Creating Triggers

Objectives

- After completing this lesson, you should be able to do the following:
  - Describe the different types of triggers
  - Describe database triggers and their uses
  - Create database triggers
  - Describe database trigger-firing rules
  - Remove database 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.

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.

Types of DML Triggers

- The trigger type determines if 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

- 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.

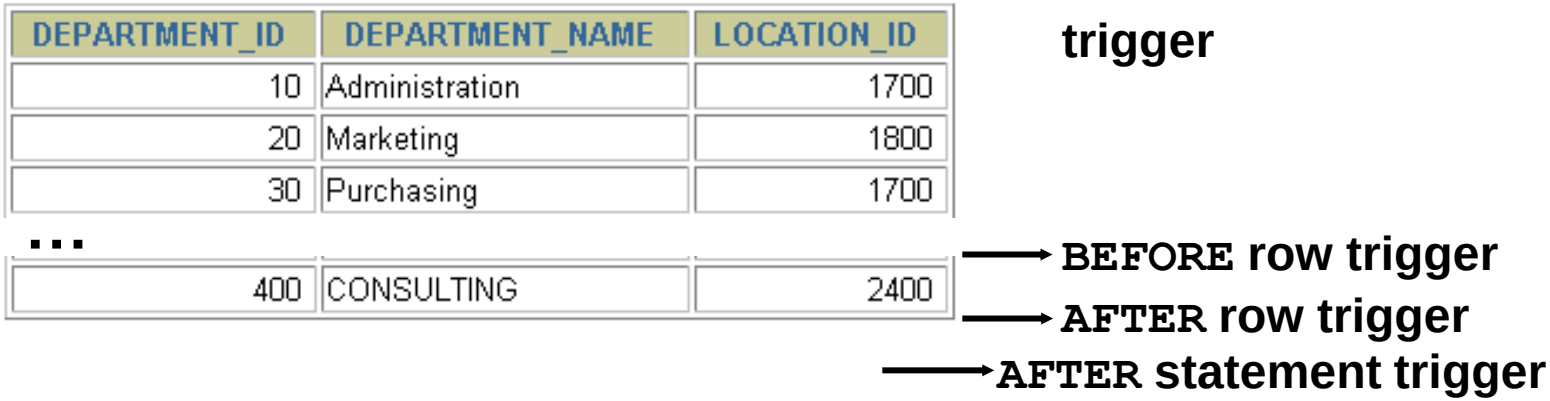
Trigger-Firing Sequence

- 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);
```

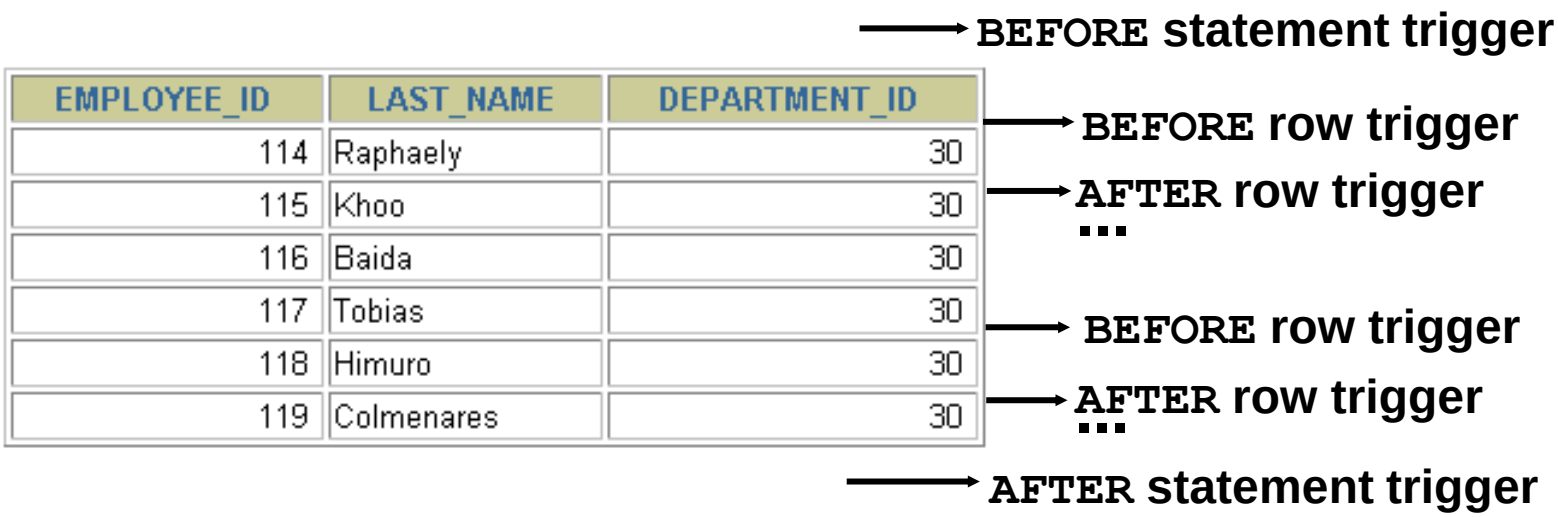
Triggering action



Trigger-Firing Sequence

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

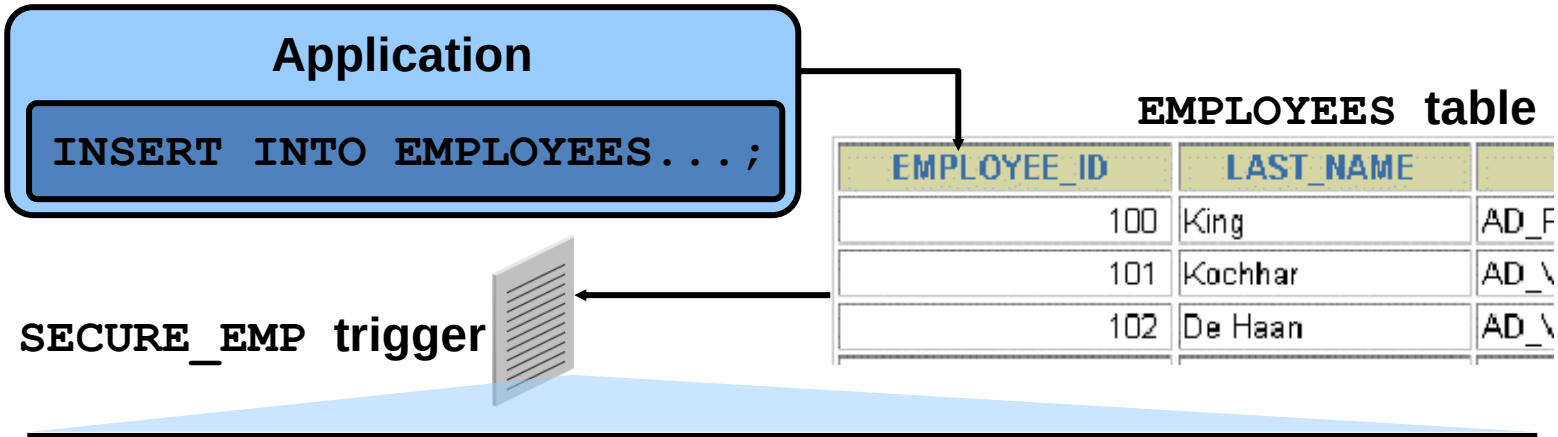
```
UPDATE employees
  SET salary = salary * 1.1
  WHERE department_id = 30;
```



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

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'
```

Using Conditional Predicates

```
CREATE OR REPLACE TRIGGER secure_emp BEFORE
INSERT OR UPDATE OR DELETE ON employees BEGIN
 IF (TO_CHAR(SYSDATE,'DY') IN ('SAT','SUN')) OR
   (TO_CHAR(SYSDATE,'HH24')
       NOT BETWEEN '08' AND '18') THEN
   IF DELETING THEN RAISE_APPLICATION_ERROR(
    -20502,'You may delete from EMPLOYEES table'||
           'only during business hours.');
   ELSIF INSERTING THEN RAISE_APPLICATION_ERROR(
    -20500,'You may insert into EMPLOYEES table'||
           'only during business hours.');
   ELSIF UPDATING('SALARY') THEN
    RAISE_APPLICATION_ERROR(-20503, 'You may '||
     'update SALARY only during business hours.');
   ELSE RAISE_APPLICATION_ERROR(-20504,'You may'||
     ' update EMPLOYEES table only during'||
     ' normal hours.');
   END IF;
 END IF;
END;
```

Creating a DML Row Trigger

```
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;
/
```

