DEPARTMENT OF COMPUTER SYSTEMS ENGINEERING
MEHRAN UNIVERSITY OF ENGINEERING & TECHNOLOGY, JAMSHORO
Database Management Systems (4th Semester) 18CS
Lab Experiment 15

Roll No: **Date of Conduct:**

SubmissionDate: **Grade Obtained:**

Problem Recognition (0.3)	Completeness & accuracy (0.4)	Timeliness (0.3)	Score (1.0)

Objective: To understand the use of Triggers in PL/SQL.

Tools: MySql/oracle.

Introduction:

Triggers: Triggers are stored programs, which are automatically executed or fired when some events occur. Triggers are, in fact, written to be executed in response to any of the following events −

- A **database manipulation (DML)** statement (DELETE, INSERT, or UPDATE)
- A **database definition (DDL)** statement (CREATE, ALTER, or DROP).
- A **database operation** (SERVERERROR, LOGON, LOGOFF, STARTUP,

  Or SHUTDOWN).

Triggers can be defined on the table, view, schema, or database with which the event is associated.

Benefits of Triggers

Triggers can be written for the following purposes −

- Generating some derived column values automatically
- Enforcing referential integrity
- Event logging and storing information on table access
- Auditing
- Synchronous replication of tables
- Imposing security authorizations
- Preventing invalid transactions

Creating Triggers

The syntax for creating a trigger is −

```
CREATE [OR REPLACE ] TRIGGER trigger_name
{BEFORE | AFTER | INSTEAD OF }
{INSERT [OR] | UPDATE [OR] | DELETE}
[OF col_name]
ON table_name
[REFERENCING OLD AS o NEW AS n]
[FOR EACH ROW]
WHEN (condition)
DECLARE
   Declaration-statements
BEGIN
   Executable-statements
EXCEPTION
   Exception-handling-statements
END;
```

Where,

- CREATE [OR REPLACE] TRIGGER trigger_name − Creates or replaces an existing trigger with the *trigger_name*.
- {BEFORE | AFTER | INSTEAD OF} − This specifies when the trigger will be executed. The INSTEAD OF clause is used for creating trigger on a view.
- {INSERT [OR] | UPDATE [OR] | DELETE} − This specifies the DML operation.
- [OF col_name] − This specifies the column name that will be updated.
- [ON table_name] − This specifies the name of the table associated with the trigger.
- [REFERENCING OLD AS o NEW AS n] − This allows you to refer new and old values for various DML statements, such as INSERT, UPDATE, and DELETE.
- [FOR EACH ROW] − This specifies a row-level trigger, i.e., the trigger will be executed for each row being affected. Otherwise the trigger will execute just once when the SQL statement is executed, which is called a table level trigger.
- WHEN (condition) − this provides a condition for rows for which the trigger would fire. This clause is valid only for row-level triggers.

LAB TASK

1. Consider Student table, in which student's marks assessment is recorded. In such schema, create a trigger so that the total and percentage of specified marks is automatically inserted whenever a record is inserted.

Task:

```
CREATE OR REPLACE TRIGGER stud_marks
BEFORE INSERT ON Student FOR EACH ROW WHEN (NEW.TID > 0 )
DECLARE
     v_subj1 student.subj1%TYPE;
     v_subj2 student.subj2%TYPE;
     v_subj3 student.subj3%TYPE;
     v_total number;
     v_percent  number;
BEGIN
   Select subj1,subj2,subj3 into v_subj1,v_subj2,v_subj3 from student where TID=1;--where into.tid ;
   v_total :=  v_subj3+ v_subj2+ v_subj3;
   v_percent := v_total * 60 / 100;
   dbms_output.put_line('TOTAL MARKS: ' || v_total);
   dbms_output.put_line('Percentage: ' || v_percent);
END;
/
insert into Student values(6, 'ABCDE', 20, 20, 20, 0, 0);
```

Script Output

Task completed in 0.091 seconds

```
Trigger STUD_MARKS compiled

TOTAL MARKS: 60
Percentage: 36


1 row inserted.
```

2. Create a row level trigger for the customers table that would fire for INSERT or UPDATE or DELETE operations performed on the CUSTOMERS table. This trigger should display the salary difference between the old values and new values.

Task:

```
SET SERVEROUTPUT ON;
CREATE OR REPLACE TRIGGER display_salary_changes
BEFORE DELETE OR INSERT OR UPDATE ON EMP
FOR EACH ROW
WHEN (NEW.EMPNO > 0)
DECLARE
   sal_diff number;
BEGIN
   sal_diff := :NEW.sal  - :OLD.sal;
   dbms_output.put_line('Old salary: ' || :OLD.sal);
   dbms_output.put_line('New salary: ' || :NEW.sal);
   dbms_output.put_line('Salary difference: ' || sal_diff);
END;
/
INSERT INTO EMP(EMPNO,ENAME,SAL) VALUES (7890,'Tahir',75000.00);
UPDATE EMP SET sal =sal+1000 where EMPNO=7890;
----18CS31--------------
```

Script Output

Task completed in 0.198 seconds

```
1 row inserted.

Old salary: 75000
New salary: 76000
Salary difference: 1000
  Difference 1000


1 row updated.
```