Fall 2021 CS307 Database Project-2

1. Title

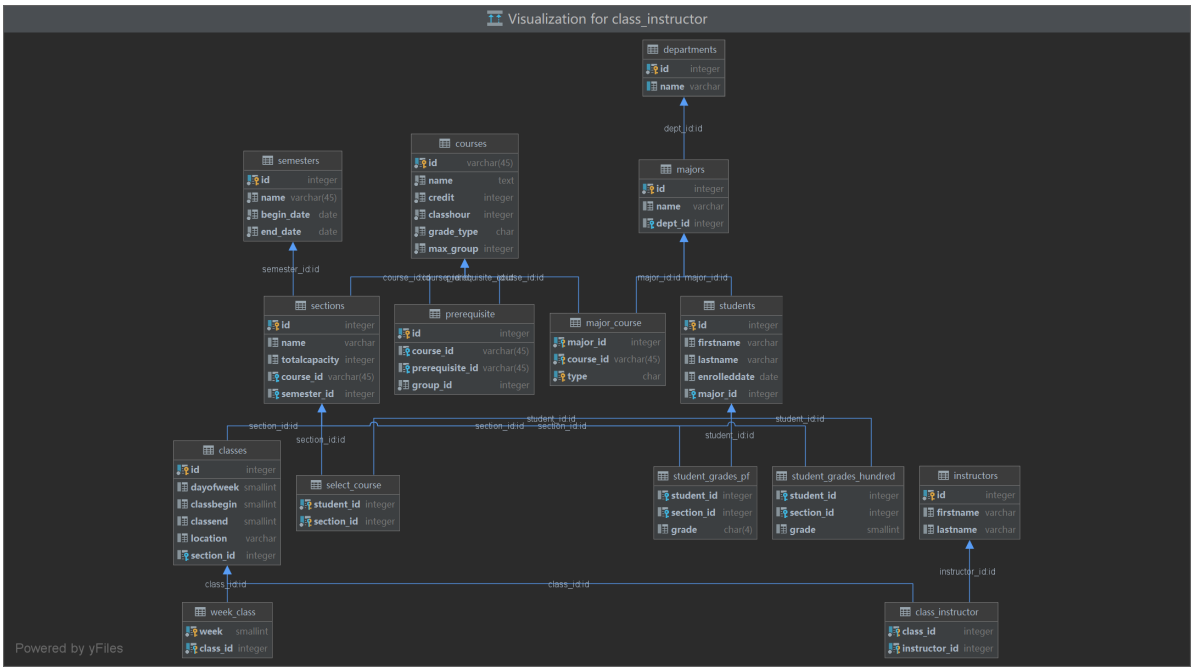
Educational Administration System --ADVANCE

2. Group members and contribution

name	SID	work	percentage
谢子晟	12011919	database design, prerequisite design	33.3%
简欣瑶	11911838	database design, most `service` design	33.3%
刘乐奇	12011327	database design, user, student, instructor design	33.3%

3. Database design

3.1 The structure of whole database



3.2 Description of database

The specific codes can be seen in the attachments. All the foreign keys are `on delete cascade on update cascade`.

3.2.1 tables

courses

column's name	data type	description
id	varchar(45)	primary key
name	text	not null
credit	int	not null, check >= 0
classhour	int	not null, check >=0
grade_type	char(1)	not null, check in ('P', 'H')
max_group	int	not null, check >= 0

semesters

column's name	data type	description
id	serial	primary key
name	varchar(45)	not null
begin_date	date	not null
end_date	date	not null

sections

column's name	data type	description
id	serial	primary key
name	varchar	
total_capacity	int	
course_id	varchar(45)	foreign key, reference courses (id)
semester_id	int	foreign key, reference semesters (id)

classes

column's name	data type	description
id	serial	primary key
dayofweek	smallint	check between 1 and 7
classbegin	smallint	check between 1 and 12
classend	smallint	check between classbegin and 12
location	varchar	
section_id	int	foreign key, reference sections (id)

prerequisite

column's name	data type	description
id	serial	primary key
course_id	varchar(45)	foreign key, reference courses (id)
prerequisite_id	varchar(45)	foreign key, reference courses (id)
group_id	int	not null

departments

column's name	data type	description
id	serial	primary key
name	varchar	unique

majors

column's name	data type	description
id	serial	primary key
name	varchar	
dept_id	int	foreign key, reference departments (id)

instructors

column's name	data type	description
id	int	primary key
firstname	varchar	
lastname	varchar	

students

column's name	data type	description
id	int	primary key
firstname	varchar	
lastname	varchar	
enrolleddate	date	
major_id	int	foreign key, reference majors (id)

