PLSQL

Lesson 04 :Triggers and its types

Lesson Objectives

To understand the following topics:

- Describe database triggers and their uses
- Describe the different types of triggers
- Create database triggers
- Describe database trigger-firing rules
- Remove database triggers
- Display trigger information

4.1: Triggers
Triggers

An event which leads to action

- Types of Triggers
  - Application : triggers when an event occurs in application
  - Database

Database triggers are stored procedures that are implicitly executed when an triggering event occurs

- The triggering event could be (Database triggers):
  - DML statements on the table
  - DDL statements
  - System events such as startup, shutdown, and error messages
  - User events such as logon and logoff

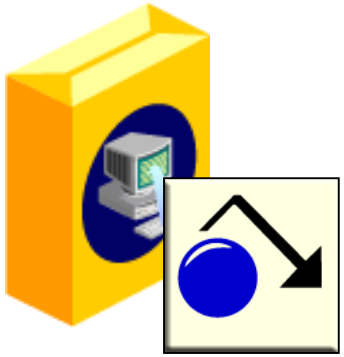
4.1: Triggers
Application and Database Triggers

Database trigger (covered in this course):

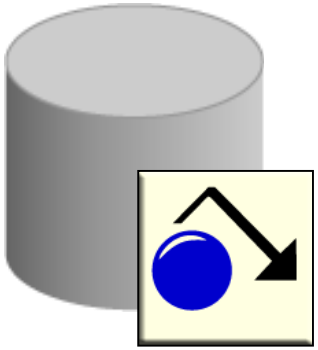
- Fires whenever a DML, a DLL, or system event occurs on a schema or database

Application trigger:

- Fires whenever an event occurs within a particular application



Application Trigger



Database Trigger

4.1: Triggers
Business Application Scenarios for Implementing Triggers

You can use triggers for:

- Security
- Auditing
- Data integrity
- Referential integrity
- Table replication
- Computing derived data automatically
- Event logging

4.1: Triggers
Available Trigger Types

Simple DML triggers

- BEFORE
- AFTER
- INSTEAD OF

Compound triggers

- Non-DML triggers
- DDL event triggers
- Database event triggers

4.1: Triggers
Parts of a Trigger

Triggering event or statement

Trigger restriction

Trigger action

Triggering Event or statement

A triggering event or statement is the SQL statement, database event, or user event that causes a trigger to fire.

A triggering event can be one or more of the following:

An INSERT, UPDATE, or DELETE statement on a specific table/view

A CREATE, ALTER, or DROP statement on any schema object

A database startup or instance shutdown

A specific error message or any error message

A user logon or logoff

4.1: Triggers
Trigger Event Types and Body

A trigger event type determines which DML statement causes the trigger to execute. The possible events are:

- INSERT
- UPDATE [OF column]
- DELETE

A trigger body determines what action is performed and is a PL/SQL block or a CALL to a procedure.

Statement-Level Triggers Versus Row-Level Triggers

Statement-Level Triggers	Row-Level Triggers
Is the default when creating a trigger	Use the FOR EACH ROW clause when creating a trigger.
Fires once for the triggering event	Fires once for each row affected by the triggering event
Fires once even if no rows are affected	Does not fire if the triggering event does not affect any rows

Trigger Restriction

Specifies a Boolean expression that must evaluate to TRUE for the trigger to fire

Trigger Action

A trigger action is the procedure that contains either PL/SQL block, Java program or C code which contains SQL statements and code to be executed

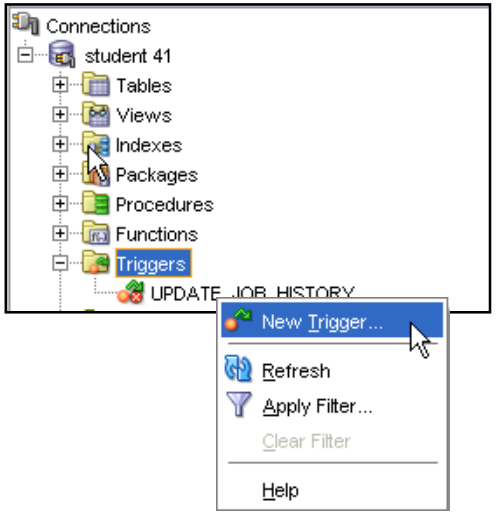
4.1: Triggers
Creating DML Triggers Using the CREATE TRIGGER Statement

