CS532 Project 2 – Using PL/SQL and JDBC to Implement Student Registration System

Due: December 5, 2016

This project is to use Oracle's PL/SQL and JDBC to create an application to support typical student registration tasks in a university. Students are strongly encouraged to form teams for this project and each team can have up to three members. Due to the time constraint, only a subset of the database tables and a subset of the needed functionalities will be implemented in this project.

1. Preparation

The following tables from the Student Registration System will be used in this project:

Students(B#, firstname, lastname, status, gpa, email, bdate, deptname)
Courses(dept_code, course#, title)
Course_credit(course#, credits)
Classes(classid, dept_code, course#, sect#, year, semester, limit, class_size)
Enrollments(B#, classid, lgrade)
Grades(lgrade, ngrade)
Prerequisites(dept_code, course#, pre_dept_code, pre_course#)

In addition, the following table is also required for this project:

Logs(logid, who, time, table_name, operation, key_value)

Each tuple in the logs table describes who (the login name of a database user) has performed what operation (insert, delete, update) on which table (give the table name) and which tuple (as indicated by the value of the primary key of the tuple) at what time. Attribute logid is the primary key of the table.

The schemas and constraints of the first six tables are the same as those used in Project 1 except that attribute faculty_B# in table Classes is removed (please make the corresponding change in the script used in Project 1). Please use the following statements to create the Prerequisites table and the Logs table:

```
create table prerequisites (dept_code varchar2(4) not null,
course# number(3) not null, pre_dept_code varchar2(4) not null,
pre_course# number(3) not null,
primary key (dept_code, course#, pre_dept_code, pre_course#),
foreign key (dept_code, course#) references courses on delete cascade,
foreign key (pre_dept_code, pre_course#) references courses on delete cascade);

create table logs (logid number(4) primary key, who varchar2(10) not null, time date not null,
table_name varchar2(12) not null, operation varchar2(6) not null, key_value varchar2(10));
```

You should populate these tables with appropriate tuples to test your programs.

2. PL/SQL Implementation (50 points)

You need to create a PL/SQL package for this application. All procedures and functions should be included in this package. Other Oracle objects such as sequences and triggers are to be created outside the package. The following requirements and functionalities need to be implemented.

1. (2 points) Use a sequence to generate the values for logid automatically when new log records are inserted into the logs table. Start the sequence with 1000.
2. (4 points) Write procedures in your package to display the tuples in each of the eight tables: students, courses, course_credit, prerequisites, classes, enrollments, grades, logs. As an example, you can write a procedure, say **show_students**, to display all students in the students table.
3. (4 points) Write a procedure in your package that, for a given student (with B# provided as a parameter), can list every class the student has taken or is taking (list all attributes except limit and class_size) as well as the letter grade and number grade the student received for the class (including null grades). If the classid is not in the Students table, report "The B# is invalid." If the student has not taken any course, report "The student has not taken any course."
4. (3 points) Write a procedure in your package that, for a given course (with dept_code and course# as parameters), can return all courses that need this course as a prerequisite (show dept_code and course# together as in CS532), including courses that need the course as prerequisite either directly and indirectly. If course C2 has course C1 as a prerequisite, C1 is a direct prerequisite of C2. If C3 has course C2 has a prerequisite, then C1 is an indirect prerequisite for C3. Please note that indirect prerequisites can be more than two levels away.
5. (3 points) Write a procedure in your package that, for a given class (with classid provided as a parameter), can list the classid and course title of the class as well as all the students (show B# and firstname) who took or are taking the class. If the class is not in the classes table, report "The classid is invalid." If no student took or is taking the class, report "No student has enrolled in the class."
6. (12 points) Write a procedure in your package to enroll a student into a class. The B# of the student and the classid of the class are provided as parameters. If the student is not in the students table, report "The B# is invalid." If the class is not in the classes table, report "The classid is invalid." If the enrollment of the student into a class would cause "class_size > limit", reject the enrollment and report "The class is full." If the student is already in the class, report "The student is already in the class." If the student is already enrolled in three other classes in the same semester and the same year, report "You are overloaded." and allow the student to be enrolled. If the student is already enrolled in four other classes in the same semester and the same year, report "Students cannot be enrolled in more than four classes in the same semester." and reject the enrollment. If the student has not completed the required prerequisite courses with at least a grade C, reject the enrollment and report "Prerequisite not satisfied." For all the other cases, the requested enrollment should be carried out successfully. You should make sure that all data are consistent after each enrollment. For example, after you successfully enrolled a student into a class, the class size of the class should be updated accordingly. Use trigger(s) to implement the updates of values caused by successfully enrolling a student into a class. It is recommended that all triggers for this project be implemented outside of the package.
7. (9 points) Write a procedure in your package to drop a student from a class (i.e., delete a tuple from the enrollments table). The B# of the student and the classid of the class are provided as parameters. If the student is not in the students table, report "The B# is invalid." If the classid is not in the classes table, report "The classid is invalid." If the student is not enrolled in the class, report "The student is not enrolled in the class." If dropping the student from the class would cause a violation of the prerequisite requirement for another class, then reject the drop attempt and report "The drop is not permitted because another class uses it as a prerequisite." In all the other cases, the student should be dropped from the class. If the class is the last class for the student, report "This student is

