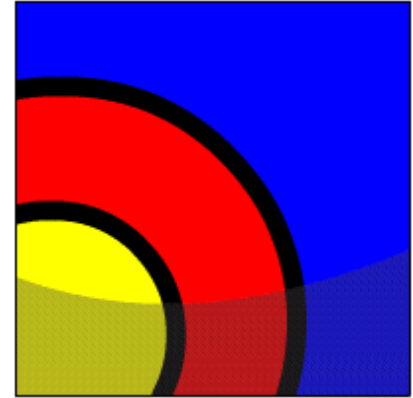


Creating DML Triggers: Part II



In this lesson, you will learn 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.



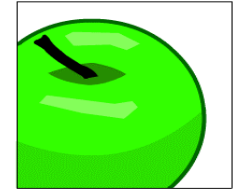


Why Learn It?

In this lesson, you learn how to create and use more powerful DML triggers. 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 additional trigger features that are covered in this lesson.



Tell Me / Show Me



Using Conditional Predicates

In the previous lesson, you saw a trigger that prevents INSERTS into EMPLOYEES 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;
```

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



Tell Me / Show Me

Using Conditional Predicates (continued)

```
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');
        ELIF INSERTING THEN RAISE_APPLICATION_ERROR
            (-20502,'You may insert into EMPLOYEES'
              || ' table only during business hours');
        ELIF UPDATING THEN RAISE_APPLICATION_ERROR
            (-20503,'You may update EMPLOYEES'
              || ' table only during business hours');
        END IF;
    END IF;
END;
```



Tell Me / Show Me

Using Conditional Predicates (continued)

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 update SALARY'
                || ' only during business hours');
        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;
```



Tell Me / Show Me

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.

Suppose you want to insert one row into the log table for each updated employee. For example, if four employees were updated, insert four rows into the log table. You need a **row trigger**.



Tell Me / Show Me

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 there are five employees in department 50, the row triggers execute five times each:

```
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
144	Vargas	50

→ **BEFORE** row trigger

→ **AFTER** row trigger
...

→ **BEFORE** row trigger

→ **AFTER** row trigger
...



Tell Me / Show Me

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`.

The `UPDATE` statement in the previous slide now inserts five rows into the log table, one for each `EMPLOYEE` row updated.

However, all five rows in the log table are identical to each other. And the log table does not show which employees were updated, or what changes were made to their salaries.



Tell Me / Show Me

Using :OLD and :NEW Qualifiers

Only within a row trigger, can you reference and use both old and new column values in the EMPLOYEES row currently being updated.

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

For example, if the UPDATE statement is changing an employee's salary from 10000 to 11000, then `:OLD.salary` has a value of 10000, and `:NEW.salary` has a value of 11000.

Now you can insert the data you need into the logging table. The next slide shows how.



Tell Me / Show Me

Using :OLD and :NEW Qualifiers (continued)

```
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?



Tell Me / Show Me

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;
```



Tell Me / Show Me

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	04-DEC-06	999	Temp emp	Smith	SA_REP	SA_REP	1000	2000
APEX_PUBLIC_USER	04-DEC-06	-	-	Temp emp	-	SA_REP	-	1000

2 rows returned in 0.01 seconds

[Download](#)



Tell Me / Show Me

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 \$15000.

```
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;
```



Tell Me / Show Me

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 "USVA_TEST_SQL01_T01.RESTRICT_SALARY", line 4
ORA-04088: error during execution of trigger 'USVA_TEST_SQL01_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.



Tell Me / Show Me

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.



Tell Me / Show Me

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
```



Tell Me / Show Me

INSTEAD OF Triggers

A Complex View (for example a view based on a join) cannot be updated. Suppose the EMP_DETAILS view is a complex view based on a join of EMPLOYEES and DEPARTMENTS. The following SQL statement fails:

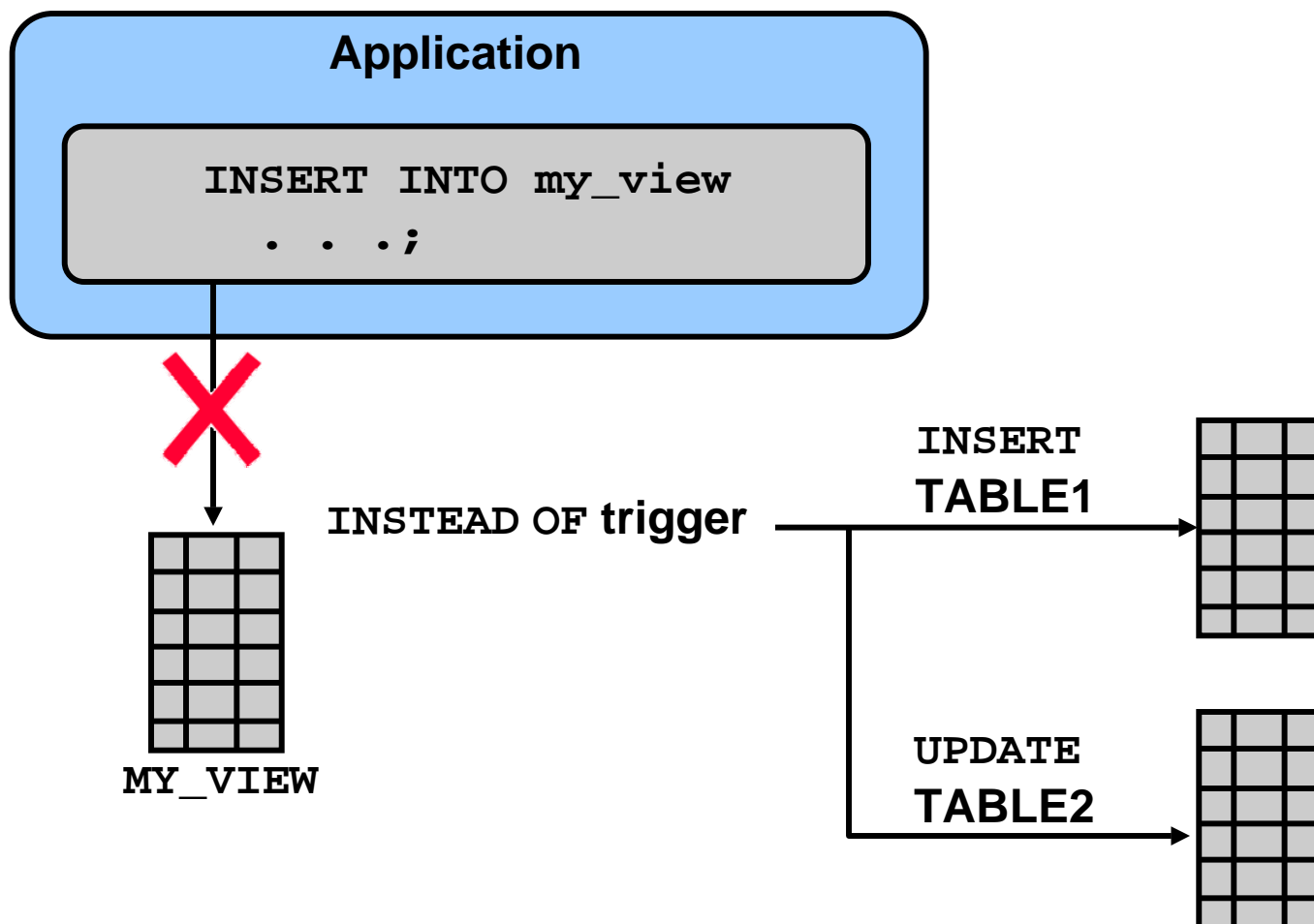
```
INSERT INTO emp_details  
VALUES (9001,'ABBOTT',3000, 10, 'Administration');
```

You can overcome this by creating a trigger that updates the two base tables directly **instead of** trying (and failing) to update the view.

INSTEAD OF triggers are always row triggers.

Tell Me / Show Me

INSTEAD OF Triggers (continued)



Tell Me / Show Me

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
102	De Haan	90
101	Kochhar	90
100	King	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
...			
9001	3000	ABBOTT	10

3

UPDATE NEW_DEPTS

DEPARTMENT_ID	DEPARTMENT_NAME	DEPT_SAL
10	Administration	7400
20	Marketing	19000
50	Shipping	17500
60	IT	19200
...		



Tell Me / Show Me

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;
```



Tell Me / Show Me

Creating an INSTEAD OF Trigger (continued)

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;
```

Tell Me / Show Me

Terminology

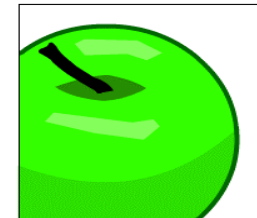
Key terms used in this lesson include:

Conditional predicate

DML row trigger

:OLD and :NEW qualifiers

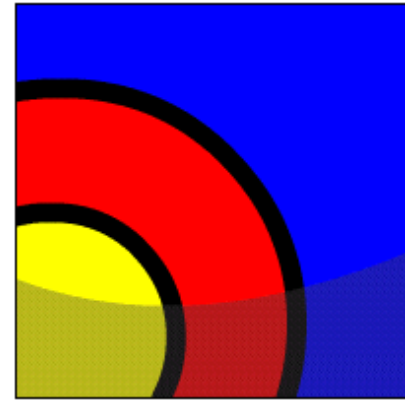
INSTEAD OF trigger



Summary

In this lesson, you learned 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





Try It / Solve It

The exercises in this lesson cover the following topics:

- Using conditional predicates in a DML trigger
- Creating a row-level trigger
- Using the `OLD` and `NEW` qualifiers in a database trigger
- Creating an `INSTEAD OF` trigger

