Lesson-04 : Database Triggers and its Types
==

What is a Trigger?
=================

An EVENT which leads to ACTION is termed as a TRIGGER.
There are basically TWO types of triggers:
[1] Application - Triggers when an event occurs in application
[2] Database - Triggers when an event occurs in a database

What are Database Triggers?
==========================

Database triggers are STORED PROCEDURES that are IMPLICITLY executed when an triggering event occurs.

DB Triggers are associated with tables, views or other database objects.

The trigger event could be:
• DML statements on the table
• DDL statements
• System events such as startup, shutdown, and error
• User events such as logon and logoff

Business App Scenarios for Implementing Triggers
~~~~~~

Refer PPTs

Available DB Trigger Types
==========================

Simple DML triggers:
- BEFORE
- AFTER
- INSTEAD OF

Compound triggers:
- Non-DML triggers
- DDL event triggers
- Database event triggers

Creating DML Triggers (Parts of a Trigger)
=====================

A triggering statement contains:
• Trigger timing
  – For table: BEFORE, AFTER
  – For view: INSTEAD OF
• Triggering event: INSERT, UPDATE, or DELETE
• Table name: On table, view
• Trigger type: Row or statement
• WHEN clause: Restricting condition
• Trigger body: PL/SQL block OR call to a procedure
            Determines what action is performed

DML Trigger Components
=====================

Trigger timing: When should the trigger fire?
• BEFORE: Execute the trigger body before the triggering DML event
          on a table.
• AFTER: Execute the trigger body after the triggering DML event on
          a table.
~~~~~~

• INSTEAD OF: Execute the trigger body instead of the triggering statement. This is used for views that are not otherwise modifiable.

Triggering user event: Which DML statement causes the trigger to execute?
You can use any of the following:
• INSERT
• UPDATE
• DELETE

Trigger type: Should the trigger body execute for each row the statement affects or only once?
• Statement: The trigger body executes once for the triggering event.
  This is the default.
  A statement trigger fires once, even if no rows are affected at all.
• Row: The trigger body executes once for each row affected by the triggering event.
  A row trigger is not executed if the triggering event affects no rows.

Trigger body: What action should the trigger perform?
The trigger body is a PL/SQL block or a call to a procedure.

Statement-Level v/s Row-Level Triggers
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
Refer PPT

Firing Sequence: Single Row Manipulation
==
Use the following firing sequence for a trigger on a table, when a single row is manipulated:

```
DML Statement
INSERT INTO departments (department_id, department_name, location_id)
VALUES (400, 'Consulting', 2400);

Triggering Action
                             <<- BEFORE Statement Trigger
------------- ------------------------------ -----------
DEPARTMENT_ID DEPARTMENT_NAME                LOCATION_ID
------------- ------------------------------ -----------
          240 Government Sales                      1700
          250 Retail Sales                          1700
          260 Recruiting                            1700
                             <<- BEFORE Row Trigger
------------- ------------------------------ -----------
          400 Consulting                   2400
------------- ------------------------------ -----------
                             <<- AFTER Row Trigger
                             <<- AFTER Statement Trigger
```

Firing Sequence: Multi-Row Manipulation
==
Refer PPT
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~

```
Creating DML Statement Triggers
===============================
Syntax:

CREATE [OR REPLACE] TRIGGER trigger_name
  timing
     event1 [OR event2 OR event3]
        ON table_name
trigger_body

NOTE: Trigger names must be unique with respect to other triggers in the
      same schema.

Example:
CREATE OR REPLACE TRIGGER secure_emp
      BEFORE -- Trigger Timing
      INSERT -- The event
            ON employees

BEGIN
      IF    (TO_CHAR(SYSDATE, 'DY') IN ('SAT', 'SUN')) OR
            (TO_CHAR(SYSDATE, 'HH24:MI') NOT BETWEEN '08:00' AND '18:00')
      THEN
            RAISE_APPLICATION_ERROR(-20500,
                  'You may insert into EMPLOYEES table only during
business hours..');
      END IF;
END; -- End of trigger

Using Conditional Predicates
============================
Conditional Predicates are names given for certain conditions.
They are implicitly defined in the Oracle DB Server.

Example: Refer 'secure_emp2' trigger

Creating a DML Row Trigger
==========================
To create a DML row trigger, the syntax is as follows:

CREATE [OR REPLACE] TRIGGER trigger_name
  timing
    event1 [OR event2 OR event3]
      ON table_name
  [REFERENCING OLD AS old | NEW AS new]
FOR EACH ROW
  [WHEN (condition)]
trigger_body

NOTE: Observe, in the syntax the FOR EACH ROW clause.

The FOR EACH ROW clause, help us in identifying the DB trigger is a
Statement-Level trigger or Row-Level trigger.


Example:
```

