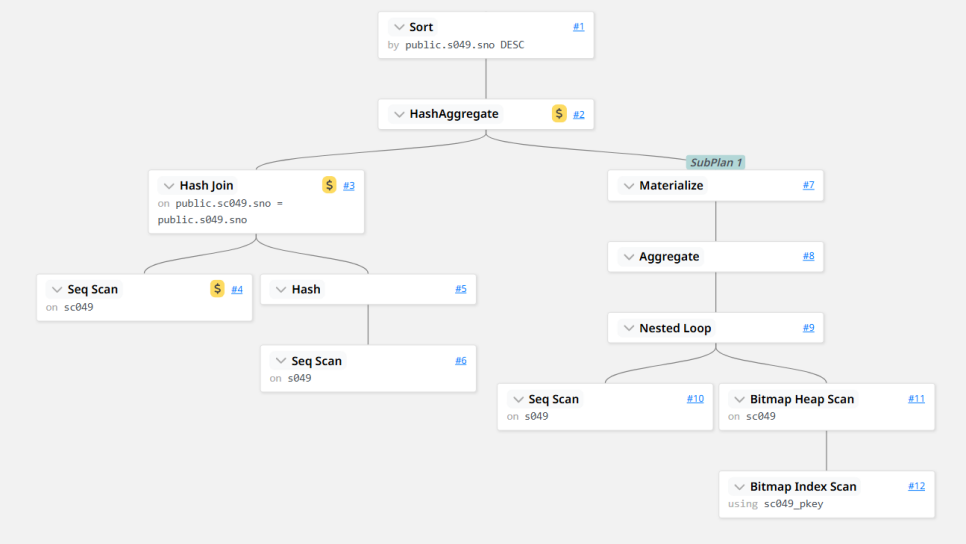
```
SELECT s049.Sno,s049.Sname, AVG(Grade) AS "Average Grade"
FROM sc049, s049
WHERE sc049.Sno=s049.Sno And GRADE IS NOT NULL
GROUP BY s049.Sno
HAVING AVG(Grade) > ALL(
   SELECT AVG(Grade)
   FROM sc049, s049
   WHERE s049.Sname='王涛' AND sc049.Sno=s049.Sno AND Grade IS NOT NULL)
ORDER BY Sno DESC;
```

运行时间：0.336s

上述 SQL 的执行过程如下：

```
Sort  (cost=9037.23..9049.75 rows=5011 width=89)
  Sort Key: public.s049.sno DESC
  ->  HashAggregate  (cost=8597.74..8729.28 rows=5011 width=89)
        Group By Key: public.s049.sno
        Filter: (SubPlan 1)
```

```
-> Hash Join  (cost=169.75..6396.62 rows=199826 width=25)
      Hash Cond: (public.sc049.sno = public.s049.sno)
      -> Seq Scan on sc049  (cost=0.00..3479.26 rows=199826 width=16)
            Filter: (grade IS NOT NULL)
      -> Hash  (cost=107.11..107.11 rows=5011 width=20)
            -> Seq Scan on s049  (cost=0.00..107.11 rows=5011 width=20)
SubPlan 1
  -> Materialize  (cost=702.43..702.44 rows=1 width=37)
        -> Aggregate  (cost=702.43..702.44 rows=1 width=37)
              -> Nested Loop  (cost=9.86..701.91 rows=208 width=5)
                    -> Seq Scan on s049  (cost=0.00..119.64 rows=1 width=11)
                          Filter: ((sname)::text = '王涛'::text)
                    -> Bitmap Heap Scan on sc049  (cost=9.86..580.19 rows=208 width=16)
                          Recheck Cond: (sno = public.s049.sno)
                          Filter: (grade IS NOT NULL)
                          -> Bitmap Index Scan on sc049_pkey  (cost=0.00..9.81 rows=208 width=0)
                                Index Cond: (sno = public.s049.sno)
```



实现 2：

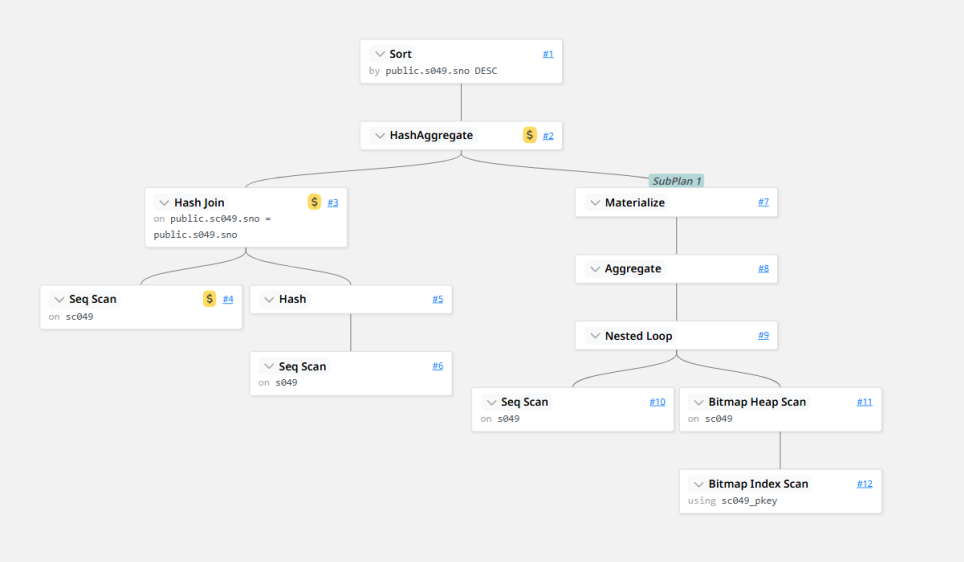
```
SELECT s049.Sno,s049.SNAME, AVG(GRADE) AS "Average Grade"
FROM sc049 INNER JOIN s049 ON sc049.Sno=s049.Sno
WHERE GRADE IS NOT NULL
GROUP BY s049.Sno
HAVING AVG(GRADE) > ALL(
   SELECT AVG(GRADE)
   FROM sc049 INNER JOIN s049 ON sc049.Sno=s049.Sno
   WHERE Sname='王涛' AND GRADE IS NOT NULL)
ORDER BY SNO DESC;
```

