EXERCISE-17

TRIGGER

DEFINITION

A trigger is a statement that is executed automatically by the system as a side effect of a modificati to the database. The parts of a trigger are,

- **Trigger statement**: Specifies the DML statements and fires the trigger body. It also specifies the table to which the trigger is associated.

- **Trigger body or trigger action**: It is a PL/SQL block that is executed when the triggerin statement is used.

- **Trigger restriction**: Restrictions on the trigger can be achieved

The different uses of triggers are as follows,

- *To generate data automatically*
- *To enforce complex integrity constraints*
- *To customize complex securing authorizations*
- *To maintain the replicate table*
- To audit data modifications

TYPES OF TRIGGERS

The various types of triggers are as follows,

- **Before**: It fires the trigger before executing the trigger statement.

- **After**: It fires the trigger after executing the trigger statement
- .
- **For each row**: It specifies that the trigger fires once per row
- .
- **For each statement**: This is the default trigger that is invoked. It specifies that th fires once per statement.

VARIABLES USED IN TRIGGERS

- :new

- :old

These two variables retain the new and old values of the column updated in the database. in these variables can be used in the database triggers for data manipulation

SYNTAX

```
create or replace trigger triggername [before/after] {DML statements}
on [tablename] [for each row/statement]
begin
----------
```

```
SQL> delete from itempls where ename='xxx';
delete from itempls where ename='xxx'
       *
ERROR at line 1:
ORA-20010: You cannot do manipulation
ORA-06512: at "STUDENT.ITTRIGG", line 2
ORA-04088: error during execution of trigger 'STUDENT.ITTRIGG'

SQL> update itempls set eid=15 where ename='yyy';
update itempls set eid=15 where ename='yyy'
       *
ERROR at line 1:
ORA-20010: You cannot do manipulation
ORA-06512: at "STUDENT.ITTRIGG", line 2
ORA-04088: error during execution of trigger 'STUDENT.ITTRIGG'
```

TO DROP THE CREATED TRIGGER

```
SQL> drop trigger ittrigg;

Trigger dropped.
```

TO CREATE A TRIGGER THAT RAISES AN USER DEFINED ERROR MESSAGE AND DOES NOT ALLOW UPDATION AND INSERTION

```
SQL> create trigger ittriggs before insert or update of salary on itempls for each row
  2  declare
  3  triggsal itempls.salary%type;
  4  begin
  5  select salary into triggsal from itempls where eid=12;
  6  if(:new.salary>triggsal or :new.salary<triggsal) then
  7  raise_application_error(-20100,'Salary has not been changed');
  8  end if;
  9  end;
 10  /

Trigger created.

SQL> insert into itempls values ('bbb',16,45000);
insert into itempls values ('bbb',16,45000)
       *
ERROR at line 1:
ORA-04098: trigger 'STUDENT.ITTRIGGS' is invalid and failed re-validation

SQL> update itempls set eid=18 where ename='zzz';
update itempls set eid=18 where ename='zzz'
       *
ERROR at line 1:
ORA-04298: trigger 'STUDENT.ITTRIGGS' is invalid and failed re-validation
```

- Cursor for loop
- Explicit cursor

```
---------------------
---------------------
exception
end;
```

USER DEFINED ERROR MESSAGE

The package "raise_application_error" is used to issue the user defined error messages

Syntax: raise_application_error(error number,'error message');

The error number can lie between -20000 and -20999.

The error message should be a character string.

TO CREATE THE TABLE 'ITEMPLS'

```
SQL> create table itempls (ename varchar2(10), eid number(5), salary number(10));
Table created.

SQL> insert into itempls values('xxx',11,10000);
1 row created.

SQL> insert into itempls values('yyy',12,10500);
1 row created.

SQL> insert into itempls values('zzz',13,15500);
1 row created.

SQL> select * from itempls;
ENAME          EID   SALARY
---------- --------- ---------
xxx            11    10000
yyy            12    10500
zzz            13    15500
```

TO CREATE A SIMPLE TRIGGER THAT DOES NOT ALLOW INSERT UPDATE AND DELETE OPERATIONS ON THE TABLE

```
SQL> create trigger ittrigg before insert or update or delete on itempls for each row
  2  begin
  3  raise_application_error(-20010,'You cannot do manipulation');
  4  end;
  5
  6  /
Trigger created.

SQL> insert into itempls values('aaa',14,34000);
insert into itempls values('aaa',14,34000)
            *
ERROR at line 1:
ORA-20010: You cannot do manipulation
ORA-06512: at "STUDENT.ITTRIGG", line 2
ORA-04088: error during execution of trigger 'STUDENT.ITTRIGG'
```

