2021 Fall CS307 Project 1

name：刘乐奇

SID：12011327

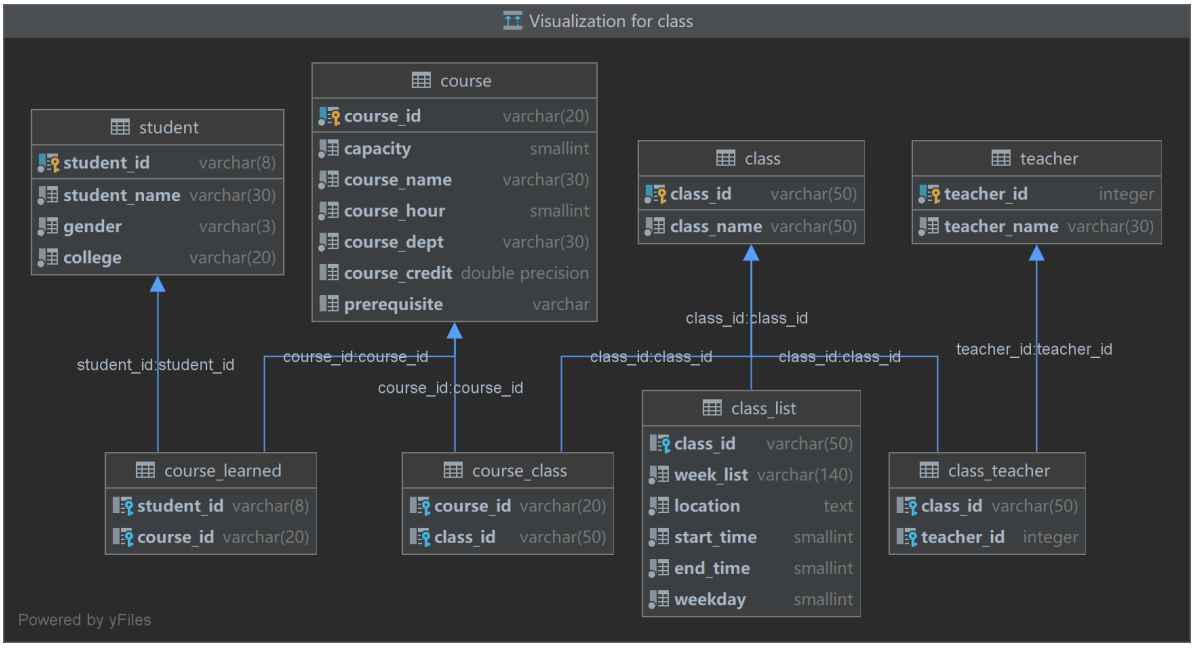
1. Project title:

Educational administration system （教务系统）

2. Database structure design

Task1: Database design

1.1 The structure of the whole database



1.2 Tables

main tables

course

columns' name	data type	explanation
course_id	varchar(20)	primary key
capacity	smallint	not null, >0
course_name	varchar(30)	not null
course_hour	smallint	not null, >=0
course_dept	varchar(30)	not null
course_credit	double precision	between 0 and 100
prerequisite	varchar	

class

columns' name	data type	explanation
class_id	varchar(50)	primary key
class_name	varchar(50)	not null

class_list

columns' name	data type	explanation
class_id	varchar(50)	foreign key, refers to *class(class_id)*
week_list	varchar(140)	not null
location	text	not null
start_time	smallint	not null, between 1 and 11
end_time	smallint	not null, between *start_time* amd 12
weekday	smallint	not null, between 1 and 8

teacher

columns' name	data type	explanation
teacher_id	integer (serial)	primary key
teacher_name	varchar(30)	not null

student

columns' name	data type	explanation
student_id	varchar(8)	primary key
student_name	varchar(30)	not null
gender	varchar(3)	not null, 'M' or 'F'
college	varchar(20)	not null

link tables

class_teacher

columns' name	data type	explanation
class_id	varchar(50)	foreign key, refers to *class(class_id)*
teacher_id	integer	foreign key, refers to *teacher(teacher_id)*

course_class

columns' name	data type	explanation
course_id	varchar(20)	foreign key, refers to *course(course_id)*
class_id	varchar(50)	foreign key, refers to *class(class_id)*

course_learned

columns' name	data type	explanation
student_id	varchar(8)	foreign key, refers to *student(student_id)*
course_id	varchar(20)	foreign key, refers to *course(course_id)*

