16

Creating Database Triggers

Copyright © Oracle Corporation, 2001. All rights reserved.

Objectives

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

- Describe different types of triggers
- Describe database triggers and their use
- Create database triggers
- Describe database trigger firing rules
- Remove database triggers

Copyright © Oracle Corporation, 2001. All rights reserved.

Types of Triggers

A trigger:

- Is a PL/SQL block or a PL/SQL procedure associated with a table, view, schema, or the database
- Executes implicitly whenever a particular event takes place
- Can be either:
  - 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

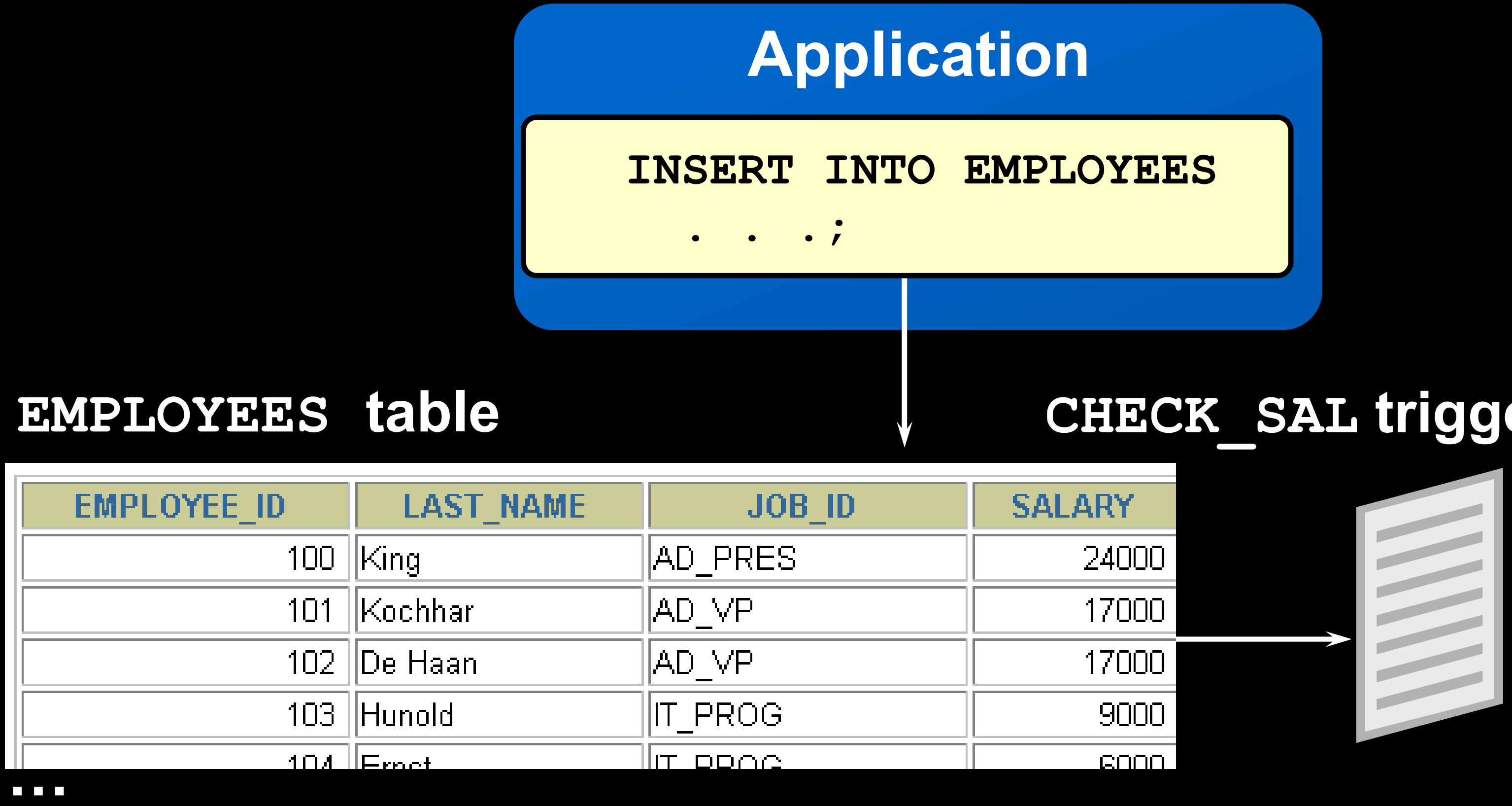
Copyright © Oracle Corporation, 2001. All rights reserved.

Guidelines for Designing Triggers

- Design triggers to:
  - Perform related actions
  - Centralize global operations
- Do not design triggers:
  - Where functionality is already built into the Oracle server
  - That duplicate other triggers
- 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.

Copyright © Oracle Corporation, 2001. All rights reserved.

Database Trigger: Example



Application

```
INSERT INTO EMPLOYEES
. . .;
```

`EMPLOYEES` table

`CHECK_SAL` trigger

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

…

Copyright © Oracle Corporation, 2001. All rights reserved.

Creating DML Triggers

A triggering statement contains:

- Trigger timing
  - For table: `BEFORE`, `AFTER`
  - For view: `INSTEAD OF`
- Triggering event: `INSERT`, `UPDATE`, or `DELETE`
- Table name: On table, view
- Trigger type: Row or statement
- `WHEN` clause: Restricting condition
- Trigger body: PL/SQL block

Copyright © Oracle Corporation, 2001. All rights reserved.

DML Trigger Components

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.

Copyright © Oracle Corporation, 2001. All rights reserved.

DML Trigger Components

Triggering user event: Which DML statement causes the trigger to execute? You can use any of the following:

- `INSERT`
- `UPDATE`
- `DELETE`

Copyright © Oracle Corporation, 2001. All rights reserved.

DML Trigger Components

Trigger type: Should the trigger body execute for each row the statement affects or only once?

- Statement: The trigger body executes once for the triggering event. This is the default. A statement trigger fires once, even if no rows are affected at all.
- Row: The trigger body executes once for each row affected by the triggering event. A row trigger is not executed if the triggering event affects no rows.

Copyright © Oracle Corporation, 2001. All rights reserved.

DML Trigger Components

Trigger body: What action should the trigger perform?

The trigger body is a PL/SQL block or a call to a procedure.

Copyright © Oracle Corporation, 2001. All rights reserved.