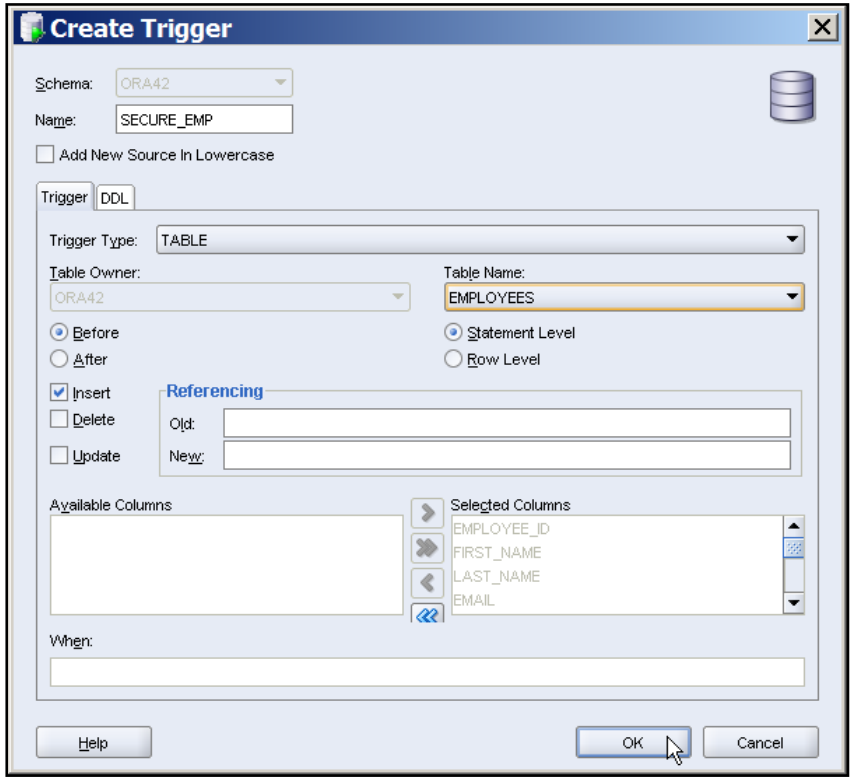
```
CREATE [OR REPLACE] TRIGGER trigger_name
timing -- when to fire the trigger
event1 [OR event2 OR event3]
ON object_name
[REFERENCING OLD AS old | NEW AS new]
FOR EACH ROW –- default is statement level trigger
WHEN (condition)]]
DECLARE]
BEGIN
... trigger_body –- executable statements
[EXCEPTION . . .]
END [trigger_name];
```

timing = BEFORE | AFTER | INSTEAD OF

event = INSERT | DELETE | UPDATE | UPDATE OF column_list

4.1: Triggers
Creating DML Triggers Using SQL Developer



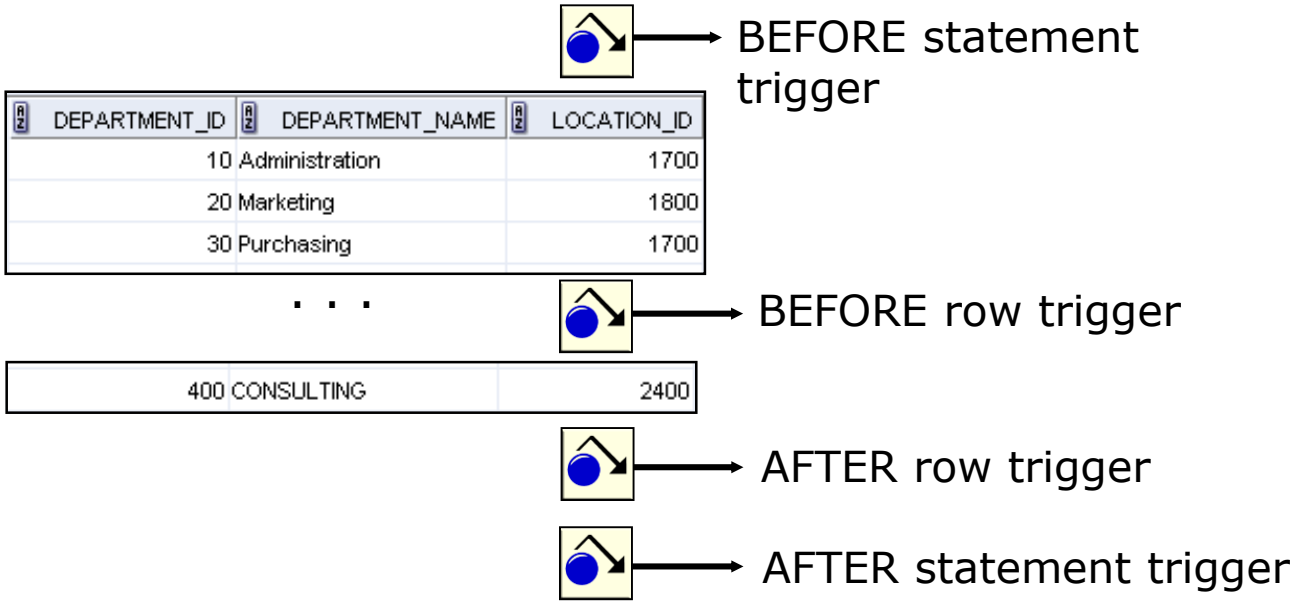



```
CREATE OR REPLACE TRIGGER secure_emp
BEFORE INSERT ON employees BEGIN
 IF (TO_CHAR(SYSDATE,'DY') IN ('SAT','SUN')) OR
     (TO_CHAR(SYSDATE,'HH24:MI') NOT BETWEEN '08:00' AND '18:00') THEN
  RAISE_APPLICATION_ERROR(-20500, 'You may insert' ||
  ' into EMPLOYEES table only during ' ||' business hours.');
  END IF;
END;
```