运行时间：0.271s

上述 SQL 的执行过程如下：

```
Sort  (cost=9037.23..9049.75 rows=5011 width=89)
```

```
Sort Key: public.s049.sno DESC
->  HashAggregate  (cost=8597.74..8729.28 rows=5011 width=89)
      Group By Key: public.s049.sno
      Filter: (SubPlan 1)
      ->  Hash Join  (cost=169.75..6396.62 rows=199826 width=25)
            Hash Cond: (public.sc049.sno = public.s049.sno)
            ->  Seq Scan on sc049  (cost=0.00..3479.26 rows=199826 width=16)
                  Filter: (grade IS NOT NULL)
            ->  Hash  (cost=107.11..107.11 rows=5011 width=20)
                  ->  Seq Scan on s049  (cost=0.00..107.11 rows=5011 width=20)
      SubPlan 1
        ->  Materialize  (cost=702.43..702.44 rows=1 width=37)
              ->  Aggregate  (cost=702.43..702.44 rows=1 width=37)
                    ->  Nested Loop  (cost=9.86..701.91 rows=208 width=5)
                          ->  Seq Scan on s049  (cost=0.00..119.64 rows=1 width=11)
                                Filter: ((sname)::text = '王涛'::text)
                          ->  Bitmap Heap Scan on sc049  (cost=9.86..580.19 rows=208 width=16)
                                Recheck Cond: (sno = public.s049.sno)
                                Filter: (grade IS NOT NULL)
                                ->  Bitmap Index Scan on sc049_pkey  (cost=0.00..9.81 rows=208 width=0)
                                      Index Cond: (sno = public.s049.sno)
```



运行过程分析：

实现 2 相比于实现 1 使用了内连接。通过上面的查询过程可以发现两者的查询的具体实现其实是相同的，这是因为 DBMS 对查询语句做了优化，但是就运行时间而言，实现 2 的运行时间相比于实现 1 更加短，这可能是由于 DBMS 对实现 1 的优化过程更长，消耗量更多的时间。

（7）查询选修计算机专业全部课程（课程编号为“CS-××”）的学生姓名及已获得学分。

实现 1：

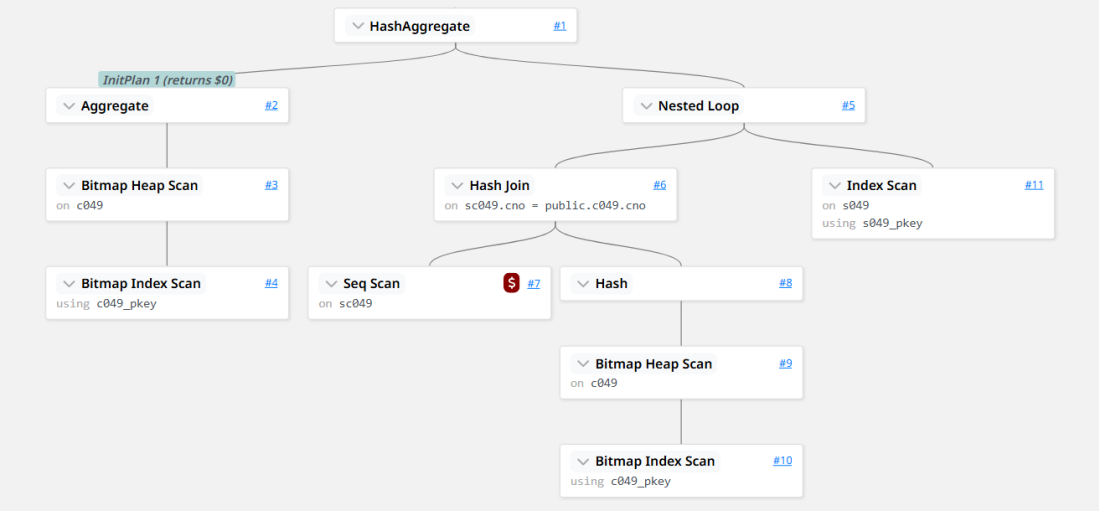
```
SELECT Sname,SUM(CREDIT) as "total credits"
FROM s049,sc049,c049
```

```
WHERE c049.Cno LIKE 'CS-%' AND
c049.Cno = sc049.Cno AND s049.Sno = sc049.Sno AND Grade BETWEEN 60 AND 100
GROUP BY s049.Sno
HAVING COUNT(*) = (
   SELECT COUNT(Cno)
   FROM c049
   WHERE Cno LIKE 'CS-%'
);
```

运行时间：0.112s

上述 SQL 的执行过程如下：

```
HashAggregate  (cost=5173.64..5178.65 rows=401 width=44)
  Group By Key: s049.sno
  Filter: (count(*) = $0)
  InitPlan 1 (returns $0)
    ->  Aggregate  (cost=17.54..17.55 rows=1 width=19)
          ->  Bitmap Heap Scan on c049  (cost=4.77..17.39 rows=61 width=11)
                Filter: (cno ~~ 'CS-%'::text)
                ->  Bitmap Index Scan on c049_pkey  (cost=0.00..4.75 rows=50 width=0)
                      Index Cond: ((cno >= 'CS-'::bpchar) AND (cno < 'CS.'::bpchar))
  ->  Nested Loop  (cost=18.15..5153.08 rows=401 width=28)
        ->  Hash Join  (cost=18.15..5020.09 rows=401 width=19)
              Hash Cond: (sc049.cno = public.c049.cno)
              ->  Seq Scan on sc049  (cost=0.00..4978.95 rows=6586 width=22)
                    Filter: ((grade >= 60::numeric) AND (grade <= 100::numeric) AND (cno ~~ 'CS-%'::text))
              ->  Hash  (cost=17.39..17.39 rows=61 width=19)
                    ->  Bitmap Heap Scan on c049  (cost=4.77..17.39 rows=61 width=19)
                          Filter: (cno ~~ 'CS-%'::text)
                          ->  Bitmap Index Scan on c049_pkey  (cost=0.00..4.75 rows=50 width=0)
                                Index Cond: ((cno >= 'CS-'::bpchar) AND (cno < 'CS.'::bpchar))
        ->  Index Scan using s049_pkey on s049  (cost=0.00..0.32 rows=1 width=20)
              Index Cond: (sno = sc049.sno)
```



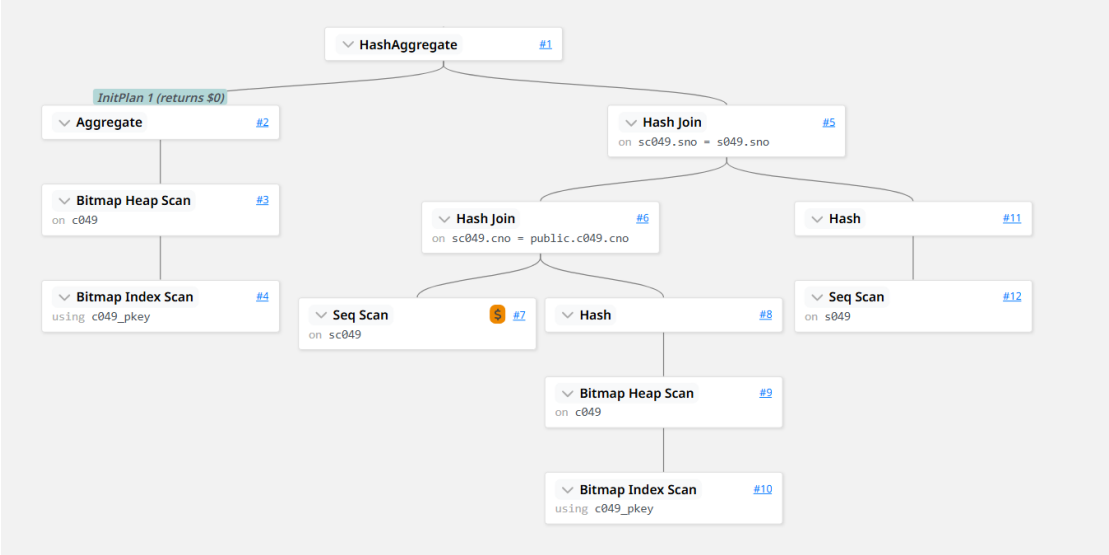
实现 2：

```
SELECT SNAME,SUM(CASE
   WHEN GRADE BETWEEN 60 AND 100 THEN
   CREDIT
   ELSE
   0
END ) "total credits"
FROM s049,sc049,c049
WHERE c049.CNO LIKE 'CS-%' AND c049.cno = sc049.cno AND s049.sno = sc049.sno
GROUP BY s049.SNO
HAVING COUNT(*) = (
   SELECT COUNT(CNO)
   FROM c049
   WHERE CNO LIKE 'CS-%'
);
```