Example:

```
CREATE OR REPLACE TRIGGER restrict_salary
      BEFORE                              -- Trigger Timing
      INSERT OR UPDATE OF salary          -- The event
            ON employees
      FOR EACH ROW                        -- ROW Trigger
BEGIN
      -- The salary is restricted to employees other than
      -- 'AD_PRES' and 'AD_VP'
      IF NOT (:NEW.job_id IN ('AD_PRES', 'AD_VP')) AND (:NEW.salary >
15000)
      THEN
            RAISE_APPLICATION_ERROR(-20404,
                  'Employee cannot earn this much amount...');
      END IF;
END; -- End of trigger
```

```
SQL> SELECT employee_id, first_name, salary, job_id FROM employees WHERE
department_id = 110;

EMPLOYEE_ID FIRST_NAME               SALARY JOB_ID
----------- -------------------- ---------- ----------
        205 Shelley                   12500 AC_MGR
        206 William                    8300 AC_ACCOUNT

SQL> UPDATE employees
  2  SET salary = 18000
  3  WHERE employee_id = 206;
UPDATE employees
       *
ERROR at line 1:
ORA-20404: Employee cannot earn this much amount...
ORA-06512: at "HR.RESTRICT_SALARY", line 5
ORA-04088: error during execution of trigger 'HR.RESTRICT_SALARY'
```

Using OLD and NEW Qualifiers

When a row level trigger fires, the PL/SQL run time engine creates and populates two data structures:

* OLD: Stores the original values of the record processed by the trigger
* NEW: Contains the new values

NEW and OLD have the same structure as a record declared using the %ROWTYPE on the table to which the trigger is attached.

Operation	Old Value	New Value
INSERT	NULL	Inserted value
UPDATE	Value before update	Value after update
DELETE	Value before delete	NULL

New and old values of the DML statements can be processed with NEW.column_name and :OLD.column_name in the trigger restriction and trigger action.

Example:

```
Example:
CREATE OR REPLACE TRIGGER audit_emp_values
      AFTER                                         -- Trigger Timing
      INSERT OR DELETE OR UPDATE        -- The event
            ON employees
      FOR EACH ROW                            -- ROW Trigger
BEGIN
      INSERT INTO audit_emp (user_name, timestamp_id,
            id, old_last_name, new_last_name, old_title,
            new_title, old_salary, new_salary)
      VALUES (USER, SYSDATE, :OLD.employee_id, :OLD.last_name,
            :NEW.last_name, :OLD.job_id, :NEW.job_id,
            :OLD.salary, :NEW.salary);
END; -- End of trigger
```

NOTE: The 'audit_emp' table needs to be created.

Once the 'audit_emp' table is created, execute the below SQL statements and observe.

```
SQL> SELECT employee_id, first_name, last_name, salary FROM employees
WHERE department_id = 110;

EMPLOYEE_ID FIRST_NAME           LAST_NAME                     SALARY
----------- -------------------- ------------------------- ----------
        205 Shelley              Higgins                        12500
        206 William              Gietz                           8300

SQL> UPDATE employees
  2  SET last_name = 'Smith', salary = 8800
  3  WHERE employee_id = 206;

1 row updated.

SQL> SELECT employee_id, first_name, last_name, salary FROM employees
WHERE department_id = 110;

EMPLOYEE_ID FIRST_NAME           LAST_NAME                     SALARY
----------- -------------------- ------------------------- ----------
        205 Shelley              Higgins                        12500
        206 William              Smith                           8800

SQL> SELECT * FROM audit_emp;

USER_NAME                      TIMESTAMP         ID OLD_LAST_NAME
------------------------------ --------- ---------- --------------------
----
NEW_LAST_NAME             OLD_TITLE  NEW_TITLE  OLD_SALARY NEW_SALARY
------------------------- ---------- ---------- ---------- ----------
HR                             04-JAN-22        206 Gietz
Smith                     AC_ACCOUNT AC_ACCOUNT       8300       8800
```

Restricting a Row Trigger
=========================
To restrict a row trigger, the WHEN clause is used.

