Database Project 3

School Information System in JDBC

**Team 4 : Evgenii, 윤진주, 정현호**

# 1 Log in process and Incorrect authentication scenario

## 1.1 Solving Process

### 1.1.1 Observing the problem

Our objective is to make a program for log in of users. Where the system is able to distinguish if the user is a student or instructor.

In order to achieve the goal, these kinds of information are needed.

1. Database of all instructors
2. Database of all students

### 1.1.2 Determining how to solve the problem

At first, we get id and name of the user as an input for our function. Then we compare the input with the available databases and retrieve a result based on it. We use **Prepared Statements** to avoid SQL injection security risk in both methods for authentication for student and instructor. Then we use **next** method on the result set to test whether or not there remains at least one unfetched tuple in the result set and if so, fetches it. After we use **getString** method to retrieve attributes from the fetched tuple.

And then we compare the retrieved result with the given input. If they match we can authenticate the user whether he/she is a student or instructor. However if the match is not found we give a message “Wrong authentication” and then by using recursion repeat process of log in again.

## 1.2 Implementation

### 1.1.1 Function prototype

**public int isStudent(int stud\_id, String stud\_name)**

The function **isStudent**() gets stud\_id(student’s id) and stud\_name(student’s name)as a parameters. It returns 1 if student is found, and 0 if student is not found.

**public int isInstructor(int inst\_id, String inst\_name)**

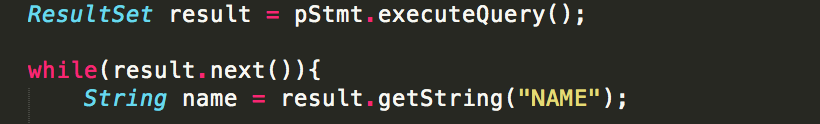
The function **isInsructor**() gets inst\_id(instructor’s id) and inst\_name(instructor’s name)as a parameters. It returns 1 if student is found, and 0 if student is not found

### 1.1.2 SQL statements

#### 1.1.1.1 preparedStatement



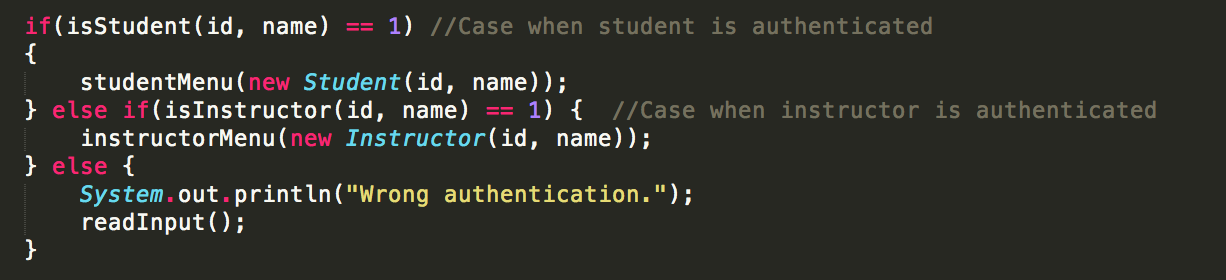
This statement is used to avoid SQL injection that can be used by malicious hackers to steal data or damage the database. So it is preferable to create SQL strings directly using user input, instead prepared statements should be used to bind user inputs.



This statement retrieves information we need from the database to compare with the input. **next** method on the result set is used to test whether or not there remains at least one unfetched tuple in the result set and if so, fetches it. Then we use **getString** method to retrieve needed attributes from the fetched tuple and compare it with the input.

