Database Project 3

School Information System in JDBC

Team 4 : Evgenii, 윤진주, 정현호

6 Course Report

6.1 Solving Process

6.1.1 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

6.1.2 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.

6.2 Implementation

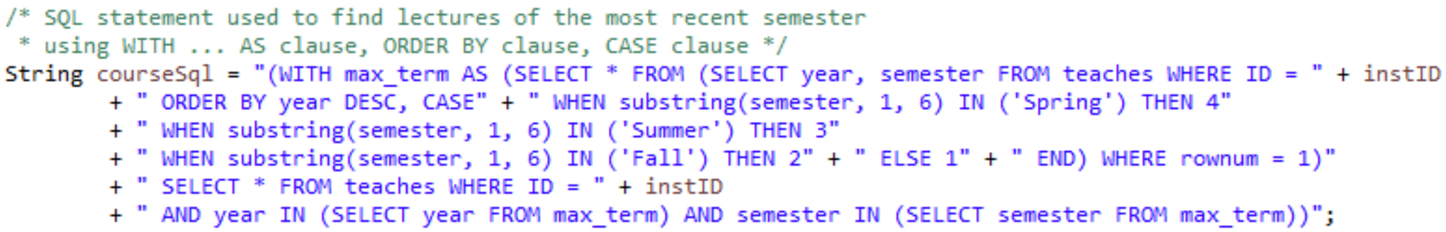
6.2.1 Function prototype

```
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.

6.2.2 SQL statements

6.2.2.1 courseSql

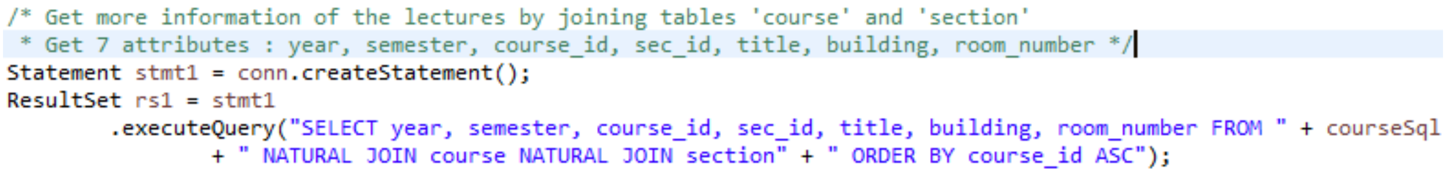


```
/* SQL statement used to find lectures of the most recent semester
 * using WITH ... AS clause, ORDER BY clause, CASE clause */
String courseSql = "(WITH max_term AS (SELECT * FROM (SELECT year, semester FROM teaches WHERE ID = " + instID
        + " ORDER BY year DESC, CASE" + " WHEN substring(semester, 1, 6) IN ('Spring') THEN 4"
        + " WHEN substring(semester, 1, 6) IN ('Summer') THEN 3"
        + " WHEN substring(semester, 1, 6) IN ('Fall') THEN 2" + " ELSE 1" + " END) WHERE rownum = 1)"
        + " SELECT * FROM teaches WHERE ID = " + instID
        + " AND year IN (SELECT year FROM max_term) AND semester IN (SELECT semester FROM max_term))";
```

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), …}.

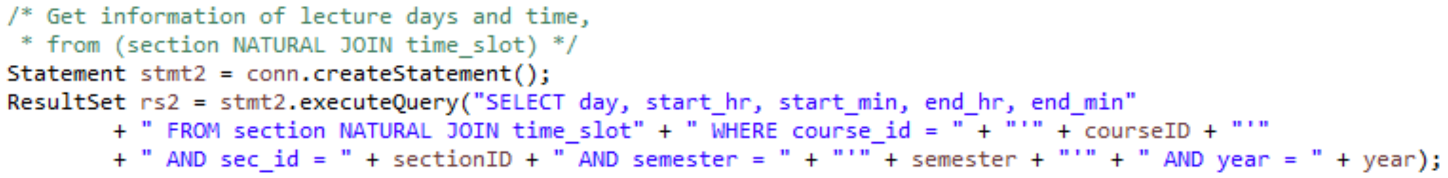
6.2.2.2 rs1 : getting basic course information



