Practical 9

Introduction to triggers and views
Creating tables with referential integrity constraints

Learning objectives

1. Write AFTER and BEFORE triggers for UPDATE, INSERT and DELETE events.
2. Manage errors in triggers.
3. CREATE VIEWs using data from tables and other views and use them with SELECT statements.
4. Create tables with referential integrity constraints.

Setting-up

- Do this practical is DBS/Prac09 directory. If Prac09 directory is not there, create it.
- **Task 2:** Same .sql files used in **Practical 06** will be used in this task.
    - If you have downloaded the .sql files earlier, please use them. If not, download and copy the given sql files from the practical 06 link to your Prac09 directory.
- In the Prac09 directory, get a Terminal and connect to the MySQL server using the usual password to start the Task1.
- Use a suitable command record all the commands and query results to *Prac09Workings,out* file
- Get another Terminal, open a sql file in Vim (> vim *Prac09Commands.sql*) or any text editor you wish to use. First create your commands in this file before pasting theSQL prompt. Use a comment line with the question number before starting each task.

1. Task 1: Trigger examples based on lecture slides

This activity use employees (Emp) and department(Dept) tables as used in the Lecture 8 . *The names of the tables are changed to Emp1 and Dept1 to make them different from Emp and Dept tables you will be using in the task 2.*

1. Create following two tables with suitable data types and foreign key constraints in the *'dsworks'* database. Make sure no NOT NULL constraint is set for salary attribute. Salary is the annual salary such as $50000.00.
   `Emp1(emp_no, name, dept_no, salary, supervisor)`

   `Dept1(dept_no, dept_name, total_sal, mgr)`
2. Assume that after an employee is added to the database, we want to modify the total salary column of the `dept` of that employee. You can create a trigger based on the example 2 given in the lecture slides to implement the above requirement as follows:

```
DELIMITER //
CREATE TRIGGER after_emp_insert
      AFTER INSERT ON Emp1
      FOR EACH ROW
            BEGIN
         IF NEW.salary IS NOT NULL THEN
                 UPDATE Dept1
                       SET total_sal = total_sal + NEW.salary
                       WHERE dept_no = NEW.dept_no;
              END IF;
            END//
DELIMITER ;
```

Implement the above trigger in a sql file and source it to create the trigger in the *'dsworks'* database.
Insert some sample values and verify the trigger is working as expected.

3. When the salary of an employee is changed, the total salary of their *dept* has to be modified. Use the example 3 given in the Lecture 8, slide 18 and implement a AFTER UPDATE trigger to satisfy the above requirement. You can add the trigger to the sql file created in the previous question(Q2) OR create a new sql file. Update some salaries and verify the trigger is working as expected.

4. In the (Q3) above, the trigger works when both the old salary and the new salary are NOT NULL. Revise the trigger code to make it works properly when either new salary is updated to NULL while the old salary was NOT NULL, OR when the old salary was NULL and it is updated to a new salary which is not a null value.

5. When an employee is removed, the total salary of the `dep` has to be changed. Write an AFTER DELETE trigger to enforce this rule. You can use the example 4 given in the Lecture 8, slide 19.

6. Assume the salary cannot be less than $24000.00. Add a suitable BEFORE UPDATE and BEFORE INSERT triggers to enforce this constraint. Show a suitable error message including the salary value when the constraint is not satisfied. (Refer Lecture 8, slide 21-23 for raising an error message)

2. Task 2: Triggers on employee database

You will use *Emp, Dept, Proj* and *Pworks* tables first used in the prac06 in this task. Follow the instructions given in Practical06, task 2 to create tables *Emp, Dept, Proj* and *Pworks* in 'dswork' database and then to insert the given set of values into the created tables. Please refer practical 06 for description of the tables.

1. Add two **attributes** with suitable data types to the *Proj* table as follows:
   `empcnt`: for recording the number of employees working on each project
   `hrstotal:` for recording the total hours to be spent on each project by all the employees assigned to it.

2. Update the values of the two new attributes in the `Proj` table based on information in the *Pworks* table of the database.

3. Write suitable triggers to maintain the attributes `empcnt` and `hrstotal` in the *Proj* table, whenever the *Pworks* table is updated using insert, delete or update operations.
   Test the triggers to verify whether they work correctly.

4. Create a trigger that will do the following:
   Whenever a new project is inserted into the *Proj* table, add rows into the *Pworks* table for all employees of the department responsible for the new project. The total hours allocated on the project should be the same as the average total hours of the existing projects.
   Test the action of the trigger by inserting some rows into *Proj* table.

3. Task 3: Creating views

1. Create a view named “ManagingDept” to show the *empno, firstname, lastname, job, managing department number* and the *name of the managing department* of each department manager. Use suitable column names for the managing department number and the name of the managing department.
   Create the view in a new sql file.
   Refer Practical 6, Task 3 Q2 for the join query related to this question.
   Use the view and show the *empno, firstname, lastname, job, managing department number* and the *name of the managing department* of each department manager.

2. Create a view to show *empno, firstname, midinit, lastname, phoneno*, *name of the managing department* of each department manager, using a suitable join between “ManagingDept” view and *Emp* tables. (You are not supposed to change the above query to directly get the solution)

3. Change the *firstname* and the *lastname* of one of the manager using a suitable update SQL statement.
   Use a SELECT * statement to show all the data of the “ManagingDept” view.
   Observe whether the results reflect the change you have done for the manager name above. Source the view creating sql file, then use the same SELECT * statement and observe the results.

