EXERCISE-17

TRIGGER

DEFINITION

A trigger is a statement that is executed automatically by the system as a side effect of a modification 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 triggering 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 the trigger 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. The values 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

```
-----------------------------
-----------------------------
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'
```

```
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

184

TO CREATE THE TABLE 'SSEMPP'

SQl > create table ssempp(eid number(10), ename varchar2(20), job varchar2(20), sal number (10),dnonumber(5)).
Table created

SQl > insert into ssempp values(1,'nala','lecturer',34000,11);
1 row created

SQl > insert into ssempp values(2,'kala',' seniorlecturer',20000,12);
1 row created.

SQL> insert into ssempp values(5,'ajay','lecturer',30000,11);
1 row created.

SQL> insert into ssempp values(6,'vijay','lecturer',18000,11);
1 row created.

SQL> insert into ssempp values(3,'nila','professor',60000,12);
1 row created.

SQL> select * from ssempp;

EID	ENAME	JOB	SAL	DNO
1	nala	lecturer	34000	11
2	kala	seniorlecturer	20000	12
5	ajay	lecturer	30000	11
6	vijay	lecturer	18000	11
3	nila	professor	60000	12

EXTRA PROGRAMS

TO WRITE A PL/SQL BLOCK TO DISPLAY THE EMPOYEE ID AND EMPLOYEE NAME USING CURSOR FOR LOOP

```
SQL> set serveroutput on;
SQL> declare
 2 begin
 3 for emy in (select eid,ename from ssempp)
 4 loop
 5 dbms_output.put_line('Employee id and employee name  are '|| emy.eid 'and'|| emy.ename);
 6 end loop;
 7 end;
 8 /
Employee id and employee name are 1 and nala
Employee id and employee name are 2 and kala
Employee id and employee name are 5 and ajay
Employee id and employee name are 6 and vijay
Employee id and employee name are 3 and nila
```

PL/SQL procedure successfully completed.

185

TO WRITE A PL/SQL BLOCK TO UPDATE THE SALARY OF ALL EMPLOYEES WHERE DEPARTMENT NO IS 11 BY 5000 USING CURSOR FOR LOOP AND TO DISPLAY THE UPDATED TABLE

```
SQL> set serveroutput on;
SQL> declare
 2 cursor cem is select eid,ename,sal,dno from ssempp where dno=11;
 3 begin
 4 --open cem;
 5 for rem in cem
 6 loop
 7 update ssempp set sal=rem.sal+5000 where eid=rem.eid;
 8 end loop;
 9 --close cem;
 10 end;
 11 /
```

PL/SQL procedure successfully completed.

SQL> select * from ssempp;

EID	ENAME	JOB	SAL	DNO
1	nala	lecturer	39000	11
2	kala	seniorlecturer	20000	12
5	ajay	lecturer	35000	11
6	vijay	lecturer	23000	11
3	nila	professor	60000	12

TO WRITE A PL/SQL BLOCK TO DISPLAY THE EMPLOYEE ID AND EMPLOYEE NAME WHERE DEPARTMENT NUMBER IS 11 USING EXPLICIT CURSORS

```
 1 declare
 2 cursor cenl is select eid,sal from ssempp where dno=11;
 3 ecode ssempp.eid%type;
 4 esal empp.sal%type;
 5 begin
 6 open cenl;
 7 loop
 8 fetch cenl into ecode,esal;
 9 exit when cenl%notfound;
 10 dbms_output.put_line(' Employee code and employee salary are' || ecode 'and'|| esal);
 11 end loop;
 12 close cenl;
 13* end;
```

SQL> /
Employee code and employee salary are 1 and 39000
Employee code and employee salary are 5 and 35000
Employee code and employee salary are 6 and 23000

PL/SQL procedure successfully completed.

186

TO WRITE A PL/SQL BLOCK TO UPDATE THE SALARY BY 5000 WHERE THE JOB IS LECTURER , TO CHECK IF UPDATES ARE MADE USING IMPLICIT CURSORS AND TO DISPLAY THE UPDATED TABLE