1.3 Code of SQL

```sql
-- main table
create table course
(
    course_id    varchar(20) primary key,
    capacity     smallint    not null check ( capacity > 0 ),
    course_name         varchar(30) not null,
    course_hour  smallint    not null check ( course_hour >= 0 ),
    course_dept  varchar(30) not null,
    course_credit       float check (course_credit between 0 and 100),
    prerequisite varchar
);

create table class
(
    class_id   varchar(50) primary key,
    class_name varchar(50) not null
);
```

```sql
create table class_list
(
    class_id   varchar(50) references class (class_id),
    week_list  varchar(140) not null,
    location   text         not null,
    start_time smallint     not null check ( start_time between 1 and 11),
    end_time   smallint     not null check ( end_time between start_time and
12),
    weekday    smallint     not null check ( weekday between 1 and 8)
);

create table teacher
(
    teacher_id serial primary key,
    teacher_name       varchar(30) not null
);

create table student
(
    student_id     varchar(8) primary key,
    student_name           varchar(30) not null,
    gender         varchar(3)  not null check ( upper(gender) = 'M' or
upper(gender) = 'F' ),
    college        varchar(20) not null
);

-- link table
create table class_teacher
(
    class_id  varchar(50) references class (class_id),
    teacher_id int references teacher (teacher_id)
);

create table course_class
(
    course_id varchar(20) references course (course_id),
    class_id  varchar(50) references class (class_id)
);

create table course_learned
(
    student_id varchar(8) references student (student_id),
    course_id  varchar(20) references course (course_id)
);
```

3. Import data

Here I used `JDBC` to link to database and insert data.

`Connector.java` is used to connect and disconnect database and load data. `JWXTParser.java` is used to rearrange the data from `course_info.json` and `select_course.csv` so that making it convenient to inserted into database.

Codes below are not the whole program. I just intercepted some main codes to explain, since some codes in my program are redundant and repeated except for some difference in minor.

3.1 Open and close database (Connector.java)

I created a class named Connector to connect and disconnect database. I used `prepareStatement` for accelarating executing large amount of nearly the same `sql` statements. (which would be discussed behind in part **4.1**)

```
import java.io.BufferedReader;
import java.io.FileReader;
import java.io.IOException;
import java.util.*;
import java.sql.*;
import java.net.URL;

public class Connector {
    private static final int BATCH_SIZE = 1000;

    private static Connection con = null;
    private static PreparedStatement stmt = null;

//    String url = "jdbc:postgresql://localhost:5432/Project1";
//    String user = "postgres";
//    String pwd = "u5398681234QWer";
    //......
    //......
}
```

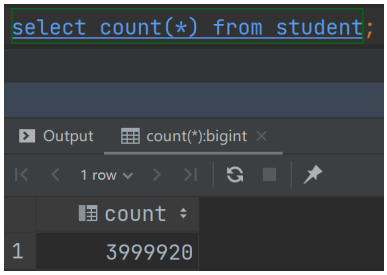
In opening the link to database, I close the automated truncation `con.setAutoCommit(false);`, which also a method to accelarate loading data to database (seen in part **4.1**). And before all actions begin, I set the sql search path to schema public `String sql = "set search_path to \"public\"";`

```
 //用于开启链接数据库
    public Connector(String url, String user, String pwd) {
        try {
            //
            Class.forName("org.postgresql.Driver");
        } catch (Exception e) {
            System.err.println("Cannot find the Postgres driver. Check
CLASSPATH.");
            System.exit(1);
        }

        Properties props = new Properties();
        props.setProperty("user", user);
        props.setProperty("password", pwd);

        try {
            con = DriverManager.getConnection(url, props);
            System.out.println("Connection succeeded");
            String sql = "set search_path to \"public\"";
            stmt = con.prepareStatement(sql);
            stmt.execute();
            con.setAutoCommit(false);
        } catch (SQLException e) {
            System.err.println("Connection failed");
            e.printStackTrace();
```

```
            System.exit(1);
        }

    }

    //用于关闭数据库
    public static void closeDB() {
        if (con != null) {
            try {
                con.commit();
                if (stmt != null) {
                    stmt.close();
                    System.out.println("Disconnection succeeded");
                }
                con.close();
                con = null;
            } catch (Exception e) {
                e.printStackTrace();
            }
        }
    }
```

