Assignment No .7

Title of Assignment: Database Trigger (All Types: Row level and Statement level triggers, Before and AfterTriggers).

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.

Note: Instructor will Frame the problem statement for writing PL/SQLblock for all types of Triggers in line with above statement.

Course Objective:

Implement PL/SQL Code block for given requirements

Course Outcome:

C306.4 Implement PL/SQL Code block for given requirements

Software Required: - Mysql

PL/SQL Trigger

Trigger is invoked by Oracle engine automatically whenever a specified event occurs.Trigger is stored intodatabase 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.

Advantages of Triggers

These are the following advantages of Triggers:

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

Creating a trigger:

Syntax for creating trigger:

1. **CREATE** [OR REPLACE] **TRIGGER** trigger_name
2. {BEFORE | **AFTER** | **INSTEAD OF** }
3. {**INSERT** [OR] | **UPDATE** [OR] | **DELETE**}

4. [**OF** col_name]

5. **ON** table_name
6. [REFERENCING OLD **AS** o NEW **AS** n]
7. [**FOR** EACH ROW]
8. **WHEN** (condition)
9. **DECLARE**
10. Declaration-statements
11. **BEGIN**
12. Executable-statements
13. EXCEPTION
14. Exception-handling-statements
15. **END**;

Here,

- CREATE [OR REPLACE] TRIGGER trigger_name: It creates or replaces an existing trigger with the trigger_name.
- {BEFORE | AFTER | INSTEAD OF} : This specifies when the trigger would 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 would 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, like INSERT, UPDATE, and DELETE.
- [FOR EACH ROW]: This specifies a row level trigger, i.e., the trigger would 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.
- he price of a product changes constantly. It is important to maintain the history of the prices of the products.
- We can create a trigger to update the 'product_price_history' table when the price of the product is updated in the 'product' table.

1) Create the 'product' table and 'product_price_history' table

```
CREATE TABLE product_price_history
(product_id number(5),

product_name varchar2(32),
```

```
supplier_name varchar2(32),
unit_price number(7,2) );
```

```
CREATE TABLE product
(product_id number(5),
product_name varchar2(32),
supplier_name varchar2(32),
unit_price number(7,2) );
```

2) Create the price_history_trigger and execute it.

```
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;

/
```

3) Lets 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.

4)If you ROLLBACK the transaction before committing to the database, the data inserted to the table is also rolled back.

Types of PL/SQL Triggers

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

1) Row level trigger - An event is triggered for each row upated, inserted or deleted.

2) Statement level trigger - An event is triggered for each sql statement executed.

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.

For Example: Let's create a table 'product_check' which we can use to store messages when triggers are fired.

```
CREATE TABLE product
(Message varchar2(50),
Current_Date number(32)

);
```

Let's create a BEFORE and AFTER statement and row level triggers for the product table.

1) BEFORE UPDATE, Statement Level: This trigger will insert a record into the table 'product_check' before a sql update statement is executed, at the statement level.

```
CREATE or REPLACE TRIGGER Before_Update_Stat_product
BEFORE

UPDATE ON product
Begin

INSERT INTO product_check
Values('Before update, statement
level',sysdate);END;

/
```

2) BEFORE UPDATE, Row Level: This trigger will insert a record into the table 'product_check' before each row is updated.

```
CREATE or REPLACE TRIGGER Before_Upddate_Row_product
BEFORE

UPDATE ON product
FOR EACH ROW
BEGIN
```

```
INSERT INTO product_check
Values('Before update row level',sysdate);
END;
/
```

3) AFTER UPDATE, Statement Level: This trigger will insert a record into the table 'product_check' after a sql update statement is executed, at the statement level.

```
CREATE or REPLACE TRIGGER After_Update_Stat_product
AFTER

UPDATE ON product
BEGIN

INSERT INTO product_check
Values('After update, statement level',
sysdate);End;

/
```

4) AFTER UPDATE, Row Level: This trigger will insert a record into the table 'product_check' after each row is updated.

```
CREATE or REPLACE TRIGGER After_Update_Row_product
AFTER

insert On product
FOR EACH ROW
BEGIN

INSERT INTO product_check
Values('After update, Row level',sysdate);
END;

/
```

Now lets execute a update statement on table product.

```
UPDATE PRODUCT SET unit_price = 800
WHERE product_id in (100,101);
```

Lets check the data in 'product_check' table to see the order in which the trigger is fired.

```
SELECT * FROM product_check;
```

Output:

Mesage	Current_Date
Before update, statement level	26-Nov-2008

Before update, row level 26-Nov-2008

After update, Row level 26-Nov-2008

Before update, row level 26-Nov-2008

After update, Row level 26-Nov-
2008After update, statement level 26-Nov-2008

Conclusion: We have implemented all types of Triggers successfully.

Activity to be Submitted by Students

1. Write pl/sql code in Trigger not to accept the existing Empno (Unique no)
2. Write pl/sql code using Trigger to salary with more than old salary