4.1: Triggers
Trigger-Firing Sequence:Single-Row Manipulation

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

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



4.1: Triggers
Trigger-Firing Sequence: Multirow Manipulation

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;
```

BEFORE statement trigger

EMPLOYEE_ID	LAST_NAME	DEPARTMENT_ID
114	Raphaely	30
115	Khoo	30
116	Baida	30
117	Tobias	30
118	Himuro	30
119	Colmenares	30

BEFORE row trigger

AFTER row trigger

. . .

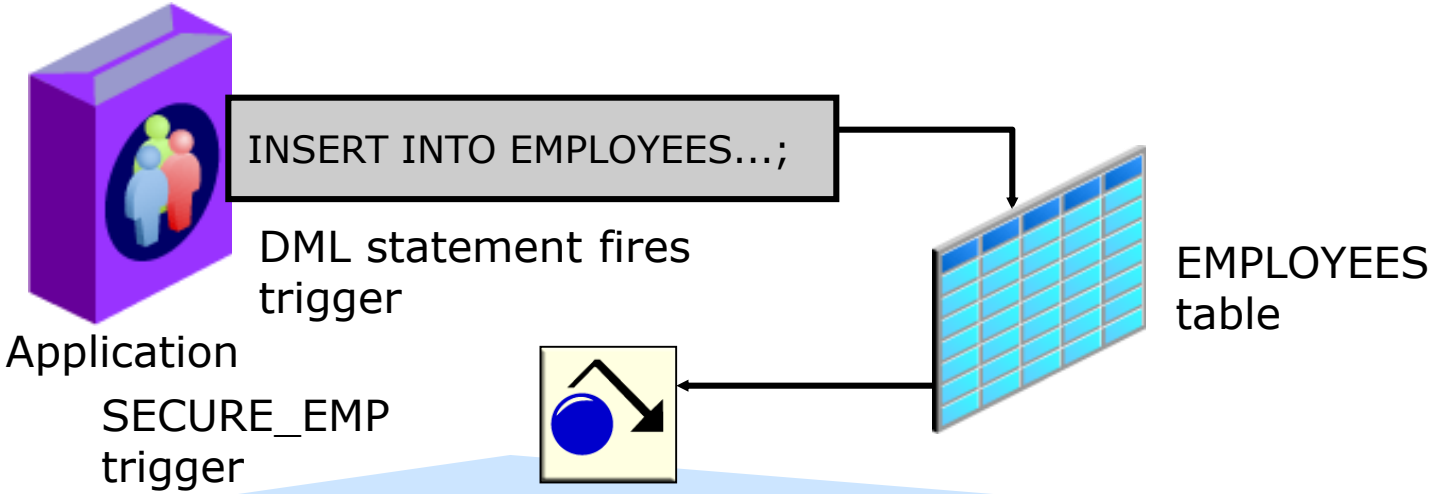
BEFORE row trigger

AFTER row trigger

. . .

AFTER statement trigger

4.1: Triggers
Creating a DML Statement Trigger Example: SECURE_EMP




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

4.1: Triggers
Testing Trigger 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);
```

