Contribution:

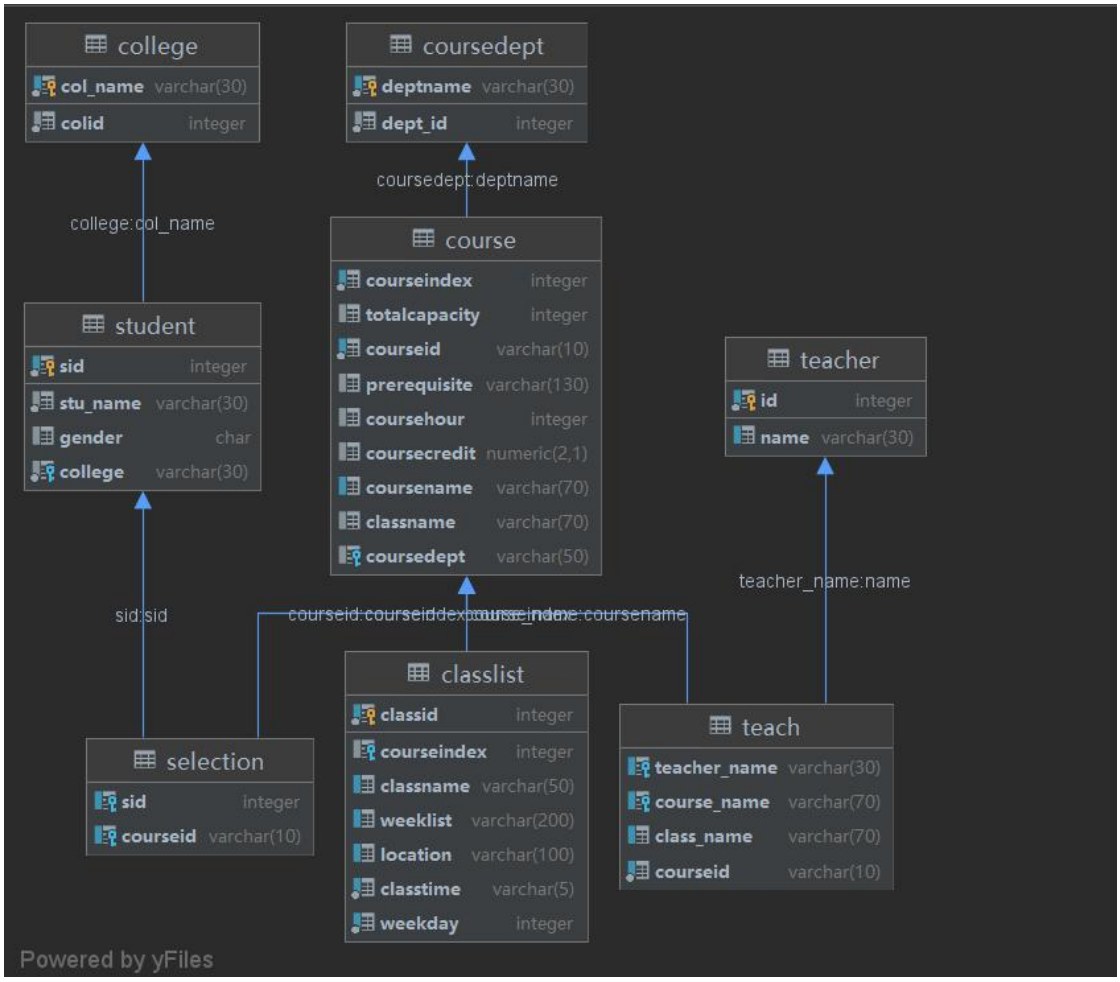
1.通过观测数据，分析数据之间的各种联系，创建了多个关系表格，并处理不合理的数据，例如去除多余的空字符，提取出 null 数据并进行合理的填入，并且将一对多的数据进行拆分，拆分成多个表格，建立一对多的关系表格，将数据有机地联系在一起，且便于查询，对以后不合理数据的导入也有防止插入的功能。