not enrolled in any classes." If the student is the last student in the class, report "The class now has no students." Again, you should make sure that all data are consistent after the drop and all updates caused by the drop should be implemented using trigger(s).

8. (5 points) Write a procedure in your package to delete a student from the students table based on a given B# (as a parameter). If the student is not in the students table, report "The B# is invalid." When a student is deleted, all tuples in the enrollments table involving the student should also be deleted (use a trigger to implement this) and this will trigger a number of actions as described in the above item (item 7).
9. (8 points) Write triggers to add tuples to the logs table automatically whenever a student is added to or deleted from the students table, or when a student is successfully enrolled into or dropped from a class (i.e., when a tuple is inserted into or deleted from the enrollments table). For a logs record for enrollments, the key value is the concatenation of the B# value, a comma, and the classid value.

3. Interface (35 points)

Implement an interactive and menu-driven interface in the bingsuns environment using Java and JDBC (see sample programs in the Project 2 folder in blackboard). More details are given below:

1. The basic requirement for the interface is a text-based menu-driven interface. You first display menu options for a user to select (e.g., 1 for displaying a table, 2 for enrolling a student into a class, …). An option may have sub-options depending on your need. Once a final option is selected, the interface may prompt the user to enter parameter values. As an example, for enrolling a student into a class, the parameter values include B# and classid. Then an operation corresponding to the selected option will be performed with appropriate message displayed.

2. **A nice GUI or Web interface will receive 5 bonus points.**

3. Your interface program should utilize as many of your PL/SQL code as possible. Some of the procedures may need to be rewritten in order for them to be used in your Java/JDBC program. In particular, in order to pass retrieved results from a PL/SQL block (e.g., show_students) to the Java/JDBC program, the block needs to be implemented as a ref cursor function (see the third sample program for an example).

4. Note that messages generated by the dbms_output package in your PL/SQL package or triggers will not be displayed on your monitor screen when you run your Java/JDBC application. You may need to regenerate these messages in your Java program or use dbms_output.get_line( ) to catch the messages generated by dbms_output.put_line( ).

4. Documentation (15 points)

Documentation consists of the following aspects:

1. Each procedure and function and every other object you create for your project needs to be explained clearly regarding its objective and usage.

2. Your code needs to be well documented with in-line comments.

3. *Team report*. If your team has more than one student, your team must submit a team report. This report should describe in detail how the team members collaborated for the project. More specifically, it needs to include the following information:
    a. Your meetings (describe when each meeting occurred and what was discussed for the project at the meeting?)
    b. What was your plan (schedule) for completing your project? How was the plan followed during the course of completing the project?
    c. Which team member is primarily responsible for which part of the project? Did the other member contribute to the task primarily assigned to the other member?
    d. What is the experience for each of you working as a team? What are the lessons (both positive and negative) learned?

4. *Personal report*. Each team member also needs to submit a separate personal report about your team activities from your perspective. This report can be as simple as a single sentence if you agree with the team report (just say that you agree with the team report). You can also use this report to detail your disagreement with the team report or simply provide additional information about the team activities beyond the team report. This report will be kept confidential by the instructor.

5. Hand-ins, Demo and Grading

1. Each team needs to hand in ONE hard copy of your entire PL/SQL code (including the package, triggers, and sequences). Only one copy for each team is needed.

2. Each team needs to submit all source code, including both PL/SQL and Java, to blackboard (Project 2 submission folder). Make sure to include information about the team members.

3. Each team a team report in hard copy.

4. Each student working in a team needs to submit a hard copy of your personal report.

5. Each team is required to demonstrate the completed project to the instructor using tuples created by the instructor. More instructions on demo will be provided before the demo.

6. The grading will be based on the quality of your code, the documentation and on how successful of your demo is.

All reports must be typed and printed and submitted to the instructor by the specified time.