select_course

column's name	data type	description
student_id	int	primary key, foreign key, reference students (id)
section_id	int	primary key, foreign key, reference sections (id)

class_instructor

column's name	data type	description
class_id	int	primary key, foreign key, reference classes (id)
instructor_id	int	primary key, foreign key, reference instructors (id)

student_grades_pf

column's name	data type	description
student_id	int	primary key, foreign key, reference students (id)
section_id	int	primary key, foreign key, reference sections (id)
grade	char(4)	check in ('PASS', 'FAIL')

student_grades_hundred

column's name	data type	description
student_id	int	foreign key, reference students (id)
section_id	int	foreign key, reference sections (id)
grade	smallint	check between 0 and 100

major_course

column's name	data type	description
major_id	int	foreign key, reference majors (id)
course_id	varchar(45)	foreign key, reference courses (id)
type	char(1)	default 'A', check in ('A', 'C', 'E')

week_class

column's name	data type	description
week	smallint	primary key, check between 1 and 16
class_id	int	primary key, references classes (id)

3.2.2 functions

getfullname

```
create or replace function getfullname(firstname varchar, lastname varchar)
returns varchar
as
$$
begin
    if (firstname ~ '[a-zA-z]' and lastName ~ '[a-zA-z]') then
        return firstname || ' ' || lastname;
    else
        return firstname || lastname;
    end if;
end
$$ language plpgsql;
```

3.3 Import data

于除prerequisite表以外的表的插入,大部分使用的方式均为最简单的直接插入，小部分存在一层以内的分类讨论

- Department的插入：直接插入

```
@Override
public int addDepartment(String name) {
    try(Connection con = SQLDataSource.getInstance().getSQLConnection();
        PreparedStatement p = con.prepareStatement( sql: "insert into departments (name) values (?);",PreparedStatement.RETURN_GEN
    ){
        p.setString( parameterIndex: 1,name.trim());
        p.executeUpdate();

        ResultSet resultSet = p.getGeneratedKeys();
        if(resultSet.next()){
                System.out.println(resultSet.getInt(1));
            return resultSet.getInt( columnIndex: 1);
        }else{
            throw new EntityNotFoundException();
        }

    }catch (SQLException e){
        e.printStackTrace();
        throw new IntegrityViolationException();
    }
}
```

- Courses的插入：分类插入

```
try (Connection con = SQLDataSource.getInstance().getSQLConnection();
PreparedStatement stmt = con.prepareStatement( sql: "insert into courses(id,name,credit,classHour,grade_type,max_group)  " +
                                                   "values(?,?,?,?,?,?)")){
    stmt.setString( parameterIndex: 1,courseId);
    stmt.setString( parameterIndex: 2,courseName);
    stmt.setInt( parameterIndex: 3,credit);
    stmt.setInt( parameterIndex: 4,classHour);
    if (grading== Course.CourseGrading.HUNDRED_MARK_SCORE){
        stmt.setString( parameterIndex: 5, x: "H");
    }else if (grading == Course.CourseGrading.PASS_OR_FAIL){
        stmt.setString( parameterIndex: 5, x: "P");
    }
    stmt.setInt( parameterIndex: 6, x: 0);
    stmt.executeUpdate();
```

- prerequisite的插入

由于一个先修课中可能会出现嵌套中仍有嵌套的“套娃结构”,所以需要对给出的prerequisite进行处理,预处理（具体处理方法详见下）

```
Prerequisite advanced = null;
if (prerequisite !=null) {
    advanced = PrerequisiteService.findAllSatisfied(prerequisite);
}
```

在这里我们给courses的各行添加了一个属性：Max_Group 表示一门课它的所有可行的先修课组合数

```
if (advanced !=null) {
    int max_group = insertAdvancedPrerequisite(courseId, advanced);
    PreparedStatement ps = con.prepareStatement( sql: "update courses set max_group = ? where max_group=0;");
    ps.setInt( parameterIndex: 1,max_group);
    ps.executeUpdate();
}
```

其默认值为0，意为没有先修课 。而在先修课被prerequisiteService处理后先修课组合数被返回，用update更新该课程的max_group默认值。

- StudentCourses的插入

先要对输入的数据进行一次判断，取用它的section_id，在sections表中得到这门课的grade_type

```
PreparedStatement stmt1 = con.prepareStatement( sql: "select grade_type from (select course_id " +
                            " from sections where id = ?) sec join courses on sec.course_id = courses.id");
stmt1.setInt( parameterIndex: 1, sectionId);

ResultSet rs = stmt1.executeQuery();
String grade_type;
if (!rs.next()) {
    throw new IntegrityViolationException();
} else {
    grade_type = rs.getString( columnIndex: 1);
    stmt1.close();
    rs.close();
}
```