Results | Script Output | Explain | Autotrace | DBMS Output | OWA Output

```
Error starting at line 1 in command:
INSERT INTO employees (employee_id, last_name, first_name, email, hire_date,
VALUES (300, 'Smith', 'Rob', 'RSMITH', SYSDATE, 'IT_PROG', 4500, 60)
Error report:
SQL Error: ORA-20500: You may insert into EMPLOYEES table only during  business hours.
ORA-06512: at "ORA42.SECURE_EMP", line 4
ORA-04088: error during execution of trigger 'ORA42.SECURE_EMP'
```

4.1: Triggers
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 normal business hours.');
      ELSIF INSERTING  THEN RAISE_APPLICATION_ERROR(
        -20500,'You may insert into EMPLOYEES table'||
        'only during normal business hours.');
      ELSIF UPDATING  ('SALARY') THEN
        RAISE_APPLICATION_ERROR(-20503, 'You may '||
        'update SALARY only normal during business hours.');
      ELSE RAISE_APPLICATION_ERROR(-20504,'You may'||
        ' update EMPLOYEES table only during'||
        ' normal business hours.');
      END IF;
    END IF;
  END;
```

4.1: Triggers
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;
```

```
UPDATE employees
SET salary = 15500
WHERE last_name = 'Russell';
```

```
Error starting at line 1 in command:
UPDATE employees
SET salary = 15500
WHERE last_name = 'Russell'
Error report:
SQL Error: ORA-20202: Employee cannot earn more than $15,000.
ORA-06512: at "ORA62.RESTRICT_SALARY", line 4
ORA-04088: error during execution of trigger 'ORA62.RESTRICT_SALARY'
```

4.1: Triggers
Using OLD and NEW Qualifiers

When a row-level trigger fires, the PL/SQL run-time engine creates and populates two data structures:

- OLD: Stores the original values of the record processed by the trigger
- NEW: Contains the new values

NEW and OLD have the same structure as a record declared using the %ROWTYPE on the table to which the trigger is attached.

Data Operations	Old Value	New Value
INSERT	NULL	Inserted value
UPDATE	Value before update	Value after update
DELETE	Value before delete	NULL

4.1: Triggers
New and Old Values

New and old values of the DML statements can be processed with :NEW.column_name and :OLD.column_name in the trigger restriction and trigger action .

Insert will have values in New variable

Update will have values in New and Old variables

Delete will have values in Old variable

4.1: Triggers
Database Triggers - DML

Create or replace trigger <trigger_name>

after/before

insert/update of <column_list>/delete on <table_name/view_name>

for each row

When (<condition>)

<pl_sql >

4.2: Trigger Types
Types of Trigger

Row triggers and Statement triggers

Before and After triggers

Instead of triggers

Triggers on system events and user events

4.2: Trigger Types
Row and Statement Triggers

Row triggers fire once for every row affected by the triggering statement

Statement triggers fire once on behalf of the triggering event

4.2: Trigger Types
Before and After triggers

Specify the trigger timing

Are fired by DML statements either before or after the execution of the DML statements

Apply to row and statement triggers

Cannot be specified on views

Trigger type combinations :

Before statement trigger

Before row trigger

After row trigger

After statement trigger

You can have multiple triggers of the same type for the same statement for any given table

4.2: Trigger Types
Firing Sequence

Before Statement trigger

Before Row trigger

After Row trigger

After Statement trigger

Fires for all the affected rows

4.2: Trigger Types
Trigger Firing Order

Starting from 11g Oracle allows you to specify trigger firing order if more than one trigger is created .

It is done using FOLLOWS keyword followed by trigger name after which current trigger is to be invoked.

Contd.,

```
 create or replace trigger test_trg2 before insert on test
for each row
follows test_trg1
begin
insert into testlog values (‘From test_trig2’);
end;
```

Instead of triggers

Complex views which cannot be modified by DML statements can be modified by using Instead of trigger

It provides a transparent way of modifying the base tables through the views

Trigger is fired instead of executing the triggering statement

4.2: Trigger Types
Triggers on System Events and User Events

Certain system events like database startup and shutdown and server error messages can be traced through triggers

User events like user logon and logoff, DDL and DML can also be traced through triggers

4.2: Trigger Types
Example of DML trigger

```
create or replace trigger emp_del
before delete on emp
for each row
begin
        insert into emp_history values
        (:old.empno,:old.ename,:old.job,:old.mgr,:old.hiredate,
        :old.sal,:old.comm,:old.deptno);

end ;
/
```