运行时间：0.057s

上述 SQL 的执行过程如下：

```
HashAggregate  (cost=4566.96..4629.60 rows=5011 width=49)
  Group By Key: s049.sno
  Filter: (count(*) = $0)
  InitPlan 1 (returns $0)
    ->  Aggregate  (cost=17.54..17.55 rows=1 width=19)
          ->  Bitmap Heap Scan on c049  (cost=4.77..17.39 rows=61 width=11)
                Filter: (cno ~~ 'CS-%'::text)
                ->  Bitmap Index Scan on c049_pkey  (cost=0.00..4.75 rows=50 width=0)
                      Index Cond: ((cno >= 'CS-'::bpchar) AND (cno < 'CS.'::bpchar))
  ->  Hash Join  (cost=187.90..4427.49 rows=9754 width=33)
        Hash Cond: (sc049.sno = s049.sno)
        ->  Hash Join  (cost=18.15..4123.62 rows=9754 width=24)
              Hash Cond: (sc049.cno = public.c049.cno)
              ->  Seq Scan on sc049  (cost=0.00..3979.83 rows=9754 width=27)
                    Filter: (cno ~~ 'CS-%'::text)
              ->  Hash  (cost=17.39..17.39 rows=61 width=19)
                    ->  Bitmap Heap Scan on c049  (cost=4.77..17.39 rows=61 width=19)
                          Filter: (cno ~~ 'CS-%'::text)
                          ->  Bitmap Index Scan on c049_pkey  (cost=0.00..4.75 rows=50 width=0)
                                Index Cond: ((cno >= 'CS-'::bpchar) AND (cno < 'CS.'::bpchar))
        ->  Hash  (cost=107.11..107.11 rows=5011 width=20)
              ->  Seq Scan on s049  (cost=0.00..107.11 rows=5011 width=20)
```



运行过程分析：

实现 2 相比实现 1 使用了 CASE 语句进行了学分的筛选。对于运行过程，实现 1 使用了 HashAggregate 、Nested Loop 和 Hash Join 的执行计划，使用 Filter 进行 60 到 100 分的成绩的选择，而实现 2 同样使用了 HashAggregate 和 Hash Join 的执行计划，但是没有使用 nested loop，使得实现 2 的查询性能会比较好。

（8）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。

实现 1：

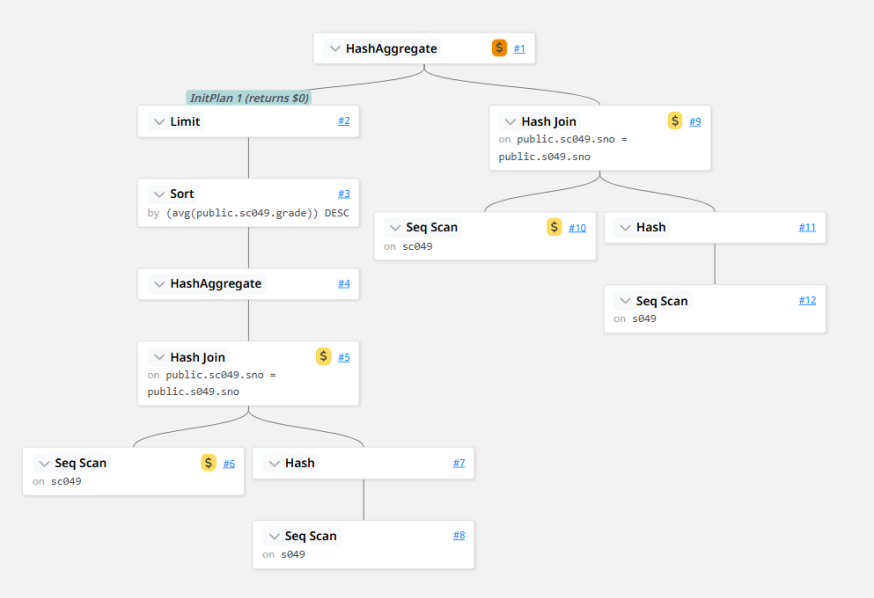
```
SELECT s049.Sno,s049.Sname
FROM s049,sc049
WHERE s049.Sno=sc049.Sno AND GRADE IS NOT NULL
GROUP BY s049.Sno
HAVING AVG(GRADE) = (
   SELECT AVG(GRADE)
   FROM s049,sc049
   WHERE s049.sno = sc049.sno AND GRADE IS NOT NULL
   GROUP BY s049.SNO
   HAVING COUNT(*) >= 3
   ORDER BY AVG(GRADE) DESC
   LIMIT 1
);
```

运行时间：0.53s

上述 SQL 的执行过程如下：

```
HashAggregate  (cost=15393.28..15468.44 rows=5011 width=57)
  Group By Key: public.s049.sno
  Filter: (avg(public.sc049.grade) = $0)
  InitPlan 1 (returns $0)
    ->  Limit  (cost=7996.53..7996.53 rows=1 width=56)
          ->  Sort  (cost=7996.53..8009.06 rows=5011 width=56)
                Sort Key: (avg(public.sc049.grade)) DESC
```

```
                ->  HashAggregate  (cost=7896.31..7971.48 rows=5011 width=56)
                      Group By Key: public.s049.sno
                      Filter: (count(*) >= 3)
                      ->  Hash Join  (cost=169.75..6397.62 rows=199826 width=16)
                            Hash Cond: (public.sc049.sno = public.s049.sno)
                            ->  Seq Scan on sc049  (cost=0.00..3480.26 rows=199826 width=16)
                                  Filter: (grade IS NOT NULL)
                            ->  Hash  (cost=107.11..107.11 rows=5011 width=11)
                                  ->  Seq Scan on s049  (cost=0.00..107.11 rows=5011 width=11)
  ->  Hash Join  (cost=169.75..6397.62 rows=199826 width=25)
        Hash Cond: (public.sc049.sno = public.s049.sno)
        ->  Seq Scan on sc049  (cost=0.00..3480.26 rows=199826 width=16)
              Filter: (grade IS NOT NULL)
        ->  Hash  (cost=107.11..107.11 rows=5011 width=20)
              ->  Seq Scan on s049  (cost=0.00..107.11 rows=5011 width=20)
```



