ORACLE®
Academy

Database Programming with PL/SQL

13-3
Creating DML Triggers: Part II

ORACLE
Academy

Objectives

This lesson covers the following objectives:

- Create a DML trigger that uses conditional predicates
- Create a row-level trigger
- Create a row-level trigger that uses OLD and NEW qualifiers
- Create an INSTEAD OF trigger
- Create a Compound Trigger

Purpose

- There might be times when you want a trigger to fire under a specific condition.
- Or, you might want a trigger to impact just a row of data.
- These are examples of the DML trigger features covered in this lesson.

Using Conditional Predicates

- In the previous lesson, you saw a trigger that prevents INSERTs into the EMPLOYEES table during the weekend:

```
CREATE OR REPLACE TRIGGER secure_emp
  BEFORE INSERT ON employees
BEGIN
  IF TO_CHAR(SYSDATE, 'DY') IN ('SAT', 'SUN') THEN
  RAISE_APPLICATION_ERROR(-20500,
   'You may insert into EMPLOYEES table only during
                                     business hours');
  END IF;
END;
```

Using Conditional Predicates

```
CREATE OR REPLACE TRIGGER secure_emp
  BEFORE INSERT ON employees
BEGIN
  IF TO_CHAR(SYSDATE, 'DY') IN ('SAT', 'SUN') THEN
  RAISE_APPLICATION_ERROR(-20500,
   'You may insert into EMPLOYEES table only during
                                    business hours');
  END IF;
END;
```

- Suppose you want to prevent any DML operation on EMPLOYEES during the weekend, but with different error messages for INSERT, UPDATE, and DELETE.
- You could create three separate triggers; however, the next slide shows how to do this with a single trigger.

Using Conditional Predicates

- The trigger keywords DELETING, INSERTING, and UPDATING are automatically declared Boolean variables which are 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

- 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;
```

- This trigger will allow other columns of EMPLOYEES to be updated at any time.

Understanding Row Triggers

- Remember that 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;
```

- This trigger inserts exactly one row into the log table, regardless of whether the triggering statement updates one employee, several employees, or no employees at all.

Understanding Row Triggers

- Suppose you want to insert one row into the log table for each updated employee.
- For example, if five employees were updated, you want to insert five rows into the log table so you have a record of each row that was changed.
- For this, you need a row trigger.

Row Trigger Firing Sequence

- A row trigger fires (executes) once for each row affected by the triggering DML statement, either just BEFORE the row is processed or just AFTER.
- If five employees are in department 50, a row trigger associated with an UPDATE on the employees table would execute five times, storing five rows in the log file, because of the following DML statement:

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

Creating a Row Trigger

```
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;
```

- You specify a row trigger using FOR EACH ROW.
- With this trigger, the UPDATE statement from the previous slide would cause five rows to be inserted into the log table, one for each EMPLOYEE row updated.
- However, all five rows in the log table would be identical, and they would not show which employee was updated or how SALARY was changed.

Using :OLD and :NEW Qualifiers

- When using a row trigger, you can reference and use both old and new column values in the EMPLOYEES row currently being updated.
- You use :OLD.column_name to reference the pre-update value, and :NEW.column_name to reference the post-update value.

Using :OLD and :NEW Qualifiers

- For example, if the UPDATE statement is changing an employee's salary from \$10,000 to \$11,000, then while the trigger is executing:
  - :OLD.salary has a value of 10000
  - :NEW.salary has a value of 11000.
  - With this information, you can now insert the data you need into the logging table.
- The next slide shows how.

Using :OLD and :NEW Qualifiers

```
CREATE OR REPLACE TRIGGER log_emps
  AFTER UPDATE OF salary ON employees FOR EACH ROW
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;
```

- To log the employee_id, does it matter whether you code :OLD.employee_id or :NEW.employee_id?
- Is there a difference?