然后根据两种情况导入到两种表中

```
if (grade == null) {
    if (grade_type.equals("P")) {
        PreparedStatement stmt2 = con.prepareStatement( sql: "insert into student_grades_pf (student_id, section_id, grade)
        stmt2.setInt( parameterIndex: 1, studentId);
        stmt2.setInt( parameterIndex: 2, sectionId);
        stmt2.setNull( parameterIndex: 3, Types.CHAR);
        stmt2.executeUpdate();
        stmt2.close();
    } else {
        PreparedStatement stmt2 = con.prepareStatement( sql: "insert into student_grades_hundred (student_id, section_id,
        stmt2.setInt( parameterIndex: 1, studentId);
        stmt2.setInt( parameterIndex: 2, sectionId);
        stmt2.setNull( parameterIndex: 3, Types.SMALLINT);
        stmt2.executeUpdate();
        stmt2.close();
    }
} else {
```

```
} else {
    if (grade instanceof PassOrFailGrade) {
        PreparedStatement stmt3 = con.prepareStatement( sql: "insert into student_grades_pf (student_id, section_id, grade)
        stmt3.setInt( parameterIndex: 1, studentId);
        stmt3.setInt( parameterIndex: 2, sectionId);
        stmt3.setString( parameterIndex: 3, ((PassOrFailGrade) grade).name());
        stmt3.executeUpdate();
        stmt3.close();
    } else if (grade instanceof HundredMarkGrade) {
        PreparedStatement stmt3 = con.prepareStatement( sql: "insert into student_grades_hundred (student_id, section_id, gr
        stmt3.setInt( parameterIndex: 1, studentId);
        stmt3.setInt( parameterIndex: 2, sectionId);
        stmt3.setShort( parameterIndex: 3, ((HundredMarkGrade) grade).mark);
        stmt3.executeUpdate();
    }
}
```

4. Prerequisite

Idea

先修课关系非常复杂，设计数据库表来存储先修课关系并判断是否满足先修课非常困难。所以我们决定在导入数据库前对先修课关系进行处理。我们发现：**判断一名学生已经修过的课程是否满足先修课条件**等价于 **判断这名学生已经修过的课程 是否 至少包含了所有满足先修课的课程组合中的一个。**

具体步骤：

1. 找出这门课程中所有是先修的课程： 获取`prerequisite`中所有的`CoursePrerequisite`。
2. 找出所有满足先修课的课程组合：遍历得到的`CoursePrerequisite`的所有组合，测试是否能通过先修课，如果可以则保留该组合。
3. 判断是否满足先修：将所有满足先修课的课程组合 和 学生修过的课程集合进行比较，如果其中一个组合被学生修过的课程集合包含，则学生满足先修。

例子：

假设课程`cs307`的先修课关系为`(A&B)||C`，那么所有满足先修课的课程组合为`B,C`,`A,C`和`A,B,C`。想选这门课的小明已经修过`A,C,D`，包含了`A,C`，所以小明满足这门课的先修。 想选这门课的小丽已经修过了`A,E`,不包含任何一个组合，所以小丽不满足这门课的先修。

实现：

- 步骤1,2在`PrerequisiteService.java`实现
- 步骤3在`MyStudentService.java`中`passedPrerequisitesForCourse`实现。

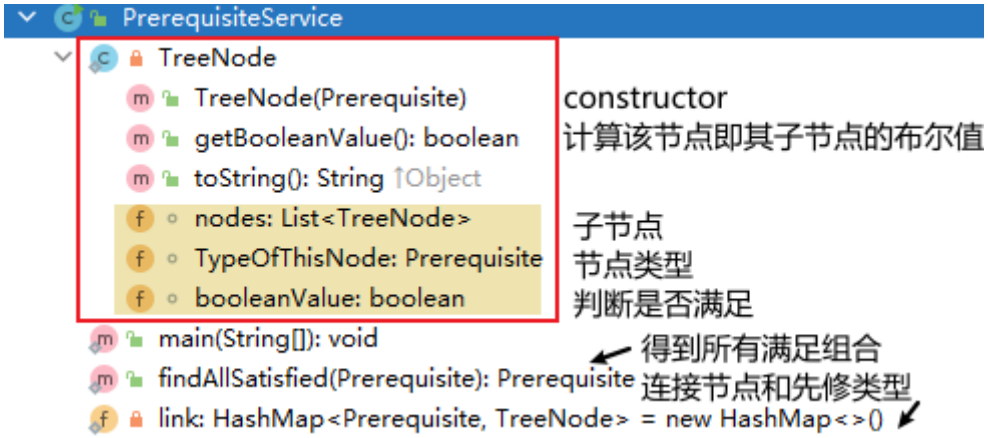
Store in Database

table : prerequisite

- id : primary key 无特殊含义
- course_id : 课程
- prerequisite_id : 课程所对应的先修课程
- group_id : 先修课程组合的id，某能满足先修课的课程组合有相同的group_id