All data loading depends on a series of methods named `toDB()`. Here I just listed one of them, which is used to insert data about student into table `student`, as an instance for insertion.

I used `stmt.addBatch();stmt.executeBatch();` for bulk submission (seen in part **4.1**). Since there are **3999920** data to be inserted, bulk submission actually make difference in speed.



`on conflict do nothing` in the `sql` statement is to prevent redundant submission of the same data.

```
    //toDB用于各个表的数据插入，下面只以插入student方法为例
    public static void toDB(String pathToCSVFile) {
        BufferedReader re = null;
        long begin = System.currentTimeMillis();
        long cnt = 0;
        if (con != null) {
            String sql = "insert into student (student_id, student_name, gender,
college) values (?,?,?,?) on conflict do nothing";
            try {
                re = new BufferedReader(new FileReader(pathToCSVFile));
                String line;
                String[] message;
                stmt = con.prepareStatement(sql);
                while ((line = re.readLine()) != null) {
                    message = line.split(",");
```

```
                    if (message.length > 1) {
                        stmt.setString(1, message[3].trim());
                        stmt.setString(2, message[0].trim());
                        stmt.setString(3, message[1].trim());
                        stmt.setString(4, message[2].trim());
                        stmt.addBatch();
                        cnt++;
                        if (cnt % BATCH_SIZE == 0) {
                            stmt.executeBatch();
                            stmt.clearBatch();
                        }
                    }
                }
                if (cnt % BATCH_SIZE != 0) {
                    stmt.executeBatch();
                    stmt.clearBatch();
                }
                con.commit();
                stmt.close();
            } catch (Exception e) {
                e.printStackTrace();
            } finally {
                try {
                    if (re != null) re.close();
                } catch (Exception e) {
                    e.printStackTrace();
                }
                long time = System.currentTimeMillis() - begin;
                long speed = 1000 * (cnt / time);
                System.out.printf("student insertion: %d(ms)\n Loading speed:
%d(records/s)\n", time, speed);
            }
        }
    }
    //......
    //......
```

3.2 import data

All the description is contained in the comments of the codes.

```
import com.google.gson.Gson;
import com.google.gson.GsonBuilder;
import com.google.gson.reflect.TypeToken;

import java.io.IOException;
import java.lang.reflect.Field;
import java.lang.reflect.Type;
import java.nio.file.*;
import java.util.*;
import java.sql.*;
// TODO: import the json library of your choice

public class JwxtParser {
    private static final String pathToJSONFile =
"src\\main\\java\\data\\course_info.json";
```

```java
    private static final String pathToCSVFile =
"src\\main\\java\\data\\select_course.csv";

    private static String url = "jdbc:postgresql://localhost:5432/Project1";
    private static String user = "postgres";
    private static String pwd = "u5398681234QWer";

    public static List<Course_first> courses;//预先放入courses

    public static Map<String, Course> courseHashMap;//course_id到course的map，从上
面的courses中去重
    public static ArrayList<c_Class> classes;
    public static HashSet<Teacher> teachers;
    public static Map<String, Teacher> teacherHashMap;

    //main()中直接调用下方定义的方法
    public static void main(String[] args) throws IOException {
        dataFromJSON();//从json中读取数据
        parseCourse();//处理course数据中的问题，去重
        dataToDB();//将数据提交至数据库
    }

    //......
    //......
}
```