4.2: Trigger Types
Example

```
create or replace trigger dept_tot_emp
after insert on emp
for each row
begin

        update dept set tot_emp = tot_emp + 1
        where deptno = :new.deptno;
end;
/
```

4.2: Trigger Types
Example of DML trigger

```
create or replace trigger  dept_tot_emp
after insert or delete or  update of deptno  on emp
for each row
begin

if inserting or updating then
            update dept set tot_emp = tot_emp + 1
            where deptno = :new.deptno;

end if;
```

4.2: Trigger Types
Example of DML trigger

```
if updating  or deleting then

        update dept set tot_emp = tot_emp - 1
        where deptno = :old.deptno;
end if;

end;
```

4.2: Trigger Types
Example of DML trigger

```
create or replace trigger  dept_tot_emp
after insert or delete or  update of deptno  on emp
for each row
when(old.deptno <> new.deptno)
begin
if inserting or updating then
            update dept set tot_emp = tot_emp + 1
           where deptno = :new.deptno;
end if;
if updating  or deleting then
        update dept set tot_emp = tot_emp - 1
        where deptno = :old.deptno;
end if;
end;
```

4.2: Trigger Types
Using OLD and NEW Qualifiers: Example

```
CREATE TABLE audit_emp (
  user_name      VARCHAR2(30),
  time_stamp     date,
  id             NUMBER(6),
  old_last_name VARCHAR2(25),
  new_last_name VARCHAR2(25),
  old_title      VARCHAR2(10),
  new_title      VARCHAR2(10),
  old_salary     NUMBER(8,2),
  new_salary     NUMBER(8,2) )
/
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;
```

4.2: Trigger Types
Using OLD and NEW Qualifiers:Example

```
INSERT INTO employees (employee_id, last_name, job_id, salary,
email, hire_date)
VALUES (999, 'Temp emp', 'SA_REP', 6000, 'TEMPEMP',
TRUNC(SYSDATE))
/
UPDATE employees
 SET salary = 7000, last_name = 'Smith'
 WHERE employee_id = 999
/
SELECT  *
FROM  audit_emp;
```

Results | Script Output | Explain | Autotrace | DBMS Output | OWA Output

Results:

	USER_NAME	TIME_STAMP	ID	OLD_LAST_NAME	NEW_LAST_NAME	OLD_TITLE	NEW_TITLE	OLD_SALARY	NEW_SALARY
1	ORA61	04-JUN-09	(null)	(null)	Temp emp	(null)	SA_REP	(null)	6000
2	ORA61	04-JUN-09	999	Temp emp	Smith	SA_REP	SA_REP	6000	7000

4.2: Trigger Types
Using the WHEN Clause to Fire a Row Trigger Based on a Condition

```
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;
/
```

4.2: Trigger Types
Summary of the Trigger Execution Model

1. Execute all BEFORE STATEMENT triggers.
2. Loop for each row affected by the SQL statement:

a. Execute all BEFORE ROW triggers for that row.

b. Execute the DML statement and perform integrity constraint checking for that row.

c. Execute all AFTER ROW triggers for that row.

3. Execute all AFTER STATEMENT triggers.

4.2: Trigger Types
Implementing an Integrity Constraint with an After Trigger

```
-- Integrity constraint violation error –2991 raised.
UPDATE employees SET department_id = 999
 WHERE employee_id = 170;
```

```
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;
/
```