Implement in Java

PrerequisiteService.java



- (class)`TreeNode`：储存当前节点的布尔值，用来进行逻辑判断
- (HashMap)`link`：记录`Prerequisite`和`TreeNode`的关联，便于互相查找。

TreeNode

```
private static class TreeNode{
        List<TreeNode> nodes; // child nodes
        Prerequisite TypeOfThisNode;
        boolean booleanValue ;

        // construct
        public TreeNode(Prerequisite p){
            nodes = new ArrayList<>();
            TypeOfThisNode = p;
            if(p instanceof CoursePrerequisite){
                CoursePrerequisite c = (CoursePrerequisite) p;
                link.put(c,this); // link
            }else if(p instanceof OrPrerequisite){
                OrPrerequisite o = (OrPrerequisite) p;
```

```
                link.put(o,this); // link
                for(Prerequisite n :((OrPrerequisite) TypeOfThisNode).terms) {
                    TreeNode child = new TreeNode(n); // grow
                    nodes.add(child);
                }
            }else {
                AndPrerequisite a = (AndPrerequisite) p;
                link.put(a,this);// link
                for(Prerequisite n :((AndPrerequisite) TypeOfThisNode).terms) {
                    TreeNode child = new TreeNode(n);// grow
                    nodes.add(child);
                }
            }
        }

        // getBooleanValue
        public boolean getBooleanValue(){
            boolean here;
            if(TypeOfThisNode instanceof CoursePrerequisite){
                here = booleanValue;
                return here;
            }else if(TypeOfThisNode instanceof OrPrerequisite){
                boolean temp = FALSE;
                for(TreeNode n : nodes){
                    temp = n.getBooleanValue() | temp;
                }
                return temp;
            }else {
                boolean temp = TRUE;
                for(TreeNode n : nodes){
                    temp = n.getBooleanValue() & temp;
                }
                return temp;
            }
        }
    }
```

link

```
// link between treeNods and Prerequisite structure
private static HashMap<Prerequisite,TreeNode> link = new HashMap<>();
```

findAllSatisfied

```
public static Prerequisite findAllSatisfied(Prerequisite prerequisite){
        if(prerequisite == null || prerequisite instanceof CoursePrerequisite){
            return prerequisite;
        }

        //////// data input and grow tree
        link = new HashMap<>();
        TreeNode tn = new TreeNode(prerequisite);

        //////// get all needed courses
        List<CoursePrerequisite> cpl = new ArrayList<>();
        for(Prerequisite key : link.keySet()){
            if(key instanceof CoursePrerequisite ){
```

```
                if(!cpl.contains(key)){
                    cpl.add((CoursePrerequisite) key);
                }
            }
        }

        //////// test all possibility : 2^n (n : # of courses mentioned in
total)
        int numOfCourse = cpl.size();
        int numOfTest = (int) Math.pow(2,numOfCourse);

        ArrayList<Prerequisite> all_satisfied = new ArrayList<>();
        // 0 does not need to be test
        for (int i = 1; i < numOfTest; i++) {
            List<Prerequisite> cp_testing = new ArrayList<>();
            // use binary to list iterate all possible
            String s = Integer.toBinaryString(i);
            String[] ss = s.split("");
            int numOfUnTest = numOfCourse- ss.length;
            // test one certain combination
            for (int j = 0; j < numOfCourse; j++) {
                TreeNode temp = link.get(cpl.get(j));
                if(j<numOfUnTest){
                    temp.booleanValue = FALSE;
                }else {
                    int x = Integer.parseInt(ss[j-numOfUnTest]);
                    temp.booleanValue =( x != 0);
                    if(x!=0){
                        cp_testing.add(cpl.get(j));
                    }
                }
            }
            // if the combination passed the result, then collect it.
            if(tn.getBooleanValue()){
                if(cp_testing.size() ==1){
                    all_satisfied.add(cp_testing.get(0));
                }else{
                    // use AND to link courses in one combination
                    all_satisfied.add(new AndPrerequisite(cp_testing));
                }
            }
         }
        // use OR to link all combinations
        return new OrPrerequisite(all_satisfied);
    }
```

