# DATABASE TRIGGERS

Appendix A

# INTRODUCTION

# INTRODUCTION

- *trigger* is a PL/SQL block that executes automatically whenever a particular event takes place.
- As already discussed that a trigger can be either a database trigger or an application trigger.

## DATABASE TRIGGER

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

## APPLICATION TRIGGER

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.

# Database Trigger Types

# Database Trigger Types

## STATEMENT TRIGGER

This trigger will only fire once irrespective of the number of rows affected

By default trigger is always statement Trigger

We cant use Co-relational identifiers(**:new & :old**) in Statement triggers

They will always fire even no rows are affected

## ROW TRIGGER

It will fire **as many times** as **Number of records are affected**

We use ***For Each*** syntax to perform the same action as many times as the records are affected

We can use Co-relational identifiers

They will not fire if no rows would get affected.

# Creating Statement Trigger

# Creating statement trigger

- 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

| | |
|---|---|
| Trigger timing: | BEFORE or AFTER |
| Triggering event: | INSERT or UPDATE or DELETE |
| Table Name: | ON table |
| Trigger Type: | Row or Statement |
| Trigger body: | DECLARE<br>BEGIN<br>END; |

# Creating statement trigger

## TRIGGER TIMINGS

**Before Trigger**: when Trigger action determine weather trigger statement should be allowed to complete or not.

In case of exception occurs in triggering action rollback occurs in the triggering statement

**After Trigger**: When the triggering statement is to be completed before executing the triggering action.

If **Before** trigger is already present then **After** trigger can be used to perform others actions with in the same triggering statement

## TRIGGERING EVENT

Triggering event or Triggering statement can be insert, update or delete statements on a particular table

# Creating statement trigger

## TRIGGERING EVENT (cont.)

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

we include a column list to identify which column(s) must be changed to fire the trigger only When the triggering event is an UPDATE. We do not 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
Row triggers have Additionally Co-relational identifiers to access old and new column values of the row being processed by the trigger.

# Creating statement trigger

## TRIGGER BODY

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

```
[DECLARE]
BEGIN
[EXCEPTION]
END;
```

## BEFORE TRIGGER STATETMENT

As already discussed if we want to check a statement before any Action(DML) to be perform on a table.

***Example*** :create a trigger to restrict ***insertion*** into the EMP table to certain business hours(8AM – 6PM) on Monday to Friday only. If a user attempted to insert a row into the EMP table on off timings or on Saturday or Sunday then user will see the message and the triggering statement
will be rolled back.

# Creating statement trigger

## BEFORE TRIGGER STATE. (cont.)

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.

Lets write the trigger to handle the problem stated in example question:

```
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;
```

# Creating statement trigger

## BEFORE TRIGGER STATE. (cont.)

```
  1  CREATE OR REPLACE TRIGGER secure_emp
  2  BEFORE INSERT ON emp
  3  BEGIN
  4  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.');
  5  END IF;
  6* END;
  7  /

Trigger created.
```

Lets Test our trigger after changing my day to a date which have Friday and 3 PM on which a successful insert must occur. Now after setting time lets insert the smartest employee Muzammil :

```
Insert into EMP(EMPNO,ENAME,JOB,MGR,HIREDATE,SAL,COMM,DEPTNO) values
(1211,'Muzammil','Analyst',7811,to_timestamp('26-JAN-99','DD-MON-RR HH.MI.SSXFF AM'),9000,null,30);
SQL> /

1 row created.
```

Record Successfully inserted

Now let change our day to Saturday and insert another talented employee Alina

# Creating statement trigger

## BEFORE TRIGGER STATE. (cont.)

Now let change our day to Saturday and insert another talented employee Alina

```
Insert into EMP(EMPNO,ENAME,JOB,MGR,HIREDATE,SAL,COMM,DEPTNO)
values (9696,'Alina','Analyst',7811,to timestamp('21-JAN-99','DD-MON-RR HH.MI.SSXFF AM'),9000,null,30)
ERROR at line 1:
ORA-20000: You may only insert into EMP during normal hours.
ORA-06512: at "SCOTT.SECURE_EMP", line 2
ORA-04088: error during execution of trigger 'SCOTT.SECURE_EMP'
```

## Using Conditional Predicates

Conditional predicates are **DELETING**, **INSERTING** and **UPDATING** keywords that will return true if respective Trigger statement is present.