A Second Example of Row Triggers

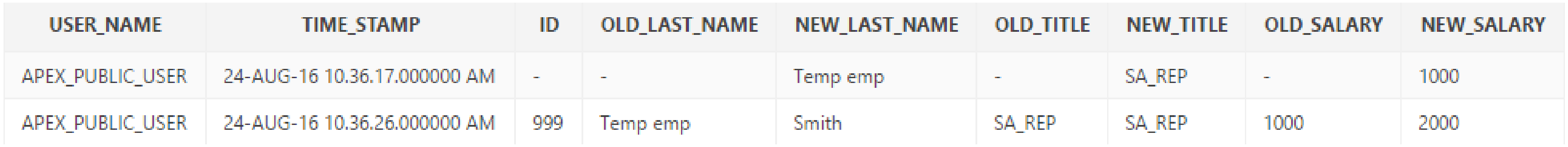
```
CREATE OR REPLACE TRIGGER audit_emp_values
  AFTER DELETE OR INSERT OR UPDATE ON employees FOR EACH ROW
BEGIN
  INSERT INTO audit_emp(user_name, time_stamp, id,
  old_last_name, new_last_name, old_title,
  new_title, old_salary, new_salary)
  VALUES (USER, SYSDATE, :OLD.employee_id,
  :OLD.last_name, :NEW.last_name, :OLD.job_id,
  :NEW.job_id, :OLD.salary, :NEW.salary);
END;
```

A Second Example: Testing the audit_emp_values Trigger

```
INSERT INTO employees
  (employee_id, last_name, job_id, salary, ...)
 VALUES (999, 'Temp emp', 'SA_REP', 1000,...);

UPDATE employees
  SET salary = 2000, last_name = 'Smith'
  WHERE employee_id = 999;
```

```
SELECT user_name, time_stamp, ...
  FROM audit_emp;
```



USER_NAME	TIME_STAMP	ID	OLD_LAST_NAME	NEW_LAST_NAME	OLD_TITLE	NEW_TITLE	OLD_SALARY	NEW_SALARY
APEX_PUBLIC_USER	24-AUG-16 10.36.17.000000 AM	-	-	Temp emp	-	SA_REP	-	1000
APEX_PUBLIC_USER	24-AUG-16 10.36.26.000000 AM	999	Temp emp	Smith	SA_REP	SA_REP	1000	2000

A Third Example of Row Triggers

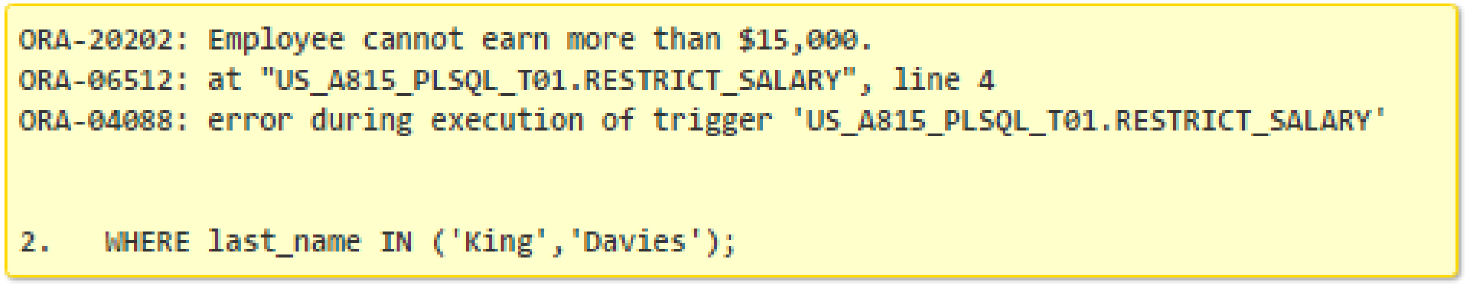
- Suppose you need to prevent employees who are not a President or Vice-President from having a salary of more than $15,000.

```
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;
```

Testing the restrict_salary Trigger:

```
UPDATE employees
SET salary = 15500
  WHERE last_name IN ('King','Davies');
```

- King is a (Vice-)President, but Davies is not.
- This UPDATE statement produces the following error:



```
ORA-20202: Employee cannot earn more than $15,000.
ORA-06512: at "US_A815_PLSQL_T01.RESTRICT_SALARY", line 4
ORA-04088: error during execution of trigger 'US_A815_PLSQL_T01.RESTRICT_SALARY'


2.   WHERE last_name IN ('King','Davies');
```

- Neither EMPLOYEES row is updated, because the UPDATE statement must either succeed completely or not at all.