Example:

Example:
```
CREATE OR REPLACE TRIGGER derive_comm_pct
      BEFORE                            -- Trigger Timing
      INSERT OR UPDATE OF salary        -- The event
            ON employees
      FOR EACH ROW                      -- ROW Trigger
      WHEN (NEW.job_id = 'SA_REP') -- Row restriction
BEGIN
      IF INSERTING THEN
            :NEW.commission_pct := 0;
      ELSIF :OLD.commission_pct IS NULL THEN
            :NEW.commission_pct := 0;
      ELSE
            :NEW.commission_pct := :OLD.commission_pct + 0.05;
      END IF;
END; -- End of trigger
```

Now, if we change the salary of any 'SA_REP', then the commission_pct will be updated too.
i.e. If the commission_pct is NULL, then it becomes ZERO. On the other hand if it is not null, then the new commission_pct will be 5% more than the existing commission_pct.

INSTEAD OF Triggers
===================

The INSTEAD OF triggers are associated with VIEWS.
With the INSTEAD OF trigger when we perform a DML, it actually does it with the underlying table.

To create an INSTEAD OF trigger, the syntax is as follows:

```
CREATE [OR REPLACE] TRIGGER trigger_name
  INSTEAD OF
    event1 [OR event2 OR event3]
      ON view_name
[REFERENCING OLD AS old | NEW AS new]
[FOR EACH ROW]
trigger_body
```

Example:
```
CREATE OR REPLACE TRIGGER emp_details_insert
      INSTEAD OF         -- Used on 'emp_details' view
      INSERT ON emp_details
      FOR EACH ROW
BEGIN
  INSERT INTO new_emps (employee_id, last_name, salary, department_id)
  VALUES (:NEW.employee_id, :NEW.last_name, :NEW.salary,
:NEW.department_id);

  UPDATE new_depts
      SET deptsal = (SELECT SUM(salary) FROM new_emps
                     WHERE department_id = :NEW.department_id)
      WHERE department_id = :NEW.department_id;
END;
```

Observe, that when we insert into 'emp_details' view, the actual data INSTEAD OF getting inserted in the view will be inserted in 'new_emps' table and updated in 'new_depts' table

Insert one row in the 'emp_details' table and record your observation.

Status of the Triggers

A trigger is in either of TWO distinct modes:

```
[1] ENABLED: The trigger runs its trigger action if a triggering
             Statement is issued and the trigger restriction (if any)
             evaluates to true (default).
[2] DISABLED: The trigger does not run its trigger action, even if a
             triggering statement is issued and the trigger restriction
             (if any) would evaluate to true.
```

Creating a Disabled Trigger

Before Oracle Database 11g, if you created a trigger whose body had a PL/SQL compilation error, then DML to the table failed.

In Oracle Database 11g, you can create a DISABLEd trigger and then enable it only when you know it will be compiled successfully.

Example:

```
CREATE OR REPLACE TRIGGER mytrg
  BEFORE INSERT ON mytable
  FOR EACH ROW
  DISABLE              <-- Observe the DISABLE clause
BEGIN
 :New.ID := my_seq.Nextval
 . . .
END;
```

Implementing an INTEGRITY CONSTRAINT with AFTER Trigger

When we try to update the 'department_id' in the 'employees' table with an invalid department id (i.e. a non-existing department_id) the Oracle DB Server throws the following error.

```
SQL> UPDATE employees
  2  SET department_id = 112
  3  WHERE employee_id = 113;
UPDATE employees
*
ERROR at line 1:
ORA-02291: integrity constraint (HR.EMP_DEPT_FK) violated - parent key
not
found
```

However, it is possible to implement an Integrity Constraint with AFTER trigger.

