**USMAN INSTITUTE OF TECHNOLOGY**

**Department of Computer Science**

**CS311 Introduction to Database Systems**

Lab#10

**Objective:**

**-** **DATABASE TRIGGERS (I)**

**Name of Student: \_\_Mansoor Ahmed\_\_\_\_\_\_\_\_\_\_\_\_**

**Roll No: \_\_\_\_18B-129-SE\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_Sec. \_\_\_\_\_A\_\_\_\_\_**

**Date of Experiment: \_\_\_\_\_24-January-2021\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_**

**Marks Obtained/Remarks: \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_**

**Signature: \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_**

**What is a trigger?**

A

*Trigger* is a PL/SQL block that executes implicitly whenever a particular event takes place. A trigger can be either a database trigger or an application trigger.

*Database triggers* execute implicitly when an INSERT, UPDATE, or DELETE statement is issued against the associated table, no matter which user is connected or which application is used.

*Application triggers* execute implicitly whenever a particular event occurs within an application. An example of an application that uses triggers extensively is one developed with Developer/2000 Form Builder.

**Note**: Database triggers can be defined only on tables, not on views. However, if a DML operation is issued against a view, triggers on the base table(s) of a view are fired.

**Guidelines for designing triggers**

* Only use database triggers for centralized, global operations that should be fired for the triggering statement, regardless of which user or application issues the statement.
* Do not define triggers to implement integrity rules that can be done by using declarative constraints.
* The excessive use of triggers can result in complex interdependencies, which may be difficult to maintain in large applications. Only use triggers when necessary, and beware of recursive and cascading effects.

**Database Trigger Types**

The trigger type determines the number of times the trigger action is to be executed: once for every row affected by the triggering statement (such as a multiple row UPDATE), or once for the triggering statement no matter how many rows it affects.

**Statement Trigger**

A statement trigger is fired once on behalf of the triggering event, even if no rows are affected at all. Statement triggers are useful if the trigger action does not depend on data of rows that are affected or data provided by the triggering event itself. For example, a trigger that performs a complex security check on the current user.

**Row Trigger**

A Row trigger fires each time the table is affected by the triggering event. If the triggering event affects no row(s), a row trigger is not executed at all.

Row triggers are useful if the trigger action depends on data of rows that are affected or data provided by the triggering event itself.

**Creating Statement Triggers**

**Syntax for creating Statement Triggers**

CREATE [OR REPLACE] TRIGGER *trigger\_name*

*Timing* *event1* [OR *event2* OR *event3*] ON *table\_name*

PL/SQL block;

**Trigger Components**

Before coding the trigger block, decide on the components of the trigger:- Trigger timing: BEFORE or AFTER

Triggering event: INSERT or UPDATE or DELETE

Table Name: ON table

Trigger Type: Row or Statement

Trigger body: DECLARE

BEGIN

END;

**Trigger Timing**

Indicates the time when the trigger fires in relation to the triggering event: BEFORE or AFTER.

BEFORE Triggers

This type of trigger is frequently used in the following situations:

* When the trigger action should determine whether that triggering statement should be allowed to complete. This allows to eliminate unnecessary processing of the triggering statement and its eventual rollback in cases where an exception is raised in the triggering action.
* To derive column values before completing a triggering INSERT or UPDATE statement. AFTER Triggers

This type of trigger is frequently used in the following situations:

* When the triggering statement is to be completed before executing the triggering action.
* If a BEFORE trigger is already present, and an after trigger can perform different actions on the same triggering statement.

**Triggering Event**

The triggering event or statement can be an INSERT, UPDATE, or DELETE statement on a table.

* When the triggering event is an UPDATE, we can include a column list to identify which column(s) must be changed to fire the trigger. We cannot specify a column list for an INSERT or for a DELETE statement, as they always affect entire rows.
* The triggering event can contain multiple DML statements. In this way, we can differentiate what code to execute depending on the statement that caused the triggers to fire.

**Trigger Body**

The trigger action defines what needs to be done when the triggering event is issued. It can contain SQL and PL/SQL statements, define PL/SQL constructs such as variables, cursors, exceptions and so on.

Additionally row triggers have access to the old and new column values of the row being processed by the trigger, using correlation names.

The trigger body is defined with an anonymous PL/SQL block.

[DECLARE]

BEGIN

[EXCEPTION]

END;

**Before Statement Trigger:**

We can create a *BEFORE statement* trigger in order to prevent the triggering operation from succeeding if a certain condition is violated.

For example, create a trigger to restrict inserts into the EMP table to certain business hours on Monday through Friday. If a user attempted to insert a row into the EMP table on Saturday, for example, the user will see the message, the trigger will fail, and the triggering statement will be rolled back.

RAISE\_APPLICATION\_ERROR is a server-side built-in procedure that prints a message to the user and causes the PL/SQL block to fail. When a database trigger fails, the triggering statement is automatically rolled back by the Oracle Server.