```
/* Get more information of the lectures by joining tables 'course' and 'section'
 * Get 7 attributes : year, semester, course_id, sec_id, title, building, room_number */
Statement stmt1 = conn.createStatement();
ResultSet rs1 = stmt1
        .executeQuery("SELECT year, semester, course_id, sec_id, title, building, room_number FROM " + courseSql
                + " NATURAL JOIN course NATURAL JOIN section" + " ORDER BY course_id ASC");
```

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.

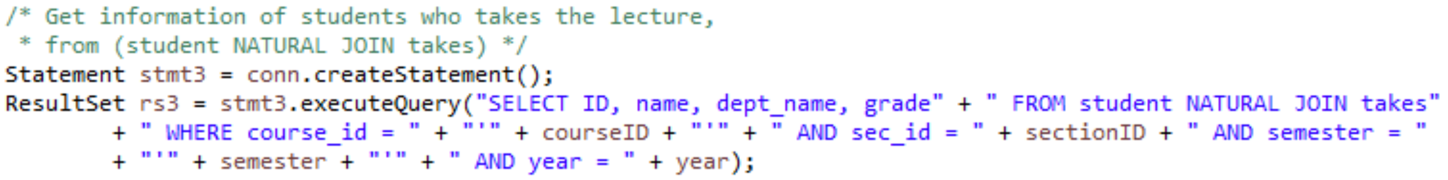
6.2.2.3 rs2 : getting course time information



```
/* Get information of lecture days and time,
 * from (section NATURAL JOIN time_slot) */
Statement stmt2 = conn.createStatement();
ResultSet rs2 = stmt2.executeQuery("SELECT day, start_hr, start_min, end_hr, end_min"
        + " FROM section NATURAL JOIN time_slot" + " WHERE course_id = " + "'" + courseID + "'"
        + " AND sec_id = " + sectionID + " AND semester = " + "'" + semester + "'" + " AND year = " + year);
```

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.

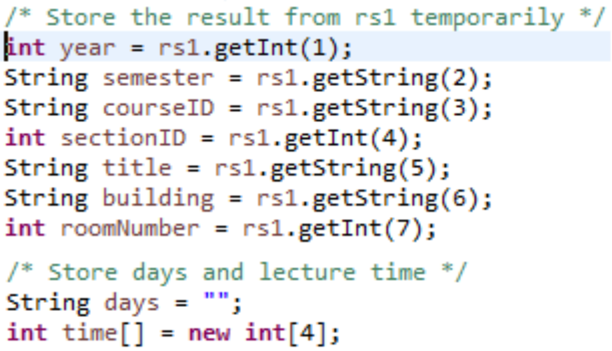
6.2.2.4 rs3 : getting students information



```
/* Get information of students who takes the lecture,
 * from (student NATURAL JOIN takes) */
Statement stmt3 = conn.createStatement();
ResultSet rs3 = stmt3.executeQuery("SELECT ID, name, dept_name, grade" + " FROM student NATURAL JOIN takes"
        + " WHERE course_id = " + "'" + courseID + "'" + " AND sec_id = " + sectionID + " AND semester = "
        + "'" + semester + "'" + " AND year = " + year);
```

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.

6.2.3 Temporary storage



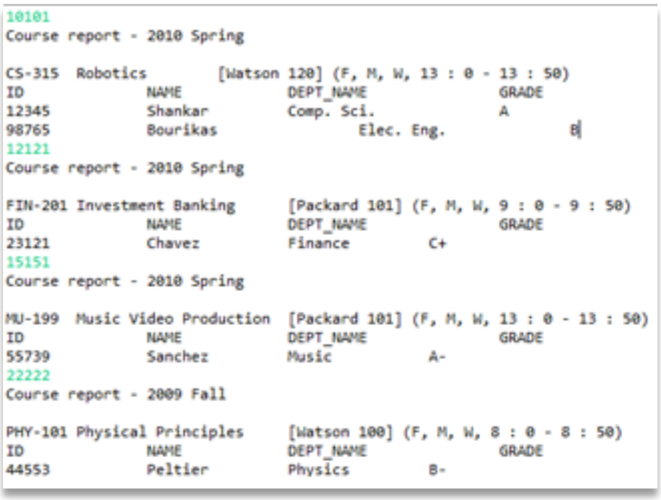
```
/* Store the result from rs1 temporarily */
int year = rs1.getInt(1);
String semester = rs1.getString(2);
String courseID = rs1.getString(3);
int sectionID = rs1.getInt(4);
String title = rs1.getString(5);
String building = rs1.getString(6);
int roomNumber = rs1.getInt(7);