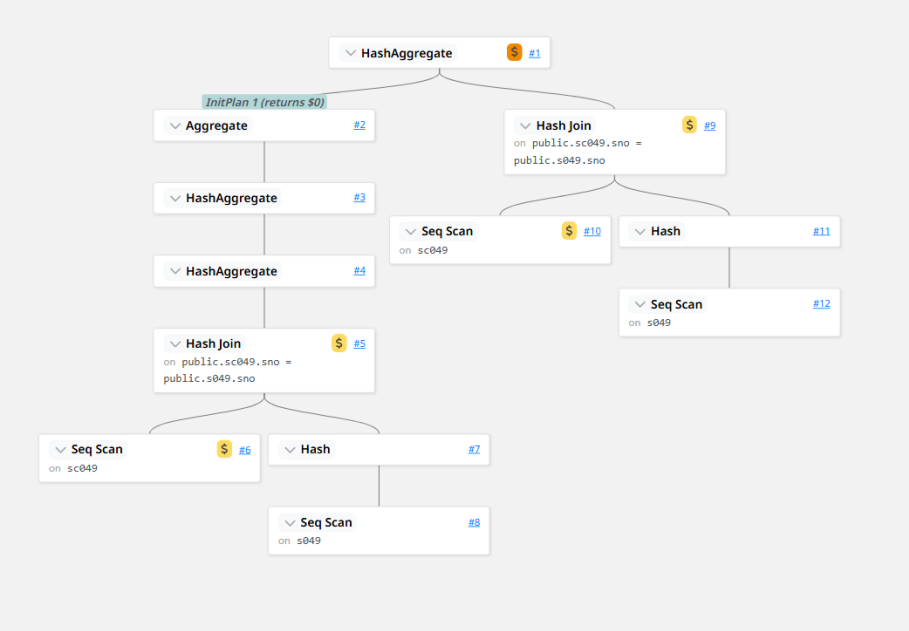
实现 2：

```
SELECT s049.Sno,s049.Sname
FROM s049 INNER JOIN sc049 ON s049.Sno=sc049.Sno
WHERE GRADE IS NOT NULL
GROUP BY s049.Sno
HAVING AVG(GRADE) = (
 SELECT MAX(avg)
 FROM (
  SELECT DISTINCT AVG(GRADE) avg
  FROM s049 INNER JOIN sc049 ON s049.sno = sc049.sno
  WHERE GRADE IS NOT NULL
  GROUP BY s049.SNO
  HAVING COUNT(*) >= 3)
);
```

运行时间：0.503s

上述 SQL 的执行过程如下：

```
HashAggregate  (cost=15493.51..15568.67 rows=5011 width=57)
  Group By Key: public.s049.sno
  Filter: (avg(public.sc049.grade) = $0)
  InitPlan 1 (returns $0)
    ->  Aggregate  (cost=8096.75..8096.76 rows=1 width=64)
          ->  HashAggregate  (cost=7984.00..8034.11 rows=5011 width=56)
                Group By Key: avg(public.sc049.grade)
                ->  HashAggregate  (cost=7896.31..7971.48 rows=5011 width=56)
                      Group By Key: public.s049.sno
                      Filter: (count(*) >= 3)
                      ->  Hash Join  (cost=169.75..6397.62 rows=199826 width=16)
                            Hash Cond: (public.sc049.sno = public.s049.sno)
                            ->  Seq Scan on sc049  (cost=0.00..3480.26 rows=199826 width=16)
                                  Filter: (grade IS NOT NULL)
                            ->  Hash  (cost=107.11..107.11 rows=5011 width=11)
                                  ->  Seq Scan on s049  (cost=0.00..107.11 rows=5011 width=11)
    ->  Hash Join  (cost=169.75..6397.62 rows=199826 width=25)
          Hash Cond: (public.sc049.sno = public.s049.sno)
          ->  Seq Scan on sc049  (cost=0.00..3480.26 rows=199826 width=16)
                Filter: (grade IS NOT NULL)
          ->  Hash  (cost=107.11..107.11 rows=5011 width=20)
                ->  Seq Scan on s049  (cost=0.00..107.11 rows=5011 width=20)
```



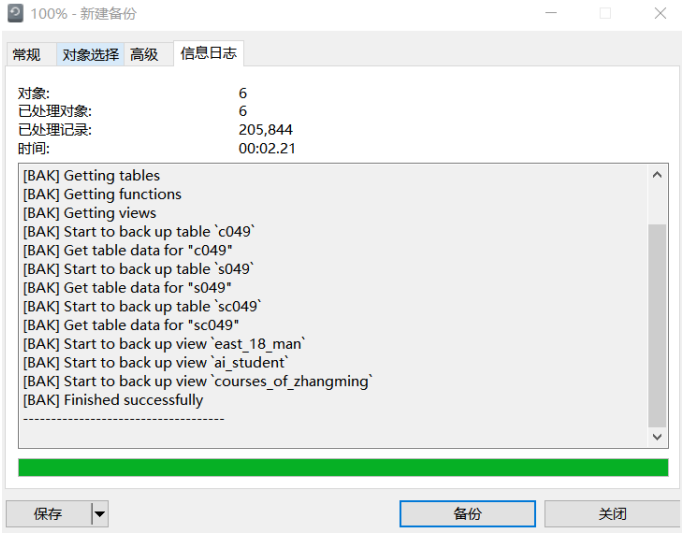
运行过程分析：

实现 2 没有使用 LIMIT 语句，并且使用了内连接进行表的连接操作，相比与实现 1，实现 2 不要对表进行排序，只需要选出表中的最大项，这使得实现 2 的计算比实现 1 简单，实现 2 的查询效率更高。

六、数据备份

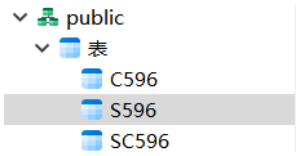
1. 数据备份

使用 Navicat 的备份功能进行数据的备份操作，得到备份文件。



2. 数据还原

对XXX同学（学号 XXXXXXXXXX）的数据库进行数据还原，可以得到其数据库中的三个表，S596，C596 和 SC596，如下所示。



3. 数据质量分析

可以看到其数据表 S、C 和 SC 的字段设计如下所示。

名	类型	长度	小数点	不是 null	键	注释
S#	varchar	10	0	☑	1	
SNAME	nvarchar2	0	0	☑		
SEX	nvarchar2	0	0	☐		
BDATE	timestamp	0	0	☐		
HEIGHT	numeric	3	2	☐		
DORM	nvarchar2	0	0	☐		