Creating Before Insert Trigger:

**CREATE OR REPLACE TRIGGER secure\_emp**

BEFORE INSERT ON emp

BEGIN

IF (TO\_CHAR(sysdate, 'DY') IN ('SAT', 'SUN'))

OR (TO\_CHAR(sysdate, 'HH24') NOT BETWEEN '08' AND '18') THEN RAISE\_APPLICATION\_ERROR (-20000, 'You may only insert into EMP during normal hours.');

END IF;

**END;**

**Using Conditional Predicates**

We can combine several triggering events into one by taking advantage of the special conditional predicates INSERTING, UPDATING, and DELETING within the trigger body. For example, create one trigger to restrict all data manipulation events on the EMP table to certain business hours, Monday through Friday. Also use BEFORE statement triggers to initialize global variables or flags, and to validate complex business rules.

CREATE OR REPLACE TRIGGER secure\_emp

BEFORE INSERT OR UPDATE OR DELETE ON emp

BEGIN

IF (TO\_CHAR(sysdate, 'DY') IN ('SAT', 'SUN'))

OR (TO\_CHAR(sysdate, 'HH24') NOT BETWEEN '08' AND '18') THEN

IF DELETING THEN

RAISE\_APPLICATION\_ERROR (-20502, 'You may only delete from EMP during normal hours.');

ELSIF INSERTING THEN

RAISE\_APPLICATION\_ERROR (-20500, 'You may only insert into EMP during normal hours.');

ELSIF UPDATING(‘SAL’) THEN

RAISE\_APPLICATION\_ERROR (-20503, 'You may only update SAL during normal hours.');

ELSE

RAISE\_APPLICATION\_ERROR (-20504, 'You may only update EMP during normal hours.');

END IF;

END IF;

END;

**After Statement Trigger:**

We can create an *AFTER Statement* trigger in order to audit the triggering operation or perform a calculation after an operation has completed.

Suppose we have a user defined audit table that lists users and counts their data manipulation operations. After any user has updated the SAL column in the EMP table, use the audit table to ensure that the number of salary changes does not exceed the maximum permitted for that user.

**User Audit Table**

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **USER\_ NAME** | **TABLE\_NAME** | **COLUMN\_NAME** | **INS** | **UPD** | **DEL** | **MAX\_INS** | **MAX\_UPD** | **MAX\_DEL** |
| SCOTT | EMP |  | 1 | 1 | 1 | 5 | 5 | 5 |
| SCOTT | EMP | SAL |  | 1 |  |  | 5 |  |
| SCOTT | EMP |  | 0 | 0 | 0 | 5 | 0 | 0 |

**After Update Trigger:**

**CREATE OR REPLACE TRIGGER check\_salary\_count**

AFTER UPDATE OF sal ON emp

DECLARE

v\_salary\_changes NUMBER;

v\_max\_changes NUMBER;

BEGIN

SELECT upd, max\_upd

INTO v\_salary\_changes, v\_max\_changes

FROM audit\_table

WHERE user\_name = user

AND tablename = 'EMP'

AND column\_name = 'SAL';

IF v\_salary\_changes > v\_max\_changes THEN

RAISE\_APPLICATION\_ERROR (-20501, 'You may only make a maximum of '

|| to\_char(v\_max\_changes) || ' changes to the sal column'); END IF; END;

**After Delete Trigger: Example**

Creating an Audit Table:

CREATE TABLE orders\_audit1\_delete

( order\_id number(5),

quantity number(4),

cost\_per\_item number(6,2),

total\_cost number(8,2),

username varchar(50),

datetrans date,

detail varchar(30)

);

Creating After Delete Trigger:

CREATE OR REPLACE TRIGGER orders\_after\_delete1

AFTER delete

ON orders

FOR EACH ROW

DECLARE

v\_username varchar2(10);

details varchar(30);

BEGIN

details:='record has been deleted'|| TO\_CHAR(:old.order\_id);

-- Find username of person performing the INSERT into the table

SELECT user INTO v\_username

FROM dual;

-- Insert record into audit table

INSERT INTO orders\_audit1\_delete

( order\_id,

quantity,

cost\_per\_item,

total\_cost,

username,datetrans,detail )

VALUES

( :old.order\_id,

:old.quantity,

:old.cost\_per\_item,

:old.total\_cost,

v\_username,sysdate,details );

END;

**Disable/Enable a database trigger**

ALTER TRIGGER *trigger\_name* *DISABLE* | *ENABLE*;

**Removing a Trigger**

DROP TIGGER *trigger\_name*;

**EXERCISE**

1. What are triggers? Differentiate between database triggers and row triggers.

Triggers are defined as saved programs that are automatically run if such events, such as CREATE, ALTER, UPDATE, INSERT, DELETE, occur. They may be defined in the database, table, display that is correlated with the occurrence.