/* Store days and lecture time */
String days = "";
int time[] = new int[4];
```

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.

6.3 Result

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



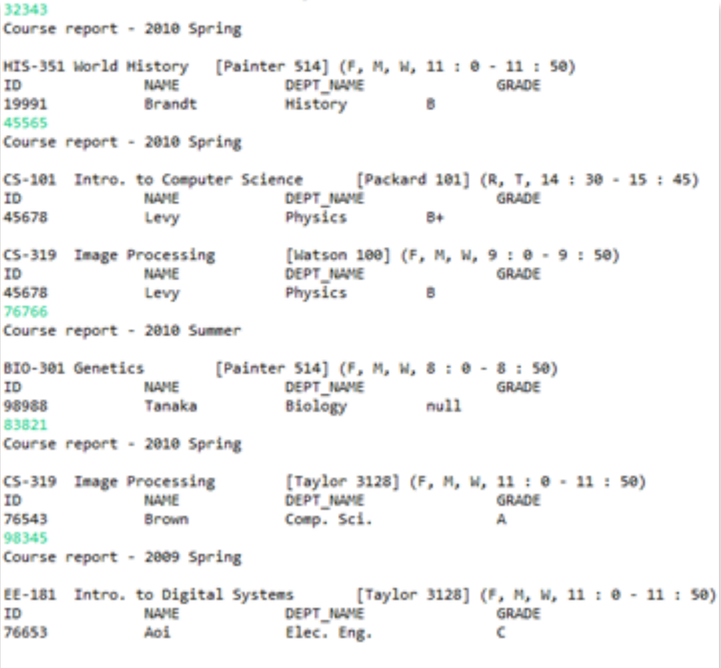
```
10101
Course report - 2010 Spring

CS-315  Robotics        [Watson 120] (F, M, W, 13 : 0 - 13 : 50)
ID              NAME            DEPT_NAME               GRADE
12345           Shankar         Comp. Sci.              A
98765           Bourikas                Elec. Eng.              B
12121
Course report - 2010 Spring

FIN-201 Investment Banking      [Packard 101] (F, M, W, 9 : 0 - 9 : 50)
ID              NAME            DEPT_NAME               GRADE
23121           Chavez          Finance         C+
15151
Course report - 2010 Spring

MU-199  Music Video Production  [Packard 101] (F, M, W, 13 : 0 - 13 : 50)
ID              NAME            DEPT_NAME               GRADE
55739           Sanchez         Music           A-
22222
Course report - 2009 Fall

PHY-101 Physical Principles     [Watson 100] (F, M, W, 8 : 0 - 8 : 50)
ID              NAME            DEPT_NAME               GRADE
44553           Peltier         Physics         B-
```



```
32343
Course report - 2010 Spring

HIS-351 World History   [Painter 514] (F, M, W, 11 : 0 - 11 : 50)
ID              NAME            DEPT_NAME               GRADE
19991           Brandt          History         B
45565
Course report - 2010 Spring

CS-101  Intro. to Computer Science      [Packard 101] (R, T, 14 : 30 - 15 : 45)
ID              NAME            DEPT_NAME               GRADE
45678           Levy            Physics         B+

CS-319  Image Processing        [Watson 100] (F, M, W, 9 : 0 - 9 : 50)
ID              NAME            DEPT_NAME               GRADE
45678           Levy            Physics         B
76766
Course report - 2010 Summer