名	类型	长度	小数点	不是 null	键	注释
C#	varchar	30	0	☑	1	
CNAME	nvarchar2	0	0	☑		
PERIOD	numeric	4	1	☐		
CREDIT	numeric	3	1	☐		
TEACHER	nvarchar2	0	0	☐		

名	类型	长度	小数点	不是 null	键	注释
S#	varchar	10	0	☑	1	
C#	varchar	30	0	☑	2	
GRADE	numeric	5	1	☐		

其中可以发现，其 S 和 C 表中的 Sno 和 Cno 为 Varchar 类型，考虑到一般学号和课程号为定长字段，其可以改为 Char 的定长类型。C 表中的学时和学分字段考虑了数据的范围，学时和学分一般不会很大，使用了 numeric 类型比较合理。

可以看到其表中的数据如下。

S#	SNAME	SEX	BDATE	HEIGHT	DORM
0199275944	明欢	女	2005-07-16 00:00:00	1.84	西22舍321
0517319758	冷甯	女	2002-06-05 00:00:00	1.81	西12舍109
0145093858	僧诗	女	2004-08-18 00:00:00	1.88	西8舍315
0542162941	晋都	男	2002-08-17 00:00:00	1.76	东11舍619
0074704842	牟真	女	2005-10-19 00:00:00	1.84	西13舍206
0038754391	糜苗	男	2005-08-19 00:00:00	1.56	东4舍423
0994715272	向博	男	2004-10-13 00:00:00	1.50	东2舍517
01032010	王涛	男	2003-04-05 00:00:00	1.72	东6舍221
01032023	孙文	男	2004-06-10 00:00:00	1.80	东6舍221
01032001	张晓梅	女	2004-11-17 00:00:00	1.58	东1舍312
01032005	刘静	女	2003-01-10 00:00:00	1.63	东1舍312
01032112	董蔚	男	2003-02-20 00:00:00	1.71	东6舍221
03031011	王倩	女	2004-12-20 00:00:00	1.66	东2舍104
03031014	赵思扬	男	2002-06-06 00:00:00	1.85	东18舍421
03031051	周剑	男	2002-05-08 00:00:00	1.68	东18舍422
03031009	田菲	女	2003-08-11 00:00:00	1.60	东2舍104
03031033	蔡明明	男	2003-03-12 00:00:00	1.75	东18舍423
03031056	曹子衿	女	2004-12-15 00:00:00	1.65	东2舍305
0361921348	聂颖聪	女	2004-02-08 00:00:00	1.67	西24舍521
0157638758	褚铃初	女	2005-02-28 00:00:00	1.84	西1舍312
0889649617	解蕾闵	男	2003-10-20 00:00:00	1.66	东14舍105

C#	CNAME	PERIOD	CREDIT	TEACHER
CS-01	数据结构	60.0	3.0	张军
CS-02	计算机组成	80.0	4.0	王亚伟
CS-04	人工智能	40.0	2.0	李蕾
CS-05	深度学习	40.0	2.0	崔昀
EE-02	数字逻辑电	100.0	5.0	胡海东
EE-03	光电子学与	40.0	2.0	石韬
EE-01	信号与系统	64.0	4.0	张明
CS-03	离散数学	64.0	4.0	陈建明
GNED110	大学生心理	32.0	2.0	刘可尊,姚斌
SOCP1001	课外实践8学	0.0	8.0	任君,温景涛,
GNED106	大学生创业	32.0	2.0	王诗航,郑旭
GNED106	大学生职业	32.0	2.0	朱宏伟,郑旭
GNED115	在线开放课	0.0	0.0	None
BSIS40012	毕业设计（	128.0	8.0	邵一圃
INFT5001	计算机基础	16.0	1.0	贾应智
INTE4001	国际贸易实	64.0	4.0	杨杨
JZSJ90012	中国传统文	16.0	1.0	孙立滨
LITE10102	中国概况	32.0	2.0	赵炜
LITE10112	当代影视鉴	32.0	2.0	张知凉
LITE10132	简明中国历	32.0	2.0	路荣

S#	C#	GRADE
0754615260	AUTO500427	47.5
0041066354	BASM312040	51.0
0102226470	ARTS517510	71.0
0999462246	GNED100229	63.5
0101352962	CHEM250109	80.0
0256561713	COMP543305	69.0
0610306025	STAT310708	71.5
0299706422	LAWS404524	74.0
0390506875	CHEM301129	74.0
0286414637	GNED114310	89.0
0498843342	ENGL203112	66.5
0160382329	INFT320340	72.5
0137007533	GERM440612	85.5
0057501007	BASM401148	77.0
0190880884	MAGT501218	77.5
0101293087	FINA543619	62.5
0855747603	POLI300514	66.0

其中 C 表的数据是学校的课程，比较合理，但是 Cno 的编号也是学校课程编号，不是统一的 XX-YY 的格式，最好是统一格式。S 表中姓名从姓名大全中爬取，没有姓名对应的性别，最好姓名和性别对应，并且学生的学号长度不统一，一般学号长度应该统一。