4. Task 4: Additional task with referential integrity setting

A database on World War II battle ships consists of four relations whose schema is as follows (taken from Exercise 2.4.3 of the text book):

```
Classes (class: char(15), ctype: char(2), country: varchar(12), numGuns:
        integer(2), bore: integer(2), displacement: integer(6))
Ships (name: char(15), class: char(15), launched: integer(4))
Battles (name: char(15), bdate: date)
Outcomes (ship: char(15), battle: char(15), result: char(7))
```

Ships are built in “classes” from the same design, and the class is usually named for the first ship of that class.
The relation Classes records the name of the class, the type (bb for battleship or bc for battlecruiser), the country that built the ship, the number of main guns, the bore (diameter of the gun barrel, in inches) of the main guns, and the displacement (weight, in tons).
Relation Ships records the name of the ship, the name of its class, and the year in which the ship was launched.
Relation Battles gives the name and date of battles involving these ships, relation Outcomes gives the result (sunk, damaged, or ok) for each ship in each battle.

1. Create above tables in the MYSQL database, choosing appropriate primary keys for them. *Include the following referential integrity constraints and handle violations due to deletion of referenced attribute value by setting the referencing attribute value to NULL or by using the cascade option if setting to NULL is not possible.*
    - Every class mentioned in ships must be mentioned in Classes.
    - Every battle mentioned in Outcomes must be mentioned in Battles.
    - Every ship mentioned in Outcomes must be mentioned in Ships.

   After creating the tables of the battle ships database and then enter data into them to test the operation of all the constraints you have declared.

   Part of the sample data from the text book can be used, some with modifications to check for violations of constraints. (Data from the text is available in the *SampleDataTask4*.docx file). But it is not necessary to enter all of the data from the text, since no queries are to be executed on this database). To test the constraints, perform some insert, delete and modify operations that work correctly as well as some that may cause violations.

   (Refer Lecture 6, slide 35 onwards for referential integrity lecture slides, examples)

5. Task 5: Additional task with triggers

1. Following constraints are given to be applied regarding the values of the number of guns and the bores of the tables created in the task 4 above:
   - The number of guns must be between 8 and 12
   - The bore of guns cannot exceed 18

A trigger can be created to enforce these constraints as follows:

```
DELIMITER //
CREATE TRIGGER ClassesInsertCheck
BEFORE INSERT ON Classes
FOR EACH ROW
BEGIN
     IF NEW.numguns < 8 THEN
          SET NEW.numguns = 8;
     ELSEIF NEW.numguns > 12 THEN
          SET NEW.numguns = 12;
     END IF;
     IF NEW.bore > 18 THEN
          SET NEW.bore = 18;
     END IF;
END //

CREATE TRIGGER ClassesUpdateCheck
BEFORE UPDATE ON Classes
FOR EACH ROW
BEGIN
     IF NEW.numguns < 8 OR NEW.numguns > 12 THEN
          SET NEW.numguns = OLD.numguns;
     END IF;
     IF NEW.bore > 18 THEN
          SET NEW.bore = OLD.bore;
     END IF;
END //

DELIMITER ;
```

   a. Create the above two triggers in the MYSQL database.

NOTE: We do this by checking every time a tuple is inserted to see that the conditions hold, and every time a trigger is updated, to make sure those conditions still hold. This requires two triggers. For the insert trigger we replace the value by the nearest boundary, while for the update trigger we simply don’t change a value to something illegal by resetting it to the old value.

b. Use suitable command and see the triggers created in the MYSQL database.

c. Enter some sample values and see whether the triggers are working.

2. Implement the following restrictions by editing the triggers given above.
   a. The ships were launched between 1913 and 1944.
   b. The result of battle on a ship is one of sunk, damaged, or ok.
   c. If a class of ship has more than 9 guns, then their bore must be no larger than 14 inches.
   d. Only Japanese ships had displacements of more than 50,000.

Enter some sample values and see whether the triggers are working as expected.

6. Submitting your work

Your prac09 directory should have *Prac09Workings,out* and Prac09Commands.sql files together with all given .sql files and set of new .sql files you have created. Zip your Prac09 directory and upload it to Blackboard under 'Assessments/In Class Practical Submissions'

Check whether you have achieved learning outcomes:

I am confident that I can,

Write a BEFORE INSERT, BEFORE UPDATE, BEFORE DELETE triggers to enforce some meaningful constraints on table data	✓
Write a AFTER INSERT, AFTER UPDATE, AFTER DELETE triggers to enforce some meaningful constraints on table data	
Manage error handling in triggers using SIGNAL SQLSTATE statements	
Create views using base tables and retrieve data from a view	
Create views using base tables and/or other views retrieve data from a view	
Create tables with suitable referential integrity constraints	

Please refer lecture slides, reading materials, and online resources and attempt again, if all the learning outcomes were not achieved. Ask your tutor and get help if you need any clarification.

It's always a good practise to try to finish the practical of a particular week, before attempting the next practical worksheet as your work will be building upon the previous week's tasks.