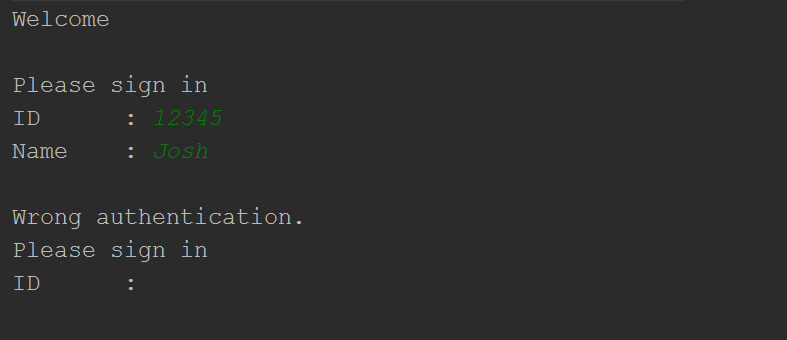
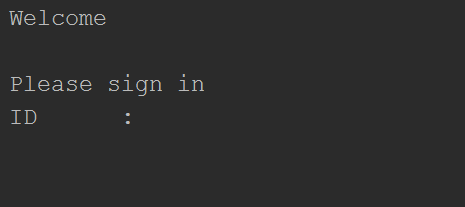
The same process is done for authentication of student.

In case if no match will not be found for student and instructor this means that input name or id is not in the database. So the function will print, “Wrong authentication”, and ask to input id and name again.



## 1.2 Result

Input sequence is simplified for reducing redundant space in output console.



# 2 Student menu

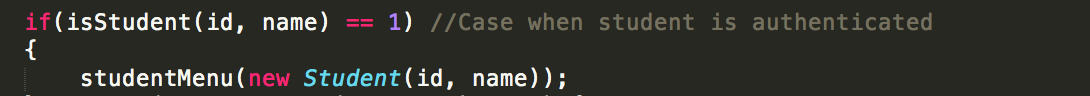
## 2.2 Solving Process

### 2.1.2 Observing the problem

After user is authenticated as a student, he/she should be given a menu where he/she is given with three choices. Among them he/she can choose to view student report or timetable of the courses he or she takes.

### 2.1.2 Determining how to solve the problem

This process can be accomplished with the help of the method isStudent() from Log in process that will help us to authenticate a student.



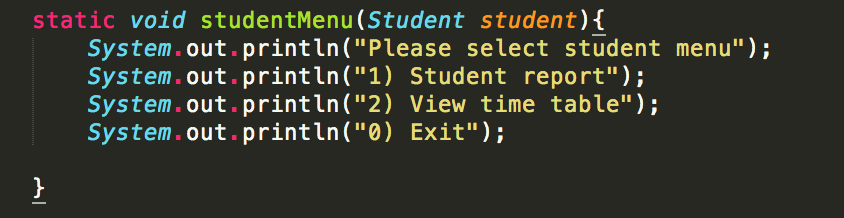
And then we call method that we wrote for student menu.

## 2.2 Implementation

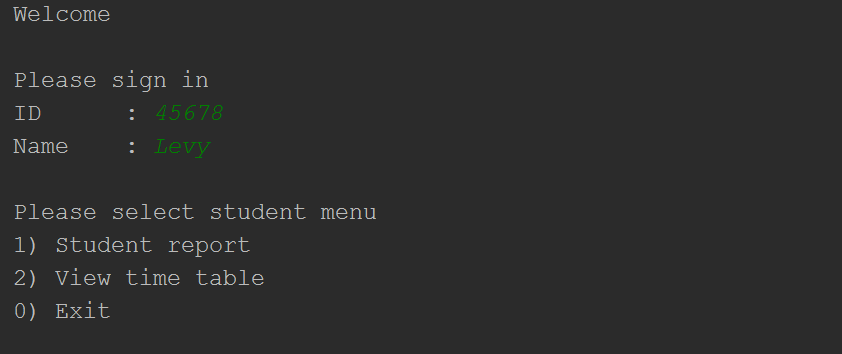
### 2.1.2 Function prototype

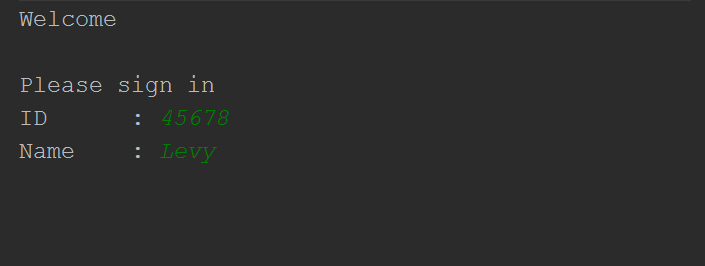
**static** **void** studentMenu(**Student** student)

The function studentMenu() prints student menu when it is called.



## 2.2 Result

Input sequence is simplified for reducing redundant space in output console.



# 3. Student Report

## Solving Process

### Observing the problem

1. Student의 정보(name, dept\_name, tot\_credits)를 출력
2. Student의 학기별 GPA를 계산하고, 학기별 들은 과목의 정보(course\_id, title, dept\_name, credits, grade) 출력
3. GPA 계산을 할 경우, 성적에 null이 입력된 과목이 있는 학기의 GPA는 출력하지 말 것
4. Reverse time 학기순으로 출력할 것

### Determining how to solve the problem

Student의 정보는 studentID가 student table의 superkey 이므로 student table로부터 추출하여 row가 1개인 result set을 만들어서 출력한다.

Student의 학기별 GPA를 계산하기 위해 group by (year, semester)와 sum() 그리고 문자로 주어진 성적을 숫자로 변환하기 위해 case문을 활용하여 계산한다. GPA 계산을 위해서는 credits도 필요하므로 takes와 course를 natural join한다.

GPA를 출력할 때, 성적에 null이 있는 경우에는 GPA : 로 출력해야 하는 데, 위의 GPA의 계산법으로는 학기에 들은 과목의 모든 성적이 null일 경우에만 GPA 가 null로 구해지고 아닐 경우에는 null 경우에는 제외하고 계산이 된다. 그러므로 계산된 GPA의 값으로는 과목 성적을 null을 받았는지 F를 받았는지를 구분하지 못한다. 그래서 count를 활용하여 null이 있는 경우를 판별한다. GPA를 계산하기 위한 table을 만들때, Count(course\_id)를 통해 학기당 들은 과목수를 세고(course\_id는 course의 super key이므로 null이 될 수 없다.), count(grade)를 한 값과 비교하면 grade의 null이 있는 경우, null을 빼고 count 되므로 count(grade)와 count(course\_id)의 값은 다르기 때문에 grade에 null이 있음을 알 수 있다.

마지막으로, (year, semester)을 reverse time으로 정렬하기 위해서는 order by 문과 case 문이 활용 된다.

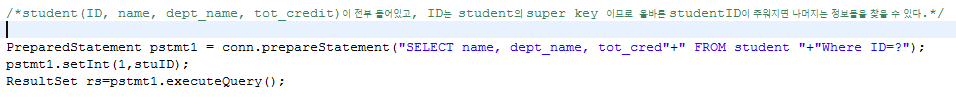
## Implementation

### Function prototype

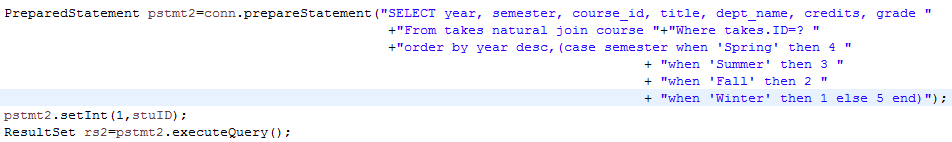


* stuID만을 인수로 가진다.