Lets write another trigger implementing conditional Predicates that will automatically call respective code of the trigger belonging to the statement that is being performed on a table.

# Creating statement trigger

***Example*** :create a trigger to restrict ***insertion***,***Updation*** and ***deletion*** into the EMP table to certain business hours(8AM – 6PM) on Monday to Friday only. If a user attempted to insert,update or delete a row into the EMP table on off timings or on Saturday or Sunday then user will see the respective different message for respective different cases and the triggering statements will be rolled back.

```
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;
```

# Creating statement trigger

```
SQL> ed
Wrote file afiedt.buf

  1  CREATE OR REPLACE TRIGGER secure_emp
  2  BEFORE INSERT OR UPDATE OR DELETE ON emp
  3  BEGIN
  4  IF (TO_CHAR(sysdate, 'DY') IN ('SAT', 'SUN')) OR (TO_CHAR(sysdate, 'HH24') NOT BETWEEN '08' AND '18') THEN
  5    IF DELETING THEN
  6      RAISE_APPLICATION_ERROR (-20502, 'You may only delete from EMP during normal hours.');
  7    ELSIF INSERTING THEN
  8      RAISE_APPLICATION_ERROR (-20500, 'You may only insert into EMP during normal hours.');
  9    ELSIF UPDATING('SAL') THEN
 10      RAISE_APPLICATION_ERROR (-20503, 'You may only update SAL during normal hours.');
 11    ELSE
 12      RAISE_APPLICATION_ERROR (-20504, 'You may only update EMP during normal hours.');
 13  END IF;
 14  END IF;
 15* END;
 16  /

Trigger created.
```

# Creating statement trigger

We havnt change the date yet so its Saturday lets update Muzammil name to “**Muzammil A**”. It must not allow us to do so.

```
  1  update emp
  2  set ename = 'Muzammil Arif'
  3* where empno=1211
SQL> /
update emp
       *
ERROR at line 1:
ORA-20504: You may only update EMP during normal hours.
ORA-06512: at "SCOTT.SECURE_EMP", line 10
ORA-04088: error during execution of trigger 'SCOTT.SECURE_EMP'
```

Now lets change our date and time to Monday date official time that is between 8 AM to 6 PM. Then run query and see response.

```
SQL> ed
Wrote file afiedt.buf

  1  update emp
  2  set ename = 'Muzammil A'
  3* where empno=1211
SQL> /

1 row updated.
```

Trigger will function accordingly to delete statements and that is the advantage of Conditional predicates

# Creating statement trigger

## AFTER TRIGGER STATEMENT

To practically understand this lets create a Audit_table with following dummy data already present in a table to restrict Maximum updates or deletion on a particular column of a particular table after

| USER_ NAME | TABLE_NAME | COLUMN_NAME | INS | UPD | DEL | MAX_INS | MAX_UPD | MAX_DEL |
|---|---|---|---|---|---|---|---|---|
| SCOTT | EMP | Ename | 1 | 1 | 1 | 5 | 5 | 5 |
| SCOTT | EMP | SAL | | 1 | | | 5 | |
| SCOTT | EMP | Comm | 0 | 0 | 0 | 5 | 0 | 0 |