MyStudentService.java

```
public boolean passedPrerequisitesForCourse(int studentId, String courseId) {
        try (Connection con = SQLDataSource.getInstance().getSQLConnection()) {

            //得到courseId的先修课列表
            PreparedStatement stmt = con.prepareStatement("select
prerequisite_id,group_id" +
                    " from (select *" +
```

```
                    "       from prerequisite" +
                    "       order by (course_id,group_id) ) a" +
                    " where course_id = ?;");
            stmt.setString(1, courseId);
            ResultSet rs = stmt.executeQuery();

            int group_id = 1;
            String prerequisite;
            int temp;
            ArrayList<ArrayList<String>> prerequisites = new ArrayList<>();
            ArrayList<String> group = new ArrayList<>();
            if (rs.next()) {
                prerequisite = rs.getString(1);
                temp = rs.getInt(2);
                if (group_id != temp) {
                    prerequisites.add(group);
                    group = new ArrayList<>();
                    group.add(prerequisite);
                    group_id++;
                } else {
                    group.add(prerequisite);
                }
                while (rs.next()) {
                    prerequisite = rs.getString(1);
                    temp = rs.getInt(2);
                    if (group_id != temp) {
                        prerequisites.add(group);
                        group = new ArrayList<>();
                        group.add(prerequisite);
                        group_id++;
                    } else {
                        group.add(prerequisite);
                    }
                }
            } else {
                //System.out.println(true);
                return true;
            }


            rs.close();

            //得到学生的选课清单
            PreparedStatement ps = con.prepareStatement("select course_id" +
                    "from students s" +
                    "     inner join student_grades_hundred sgh on s.id =
sgh.student_id" +
                    "        inner join sections s2 on s2.id = sgh.section_id" +
                    "where s.id = ?" +
                    "union" +
                    "select course_id" +
                    "from students s" +
                    "     inner join student_grades_pf sgp on s.id =
sgp.student_id" +
                    "        inner join sections s2 on s2.id = sgp.section_id" +
                    "where s.id = ?;");
            ps.setInt(1, studentId);
            ps.setInt(2, studentId);
```

```
            ResultSet rs2 = ps.executeQuery();

            ArrayList<String> selected = new ArrayList<>();

            String courseTemp;
            if (rs2.next()) {
                courseTemp = rs2.getString(1);
                selected.add(courseTemp);
                while (rs2.next()) {
                    courseTemp = rs2.getString(1);
                    selected.add(courseTemp);
                }
            } else {
                return false;
            }

            //挨个比对prerequisites各个组与selected的内容
            //只要某个组里面的所有元素都在selected中有出现，就返回true，如果所有组都没有满足的，返回false
            boolean isSatisfied = false;
            boolean isSatisfiedPartly;
            for (ArrayList<String> t : prerequisites) {
                isSatisfiedPartly = true;
                for (String s : t) {
                    if (!selected.contains(s)) {
                        isSatisfiedPartly = false;
                    }
                }
                if (isSatisfiedPartly) {
                    isSatisfied = true;
                    break;
                }
            }
            return isSatisfied;

        } catch (Exception e) {
            e.printStackTrace();
            throw new IntegrityViolationException();
        }
//        return false;
    }
```

5. searchCourse 实现

在完成`searchCourse()`这个方法时，写了一些辅助类和辅助方法如下：

类名/方法名	作用
`public static class CouToSec`	将Course和对应的CourseSection链接起来
`public static class searchProperty`	集合了一系列课程的性质，用于应对`searchCourse()`中的四个`boolean`参数
`public List<String> getCourseType`	通过searchCourseType获得全部符合条件的courseId

```
List<CourseSearchEntry> searchCourse(int studentId, int semesterId, @Nullable String searchCid,
                                     @Nullable String searchName, @Nullable String searchInstructor,
                                     @Nullable DayOfWeek searchDayOfWeek, @Nullable Short searchClassTime,
                                     @Nullable List<String> searchClassLocations,
                                     CourseType searchCourseType,
                                     boolean ignoreFull, boolean ignoreConflict,
                                     boolean ignorePassed, boolean ignoreMissingPrerequisites,
                                     int pageSize, int pageIndex);
```

`searchCourse`中有许多`@Nullable`参数，是这个方法设计的难点。由于数据库中`null`的特性，显然直接用等号来判断是十分不正确的，但是我们可以通过在sql语句中使用`where ? is null or  col = ?`来较为容易地规避掉不确定是否为null的情况。部分sql语句如左图，部分对应的参数设置如右图。

```
left outer join select_course as sc on sc.section_id = sec.id" +
where (cou.id like ('%' || ? || '%') or ? is null)" +
  and (cou.name || '[' || sec.name || ']' like '%' || ? || '%' or ? is null)" +
  and (? is null or i.firstname || i.lastname like ('%' || ? || '%')" +
    or (i.firstname || ' ' || i.lastname like ('%' || ? || '%'))" +
    or i.firstname like ('%' || ? || '%') or i.lastname like ('%' || ? || '%'))"
  and (? is null or cla.dayofweek = ?)" +
  and (? is null or ? between cla.classbegin and cla.classend)" +
  and (sec.semester_id = ?)" +
  and (? is null or cla.location like ('%' || ? || '%'))" +
```

```
//@Nullable String searchCid
if (searchCid == null) {
    stmt.setNull( parameterIndex: 1, Types.VARCHAR);
    stmt.setNull( parameterIndex: 2, Types.VARCHAR);
} else {
stmt.setString( parameterIndex: 1, searchCid);
stmt.setString( parameterIndex: 2, searchCid);
}

//@Nullable String searchName
if (searchName == null) {
    stmt.setNull( parameterIndex: 3, Types.VARCHAR);
    stmt.setNull( parameterIndex: 4, Types.VARCHAR);
} else {
stmt.setString( parameterIndex: 3, searchName);
stmt.setString( parameterIndex: 4, searchName);
}
```

