Oracle 11g – PL SQL

PL SQL Triggers

Objectives

After completing this lesson, you should be able to do the following:

- Describe Triggers
- Identify the Trigger Event Types, Body, and Firing (Timing)
- Differences between Statement Level Triggers and Row Level Triggers
- Create DML Triggers using the CREATE TRIGGER Statement and SQL Developer
- Create Instead of and Disabled Triggers
- How to Manage, Test and Remove Triggers
- INSTEAD OF Triggers

Types of Triggers

A trigger:

- Is a PL/SQL block or a PL/SQL procedure associated with a table, view, schema, or database
- Executes implicitly whenever a particular event takes place
- Can be either of the following:
  - Application trigger: Fires whenever an event occurs with a particular application
  - Database trigger: Fires whenever a data event (such as DML) or system event (such as logon or shutdown) occurs on a schema or database

Guidelines for Designing Triggers

- You can design triggers to:
  - Perform related actions
  - Centralize global operations
- You must not design triggers:
  - Where functionality is already built into the Oracle server
  - That duplicate other triggers
- You can create stored procedures and invoke them in a trigger, if the PL/SQL code is very lengthy.
- The excessive use of triggers can result in complex interdependencies, which may be difficult to maintain in large applications.

Trigger Event Types and Body

A trigger event:

- Determines which DML statement causes the trigger to execute
- Types are:
  - `INSERT`
  - `UPDATE [OF column]`
  - `DELETE`

A trigger body:

- Determines what action is performed
- Is a PL/SQL block or a `CALL` to a procedure

Types of Triggers (DML)

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

Trigger Timing (DML)

When should the trigger fire?

- `BEFORE`: Execute the trigger body before the triggering DML event on a table.
- `AFTER`: Execute the trigger body after the triggering DML event on a table.
- `INSTEAD OF`: Execute the trigger body instead of the triggering statement. This is used for views that are not otherwise modifiable.

Note: If multiple triggers are defined for the same object, then the order of firing triggers is arbitrary.

Creating DML Triggers

Create DML statement or row type triggers by using:

```
CREATE [OR REPLACE] TRIGGER trigger_name
 timing
 event1 [OR event2 OR event3]
ON object_name
[[REFERENCING OLD AS old | NEW AS new]
 FOR EACH ROW
 [WHEN (condition)]]
trigger_body
```

- A statement trigger fires once for a DML statement.
- A row trigger fires once for each row affected.

Note: Trigger names must be unique with respect to other triggers in the same schema.

Trigger-Firing Sequence

Use the following firing sequence for a trigger on a table when a single row is manipulated:

DML statement

```
INSERT INTO departments
   (department_id,department_name, location_id)
VALUES (400, 'CONSULTING', 2400);
```

Triggering action

DEPARTMENT_ID	DEPARTMENT_NAME	LOCATION_ID
10	Administration	1700
20	Marketing	1800
30	Purchasing	1700
...		
400	CONSULTING	2400