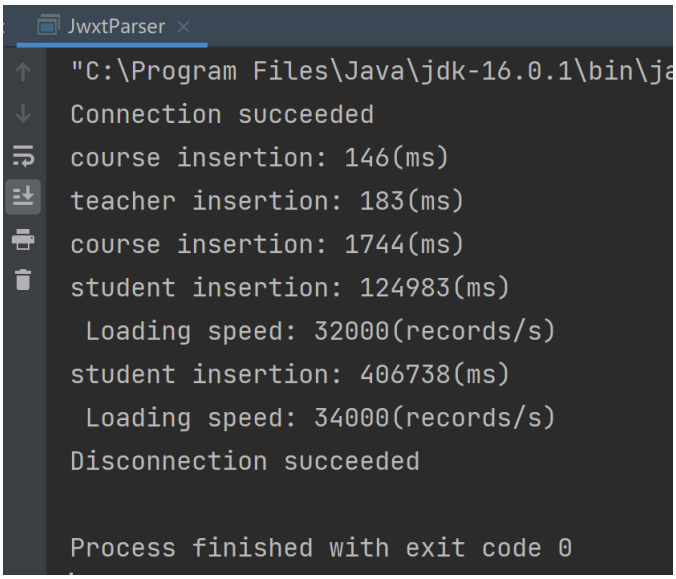
```java
    public static void dataFromJSON() throws IOException {
        String content = Files.readString(Path.of(pathToJSONFile));//content中有
json文件的全部内容
        //防止中文括号
        content = content.replaceAll("）", ")");
        content = content.replaceAll("（", "(");
        //先修中的“或者”“并且”转换
        content = content.replaceAll("或者", "or");
        content = content.replaceAll("并且", "and");
        //利用Gson将content转成Course_first类型的List，以便于后面的操作
        Gson gson = new GsonBuilder().setPrettyPrinting().create();
        courses = gson.fromJson(content, new TypeToken<List<Course_first>>() {
        }.getType());
    }

    //处理Course_first List中的问题
    public static void parseCourse() {
        classes = new ArrayList<>();
        teachers = new HashSet<>();
        courseHashMap = new HashMap<>();
        teacherHashMap = new HashMap<>();
        for (Course_first cou_f : courses) {、
            //course去重
            if (!courseHashMap.containsKey(cou_f.courseId.trim())) {
                Course cou = new Course(cou_f);
                courseHashMap.put(cou.course_id, cou);
            }
            c_Class cla = new c_Class(cou_f.className.trim(),
courseHashMap.get(cou_f.courseId.trim()));
            for (ClassList_first cl_f : cou_f.classList) {
                ClassList cl = new ClassList(cl_f);
```

```
                cla.classList.add(cl);
            }
            if (cou_f.teacher != null) {
                String[] tea = cou_f.teacher.split(",");
                for (String name : tea) {
                    Teacher teacher = new Teacher(name.trim());
                    teachers.add(teacher);
                    teacherHashMap.put(teacher.name, teacher);
                    cla.teachers.add(teacher);
                }
            }
            classes.add(cla);
        }

    }


    public static void dataToDB() {
        Connector connection = new Connector(url, user, pwd);

        //插入course
        long begin = System.currentTimeMillis();
        for (Course cou : courseHashMap.values()) {
            Connector.toDB(cou);
        }
        System.out.printf("course insertion: %d(ms)\n",
System.currentTimeMillis() - begin);

        //插入teacher
        begin = System.currentTimeMillis();
        for (Teacher te : teachers) {
            Connector.toDB(te);
        }
        System.out.printf("teacher insertion: %d(ms)\n",
System.currentTimeMillis() - begin);


        //插入class, course_class
        begin = System.currentTimeMillis();
        for (c_Class cla : classes) {
            Connector.toDB(cla);
            Connector.toDB(cla.course, cla);
            for (ClassList cl : cla.classList) {
                Connector.toDB(cl, cla);
            }
            Connector.toDB(cla, 0);
        }
        System.out.printf("course insertion: %d(ms)\n",
System.currentTimeMillis() - begin);

        //插入student
        Connector.toDB(pathToCSVFile);
        //插入course_learned
        Connector.toDB(pathToCSVFile, 0);

        Connector.closeDB();
    }
```

```
}
//......
//......
//后面是一些自定义的类，用于储存.json和.csv中各种数据
```

3.3 Time cost

Not well performed. Maybe there is an improvement.



