Semester 1 Project Enhancements

Instructions for Students

The midterm project is divided into 5 parts. You can use the 5 parts in various ways depending on how much time you wish to devote to this project or on the technical level of your students.

- All 5 parts can be assigned to individual students. There are dependencies between the parts, so 1 must be completed before 2 and 3; 4 must be completed before 5.
- The class can be divided into teams and the teams can complete all 5 parts. There are dependencies between the parts, so 1 must be completed before 2 and 3; 4 must be completed before 5.

The code for two packages e*nrollments_package* and *admin_tools_package* must be created from the anonymous blocks your students wrote in the first MidTerm Project. If these files are lost, there were suggested solutions in the teacher document for Part I; you can use these solutions to re-create the programs. You are instructed to modify the existing programs from Part I, to incorporate the required topics for this project.

The students are also required to create a new package *manage_triggers_package* to disable/enable triggers for a table or to compile a specific trigger.

Project setup: The Data

This project will build on the case study called STUDENT ADMINISTRATION or SA. A set of database tables is used to manage schools" course offerings as delivered by instructors inmany classes over time. Information is stored about classes that are offered, the students who take classes, and the grades the students receive on various assessments. The school administrators can use the SA database to manage the class offerings and to assign instructors. Teachers can also use the SA database to track student performance. The database objects for this project are already in the students" accounts and they are asfollows:

Tables:

INSTRUCTORS SECTIONS COURSES CLASSES
ASSESSMENTS STUDENTS ENROLLMENTS
CLASS_ASSESSMENTSERROR_LOG
GRADE_CHANGESSequence:
ASSESSMENT_ID_SEQ

Part 1: Procedures, Functions and Packages

In this section the students start by re-writing the anonymous blocks from MidTerm I to become procedures and functions.

1. Find the file saved from called *enroll_student_in_class.sql* from MidTerm I. Convert this to a procedure and have it accept a STU_ID and CLASS_ID as input parameters. Use "today"s date" for the ENROLLMENT_DATE and the string „Enrolled" for the STATUS. Raise an exception if the accepted student is already enrolled in the accepted class. In your exception handler, display a message stating the student is already enrolled in the class.

2. Find the file called *drop_student_from_class.sql* from MidTerm I. Convert it to a procedure that accepts a STU_ID and CLASS_ID as input parameters. If the DELETE fails because the student is not in the class, raise a user_defined exception to display a message stating the student is not in the class.

3. Find the file called *student_class_list.sql* from MidTerm I. Rewrite it to be a procedure that displays all of the classes a student has been enrolled in within the most recent 10 years. For example: If you run your procedure on May 10, 2010, you should display all enrollments between May 10, 2000 and May 10, 2010. Accept the STU_ID as an input parameter. For each enrollment, display the ENROLLMENT_DATE, CLASS_ID and STATUS.

4. Find the file called *add_new_classes.sql* from MidTerm I. Rewrite it as a Procedure and have it accept the following IN parameters:
   a. number of new classes required. Set a default value of 1.
   b. Course id; For each new class, use "today"as the START_DATE.
   c. Period, to specify what days the class meets.
   d. Frequency, to specify how often it meets.
   e. Instructor id, to specify who is teaching the class(s).

5. Find the file called *course_roster.sql* from MidTerm I and rewrite it as a procedure. Accept the INSTR_ID and COURSE_ID as input parameters. For each ENROLLMENT, display: CLASS_ID, STATUS, Student FIRST_NAME and LAST_NAME.

6. Find the file called *convert_grade.sql* from MidTerm I and rewrite it to be a function Use an IN parameter to enter the number grade. RETURN a CHAR value. Use the following rules: A:90 or above, B: >=80 and<90 , C: >=70 and < 80, D: >=60 and < 70, F:<60.

7. Find the file called *student_count.sql* and rewrite it as a function that will RETURN the number of students in a particular class. Accept a CLASS_ID as an IN parameter.
8. Create a package called *enrollments_package* which will contain the procedures you created in A, B, and C. Make all procedures public. Comment your procedures to explain their purpose and functionality.

9. Find the program saved in the file *create_assignment.sql.* Rewrite it as a procedure that accepts the assignment description as an input parameter.

10. Find the file called *enter_student_grade.sql* and rewrite it as a procedure that a teacher can run to insert the student's grade on a particular assignment. Accept a NUMERIC_GRADE, CLASS_ASSESSMENT_ID, CLASS_ID, STU_ID and ASSESSMENT_ID as IN parameters. Use "today"s" date for the DATE_TURNED_IN.

11. Rewrite the program stored in the file *show_missing_grades.sql* to be a procedure. Accept a start_date and end_date to establish a date range. Display only enrollments between those two dates. Write your procedure so the start_date and end_date are optional. If both dates are not entered, display all applicable enrollments for the past year, and include a note about the date range. For each enrollment, list the CLASS_ID, STU_ID, and STATUS. Order the output by ENROLLMENT_DATE with the most recent enrollments first.

12. Find the file called *compute_average_grade.sql* and rewrite it as a function. Accept a CLASS_ID. Return the average grade.