Firing Sequence

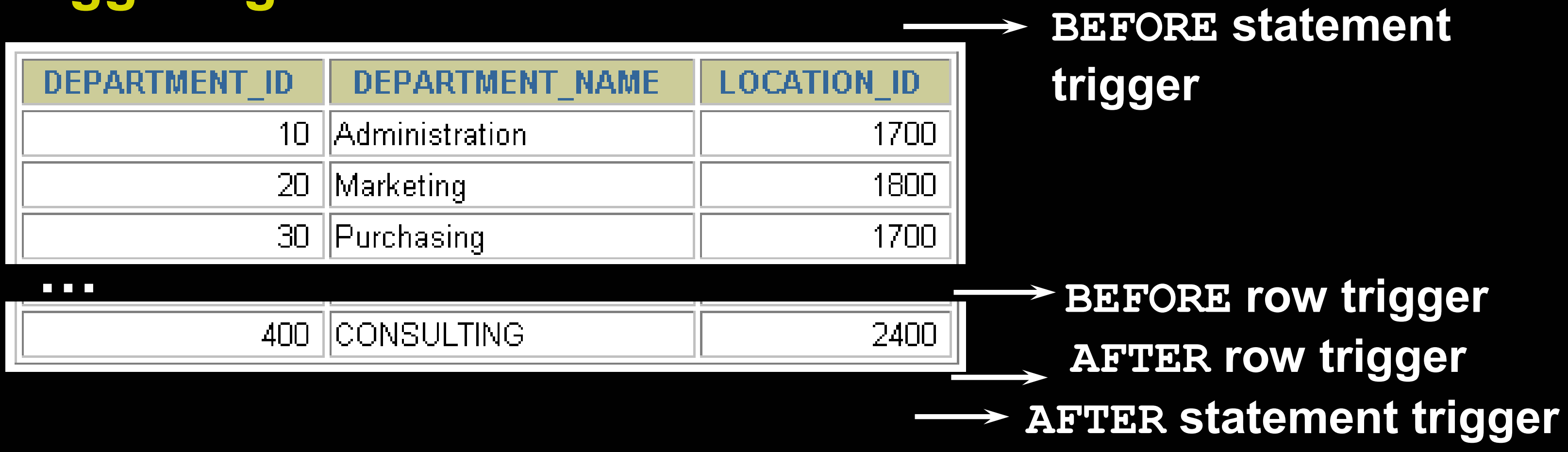
Use the following firing sequence for a trigger on a table, when a single row is manipulated:

DML statement

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

```
1 row created.
```

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

Copyright © Oracle Corporation, 2001. All rights reserved.

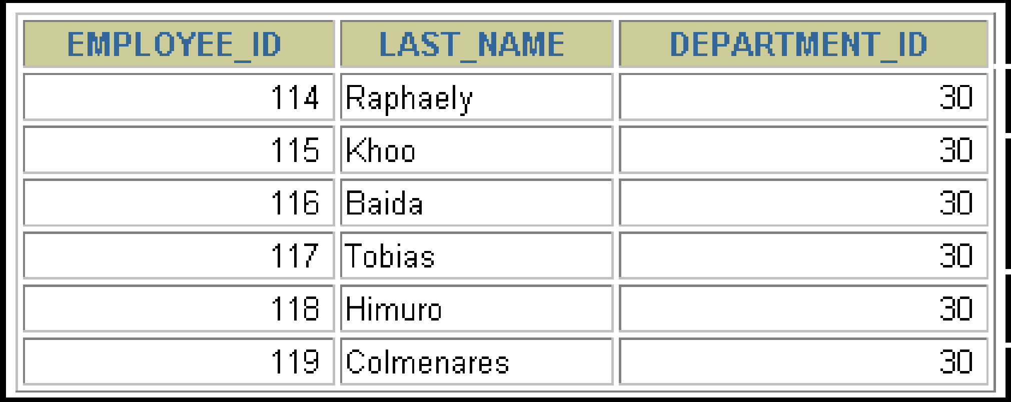
Firing Sequence

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;
```

```
6 rows updated.
```

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

Copyright © Oracle Corporation, 2001. All rights reserved.

Syntax for Creating DML Statement Triggers

Syntax:

```
CREATE [OR REPLACE] TRIGGER trigger_name
    timing
        event1 [OR event2 OR event3]
            ON table_name
trigger_body
```

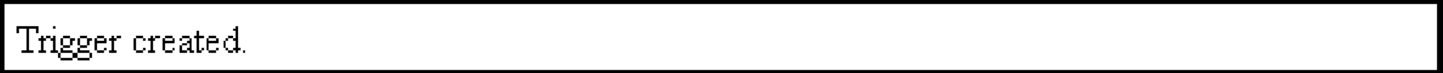
Note: Trigger names must be unique with respect to other triggers in the same schema.

Copyright © Oracle Corporation, 2001. All rights reserved.

Creating DML Statement Triggers

Example:

```
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;
/
```

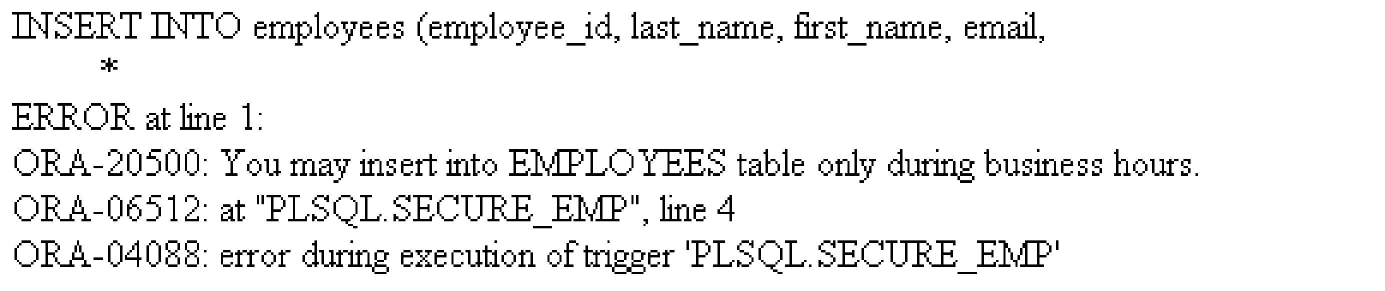


```
Trigger created.
```

Copyright © Oracle Corporation, 2001. All rights reserved.

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'
```

Copyright © Oracle Corporation, 2001. All rights reserved.