由于返回值是`List<CourseSearchEntry>`，在查看了注释后发现，该类有四个属性（Course，CourseSection，Set< CourseSectionClass >，List< String >）（最后的`List<String> conflictCourseNames`未在截图里面出现），前三个的关系是一个course对应一个section，一个section对应多个class（用`Set`是为了去重，防止多个相同的class在一个CourseSearchEntry中）。

```java
/**
 * The course of the searched section
 */
public Course course;

/**
 * The searched course section
 */
public CourseSection section;

/**
 * All classes of the section
 */
public Set<CourseSectionClass> sectionClasses;
```

于是，我们将处理好的Course和CourseSection对象合为一个CouToSec对象，并重写它的`equals()`和`hashCode()`方法，便于与后续class在`Map`中处理。由于可能会出现学生选课后使得CourseSection的leftCapacity不一样（但实际上仍被视为同一个CourseSection），导致`Map`中出现映射错误，于是在重写的`hasCode()`方法中只加入Course、CourseSection的id、CourseSection的name。

```java
public static class CouToSec {
    Course cou;
    CourseSection sec;

    public CouToSec(Course cou, CourseSection sec) {
        this.cou = cou;
        this.sec = sec;
    }

    @Override
    public boolean equals(Object o) {
        if (this == o) return true;
        if (o == null || this.getClass() != o.getClass()) return false;
        CouToSec ano = (CouToSec) o;
        return Objects.equals(cou, ano.cou) && Objects.equals(sec, ano.sec);
    }

    @Override
    public int hashCode() { return Objects.hash(cou, sec.id, sec.name); }
}
```

为了应对参数中的四个`boolean`以及`CourseType`，设计了一个类来存储所有性质。并且在处理后可通过`(sp.Full && ignoreFull)||(sp.Passed && ignorePassed)||(sp.Conflict && ignoreConflict)||(sp.MissingPrerequisites && ignoreMissingPrerequisites)`来一次性解决全部`boolean`，然后用`List<String> courseIdList = getCourseType(con, studentId, searchCourseType)`和`courseIdList.contains(cse.course.id)`来检查`CourseType`是否符合要求。

```java
public static class searchProperty {
    boolean Full = false;
    boolean Conflict = false;
    boolean Passed = false;
    boolean MissingPrerequisites = false;
}
```

```java
//处理ignoreFull, ignoreConflict, ignorePassed, ignoreMissingPrerequisites
Map<CourseSearchEntry, searchProperty> propertyMap = new HashMap<>();
for (CourseSearchEntry cse : tmp1) {
    searchProperty sp = new searchProperty();
    sp.Full = isClassFull(cse.section.id);
    sp.Conflict = isCourseConflict(studentId, cse.section.id);
    sp.Passed = isCoursePassed(studentId, cse.section.id);
    sp.MissingPrerequisites = !passedPrerequisitesForCourse(studentId, cse.course.id);
    sp.courseType = getCourseType(studentId, cse.course.id);
    propertyMap.put(cse, sp);
}
List<CourseSearchEntry> tmp2 = new ArrayList<>();
for (Map.Entry<CourseSearchEntry, searchProperty> entry : propertyMap.entrySet()) {
    CourseSearchEntry cse = entry.getKey();
    searchProperty sp = entry.getValue();
    //直接跳过
    if ((sp.Full && ignoreFull) || (sp.Passed && ignorePassed)
            || (sp.Conflict && ignoreConflict) || (sp.MissingPrerequisites && ignoreMissingPrerequisites))
        continue;
    //检查CourseType
    if (searchCourseType == CourseType.ALL || sp.courseType == searchCourseType) tmp2.add(cse);
}
```

courseType：

根据输入的`studentId`找到该学生所在的`major`。根据此`major`，在`major_course`中根据输入的参数`searchCourseType`，筛选得到相应的`courseId`。

