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

Due: November 28, 2018

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**. Members of a team may come from both sessions of database systems. Each team needs to submit the team information (team members) to the following table: meng.proj2_teams18(membernames varchar2(70) not null, submission_date date) using "insert into meng.proj2_teams18 values (mnames, sysdate);", where mnames is a single string composed of the full names of all team members with members separated by semi-colon.

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#, first_name, last_name, status, gpa, email, bdate, deptname)
TAs(B#, ta_level, office)
Courses(dept_code, course#, title)
Classes(classid, dept_code, course#, sect#, year, semester, limit, class_size, room, TA_B#)
Enrollments(B#, classid, lgrade)
Prerequisites(dept_code, course#, pre_dept_code, pre_course#)

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

Logs(log#, op_name, op_time, table_name, operation, key_value)

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

The schemas and constraints of the first five tables are the same as those used in Project 1. Please use the following statements to create the Prerequisites table and the Logs table (you can add them to the script file):

```
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 (log# number(4) primary key, op_name varchar2(10) not null, op_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 log# automatically when new log records are inserted into the logs table. Start the sequence with 100 with an increment of 1.
2. (4 points) Write procedures in your package to display the tuples in each of the seven tables for this project. As an example, you can write a procedure, say **show_students**, to display all students in the students table.
3. (3 points) Write a procedure in your package that, for a given class (with classid provided as an in parameter), will list the B#, the first name and last name of the TA of the class. If the class does not have a TA, report "The class has no TA." If the provided classid is invalid (i.e., not in the Classes table), report "The classid is invalid."
4. (4 points) Write a procedure in your package that, for a given course (with dept_code and course# as parameters), can return all its prerequisite courses (show dept_code and course# together as in CS532), including both direct and indirect prerequisite courses. If course C1 has course C2 as a prerequisite, C2 is a direct prerequisite. If C2 has course C3 has a prerequisite, then C3 is an indirect prerequisite for C1. Please note that indirect prerequisites can be more than two levels away. If the provided (dept_code, course#) is invalid, report "dept_code || course# does not exist." – show dept_code and course# together as in CS532.
5. (14 points) Write a procedure in your package to enroll a student into a class (i.e., insert a tuple into the Enrollments table). The B# of the student and the classid of the class are provided as parameters (all new enrollments will have a null value for lgrade). 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 class is not offered in the current semester (i.e., Fall 2018), reject the enrollment and report "Cannot enroll into a class from a previous semester." If the class is already full before the enrollment request, reject the enrollment request and report "The class is already full." If the student is already in the class, report "The student is already in the class." If the student is already enrolled in four other classes in the same semester and the same year, report "The student will be overloaded with the new enrollment." but still allow the student to be enrolled. If the student is already enrolled in five other classes in the same semester and the same year, report "Students cannot be enrolled in more than five 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 need to 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 increased by 1. 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.)
6. (10 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 the class is not offered in Fall 2018, reject the drop attempt

and report "Only enrollment in the current semester can be dropped." If dropping the student from the class would cause a violation of the prerequisite requirement for another class, reject the drop attempt and report "The drop is not permitted because another class the student registered uses it as a prerequisite." In all the other cases, the student will 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 a successful enrollment drop and all updates caused by the drop need to be implemented using trigger(s).

7. (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). Note that such a deletion may trigger a number of actions as described in the above item (item 6).
8. (8 points) Write triggers to add tuples to the Logs table automatically whenever a student is 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 MyCourses). 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 from the terminal. 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 10 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 can either 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 reasonable 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 another member?

4. *Personal report*. Each team member also needs to submit a separate personal report about your team activities from your perspective. Specifically, the personal report should contain the following information (this report will be kept confidential by the instructor):
    a. Whether you agree or disagree with the team report.
    b. If you disagree with the team report, provide the reason(s) for the disagreement.
    c. Describe your experience working as a team. If there are lessons learned, both positive and negative, please mention them too.

5. Hand-ins, Demo and Grading

1. Each team submit the team report in hard copy. Make sure to include information about the team members on the first page. All members need to sign this page.

2. 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.

3. Each team needs to submit all source code, including both PL/SQL and Java, to MyCourses (Project 2 submission folder). A readme.txt file describing how to run your code needs to be included.

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.