Table of Contents

[CHAPTER 23: ADVANCED TRIGGERS 2](#_Toc416878829)

[Theory 2](#_Toc416878830)

[AIM 3](#_Toc416878831)

[Lab Exercise 23: ADVANCED TRIGGERS 4](#_Toc416878832)

[1. Statement-level Trigger 5](#_Toc416878833)

[2. Row-level Trigger 15](#_Toc416878834)

[3. Trigger with Conditions 25](#_Toc416878835)

[4. INSTEAD OF Trigger 41](#_Toc416878836)

[5. Compound DML Trigger 45](#_Toc416878837)

[SUMMARY 52](#_Toc416878838)

[REFERENCES 53](#_Toc416878839)

[INDEX 54](#_Toc416878840)

# CHAPTER 23: ADVANCED TRIGGERS

## Theory

In previous chapter, you investigated triggers "ON DATABASE" and "ON SCHEMA" level. In this chapter, you will investigate triggers on Table or View level. It is sometimes called DML triggers. You can create DML trigger on Tables or View to be fired on:

1. INSERT
2. UPDATE
3. DELETE

DML trigger fires due to one or more of the previous events, but the exact time when the trigger is executed in relative to the event can be:

1. BEFORE
2. AFTER
3. COMPOUND (BEFORE & AFTER)

One small thing here is that what do you mean by INSERT, UPDATE, and DELETE events? Do you mean to run your trigger due to executing these DML statements once for the event OR for each row affected by that DML statement? Fortunately, Oracle supports both; use "FOR EACH ROW" clause to run the trigger in row level. Otherwise, it runs on statement level.

If the trigger is created to handle more than one event, say INSERT and UPDATE, then it would be very useful to know what exactly the event while in the trigger code. For this purpose, Oracle provides conditional predicates: DELETING, UPDATING, and INSERTING.

Oracle supports INSTEAD OF trigger; a trigger created on a noneditioning view, or on a nested table column of a noneditioning view. Oracle executes INSTEAD OF trigger instead of the DML statement.

While in row level trigger, you are not allowed to touch (query or DML) the same table you building the trigger on. For example, if you create a trigger "x" on table "t", then you can't query or do any DML statements on "t" table. If you try, you will get "ORA-04091: mutating-table error " exception. To work around this issue, Oracle supports Compound DML trigger. In Compound trigger, the trigger

## AIM

The AIM of the following exercise is to create advanced triggers.

The steps involved will include:

* Statement-level Trigger
* Row-level Trigger
* Trigger with Conditions
* INSTEAD OF Trigger
* Compound DML Trigger

In general, lab exercises are done in sequential order. Thus, it is assumed that you successfully completed the previous labs. However, not all previous labs are required. Please be sure to run the following lab before proceeding:

* Installing Oracle Database 12c.

Estimated Completion Time:

35 minutes

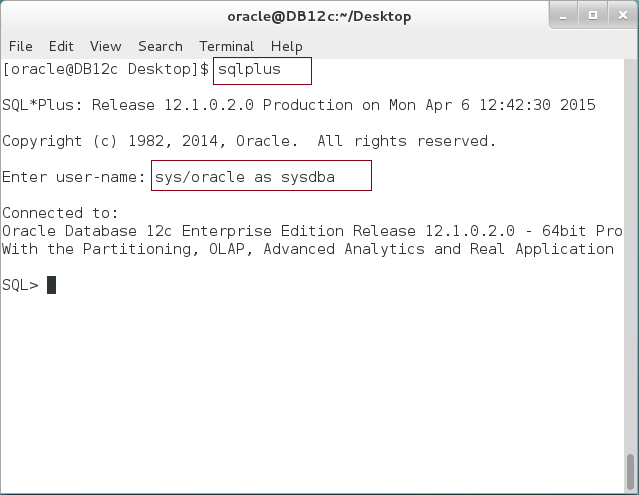
# Lab Exercise 23: ADVANCED TRIGGERS

|  |
| --- |
|  |

## Statement-level Trigger

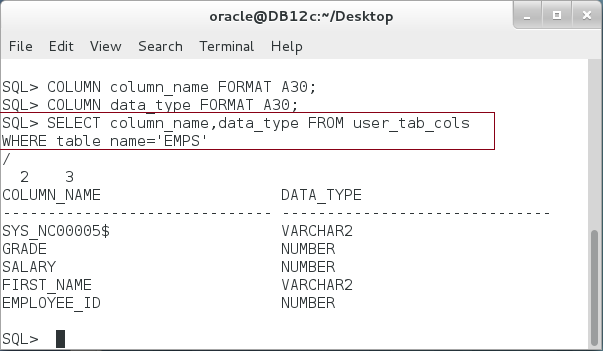
**Step 1:** Open the Terminal, open SQL\*Plus console and connect to hr schema.

|  |  |
| --- | --- |
| Command | Description |
| sqlplus | Open SQL\*Plus console. |
| sys/oracle as sysdba | connect to **sys** schema. |