#### rs: student의 기본 정보

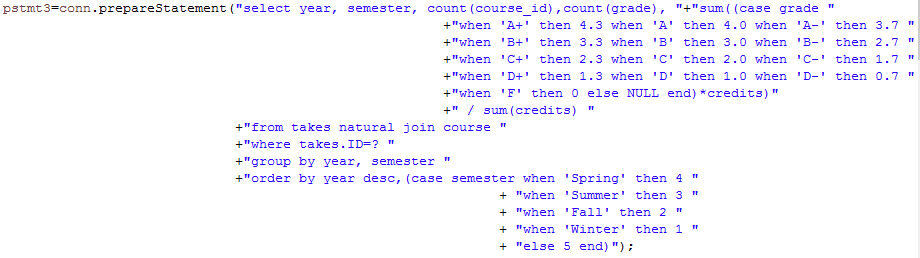


#### 3.2.3 rs2 : stuID가 들은 과목 정보를 reverse time 으로 정렬



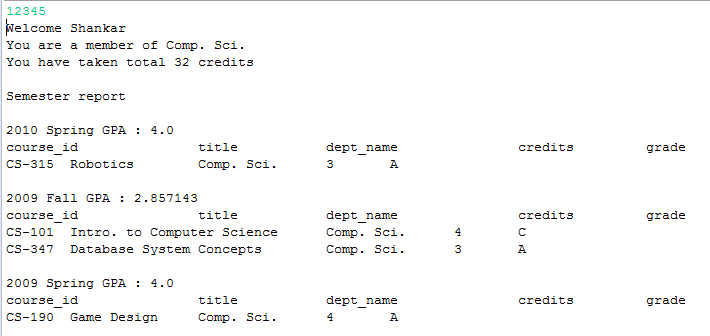
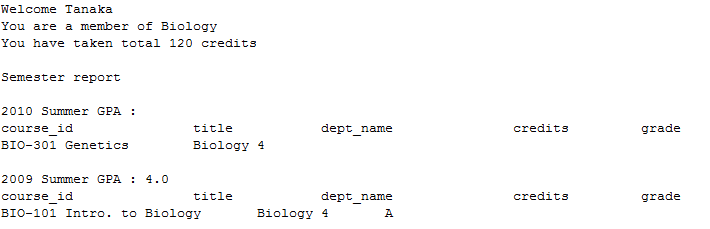
조사하고 싶은 student의 id가 주어졌을 때, 그 id를 활용하여 takes natural join course 로부터 (year, semester, course\_id, title, dept\_name, credits, grade)를 추출하고 이를 ordery by 문을 활용해 reverse time 으로 정렬한다. Reverse time으로 정렬하기 위해 year는 내림차순으로, semester는 winter, fall, summer, spring 순으로 정렬해야 한다. 그런데 이는 일반적인 string 정렬 방식이 아니므로 case 문을 활용하여 우선순위를 줘서 정렬을 시킨다.

#### rs3 : student가 학기당 들은 과목수와 GPA



Student의 학기당 GPA를 계산하기 위해 group by 를 활용하여 (year, semester)로 data를 묶어 주고, case문을 활용하여 문자인 grade를 숫자에 대응시켜 sum과 / 를 이용하여 GPA를 구한다. 그리고 grade에 null이 있는지 없는지를 판별 하기 위한 count( course\_id), count(grade) 도 구해준다. 마지막으로 order by 문을 활용해서 reverse time으로 정렬시키는 sql이고, 그 결과를 가지고 잇는 result set rs3 이다.

## Result

대표적인 몇 학생의 결과  


# View Time Table

## Solving Process

### Observing the problem

1. 학생이 들은 과목이 있는 (year, semester) 을 조사하고 출력
2. 선택된 (year, semester)의 과목의 자세한 정보(course\_id, title, day, start\_time, end\_time) 을 출력
3. 단, (year, semester)는 reverse time으로 정렬

### Determining how to solve the problem

학생이 들은 과목이 있는 (year, semester)는 각각의 학생마다 다르므로 학생이 log\_in 을 하면 distinct문을 활용해서 조사를 하고 출력한다.

선택된 (year, semester)에 열린 과목의 자세한 정보를 출력하기 위해서는 takes(ID), section(time\_slot\_id), course(course\_id, title), time\_slot (day, start\_time, end\_time)이 필요하고, sql을 활용해서 하려면 4개의 table을 natural join 해야 하는데, 이를 하게 되면 굉장히 redudent가 심해지고 data가 굉장히 많고 크다면 용량의 문제도 생길 수 있으므로 이를 나눠서 처리하려고 한다.