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;
```

Copyright © Oracle Corporation, 2001. All rights reserved.

Creating a DML Row Trigger

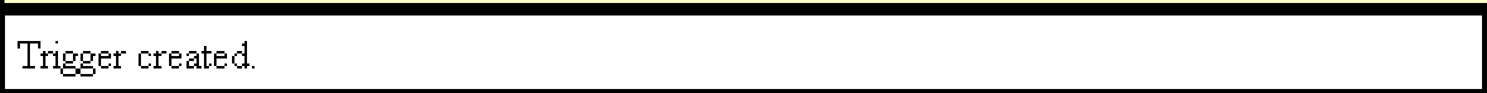
Syntax:

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

Copyright © Oracle Corporation, 2001. All rights reserved.

Creating DML Row Triggers

```
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 this amount');
     END IF;
END;
/
```

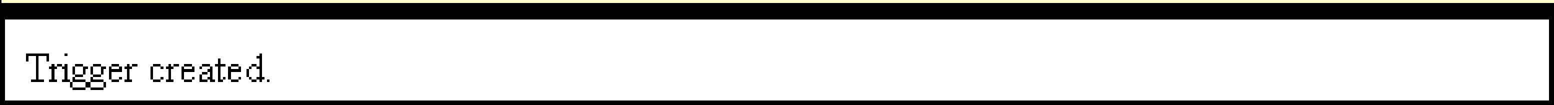


```
Trigger created.
```

Copyright © Oracle Corporation, 2001. All rights reserved.

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_table (user_name, timestamp,
     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;
/
```



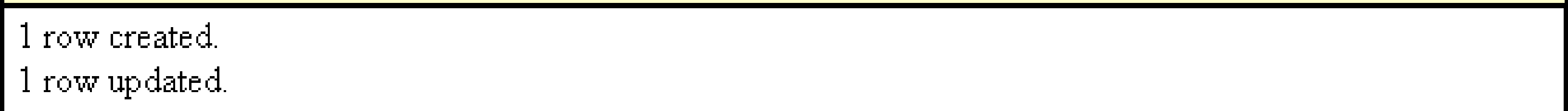
```
Trigger created.
```

Copyright © Oracle Corporation, 2001. All rights reserved.

Using OLD and NEW Qualifiers: Example Using Audit_Emp_Table

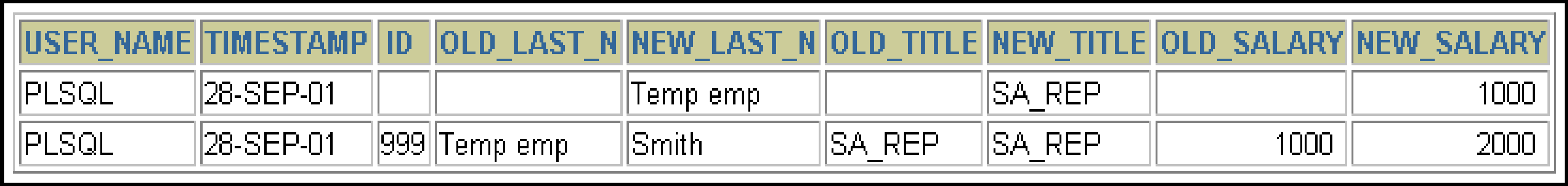
```
INSERT INTO employees
        (employee_id, last_name, job_id, salary, ...)
 VALUES (999, 'Temp emp', 'SA_REP', 1000, ...);

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



```
1 row created.
1 row updated.
```

```
SELECT user_name, timestamp, ... FROM audit_emp_table
```

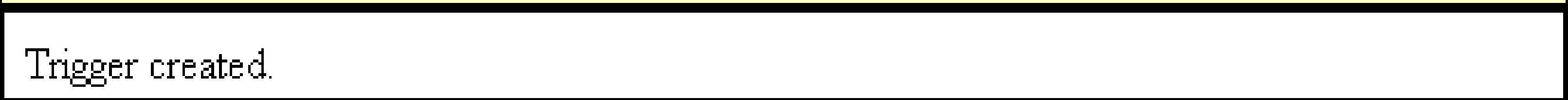


USER_NAME	TIMESTAMP	ID	OLD_LAST_N	NEW_LAST_N	OLD_TITLE	NEW_TITLE	OLD_SALARY	NEW_SALARY
PLSQL	28-SEP-01			Temp emp		SA_REP		1000
PLSQL	28-SEP-01	999	Temp emp	Smith	SA_REP	SA_REP	1000	2000

Copyright © Oracle Corporation, 2001. All rights reserved.

Restricting a Row Trigger

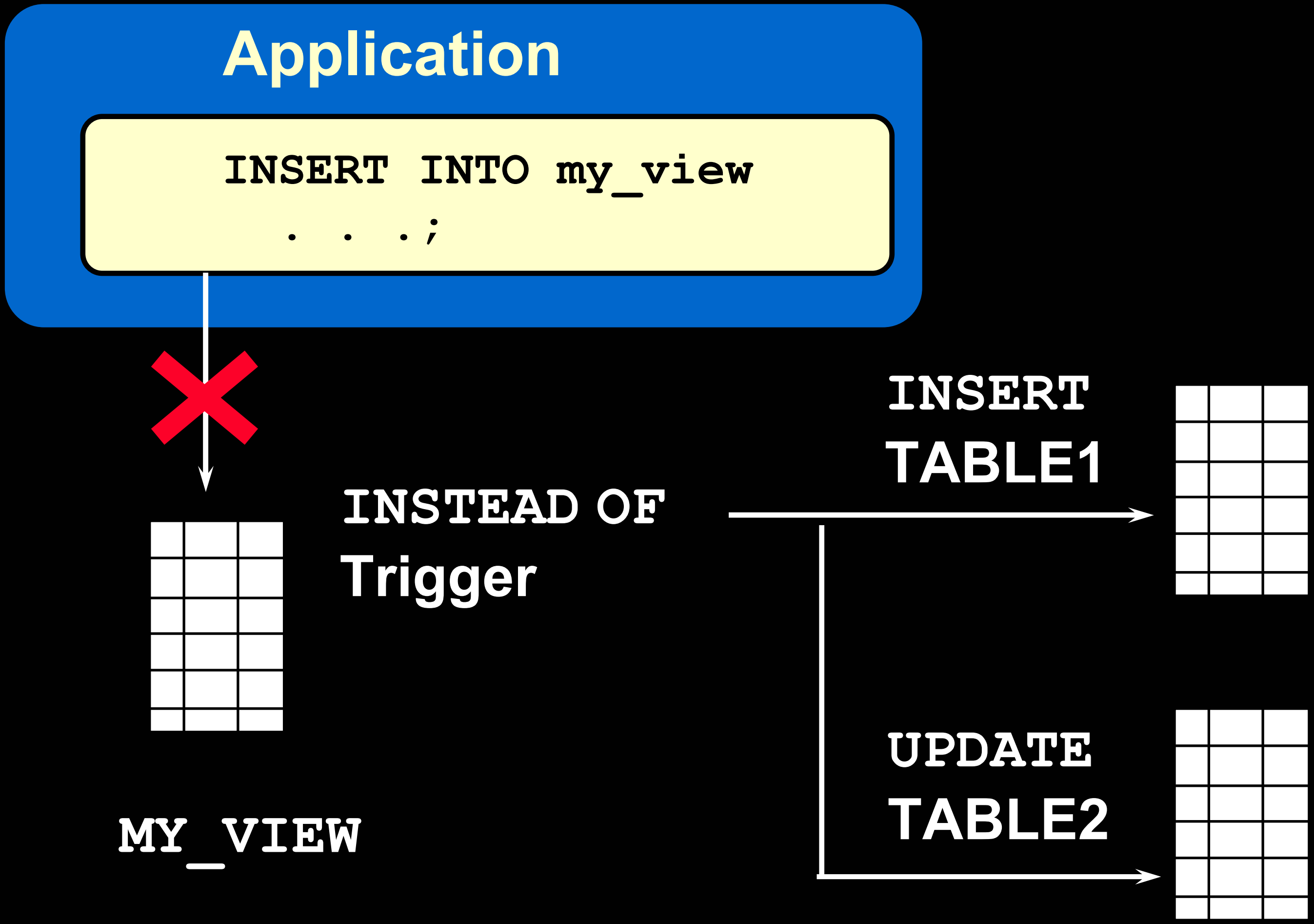
```
CREATE OR REPLACE TRIGGER derive_commission_pct
  BEFORE INSERT OR UPDATE OF salary ON employees
  FOR EACH ROW
  WHEN (NEW.job_id = 'SA_REP')
BEGIN
  IF  INSERTING
     THEN :NEW.commission_pct := 0;
  ELSIF :OLD.commission_pct IS NULL
     THEN :NEW.commission_pct := 0;
  ELSE
    :NEW.commission_pct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



```
Trigger created.
```

