Triggers

Oracle Academy Study Materials

Creating Triggers – Trigger Definition

A trigger:

- A trigger is a PL/SQL block that is stored in the database and fired (executed) in response to a specified event.
- The Oracle database automatically (implicitly) executes a trigger when specified conditions occur
- Is associated with a table, view, schema, or database

Creating Triggers – Trigger Event Types

You can write triggers that fire whenever one of the following operations occurs in the database:

- A database manipulation (DML) statement (DELETE, INSERT, or UPDATE).
- A database definition (DDL) statement (CREATE, ALTER, or DROP).
- A database operation such as SERVERERROR, LOGON, LOGOFF, STARTUP, or SHUTDOWN.

You can use triggers for:

- Security
- Auditing
- Data integrity
- Referential integrity
- Table replication
- Computing derived data automatically
- Event logging

Creating Triggers – Trigger Event Types

You can use triggers to:

- Enhance complex database security rules
- Create auditing records automatically
- Enforce complex data integrity rules
- Create logging records automatically
- Prevent tables from being accidentally dropped
- Prevent invalid DML transactions from occurring
- Generate derived column values automatically
- Maintain synchronous table replication
- Gather statistics on table access
- Modify table data when DML statements are issued against views

Creating Triggers – Available Trigger Types

- Simple DML triggers
  - `BEFORE`
  - `AFTER`
  - `INSTEAD OF`
- Compound triggers
- Non-DML triggers
  - DDL event triggers
  - Database event triggers
- A trigger event type determines which DML statement causes the trigger to execute. The possible events are:
  - `INSERT`
  - `UPDATE [OF column]`
  - `DELETE`
- A trigger body determines what action is performed and is a PL/SQL block or a `CALL` to a procedure

Creating database event triggers

```
CREATE TABLE log_table (
  user_id   VARCHAR2(30),
  logon_date  DATE);

CREATE OR REPLACE TRIGGER logon_trigg
AFTER LOGON ON DATABASE
BEGIN
  INSERT INTO log_table (user_id, logon_date)
  VALUES (USER, SYSDATE);
END;
```

Creating DML Triggers

```
CREATE [OR REPLACE] TRIGGER trigger_name
timing -- when to fire the trigger
event1 [OR event2 OR event3]
ON object_name
[REFERENCING OLD AS old | NEW AS new]
FOR EACH ROW -- default is statement level trigger
WHEN (condition)]]
DECLARE]
BEGIN
... trigger_body -- executable statements
[EXCEPTION . . .]
END [trigger_name];
```

```
timing =  BEFORE | AFTER | INSTEAD OF
```

```
event = INSERT | DELETE | UPDATE | UPDATE OF column_list
```

Creating Triggers - Trigger Timing

- `BEFORE`: Execute the trigger body before the triggering DML event on a table and is frequently used in the following situations:
  - To determine whether the triggering statement should be allowed to complete (This eliminates unnecessary processing and enables a rollback in cases where an exception is raised in the triggering action.)
  - To derive column values before completing an INSERT or UPDATE statement
  - To initialize global variables or flags, and to validate complex business rules
- `AFTER`: Execute the trigger body after the triggering DML event on a table and is frequently used in the following situations:
  - To complete the triggering statement before executing the triggering action
  - To perform different actions on the same triggering statement if a BEFORE trigger is already present
- `INSTEAD OF`: Execute the trigger body instead of the triggering statement. This is used for views that are not otherwise modifiable.

Trigger Timings and Events Examples

– The first trigger executes immediately before an employee's salary is updated:

```
CREATE OR REPLACE TRIGGER sal_upd_trigg
BEFORE UPDATE OF salary ON employees
BEGIN ... END;
```

– The second trigger executes immediately after an employee is deleted:

```
CREATE OR REPLACE TRIGGER emp_del_trigg
AFTER DELETE ON employees
BEGIN ... END;
```

Trigger Timings and Events Examples

– You can restrict an UPDATE trigger to updates of a specific column or columns:

```
CREATE OR REPLACE TRIGGER sal_upd_trigg
BEFORE UPDATE OF salary, commission_pct ON employees
BEGIN ... END;
```

– A trigger can have more than one triggering event:

```
CREATE OR REPLACE TRIGGER emp_del_trigg
AFTER INSERT OR DELETE OR UPDATE ON employees
BEGIN ... END;
```

How often and when does a statement trigger fire?

A statement trigger fires only once for each execution of the triggering statement (even if no rows are affected or the triggering DML statement affects many rows).

```
INSERT INTO departments
  (department_id,department_name, location_id)
VALUES (400, 'CONSULTING', 2500);
```

DEPARTMENT_ID	DEPARTMENT_NAME	LOCATION_ID
10	Administration	1700
20	Marketing	1800
50	Shipping	1500

→ **BEFORE statement trigger**

400	CONSULTING	2500

→ **AFTER statement trigger**

```
UPDATE employees
  SET salary = salary * 1.1
  WHERE department_id = 50;
```

EMPLOYEE_ID	LAST_NAME	DEPARTMENT_ID
124	Mourgos	50
141	Rajs	50
142	Davies	50
143	Matos	50

→ **BEFORE statement trigger**

144	Vargas	50

→ **AFTER statement trigger**

Creating Triggers - Trigger Timing

- If multiple triggers are defined for the same object, then the order of firing triggers is arbitrary.
  - If the order in which they fire is important, then you can control the firing order using the FOLLOWS clause.
  - If it is practical, you should consider replacing the set of individual triggers for a particular timing point with a single compound trigger that explicitly codes the actions in the order you intend.

```
CREATE OR REPLACE TRIGGER
trigger_follows_test_trg_1
BEFORE INSERT ON trigger_follows_test
FOR EACH ROW
FOLLOWS trigger_follows_test_trg_2
BEGIN

DBMS_OUTPUT.put_line('TRIGGER_FOLLOWS_TEST_TRG_1
- Executed');
END;
```

Creating Triggers

Creating a DML Statement Trigger Example: SECURE_EMP

```
CREATE OR REPLACE TRIGGER secure_emp
BEFORE INSERT 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 '
     ||' normal business hours.');
   END IF;
  END;
```

```
INSERT INTO employees (employee_id, last_name,
first_name, email, hire_date, job_id, salary, department_id)
VALUES (300, 'Smith', 'Rob', 'RSMITH', SYSDATE,
'IT_PROG', 4500, 60);
```

Using Conditional Predicates

- Suppose you want to perform any DML operation, but with different actions for INSERT, UPDATE and DELETE.
- You could create three separate triggers, however…
- There are trigger keywords DELETING, INSERTING, and UPDATING, which are automatically declared Boolean variables and set to TRUE or FALSE by the Oracle server.

```
CREATE OR REPLACE TRIGGER secure_emp
  BEFORE INSERT OR UPDATE OR DELETE ON employees
BEGIN
  IF TO_CHAR(SYSDATE, 'DY') IN ('SAT', 'SUN') THEN
  IF DELETING THEN RAISE_APPLICATION_ERROR
  (-20501,'You may delete from EMPLOYEES table only during business
hours');
  ELSIF INSERTING THEN RAISE_APPLICATION_ERROR
  (-20502,'You may insert into EMPLOYEES table only during business
hours');
  ELSIF UPDATING THEN RAISE_APPLICATION_ERROR
  (-20503,'You may update EMPLOYEES table only during business hours');
  END IF;
  END IF;
END;
```

Using Conditional Predicates

– Moreover, you can use conditional predicates to test for UPDATE on a specific column:

```
CREATE OR REPLACE TRIGGER secure_emp
  BEFORE UPDATE ON employees
BEGIN
  IF UPDATING('SALARY') THEN
  IF TO_CHAR(SYSDATE, 'DY') IN ('SAT', 'SUN')
  THEN RAISE_APPLICATION_ERROR
    (-20501,'You may not update SALARY on the weekend');
  END IF;
  ELSIF UPDATING('JOB_ID') THEN
  IF TO_CHAR(SYSDATE, 'DY') = 'SUN'
  THEN RAISE_APPLICATION_ERROR
    (-20502, 'You may not update JOB_ID on Sunday');
  END IF;
  END IF;
END;
```

Understanding Row Triggers

- A statement trigger executes only once for each triggering DML statement:

```
CREATE OR REPLACE TRIGGER log_emps
  AFTER UPDATE OF salary ON employees
BEGIN
  INSERT INTO log_emp_table (who, when)
  VALUES (USER, SYSDATE);
END;
```

- Suppose you want to insert one row into the log table for each updated employee.
- For this, you need a row trigger.
- You specify a row trigger using FOR EACH ROW.

```
CREATE OR REPLACE TRIGGER log_emps
  AFTER UPDATE OF salary ON employees FOR EACH ROW
BEGIN
  INSERT INTO log_emp_table (who, when)
  VALUES (USER, SYSDATE);
END;
```

Creating Triggers – Types of DML Triggers

- The trigger type determines whether the body executes for each row or only once for the triggering statement.
- A statement trigger:
  - Executes once for the triggering event
  - Is the default type of trigger
  - Fires once even if no rows are affected at all
- A row trigger:
  - Executes once for each row affected by the triggering event
  - Is not executed if the triggering event does not affect any rows
  - Is indicated by specifying the `FOR EACH ROW` clause

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
- You use `:OLD.column_name` to reference the pre-update value, and `:NEW.column_name` to reference the post-update value.

Data Operations	Old Value	New Value
`INSERT`	`NULL`	Inserted value
`UPDATE`	Value before update	Value after update
`DELETE`	Value before delete	`NULL`

Creating Triggers

Using OLD and NEW Qualifiers

```
CREATE OR REPLACE TRIGGER audit_emp_values
AFTER DELETE OR INSERT OR UPDATE ON employees
FOR EACH ROW
BEGIN
  INSERT INTO audit_emp(user_name, time_stamp, 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;
/
```

Creating a DML Row Trigger

```
CREATE OR REPLACE TRIGGER restrict_salary
BEFORE INSERT OR UPDATE OF salary ON employees
FOR EACH ROW
BEGIN
  IF NOT (:NEW.job_id IN ('AD_PRES', 'AD_VP'))
     AND :NEW.salary > 15000 THEN
    RAISE_APPLICATION_ERROR (-20202,
      'Employee cannot earn more than $15,000.');
  END IF;
END;
/
```

Using the REFERENCING Clause

- Instead of :OLD and :NEW qualifiers, different qualifiers' names can be used by including a REFERENCING clause.
- The aliases for OLD and NEW are called correlation-names.

```
CREATE OR REPLACE TRIGGER log_emps
  AFTER UPDATE OF salary ON old
 REFERENCING OLD as former NEW as latter FOR EACH ROW
BEGIN
  INSERT INTO log_emp_table (who, when, which_employee,
                             old_salary, new_salary)
  VALUES (USER, SYSDATE, :former.employee_id,
                        :former.salary, :latter.salary);
END;
```

Using the WHEN clause

```
CREATE OR REPLACE TRIGGER restrict_salary
 AFTER UPDATE of salary ON employees FOR EACH ROW
BEGIN
 IF :NEW.salary > :OLD.salary THEN
 INSERT INTO
 log_emp_table(who,when,which_employee,old_salary,new_salary)
 VALUES(USER,SYSDATE,:OLD.employee_id,:OLD.salary,:NEW.salary);
 END IF;
END;
```

– IF condition can be coded in the trigger header, just before the BEGIN clause:

```
CREATE OR REPLACE TRIGGER restrict_salary
  AFTER UPDATE of salary ON copy_employees FOR EACH ROW
  WHEN(NEW.salary > OLD.salary)
BEGIN
  INSERT INTO log_emp_table
  (who,when,which_employee,old_salary,new_salary)
  VALUES(USER,SYSDATE,:OLD.employee_id,:OLD.salary,
                                        :NEW.salary);
END;
```

Creating Triggers

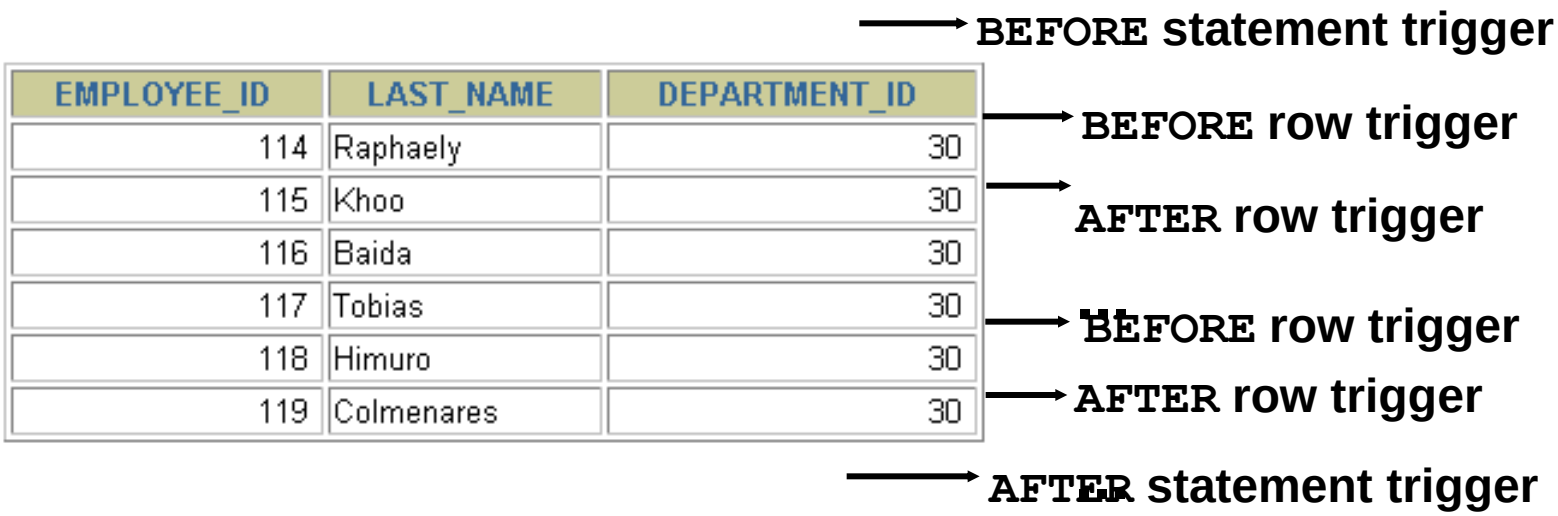
Restricting a Row Trigger: Example

```
CREATE OR REPLACE TRIGGER derive_commission_pct
BEFORE INSERT OR UPDATE OF salary ON employees
FOR EACH ROW
WHEN (NEW.job_id = 'SA_REP')
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;
/
```

Creating Triggers

Trigger-Firing Sequence

```
UPDATE employees
  SET salary = salary * 1.1
  WHERE department_id = 30;
```



Creating Triggers

Implementing an Integrity Constraint with a Trigger

```
UPDATE employees SET department_id = 999
 WHERE employee_id = 170;
-- Integrity constraint violation error
```

```
CREATE OR REPLACE TRIGGER employee_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
```

```
UPDATE employees SET department_id = 999
 WHERE employee_id = 170;
-- Successful after trigger is fired
```

INSTEAD OF Triggers

- Problem: Underlying tables cannot be updated using a complex view (for example a view based on a join).
- Example: Suppose the EMP_DETAILS view is a complex view based on a join of EMPLOYEES and DEPARTMENTS. The following SQL statement fails:

```
INSERT INTO emp_details
  VALUES (9001, 'ABBOTT', 3000, 10, 'Administration');
```

- You can overcome this by creating an INSTEAD OF trigger that updates the underlying tables directly instead of trying (and failing) to update the view.
- Notice: INSTEAD OF triggers are always row triggers.

INSTEAD OF Triggers

```
CREATE TABLE new_emps AS
 SELECT employee_id,last_name,salary,department_id
    FROM employees;

CREATE TABLE new_depts AS
 SELECT d.department_id,d.department_name,
        sum(e.salary) dept_sal
    FROM employees e, departments d
 WHERE e.department_id = d.department_id;

CREATE VIEW emp_details AS
 SELECT e.employee_id, e.last_name, e.salary,
        e.department_id, d.department_name
 FROM employees e, departments d
 WHERE e.department_id = d.department_id
GROUP BY d.department_id,d.department_name;
```

INSTEAD OF Triggers

```
CREATE OR REPLACE TRIGGER new_emp_dept
  INSTEAD OF INSERT ON emp_details
BEGIN
  INSERT INTO new_emps
  VALUES (:NEW.employee_id, :NEW.last_name,
    :NEW.salary, :NEW.department_id);
  UPDATE new_depts
  SET dept_sal = dept_sal + :NEW.salary
  WHERE department_id = :NEW.department_id;
END;
```

DDL and Database Event Triggers

- DDL triggers are fired by DDL statements: CREATE, ALTER or DROP.
- Database Event triggers are fired by non-SQL events in the database, for example:
  - A user connects to, or disconnects from, the database.
  - The DBA starts up, or shuts down, the database.
  - A specific exception is raised in a user session.
- Triggers on DDL Statements Syntax
  - ON DATABASE fires the trigger for DDL on all schemas in the database
  - ON SCHEMA fires the trigger only for DDL on objects in your own schema

```
CREATE [OR REPLACE] TRIGGER trigger_name
Timing
[ddl_event1 [OR ddl_event2 OR ...]]
ON {DATABASE|SCHEMA}
trigger_body
```

DDL and Database Event Triggers

– Example of a DDL Trigger: Write a log record every time a new database object is created in your schema:

```
CREATE OR REPLACE TRIGGER log_create_trigg
AFTER CREATE ON SCHEMA
BEGIN
  INSERT INTO log_table
  VALUES (USER, SYSDATE);
END;
```

– The trigger fires whenever any type of object is created.

– You cannot create a DDL trigger that refers to a specific database object.

DDL and Database Event Triggers

– Example of a DDL Trigger: Prevent any objects being dropped from your schema:

```
CREATE OR REPLACE TRIGGER prevent_drop_trigg
BEFORE DROP ON SCHEMA
BEGIN
  RAISE_APPLICATION_ERROR
  (-20203, 'Attempted drop - failed');
END;
```

– The trigger fires whenever any (type of) object is dropped.

– Again, you cannot create a DDL trigger that refers to a specific database object.

Triggers on Database Events

Guidelines:

- You cannot use INSTEAD OF with database event triggers.
- You can define triggers to respond to such system events as LOGON, SHUTDOWN, and even SERVERERROR.
- Database event triggers can be created ON DATABASE or ON SCHEMA, except that ON SCHEMA cannot be used with SHUTDOWN and STARTUP events.

```
CREATE OR REPLACE TRIGGER logon_trig
AFTER LOGON ON SCHEMA
BEGIN
  INSERT INTO log_trig_table(user_id,log_date,action)
  VALUES (USER, SYSDATE, 'Logging on');
END;
```

```
CREATE OR REPLACE TRIGGER logoff_trig
BEFORE LOGOFF ON SCHEMA
BEGIN
  INSERT INTO log_trig_table(user_id,log_date,action)
  VALUES (USER, SYSDATE, 'Logging off');
END;
```

Triggers on Database Events

– Example: A SERVERERROR Trigger to keep a log of any ORA-00942 errors that occur in your sessions:

```
CREATE OR REPLACE TRIGGER servererror_trig
AFTER SERVERERROR ON SCHEMA
BEGIN
  IF (IS_SERVERERROR (942)) THEN
 INSERT INTO error_log_table ...
  END IF;
END;
```

– If the IS_SERVERERROR … conditional test is omitted, the trigger will fire when any Oracle server error occurs.

Creating Triggers

Comparison of Database Triggers and Stored Procedures

Triggers	Procedures
Defined with CREATE TRIGGER	Defined with CREATE PROCEDURE
Data dictionary contains source code in USER_TRIGGERS.	Data dictionary contains source code in USER_SOURCE.
Implicitly invoked by DML	Explicitly invoked
COMMIT, SAVEPOINT, and ROLLBACK are not allowed.	COMMIT, SAVEPOINT, and ROLLBACK are allowed.

CALL Statements in a Trigger

```
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]
  CALL procedure_name
```

– There is no END; statement, and no semicolon at the end of the CALL statement.

```
CREATE OR REPLACE TRIGGER log_employee
BEFORE INSERT ON EMPLOYEES
  CALL log_execution
```

Mutating Tables and Row Triggers

- A mutating table is a table that is currently being modified by a DML statement.
- A row trigger cannot SELECT from a mutating table, because it would see an inconsistent set of data (the data in the table would be changing while the trigger was trying to read it).
- However, a row trigger can SELECT from a different table if needed.
- This restriction does not apply to DML statement triggers, only to DML row triggers.
- To avoid mutating table errors:
  - A row-level trigger must not query or modify a mutating table.
  - A statement-level trigger must not query or modify a mutating table if the trigger is fired as the result of a CASCADE delete.
  - Reading and writing data using triggers is subject to certain rules. The restrictions apply only to row triggers, unless a statement trigger is fired as a result of ON DELETE CASCADE.

Mutating Tables and Row Triggers

```
CREATE OR REPLACE TRIGGER emp_trigg

  AFTER INSERT OR UPDATE OR DELETE ON employees
    -- EMPLOYEES is the mutating table

  FOR EACH ROW

BEGIN

 SELECT … FROM employees …    -- is not allowed

 SELECT … FROM departments …  -- is allowed

 …

END;
```

Mutating Tables and Row Triggers – Example

```
CREATE OR REPLACE TRIGGER check_salary
  BEFORE INSERT OR UPDATE OF salary, job_id ON
employees
  FOR EACH ROW
DECLARE
  v_minsalary employees.salary%TYPE;
  v_maxsalary employees.salary%TYPE;
BEGIN
 SELECT MIN(salary), MAX(salary)
 INTO v_minsalary, v_maxsalary
 FROM employees
 WHERE job_id = :NEW.job_id;
  IF :NEW.salary < v_minsalary OR
   :NEW.salary > v_maxsalary THEN
   RAISE_APPLICATION_ERROR(-20505,'Out of range');
  END IF;
END;
```