DATABASE TRIGGERS

- Database trigger – a stored PL/SQL program unit that is associated with a specific database table, or with certain view types – can also be associated with a system event such as database startup.
- Two sections:
  - A named database event
  - A PL/SQL block that will execute when the event occurs
- Triggers execute (fire) automatically for specified SQL DML operations – INSERT, UPDATE, or DELETE affecting one or more rows of a table.

DATABASE TRIGGERS -tasks

- Database triggers can be used to perform any of the following tasks:
  - Audit data modification.
  - Log events transparently.
  - Enforce complex business rules.
  - Derive column values automatically.
  - Implement complex security authorizations.
  - Maintain replicate tables.
  - Publish information about events for a publish-subscribe environment such as that associated with web programming.

- Triggers:
  - are named PL/SQL blocks with declarative, executable, and exception handling sections.
  - are stand-alone database objects – they are not stored as part of a package and cannot be local to a block.
  - do not accept arguments.

- To create/test a trigger, you (not the system user of the trigger) must have appropriate access to all objects referenced by a trigger action.

- Example: To create a BEFORE INSERT trigger for the *employee* table requires you to have INSERT ROW privileges for the table.

Create Trigger Syntax

```
CREATE [OR REPLACE] TRIGGER trigger_name
{BEFORE|AFTER|INSTEAD OF} triggering_event
  [referencing_clause] ON {table_name | view_name}
[WHEN condition] [FOR EACH ROW]
[DECLARE
    Declaration statements]
BEGIN
    Executable statements
[EXCEPTION
    Exception-handling statements]
END;
```

The trigger body must have at least the executable section.

- The declarative and exception handling sections are optional.
- When there is a declarative section, the trigger body must start with the DECLARE keyword.
- The WHEN clause specifies the condition under which a trigger should fire.

Trigger Types

- BEFORE and AFTER Triggers – trigger fires before or after the triggering event. Applies only to tables.
- INSTEAD OF Trigger – trigger fires instead of the triggering event. Applies only to views.
- Triggering_event – a DML statement issued against the table or view named in the ON clause – example: INSERT, UPDATE, or DELETE.
- DML triggers are fired by DML statements and are referred to sometimes as row triggers.
- FOR EACH ROW clause – a ROW trigger that fires once for each modified row.
- STATEMENT trigger – fires once for the DML statement.
- Referencing_clause – enables writing code to refer to the data in the row currently being modified by a different name.

Conditional Predicates for Detecting Triggering DML Statement

Conditional Predicate	TRUE if and only if:
INSERTING	An INSERT statement fired the trigger.
UPDATING	An UPDATE statement fired the trigger.
UPDATING ('*column*')	An UPDATE statement that affected the specified column fired the trigger.
DELETING	A DELETE statement fired the trigger.

```
SET SERVEROUTPUT On
CREATE OR REPLACE TRIGGER t
  BEFORE
    INSERT OR
    UPDATE OF salary, deptno OR
    DELETE   ON emp
BEGIN
  CASE
    WHEN INSERTING THEN
      DBMS_OUTPUT.PUT_LINE('Inserting');
    WHEN UPDATING('salary') THEN
      DBMS_OUTPUT.PUT_LINE('Updating salary');
    WHEN UPDATING(‘deptno') THEN
      DBMS_OUTPUT.PUT_LINE('Updating department ID');
    WHEN DELETING THEN
      DBMS_OUTPUT.PUT_LINE('Deleting');
  END CASE;
END;
/
```

ROW Trigger – Accessing Rows

- Access data on the row currently being processed by using two correlation identifiers named **:old** and **:new**. These are special Oracle bind variables.
- The PL/SQL compiler treats the **:old** and **:new** records as records of type trigger_Table_Name%ROWTYPE.
- To reference a column in the triggering table, use the notation shown here where the *ColumnName* value is a valid column in the triggering table.

```
:new.ColumnName
:old.ColumnName
```

Empno	Ename	Sal
100	Raj	129399

:old.sal is 123999

Empno	Ename	Sal
100	Raj	116000

:new.sal is 116000

```
SET SERVEROUTPUT On
CREATE OR REPLACE TRIGGER t1
  BEFORE INSERT OR UPDATE OF salary, deptno OR
    DELETE ON emp FOR EACH ROW
BEGIN
  CASE
    WHEN INSERTING THEN
      DBMS_OUTPUT.PUT_LINE('Inserting '|| :NEW.EMPNO||:NEW.SAL);
    WHEN UPDATING('sal') THEN
      DBMS_OUTPUT.PUT_LINE('Updating salary'||:OLD.SAL||'---'||:NEW.SAL);
    WHEN UPDATING('DEPTNO') THEN
      DBMS_OUTPUT.PUT_LINE('Updating department ID'||:OLD.DEPTNO||:NEW.DEPTNO);
    WHEN DELETING THEN
      DBMS_OUTPUT.PUT_LINE('Deleting');
      DBMS_OUTPUT.PUT_LINE(:OLD.EMPNO||'--'||:OLD.ENAME||'--'||:OLD.SAL||'--
'||:OLD.DEPTNO);
  END CASE;
END;
```

Bind Variables :old and :new Defined

DML Statement	:old	:new
INSERT	Undefined – all column values are NULL as there is no "old" version of the data row being inserted.	Stores the values that will be inserted into the new row for the table.
UPDATE	Stores the original values for the row being updated before the update takes place.	Stores the new values for the row – values the row will contain after the update takes place.
DELETE	Stores the original values for the row being deleted before the deletion takes place.	Undefined – all column values are NULL as there will not be a "new" version of the row being deleted.

Example

```
CREATE TABLE Emp_log (
  Emp_id      NUMBER(4),
  Log_date    DATE,
  New_salary NUMBER(7,2),
  Action      VARCHAR2(20));
```

Example-

```
CREATE OR REPLACE TRIGGER log_salary_increase
  AFTER UPDATE OF salary ON employee
  FOR EACH ROW
BEGIN
  INSERT INTO Emp_log (Emp_id, Log_date, New_salary,
Action)
  VALUES (:NEW.empno, SYSDATE, :NEW.salary, ‘Insert
New Salary');
END;
/
```

Example-Statement Trigger

Create the following table and do not insert records.

```
CREATE TABLE users_log ( User_name varchar2(10),Operation varchar2(10), Login_Date Date ) ;

CREATE OR REPLACE TRIGGER note_hr_logoff_trigger
  BEFORE LOGOFF
  ON mca2020.SCHEMA
BEGIN
  INSERT INTO users_log VALUES (USER, 'LOGOFF', SYSDATE);
END;
/
```

Example-Statement Trigger

Create the following table and do not insert records.

```
CREATE TABLE users_log ( User_name varchar2(10),Operation varchar2(10), Login_Date Date ) ;
```

```
create or replace trigger emp_sal_update before update of sal on emp
begin
if to_char(sysdate,'DY') = 'SUN' then
raise_application_error(-20111,'No changes can be made on sunday.');
else
dbms_output.put_line(' Today is not SUNDAY, Let us WOrk');
end if;
end;
/
```

t.sql

Dropping a Trigger

- The DROP TRIGGER statement drops a trigger from the database.
- If you drop a table, all associated table triggers are also dropped.
- The syntax is:

```
DROP TRIGGER trigger_name;
```