Testing the restrict_salary Trigger:

- King’s salary update will be rolled back, because every SQL statement must either complete 100% successfully or not at all. This rule is called “statement-level consistency” and is a basic rule of the Oracle database.
- The error message does not show which row(s) violated the check and were therefore not updated. But we could easily show this by modifying the trigger code to:

```
… RAISE_APPLICATION_ERROR (-20202, 'Employee ' ||
  :NEW.employee_id || ' cannot earn more than $15,000.');
```

A Fourth Example: Implementing an Integrity Constraint With a Trigger

- The EMPLOYEES table has a foreign key constraint on the DEPARTMENT_ID column of the DEPARTMENTS table.
- DEPARTMENT_ID 999 does not exist, so this DML statement violates the constraint and the employee row is not updated:

```
UPDATE employees
SET department_id = 999
  WHERE employee_id = 124;
```

- You can use a trigger to create the new department automatically. The next slide shows how.

A Fourth Example: Creating the Trigger:

```
CREATE OR REPLACE TRIGGER employee_dept_fk_trg
  BEFORE UPDATE OF department_id ON employees FOR EACH ROW
DECLARE
  v_dept_id  departments.department_id%TYPE;
BEGIN
  SELECT department_id INTO v_dept_id FROM departments
  WHERE department_id = :NEW.department_id;
EXCEPTION
  WHEN NO_DATA_FOUND THEN
  INSERT INTO departments VALUES(:NEW.department_id,
  'Dept '||:NEW.department_id, NULL, NULL);
END;
```

- Let’s test it:

```
UPDATE employees
SET department_id = 999
  WHERE employee_id = 124;
-- Successful after trigger is fired
```

Using the REFERENCING Clause

- Look again at the first example of a row trigger:

```
CREATE OR REPLACE TRIGGER log_emps
  AFTER UPDATE OF salary ON employees FOR EACH ROW
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;
```

- What if the EMPLOYEES table had a different name?
- What if it was called OLD instead?
- OLD is not a good name, but is possible.
- What would our code look like now?

Using the REFERENCING Clause

```
CREATE OR REPLACE TRIGGER log_emps
  AFTER UPDATE OF salary ON old FOR EACH ROW
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;
```

- The word "old" in this code means two things: it is a value qualifier (like :NEW) and also a table name.
- The code will work, but is confusing to read.
- We don't have to use :OLD and :NEW.
- We can use different qualifiers by including a REFERENCING clause.

Using the REFERENCING Clause

```
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;
```

- FORMER and LATTER are called correlation-names.
- They are aliases for OLD and NEW.
- We can choose any correlation names we like (for example TOM and MARY) as long as they are not reserved words.
- The REFERENCING clause can be used only in row triggers.

Using the WHEN clause

- Look at this trigger code. It records salary changes only if the new salary is greater than the old salary.

```
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;
```

- The whole trigger body is a single IF statement.
- In real life, this could be many lines of code, including CASE statements, loops, and other constructs.