→ BEFORE statement trigger

→ BEFORE row trigger

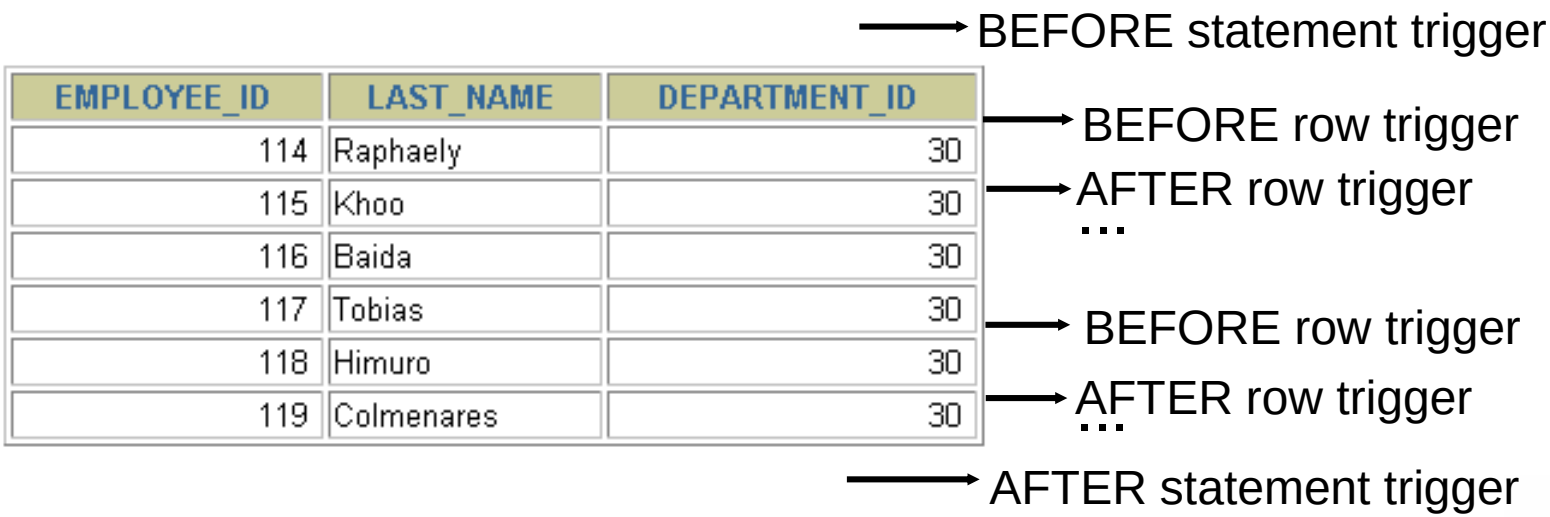
→ AFTER row trigger

→ AFTER statement trigger

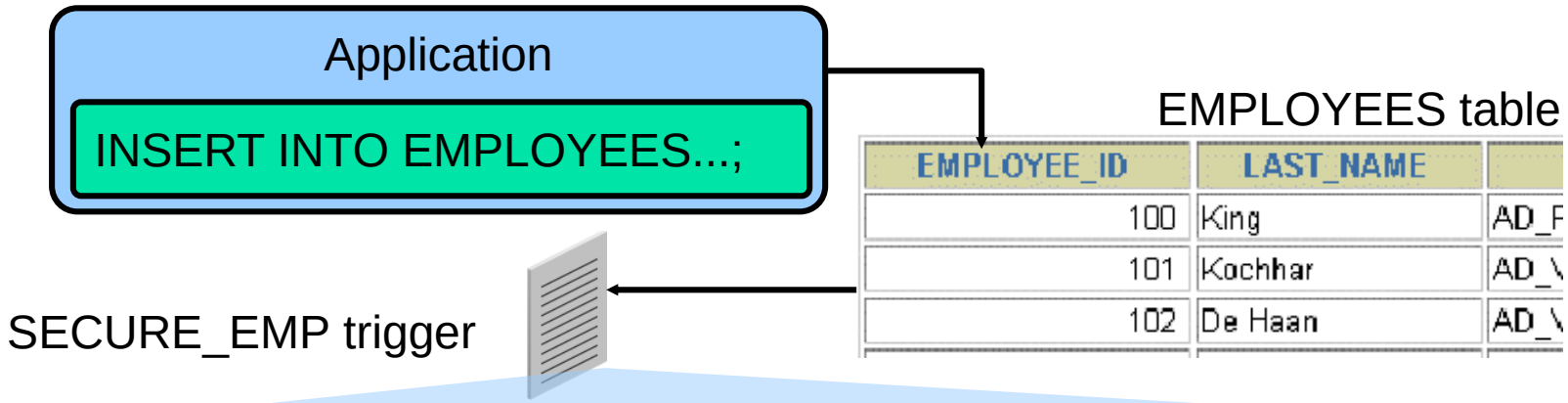
Trigger-Firing Sequence

Use the following firing sequence for a trigger on a table when many rows are manipulated:

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



Creating a DML Statement Trigger




```
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 '
    ||' business hours.');
  END IF;
END;
```

Testing SECURE_EMP

```
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);
```

```
INSERT INTO employees (employee_id, last_name, first_name, email,
       *
ERROR at line 1:
ORA-20500: You may insert into EMPLOYEES table only during business hours.
ORA-06512: at "PLSQL.SECURE_EMP", line 4
ORA-04088: error during execution of trigger 'PLSQL.SECURE_EMP'
```