Using OLD and NEW Qualifiers

```
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;
/
```

Using OLD and NEW Qualifiers: Example Using audit_emp

```
INSERT INTO employees
 (employee_id, last_name, job_id, salary, ...)
VALUES (999, 'Temp emp', 'SA_REP', 6000,...);

UPDATE employees
 SET salary = 7000, last_name = 'Smith'
 WHERE employee_id = 999;
```

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

USER_NAME	TIMESTAMP	ID	OLD_LAST_N	NEW_LAST_N	OLD_TITLE	NEW_TITLE	OLD_SALARY	NEW_SALARY
PLSQL	28-SEP-01			Temp emp		SA_REP		1000
PLSQL	28-SEP-01	999	Temp emp	Smith	SA_REP	SA_REP	1000	2000

Restricting a Row Trigger: Example

```
CREATE OR REPLACE TRIGGER derive_commission_pct
BEFORE INSERT OR UPDATE OF salary ON employees
FOR EACH ROW
WHEN (NEW.job_id = 'SA_REP')
BEGIN
 IF INSERTING THEN
   :NEW.commission_pct := 0;
 ELSIF :OLD.commission_pct IS NULL THEN
   :NEW.commission_pct := 0;
 ELSE
   :NEW.commission_pct := :OLD.commission_pct+0.05;
 END IF;
END;
/
```

Summary of Trigger Execution Model

1. Execute all `BEFORE STATEMENT` triggers.
2. Loop for each row affected:
   a. Execute all `BEFORE ROW` triggers.
   b. Execute the DML statement and perform integrity constraint checking.
   c. Execute all `AFTER ROW` triggers.
3. Execute all `AFTER STATEMENT` triggers.

- Note: Integrity checking can be deferred until the `COMMIT` operation is performed.

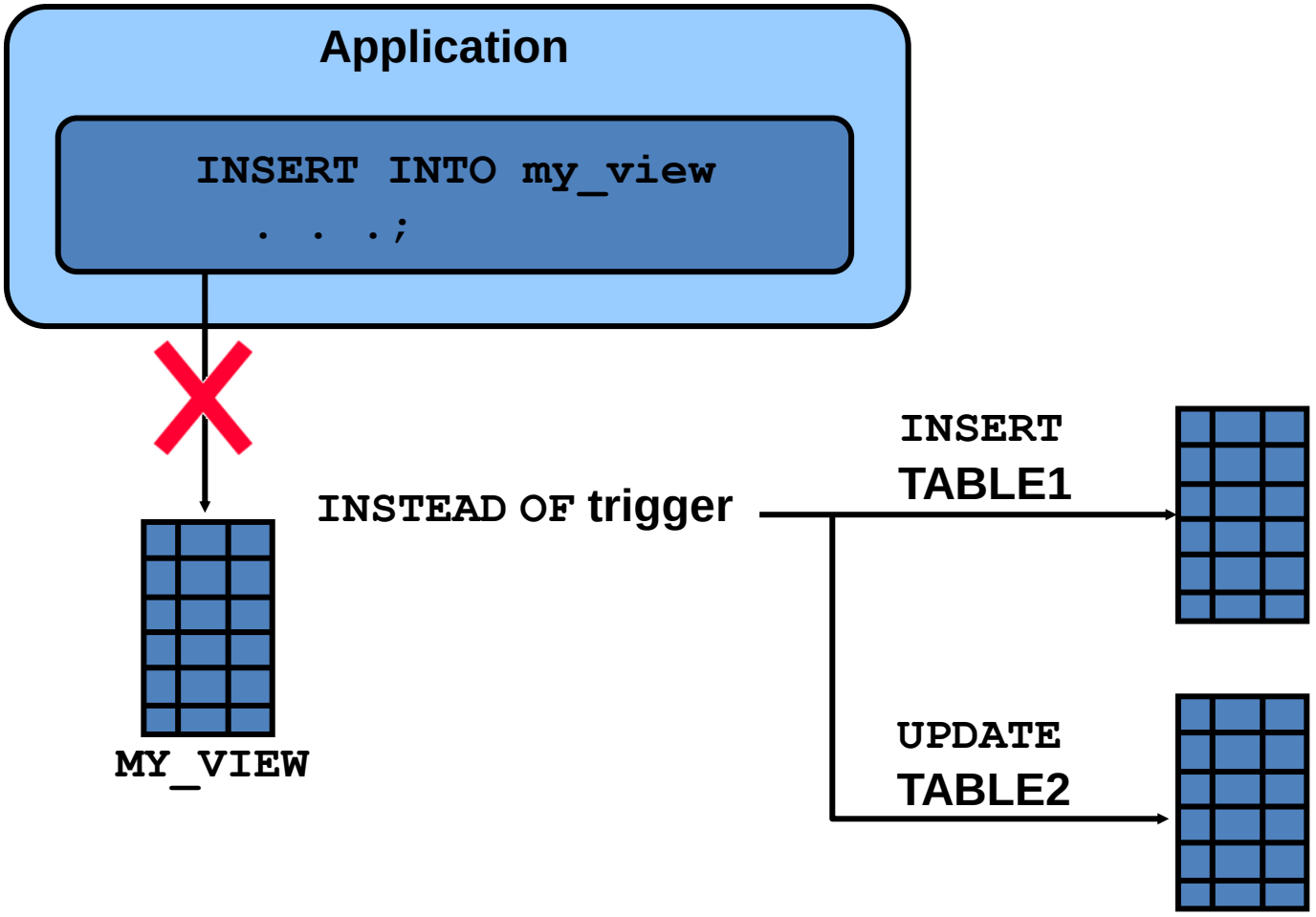
Implementing an Integrity Constraint with a Trigger

```
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;
/
```