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

```
1 rows updated
```

4.2: Trigger Types
Database trigger – Instead off

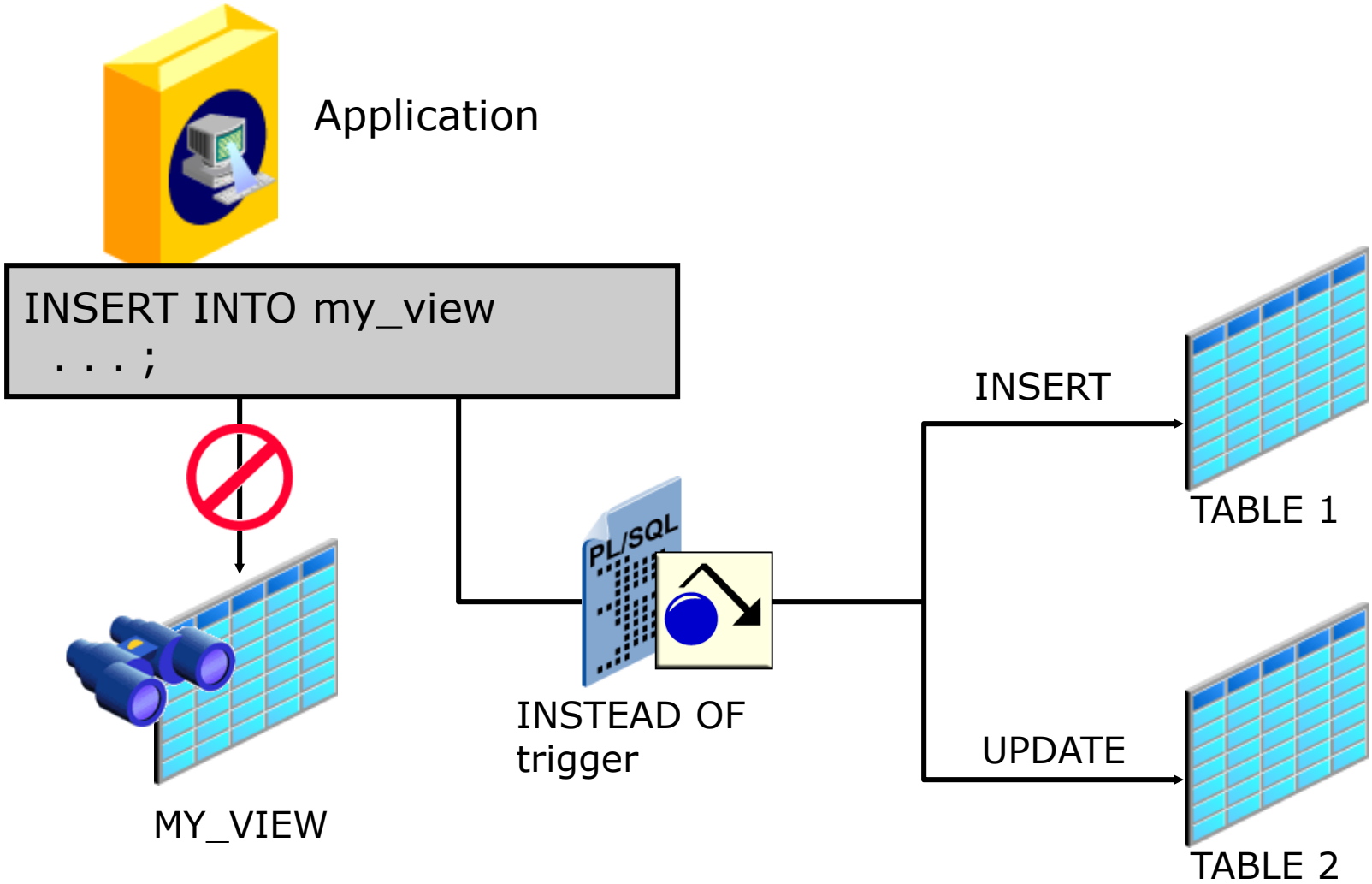
Create or replace trigger <trigger_name>

Instead of insert on<table_name/view_name>

for each row

<pl_sql

4.2: Trigger Types
INSTEAD OF Triggers

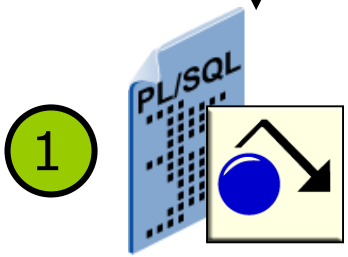


4.2: Trigger Types
Instead of Trigger

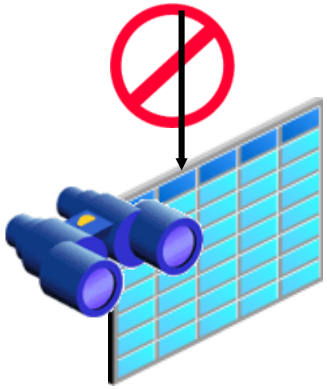
```
create or replace trigger emp_details_insert
Instead of insert on emp_details
for each row
begin

insert into emp(EMPNO,ENAME,JOB,MGR,HIREDATE,DEPTNO)
values(:new.empno,:new.ename,:new.job,:new.mgr,:new.hiredate,:new.deptno);

insert into emp_addr(EMPNO,ADDRESS,CONTACT)
values (:new.empno,:new.address,:new.contact);

end;
/
```