附录

一、JAVA 代码

```
package src.gaussdb_jdbc;

import java.io.BufferedReader;
import java.io.IOException;
import java.sql.Connection;
import java.sql.DriverManager;
import java.sql.ResultSet;
import java.sql.SQLException;
import java.sql.Statement;

import java.io.FileReader;
import java.util.concurrent.TimeUnit;

public class openGaussMYDB {

    static final String JDBC_DRIVER = "org.postgresql.Driver";
    static final String DB_URL = "jdbc:postgresql://192.168.150.134:7654/mydb";
```

```
// 数据库的用户名与密码，需要根据自己的设置
static final String USER = "testuser";
static final String PASS = "A1a2a3a4a5$";


public static void ExecSQL(Connection conn, String filename) throws IOException,
        InterruptedException {
    FileReader fr=new FileReader(filename);
    BufferedReader br=new BufferedReader(fr);
    String line;
    String buffer="";
    while ((line=br.readLine())!=null) {
        if (!line.contains("--")){
            buffer = buffer + line + " ";
        }
        if (line.contains(";")){
            // 执行 sql 语句
            // System.out.println("Executing: " + buffer);
            if (buffer.contains("INSERT")){
                System.out.println("SQL: INSERT");
            }
            if (buffer.contains("DELETE")){
                System.out.println("SQL: DELETE");
            }
            Statement stmt = null;
            try {
                stmt = conn.createStatement();
                stmt.execute(buffer);
                stmt.close();
                TimeUnit.MICROSECONDS.sleep(1000);
            } catch (SQLException e) {
                System.out.println("Error occurs when executing " + buffer);
                if (stmt != null) {
                    try {
                        stmt.close();
                    } catch (SQLException e1) {
                        e1.printStackTrace();
                    }
                }
                e.printStackTrace();
            }
            buffer = "";
        }
    }
    br.close();
    fr.close();
}

public static void testSQL(Connection conn){
    try{
        // 执行查询
        System.out.println(" 实例化 Statement 对象...");
        Statement stmt = conn.createStatement();
        String sql;
        sql = "SELECT * FROM sc049";
        ResultSet rs = stmt.executeQuery(sql);

        int count = 0;
        // 展开结果集数据库
        while(rs.next()){
            // 通过字段检索
            String Sno  = rs.getString("Sno");
            String Cno = rs.getString("Cno");
            String Grade = rs.getString("Grade");


            // 输出数据
            System.out.print("Sno: " + Sno);
            System.out.print(", Cno: " + Cno);
            System.out.print(", Grade: " + Grade);
            System.out.print("\n");
```

```
            count++;
        }
        // 完成后关闭
        rs.close();
        stmt.close();
        System.out.println("Total: " + count);

    }catch(SQLException se){
        // 处理 JDBC 错误
        se.printStackTrace();
    } catch (Exception e) {
        e.printStackTrace();
    }
}

public static void ExecSQL_Times(Connection conn, String filename, Integer lines) throws IOException,
        InterruptedException {
    // 读取 1000 行就执行一次 SQL
    FileReader fr=new FileReader(filename);
    BufferedReader br=new BufferedReader(fr);
    String line;
    String buffer="";
    int line_count = 0;
    String first_line = "";
    // 读入第一行，为 SQL 语句的第一部分
    if ((line=br.readLine())!=null) {
        first_line = line;
    }
    // 每 1000 行执行一次 SQL
    buffer = first_line;
    while ((line=br.readLine())!=null) {
        if (!line.contains("--")){
            buffer = buffer + line + " ";
            line_count++;
        }
        if (line_count == lines || line.contains(";")){
            line_count = 0;
            // 将 buffer 中的最后一个字符设置为';'
            buffer = buffer.substring(0, buffer.length() - 2) + ";";
            // 执行 sql 语句
            System.out.println("Executing: " + buffer);

            Statement stmt = null;
            try {
                stmt = conn.createStatement();
                stmt.execute(buffer);
                stmt.close();
                TimeUnit.MICROSECONDS.sleep(1000);
            } catch (SQLException e) {
                System.out.println("Error occurs when executing " + buffer);
                if (stmt != null) {
                    try {
                        stmt.close();
                    } catch (SQLException e1) {
                        e1.printStackTrace();
                    }
                }
                e.printStackTrace();
            }

            buffer = first_line;
        }


    }

    br.close();
    fr.close();
}
```

```
public static void main(String[] args) {
    Connection conn = null;
    try{
        // 注册 JDBC 驱动
        Class.forName(JDBC_DRIVER);

        // 打开链接
        System.out.println("连接数据库...");
        conn = DriverManager.getConnection(DB_URL,USER,PASS);

        // 测试 SQL
        testSQL(conn);

        // 读入 S049_1.txt 文件并执行 sql 语句
        // ExecSQL(conn, "./scripts/S049_1.txt");

        // 读入 C049_1.txt 文件并执行 sql 语句
        // ExecSQL(conn, "./scripts/C049_1.txt");

        /*
        // 并行执行 SC049_1.txt 文件中的和 delete_less_than_60.txt 文件中的 sql 语句
        Connection finalConn = conn;
        Thread t1 = new Thread(new Runnable() {
            @Override
            public void run() {
                try {
                    ExecSQL_Times(finalConn, "./scripts/SC049_1.txt", 100);
                } catch (IOException | InterruptedException e) {
                    e.printStackTrace();
                }
            }
        });

        Thread t2 = new Thread(new Runnable() {
            @Override
            public void run() {
                try {
                    TimeUnit.SECONDS.sleep(2);
                    ExecSQL(finalConn, "./scripts/delete_less_than_60.txt");
                } catch (IOException | InterruptedException e) {
                    e.printStackTrace();
                }
            }
        });

        t1.start();
        t2.start();

        t1.join();
        t2.join();
        */

        // 读入 S049_2.txt 文件并执行 sql 语句
        // ExecSQL(conn, "./scripts/S049_2.txt");

        // 读入 C049_2.txt 文件并执行 sql 语句
        // ExecSQL(conn, "./scripts/C049_2.txt");

        // 读入 SC049_2.txt 文件并执行 sql 语句
        // ExecSQL(conn, "./scripts/SC049_2.txt");


        conn.close();

    }catch(SQLException se){
        // 处理 JDBC 错误
        se.printStackTrace();
    }catch(Exception e){
        // 处理 Class.forName 错误
        e.printStackTrace();
    }
```

```
    }
}
```

二、Python 代码

爬取课程

```
import requests
from lxml import etree
from selenium import webdriver
import time
import pandas as pd

detail_schools = ['北京大学', '清华大学', '复旦大学', '上海交通大学', '中国人民大学', '武汉大学', '中山大学',
                  '南京大学', '山东大学', '华中科技大学',
                  '西安交通大学', '四川大学', '东南大学', '中南大学', '哈尔滨工业大学', '北京航空航天大学', '
同济大学',
                  '天津大学', '华南理工大学',
                  '北京师范大学', '东北大学', '大连理工大学', '华东师范大学', '吉林大学', '重庆大学', '西北工
业大学',
                  '南开大学', '北京理工大学',
                  '兰州大学', '厦门大学', '湖南大学', '南京航空航天大学', '西南交通大学', '中国农业大学',
                  '北京交通大学', '中国矿业大学', '西北农林科技大学',
                  '中国海洋大学', '中国地质大学', '华中农业大学', '华东理工大学', '东北农业大学', '南京农业大
学',
                  '中国药科大学', '南京理工大学',
                  '中国石油大学', '北京化工大学', '北京邮电大学', '北京林业大学', '北京中医药大学', '中国传媒
大学',
                  '北京工商大学', '北京体育大学', '北京外国语大学', '中国音乐学院', '中央民族大学', '中国政法
大学', '华北电力大学',
                   '华北理工大学', '南京艺术学院', '上海大学', '华东政法大学', '上海外国语大学', '上海财经大
学', '上海师范大学']
for detail_school in detail_schools:

    # time.sleep(5)
    start = time.time()
    url = 'https://www.icourse163.org/university/view/all.htm#/'
    headers = {
        'user-agent': 'Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko)
Chrome/89.0.4389.90 Safari/537.36 Edg/89.0.774.54'
    }
    r = requests.get(url).text
    tree = etree.HTML(r)
    # 获取大学链接
    href = tree.xpath('//a[@class="u-usity f-fl"]/@href')
    # 获取大学名称
    name = tree.xpath('//a[@class="u-usity f-fl"]/img/@alt')

    driver = webdriver.Edge()
    driver.maximize_window()


    def get_school_all_course():
        while (True):
            html = driver.page_source
            tree = etree.HTML(html)

            # 获取课程的链接
            hrefList = tree.xpath('//div[@class="um-spoc-course-list_wrap"]/div/a/@href')
            # 获取课程名称
            nameList = tree.xpath(
                '//div[@class="um-spoc-course-list_wrap"]/div/a/div[@class="u-courseCardWithTime-
teacher"]/span/text()')
            # print(len(nameList))
            for x, y in zip(hrefList, nameList):
                x = 'https:' + x
                course_name.append(y)
```

```
                course_url.append(x)

            flag = (int(len(nameList)) != 20)

            if flag:
                break
            else:
                try:
                    # 点击下一页
                    a = driver.find_element_by_link_text('下一页')
                    a.click()
                    time.sleep(5)
                except:
                    return None
                get_school_all_course()


    def save(data, school):
        df = pd.DataFrame(data)
        df.to_csv('./{}.csv'.format(school), mode='a', encoding='ANSI', index=False, header=False)


    # 课程的链接
    course_url = []
    # 课程的名称
    course_name = []
    # time.sleep(5)
    for x, y in zip(href, name):
        school_link = 'https://www.icourse163.org' + x
        dict = {
            "大学名称": y,
            "大学课程链接": school_link
        }
        if dict["大学名称"] == detail_school:
            new_url = dict["大学课程链接"]
            driver.get(new_url)
            get_school_all_course()

            for i, j in zip(course_name, course_url):
                new_dict = {
                    "课程名称": [i],
                    "课程链接": [j]
                }
                save(new_dict, 'course')
            driver.close()
            end = time.time()
            print(detail_school, '慕课的所有课程保存完成')
            deltatime = end - start
            print("项目所用时间为:", deltatime)
```