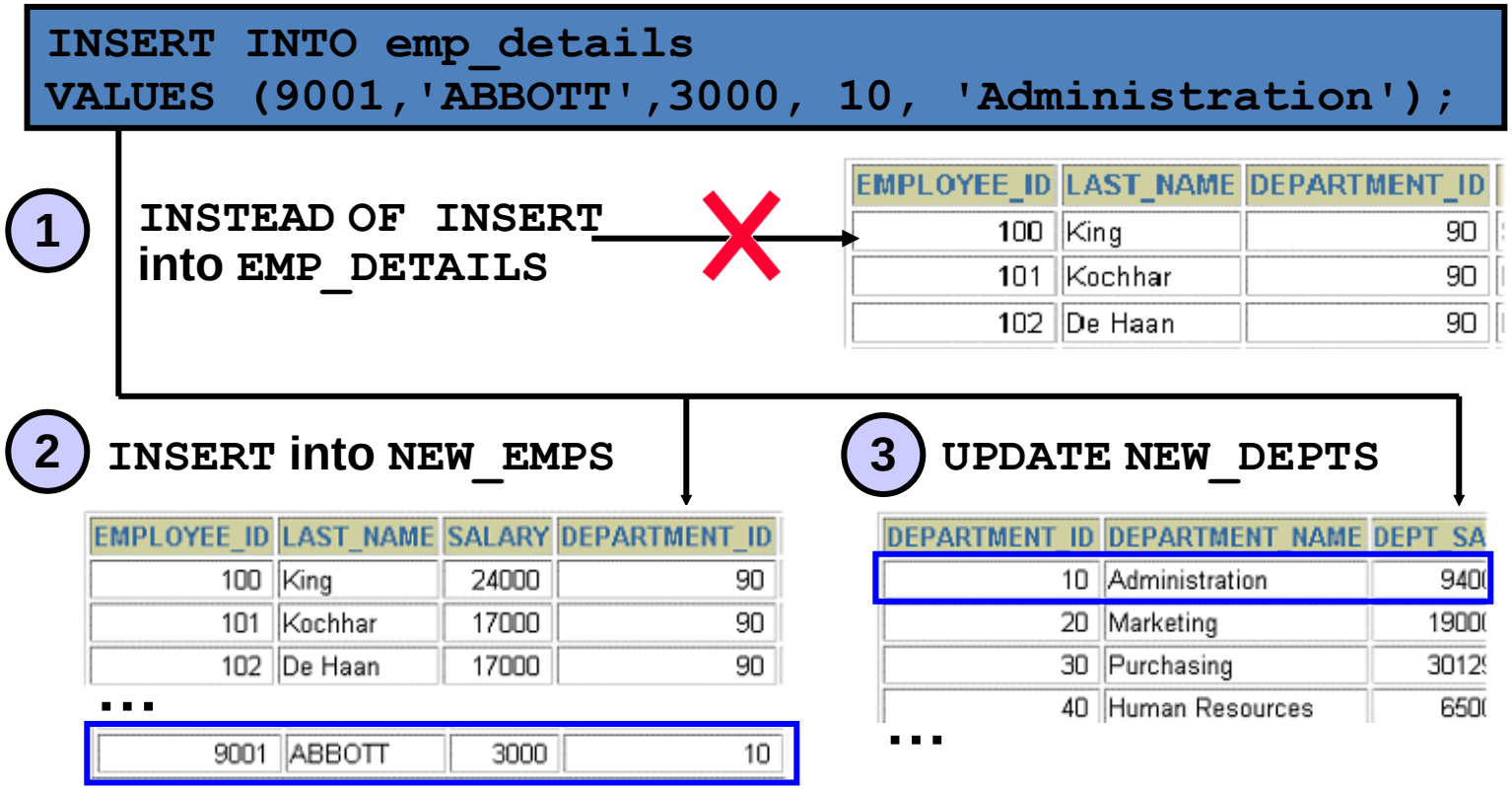
```
UPDATE employees SET department_id = 999
 WHERE employee_id = 170;
-- Successful after trigger is fired
```