2.以导入学生 csv 文件为例，对导入四百万条数据进行四次优化，最后显著提高了导入的效率，在三四十秒的时间里面就可以完整地导入数据。

Task1:

首先观察学生的表格，一个学生可以选择多门课程，由于在数据库的表中，对于同一个元组，同一个属性的值不能有多个，所以可以由此可以构建三个表，将学生选择的课程独立开，建立一个 selection 的新表格，通过 selection 中的 sid 外键连接学生，courseid 外键连接课程（course）可以建立学生和课程一对多的关系，进而对 student 表而言只有四个属性，代表了学生的个人信息（名字、书院、性别等等），由于学生的学号是唯一的，就可以将学生的 sid 作为主键存储。其次为了方便管理以及后续数据库数据的加入，我还将学生对应的书院单独拆开形成了另一个表格，建立学生和书院的外键联系，防止后续如果要加入学生，但是他的书院名称对不上书院表里面的名称的不合理情况。

对于课程 course 表格，一开始观察 course 表格，发现 classlist 属性包含的信息非常多，而两者包含的信息层次也是不一样的，一个 course 可以有多个 classlist,所以我将 classlist 单独分开作为一个新表格，classlist 包含一个唯一的自增 id 序列作为主键，其次由于一个 course 可以对多个 classlist，为了将他们联系在一起，我给 course 表添加了一个属性，courseindex，使得 classlist 可以通过 courseindex 和 course 表格联系在一起，这样，就成功建立了 course 和 classlist 的一对多的关系。通过检索数据，发现有些班（class）的开课地点存在空字符串，进一步处理数据，将空字符串填写为 null 添加到数据库中。进而通过观察表格，发现一个课程（course）可以对应多个老师进行授课，进而由于数据库的范式，我将 teacher 单独拆分为一个独立的表格，并且建立了一个阐述他们之间教学关系的 teach 表格，teach 表格通过 teacher_name 外键连接 teacher 表格的 name 属性，通过 coursename 外键连接 course 表格里面的 courseid，为了更加详细的阐述 teach 的关系，我另外给 teach 表格增添了 class_name 和 courseid 两个属性，这样 teacher 和 classlist 以及 course 的关系就在表格显示的更为清楚。在插入 teacher 表格数据的时候，经过数据检索发现 teacher 的数据中有额外的空格以及空数据，经过处理，去掉了额外的空格，将空数据存储为 null 加入到 teacher 列表中。同样的，为了后续课程数据插入的合理性，我将 course 表格中的 courseDept 属性拆分成一个独立的表格，以开课院系唯一非空的名称作为主键，添加了自增 ID，进一步唯一标识院系，整理发现有 23 个开课院系，之后在添加课程信息的时候如果不存在这个院系，则需先在 coursedept 表格中添加相关院系，否则 course 数据将会被视为不合理数据，无法插入。



这样，就建立起了第一幅图中的表之间的关系图。

Task2:

以导入学生的 csv 文件为例，阐述如何设计程序对文件进行导入：

首先是建立一个学生类型 student_infos: 在列表中对学生的选课信息进行处理，切割成一个字符串列表，方便之后在方法中对数据的处理。

```
class student_info{
    String stu_name ;
    String gender;
    String college;
    int sid;
    ArrayList<String> course = new ArrayList<>();
    student_info(String[] str){
        stu_name = str[0];
        gender = str[1];
        college = str[2];
        sid = Integer.valueOf(str[3]);
        for(int i = 4; i < str.length; i ++){
            course.add(str[i]);
```

```
        }
    }
    public String toString(){
        return String.format("The student is %s, %s, %s, %d, %s",
stu_name,gender, college, sid, course);
    }
```

之后建立一个存储学生 csv 文件信息的列表，使用 File 读取文件

```
ArrayList student_infos = new ArrayList<student_info>();

File file = new
File("C:\\Users\\86181\\Downloads\\data\\select_course.csv");
Scanner input = new Scanner(file);
while(input.hasNextLine()){
    String str1 = input.nextLine();
    String[] str_split = str1.split(",");
    student_infos.add(new student_info(str_split));
}
```