****

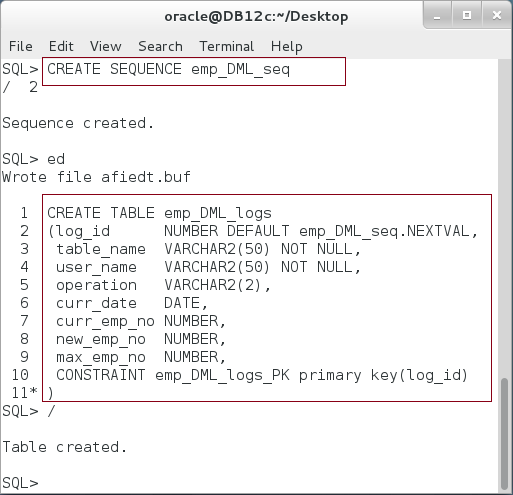
**Step 2:** Execute the following query:

|  |  |
| --- | --- |
| Command | Description |
| COLUMN column\_name FORMAT A30; |  |
| COLUMN data\_type FORMAT A30; |  |
| **SELECT** column\_name,data\_type **FROM user\_tab\_cols** | View the columns of "EMPS" table. |
| WHERE table\_name=**'EMPS'** |
| / |  |



**Step 3:** We are going to create a log table to track changes on EMPS table. Execute the following DDL statements:

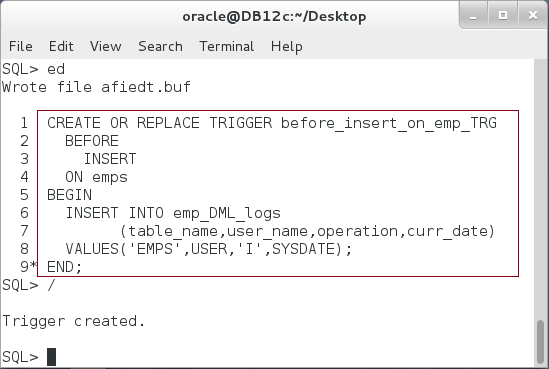
|  |  |
| --- | --- |
| Command | Description |
| CREATE SEQUENCE emp\_DML\_seq | Create sequence |
| / |
| CREATE TABLE emp\_DML\_logs | Create log table |
| (log\_id NUMBER DEFAULT emp\_DML\_seq.NEXTVAL, |
| table\_name VARCHAR2(50) NOT NULL, |
| user\_name VARCHAR2(50) NOT NULL, |
| operation VARCHAR2(2), |
| curr\_date DATE, |
| curr\_emp\_no NUMBER, |
| new\_emp\_no NUMBER, |
| max\_emp\_no NUMBER, |
| CONSTRAINT emp\_DML\_logs\_PK primary key(log\_id) |
| ) |  |
| / |  |

****

**Please note:** the primary key is automatically generated by the sequence: emp\_DML\_seq. Each time a new record is inserted into the table, the sequence is incremented by one.

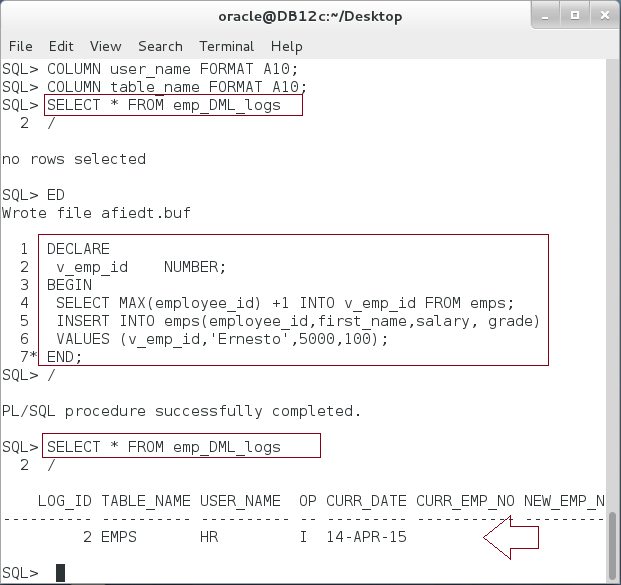
**Step 4:** Create the first statement-level trigger on EMPS table as shown below:

|  |  |
| --- | --- |
| Command | Description |
| CREATE OR REPLACE TRIGGER before\_insert\_on\_emp\_TRG |  |
| **BEFORE** | The trigger should run before running INSERT statement on the table |
| **INSERT** |
| ON emps |
| BEGIN |  |
| **INSERT INTO emp\_DML\_logs** | Save log data into "emp\_DML\_logs" table. |
| (table\_name,user\_name,operation,curr\_date) |
| VALUES('EMPS',USER,'I',SYSDATE); |
| END; |
| / |  |



**Step 5:** Test the previous trigger by inserting new record into "EMPS" table and viewing the content of "emp\_DML\_logs" table as shown below:

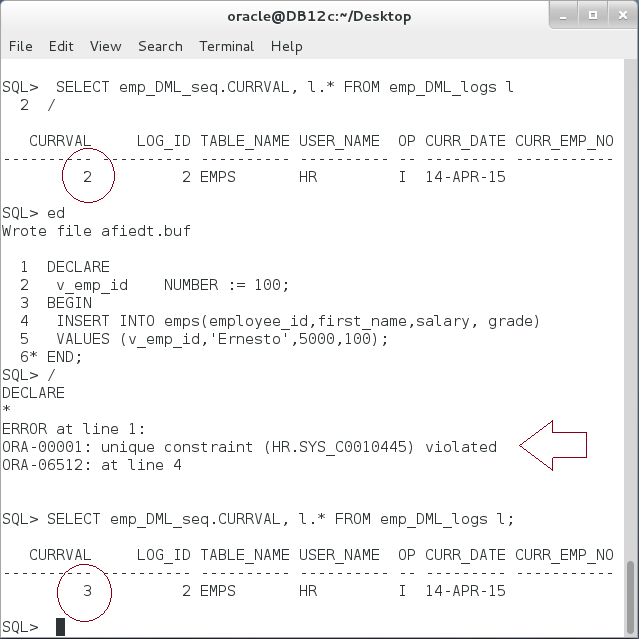
|  |  |
| --- | --- |
| Command | Description |
| COLUMN user\_name FORMAT A10; |  |
| COLUMN table\_name FORMAT A10; |  |
| SELECT \* FROM emp\_DML\_logs | View log table first |
| / |
| DECLARE |  |
| v\_emp\_id NUMBER; |  |
| BEGIN |  |
| SELECT MAX(employee\_id) +1 INTO v\_emp\_id FROM emps; | Insert new record into EMPS table. |
| INSERT INTO emps(employee\_id,first\_name,salary, grade) |
| VALUES (v\_emp\_id,'Ernesto',5000,100); |
| END; |
| / |  |
| SELECT \* FROM emp\_DML\_logs | View the effect on log table. |
| / |

****

**Please note:** the trigger runs before running insert statement on EMPS table and insert a record into "emp\_DML\_logs" table.

**Step 6:** Now try to insert invalid values into EMPS table and note the changes in the sequence value. Execute the following block:

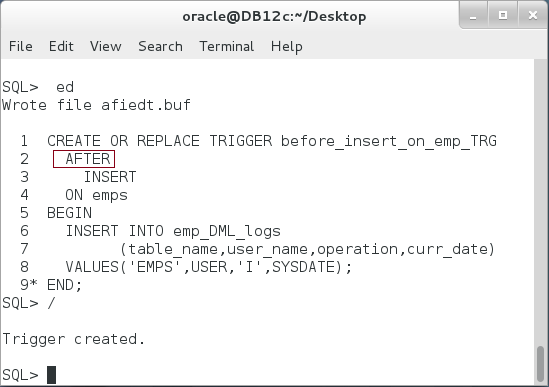
|  |  |
| --- | --- |
| Command | Description |
| SELECT **emp\_DML\_seq.CURRVAL**, l.\* FROM emp\_DML\_logs l |  |
| / |  |
| DECLARE |  |
| v\_emp\_id NUMBER := 100; |  |
| BEGIN |  |
| INSERT INTO emps(employee\_id,first\_name,salary, grade) |  |
| VALUES (v\_emp\_id,'Ernesto',5000,100); |  |
| END; |  |
| / |  |
| SELECT **emp\_DML\_seq.CURRVAL**, l.\* FROM emp\_DML\_logs l |  |
| / |  |

****

**Please note:** although the statement failed, the sequence was increased. **WHY?**

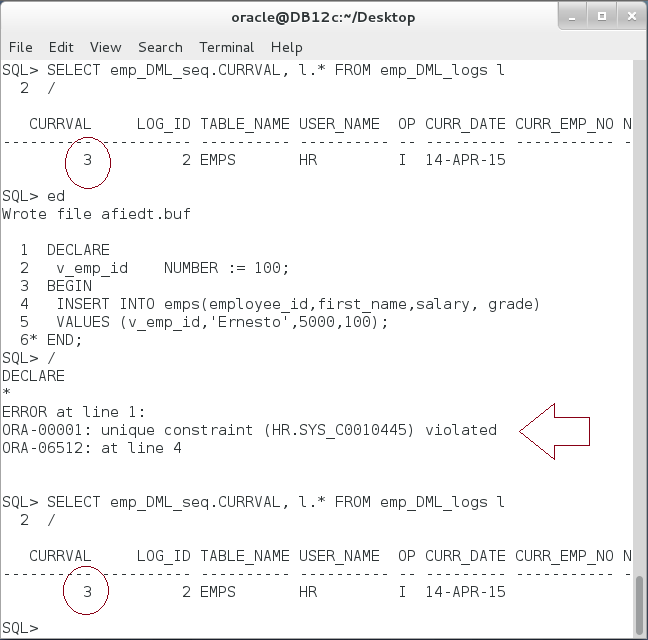
**Step 7:** Re-create the previous trigger so it fires **AFTER** the statement instead of BEFORE as shown below:

|  |
| --- |
| Command |
| CREATE OR REPLACE TRIGGER before\_insert\_on\_emp\_TRG |
| **AFTER** |
| INSERT |
| ON emps |
| BEGIN |
| INSERT INTO emp\_DML\_logs |
| (table\_name,user\_name,operation,curr\_date) |
| VALUES('EMPS',USER,'I',SYSDATE); |
| END; |
| / |