1. Takes와 section을 natural join이 하여 학생이 선택된 (year, semester)에 들은 과목의 course\_id, time\_slot\_id 를 조사한다.
2. Course\_id 는 course의 super key 이고, time slot id 는 time slot 의 super key 이므로 간단한 sql 문으로 필요한 정보들을 구해준다.
3. while문을 활용하여 결과들을 합친다.

이렇게 하게 되면 result set을 여러 번 구해야 하는 것이 단점이다.

## Implementation

### Function prototype



viewTimeTable은 Student Report와 마찬가지로 stuID만 인수로 갖는다.

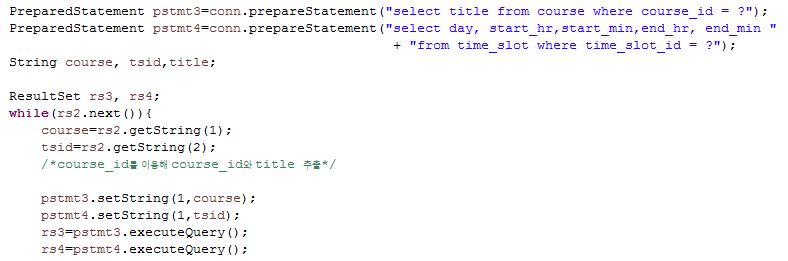
### Rs1 : distinct (year, semester) table

Distinct 구문을 활용하여 (year, semester)를 구하고, order by 구문으로 reverse time으로 정렬한다.

### Rs2 : (course\_id, time\_slot\_id) table

Student.ID와 year와 semester가 주어졌을 때, select 구문으로 course\_id, time\_slot\_id를 takes natural join section에서 추출한 Result set.

### Rs3 : (course\_id,title) in course, Rs4 : (day, start\_hr, start\_min, end\_hr,end\_min)

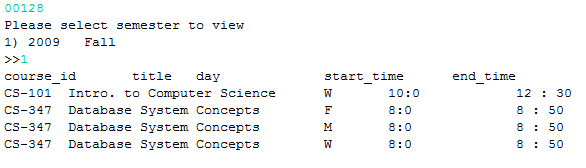


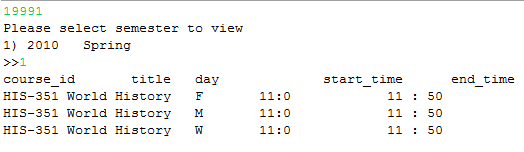
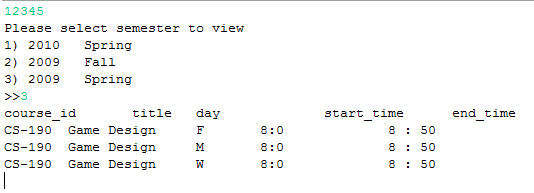
Rs2 의 (course\_id, time\_slot\_id) 로 rs3, rs4를 만든다

### PreparedStatement

정해진 sql 문에 (course\_id, time\_slot\_id)만 바꾸어 넣어 여러 번 sql을 실행시키므로 preparedStatement를 활요하는 것이 더 좋다.

## Result

대표적인 예시 몇 개   




# 5 Instructor menu

## 5.2 Solving Process

### 5.1.2 Observing the problem

After user is authenticated as a instructor, he should be given a menu where he is given with three choices. Among them he can choose to view course report or advisee report.

### 5.1.2 Determining how to solve the problem

This process can be accomplished with the help of the method isInstructor() from Log in process that will help us to authenticate a student.



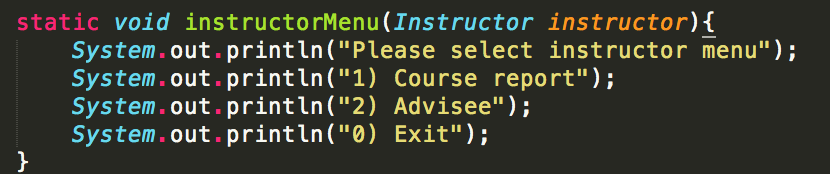
And then we call method that we wrote for instructor menu.