Copyright © Oracle Corporation, 2001. All rights reserved.

INSTEAD OF Triggers



Application

```
INSERT INTO my_view
  . . .;
```

Copyright © Oracle Corporation, 2001. All rights reserved.

Creating an INSTEAD OF Trigger

Syntax:

```
CREATE [OR REPLACE] TRIGGER trigger_name
  INSTEAD OF
    event1 [OR event2 OR event3]
      ON view_name
   [REFERENCING OLD AS old | NEW AS new]
[FOR EACH ROW]
trigger_body
```

Copyright © Oracle Corporation, 2001. All rights reserved.

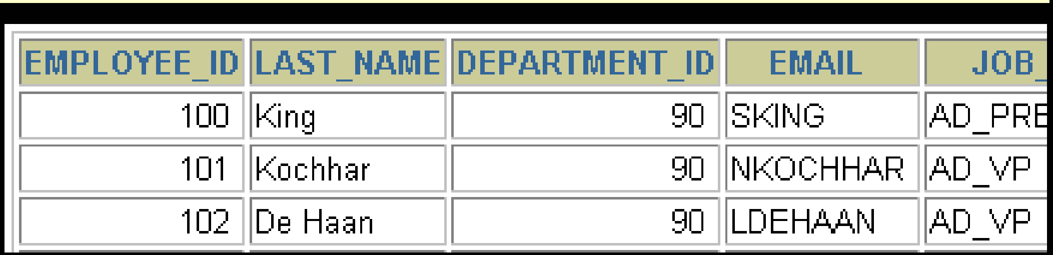
Creating an INSTEAD OF Trigger

INSERT into EMP_DETAILS that is based on EMPLOYEES and DEPARTMENTS tables

1

```
INSERT  INTO  emp_details(employee_id, ... )
VALUES(9001,'ABBOTT',3000,10,'abbott.mail.com','HR_MAN');
```

INSTEAD OF INSERT
into EMP_DETAILS →



EMPLOYEE_ID	LAST_NAME	DEPARTMENT_ID	EMAIL	JOB_
100	King	90	SKING	AD_PRE
101	Kochhar	90	NKOCHHAR	AD_VP
102	De Haan	90	LDEHAAN	AD_VP

...

Copyright © Oracle Corporation, 2001. All rights reserved.

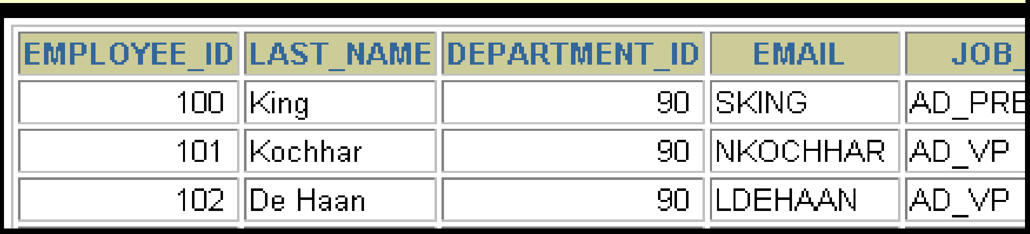
Creating an INSTEAD OF Trigger

INSERT into EMP_DETAILS that is based on EMPLOYEES and DEPARTMENTS tables

1

```
INSERT INTO emp_details(employee_id, ... )
VALUES(9001,'ABBOTT',3000,10,'abbott.mail.com','HR_MAN');
```

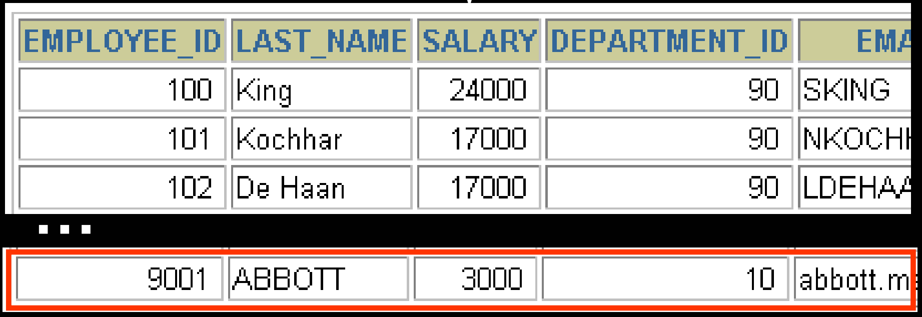
INSTEAD OF INSERT into EMP_DETAILS



EMPLOYEE_ID	LAST_NAME	DEPARTMENT_ID	EMAIL	JOB_
100	King	90	SKING	AD_PRE
101	Kochhar	90	NKOCHHAR	AD_VP
102	De Haan	90	LDEHAAN	AD_VP

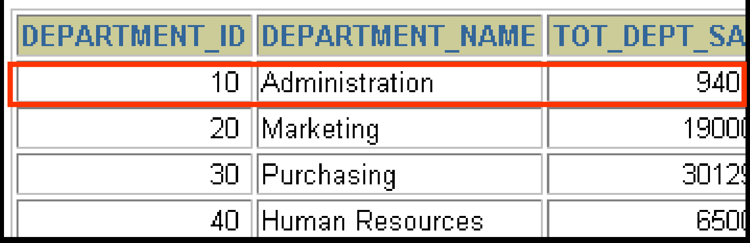
...