INSTEAD OF Triggers



Creating an INSTEAD OF Trigger

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



Creating an INSTEAD OF Trigger

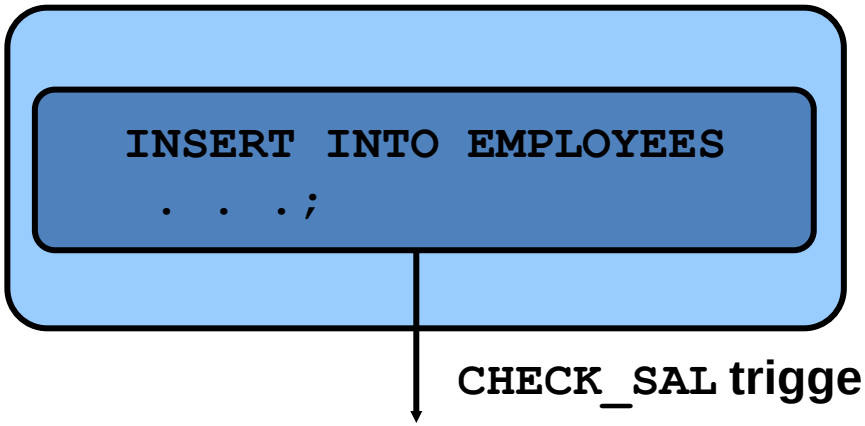
- Use INSTEAD OF to perform DML on

```
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;
```

Comparison of Database Triggers and Stored Procedures

Triggers	Procedures
Defined with `CREATE TRIGGER`	Defined with `CREATE PROCEDURE`
Data dictionary contains source code in `USER_TRIGGERS`.	Data dictionary contains source code in `USER_SOURCE`.
Implicitly invoked by DML	Explicitly invoked
`COMMIT`, `SAVEPOINT`, and `ROLLBACK` are not allowed.	`COMMIT`, `SAVEPOINT`, and `ROLLBACK` are allowed.

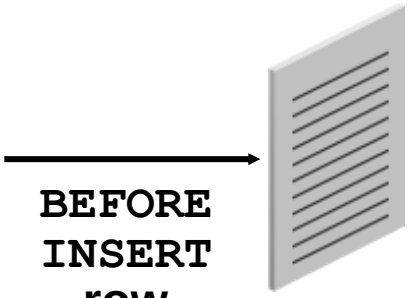
Comparison of Database Triggers and Oracle Forms Triggers



EMPLOYEES table

EMPLOYEE_ID	LAST_NAME	JOB_ID	SALARY
100	King	AD_PRES	24000
101	Kochhar	AD_VP	17000
102	De Haan	AD_VP	17000
103	Hunold	IT_PROG	9000
104	Ernst	IT_PROG	6000

...



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;
```

```
DROP TRIGGER secure_emp;
```

- Example:

- 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.

Summary

- In this lesson, you should have learned how to:
  - Create database triggers that are invoked by DML operations
  - Create statement and row trigger types
  - Use database trigger-firing rules
  - Enable, disable, and manage database triggers
  - Develop a strategy for testing triggers
  - Remove database triggers

Practice 10: Overview

- This practice covers the following topics:
  - Creating row triggers
  - Creating a statement trigger
  - Calling procedures from a trigger