## 5.2 Implementation

### 5.1.2 Function prototype

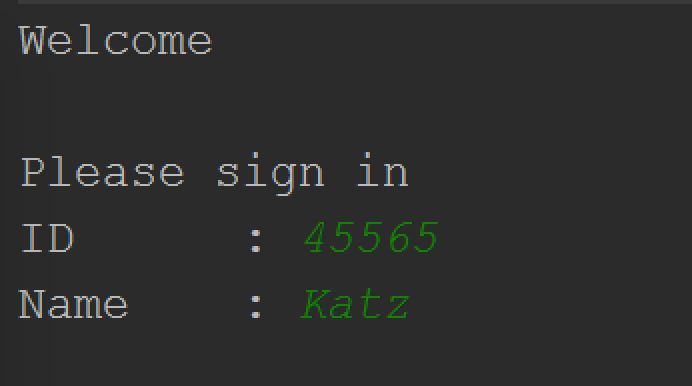
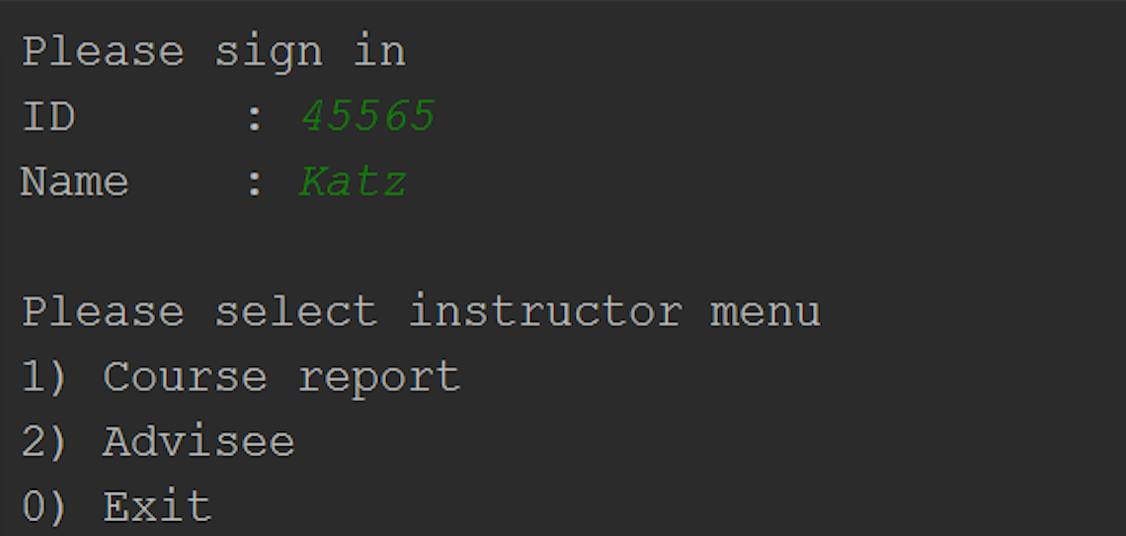
**static** **void** instructorMenu(**Instructor** instructor)

The function instructorMenu() prints instructor menu when it is called.



## 5.2 Result

Input sequence is simplified for reducing redundant space in output console.



# Course Report

## Solving Process

### Observing the problem

Our objective is to retrieve course information of the most recent semester taught by the instructor. In order to achieve the goal, these kinds of information are needed.

1. Basic information (year and semester) of the most recent semester
2. Courses opened in that most recent semester evaluated in 1
3. Additional information of those courses evaluated in 2
4. Information of students taking those courses evaluated in 2

### Determining how to solve the problem

First of all, getting the most recent semester of given instructor is needed. It can be done by using SQL statement with **ORDER BY** clause. Sorting by year is easy, but sorting by semester would be tough, because semesters are stored as VARCHAR, which would be sorted in lexicographical order as default.

Next, we have to retrieve a list of courses which is opened in the most recent semester. It can be done by applying **WITH** … **AS** clause to the statement of the first step.

After getting the list, basic information of courses in the list is needed, such as course\_id, sec\_id, title, building, and room\_number of them. It can be done by joining 3 tables: teaches, course, and section. The result of this step would be temporarily stored in application level, for further evaluation.