2 INSERT into NEW_EMPS



EMPLOYEE_ID	LAST_NAME	SALARY	DEPARTMENT_ID	EMA
100	King	24000	90	SKING
101	Kochhar	17000	90	NKOCHH
102	De Haan	17000	90	LDEHAA
...				
9001	ABBOTT	3000	10	abbott.m

3 UPDATE NEW_DEPTS



DEPARTMENT_ID	DEPARTMENT_NAME	TOT_DEPT_SA
10	Administration	940
20	Marketing	1900
30	Purchasing	3012
40	Human Resources	650

...

Copyright © Oracle Corporation, 2001. All rights reserved.

Differentiating Between Database Triggers and Stored Procedures

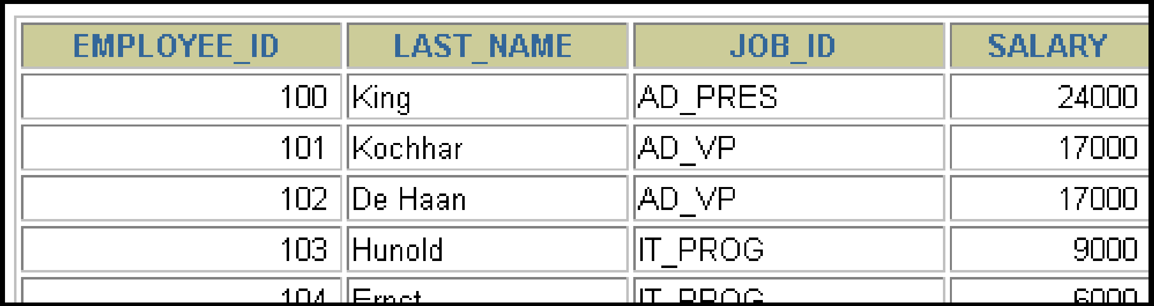
Triggers	Procedures
Defined with `CREATE TRIGGER`	Defined with `CREATE PROCEDURE`
Data dictionary contains source code in `USER_TRIGGERS`	Data dictionary contains source code in `USER_SOURCE`
Implicitly invoked	Explicitly invoked
`COMMIT`, `SAVEPOINT`, and `ROLLBACK` are not allowed	`COMMIT`, `SAVEPOINT`,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

Copyright © Oracle Corporation, 2001. All rights reserved.

Differentiating Between Database Triggers and Form Builder Triggers

```
INSERT INTO EMPLOYEES
  . . .;
```

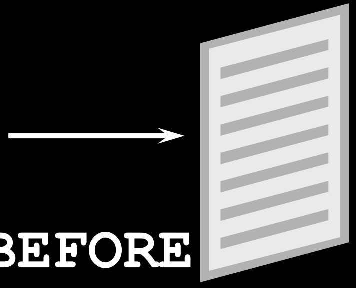
EMPLOYEES table



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

...

CHECK_SAL trigger



BEFORE INSERT row

Copyright © Oracle Corporation, 2001. All rights reserved.

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
```

Copyright © Oracle Corporation, 2001. All rights reserved.

DROP TRIGGER Syntax

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

```
DROP TRIGGER trigger_name;
```

Example:

```
DROP TRIGGER secure_emp;
```

```
Trigger dropped.
```

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

Copyright © Oracle Corporation, 2001. All rights reserved.

Trigger Test Cases

- 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 upon other triggers.
- Test the effect of other triggers upon the trigger.

Copyright © Oracle Corporation, 2001. All rights reserved.

Trigger Execution Model and Constraint Checking

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

Copyright © Oracle Corporation, 2001. All rights reserved.

Trigger Execution Model and Constraint Checking: Example

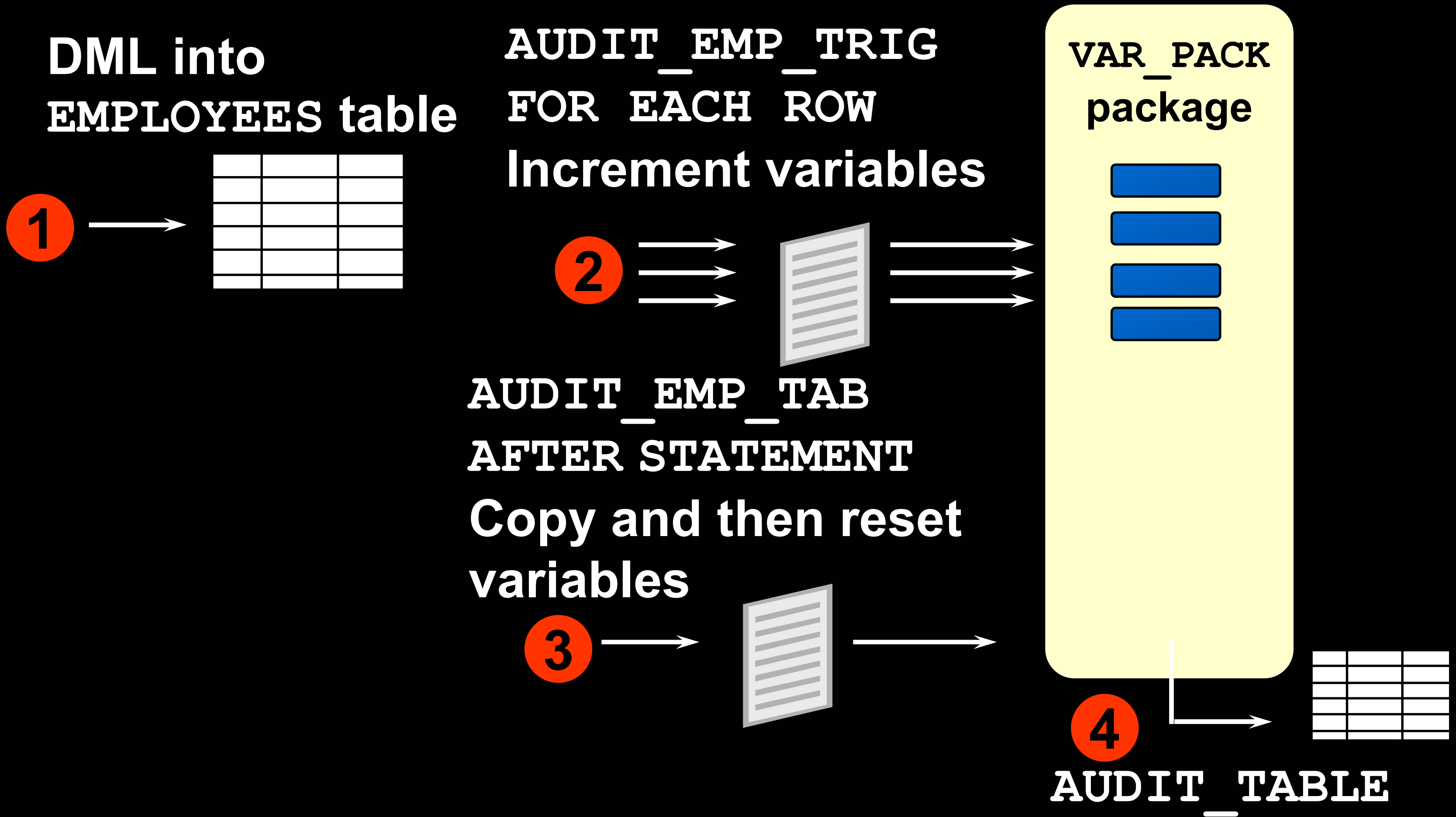
```
UPDATE employees SET department_id = 999
 WHERE employee_id = 170;
-- Integrity constraint violation error
```