We can create this table by following DDL:

```
CREATE TABLE AUDIT_TABLE
(
  USER_NAME VARCHAR2(30) , TABLE_NAME VARCHAR2(30) , COLUMN_NAME VARCHAR2(30)
, INS NUMBER(1) , UPD NUMBER(1) , DEL NUMBER(1) , MAX_INS NUMBER(1) , MAX_UPD NUMBER(1) , MAX_DEL NUMBER(1)
);
Insert into SCOTT.AUDIT_TABLE (USER_NAME,TABLE_NAME,COLUMN_NAME,INS,UPD,DEL,MAX_INS,MAX_UPD,MAX_DEL) values
('SCOTT','EMP','Ename',1,1,1,5,5,5);
Insert into SCOTT.AUDIT_TABLE (USER_NAME,TABLE_NAME,COLUMN_NAME,INS,UPD,DEL,MAX_INS,MAX_UPD,MAX_DEL) values
('SCOTT','EMP','SAL',null,1,null,null,5,null);
Insert into SCOTT.AUDIT_TABLE (USER_NAME,TABLE_NAME,COLUMN_NAME,INS,UPD,DEL,MAX_INS,MAX_UPD,MAX_DEL) values
('SCOTT','EMP','Comm',0,0,0,5,0,0);
```

# Creating statement trigger

AFTER TRIGGER STATEMENT

***Example*** :create a trigger to restrict no of updates to ***Sal*** column of employee table to a value defined in the ***MAX_UPD*** column of a audit table.

```
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;
```

# Creating statement trigger

## AFTER TRIGGER STATEMENT

```
 1  CREATE OR REPLACE TRIGGER check_salary_count
 2  AFTER UPDATE OF sal ON emp
 3  DECLARE
 4    v_salary_changes NUMBER;
 5    v_max_changes NUMBER;
 6  BEGIN
 7  SELECT UPD, MAX_UPD INTO v_salary_changes, v_max_changes FROM audit_table
 8  WHERE USER_NAME = user AND TABLE_NAME = 'EMP' AND column_name = 'SAL';
 9  IF v_salary_changes > v_max_changes THEN
10  RAISE_APPLICATION_ERROR (-20501, 'You may only make a maximum of '|| to_char(v_max_changes) || ' changes to the sal column');
11  END IF;
12* END;
13  /

Trigger created.
```

We are feeling another trigger’s need which will update our audit table on every Insert, update and Delete. So we will create now a Row Trigger lets study Row trigger.

# Creating Row Trigger

# Creating Row trigger

- Syntax for creating Row Triggers

CREATE [OR REPLACE] TRIGGER ***trigger_name***
***Timing event1*** [OR *event2* OR *event3*]
ON ***table_name***
***FOR EACH ROW***
[WHEN condition]
PL/SQL block;

This is a new part that is representing It will fire **as many times** as **Number of records are affected**

Lets understand **after row** trigger statement with practical example:

***Example*** :create a trigger that will track out all inserts and deletes in *employee table* and increment ***del*** column value and ***ins*** column value in *audit_table*. Increment ***upd*** column value for any column update or for sal column update[will give 2 different conditions to cater this].

# Creating Row trigger

## AFTER ROW TRIGGER

```
CREATE OR REPLACE TRIGGER audit_emp
AFTER DELETE OR INSERT OR UPDATE ON emp
FOR EACH ROW
BEGIN
	IF DELETING THEN
		UPDATE audit_table SET del = del + 1 WHERE user_name = user AND table_name = 'EMP' AND column_name IS NULL;
	ELSIF INSERTING THEN
		UPDATE audit_table SET ins = ins + 1 WHERE user_name = user AND table_name = 'EMP' AND column_name IS NULL;
	ELSIF UPDATING('SAL') THEN
		UPDATE audit_table SET upd = upd + 1 WHERE user_name = user AND table_name = 'EMP' AND column_name = 'SAL';
	ELSE
		UPDATE audit_table SET upd = upd + 1 WHERE user_name = user AND table_name = 'EMP' AND column_name IS NULL;
	END IF;
END;
```