Time information of courses is also needed. We assume that start\_time and end\_time of one course is same, regardless of day of the week. For example, a course of time\_slot\_id A opens on Mondays, Wednesdays, and Fridays, but start\_time and end\_time are fixed to 8:00 to 8:50. So, all we have to do is retrieving days and time of the courses by querying to the table, which is made by joining 2 tables: section, and time\_slot. In this step, stored values of course\_id, sec\_id, semester, and year would be used.

Finally, information of students taking the courses is needed. It can be done by joining 2 tables: student and takes. In this step, stored values of course\_id, sec\_id, semester, and year would also be used.

## Implementation

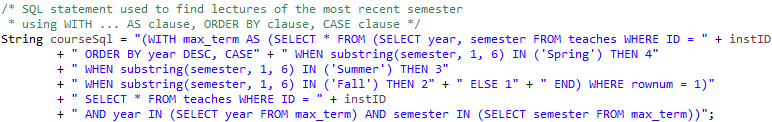
### Function prototype

**public** **static** **void** courseReport(**int** instID) **throws** Exception

The function courseReport() gets instID(instructor ID) as a parameter. It returns nothing, because its mission is just printing the results.

### SQL statements

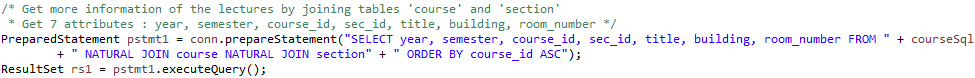
#### courseSql



This statement is used for getting courses which is opened in the most recent semester. **CASE** statement with substring() method is used to sort semester in chronological order, instead of lexicographical order. By matching integer value to each string representing seasons, semesters can be sorted in chronological order. We sort the courses by year DESC first, then sort by semester, and get the first row from the result. The first row will contain information of the most recent semester. Its return form would be look like this: (2010, Spring).

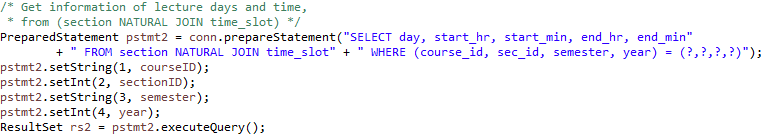
We store the basic information of the most recent semester temporarily, by using **WITH** … **AS** clause. And this information is used in the last part of the statement for retrieving coursed of the most recent semester. Its return form would be look like this: {(10101, CS-315, 1, Spring, 2010), …}.

#### rs1 : getting basic course information



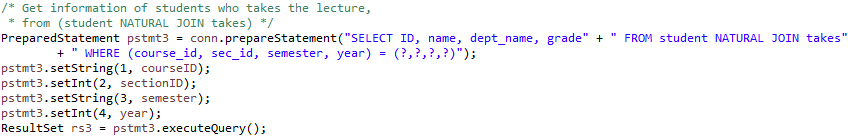
This statement retrieves basic information of the courses opened in the most recent semester. This information can be retrieved from (result of courseSql) **NATURAL JOIN** course **NATURAL JOIN** section. In fact, we do not need to retrieve information of year and semester for evaluation, but we need it for printing the phrase like ‘Course report – 2010 Spring’. Results of this query are stored in temporary storage in application level, which is shown later in 6.2.3.

#### rs2 : getting course time information



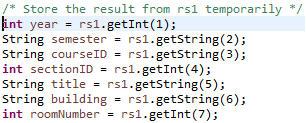
This statement gets course time information, including days, start time, and end time. This information can be retrieved from section **NATURAL JOIN** time\_slot. Temporary stored values from 6.2.2.2 are used here to form the SQL statement. Results of this query are also stored temporarily in application level, for printing results.