爬取人名

```
# 从 https://www.resgain.net/name_list.rhtml?fid=i i 是一个整数，爬取 7000 个姓名

import requests
from bs4 import BeautifulSoup
import re
import time
import random
import os


def get_name():
    total_name = 7000
    url = 'https://www.resgain.net/name_list.rhtml?fid='
    headers = {
        'User-Agent': 'Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko)
Chrome/58.0.3029.110 Safari/537.3'}
    for i in range(1, 500):
        try:
            response = requests.get(url + str(i), headers=headers)
            response.encoding = 'utf-8'
```

```
            soup = BeautifulSoup(response.text, 'html.parser')
            name_list = soup.find_all('div', class_='cname')[0:int(total_name//500+1)]
            for name in name_list:
                name = name.get_text()
                name = re.sub(r'\s+', '', name)
                with open('name.txt', 'a', encoding='utf-8') as f:
                    f.write(name + '\n')
            print('已爬取第{}页'.format(i))
            time.sleep(random.randint(1, 3))
        except Exception as e:
            print(e)
            continue
    print('爬取完成')


if __name__ == '__main__':
    get_name()
```

数据生成代码

生成 S049 内容

```
# S049 （S#, SNAME, SEX, BDATE, HEIGHT, DORM）

import random

# 生成学号，从 03032000 到 04000000，随机选取 5000 个学号，不重复
def generate_S():
    S_list = []
    for i in range(3002000, 4000000):
        S_list.append(i)
    random.shuffle(S_list)
    S_list = S_list[0:5000]
    # 生成学号，转为字符串，每个学号占 8 位，不足 8 位前面补 0
    S_list = [str(S) for S in S_list]
    S_list = ['0'*(8-len(S)) + S for S in S_list]

    # 存储学号到 data/S.txt
    with open('data/S.txt', 'w', encoding='utf-8') as f:
        for S in S_list:
            f.write(S + '\n')

    return S_list


def generate_SNAME():
    with open('data/shuffle_name.txt', 'r', encoding='utf-8') as f:
        name_list = f.readlines()[0:5000]
    return name_list


# 生成 5000 个男.女
def generate_SEX():
    SEX_list = ['男']*2500 + ['女']*2500
    random.shuffle(SEX_list)
    return SEX_list


# 生成 5000 个生日，从 2001 到 2005
def generate_BDATE():
    BDATE_list = []
    year = ['2001', '2002', '2003', '2004', '2005']
    month = ['1', '2', '3', '4', '5', '6', '7', '8', '9', '10', '11', '12']
    day = ['1', '2', '3', '4', '5', '6', '7', '8', '9', '10',
           '11', '12', '13', '14', '15', '16', '17', '18', '19', '20',
           '21', '22', '23', '24', '25', '26', '27', '28', '29', '30', '31']
    days_in_month = [31, 28, 31, 30, 31, 30, 31, 31, 30, 31, 30, 31]
    for i in range(5000):
        y = random.choice(year)
        m = random.choice(month)
        if m == '2':
```

```
            if int(y) % 4 == 0 and int(y) % 100 != 0 or int(y) % 400 == 0:
                d = random.choice(day[0:29])
            else:
                d = random.choice(day[0:28])
        else:
            d = random.choice(day[0:days_in_month[int(m)-1]])
        BDATE_list.append(y + '-' + m + '-' + d)

    return BDATE_list


# 生成5000个身高，从1.50到1.90，保留两位小数
def generate_HEIGHT():
    HEIGHT_list = []
    for i in range(5000):
        HEIGHT_list.append(round(random.uniform(1.50, 1.90), 2))
    return HEIGHT_list


# 生成5000个宿舍号，东/南/西/北 1-20舍，一舍8层，一层20号
def generate_DORM():
    DORM_list = []
    number = ['01', '02', '03', '04', '05', '06', '07', '08', '09', '10',
              '11', '12', '13', '14', '15', '16', '17', '18', '19', '20']
    for i in range(5000):
        dorm = random.choice(['东', '南', '西', '北']) + str(random.randint(1, 20)) + '舍' +
str(random.randint(1, 8)) + random.choice(number)
        DORM_list.append(dorm)
    return DORM_list


if __name__ == '__main__':
    S = generate_S()
    SNAME = generate_SNAME()
    SEX = generate_SEX()
    BDATE = generate_BDATE()
    HEIGHT = generate_HEIGHT()
    DORM = generate_DORM()

    SQL = 'INSERT INTO S049 VALUES'
    with open('S049.txt', 'w', encoding='utf-8') as f:
        f.write(SQL + '\n')
        for i in range(5000):
            SQL = '(\'' + S[i] + '\', \'' + SNAME[i].strip() + '\', \'' + SEX[i] + '\', \'' + BDATE[i] + '\',
' + str(HEIGHT[i]) + ', \'' + DORM[i] + '\')'
            if i != 4999:
                SQL += ','
            else:
                SQL += ';'
            f.write(SQL + '\n')
```