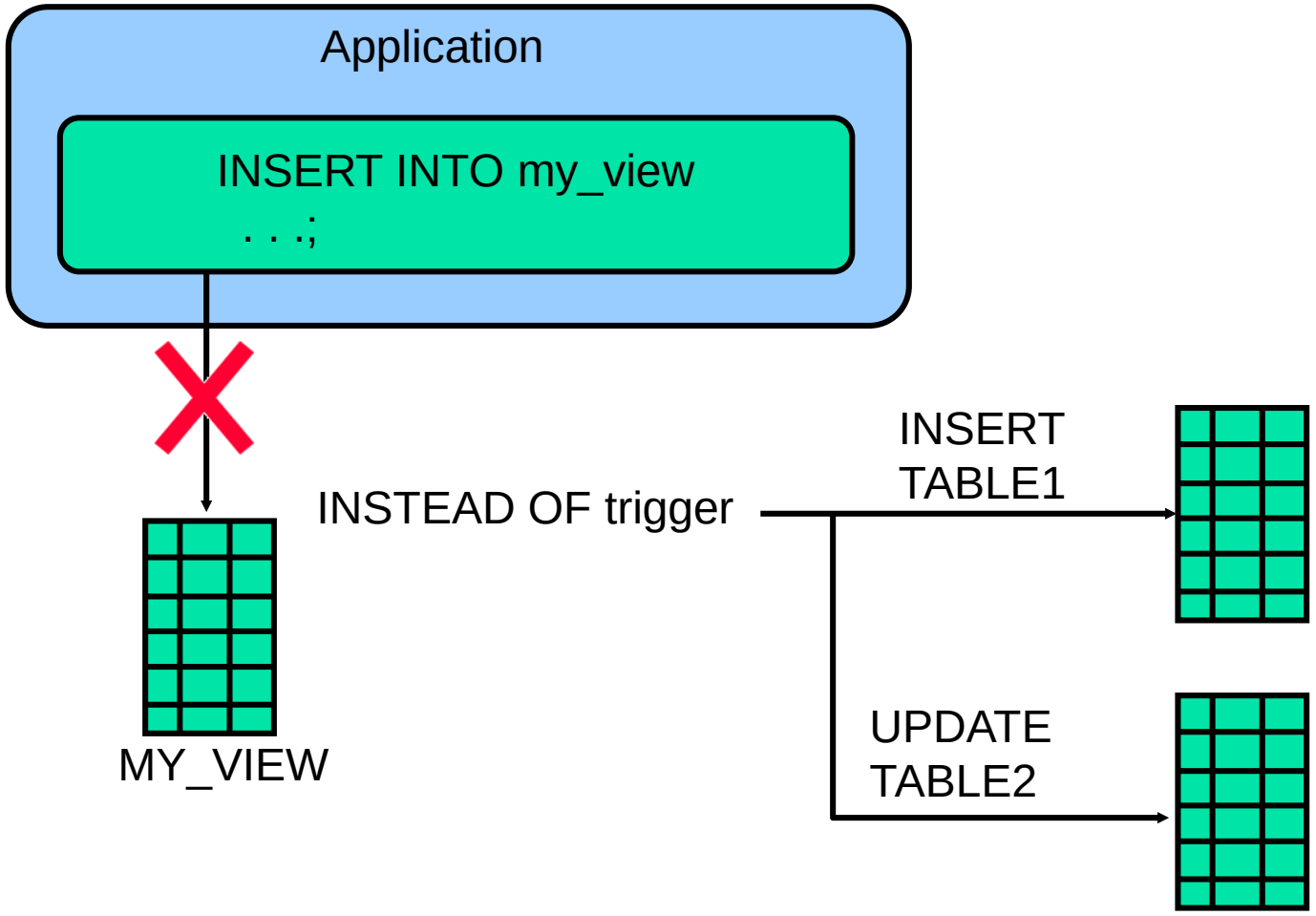
Implementing an Integrity Constraint with a Trigger

```
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
    NULL; -- mask exception if department exists
END;
/
```