3.4 performance

1.
```
select student_id,course_id
from course_learned
where student_id = '14999969';
```

result:

3 rows

	student_id	course_id
1	14999969	PHY205-15
2	14999969	ME103
3	14999969	PHY104B

in select_course.csv:

```
3999891    毛劳娜,M,赫奇帕奇(Hufflepuff),14999969,PHY205-15,ME103,PHY104B
```

correct result.

2.
```
select course_name,
       teacher_name
from ( class_teacher clt
    join course_class cc
    on clt.class_id = cc.class_id
         ) ct
         join course c on ct.course_id = c.course_id
         join teacher t on ct.teacher_id = t.teacher_id;
```

result:

788 rows

	course_name	teacher_name
1	体育IV	何紫琳
2	体育IV	赖莎
3	体育IV	魏伟成
4	体育IV	体育外聘教师
5	体育IV	体育外聘教师
6	体育IV	卢阳
7	体育IV	赵一品
8	体育IV	何紫琳
9	体育IV	白波
10	体育IV	魏伟成
11	体育IV	体育外聘教师
12	体育IV	卢阳
13	体育IV	赵飞
14	体育IV	何紫琳
15	体育IV	白波

3.5 Privilege management

create `newuser` and grant it by some privileges.

```
create user newuser with noinherit login password '123456';
grant connect on database "Project1" to newuser;
grant all privileges on database "Project1" to newuser;
```

```
Project1-# \du
                                   角色列表
 角色名称 |                          属性                          | 成员属于
----------+--------------------------------------------------------+----------
 newuser  | 没有继承                                               | {}
 postgres | 超级用户, 建立角色, 建立 DB, 复制, 绕过RLS             | {}
```

Symmetrically, we can revoke the user's privileges and drop the user when it no longer wanted to exist.

```
grant all privileges on database "Project1" to postgres;
grant connect on database "Project1" to postgres;
revoke all privileges on database "Project1" from newuser;
drop user newuser;
```

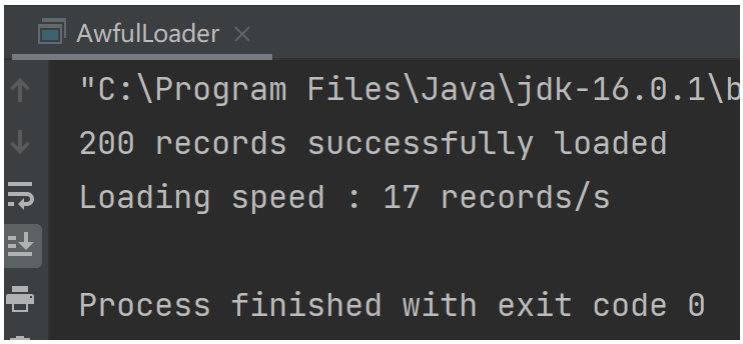
```
Project1-# \du
                                   角色列表
 角色名称 |                          属性                          | 成员属于
----------+--------------------------------------------------------+----------
 postgres | 超级用户, 建立角色, 建立 DB, 复制, 绕过RLS             | {}
```

4. Compare Efficiency

4.1 Demo loader

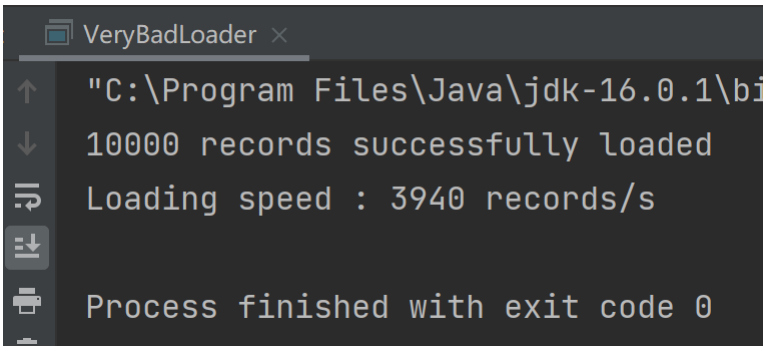
1. Awful

Everytime when `INSERT` , it needs to open and close links to database, bringing many cost. Besides, using string concatenation to build up `sql` statements also lows the efficiency. Though only 200 records were loaded, it is clear that its speed is very slow.