生成C049内容

```
# C049 （C#, CNAME, PERIOD, CREDIT, TEACHER）

import random
def generate_C():
    deps = ['CS', 'EE', 'SC', 'AI', 'SE', 'DS', 'IE', 'ME', 'CE', 'AE',
            'MT', 'PE', 'CH', 'BI', 'MA', 'PH', 'EN', 'HI', 'MU', 'AR']
    # 不能生成重复的课程号
    exist_C = ['CS-01', 'CS-02', 'CS-03', 'CS-04', 'CS-05', 'EE-01', 'EE-02', 'EE-03']
    C_list = []
    for i in range(1000):
        dep = random.choice(deps)
        num = str(random.randint(1, 99)).zfill(2)
        C = dep + '-' + num
        while C in exist_C:
            dep = random.choice(deps)
            num = str(random.randint(1, 99)).zfill(2)
            C = dep + '-' + num
```

```
        exist_C.append(C)
        C_list.append(C)

    # 存储课程号到 data 文件夹
    with open('data/C.txt', 'w', encoding='utf-8') as f:
        for i in range(1000):
            f.write(C_list[i] + '\n')

    return C_list


def generate_CNAME():
    with open('data/clear_course.txt', 'r', encoding='utf-8') as f:
        cname_list = f.readlines()[0:1000]
    return cname_list


def generate_PERIOD():
    PERIOD_list = []
    period = [24, 32, 48, 64, 40, 16, 56, 72, 80, 88, 60, 80, 100]
    for i in range(1000):
        PERIOD_list.append(random.choice(period))
    return PERIOD_list


def generate_CREDIT():
    CREDIT_list = []
    credit = [0.5, 1, 1.5, 2, 2.5, 3, 4, 5]
    for i in range(1000):
        CREDIT_list.append(random.choice(credit))
    return CREDIT_list


def generate_TEACHER():
    with open('data/shuffle_name.txt', 'r', encoding='utf-8') as f:
        # 选后 1000 个
        name_list = f.readlines()
        TEACHER_list = name_list[len(name_list)-1000:]

    return TEACHER_list


if __name__ == '__main__':
    C = generate_C()
    CNAME = generate_CNAME()
    PERIOD = generate_PERIOD()
    CREDIT = generate_CREDIT()
    TEACHER = generate_TEACHER()

    with open('C049.txt', 'w', encoding='utf-8') as f:
        SQL = 'INSERT INTO C049 VALUES\n'
        for i in range(1000):
            if i == 999:
                SQL     +=     f"('{C[i]}',     '{CNAME[i].strip()}',     {PERIOD[i]},     {CREDIT[i]},
'{TEACHER[i].strip()}');"
            else:
                SQL     +=     f"('{C[i]}',     '{CNAME[i].strip()}',     {PERIOD[i]},     {CREDIT[i]},
'{TEACHER[i].strip()}'),\n"
        f.write(SQL)
```

生成 SC049 内容

```
# SC049 (S, C, GRADE), 生成 200000 条记录

import random

def read_S():
```

```
    with open('S049.txt', 'r', encoding='utf-8') as f:
        # 读取 S049.txt 文件，返回一个列表
        S_list = f.readlines()
    S_list = S_list[1:]
    S_list = [S[2:10] for S in S_list]
    return S_list

def read_C():
    with open('C049.txt', 'r', encoding='utf-8') as f:
        C_list = f.readlines()
    C_list = C_list[1:]
    C_list = [C[2:7] for C in C_list]
    return C_list

def generate_SC():
    S_list = read_S()
    C_list = read_C()

    # 前 5000 条记录
    S_list_1 = S_list[:1000]
    C_list_1 = C_list[:100]

    SC_list_1 = []
    S_and_C_1 = []
    for S in S_list_1:
        for C in C_list_1:
            S_and_C_1.append((S.strip(), C.strip()))

    S_and_C_1 = random.sample(S_and_C_1, 20000)
    print('S_and_C_1 generated successfully!')

    for i in range(20000):
        grade = random.randint(40, 100)
        if random.randint(0, 1) == 1 and grade < 100:
            grade += 0.5
        SC_list_1.append((S_and_C_1[i][0], S_and_C_1[i][1], grade))

    print('SC_list_1 generated successfully!')

    # 从中选出 200 条分数小于 60 的记录
    less_than_60 = [('01032023', 'CS-01', 55.0), ('03031051', 'EE-02', 58.0)]
    for record in SC_list_1:
        if record[2] < 60 and random.randint(0, 1) == 1 and record not in less_than_60:
            less_than_60.append(record)
        if len(less_than_60) == 200:
            break

    print('less_than_60 generated successfully!')

    SC_list_2 = []
    S_and_C_2 = []
    flag = 0
    for S in S_list:
        for C in C_list:
            if (S.strip(), C.strip()) in S_and_C_1:
                continue
            S_and_C_2.append((S.strip(), C.strip()))
            # print(len(S_and_C_2))
            if len(S_and_C_2) == 180000:
                flag = 1
                break
        if flag == 1:
            break

    print('S_and_C_2 generated successfully!')

    for i in range(180000):
        grade = random.randint(40, 100)
        if random.randint(0, 1) == 1 and grade < 100:
            grade += 0.5
        SC_list_2.append((S_and_C_2[i][0], S_and_C_2[i][1], grade))
```

```
    print('SC_list_2 generated successfully!')


    return SC_list_1, SC_list_2, less_than_60


if __name__ == '__main__':
    SC1, SC2, less_than_60 = generate_SC()
    print('SC and less_than_60 generated successfully!')

    with open('SC049_1.txt', 'w', encoding='utf-8') as f:
        SQL = 'INSERT INTO SC049 VALUES\n'
        for i in range(len(SC1)):
            if i == len(SC1) - 1:
                SQL += f"('{SC1[i][0]}', '{SC1[i][1]}', {SC1[i][2]});"
            else:
                SQL += f"('{SC1[i][0]}', '{SC1[i][1]}', {SC1[i][2]}),\n"
        f.write(SQL)

    with open('SC049_2.txt', 'w', encoding='utf-8') as f:
        SQL = 'INSERT INTO SC049 VALUES\n'
        for i in range(len(SC2)):
            if i == len(SC2) - 1:
                SQL += f"('{SC2[i][0]}', '{SC2[i][1]}', {SC2[i][2]});"
            else:
                SQL += f"('{SC2[i][0]}', '{SC2[i][1]}', {SC2[i][2]}),\n"
        f.write(SQL)

    with open('delete_less_than_60.txt', 'w', encoding='utf-8') as f:
        # SQL 删除 200 条分数小于 60 的记录
        SQL = 'DELETE FROM SC049 WHERE (Sno, Cno, GRADE) IN (\n'
        for i in range(200):
            if i == 199:
                SQL += f"('{less_than_60[i][0]}', '{less_than_60[i][1]}', {less_than_60[i][2]}) );"
            else:
                SQL += f"('{less_than_60[i][0]}', '{less_than_60[i][1]}', {less_than_60[i][2]}),\n"
        f.write(SQL)
```