Using the WHEN clause

- We can code our IF condition 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;
```

- This code is easier to read, especially if the trigger body is long and complex.
- The WHEN clause can be used only with row triggers.

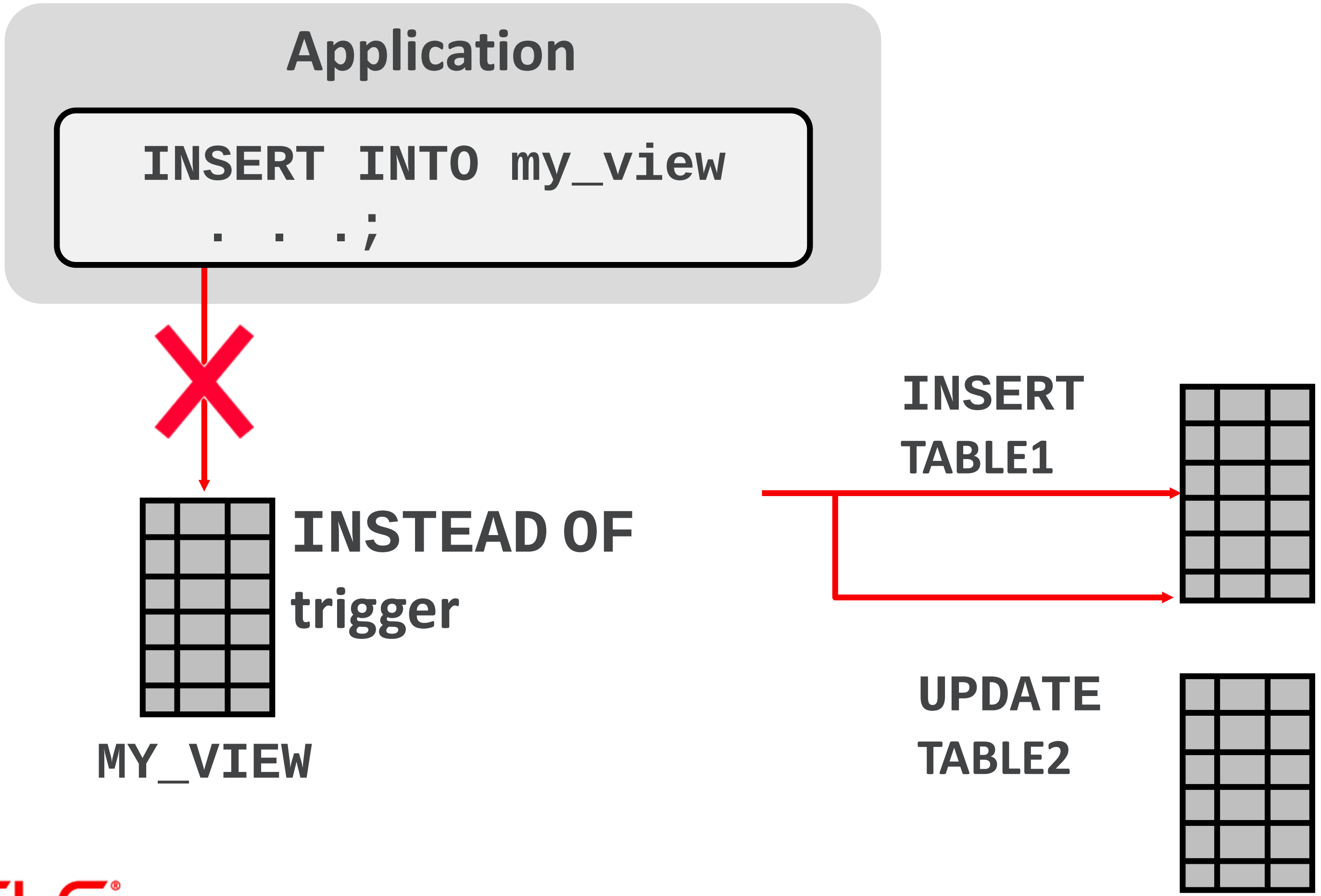
INSTEAD OF Triggers

- Underlying tables cannot be updated using a Complex View (for example a view based on a join).
- 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.
- INSTEAD OF triggers are always row triggers.

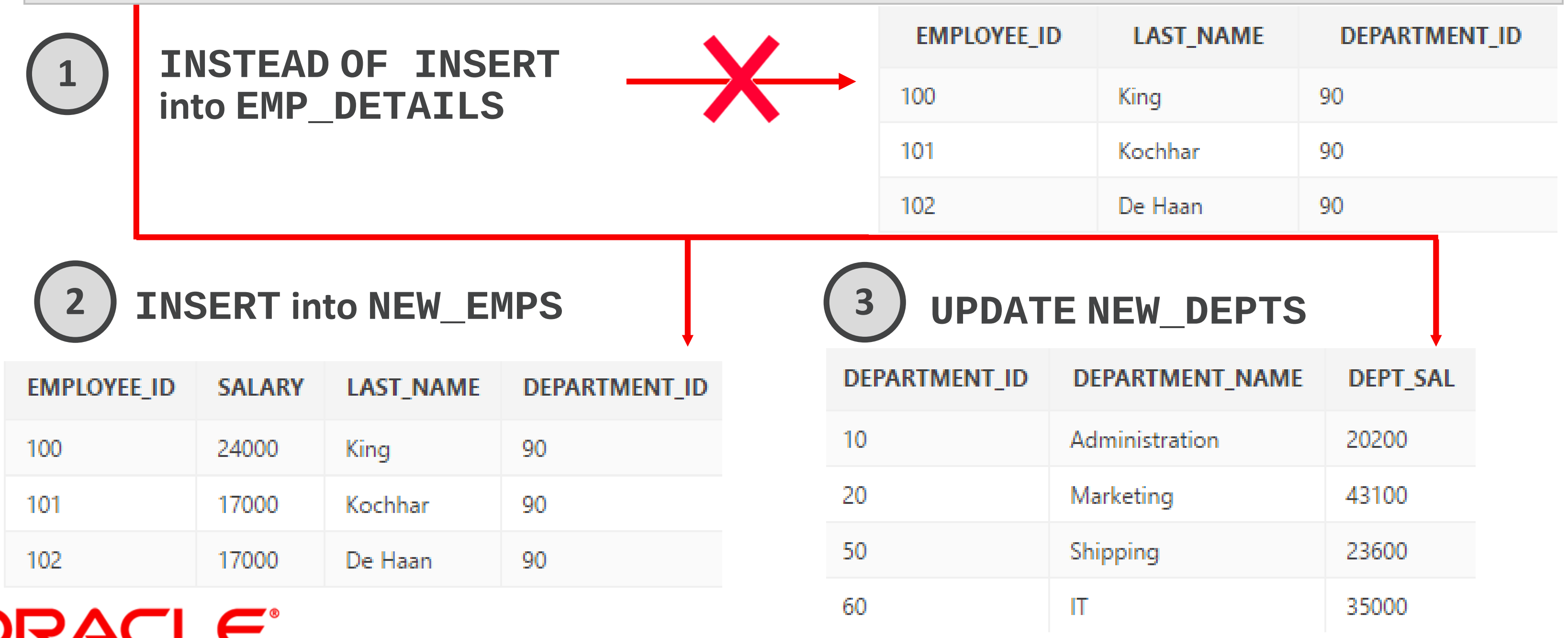
INSTEAD OF Triggers



An Example of an INSTEAD OF Trigger

- Perform the INSERT into the EMP_DETAILS view that is based on the NEW_EMPS and NEW_DEPTS tables:

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



1 INSTEAD OF INSERT into EMP_DETAILS

EMPLOYEE_ID	LAST_NAME	DEPARTMENT_ID
100	King	90
101	Kochhar	90
102	De Haan	90

2 INSERT into NEW_EMPS

EMPLOYEE_ID	SALARY	LAST_NAME	DEPARTMENT_ID
100	24000	King	90
101	17000	Kochhar	90
102	17000	De Haan	90

3 UPDATE NEW_DEPTS

DEPARTMENT_ID	DEPARTMENT_NAME	DEPT_SAL
10	Administration	20200
20	Marketing	43100
50	Shipping	23600
60	IT	35000

Creating an INSTEAD OF Trigger

- Step 1: Create the tables and the Complex View:

```
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
  GROUP BY d.department_id,d.department_name;

CREATE VIEW emp_details AS
  SELECT e.employee_id, e.last_name, e.salary,
    e.department_id, d.department_name
  FROM new_emps e, new_depts d
  WHERE e.department_id = d.department_id;
```

Creating an INSTEAD OF Trigger

- Step 2: Create the INSTEAD OF Trigger:

```
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;
```

- INSTEAD OF triggers are always row triggers.

Row Triggers Revisited

- Look at this row trigger. It adds a row to the LOG_TABLE whenever an employee's salary changes.

```
CREATE OR REPLACE TRIGGER log_emps
  AFTER UPDATE OF salary ON employees FOR EACH ROW
BEGIN
  INSERT INTO log_table (employee_id, change_date, salary)
  VALUES (:OLD.employee_id, SYSDATE, :NEW.salary);
END;
```

Row Triggers Revisited

- What if there are one million employees and you give every employee a 5% salary increase?

```
UPDATE employees
SET salary = salary * 1.05;
```

- The row trigger will automatically execute one million times, INSERTing one row each time.
- This will be very slow.

Row Triggers Revisited

- Earlier in the course you learned how to use Bulk Binding (FORALL) to speed up DML.
- Can we use FORALL in our trigger?

```
CREATE OR REPLACE TRIGGER log_emps
  AFTER UPDATE OF salary ON employees FOR EACH ROW
DECLARE
  TYPE t_log_emp IS TABLE OF log_table%ROWTYPE
  INDEX BY BINARY_INTEGER;
  log_emp_tab  t_log_emp;
BEGIN
  ... Populate log_emp_tab with employees' change data
  FORALL i IN log_emp_tab.FIRST..log_emp_tab.LAST
  INSERT INTO log_table VALUES log_emp_tab(i);
END;
```

Row Triggers Revisited

- No, this will not work.
- Hint: remember this is a row trigger.
- Think about the scope of the LOG_EMP_TAB collection variable.