****

**Step 8:** Re-execute the failed statement test again and notice the different:

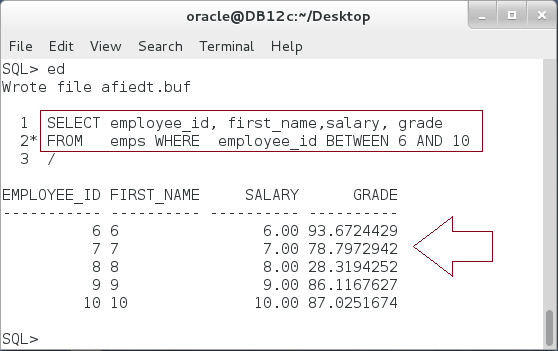
|  |  |
| --- | --- |
| Command | Description |
| SELECT **emp\_DML\_seq.CURRVAL**, l.\* FROM emp\_DML\_logs l |  |
| / |  |
| DECLARE |  |
| v\_emp\_id NUMBER := 100; |  |
| BEGIN |  |
| INSERT INTO emps(employee\_id,first\_name,salary, grade) |  |
| VALUES (v\_emp\_id,'Ernesto',5000,100); |  |
| END; |  |
| / |  |
| SELECT **emp\_DML\_seq.CURRVAL**, l.\* FROM emp\_DML\_logs l |  |
| / |  |

****

**Please note:** this time, the sequence current value remained as it was before the failed statement. **WHY?**

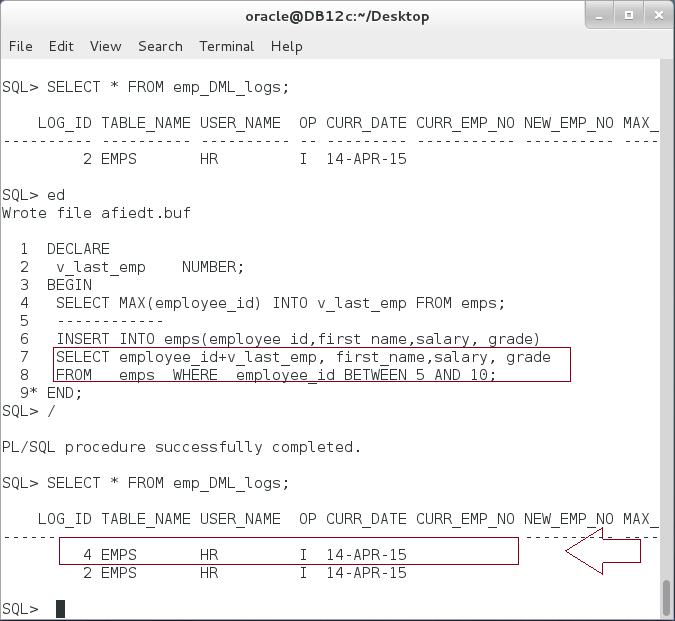
**Step 9:** Execute the following query:

|  |  |
| --- | --- |
| Command | Description |
| SELECT employee\_id, first\_name,salary, grade |  |
| FROM emps WHERE employee\_id BETWEEN 6 AND 10 |  |
| / |  |

****

**Step 10:** Execute the following query:

|  |  |
| --- | --- |
| Command | Description |
| SELECT \* FROM emp\_DML\_logs |  |
| / |  |
| DECLARE |  |
| v\_last\_emp NUMBER; |  |
| BEGIN |  |
| SELECT MAX(employee\_id) INTO v\_last\_emp FROM emps; |  |
| ------------ |  |
| **INSERT** INTO emps(employee\_id,first\_name,salary, grade) | Insert 5 rows at once. |
| **SELECT** employee\_id+v\_last\_emp, first\_name,salary, grade |
| **FROM** emps **WHERE** employee\_id **BETWEEN 6 AND 10;** |
| END; |  |
| / |  |
| SELECT \* FROM emp\_DML\_logs |  |
| / |  |

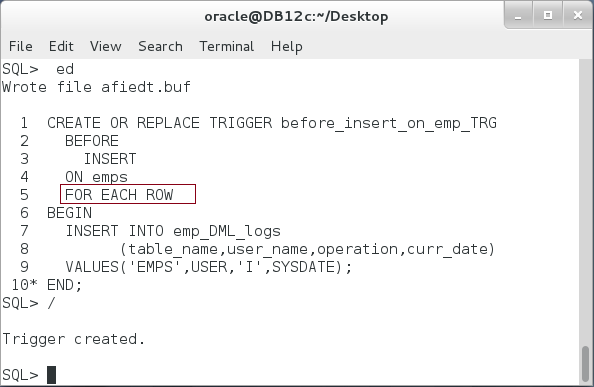
****

**Please note:** Although you insert 5 rows into EMPS table, only one record inserted into "emp\_DML\_logs" table. **WHY?**