4.2: Trigger Types
Creating an INSTEAD OF Trigger: Example

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

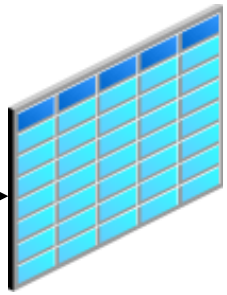


1 INSTEAD OF INSERT into EMP_DETAILS

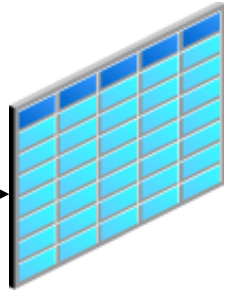


EMP_DETAILS view

2



INSERT into NEW_EMPS table



3 UPDATE NEW_DEPTS table

4.2: Trigger Types
Example of Instead of Trigger

EMP table

EMPNO
ENAME
JOB
MGR
HIREDATE
SAL
COMM
DEPTNO

EMP_ADDR table

EMPNO
ADDRESS
CONTACT

4.2: Trigger Types
Defining Complex View

```
create view emp_details
as
select e.empno,ename,address,contact,job,mgr,hiredate,deptno from emp
e,emp_addr a
where e.empno = a.empno
```

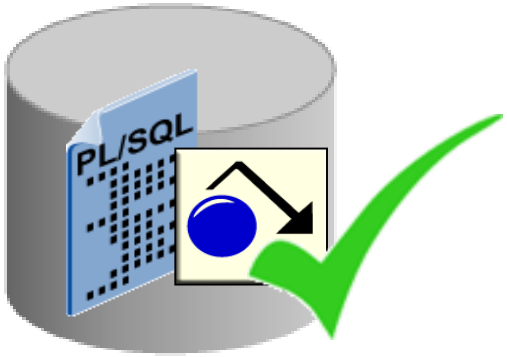
4.2: Trigger Types
Creating an INSTEAD OF Trigger to Perform DML on Complex Views

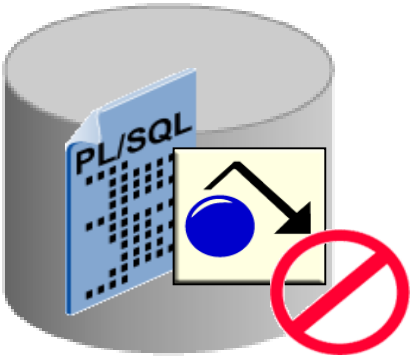
```
CREATE TABLE new_emps AS
 SELECT employee_id,last_name,salary,department_id
      FROM employees;

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;
```

4.2: Trigger Types
The Status of a Trigger

A trigger is in either of two distinct modes:

- Enabled: The trigger runs its trigger action if a triggering statement is issued and the trigger restriction (if any) evaluates to true (default).
- Disabled: The trigger does not run its trigger action, even if a triggering statement is issued and the trigger restriction (if any) would evaluate to true.





4.2: Trigger Types
Creating a Disabled Trigger

Before Oracle Database 11g, if you created a trigger whose body had a PL/SQL compilation error, then DML to the table failed.

In Oracle Database 11g, you can create a disabled trigger and then enable it only when you know it will be compiled successfully.

```
CREATE OR REPLACE TRIGGER mytrg
  BEFORE INSERT ON mytable FOR EACH ROW
  DISABLE
BEGIN
  :New.ID := my_seq.Nextval;
. . .
END;
/
```