- `ALL`：不做筛选
- `MAJOR_COMPULSORY`：筛选`major_course`中类型为`C`的`courseId`
- `MAJOR_ELECTIVE`：筛选`major_course`中类型为`E`的`courseId`
- `CROSS_MAJOR`：筛选`major_course`中不属于学生对应`major`的`courseId`
- `PUBLIC`：筛选出不满足上诉条件的`courseId`

```java
    //通过searchCourseType来找找到全部符合条件的courseId
    public List<String> getCourseType(Connection con, int studentId, CourseType
searchCourseType) {
        List<String> courseId = new ArrayList<>();
        if (searchCourseType.equals(CourseType.ALL)) {
            return courseId;
        }
        String sql;
        boolean che = false;
        if (searchCourseType.equals(CourseType.MAJOR_COMPULSORY)) {
            sql = "select course_id from major_course where type = 'C' and
major_id = ?; ";
        } else if (searchCourseType.equals(CourseType.MAJOR_ELECTIVE)) {
            sql = "select course_id from major_course where type = 'E' and
major_id = ?; ";
        } else if (searchCourseType.equals(CourseType.CROSS_MAJOR)) {
            sql = "select course_id from major_course where major_id!= ?; ";
        } else {//!PUBLIC -> ALL MAJOR
            sql = "select distinct course_id from major_course; ";
            che = true;
        }
        try (PreparedStatement stmt = con.prepareStatement(sql);
             PreparedStatement stmtMajor = con.prepareStatement("select major_id
from students where id = ? ")) {
```

```
            if (che) {
                ResultSet rs = stmt.executeQuery();
                while (rs.next()) {
                    courseId.add(rs.getString(1));
                }
                rs.close();
                return courseId;
            }
            stmtMajor.setInt(1, studentId);
            ResultSet ma = stmtMajor.executeQuery();
            if (ma.next()) {
                int majorId = ma.getInt(1);
                stmt.setInt(1, majorId);
                ResultSet rs = stmt.executeQuery();
                while (rs.next()) {
                    courseId.add(rs.getString(1));
                }
                rs.close();
                ma.close();
                return courseId;
            }
        } catch (Exception e) {
            e.printStackTrace();
            throw new IntegrityViolationException();
        }
        return courseId;
    }
```

6. enrollCourse 实现

在`enrollCourse()`中，同样设计了一些辅助方法。

类名/方法名	作用
`public EnrollResult firstTwoResult(int studentId, int sectionId)`	用于判断`COURSE_NOT_FOUND`和`ALREADY_ENROLLED`
`public boolean isCoursePassed(int studentId, int sectionId)`	用于判断`ALREADY_PASSED`
`public boolean isCourseConflict(int studentId, int sectionId)`	用于判断`COURSE_CONFLICT_FOUND`
`public boolean isClassFull(int sectionId)`	用于判断`COURSE_IS_FULL`

另外，对`PREREQUISITE_NOT_FULFILLED`则调用了本类中的另一个方法`passedPrerequisitesForCourse()`。而`SUCCESS`和`UNKNOWN_ERROR`则在前面所有判断都不符合时，通过一个简单的sql语句和`executeUpdate()`来判断。

```
//SUCCESS UNKNOWN_ERROR
stmt = con.prepareStatement( sql: "insert into select_course(student_id,section_id) values (?,?)");
stmt.setInt( parameterIndex: 1, studentId);
stmt.setInt( parameterIndex: 2, sectionId);
int cnt = stmt.executeUpdate();
if (cnt == 1) return EnrollResult.SUCCESS;
else return EnrollResult.UNKNOWN_ERROR;
```

在`getCourseTable()`中，比较难的是怎么能获得date的所在周。通过查看semester.json文件可以发现，每个学期的begin_date刚好是该学期第一周的周一，于是我们可以通过`(? - sm.begin_date) / 7 + 1`来获得date的所在周（? 填入date）。