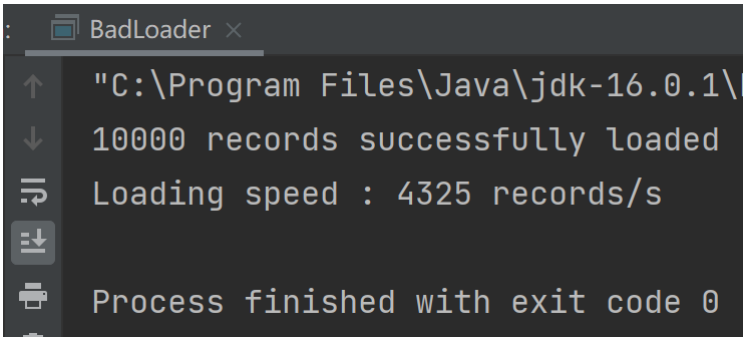
2. VeryBad

Reuse the link to databse. Having executed all the `sql` statements then close the link to database, which increasing the efficiency in some degree. However, it still uses string concatenation to build up `sql` statements.



3. Bad

It uses `prepareStatment` to shorten the time cost by a mount of same statements.



4. Average

Close `AutoCommit` and use `trancation` to write all the executed `sql` statements to the disk.

```
AverageLoader
"C:\Program Files\Java\jdk-16.0.1\bi
10000 records successfully loaded
Loading speed : 9337 records/s

Process finished with exit code 0
```

5. Good

set `BATCH_SIZE` so that send many statements to database server at one time, lowing th cost of time.

```
GoodLoader
"C:\Program Files\Java\jdk-16.0.1\bi
10000 records successfully loaded
Loading speed : 35842 records/s

Process finished with exit code 0
```

4.2 Other accelerate methods

1. unlogged

When creating the table, close the log.

```
create unlogged table students
(
    studentid varchar,
    name      varchar
)
```

The same `GoodLoader` in demo loader. It is clear that it actually speeds up by 27.24% approximately.

Before unlogged:

```
GoodLoader
"C:\Program Files\Java\jdk-16.0.1\bi
10000 records successfully loaded
Loading speed : 35842 records/s

Process finished with exit code 0
```

After unlogged:

```
GoodLoader
"C:\Program Files\Java\jdk-16.0.1\b
10000 records successfully loaded
Loading speed : 49261 records/s

Process finished with exit code 0
```

2. index

Using multi-thread program to simulate multi-query. Here simulate 200-query at the same time.

Before adding index:

After adding index:

```
"C:\Program Files\Java\
Before speed : 6 ms
Before speed : 105 ms
Before speed : 101 ms
Before speed : 106 ms
Before speed : 100 ms
Before speed : 103 ms
Before speed : 97 ms
Before speed : 102 ms
Before speed : 103 ms
```

```
"C:\Program Files\Java
After speed : 92 ms
After speed : 25 ms
After speed : 93 ms
After speed : 15 ms
After speed : 91 ms
After speed : 91 ms
After speed : 89 ms
After speed : 90 ms
After speed : 89 ms
```