然后创建方法 addOneStudent，对每一个学生进行遍历，每一次遍历开启一次数据库对单个学生进行信息导入，添加完一个学生之后关闭数据库。

```
        //首次构建
public void addOneStudent(List<student_info> student_info) {
        long begin = System.currentTimeMillis();
        try {
            for(student_info student_info1: student_info){
                getConnection();

                addOneStudent(student_info1);

                closeConnection();          }
        } catch (SQLException e) {
            e.printStackTrace();
        }
    }
    private void addOneStudent(student_info student_info) throws
SQLException {
        int result = 0;

        String sql = "insert into student(stu_name, gender, college, sid)"
                + "values (?,?,?,?)";
        PreparedStatement preparedStatement = con.prepareStatement(sql);
```

```
        preparedStatement.setString(1,student_info.stu_name);
        preparedStatement.setString(2, student_info.gender);
        preparedStatement.setString(3,student_info.college);
        preparedStatement.setInt(4,student_info.sid);
        result = preparedStatement.executeUpdate();

    }
```

但是初次构建导入学生数据的速度非常慢，只能达到六条记录/秒，经过对比 lab4 的几个类，

```
//The speed is 6 records/s
```

以及询问老师，网上查询资料，我发现可以通过添加一个常数，这个常数为 2 的指数，关闭程序的自动提交功能，在经过了一定量的数据积压时候统一提交，为了防止漏掉最后几个数据，在方法的最后，进行收尾提交，加上将 getconnection()方法放在执行方法开始，避免每遍历一个学生就开启一次数据库这种非常耗时的操作，这样显著的提升了导入数据的效率以及速度

```
//初次优化
public void addOneStudent(List<student_info> student_info) {
    long begin = System.currentTimeMillis();
    getConnection();
    int i  = 0;
    double j = 0;
    int buffer = 1024;
    try {
        con.setAutoCommit(false);
        for(student_info student_info1: student_info){
            addOneStudent(student_info1);
            if (++i % buffer == 0) {
                con.commit();
                System.out.println(i);
            }
        }

        con.commit();
    } catch (SQLException e) {
        e.printStackTrace();
    }
    finally {
        closeConnection();
    }
}
private void addOneStudent(student_info student_info) throws
SQLException {
    int result = 0;
```

```
    String sql = "insert into student(stu_name, gender, college, sid)"
            + "values (?,?,?,?)";
    PreparedStatement preparedStatement = con.prepareStatement(sql);
    preparedStatement.setString(1,student_info.stu_name);
    preparedStatement.setString(2, student_info.gender);
    preparedStatement.setString(3,student_info.college);
    preparedStatement.setInt(4,student_info.sid);
    result = preparedStatement.executeUpdate();

}
//The speed is 2487.5621890547263 records/s
```

但是这样的速度还是不能令人满意，于是进行了第二次优化，利用 JDBC 的分批操作，使用了 PreparedStatement 类里面的 addbatch 方法，结合第二次优化的分批执行数据导入的方法，在添加了一部分数据进入缓存区了之后集中对这些分批的数据进行处理，执行结束后就清除缓存区，同时为了保证数据的完整性，在方法结束再次执行分批处理，清除分批缓存区。

```
//第二次优化(addbatch)
public void addOneStudent(List<student_info> student_info) {

        long begin = System.currentTimeMillis();
        getConnection();
        int i  = 0;
        int bar = 1024;
        try {
            String sql = "insert into student(stu_name, gender, college,
sid)"
                    + "values (?,?,?,?)";
            PreparedStatement preparedStatement =
con.prepareStatement(sql);

            for(student_info student_info1: student_info){
                preparedStatement.setString(1,student_info1.stu_name);
                preparedStatement.setString(2, student_info1.gender);
                preparedStatement.setString(3,student_info1.college);
                preparedStatement.setInt(4,student_info1.sid);

                preparedStatement.addBatch();

                if(++i % bar == 0){
                    preparedStatement.executeBatch();
                    preparedStatement.clearBatch();
                }
                if(i == 1000000){
                    System.out.println("The speed is " + i*
```

```
1000.0/(System.currentTimeMillis() - begin) + "records/s");
                }
            }
            preparedStatement.executeBatch();
            preparedStatement.clearBatch();
        } catch (SQLException e) {
            e.printStackTrace();
        }
        finally {
            closeConnection();
        }
    }
//The speed is 11025.358324145534records/s
```