1. Differentiate between Statement and Row triggers?

| **Row Level Triggers** | **Statement Level Triggers** |
| --- | --- |
| Row level triggers executes once for each and every row in the transaction. | Statement level triggers executes only once for each single transaction. |
| Specifically used for data examining purpose. | Used for applying all additional security on the transactions performed on the table. |
| “FOR EACH ROW” clause is present in CREATE TRIGGER command. | “FOR EACH ROW” clause is omitted in CREATE TRIGGER command. |
| If multiple rows are to be inserted into a table, the row level trigger would execute multiple times. | If multiple rows are to be inserted into a table, the statement level trigger would execute only once. |

1. What is meant by triggering event? Give examples.

A SQL comment, database event, or user event that triggers a trigger to fire is a triggering event or statement. One or more of the following may be a triggering event:

* On a particular table, an INSERT, UPDATE, or DELETE statement (or view, in some cases)
* A CREATE, Modify, or DROP declaration for any object in a schema
* Start of a database or shutdown of instances
* A particular message of error or any message of error

... UPDATE OF parts\_on\_hand ON inventory ...

... INSERT OR UPDATE OR DELETE OF inventory ...

1. Give three examples of a situation when a BEFORE statement trigger is needed?

CREATE TRIGGER bt BEFORE UPDATE OR DELETE OR INSERT ON sal

CREATE TRIGGER rt BEFORE UPDATE OR DELETE OR INSERT ON sal

... BEFORE DELETE OR INSERT OR UPDATE OF ename ON Emp\_tab ...

**EXAMPLE 1:**

CREATE TRIGGER SALARY\_CHECK

BEFORE INSERT OR UPDATE OF SALARY, DOCTOR\_ID ON DOCTOR

FOR EACH ROW

WHEN (NEW.DOCTOR\_ID <> 'AD\_VP')

CALL CHECK\_SALARY (: NEW.DOCTOR\_ID,:NEW.SALARY, :NEW DOCTOR NAME)

**EXAMPLE 2:**

Create trigger SCHEMA.TRIGGERNAME BEFORE

DELETE OR UPDATE OR INSERT ON

SCHEMA.TABLE\_NAME PL/SQL\_BLOCK

**EXAMPLE 3:**

CREATE OR REPLACE TRIGGER emp\_alert\_trig

BEFORE INSERT ON emp

BEGIN

DBMS\_OUTPUT.PUT\_LINE('New employees are about to be added');

END;

1. Give three examples of a situation when a AFTER statement trigger is needed?

CREATE TRIGGER at AFTER UPDATE OR DELETE OR INSERT ON sal

create trigger pt after update on p for each row begin

CREATE OR REPLACE TRIGGER Foo AFTER CREATE ON DATABASE

**EXAMPLE 1:**

CREATE TABLE empauditlog (

audit\_date DATE,

audit\_user VARCHAR2(20),

audit\_desc VARCHAR2(20)

);

CREATE OR REPLACE TRIGGER emp\_audit\_trig

AFTER INSERT OR UPDATE OR DELETE ON emp

DECLARE

v\_action VARCHAR2(20);

BEGIN

IF INSERTING THEN

v\_action := 'Added employee(s)';

ELSIF UPDATING THEN

v\_action := 'Updated employee(s)';

ELSIF DELETING THEN

v\_action := 'Deleted employee(s)';

END IF;

INSERT INTO empauditlog VALUES (SYSDATE, USER,

v\_action);

END;

**EXAMPLE 2:**

CREATE OR REPLACE TRIGGER trigger\_test\_as\_trg

AFTER INSERT OR UPDATE OR DELETE ON trigger\_test

BEGIN

trigger\_test\_api.g\_tab.extend;

CASE

WHEN INSERTING THEN

trigger\_test\_api.g\_tab(trigger\_test\_api.g\_tab.last) := 'AFTER STATEMENT - INSERT';

WHEN UPDATING THEN

trigger\_test\_api.g\_tab(trigger\_test\_api.g\_tab.last) := 'AFTER STATEMENT - UPDATE';

WHEN DELETING THEN

trigger\_test\_api.g\_tab(trigger\_test\_api.g\_tab.last) := 'AFTER STATEMENT - DELETE';

END CASE;

FOR i IN trigger\_test\_api.g\_tab.first .. trigger\_test\_api.g\_tab.last LOOP

DBMS\_OUTPUT.put\_line(trigger\_test\_api.g\_tab(i));

END LOOP;

trigger\_test\_api.g\_tab.delete;

END trigger\_test\_as\_trg;

**EXAMPLE 3:**

create or replace trigger log\_salary\_increase

after update on emp\_table

for each row

when (new.sal > 1000)

begin

insert into emp\_log (emp\_id, join\_date, new\_salary, position)

values (:new.empid, sysdate, :new.sal, 'new sal');

end;