## Row-level Trigger

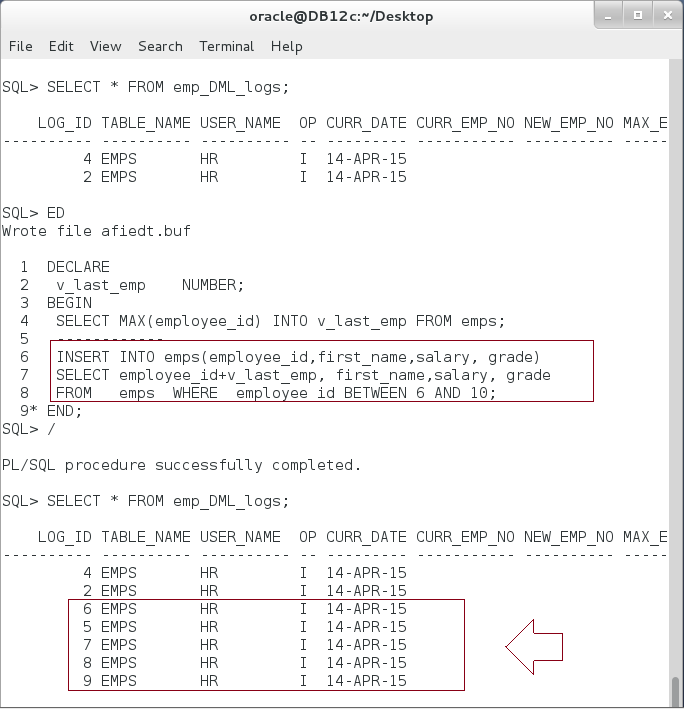
**Step 1:** Re-Create the previous trigger to add "FOR EACH ROW" clause as shown below:

|  |
| --- |
| Command |
| CREATE OR REPLACE TRIGGER before\_insert\_on\_emp\_TRG |
| BEFORE |
| INSERT |
| ON emps |
| **FOR EACH ROW** |
| BEGIN |
| INSERT INTO emp\_DML\_logs |
| (table\_name,user\_name,operation,curr\_date) |
| VALUES('EMPS',USER,'I',SYSDATE); |
| END; |
| / |

****

**Step 2:** Re-execute the test block to insert 5 rows into EMPS and notice "emp\_DML\_logs" table as shown below:

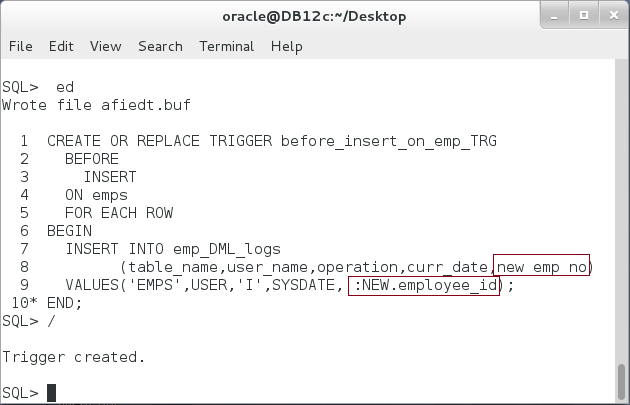
|  |  |
| --- | --- |
| Command | Description |
| SELECT \* FROM emp\_DML\_logs |  |
| / |  |
| DECLARE |  |
| v\_last\_emp NUMBER; |  |
| BEGIN |  |
| SELECT MAX(employee\_id) INTO v\_last\_emp FROM emps; |  |
| ------------ |  |
| **INSERT** INTO emps(employee\_id,first\_name,salary, grade) | Insert 5 rows at once. |
| **SELECT** employee\_id+v\_last\_emp, first\_name,salary, grade |
| **FROM** emps **WHERE** employee\_id **BETWEEN 6 AND 10;** |
| END; |  |
| / |  |
| SELECT \* FROM emp\_DML\_logs |  |
| / |  |



**Please note:** this time, there are 5 new rows added to the log table. **WHY?**

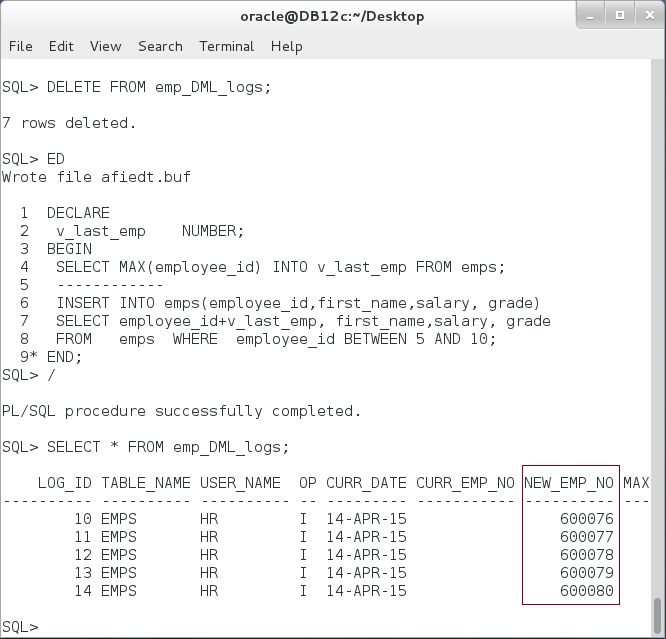
**Step 3:** Re-create the previous trigger to access newly instead row values. Execute the following:

|  |  |
| --- | --- |
| Command | Description |
| CREATE OR REPLACE TRIGGER before\_insert\_on\_emp\_TRG |  |
| BEFORE |  |
| INSERT |  |
| ON emps |  |
| **FOR EACH ROW** |  |
| BEGIN |
| INSERT INTO emp\_DML\_logs |
| (table\_name,user\_name,operation,curr\_date,new\_emp\_no) |  |
| VALUES('EMPS',USER,'I',SYSDATE, **:NEW.employee\_id**); |  |
| END; |  |
| / |  |