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

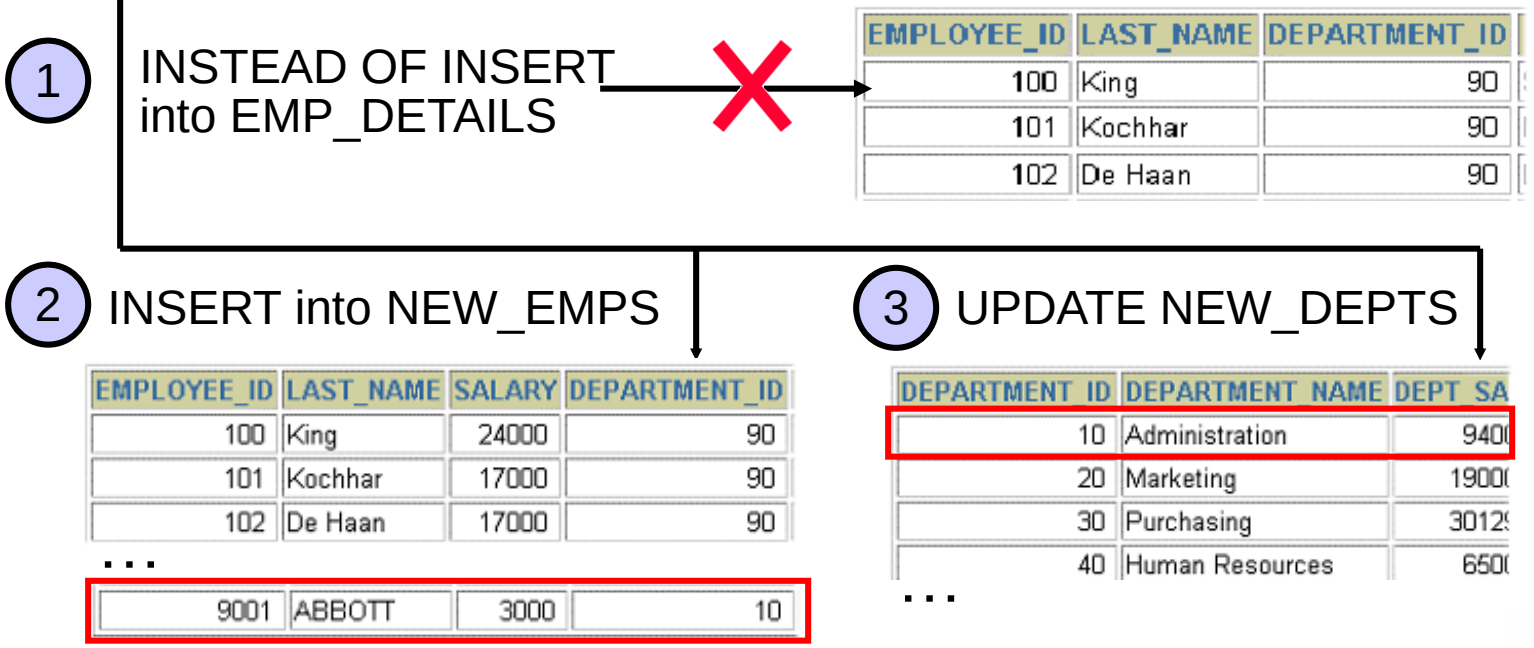
INSTEAD OF Triggers



Creating an INSTEAD OF Trigger

Perform the INSERT into EMP_DETAILS that is based on EMPLOYEES and DEPARTMENTS tables:

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



Creating an INSTEAD OF Trigger

Use INSTEAD OF to perform DML on complex views:

```
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;
```

Managing Triggers

- Disable or reenable a database trigger:

```
ALTER TRIGGER trigger_name DISABLE | ENABLE
```

- Disable or reenable all triggers for a table:

```
ALTER TABLE table_name DISABLE | ENABLE
  ALL TRIGGERS
```

- Recompile a trigger for a table:

```
ALTER TRIGGER trigger_name COMPILE
```

Removing Triggers

To remove a trigger from the database, use the `DROP TRIGGER` statement:

```
DROP TRIGGER trigger_name;
```

Example:

```
DROP TRIGGER secure_emp;
```

Note: All triggers on a table are removed when the table is removed.

Testing Triggers

- Test each triggering data operation, as well as nontriggering data operations.
- Test each case of the `WHEN` clause.
- Cause the trigger to fire directly from a basic data operation, as well as indirectly from a procedure.
- Test the effect of the trigger on other triggers.
- Test the effect of other triggers on the trigger.