Here, we will update the 'department_id' with a non-existing department id in the 'departments' table and AFTER doing so we shall map the new department_id with the new name 'Dept-<no>' in the 'departments' table.

This is achieved by using the AFTER trigger as shown below:

Example:

```
CREATE OR REPLACE TRIGGER emp_dept_fk_trg
      AFTER
      UPDATE OF department_id
            ON employees
      FOR EACH ROW
BEGIN
      INSERT INTO departments
            VALUES (:NEW.department_id, 'Dept-
'||:NEW.department_id,NULL,NULL);
EXCEPTION
      WHEN DUP_VAL_ON_INDEX THEN
            NULL;  -- Do nothing if Deparment exists
END;
```

Once, the 'emp_dept_fk_trg' trigger is created, if we update the 'employees' table with a non-existing 'department_id' it still works and inserts the new details in the 'departments' table.

Managing Triggers
=================

Disable or reenable a database trigger:
```
ALTER TRIGGER trigger_name DISABLE | ENABLE;
```

Disable or reenable all triggers for a table:
```
ALTER TABLE table_name DISABLE | ENABLE ALL TRIGGERS;
```

Recompile a trigger for a table:
```
ALTER TRIGGER trigger_name COMPILE;
```

Dropping Trigger
================

To remove a trigger from the database, use the DROP TRIGGER statement.

Syntax:
```
DROP TRIGGER trigger_name;
```

Example:
```
DROP TRIGGER secure_emp;
```

NOTE: All triggers on a table are dropped when the table is dropped.

Trigger Test Cases | Testing Triggers
=====================================

- Test each triggering data operation, as well as non-triggering data operations.
- Test each case of the WHEN clause.
- Cause the trigger to fire directly from a basic data operation, as well as indirectly from a procedure.
- Test the effect of the trigger upon other triggers.
- Test the effect of other triggers upon the trigger.

Viewing Trigger Information
===========================

You can view the following trigger information:
- USER_OBJECTS data dictionary view: object information
- USER_TRIGGERS data dictionary view: the text of the trigger
- USER_ERRORS data dictionary view: PL/SQL syntax errors (compilation errors) of the trigger

Using USER_TRIGGERS Data Dictionary
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
The description of the USER_TRIGGERS data dictionary is as follows:

```
SQL> desc USER_TRIGGERS
 Name                                      Null?    Type
 ----------------------------------------- -------- ----------------------------
 TRIGGER_NAME                                       VARCHAR2(128)
 TRIGGER_TYPE                                       VARCHAR2(16)
 TRIGGERING_EVENT                                   VARCHAR2(246)
 TABLE_OWNER                                        VARCHAR2(128)
 BASE_OBJECT_TYPE                                   VARCHAR2(18)
 TABLE_NAME                                         VARCHAR2(128)
 COLUMN_NAME                                        VARCHAR2(4000)
 REFERENCING_NAMES                                  VARCHAR2(422)
 WHEN_CLAUSE                                        VARCHAR2(4000)
 STATUS                                             VARCHAR2(8)
 DESCRIPTION                                        VARCHAR2(4000)
 ACTION_TYPE                                        VARCHAR2(11)
 TRIGGER_BODY                                       LONG
 CROSSEDITION                                       VARCHAR2(7)
 BEFORE_STATEMENT                                   VARCHAR2(3)
 BEFORE_ROW                                         VARCHAR2(3)
 AFTER_ROW                                          VARCHAR2(3)
 AFTER_STATEMENT                                    VARCHAR2(3)
 INSTEAD_OF_ROW                                     VARCHAR2(3)
 FIRE_ONCE                                          VARCHAR2(3)
 APPLY_SERVER_ONLY                                  VARCHAR2(3)
```

Now, to get the trigger info, we can query the USER_TRIGGERS table as follows:

```
SQL> SELECT trigger_type, trigger_body
  2  FROM user_triggers
  3  WHERE trigger_name = 'SECURE_EMP';

TRIGGER_TYPE
----------------
TRIGGER_BODY
--------------------------------------------------------------------------------
BEFORE STATEMENT
BEGIN
        IF      (TO_CHAR(SYSDATE, 'DY') IN ('SAT', 'SUN')) OR
                (TO_CHAR(SYSDATE, 'HH
```
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~