****

**Step 4:** Test the previous trigger using the following block:

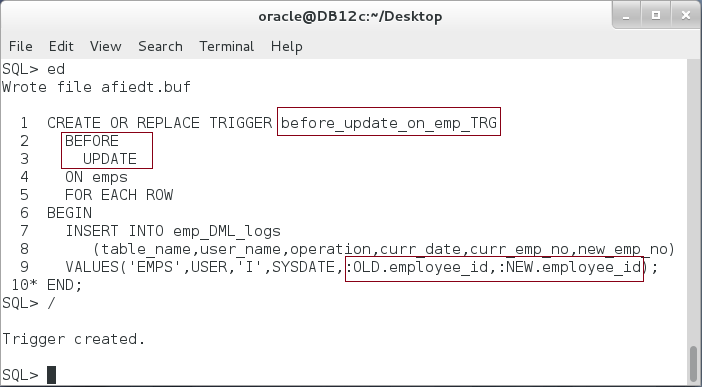
|  |  |
| --- | --- |
| Command | Description |
| **DELETE FROM** emp\_DML\_logs | Delete all things from log table. |
| / |
| DECLARE |  |
| v\_last\_emp NUMBER; |  |
| BEGIN | Insert 5 rows into "EMPS" table. |
| SELECT MAX(employee\_id) INTO v\_last\_emp FROM emps; |
| ------------ |
| INSERT INTO emps(employee\_id,first\_name,salary, grade) |
| SELECT employee\_id+v\_last\_emp, first\_name,salary, grade |
| FROM emps WHERE employee\_id BETWEEN 5 AND 10; |
| END; |  |
| / |  |
| SELECT \* FROM emp\_DML\_logs | View the log table. |
| / |

****

**Please note:** now you can store the new values of EMPS row into "emp\_DML\_logs".

**Step 5:** Create new trigger "**before\_update\_on\_emp\_TRG**" similar to the previous trigger "**before\_insert\_on\_emp\_TRG** " but to track update on "EMPS" as shown below:

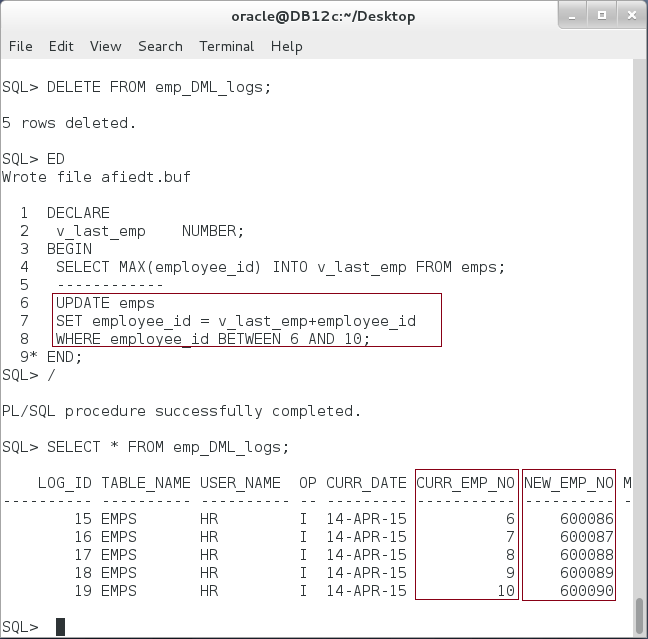
|  |
| --- |
| Line |
| CREATE OR REPLACE TRIGGER **before\_update\_on\_emp\_TRG** |
| **BEFORE** |
| **UPDATE** |
| ON emps |
| FOR EACH ROW |
| BEGIN |
| INSERT INTO emp\_DML\_logs |
| (table\_name,user\_name,operation,curr\_date,**curr\_emp\_no,new\_emp\_no**) |
| VALUES('EMPS',USER,'I',SYSDATE**,:OLD.employee\_id,:NEW.employee\_id**); |
| END; |
| / |



**Please note:** FOR EACH ROW trigger that tracks UPDATE event has two row values, OLD and NEW. OLD represents the value before the UPDATE event and NEW is the value of row after the UPDATE event.