```
SQL> declare
  2  county number;
  3  begin
  4  update ssempp set sal=sal+10000 where job='lecturer';
  5  county=sql%rowcount;
  6  if county > 0 then
  7  dbms_output.put_line('The number of rows are '|| county);
  8  end if;
  9  if sql %found then
 10  dbms_output.put_line('Employee record modification successful');
 11  else if sql%notfound then
 12  dbms_output.put_line('Employee record is not found');
 13  end if;
 14  end if;
 15  end;
 16  /
The number of rows are 3

Employee record modification successful

PL/SQL procedure successfully completed.

SQL> select * from ssempp;
```

EID	ENAME	JOB	SAL	DNO
1	nala	lecturer	44000	11
2	kala	seniorlecturer	20000	12
5	ajay	lecturer	40000	11
6	vijay	lecturer	28000	11
3	nila	professor	60000	12

PROGRAMS

TO DISPLAY HELLO MESSAGE

```
SQL> set serveroutput on;
SQL> declare
  2  a varchar2(20);
  3  begin
  4  a:='Hello';
  5  dbms_output.put_line(a);
  6  end;
  7  /
Hello

PL/SQL procedure successfully completed.
```

TO INPUT A VALUE FROM THE USER AND DISPLAY IT

187

```
SQL> set serveroutput on;
SQL> declare
  2  a varchar2(20);
  3  begin
  4  a:=&a;
  5  dbms_output.put_line(a);
  6  end;
  7  /
Enter value for a: 5
old   4: a:=&a;
new   4: a:=5;
5
```

PL/SQL procedure successfully completed.

GREATEST OF TWO NUMBERS

SQL> set serveroutput on;

```
SQL> declare
  2  a number(7);
  3  b number(7);
  4  begin
  5  a:=&a;
  6  b:=&b;
  7  if(a>b) then
  8  dbms_output.put_line (' The grerater of the two is'|| a);
  9  else
 10  dbms_output.put_line (' The grerater of the two is'|| b);
 11  end if;
 12  end;
 13  /
Enter value for a: 5
old   5: a:=&a;
new   5: a:=5;
Enter value for b: 9
old   6: b:=&b;
new   6: b:=9;
The grerater of the two is9
```

PL/SQL procedure successfully completed.

GREATEST OF THREE NUMBERS

SQL> set serveroutput on;

```
SQL> declare
  2  a number(7);
  3  b number(7);
  4  c number(7);
  5  begin
  6  a:=&a;
  7  b:=&b;
  8  c:=&c;
```

```
 9  if(a>b and a>c) then
10  dbms_output.put_line (' The greatest of the three is ' || a);
11  else if (b>c) then
12  dbms_output.put_line (' The greatest of the three is ' || b);
13  else
14  dbms_output.put_line (' The greatest of the three is ' || c);
15  end if;
16  end if;
17  end;
18  /
```

Enter value for a: 5
old   6: a:=&a;
new   6: a:=5;
Enter value for b: 7
old   7: b:=&b;
new   7: b:=7;
Enter value for c: 1
old   8: c:=&c;
new   8: c:=1;
The greatest of the three is 7

PL/SQL procedure successfully completed.

PRINT NUMBERS FROM 1 TO 5 USING SIMPLE LOOP

SQL> set serveroutput on;

```
SQL> declare
  2  a number:=1;
  3  begin
  4  loop
  5  dbms_output.put_line (a);
  6  a:=a+1;
  7  exit when a>5;
  8  end loop;
  9  end;
 10  /
1
2
3
4
5
```

PL/SQL procedure successfully completed.

PRINT NUMBERS FROM 1 TO 4 USING WHILE LOOP

SQL> set serveroutput on;