13. Find the file called *count_classes_per_course.sql* and rewrite it as a function. Accept a COURSE_ID. Return the number of classes offered for that course.

14. Convert the file *show_class_offerings.sql* to a procedure. Accept a start date and end date. For each class found, display the CLASS_ID, START_DATE, instructor FIRST_NAME and LAST_NAME, course TITLE and SECTION_CODE, and average grade. Find the average grade by a call to the function *compute_average_grade*.

15. Create a package called *admin_tools_package* incorporating procedure and functions you wrote in steps K to N. Make the following public: *show_missing_grades, show_class_offerings*, *count_classes_per_course*. Make the following private: *compute_average_grade.*

Part 2: Managing Students and Grades

The *enrollments_package* you created in Part 1 contains the following public procedures:

1. Procedure *enroll_student_in_class* (p_stu_id IN enrollments.stu_id%TYPE, p_class_id IN enrollments.class_id%TYPE)
2. Procedure *drop_student_from_class*(p_stu_id IN enrollments.stu_id%TYPE, p_class_id IN enrollments.class_id%TYPE)
3. Procedure *student_class_list* (p_stu_id IN enrollments.stu_id%TYPE)

The Assignment and Deliverables:

Modify the *student_class_list* procedure in the package, to add the following functionality:

1. Utilize the overloading feature of the PLSQL package, to overload the *student_class_list* procedure as follows:
   - When the STU_ID parameter is passed, the procedure should display a list of classes in which the student has been enrolled, within the most recent 6 years.
   - When the procedure is called without a parameter, the procedure should display a list of classes for all students in which they have been enrolled, within the most recent 6 years.

2. Create a procedure *read_external_file*, to read an external text file stored outside the database as an operating system text file. Use DBMS_OUTPUT to display the content of the external file.

   The external file is named „*student_class_list.txt'* and is stored in the operating system directory referenced by the Oracle directory object ''WF_FLAGS'.

   Your output should look something like this: Enrollment Report

```
Student Id  Enrollment Date   Class id
101         12-Aug-2004              1
102         12- Aug -2004            1
103         12- Aug -2004            1
104         12- Aug -2004            1
*** END OF REPORT ***
```

   - Hints: The Exceptions used with UTL_FILE.GET_LINE:
     - INVALID_FILEHANDLE
     - INVALID_OPERATION
     - READ_ERROR
     - NO_DATA_FOUND
     - VALUE_ERROR
   - Directory object name is: 'WF_FLAGS' and must be referenced in capital letters.

Part 3: School Administrator's Tools (admin_tools_package)

The *admin_tools_package* you created in Part I contains the following programs:

1. Procedure *show_missing_grades* (p_start_date IN DATE DEFAULT NULL, p_end_date IN DATE DEFAULT NULL)
2. Procedure *show_class_offerings* (p_start_date IN DATE, p_end_date IN DATE)
3. Function *count_classes_per_course* (p_course_id IN classes.course_id%TYPE) RETURN NUMBER
4. Function *compute_average_grade* (p_class_id IN enrollments.class_id%TYPE) RETURN NUMBER This function is private in this package.

The Assignment and Deliverables:

1. Utilize the forward declaration concept to be able to move the body of the private function *compute_average_grade* to anywhere in the package body. Recompile and test the package.

Part 4: Create Database Triggers

1. Create the *grade_change_history* table as follows:

```
CREATE TABLE grade_change_history
 (time_stamp       DATE,
  stu_id           NUMBER(7,0),
  class_id         NUMBER(6,0),
  enroll_date      DATE,
  old_final_grade  CHAR(1),
  new_final_grade  CHAR(1));
```

2. Create a row level trigger *audit_grade_change* to keep a history of all the changes made to students" final letter grade. The grade change is recorded every time the FINAL_LETTER_GRADE field is updated in the ENROLLMENTS table.

   - Every time the trigger is fired, it should insert a record in the GRADE_CHANGE_HISTORY table, recording the old grade and the new grade for each student.

   - Test your trigger by updating the final_letter_grade for a student, in the ENROLLMENTS table.

Part 5: Create manage_triggers_package

The Assignment and Deliverables:

1. Create a package *manage_triggers_package* that contains two overloaded functions called *manage_triggers*. The functions are invoked to disable/enable all triggers for a table, or to compile a trigger.
   - Use Native Dynamic SQL to execute the DDL commands programmatically.
   - Code an exception handling block to display a message if the DDL command fails.

   Use the following guidelines:
   - When the function is called with two parameters: manage_triggers (p_tablename, p_action)
     - Pass a table name to P_TABLENAME parameter.
     - Pass „disable‟ or „enable‟ string to P_ACTION parameter.

   - When the function is called with only one parameter manage_triggers (p_trigger_name)
     - Pass a trigger name to P_TRIGGER_NAME parameter.

   Hints:
     - Use the ALTER TABLE command to disable/enable all triggers of a table programmatically.
     - Use the ALTER TRIGGER command to compile the trigger programmatically.