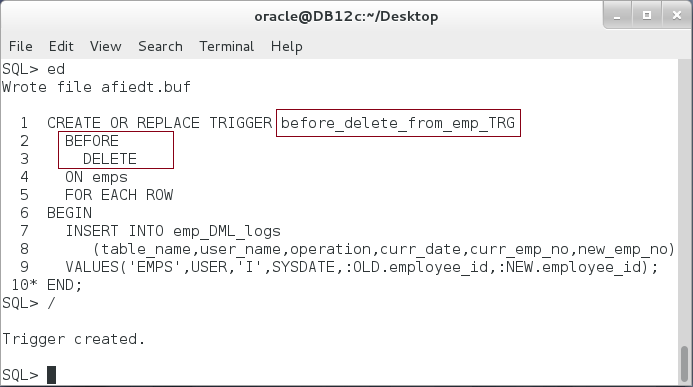
**Step 6:** Test the previous trigger using the following block:

|  |  |
| --- | --- |
| Line |  |
| DELETE FROM emp\_DML\_logs |  |
| / |  |
| DECLARE |  |
| v\_last\_emp NUMBER; |  |
| BEGIN |  |
| SELECT MAX(employee\_id) INTO v\_last\_emp FROM emps; |  |
| ------------ |  |
| **UPDATE** emps | Update statement that affects 5 rows |
| SET employee\_id = v\_last\_emp+employee\_id |
| WHERE employee\_id BETWEEN 6 AND 10; |
| END; |
| / |  |
| SELECT \* FROM emp\_DML\_logs |  |
| / |  |



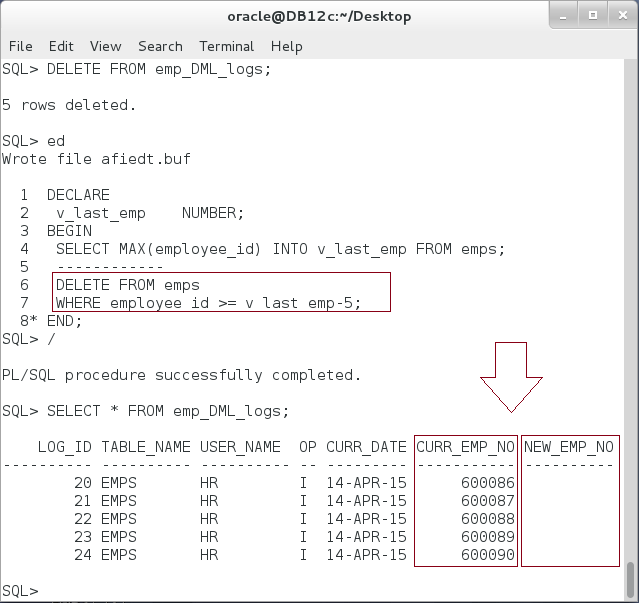
**Step 7:** Finally, create a trigger to track DELETE event on EMPS table as shown below:

|  |
| --- |
| Line |
| CREATE OR REPLACE TRIGGER **before\_delete\_from\_emp\_TRG** |
| **BEFORE** |
| **DELETE** |
| ON emps |
| FOR EACH ROW |
| BEGIN |
| INSERT INTO emp\_DML\_logs |
| (table\_name,user\_name,operation,curr\_date,curr\_emp\_no,new\_emp\_no) |
| VALUES('EMPS',USER,'I',SYSDATE,:OLD.employee\_id,:NEW.employee\_id); |
| END; |
| / |



**Step 8:** Test the last trigger by using the following block:

|  |
| --- |
| Line |
| DELETE FROM emp\_DML\_logs |
| / |
| DECLARE |
| v\_last\_emp NUMBER; |
| BEGIN |
| SELECT MAX(employee\_id) INTO v\_last\_emp FROM emps; |
| ------------ |
| **DELETE** **FROM** emps |
| **WHERE** employee\_id >= v\_last\_emp-5; |
| END; |
| / |
| SELECT \* FROM emp\_DML\_logs |
| / |

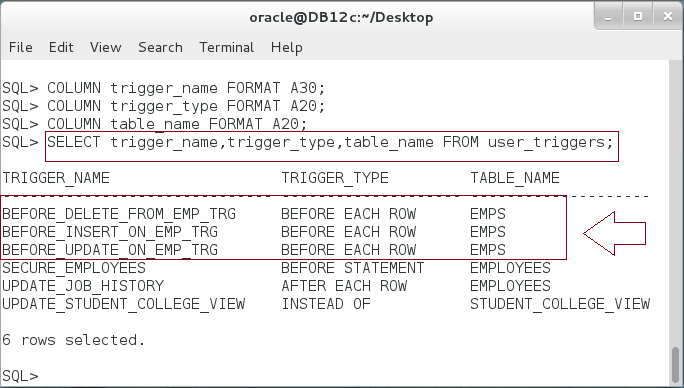


**Please note:** Although NEW and OLD rows are available for INSERT, UPDATE, and DELETE triggers; only UPDATE trigger has values for both OLD and NEW. DELETE trigger has values in OLD row only while INSERT trigger has values in NEW row only.