```
CREATE OR REPLACE TRIGGER log_emps
  AFTER UPDATE OF salary ON employees FOR EACH ROW
DECLARE
  TYPE t_log_emp IS TABLE OF log_table%ROWTYPE
  INDEX BY BINARY_INTEGER;
  log_emp_tab  t_log_emp;
BEGIN
  ... Populate log_emp_tab with employees’ change data
  FORALL i IN log_emp_tab.FIRST..log_emp_tab.LAST
  INSERT INTO log_table VALUES log_emp_tab(i);
END;
```

Row Triggers Revisited

- Trigger variables lose scope at the end of each execution of the trigger.
- So each time the row trigger is fired, all the data already collected in LOG_EMP_TAB is lost.
- To avoid losing this data, we need a trigger that fires only once – a statement trigger.
- But to reference column values from each row (using :OLD and :NEW) we need a row trigger.
- But a single trigger cannot be both a row trigger and a statement trigger at the same time.
- Right?

Row Triggers Revisited

- Wrong!
- We create a Compound Trigger.

What is a Compound Trigger?

- A single trigger that can include actions for each of the four possible timing points: before the triggering statement, before each row, after each row, and after the triggering statement.
- A Compound Trigger has a declaration section and a section for each of its timing points.
- You do not have to include all the timing points, just the ones you need.
- The scope of Compound Trigger variables is the whole trigger, so they retain their scope throughout the whole execution.

Compound Trigger Structure

```
CREATE OR REPLACE TRIGGER trigger_name
 FOR dml_event_clause ON table_name COMPOUND TRIGGER
          -- Initial section
           -- Declarations
           -- Subprograms
          -- Optional section
          BEFORE STATEMENT IS ...;
          -- Optional section
          AFTER STATEMENT IS ...;
          -- Optional section
          BEFORE EACH ROW IS ...;
          -- Optional section
          AFTER EACH ROW IS ...;
```

ORACLE
Academy

Compound Triggers: an Example:

- This example has a declaration section and two of the four possible timing point sections.

```
CREATE OR REPLACE TRIGGER log_emps
  FOR UPDATE OF salary ON employees
COMPOUND TRIGGER
  TYPE t_log_emp IS TABLE OF log_table%ROWTYPE
  INDEX BY BINARY_INTEGER;
  log_emp_tab  t_log_emp;
AFTER EACH ROW IS
BEGIN
  ... Populate log_emp_tab with employees’ change data
END AFTER EACH ROW;
AFTER STATEMENT IS
BEGIN
  FORALL ...
END AFTER STATEMENT;
END log_emps;
```

Compound Triggers Example: The Full Code

```
CREATE OR REPLACE TRIGGER log_emps
FOR UPDATE OF salary ON copy_employees  COMPOUND TRIGGER

TYPE t_log_emp IS TABLE OF log_table%ROWTYPE INDEX BY
BINARY_INTEGER;
log_emp_tab t_log_emp;
v_index BINARY_INTEGER := 0;

AFTER EACH ROW IS
BEGIN
v_index := v_index + 1;
log_emp_tab(v_index).employee_id := :OLD.employee_id;
log_emp_tab(v_index).change_date := SYSDATE;
log_emp_tab(v_index).salary := :NEW.salary;
END AFTER EACH ROW;

AFTER STATEMENT IS
BEGIN
FORALL I IN log_emp_tab.FIRST..log_emp_tab.LAST
INSERT INTO log_table VALUES log_emp_tab(i);
END AFTER STATEMENT;
END log_emps;
```

Terminology

Key terms used in this lesson included:

- Conditional predicate
- Compound trigger
- DML row trigger
- INSTEAD OF trigger
- :OLD and :NEW qualifiers

Summary

In this lesson, you should have learned how to:

- Create a DML trigger that uses conditional predicates
- Create a row-level trigger
- Create a row-level trigger that uses OLD and NEW qualifiers
- Create an INSTEAD OF trigger
- Create a Compound Trigger

ORACLE®

Academy