```
CREATE OR REPLACE TRIGGER constr_emp_trig
 AFTER UPDATE ON employees
  FOR EACH ROW
BEGIN
  INSERT INTO departments
    VALUES (999, 'dept999', 140, 2400);
END;
/
```

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

Copyright © Oracle Corporation, 2001. All rights reserved.

A Sample Demonstration for Triggers Using Package Constructs



DML into `EMPLOYEES` table

1

`AUDIT_EMP_TRIG`

`FOR EACH ROW`

Increment variables

2

`VAR_PACK` package

`AUDIT_EMP_TAB`

`AFTER STATEMENT`

Copy and then reset variables

3

4

`AUDIT_TABLE`

Copyright © Oracle Corporation, 2001. All rights reserved.

After Row and After Statement Triggers

```
CREATE OR REPLACE TRIGGER audit_emp_trig
AFTER    UPDATE or INSERT or DELETE on EMPLOYEES
FOR EACH ROW
BEGIN
  IF      DELETING    THEN  var_pack.set_g_del(1);
  ELSIF   INSERTING   THEN  var_pack.set_g_ins(1);
  ELSIF   UPDATING ('SALARY')
                     THEN  var_pack.set_g_up_sal(1);
  ELSE    var_pack.set_g_upd(1);
  END IF;
END audit_emp_trig;
/
```

```
CREATE OR REPLACE TRIGGER audit_emp_tab
AFTER    UPDATE or INSERT or DELETE on employees
BEGIN
  audit_emp;
END audit_emp_tab;
/
```

Copyright © Oracle Corporation, 2001. All rights reserved.

Demonstration: VAR_PACK Package Specification

var_pack.sql

```
CREATE OR REPLACE PACKAGE var_pack
IS
-- these functions are used to return the
-- values of package variables
  FUNCTION g_del RETURN NUMBER;
  FUNCTION g_ins RETURN NUMBER;
  FUNCTION g_upd RETURN NUMBER;
  FUNCTION g_up_sal RETURN NUMBER;
-- these procedures are used to modify the
-- values of the package variables
  PROCEDURE set_g_del    (p_val  IN  NUMBER);
  PROCEDURE set_g_ins    (p_val  IN  NUMBER);
  PROCEDURE set_g_upd    (p_val  IN  NUMBER);
  PROCEDURE set_g_up_sal (p_val  IN  NUMBER);
END var_pack;
/
```

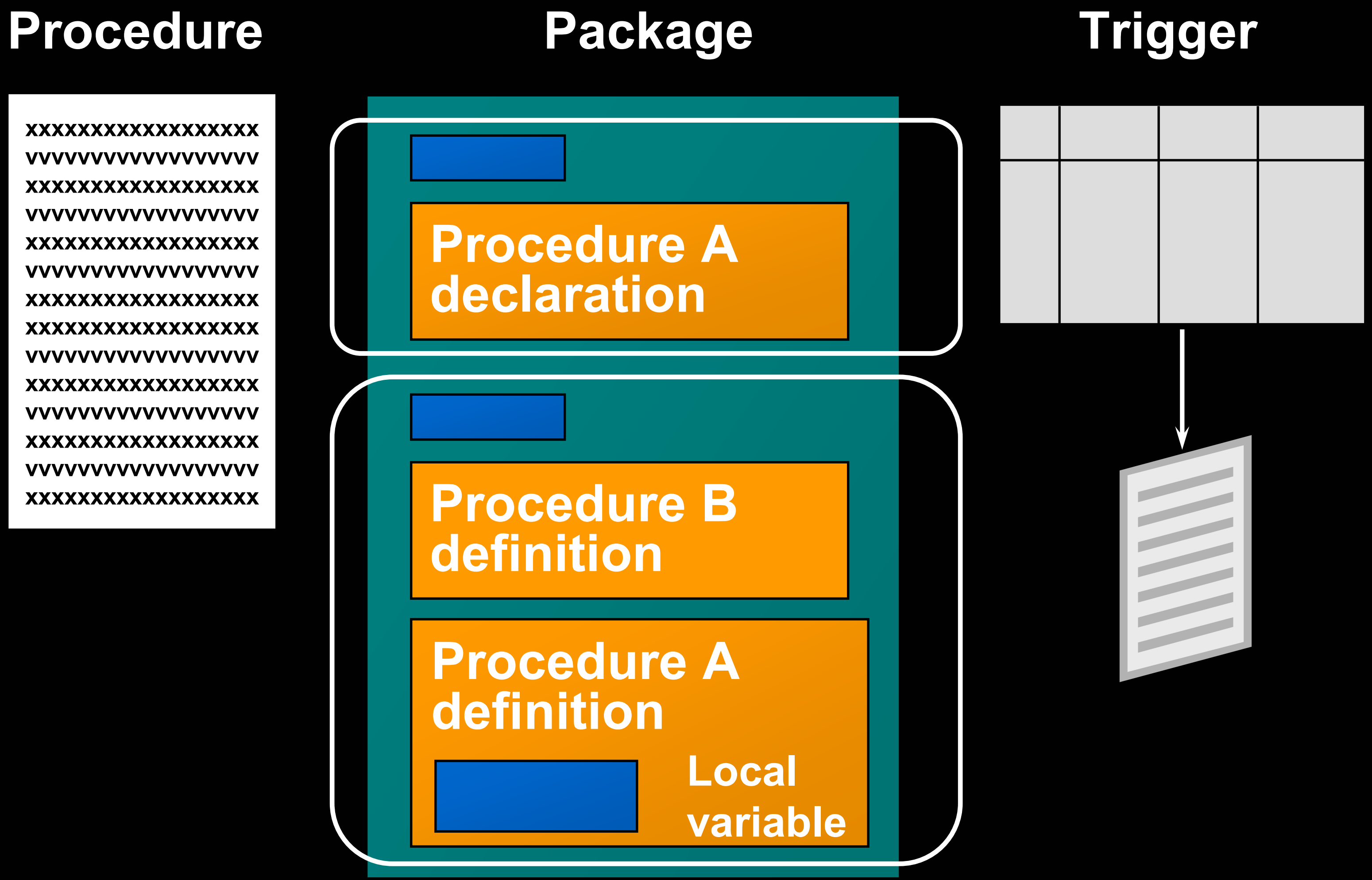
Copyright © Oracle Corporation, 2001. All rights reserved.

Demonstration: Using the AUDIT_EMP Procedure

