

Ex.No.: 13

Date:

12/11/25

WORKING WITH TRIGGER 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.

TABLE CREATION:

```
create table employeebonus(empno number(5)constraint emppk primary key, empname
varchar2(25)not null, experience number(2)not null, bonus number(7,2));
```

Table created.

TRIGGER CREATION FOR BONUS CALCULATION:

```
SQL> set serveroutput on
SQL> create or replace trigger employeebonus_tgr
after insert on employeebonus
declare
cursor emp is select * from employeebonus;
emprec employeebonus%rowtype;
begin
```

```
open emp;
loop
fetch emp into emprec;
exit when emp%notfound;
if(emprec.experience<5) then
emprec.bonus:=5000;
elsif(emprec.experience>=5 and emprec.experience<8) then
emprec.bonus:=8000;
else
emprec.bonus:=10000;
end if;
update employeebonus set bonus=emprec.bonus where empno=emprec.empno;
end loop;
close emp;
dbms_output.put_line('Bonus calculated and Updated Sucessfully');
end;
/
```

Trigger created.

TABLE DESCRIPTION:

SQL> desc employeebonus;

Name Null? Type

EMPNO NOT NULL NUMBER(5)
EMPNAME NOT NULL VARCHAR2(25)
EXPERIENCE NOT NULL NUMBER(2)
BONUS NUMBER(7,2)

RECORD INSERTION:

SQL> insert into employeebonus(empno,empname,experience)
values(&empno,'&empname',&experience);

Enter value for empno: 101

Enter value for empname: murugan

Enter value for experience: 25

old 1: insert into employeebonus(empno,empname,experience)

```
values(&empno,'&empname'.&experience)
new 1: insert into employeebonus(empno,empname,experience)
values(101,'murugan',25)
Bonus calculated and Updated Successfully
1 row created.
```

RECORD SELECTION:

```
SQL> select * from employeebonus;
EMPNO EMPNAME EXPERIENCE BONUS
-----
101 murugan 25 10000
102 suresh 3 5000
103 akash 7 8000
104 mahesh 2 5000
```

RESULT:

Thus, the above program was Created and Executed Successfully.

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.

```
DELIMITER
CREATE OR REPLACE TRIGGER Prevent_parent_delete
BEFORE DELETE ON departments
FOR EACH ROW
DECLARE
    child_count NUMBER;
BEGIN
    SELECT COUNT(*) INTO child_count FROM child_table
    WHERE parent_id = :OLD.parent_id;
    IF child_count > 0 THEN
        RAISE_APPLICATION_ERROR(-20001,'cannot delete');
    END IF;
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 Check_duplicate_email
BEFORE INSERT OR UPDATE ON students
FOR EACH ROW
DECLARE
    dup_count NUMBER;
BEGIN
    SELECT COUNT(*) INTO dup_count FROM students
    WHERE email = :NEW.email
    AND ( -- for update, exclude the current record
        from check : NEW.rowid IS NULL OR rowid !=:
        New.rowid);
    IF dup_count > 0 THEN
        RAISE_APPLICATION_ERROR(-20002,'Duplicate email
        value not allowed');
    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 restrict_insert
BEFORE INSERT ON mytable
FOR EACH ROW
DECLARE
    total_amt NUMBER;
    threshold NUMBER := 1000; -- set your value
BEGIN
    SELECT NVL(SUM(amount), 0) INTO total_amt
    FROM my-table;
    IF (total_amount + :NEW.amount) > threshold THEN
        RAISE_APPLICATION_ERROR(-20001, 'exceeds');
    END IF;
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 employee-audit (
    audit_id NUMBER GENERATED ALWAYS AS
    IDENTITY PRIMARY KEY,
    employeeid NUMBER,
    old_salary NUMBER,
    new_salary NUMBER,
    old_department_id NUMBER,
    changed_on TIMESTAMP DEFAULT CURRENT_TIMESTAMP);
CREATE OR REPLACE TRIGGER trg_employee-audit
AFTER UPDATE OF salary ON employees
FOR EACH ROW
BEGIN
    IF :OLD.salary <> :NEW.salary THEN
        INSERT INTO employee-audit (employee_id, new_salary)
        VALUES (:NEW.employee_id, :NEW.salary);
    END IF;
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 audit_log (
    table-name VARCHAR(50),
    operation VARCHAR(10));
CREATE OR REPLACE TRIGGER trg_employees-audit
AFTER INSERT OR UPDATE OR DELETE ON employees
BEGIN
    INSERT INTO audit_log (table-name, operation)
    VALUES ('EMPLOYEES',
        CASE
            WHEN INSERTING THEN 'INSERT'
            WHEN UPDATING THEN 'UPDATE'
            WHEN DELETING THEN 'DELETE'
        END);
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_running-total
BEFORE INSERT ON my-table
FOR EACH ROW
DECLARE
    total NUMBER;
BEGIN
    SELECT NVL(SUM(amount), 0)
```

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 trg-validate-order
BEFORE INSERT ON Orders
FOR EACH ROW
DECLARE
    Stock NUMBER;
BEGIN
    SELECT stock INTO Stock FROM items WHERE item_id
    = :NEW.item_id;
    IF :NEW.quantity > Stock THEN
        RAISE_APPLICATION_ERROR(-20001, 'Insufficient Stock');
    END IF;
END;

```

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

X
X
B

RESULT: Thus all the above PL/SQL statements are executed.