```
SQL> declare
  2  a number:=1;
  3  begin
```

```
4  while(a- 5)
5  loop
6  dbms_output.put_line (a);
7  a:=a+1.
8  end loop;
9  end.
10 /
```
1
2
3
4
PL/SQL procedure successfully completed.

PRINT NUMBERS FROM 1 TO 5 USING FOR LOOP
SQL> set serveroutput on;

```
SQL> declare
2  a number:=1;
3  begin
4  for a in 1..5
5  loop
6  dbms_output.put_line (a);
7  end loop;
8  end;
9 /
```
1
2
3
4
5
PL/SQL procedure successfully completed.

PRINT NUMBERS FROM 1 TO 5 IN REVERSE ORDER USING FOR LOOP
SQL> set serveroutput on;
```
SQL> declare
2  a number:=1;
3  begin
4  for a in reverse 1..5
5  loop
6  dbms_output.put_line (a);
7  end loop;
8  end;
9 /
```
5
4
3
2
1
PL/SQL procedure successfully completed.

TO CALCULATE AREA OF CIRCLE
```
SQL> set serveroutput on;
SQL> declare
2  pi constant number(4,2):=3.14;
3  a number(20);
4  r number(20);
5  begin
```

```
6 r:=&r;
7 a:= pi* power(r,2);
8 dbms_output_put_line (' The area of circle is ' || a);
9 end;
10 /
Enter value for r: 2
old  6: r:=&r;
new  6: r:=2;
The area of circle is 13
PL/SQL procedure successfully completed.
```

TO CREATE SACCOUNT TABLE

```
SQL> create table saccount ( accno number(5), name varchar2(20), bal number(10));
Table created.
SQL> insert into saccount values ( 1,'mala',20000);
1 row created.
SQL> insert into saccount values (2,'kala',30000);
1 row created.
SQL> select * from saccount;
```

ACCNO NAME	BAL
1 mala	20000
2 kala	30000

```
SQL> set serveroutput on;
SQL> declare
2  a_bal number(7);
3  a_no varchar2(20);
4  debit number(7):=2000;
5  minamt number(7):=500;
6  begin
7  a_no:=&a_no;
8  select bal into a_bal from saccount where accno= a_no;
9  a_bal:= a_bal-debit;
10 if (a_bal > minamt) then
11 update  saccount  set bal=bal-debit where accno=a_no;
12 end if;
13 end;
14
15 /
Enter value for a_no: 1
old  7: a_no:=&a_no;
new  7: a_no:=1;


PL/SQL procedure successfully completed.
SQL> select * from saccount;
```

ACCNO NAME	BAL
1 mala	18000
2 kala	30000

TO CREATE TABLE SROUTES

```
SQL> create table sroutes ( rno number(5), origin varchar2(20), destination varchar2(20), fare numbe
r(10), distance number(10));
Table created.
```

```
SQL> insert into sroutes values ( 2, 'chennai', 'dindugal', 400,230);
1 row created.
SQL> insert into sroutes values ( 3, 'chennai', 'madurai', 250,300);
1 row created.
SQL> insert into sroutes values ( 6, 'thanjavur', 'palani', 350,370);
1 row created.
SQL> select * from sroutes;
```

RNO	ORIGIN	DESTINATION	FARE	DISTANCE
2	chennai	dindugal	400	230
3	chennai	madurai	250	300
6	thanjavur	palani	350	370

```
SQL> set serveroutput on;

SQL> declare
 2  route sroutes.rno % type;
 3  fares sroutes.fare % type;
 4  dist sroutes.distance % type;
 5  begin
 6  route:=&route;
 7  select fare, distance into fares , dist from sroutes where rno=route;
 8  if (dist < 250) then
 9  update sroutes set fare=300 where rno=route;
10  else if dist between 250 and 370 then
11  update sroutes set fare=400 where rno=route;
12  else if (dist > 400) then
13  dbms_output.put_line('Sorry');
14  end if;
15  end if;
16  end if;
17  end;
18  /
Enter value for route: 3
old   6: route:=&route;
new   6: route:=3;

PL/SQL procedure successfully completed.