Program 1

Write a code in PL/SQL to develop a trigger that enforces referential integrity by preventing the deletion of a parent record if child records exist.

```
Create OR REPLACE TRIGGER trg_prevent_parent_delete
Before DELETE ON department
for EACH ROW
DECLARE.
V_Count Number ;
BEGIN
SELECT Count (*) INTO V_Count from employee where
dept_id =: OLD.dept_id;
   If V_Count > 0 Then
    RAISE_Application_Error (-2000, 'cannot delete parent with
. child records exists in Employee table,');
   END;
/
```

Program 2

Write a code in PL/SQL to create a trigger that checks for duplicate values in a specific column and raises an exception if found.

```
CREATE OR REPLACE TRIGGER
  trg_check_duplicate_email
BEFORE INSERT OR UPDATE ON Students'
  FOR EACH ROW
  DECLARE
  V_Count Number;
   BEGIN
  SELECT Count(*) INTO V_Count from Students' where
  email =: NEW.email;
  If v_Count>0 THEN.
   RAISE_APPLICATION_ERROR(-20002, 'Duplicate email
   detected. Each email must be unique:');
     END If;
   END;
```

190

Program 3

Write a code in PL/SQL to create a trigger that restricts the insertion of new rows if the total of a column's values exceeds a certain threshold.

```
CREATE OR REPLACE TRIGGER trg_limit_total_salary
  Before INSERT ON employee
  for each row
    DECLARE
  V_Total Number;
   V_Threshold CONSTANT Number := 100000;
  BEGIN
  Select NVL (Sum(Salary),0) INTO employee_audit
                (emp_id, old_Sday, new_salary, dept
  , change_by)
  VALUES (:OLD.emp_id, :OLD.Salary, :NEW Salary, $
  USER);
  END;
/
```

Program 4

Write a code in PL/SQL to design a trigger that captures changes made to specific columns and logs them in an audit table.

```
CREATE TABLE activity_log(
    table_name VARCHAR2(50),
    operation_type VARCHAR2(20),
    user_name VARCHAR2(30),
    activity_date DATE
);

CREATE OR REPLACE TRIGGER trg_user_activity
AFTER INSERT OR Update OR Delete ON employee
BEGIN
INSERT INTO employee_audit (emp_id, old_salary, new_salary, change_date,
changedby)
VALUES (:OLD.emp_id, OLDsalary, :New.Salary, SysDate,
USER);

END;
/
```

Program 5

Write a code in PL/SQL to implement a trigger that records user activity (inserts, updates, deletes) in an audit log for a given set of tables.

```
CREATE TABLE activity_log (
    table_name VARCHAR2(50),
    operation_type VARCHAR2(20),
    user_name VARCHAR2(30),
    activity_date DATE
);

CREATE OR REPLACE TRIGGER trg_user_activity
AFTER INSERT OR UPDATE OR DELETE ON employee
BEGIN
    INSERT INTO activity_log (table_name, operation_type,
    user_name, activity_date)
    VALUES ('employee', ORA_SYSEVENT, USER,
        SYSDATE);
END;
/
```

Program 7

Write a code in PL/SQL to implement a trigger that automatically calculates and updates a running total column for a table whenever new rows are inserted.

```
CREATE OR REPLACE TRIGGER
  trg_Check_Stock_availability
  Before INSERT ON Order
For EACH Row
  DECLARE
  v_Stock Number;
  BEGIN
SELECT quantity_in_stock INTO v_Stock from Inventory
where item_id = :NEW.Item_id;
IF v_Stock < :NEW.order_quantity Then
  RAISE_Application_Error(-20004,'Insufficient Stock
available for the requested Item!');
    END IF;
  END;
/
```

Program 8

Write a code in PL/SQL to create a trigger that validates the availability of items before allowing an order to be placed, considering stock levels and pending orders.

```
CREATE OR REPLACE TRIGGER Check_it
  BEFORE INSERT ON orders
  FOR EACH ROW
  DECLARE
      V_Stock NUMBER;
  BEGIN
    ~~SELECT stock_qty INTO V_Stock~~
    IF Sales >= 100000 THEN
        incentive := Sales * 0.10;
    ELSIF Sales >= 50000 THEN
        incentive := Sales * 0.05;
    ELSE
        incentive := 0;
    END IF;
    DBMS_Output.Put_LINE ('Sales :' ||Sales);
    DBMS_Output.Put_LINE ('Incentive :' || incentive);
END;
/
BEGIN
    calc_incentive;
END;
/
```

Evaluation Procedure	Marks awarded
PL/SQL Procedure(5)	5
Program/Execution (5)	5
Viva(5)	5
Total (15)	15
Faculty Signature	