```
CREATE OR REPLACE PROCEDURE audit_emp  IS
  v_del     NUMBER   := var_pack.g_del;
  v_ins     NUMBER   := var_pack.g_ins;
  v_upd     NUMBER   := var_pack.g_upd;
  v_up_sal  NUMBER   := var_pack.g_up_sal;
BEGIN
  IF  v_del + v_ins + v_upd != 0  THEN
    UPDATE audit_table SET
      del = del + v_del, ins = ins + v_ins,
      upd = upd + v_upd
    WHERE user_name=USER AND tablename='EMPLOYEES'
    AND   column_name IS NULL;
  END IF;
  IF v_up_sal != 0   THEN
    UPDATE audit_table SET upd = upd + v_up_sal
    WHERE user_name=USER AND tablename='EMPLOYEES'
    AND   column_name = 'SALARY';
  END IF;
-- resetting global variables in package VAR_PACK
  var_pack.set_g_del (0); var_pack.set_g_ins (0);
  var_pack.set_g_upd (0); var_pack.set_g_up_sal (0);
END audit_emp;
```

Copyright © Oracle Corporation, 2001. All rights reserved.

Summary



Copyright © Oracle Corporation, 2001. All rights reserved.

Practice 16 Overview

This practice covers the following topics:

- Creating statement and row triggers
- Creating advanced triggers to add to the capabilities of the Oracle database

Copyright © Oracle Corporation, 2001. All rights reserved.

PRAKTIKUM PL/SQL
(Trigger)

OLEH :
WIRATMOKO YUWONO, ST

PRAKTIKUM PL/SQL (Trigger)

1. Statement Trigger dan Row Trigger

Syntax Statement trigger :

```
CREATE [OR REPLACE] TRIGGER trigger_name
timing
            event1 [OR event2 OR event3]
            ON table_name
 trigger_body(PL/SQL)
```

Syntax Row trigger

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

2. Contoh statement trigger

Penulis membuat table **log** pada user scott dengan field sebagai berikut :

- Field **Tanggal** Type Data **Date**
- Field **Komentar** Type Data **Varchar2(100)**

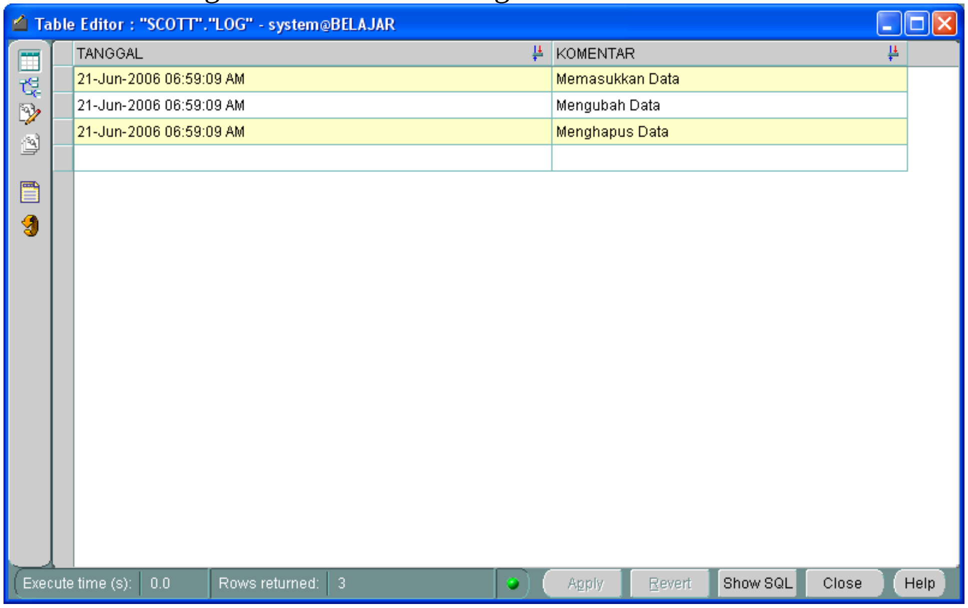
A. Mendeteksi operasi DML pada table EMP diuser Scott dan kemudian histori operasi DML tersebut dicatat pada table log

```
CREATE OR REPLACE TRIGGER TLOG
        AFTER
        INSERT
        OR UPDATE
        OR DELETE ON EMP
DECLARE
BEGIN
 IF INSERTING THEN
  insert into log (tanggal,komentar) values (sysdate,'Memasukkan Data');
 ELSIF UPDATING THEN
  insert into log (tanggal,komentar) values (sysdate,'Mengubah Data');
 ELSIF DELETING THEN
  insert into log (tanggal,komentar) values (sysdate,'Menghapus Data');
 END IF;
END;
```

- Kemudian lakukan operasi DML sebagai berikut :

```
insert into emp values (8000,'Moko','SALESMAN',7900,SYSDATE,8000,null,null);
update emp set ename='Wiratmoko',sal=10000 where empno=8000;
delete from emp where empno=8000;
commit;
```

- Maka table log akan berisi data sebagai berikut :



TANGGAL	KOMENTAR
21-Jun-2006 06:59:09 AM	Memasukkan Data
21-Jun-2006 06:59:09 AM	Mengubah Data
21-Jun-2006 06:59:09 AM	Menghapus Data

Execute time (s): 0.0 Rows returned: 3

3. Contoh Row Trigger

Penulis membuat beberapa table pada user scott dengan field sebagai berikut :

- Table **Barang** :

1. Field **id** type data **number(10)** Not Null
2. Field **nama** type data **varchar2(50)** Not Null
3. Field **spesifikasi** type data **varchar2(200)** Null
4. Field **jumlah** type data **number(5)** Null
5. Field **satuan** type data **varchar2(20)** Null
6. Field **tanggal** type data **date** null

- Table **Log :**

1. Field **tanggal** type data **date** Not Null
2. Field **komentar** type data **varchar2(100)** Not Null

1. Contoh kasus

1. Buatlah row trigger (namai TLAT1) yang melakukan pengecekan terhadap update data pada tabel barang. Timing trigger yang dipakai adalah BEFORE. Body trigger berisi, insert data pada tabel LOG, dengan ketentuan.

- Field tanggal berisi data baru dari field tanggal dari tabel barang
- Field komentar berisi ‘Ubah data dari ‘ + data lama dari field nama dari tabel barang

Jawaban :