#### rs3 : getting students information



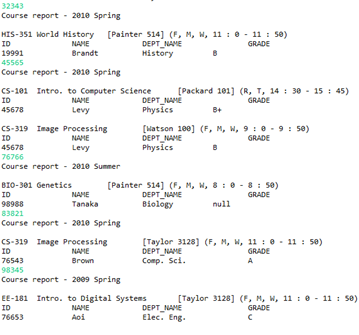
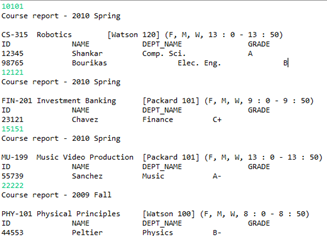
This statement is used to get information of students taking the courses. This information can be retrieved from student **NATURAL JOIN** takes. Temporary stored values from 6.2.2.2 are also used here to form the SQL statement.

### Temporary storage


These variables are used to store values temporarily. They are usually used for printing the result, but they are sometimes included in SQL query statements, which are shown above in 6.2.2.

## Result

Input sequence is simplified for reducing redundant space in output console.  


# Advisee Report

## Solving Process

### Observing the problem

Since we assume that there are no invalid inputs, there is no need to check the ID-name integrity of input values. Therefore, we don’t have to make use of table *instructor*, and use only two tables, *advisor* and *student*. Joining these two tables using attributes *advisor.s\_id* and *student.ID* will return students’ information along with IDs of advisors. Retrieving information of advisee students can be done by querying on the joined table using instructor’s ID given as an input.

### Determining how to solve the problem

This process can be done by a single, simple SQL statement. To be more specific, it can be done by

1. Joining two tables, *advisor* and *student* with attributes *advisor.s\_id* and *student.ID*  
   (**FROM** *advisor* *A* **JOIN** *student S* **ON** (*A.s\_id* = *S.ID*))
2. SELECT students from the joined table using instructor’s ID, which is given as an input.  
   (**SELECT** *S.ID, S.name, S.dept\_name, S.tot\_cred* **FROM** … **WHERE** *A.i\_id* = [instructor\_ID])

## Implementation

### Function prototype

**public** **static** **void** adviseeReport(**int** instID) **throws** Exception

The function adviseeReport() gets a parameter instID, which is an ID of instructor used in SQL statement. And it returns nothing, because its mission is just printing the results.

### SQL statement

As described in 7.1.2, a single SQL statement is used as shown below,

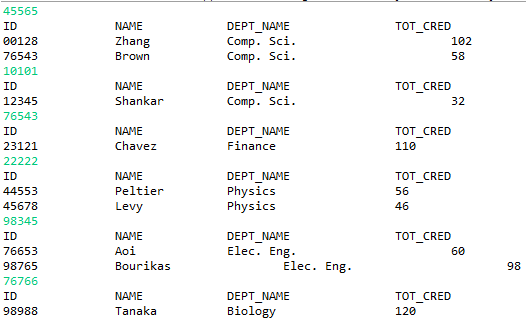
"SELECT S.ID, S.name, S.dept\_name, S.tot\_cred FROM advisor A JOIN student S ON (S.ID = A.s\_id) WHERE A.i\_id = ?"

and there is no additional manipulation after using this SQL statement.

### PreparedStatement

PreparedStatement object is used for executing SQL statement. PreparedStatement is a type of Statement which is more convenient and efficient than Statement when executing SQL statements. SQL statements given to PreparedStatement can be re-used by calling setter methods, such as setString() or setInt(). So it would be convenient when using the same statement with different parameters. Also, SQL statement is given to the PreparedStatement object at the time it is created, so the SQL statement can be precompiled. It results in speed-up in execution time. One more benefit of using PreparedStatement is that it can prevent malicious attack on databases, such as SQL injection. PreparedStatement automatically deals with this issue, and it makes the code not vulnerable to SQL injection.

## Result

Input sequence is simplified for reducing redundant space in output console.  


# Contribution

## Evgenii

* Implementation of 1. Log-in Process
* Implementation of 2. Student Menu
* Implementation of 5. Instructor Menu

## 윤진주

* Implementation of 3. Student Report
* Implementation of 4. View Timetable

## 정현호

* Implementation of 6. Course Report
* Implementation of 7. Advisee Report
* Code merge into one project
* Testing codes