CSC 352 / 452: Database Programming

assignment #6 (60 Points)

**CSC 352/452-501: Due on Tuesday, 8/18/2015 at 11:59PM**

**CSC 352/452-510: Due on Wednesday, 8/19/2015 at 11:59PM**

Unless prior arrangements are made, homework turned in late but within 24 hours of the due time will be graded at 75% credit, homework turned in between 24 and 48 hours will be graded at 50% credit, and homework turned in later than 48 hours will not be accepted.

**1) (30 points)**

Employees may move to different departments. We want to keep track of the departments where each employee has been. To do so we create a new table emp\_dept\_info that keeps track of such history.

EMP\_DEPT\_INFO(EMPLOYEE\_ID, EMPLOYEE\_NAME, OLD\_DEPARTMENT\_NAME,

NEW\_DEPARTMENT\_NAME, OPERATION\_TIME);

Write a trigger emp\_dept\_trg that monitors the employee table as follows.

* When a row (record) is inserted into the employee table, the trigger automatically inserts a row (record) into the emp\_dept\_info table in any situations.
* The OLD\_DEPARTMENT\_NAME is always 'N/A'.
* If the new department\_id is NULL, the NEW\_DEPARTMENT\_NAME will be 'TBA'.
* When an employee changes his/her department (the old department\_id is not equal to the new department\_id), the trigger automatically inserts a row (record) into the emp\_dept\_info table.
* If both the old department\_id and new department\_id are NULL (from NULL department to NULL department), the trigger does not insert a row (record) into the emp\_dept\_info table.
* If the old department\_id is NULL, the OLD\_DEPARTMENT\_NAME will be 'TBA'.
* If the new department\_id is NULL, the NEW\_DEPARTMENT\_NAME will be 'TBA'.
* The SYSDATE can be used in the OPERATION\_TIME column.
* You can assume that the insert/update statements do not violate the integrity constraints between the department and employee tables.
* No temporary table/view/procedure/function is allowed in your program.
* You can only use the department, employee, and emp\_dept\_info tables in your trigger. You will get a zero point if you use a different table (e.g., different table names, column names, or data types) in your trigger.

Step 1) (0 point) Create the emp\_dept\_info table,

CREATE TABLE emp\_dept\_info

(

EMPLOYEE\_ID NUMBER(4) NOT NULL,

EMPLOYEE\_NAME VARCHAR2(50) NOT NULL,

OLD\_DEPARTMENT\_NAME VARCHAR2(50) NOT NULL,

NEW\_DEPARTMENT\_NAME VARCHAR2(50) NOT NULL,

OPERATION\_TIME DATE NOT NULL

);

Step 2) Create the trigger emp\_dept\_trg.

You will get a zero point if you use a different trigger name.

Step 3) Test your trigger.

You need to create/run some test cases to check your trigger. You do not need to submit your test cases.

**2) (30 points)**

Create a trigger called min\_sal\_trg on the employee table. When an INSERT or UPDATE statement is issued against the employee table, the trigger is fired to ensure that the value of the SALARY column meets **the criteria in the minimum\_salary table**. (For example, you can find that the minimum salary for a programmer is 800 from the minimum\_salary table. Your trigger ensures that the salary for a programmer in the employee table is greater than or equal to 800.)

Step 1) (0 point) Create a table minimum\_salary as follows.

CREATE TABLE minimum\_salary

(

JOB VARCHAR2(50) PRIMARY KEY,

MINIMUM\_SALARY NUMBER(7, 2) NOT NULL

);

Step 2) (0 point) Populate the minimum\_salary table as follows.

INSERT INTO minimum\_salary VALUES ('ANALYST', 2000);

INSERT INTO minimum\_salary VALUES ('DATABASE ADMINISTRATOR', 2500);

INSERT INTO minimum\_salary VALUES ('PRESIDENT', 4800);

INSERT INTO minimum\_salary VALUES ('PROGRAMMER', 800);

INSERT INTO minimum\_salary VALUES ('PUBLIC ACCOUNTANT', 2400);

INSERT INTO minimum\_salary VALUES ('SALESMAN', 1800);

INSERT INTO minimum\_salary VALUES ('VICE PRESIDENT', 3800);

INSERT INTO minimum\_salary VALUES ('DEFAULT', 1800);

COMMIT;

Step 3) Create the trigger min\_sal\_trg.

* The minimum\_salary table is read-only. Your trigger **cannot** modify any rows in the minimum\_salary table.
* You must get the minimum salaries from the minimum\_salary table in your program.
* Hard coding, except the string'DEFAULT', is not allowed in your program (e.g., IF job = 'SALESMAN' THEN v\_min\_sal = 1800 …).
* If the job cannot be found from the minimum\_salary table (e.g., Program Facilitator), the job is considered as “DEFAULT”. (You need to check whether the salary is equal to or greater than the minimum salary for job = 'DEFAULT'.)
* If the salary is equal to or greater than the minimum salary of the corresponding job, the trigger does not change anything.
* If the salary is less than the minimum salary of the corresponding job, the trigger increases the salary to the minimum salary of the corresponding job.
* No temporary table/view/procedure/function is allowed in your program.
* To avoid a **mutating table error**, please take a look examples on page 6, class handout 9. (Hint: you cannot use some INSERT/UPDATE statements to modify the employee table in your trigger.)
* You will get a zero point if you use a different trigger name.
* If you modified the employee table created in Assignment #1, you need to delete and re-populate it.

Step 4) Test your trigger.

You need to create/run some test cases to check whether the values of the salary column in the employee table meet the criteria in the minimum\_salary table when rows (records) are inserted or updated. You do not need to submit your test cases.

**Please submit a text file containing all the source codes to D2L before or on due date.**