BIO-301 Genetics        [Painter 514] (F, M, W, 8 : 0 - 8 : 50)
ID              NAME            DEPT_NAME               GRADE
98988           Tanaka          Biology         null
83821
Course report - 2010 Spring

CS-319  Image Processing        [Taylor 3128] (F, M, W, 11 : 0 - 11 : 50)
ID              NAME            DEPT_NAME               GRADE
76543           Brown           Comp. Sci.              A
98345
Course report - 2009 Spring

EE-181  Intro. to Digital Systems       [Taylor 3128] (F, M, W, 11 : 0 - 11 : 50)
ID              NAME            DEPT_NAME               GRADE
76653           Aoi             Elec. Eng.              C
```

7 Advisee Report

7.1 Solving Process

7.1.1 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.

7.1.2 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])

7.2 Implementation

7.2.1 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.

7.2.2 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 = ?"
```

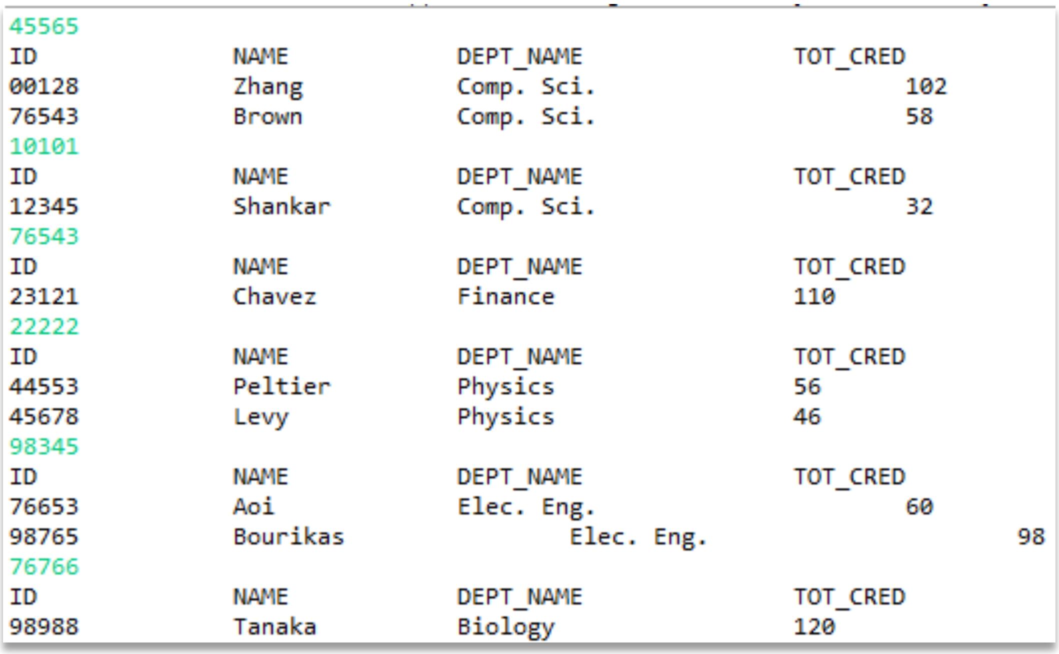
and there is no additional manipulation after using this SQL statement.

7.2.3 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.

7.3 Result

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



```
45565
ID              NAME            DEPT_NAME               TOT_CRED
00128           Zhang           Comp. Sci.                      102
76543           Brown           Comp. Sci.                      58
10101
ID              NAME            DEPT_NAME               TOT_CRED
12345           Shankar         Comp. Sci.                      32
76543
ID              NAME            DEPT_NAME               TOT_CRED
23121           Chavez          Finance                 110
22222
ID              NAME            DEPT_NAME               TOT_CRED
44553           Peltier         Physics                 56
45678           Levy            Physics                 46
98345
ID              NAME            DEPT_NAME               TOT_CRED
76653           Aoi             Elec. Eng.                      60
98765           Bourikas                Elec. Eng.                      98
76766
ID              NAME            DEPT_NAME               TOT_CRED
98988           Tanaka          Biology                 120
```