源码如下。

```
@Override
    public CourseTable getCourseTable(int studentId, Date date) {
        try (Connection con = SQLDataSource.getInstance().getSQLConnection()) {
            String sql = "select sm.id                                as smid,"
+
                    "       (? - sm.begin_date) / 7 + 1          as smweek," +
                    "       cou.name                             as coursename,"
+
                    "       i.id                                 as iid," +
                    "       getfullname(i.firstname, i.lastname) as iname," +
                    "       cla.classbegin," +
                    "       cla.classend," +
                    "       cla.location," +
                    "       cla.dayofweek," +
                    "       sec.name                             as sectionname"
+
                    "from semesters as sm" +
                    "         inner join sections sec on sec.semester_id =
sm.id" +
                    "         inner join classes cla on sec.id = cla.section_id
and cla.dayofweek = (? - sm.begin_date) / 7 + 1" +
                    "         inner join courses cou on cou.id = sec.course_id"
+
                    "         inner join class_instructor ci on ci.class_id =
cla.id" +
                    "         inner join instructors i on i.id =
ci.instructor_id" +
                    "         inner join" +
                    "     ((select sc.section_id from select_course sc where
sc.student_id = ?)" +
                    "      union all" +
                    "      (select sgp.section_id from student_grades_pf sgp
where sgp.student_id = ?)" +
                    "      union all" +
                    "      (select sgh.section_id from student_grades_hundred
sgh where sgh.student_id = ?)) scs" +
                    "     on scs.section_id = sec.id" +
                    "where ? between sm.begin_date and sm.end_date";

            PreparedStatement stmt = con.prepareStatement(sql);
            stmt.setDate(1, date);
            stmt.setDate(2, date);
```

```
            stmt.setInt(3, studentId);
            stmt.setInt(4, studentId);
            stmt.setInt(5, studentId);
            stmt.setDate(6, date);

            ResultSet rs = stmt.executeQuery();

            CourseTable courseTable = new CourseTable();
            courseTable.table = new HashMap<>();
            for (DayOfWeek d : DayOfWeek.values()) {
                courseTable.table.put(d, new HashSet<>());
            }

            while (rs.next()) {
                CourseTable.CourseTableEntry entry = new
CourseTable.CourseTableEntry();
                entry.courseFullName = String.format("%s[%s]", rs.getString(3),
rs.getString(10));

                Instructor instructor = new Instructor();
                instructor.id = rs.getInt(4);
                instructor.fullName = rs.getString(5);
                entry.instructor = instructor;

                entry.classBegin = rs.getShort(6);
                entry.classEnd = rs.getShort(7);
                entry.location = rs.getString(8);

                DayOfWeek day = DayOfWeek.of(rs.getInt(9));
                courseTable.table.get(day).add(entry);
            }
            return courseTable;
        } catch (Exception e) {
            e.printStackTrace();
            throw new IntegrityViolationException();
        }
    }
```

7. dropCourse

dropCourse的目的是将指定学生指定section的选课删除，如果该同学这门课已经有了成绩，就不做处理，抛出IllegalStateException。我们小组一开始的设计是：
先在student_grades_hundred和student_grades_pf两表中进行对指定学生和section的查询加入result set中有结果，就说明这门课该同学有成绩需要抛出IllegalStateException，如果没有结果就正常进行在select_course的删除操作

```
    PreparedStatement stmt1 = con.prepareStatement(
            sql: "select grade from (select grade::varchar from student_grades_pf where student_id = ? and section_id = ? " +
                    "      union all " +
                    "      select grade::varchar from student_grades_hundred where student_id = ? and section_id = ?) sg; ");
    stmt1.setInt( parameterIndex: 1, studentId);
    stmt1.setInt( parameterIndex: 2, sectionId);
    stmt1.setInt( parameterIndex: 3, studentId);
    stmt1.setInt( parameterIndex: 4, sectionId);

    ResultSet rs = stmt1.executeQuery();
    if (rs.next() && rs.getString( columnIndex: 1) != null) {
        throw new IllegalStateException();
    }
    rs.close();

    PreparedStatement stmt2 = con.prepareStatement( sql: "delete from select_course where student_id = ? and section_id = ?");
    stmt2.setInt( parameterIndex: 1, studentId);
    stmt2.setInt( parameterIndex: 2, sectionId);
    stmt2.executeUpdate();
    stmt2.close();
} catch (SQLException e) {
    e.printStackTrace();
    throw new IntegrityViolationException();
}
```

结果如下图：

```
Test drop course: 377124
Test drop course time: 2611.23s
```

显然太过笨重，一次处理需要花费将近一个小时的时间。而在几番讨论以后，我们依然没有得出在保留原有思路的基础上能提升性能的算法。不管哪一种都摆脱不了每次调用dropCourse方法，都需要与数据库进行两次交互的结构：一次做筛选 一次做delete操作。

最后我们决定放弃部分准确性追求效率，不再对输入的数据做筛选直接放到sql中运行

```
//不考虑该学生已有分数的课程
PreparedStatement stmt=con.prepareStatement( sql: "delete from select_course where student_id=? and section_id=?")

stmt.setInt( parameterIndex: 1,studentId);
stmt.setInt( parameterIndex: 2,sectionId);

int state=stmt.executeUpdate();

if(state<=0){
    throw new IllegalStateException();
}
```

结果如下图：

```
Test drop course: 416061
Test drop course time: 3.98s
```

结果发现不仅运行时间大幅缩短，而且正确率也有显著提升，此次对比实验优化率高达72383.71%