Impressive! Speed up for approximately 17.94%

Program for simulating multi-query

```java
import java.util.*;
import java.sql.*;

public class MultiQuery {
    public static void main(String[] args) {
        for (int i = 0; i < 200; i++) {
            new ThreadLoader(11000001 + i).start();
        }
    }

    static class ThreadLoader extends Thread {
        public int id;

        public ThreadLoader(int id) {
            this.id = id;
        }

        public void run() {
            String url = "jdbc:postgresql://localhost:5432/Project1";
            Properties props = new Properties();
            props.setProperty("user", "postgres");
            props.setProperty("password", "u5398681234QWer");
            long start = 0;
            try {
                Connection con = DriverManager.getConnection(url, props);
                con.setAutoCommit(false);

                start = System.currentTimeMillis();
                Statement stmt = con.createStatement();
                stmt.executeQuery("select count(*) from students where
studentid > " + id);

                con.commit();
                stmt.close();
                con.close();
            } catch (SQLException se) {
                System.err.println("SQL error: " + se.getMessage());
            } finally {
                System.out.println("After speed : " +
(System.currentTimeMillis() - start) + " ms");
            }
        }
    }
```

```
}
```

4.3 File Iuput VS JDBC Input

First I wrote a program to generate some sql statements into `students.sql`, which is used to insert data into table `students`.

```
public static void writeFile(ArrayList<Student> students) throws IOException {
        BufferedWriter bw = null;
        String sql = "insert into students (student_id, student_name, gender,
college) values (\'%s\',\'%s\',\'%s\',\'%s\')\n";
        try {
            bw = new BufferedWriter(new
FileWriter("src/main/java/data/students.sql"));
            for (Student s : students) {
                bw.append(String.format(sql, s.student_id, s.name, s.gender,
s.college));
            }
        } catch (Exception e) {
            e.printStackTrace();
        } finally {
            try {
                assert bw != null;
                bw.close();
            }catch (Exception e) {
                e.printStackTrace();
            }
            System.out.println("write finished");
        }
    }
```

```
insert into students (student_id, student_name, gender, college) values ('11000001','苗级彩','F','阿兹卡班(Azkaban
insert into students (student_id, student_name, gender, college) values ('11000002','潘冰泪','F','斯莱特林(Slyther
insert into students (student_id, student_name, gender, college) values ('11000003','奚够啊','M','斯莱特林(Slyther
insert into students (student_id, student_name, gender, college) values ('11000004','韩落门','M','赫奇帕奇(Huffle
insert into students (student_id, student_name, gender, college) values ('11000005','齐油找','M','拉文克劳(Ravenc
insert into students (student_id, student_name, gender, college) values ('11000006','昌珍初','M','阿兹卡班(Azkaban
insert into students (student_id, student_name, gender, college) values ('11000007','皮底史','F','赫奇帕奇(Huffle
insert into students (student_id, student_name, gender, college) values ('11000008','伍等破','F','斯莱特林(Slyther
insert into students (student_id, student_name, gender, college) values ('11000009','卜务机','F','斯莱特林(Slyther
insert into students (student_id, student_name, gender, college) values ('11000010','雷深切','M','格兰芬多(Gryffin
insert into students (student_id, student_name, gender, college) values ('11000011','潘压店','F','阿兹卡班(Azkaban
insert into students (student_id, student_name, gender, college) values ('11000012','卜恐质','M','赫奇帕奇(Huffle
insert into students (student_id, student_name, gender, college) values ('11000013','宋晚忆','F','赫奇帕奇(Huffle
insert into students (student_id, student_name, gender, college) values ('11000014','和寻玩','M','斯莱特林(Slyther
insert into students (student_id, student_name, gender, college) values ('11000015','董问岁','F','赫奇帕奇(Huffle
insert into students (student_id, student_name, gender, college) values ('11000016','云基怪','F','阿兹卡班(Azkaban
insert into students (student_id, student_name, gender, college) values ('11000017','王里是','M','格兰芬多(Gryffin
```

Then I use this file to insert data into database and calculate the time cost.

```
Summary: 3,999,920 of 3,999,920 statements executed in 11 min, 49 sec, 538 ms
```

Comparing to the JDBC, it is a pity that FileIO is much slower than JDBC. (about 82.37%)

```
student insertion: 124983(ms)
 Loading speed: 32000(records/s)
```

5 Summary

This program design a related database that contains courses data and some information about teachers and students.