这样导入四百万条数据依然还是需要不少时间，于是进行了第三次优化，在这次优化当中我关闭了自动提交，在代码结束之后再统一对未处理的数据进行提交，结合了第一二次优化的结果，最后的速度提升了四倍

```
//第三次优化
public void addOneStudent(List<student_info> student_info) {

        long begin = System.currentTimeMillis();
        getConnection();
        int i  = 0;
        int counter = 0;
        int bar = 1024;


        String sql = "insert into student(stu_name, gender, college, sid)
"+ "values (?,?,?,?)";
        try {
                PreparedStatement preparedStatement =
con.prepareStatement(sql);
                        con.setAutoCommit(false);
                for (student_info student_info1 : student_info) {
                        preparedStatement.setString(1,
student_info1.stu_name);
                        preparedStatement.setString(2,
student_info1.gender);
                        preparedStatement.setString(3,
student_info1.college);
                        preparedStatement.setInt(4, student_info1.sid);
                        preparedStatement.addBatch();

                    if (++i % bar == 0) {
```

```
                    preparedStatement.executeBatch();
                    preparedStatement.clearBatch();
                }
            }
            con.commit();

            if (i == 1000000) {
                System.out.println("The speed is " + i * 1000.0 /
(System.currentTimeMillis() - begin) + "records/s");
            }
        } catch (SQLException e) {
            e.printStackTrace();
        }
        finally {
            closeConnection();
                System.out.println("The speed is " + i * 1000.0 /
(System.currentTimeMillis() - begin) + "records/s");
        }
    }
//The speed is 44536.58753841357records/s
```

经过网上查询资料，发现 insert 语句在实际执行的时候也会损耗一定时间，于是我尝试着将 insert 语句进行拼接，经过计算，发现将 320 个 values 值进行拼接可以达到显著的速度提升效果，结合前面所提到的三种优化，最终达到了 13w 条数据/秒的效率，相对之前的优化都达到了显著提升。导入所有的将近四百万条数据只需要半分钟多的时间。

```
//第四次优化（拼接 insert）
public void addOneStudent(List<student_info> student_info) {

    long begin = System.currentTimeMillis();
    getConnection();
    int i = 0;
    int counter = 0;
    int bar = 2048;
    StringBuilder sql = new StringBuilder("insert into student(stu_name,
gender, college, sid) values ");
    String sql2 = "(?,?,?,?)";
    final int NUMBER_OF_SIZE = 80;
    //循环 80 次，拼接 320 个栏子
    for (int cnt = 0; cnt < NUMBER_OF_SIZE; cnt++) {
        if (cnt == NUMBER_OF_SIZE - 1) sql.append(sql2).append(";");
        else sql.append(sql2).append(",");
    }
    PreparedStatement preparedStatement = null;

    try {
```

```
        con.setAutoCommit(false);
        int num = 0;
        preparedStatement = con.prepareStatement(sql.toString());
        for (student_info student_info1 : student_info) {
            ++i;
            {
                num += 4;
                preparedStatement.setString(num - 3,
student_info1.stu_name);
                preparedStatement.setString(num - 2,
student_info1.gender);
                preparedStatement.setString(num - 1,
student_info1.college);
                preparedStatement.setInt(num, student_info1.sid);
            }
            if (num / 4 % NUMBER_OF_SIZE == 0) {
                preparedStatement.addBatch();
                num = 0;
                counter ++;
            }
            if (counter > 0 && counter % bar == 0) {
                preparedStatement.executeBatch();
                preparedStatement.clearBatch();
            }
        }
        preparedStatement.executeBatch();
        con.commit();
    } catch (SQLException e) {
        e.printStackTrace();
    } finally {
        closeConnection();
        System.out.println("The speed is " + i * 1000L /
(System.currentTimeMillis() - begin) + "records/s");
    }
}
// The speed is 131278records/s
```