Oracle for Developers (PL/SQL)

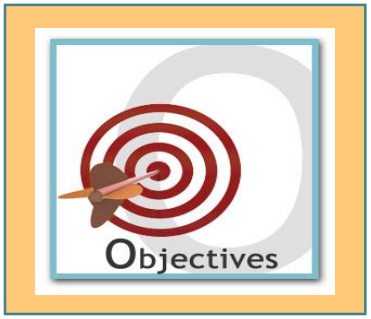
Database Triggers

Capgemini

Lesson Objectives

To understand the following topics:

- Concept of Database Triggers
- Types of Triggers
- Disabling and Dropping Triggers
- Restriction on Triggers
- Order of Trigger firing
- Using :Old and :New values, WHEN clause, Trigger predicates



Concept of Database Triggers

Database Triggers:

- Database Triggers are procedures written in PL/SQL, Java, or C that run (fire) implicitly:
  - whenever a table or view is modified, or
  - when some user actions or database system actions occur
- They are stored subprograms.

Concept of Database Triggers

- You can write triggers that fire whenever one of the following operations occur:
  - User events:
    - DML statements on a particular schema object
    - DDL statements issued within a schema or database
    - user logon or logoff events
  - System events:
    - server errors
    - database startup
    - database shutdown

Usage of Triggers

- Triggers can be used for:
  - maintaining complex integrity constraints.
  - auditing information, that is the Audit trail.
  - automatically signaling other programs that action needs to take place when changes are made to a table.

Syntax of Triggers

Syntax:

```
CREATE or  REPLACE TRIGGER Trg_Name
{BEFORE | AFTER}
{event} OF Column_Names
ON Table_Name
[FOR EACH ROW]
[WHEN restriction]
BEGIN
        PL/SQL statements;
END Trg_Name ;
```

Types of Triggers

Type of Trigger is determined by the triggering event, namely:

- INSERT
- UPDATE
- DELETE

- Triggers can be fired:
  - before or after the operation.
  - on row or statement operations.
- Trigger can be fired for more than one type of triggering statement.

More on Triggers

Category	Values	Comments
Statement	INSERT, DELETE, UPDATE	Defines which kind of DML statement causes the trigger to fire.
Timing	BEFORE, AFTER	Defines whether the trigger fires before the statement is executed or after the statement is executed.
Level	Row or Statement	• If the trigger is a row-level trigger, it fires once for each row affected by the triggering statement. • If the trigger is a statement-level trigger, it fires once, either before or after the statement. • A row-level trigger is identified by the FOR EACH ROW clause in the trigger definition.

Example 1

```
CREATE TABLE Account_log
(
deleteInfo VARCHAR2(20),
logging_date DATE
)
```

Trigger creation code

```
CREATE or REPLACE TRIGGER

After_Delete_Row_product

 AFTER   delete On Account_masters
 FOR EACH ROW
 BEGIN
 INSERT INTO  Account_log
 Values('After delete, Row level',sysdate);
 END;
```

Restrictions on Triggers

The use of Triggers has the following restrictions:

- Triggers should not issue transaction control statements (TCL) like COMMIT, SAVEPOINT.
- Triggers cannot declare any long or long raw variables.
- :new and :old cannot refer to a LONG datatype.

Disabling and Dropping Triggers

To disable a trigger:

```
ALTER  TRIGGER Trigger_Name DISABLE/ENABLE
```

To drop a trigger (by using drop trigger command):

```
DROP TRIGGER Trigger_Name
```

Order of Trigger Firing

Order of Trigger firing is arranged as:

- Execute the “before statement level” trigger.
- For each row affected by the triggering statement:
  - Execute the “before row level” trigger.
  - Execute the statement.
  - Execute the “after row level” trigger.
- Execute the “after statement level” trigger.

Using :Old & :New values in Triggers

Triggering statement	:Old	:New
INSERT	**Undefined – all fields are null.**	**Values that will be inserted when the statement is complete.**
UPDATE	**Original values for the row before the update.**	**New values that will be updated when the statement is complete.**
DELETE	**Original values before the row is deleted.**	**Undefined – all fields are NULL.**

- **Note:** They are valid only within row level triggers and not in statement level triggers.

Using WHEN clause

Use of WHEN clause is valid for row-level triggers only.

Trigger body is executed for rows that meet the specified condition.

Example 2

```
CREATE TABLE Account_masters
(
account_no NUMBER(6) PRIMARY KEY,
cust_id NUMBER(6) ,
account_type CHAR(3) CONSTRAINT chk_acc_type CHECK(account_type IN
('SAV','SAL')) ,
Ledger_balance NUMBER(10)
)
```

Trigger creation code

```
CREATE OR REPLACE TRIGGER trg_acc_master_ledger
   before INSERT OR UPDATE OF Ledger_balance ,account_type  ON
Account_masters
   FOR EACH ROW

   WHEN (NEW.account_type ='SAV')
   DECLARE

    v_led_bal NUMBER(10);
   BEGIN
    v_led_bal:= :NEW.Ledger_balance;

    IF v_led_bal < 5000 THEN
     :NEW.Ledger_balance := 5000;
    END if;
   END trg_acc_master_ledger;
```

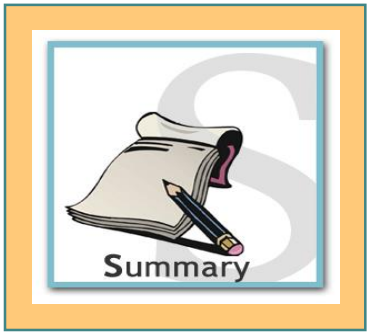
Summary

Database Triggers are procedures written in PL/SQL, Java, or C that run (fire) implicitly: Database Triggers are procedures written in PL/SQL, Java, or C that run (fire) implicitly:

- whenever a table or view is modified, or
- when some user actions or database system actions occur

There are three types of triggers:

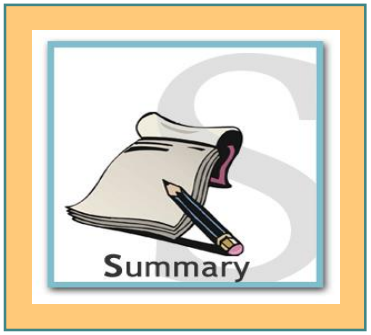
- Statement based triggers
- Timing based triggers
- Level based triggers



Summary

Disabling and Dropping triggers can be done instead of actually removing the triggers

Order of trigger firing is decided depending on the type of triggers used in the sequence



Review – Questions

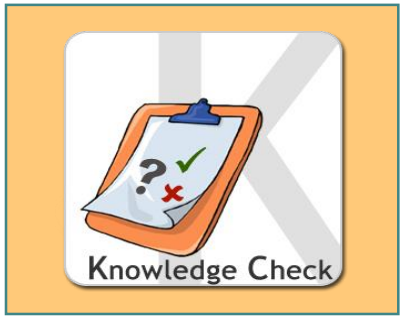
Question 1: Triggers should not issue Transaction Control Statements (TCL).

- True / False

Question 2: BEFORE DROP and AFTER DROP triggers are fired when a schema object is dropped.

- True / False

Question 3: The :new and :old records must be used in WHEN clause with a colon.



Review – Questions

Question 4: A ___ is a table that is currently being modified by a DML statement.

Question 5: A ___ is fired once on behalf of the triggering statement, regardless of the number of rows in the table that the triggering statement affects.