# Creating Row trigger

## AFTER ROW TRIGGER (cont)

```
 1  CREATE OR REPLACE TRIGGER audit_emp
 2  AFTER DELETE OR INSERT OR UPDATE ON emp
 3  FOR EACH ROW
 4  BEGIN
 5  IF DELETING THEN
 6  UPDATE audit_table SET del = del + 1 WHERE user_name = user AND table_name = 'EMP' AND column_name IS NULL;
 7  ELSIF INSERTING THEN
 8  UPDATE audit_table SET ins = ins + 1 WHERE user_name = user AND table_name = 'EMP' AND column_name IS NULL;
 9  ELSIF UPDATING('SAL') THEN
10  UPDATE audit_table SET upd = upd + 1 WHERE user_name = user AND table_name = 'EMP' AND column_name = 'SAL';
11  ELSE
12  UPDATE audit_table SET upd = upd + 1 WHERE user_name = user AND table_name = 'EMP' AND column_name IS NULL;
13  END IF;
14* END;
15  /

Trigger created.
```

Before updating ‘**Muzamil A**’ salary let view audit table record:

```
SQL> select * from audit_table;

USER_NAME  TABLE_NAME  COLUMN_NAME     INS    UPD    DEL   MAX_INS    MAX_UPD    MAX_DEL
---------- ----------- ------------ ----- ------ ------ -------- ---------- ----------
SCOTT      EMP         Ename            1      1      1        5          5          5
SCOTT      EMP         SAL                     1                          5
SCOTT      EMP         Comm             0      0      0        5          0          0
```

```
SQL> update emp set sal = 10000 where empno = 1211;

1 row updated.
```

# Creating Row trigger

## AFTER ROW TRIGGER (cont)

With or even without commit u will see ur audit_table has been updated now lets check it

```
SQL> select * from audit_table;

USER_NAME   TABLE_NAME   COLUMN_NAME          INS        UPD        DEL    MAX_INS    MAX_UPD    MAX_DEL
---------- ------------ ------------------- ---------- ---------- ---------- ---------- ---------- ----------
SCOTT       EMP          Ename                    1          1          1          5          5          5
SCOTT       EMP          SAL                                 2                                     5
SCOTT       EMP          Comm                     0          0          0          5          0          0
```

Trigger Updated my **audit_table** as soon as I update **emp** table

In case I Rollback, it will definitely reverse both table values means salary of '**Muzamill A**' and above value of UPD column back to 1 of **audit_table**

## Using Old and New Qualifiers

In order to see real essence of triggers let us first create a table named as AUDIT_EMP_VALUES. Which will track the change in **employee name**, **employee job** and its **salary**.

# Creating Row trigger

## Using Old and New Qualifiers (cont)

```
SQL> ed
Wrote file afiedt.buf

  1  CREATE TABLE AUDIT_EMP_VALUES
  2  (
  3    USER_NAME VARCHAR2(30)
  4  , TIMESTAMP DATE
  5  , EMPID NUMBER(4)
  6  , OLD_NAME VARCHAR2(30)
  7  , NEW_NAME VARCHAR2(30)
  8  , OLD_JOB VARCHAR2(30)
  9  , NEW_JOB VARCHAR2(30)
 10  , OLD_SAL NUMBER(10)
 11  , NEW_SAL NUMBER(10)
 12* )
 13  /

Table created.
```

Now create a trigger which will use OLD and NEW Quantifiers and record old (previous) values to new(updated) value on any DML operation. We write Quantifier after colon(:) as **:OLD.Columnname** or **:NEW.Columnname**

```
CREATE OR REPLACE TRIGGER audit_emp_values
AFTER DELETE OR INSERT OR UPDATE ON emp
FOR EACH ROW
BEGIN
        INSERT INTO audit_emp_values (user_name, timestamp, EMPID,
        OLD_NAME,NEW_NAME,OLD_JOB,NEW_JOB, OLD_SAL, NEW_SAL)
        VALUES (USER, SYSDATE, :old.empno, :old.ename, :new.ename, :old.job,
        :new.job,:old.sal, :new.sal);
END;
```

**On every insert update and Delete in emp table there Will be a log(insert a value) In audit_emp_values table**

# Creating Row trigger

## Using Old and New Qualifiers (cont)

```
SQL> ed
Wrote file afiedt.buf

  1  CREATE OR REPLACE TRIGGER audit_emp_values
  2  AFTER DELETE OR INSERT OR UPDATE ON emp
  3  FOR EACH ROW
  4  BEGIN
  5  INSERT INTO audit_emp_values (USER_NAME,TIMESTAMP, EMPID, OLD_NAME,NEW_NAME,OLD_JOB,NEW_JOB, OLD_SAL, NEW_SAL)
  6  VALUES (USER, SYSDATE, :old.empno, :old.ename, :new.ename, :old.job, :new.job,:old.sal, :new.sal);
  7* END;
SQL> /

Trigger created.
```

Lets update a single row

```
SQL> update emp set sal = 11000 where empno= 1211;

1 row updated.

SQL> commit;

Commit complete.
```

# Creating Row trigger

## Using Old and New Qualifiers (cont)

```
SQL> select * from audit_emp_values;

USER_NAME  TIMESTAMP      EMPID OLD_NAME    NEW_NAME     OLD_JOB      NEW_JOB      OLD_SAL    NEW_SAL
---------- --------- ---------- ----------- ----------- --------- ------------- --------- ---------- ---
SCOTT      24-OCT-16       1211 Muzammil A  Muzammil A   Analyst      Analyst        10000      11000
```

This Log is clearly stating salary has been changed

**By MR SCOTT at specified time**

Lets insert a new employee **Maha** and check the record again :

```
  1* Insert into EMP(EMPNO,ENAME,JOB,MGR,HIREDATE,SAL,COMM,DEPTNO) values (9898,'Maha','MANAGER',7811,to_timestamp('11-FEB-81','DD-MON-RR HH.MI.SSXFF AM'),9000,null,30)
SQL> /

1 row created.
```

For newly inserted record u may see OLD values are null which will clearly indicate u about newly inserted record

```
SQL> select * from audit_emp_values;

USER_NAME  TIMESTAMP   EMPID OLD_NAME    NEW_NAME    OLD_JOB    NEW_JOB    OLD_SAL    NEW_SAL
---------- --------- ------- ----------- ----------- --------- --------- ---------- -------
SCOTT      24-OCT-16    1211 Muzammil A  Muzammil A  Analyst   Analyst      10000      11000
SCOTT      24-OCT-16                     Maha                  MANAGER                  9000
```

# Creating Row trigger

## Before Row Trigger

To restrict the trigger action to those rows that satisfy a certain condition, we check a condition ***before*** trigger action via WHEN clause.

**Example:** **Create a trigger on the EMP table specifically for all 'SALESMAN' to set an employee's commission to 0 and for updation case if old commission was null set it to zero else apply this formula to new commission :**

**New Commission = OLD Comm + ( updated sal / OLD Sal)**

```
CREATE OR REPLACE TRIGGER derive_commission_pct
BEFORE INSERT OR UPDATE OF SAL ON emp
FOR EACH ROW
WHEN (new.job = 'SALESMAN')
BEGIN
        IF INSERTING THEN :new.comm. := 0;
        ELSE /* UPDATE of salary */
                IF :old.comm. IS NULL THEN
                        :new.comm. := 0;
                ELSE
                        :new.comm. := :old.comm. + (:new.sal / :old.sal);
                END IF;
        END IF;
END;
```

**Try this by your self now u are xperts in triggers ☺**

# Triggers Vs Stored Procedures

# Database Trigger Types

## TRIGGERS

Implicitly invoked against some explicit event (DML operation)

COMMIT, ROLLBACK And SAVEPOINT statements are not allowed within the trigger body

## STORED PROCEDURES

Explicitly invoked

COMMIT, ROLLBACK And SAVEPOINT statements are permitted within the procedure body

# DROP OR DISABLE TRIGGERS

# Creating Row trigger

## DISABLE TRIGGER

**Disable/Enable a database trigger in case doing any problem:**

ALTER TRIGGER *trigger_name DISABLE | ENABLE*;

## DROP TRIGGER

**Removing a Trigger**

DROP TIGGER *trigger_name*;

# THANK YOU FOR WATCHING!

!!!Take Care !!!