```
CREATE OR REPLACE TRIGGER TLAT1
        BEFORE
        UPDATE
        ON barang
        FOR EACH ROW
DECLARE
BEGIN
        Insert into log (tanggal,komentar) values (:new.tanggal,'Ubah data dari '||
        :old.nama);
END;
```

- Kemudian lakukan operasi DML sebagai berikut :

```
insert into barang (id,nama,spesifikasi,jumlah,satuan,tanggal) values (1,'Jeruk',
'Jeruk Malang',10,'Buah',sysdate);
update barang set nama='Mangga',spesifikasi='Mangga Gadung',
tanggal='01-Jan-2006'  where id=1;
commit;
```

- Maka table log akan berisi data sebagai berikut :

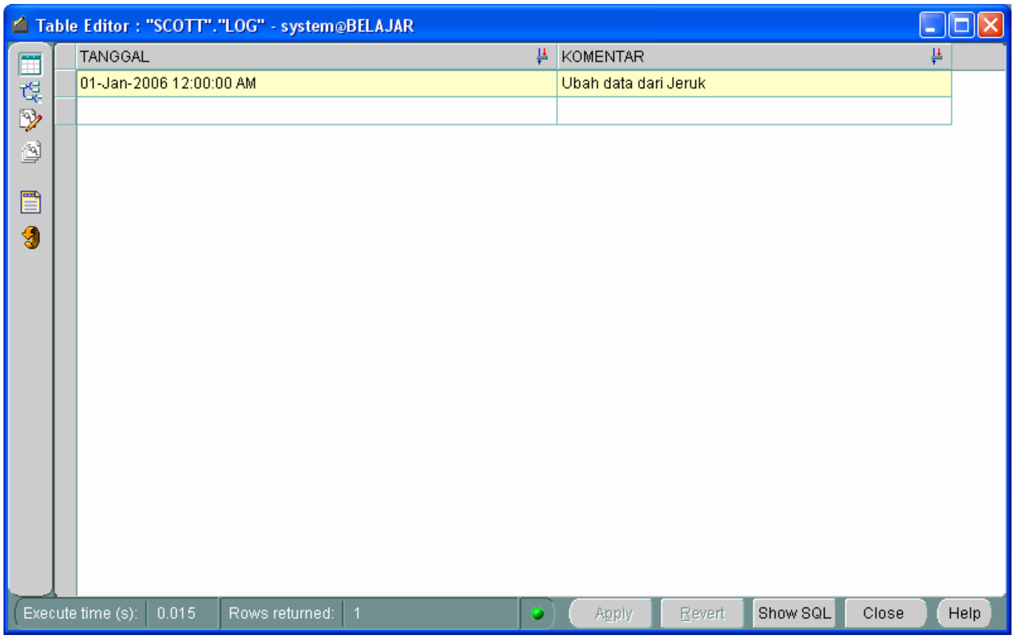


Table Editor : "SCOTT"."LOG" - system@BELAJAR

TANGGAL	KOMENTAR
01-Jan-2006 12:00:00 AM	Ubah data dari Jeruk

Execute time (s): 0.015 Rows returned: 1

2. Buatlah row trigger (namai TLAT2) yang melakukan pengecekan terhadap delete data pada tabel barang. Timing trigger yang dipakai adalah AFTER. Body trigger berisi insert data pada tabel LOG dengan ketentuan.
   - Jika data pada field jumlah pada field barang yang dihapus adalah berisi kurang dari 50 maka masukkan data pada tabel LOG.
     - Field tanggal berisi data lama dari field tanggal dari tabel barang
     - Field komentar berisi ‘hapus Data dengan jumlah kurang dari 50 yaitu ‘ + data lama dari field jumlah
   - Jika data pada field jumlah pada field barang yang dihapus adalah berisi lebih dari 50 maka masukkan data pada tabel LOG.
     - Field tanggal berisi data lama dari field tanggal dari tabel barang
     - Field komentar berisi ‘hapus Data dengan jumlah lebih dari 50 yaitu ‘ + data lama dari field jumlah

Jawaban :

```
CREATE OR REPLACE TRIGGER TLAT2
        AFTER
        DELETE
        ON barang
        FOR EACH ROW
DECLARE
BEGIN
        IF :old.jumlah<50 then
                Insert into log (tanggal,komentar) values (:old.tanggal,'Hapus data
                Dengan jumlah kurang dari 50 yaitu '||    :old.jumlah);
        ELSE
                Insert into log (tanggal,komentar) values (:old.tanggal,'Hapus data
                Dengan jumlah lebih dari 50 yaitu '||       :old.jumlah);
        END IF;
END;
```

- Kemudian lakukan operasi DML sebagai berikut :

```
insert into barang (id,nama,spesifikasi,jumlah,satuan,tanggal) values (1,'Jeruk',
'Jeruk Malang',10,'Buah',sysdate);
update barang set nama='Mangga',spesifikasi='Mangga Gadung',
tanggal='01-Jan-2006'  where id=1;
commit;
```

TUGAS PRAKTIKUM

Buat Laporan Resmi dari praktikum ini.

1. Tuliskan script membuat table dengan field dan tipe data sbb :

   Table Barang :

   a. field **id_barang** type data **number(10)** Not Null
   b. field **nama** type data **varchar2(50)** Not Null
   c. field **spesifikasi** type data **varchar2(200)** Null
   d. field **jumlah** type data **number(5)** Not Null
   e. field **harga** type data **number(20)** Not Null
   f. field **satuan** type data **varchar2(20)** Null

   Table Transaksi

   a. field **nomor_transaksi** type data **number(10)** Not Null
   b. field **id_barang** type data **number(10)** Not Null
   c. field **tanggal** type data **date** Not Null
   d. field **jumlah** type data **number(5)** Not Null

   Table history

   a. field **id_barang** type data **number(10)** Not Null
   b. field **tanggal** type data **date** Not Null
   c. field **stock** type data **number(10)** Not Null
   d. field **tipe_transaksi** data **varchar2(20)** Not Null

2. Buat row trigger dengan timing BEFORE pada segala operasi DML pada table transaksi dengan aturan sbb :

   - Segala pengubahan data pada field jumlah ditable transaksi akan mengubah data pada field jumlah di table barang. Artinya field jumlah di table barang merupakan data stock barang pada item barang tertentu (atau berasosiasi dengan field id_barang)
   - Segala pengubahan pada field jumlah ditable transaksi akan dicatat pada table history.

     Khusus tipe transaksi : anda isikan ”Tambah Data/Ubah Data/Hapus Data” sesuai dengan tipe DML pada table transaksi.