ASSIGNMENT NO.08

TITLE: PL/SQL TRIGGERS: ROW LEVEL AND STATEMENT LEVEL TRIGGERS, BEFORE AND AFTER TRIGGERS

PROBLEM STATEMENT: Write a database trigger on Library table. The System should keep track of the records that are being updated or deleted. The old value of updated or deleted records should be added in Library_Audit table.

Write a before trigger for Insert, update event considering following requirement:
Emp(e_no, e_name, salary)
I) Trigger action should be initiated when salary is tried to be inserted is less than Rs. 50,000/-
II) Trigger action should be initiated when salary is tried to be updated for value less than Rs. 50,000/-
Action should be rejection of update or Insert operation by displaying appropriate error message. Also the new values expected to be inserted will be stored in new table Tracking(e_no, salary).

OBJECTIVES:

- To learn how to create and test DML trigger
- To identify when triggers should be used.

OUTCOMES:

- Students will be able to create and test DML trigger.
- Students will be able to identify the need and when to use triggers.

SOFTWARE & HARDWARE REQUIREMENTS:

1. 64-bit Open source Linux or its derivative
2. Oracle Database Server

THEORY:

Trigger is invoked by Oracle engine automatically whenever a specified event occurs. Trigger is stored into database and invoked repeatedly, when specific condition match.
Triggers are stored programs, which are automatically executed or fired when some event occurs. Triggers are 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 could be defined on the table, view, schema, or database with which the event is associated.

TRIGGER SYNTAX:

```
CREATE OR REPLACE TRIGGER <TriggerName>
{BEFORE, AFTER}
{DELETE, INSERT, UPDATE [OF COLUMN,.....]}
ON <TABLE NAME>
[FOR EACH ROW]
DECLARE
        BEGIN
        EXCEPTION
END;
```

Types of PL/SQL Triggers

There are two types of triggers based on which level it is triggered.

1. ROW LEVEL TRIGGER: ROW trigger fire for each and every record which are performing INSERT, UPDATE, DELETE from the database table.

For example, if an UPDATE statement updates multiple rows of a table, a row trigger is fired once for each row affected by the UPDATE statement. If a triggering statement affects no rows, a row trigger is not run.

2. STATEMENT LEVEL TRIGGER: Statement trigger fire only once for each statement.

For example, if a DELETE statement deletes several rows from a table, a statement-level DELETE trigger is fired only once.

We can use a statement trigger to:

- Make a complex security check on the current time or user
- Generate a single audit record

COMBINATION TRIGGER: Combination trigger are combination of two trigger types

Before Statement Trigger: Trigger fire only once for each statement before the triggering DML statement.

Before Row Trigger: Trigger fire for each and every record before the triggering DML statement.

After Statement Trigger: Trigger fire only once for each statement after the triggering DML statement executing.

After Row Trigger: Trigger fire for each and every record after the triggering DML statement executing.

PL/SQL Trigger Execution Hierarchy

The following hierarchy is followed when a trigger is fired.

1) BEFORE statement trigger fires first.

2) Next BEFORE row level trigger fires, once for each row affected.

3) Then AFTER row level trigger fires once for each affected row. This events will alternates between BEFORE and AFTER row level triggers.

4) Finally the AFTER statement level trigger fires.

Example:

```
CREATE or REPLACE TRIGGER price_history_trigger
BEFORE UPDATE OF unit_price
ON product
FOR EACH ROW
BEGIN
INSERT INTO product_price_history
VALUES
(:old.product_id,
:old.product_name,
:old.supplier_name,
:old.unit_price);
END
```

Let's update the price of a product.

```
UPDATE PRODUCT SET unit_price = 800 WHERE product_id = 100
```

Once the above update query is executed, the trigger fires and updates the 'product_price_history' table with old values of product_id, product_name,supplier_name and unit_price.

Example:

```
create or replace trigger t4 before insert on emp for each row
declare
begin
if :new.e_no>200 then
raise_application_error(-20003,'eno should not be greater than 200');
end if;
end;
```

The above before insert trigger will prevent the user from inserting a record where employee number greater than 200.

CONCLUSION: