
ORACLE
Academy

Database Programming with PL/SQL

13-4
Creating DDL and Database Event Triggers

ORACLE Academy

Objectives

This lesson covers the following objectives:

- Describe events that cause DDL and database event triggers to fire
- Create a trigger for a DDL statement
- Create a trigger for a database event
- Describe the functionality of the CALL statement
- Describe the cause of a mutating table

Purpose

- What if you accidentally drop an important table?
- If you have a backup copy of the table data, you can retrieve the lost data.
- But it might be important to know exactly when the table was dropped.
- For security reasons, a Database Administrator might want to keep an automatic record of who has logged into a database, and when.
- These are two examples of the uses of DDL and Database Event triggers.

What are 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.

Creating 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
```

Example of a DDL Trigger

- You want to 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.

A Second Example of a DDL Trigger

- You want to 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.

Creating Triggers on Database Events Syntax

- ON DATABASE fires the trigger for events on all sessions in the database.
- ON SCHEMA fires the trigger only for your own sessions.

```
CREATE [OR REPLACE] TRIGGER trigger_name
timing
[database_event1 [OR database_event2 OR ...]]
ON {DATABASE|SCHEMA}
trigger_body
```

Creating Triggers on Database Events Guidelines

- Remember, 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.

Example 1: LOGON and LOGOFF Triggers

```
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;
```

Example 2: A SERVERERROR Trigger

- You want 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.

CALL Statements in a Trigger

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

```
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
```

```
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.

Mutating Tables and 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 Table: 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;
```

Mutating Table: Example

```
UPDATE employees
  SET salary = 3400
  WHERE last_name = 'Davies';
```

```
ORA-04091: table US_NLH2_PLSQL_T01.COPY_EMPLOYEES is mutating, trigger/function may
not see it
ORA-06512: at "US_NLH2_PLSQL_T01.CHECK_SALARY", line 5
ORA-04088: error during execution of trigger 'US_NLH2_PLSQL_T01.CHECK_SALARY'


3.   WHERE last_name = 'Davies';
```

More Possible Uses for Triggers

- You should not create a trigger to do something that can easily be done in another way, such as by a check constraint or by suitable object privileges.
- But sometimes you must create a trigger because there is no other way to do what is needed.
- The following examples show just three situations where a trigger must be created.
- There are many more!

Uses for Triggers: First Example

- Database security (who can do what) is normally controlled by system and object privileges.
- For example, user SCOTT needs to update EMPLOYEES rows:

```
GRANT UPDATE ON employees TO scott;
```

- But privileges alone cannot control when SCOTT is allowed to do this.
- For that, we need a trigger:

```
CREATE OR REPLACE TRIGGER weekdays_emp
  BEFORE UPDATE ON employees
BEGIN
 IF (TO_CHAR (SYSDATE, 'DY') IN ('SAT','SUN')) THEN
 RAISE_APPLICATION_ERROR(-20506,'You may only change data during
   normal business hours.');
 END IF; END;
```

Uses for Triggers: Second Example

- Database integrity (what DML is allowed) is normally controlled by constraints.
- For example, every employee must have a salary of at least $500:

```
ALTER TABLE employees ADD
  CONSTRAINT ck_salary CHECK (salary >= 500);
```

- If a business rule states that employees' salaries can be raised but not lowered, this constraint will not prevent an employee's salary being lowered from $700 to $600.
- For that, we need a row trigger.
- The code for this is shown on the next slide.

Uses for Triggers: Second Example

- Now we don't need the constraint any more.

```
CREATE OR REPLACE TRIGGER check_salary
  BEFORE UPDATE of salary ON employees
  FOR EACH ROW
  WHEN(NEW.salary < OLD.salary OR NEW.salary < 500)
BEGIN
  RAISE_APPLICATION_ERROR (-20508,  'Do not decrease
                                          salary.');
END;
```

Uses for Triggers: Second Example

- Taking this example further, what if the minimum salary changes from time to time? Next year it may be $550, not $500. We don't want to drop and recreate the constraint every time.
- We would create a single-row, single-column table which stores the minimum salary:

```
CREATE TABLE minsal (min_salary  NUMBER(8,2));

INSERT INTO minsal (min_salary) VALUES (500);
```

And later, if the minimum salary changes to $550, we simply:

```
UPDATE minsal
SET min_salary = 550;
```

Uses for Triggers: Second Example

- Our trigger would be coded:

```
CREATE OR REPLACE TRIGGER check_salary
  BEFORE UPDATE OF salary ON employees
  FOR EACH ROW
DECLARE
  v_min_sal  minsal%min_salary%TYPE;
BEGIN
 SELECT min_salary INTO v_min_sal
  FROM minsal;
 IF :NEW.salary < v_min_sal
  OR :NEW.salary < :OLD.salary THEN
     RAISE_APPLICATION_ERROR (-20508,
     'Do not decrease salary.');
 END IF;
END;
```

Uses for Triggers: Third Example

- You need to create a report showing the total salary bill for a department.
- You can declare and use this cursor:

```
...
CURSOR tot_sals IS
  SELECT SUM(salary)
  FROM employees
  WHERE department_id = p_dept_id;
...
```

Uses for Triggers: Third Example

```
...
CURSOR tot_sals IS
  SELECT SUM(salary)
  FROM employees
  WHERE department_id = p_dept_id;
...
```

- But what if, in a large organization, there are 10,000 employees in the department?
- FETCHing 10,000 rows from the EMPLOYEES table may be too slow.
- The next slides show a much faster way to do this.

Uses for Triggers: Third Example

- First, we add a new column to the DEPARTMENTS table to store the total salary bill for each department:

```
ALTER TABLE DEPARTMENTS
  ADD total_salary NUMBER(12,2);
```

- Populate this column with the current total dept salary:

```
UPDATE departments d
  SET total_salary =
  (SELECT SUM(salary) FROM employees
   WHERE department_id = d.department_id);
```

- A DML row trigger will keep this new column up to date when salaries are changed.

Uses for Triggers: Third Example

```
CREATE OR REPLACE PROCEDURE increment_salary
  (p_id IN NUMBER, p_new_sal IN NUMBER) IS
BEGIN
 UPDATE copy_departments
 SET total_salary = total_salary + NVL(p_new_sal,0)
 WHERE  department_id = p_id;
END increment_salary;
```

```
CREATE OR REPLACE TRIGGER compute_salary
AFTER INSERT OR UPDATE OF salary OR DELETE
ON employees FOR EACH ROW
BEGIN
 IF DELETING THEN   increment_salary
   (:OLD.department_id,(:OLD.salary * -1));
 ELSIF UPDATING THEN  increment_salary
   (:NEW.department_id,(:NEW.salary - :OLD.salary));
 ELSE       increment_salary
   (:NEW.department_id,:NEW.salary);
 END IF;
END;
```

Terminology

Key terms used in this lesson included:

- CALL statement
- Database Event trigger
- DDL trigger
- Mutating table

Summary

In this lesson, you should have learned how to:

- Describe events that cause DDL and database event triggers to fire
- Create a trigger for a DDL statement
- Create a trigger for a database event
- Describe the functionality of the CALL statement
- Describe the cause of a mutating table


ORACLE
Academy