SQL> select * from sroutes;
```

RNO	ORIGIN	DESTINATION	FARE	DISTANCE
2	chennai	dindugal	400	230
3	chennai	madurai	400	300
6	thanjavur	palani	350	370

TO CREATE SCA LCULATE TABLE

```
SQL> create table scalculate ( radius number(3), area number(5,2));
Table created.
SQL> desc scalculate;
Name                            Null?   Type
```

```
                                             NUMBER(3)
       RADIUS                                NUMBER(5.2)
       AREA

    SQL> set serveroutput on;

    SQL> declare
      2  pi constant number(4,2):=3.14;
      3  area number(5,2);
      4  radius number(3);
      5  begin
      6  radius:=3;
      7  while (radius <=7)
      8  loop
      9  area:= pi* power(radius,2);
     10  insert into scalculate values (radius,area);
     11  radius:=radius+1;
     12  end loop;
     13  end;
     14  /

    PL/SQL procedure successfully completed.

    SQL> select * from scalculate;
       RADIUS     AREA
    --------- ---------
         3    28.26
         4    50.24
         5    78.5
         6    113.04
         7    153.86
```

TO CALCULATE FACTORIAL OF A GIVEN NUMBER

```
SQL> set serveroutput on;
SQL> declare
  2  f number(4):=1;
  3  i number(4);
  4  begin
  5  i:=&i;
  6  while(i>=1)
  7  loop
  8  f:=f*i;
  9  i:=i-1;
 10  end loop;
 11  dbms_output.put_line('The value is ' || f);
 12  end;
 13  /
Enter value for i: 5
old   5: i:=&i;
new   5: i:=5;
The value is 120

PL/SQL procedure successfully completed.
```

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.

```sql
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(-20001,'Cannot delete parent record. child records exist in EMPLOYEE table!');
   END IF;
END;
```

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;
```

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 V_total FROM employee;
  IF (V_total + :NEW.Salary) > V_threshold THEN
    RAISE APPLICATION_ERROR (-20003, "Total Salary
              exceeds the allowed
                    threshold.");
  END IF;
END;
```

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 employee_audit(
emp_id NUMBER,
old_salary NUMBER,
NEW_Salary NUMBER,
Change_date Date,
Change_by VARCHAR2(30)
);

CREATE OR REPLACE TRIGGER trg-audit_Salary
                                        change
AFTER UPDATE OF Salary ON employee
FOR EACH ROW
BEGIN
     INSERT INTO employee-audit (emp_id, old_sala
new_Salary, change-date, changed_by)
  VALUES(: OLD.emp_id ,: OLD.salary ,: 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 VARCHAR 2(50)
    Operation_type VARCHAR(20),
    user_name VARCHAR 2 (30),
    activity_date DATA.
);

CREATE OR REPLACE TRIGGER
trg_user_activity
AFTER INSERT OR UPDATE OR DELETE ON
employee
BEGIN
    INSERT_type, user_name, activity_date)
    VALUER ('EMPLOYEE', ORA_SYSEVEWT, 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 TABLE Sales (
    Salaryid NUMBER,
    amount NUMBER,
    running - total NUMBER
);

CREATE OR REPLACE TRIGGER
trg-update-running-total
AFTER INSERT ON Sales
FOR EACH ROW
DECLARE
    V-total NUMBER;
BEGIN
    SELECT NUL (SUM (amount), 0) INTO. V-total
FROM Sales;
    UPDATE Sales SET running-total = V-total
WHERE Sale-id =: NEW. Sale-id;
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.

```sql
CREATE OR REPLACE TRIGGER
trg_ck_Stock_avabability
BEFORE INSERT ON order
FOR EACH ROW
DECLARE
  V_stock NUMBER;
BEGIN
  SELECT quantity_in_stock INTO V_stock FROM
  RAISE_APPLICATION_ERROR (-20004);
  Insufficient Stock available for the requested
                                        item;);

END IF;
END;
/
```

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