

## Database Programming with PL/SQL

### 13-3: Creating DML Triggers: Part II

#### Practice Activities

##### Vocabulary

Identify the vocabulary word for each definition below.

<b>Compound Trigger</b>	is a single trigger that can include actions for each of the four possible timing points
<b>INSTEAD OF trigger</b>	a trigger which replaces a DML statement on a complex view with DML statements on the tables on which the view is based
<b>Conditional Predicate</b>	predefined Boolean variables INSERTING, DELETING and UPDATING which can be tested in a trigger body to take different code paths depending on which DML statement caused the trigger to fire
<b>OLD and :NEW qualifiers</b>	enables a row trigger to access column values in the table row currently being modified by the triggering statement
<b>DML Row Trigger</b>	a DML trigger which fires once for each row affected by the triggering DML statement

##### Try It / Solve It

1. Retrieve the code for the AFTER INSERT trigger you created in the previous practice, question 2B. If you have lost the code, here it is again:

```
CREATE OR REPLACE TRIGGER emp_audit_trigg
  AFTER INSERT ON employees
BEGIN
```

```
INSERT INTO audit_table (action) VALUES
('Inserting');

END;
```

2. Modify this trigger so that a DELETE on the EMPLOYEES table will fire the same trigger. Use the conditional predicates so an INSERT adds a row to the AUDIT\_EMP table with 'Inserted' for the action column and a DELETE adds a row with 'Deleted' in the action column. Save the script and test your trigger by inserting an employee row and then deleting the same row, querying the AUDIT\_EMP table each time.

```
CREATE OR REPLACE TRIGGER emp_audit_trigg
AFTER INSERT OR DELETE ON employees
BEGIN
IF INSERTING THEN
INSERT INTO audit_table (action) VALUES
('Inserting');
ELSE IF DELETING THEN INSERT INTO audit_table (action)
VALUES
('DELETING');
END IF;
END;
```

3. Add a new column called emp\_id to the AUDIT\_EMP table. This column will contain the employee id of the worker whose record was inserted or deleted. Modify your trigger to be a row trigger so it will fire once for each row affected. The INSERTs into the AUDIT\_EMP table should now include the employee id of the affected employee. INSERT and DELETE one or more employees. Query the AUDIT\_EMP table to see the audit trail.

```
CREATE OR REPLACE TRIGGER emp_audit_trigg
```

**AFTER INSERT OR DELETE ON employees FOR EACH  
ROW**

**BEGIN**

**IF INSERTING THEN**

**INSERT INTO audit\_table (action, emp\_id)  
VALUES ('Inserting', :OLD.employee\_id);  
ELSE IF DELETING THEN  
INSERT INTO audit\_table (action, emp\_id)  
VALUES ('DELETING', :OLD.employee\_id);  
END IF;**

**END;**

4. To practice using INSTEAD OF triggers, complete the following steps.

- A. Execute the following statement to create a table called DEPT\_COUNT that keeps track of how many employees are in each department.

**CREATE TABLE dept\_count  
AS SELECT department\_id dept\_id, count(\*)  
count\_emps FROM employees  
GROUP BY department\_id;**

- B. Execute the following statement to create a view of the EMPLOYEES table called EMP\_VU.

**CREATE VIEW emp\_vu  
AS SELECT employee\_id, last\_name, department\_id  
FROM employees;**

- C. Create an INSTEAD OF row trigger on EMP\_VU that increases the current count for a department by 1 if a new employee is added and subtracts 1 from the count for a department if an employee is deleted.

```

CREATE OR REPLACE TRIGGER t_departamentos INSTEAD OF INSERT OR
DELETE ON EMP_VU
BEGIN
IF INSERTING THEN
UPDATE DEPT_COUNT
SET count_emps = count_emps +1
where dept_id= :new.department_id;
END IF;

IF DELETING THEN
UPDATE DEPT_COUNT
SET count_emps = count_emps -1
where dept_id= :new.department_id;
END IF;
END;

```

- D. Look at the counts for all departments in DEPT\_COUNT. Test to see if your trigger fires correctly by inserting a row into EMP\_VU. Look at the count for the department of the new employee. Delete a row from EMP\_VU. Look at the count for the department where the employee was just deleted.

5. In this question, you will create a compound trigger. Once again, you will use the AUDIT\_TABLE you created in a previous exercise. If you have lost that table, below is the code to recreate it.

```

CREATE TABLE audit_table
(action VARCHAR2(50), user_name
VARCHAR2(30) DEFAULT USER,
last_change_date TIMESTAMP DEFAULT SYSTIMESTAMP,
emp_id          NUMBER(6));

```

- A. Create a compound trigger emp\_audit\_trigg on the EMPLOYEES table for the following events: when updating the salary column of the EMPLOYEES table, enter the value 'Updating' into the action column of the AUDIT\_TABLE before the change occurs. Next, once the action is complete, change the action to 'Update complete; old salary was (old\_sal); new salary is (new\_sal)' where old\_sal is the original salary before the UPDATE, and new\_sal is the new salary.

- B. Test your trigger by updating the salary of employee\_id = 124 to be 1200, then querying the AUDIT\_TABLE to see that it contains a new row.