4.2: Trigger Types
Managing Triggers Using the ALTER and DROP SQL Statements

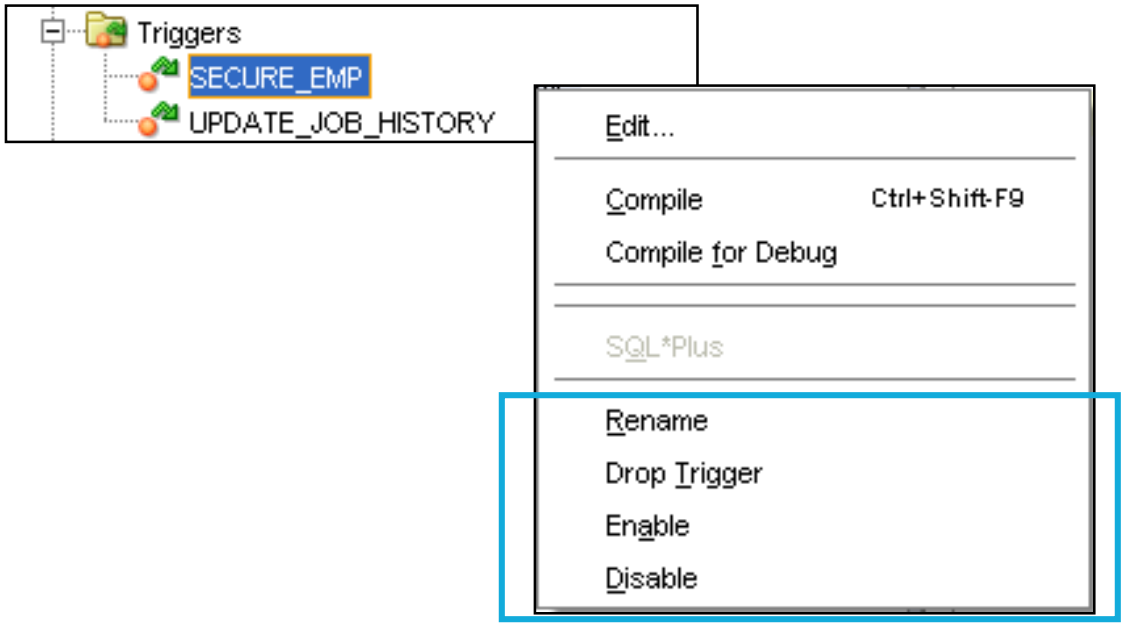
```
-- Disable or reenable a database trigger:

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;
```

```
-- Remove a trigger from the database:

DROP TRIGGER trigger_name;
```

4.2: Trigger Types
Managing Triggers Using SQL Developer



4.2: Trigger Types
Testing Triggers

Test each triggering data operation, as well as non-triggering 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.

4.2: Trigger Types
Viewing Trigger Information

You can view the following trigger information:

Data Dictionary View	Description
USER_OBJECTS	Displays object information
USER/ALL/DBA_TRIGGERS	Displays trigger information
USER_ERRORS	Displays PL/SQL syntax errors for a trigger

4.2: Trigger Types
Using USER_TRIGGERS

```
DESCRIBE user_triggers
```

```
DESCRIBE user_triggers
Name                           Null     Type
------------------------------ -------- ----------------------------------------------------
TRIGGER_NAME                            VARCHAR2(30)
TRIGGER_TYPE                            VARCHAR2(16)
TRIGGERING_EVENT                        VARCHAR2(227)
TABLE_OWNER                             VARCHAR2(30)
BASE_OBJECT_TYPE                        VARCHAR2(16)
TABLE_NAME                              VARCHAR2(30)
COLUMN_NAME                             VARCHAR2(4000)
REFERENCING_NAMES                       VARCHAR2(128)
WHEN_CLAUSE                             VARCHAR2(4000)
STATUS                                  VARCHAR2(8)
DESCRIPTION                             VARCHAR2(4000)
ACTION_TYPE                             VARCHAR2(11)
TRIGGER_BODY                            LONG()
CROSSEDITION                            VARCHAR2(7)
BEFORE_STATEMENT                        VARCHAR2(3)
BEFORE_ROW                              VARCHAR2(3)
AFTER_ROW                               VARCHAR2(3)
AFTER_STATEMENT                         VARCHAR2(3)
INSTEAD_OF_ROW                          VARCHAR2(3)
FIRE_ONCE                               VARCHAR2(3)
APPLY_SERVER_ONLY                       VARCHAR2(3)

21 rows selected
```

```
SELECT trigger_type, trigger_body
FROM user_triggers
WHERE trigger_name = 'SECURE_EMP';
```

4.2: Trigger Types
Managing Triggers

Disable or re-enable a database trigger Alter trigger trigger_name disable | enable

Disable or re-enable all triggers for a table Alter table table name disable | enable all triggers

Recompile a trigger Alter trigger name compile

SUMMARY

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

Review Question

Question 1: Triggers should not issue Transaction Control Statements (TCL)

- True / False

Question 2: BEFORE DROP and AFTER DROP triggers are fired when a schema object is dropped

- True / False

Question 3: The :new and :old records must be used in WHEN clause with a colon

- True / False

Review Question

Question 4: A ___ is a table that is currently being modified by a DML statement

Question 5: A ___ is fired once on behalf of the triggering statement, regardless of the number of rows in the table that the triggering statement affects