## Trigger with Conditions

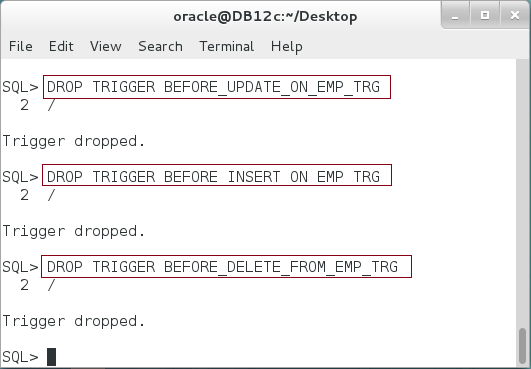
**Step 1:** So far, you have three triggers that actually intend to do the same job, monitoring transactions done on EMPS table. To view your triggers, execute the following query:

|  |  |
| --- | --- |
| Line | Description |
| COLUMN trigger\_name FORMAT A30; |  |
| COLUMN trigger\_type FORMAT A20; |  |
| COLUMN table\_name FORMAT A20; |  |
| **SELECT** trigger\_name,trigger\_type,table\_name **FROM user\_triggers** |  |
| / |  |



**Step 2:** To view your own triggers and their statuses, use the following command:

|  |  |
| --- | --- |
| Line | Description |
| DROP TRIGGER BEFORE\_UPDATE\_ON\_EMP\_TRG |  |
| / |  |
| DROP TRIGGER BEFORE\_INSERT\_ON\_EMP\_TRG |  |
| / |  |
| DROP TRIGGER BEFORE\_DELETE\_FROM\_EMP\_TRG |  |
| / |  |

****

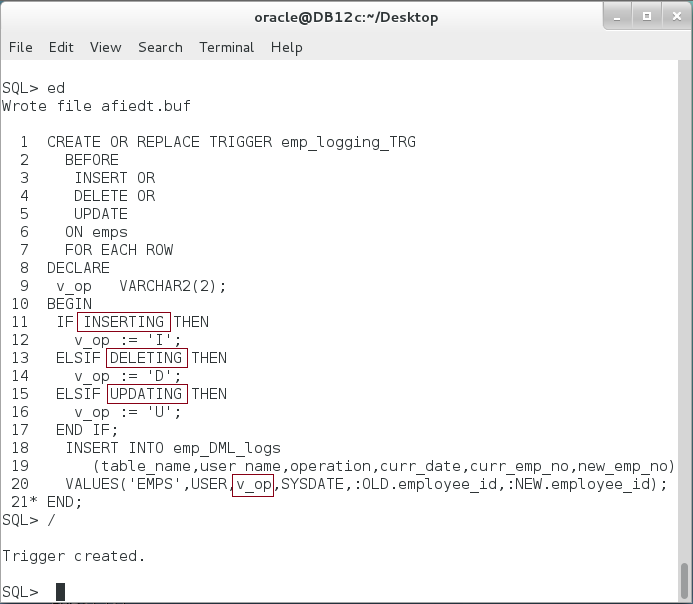
**Step 3:** Instead of the previous three triggers, you can create one trigger that can fire for update, insert, or delete on EMPS table as shown below:

|  |
| --- |
| Line |
| CREATE OR REPLACE TRIGGER emp\_logging\_TRG |
| **BEFORE** |
| **INSERT** OR |
| **DELETE** OR |
| **UPDATE** |
| ON emps |
| FOR EACH ROW |
| BEGIN |
| INSERT INTO emp\_DML\_logs |
| (table\_name,user\_name,operation,curr\_date,curr\_emp\_no,new\_emp\_no) |
| VALUES('EMPS',USER,'I',SYSDATE,:OLD.employee\_id,:NEW.employee\_id); |
| END; |
| / |

**Please note:** the trigger does the same as the previous three triggers did. **Check it by** **yourself**.

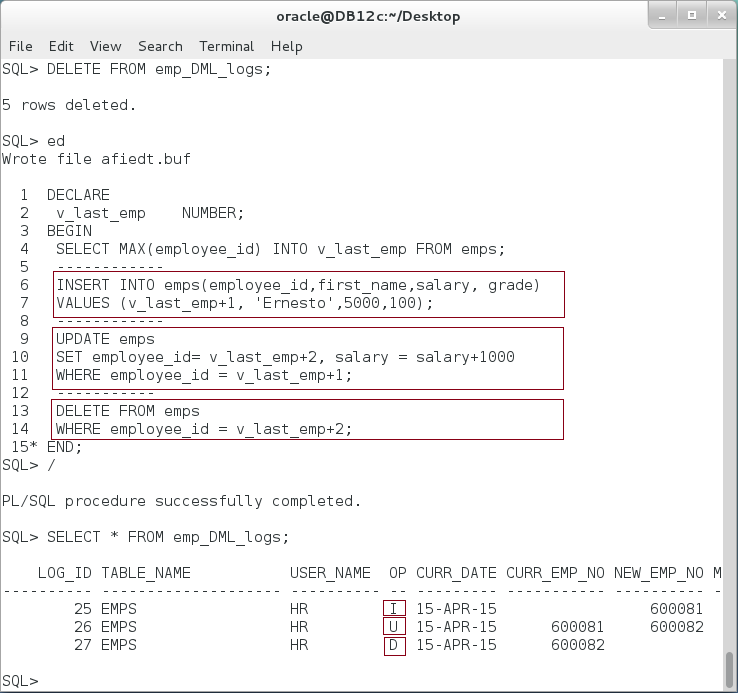
**Step 4:** The previous trigger has one little issue, if you notice. It didn't store the real operation letter; all operations would be stored as "I". You can know what exactly the event fires the trigger. Re-create the trigger as shown below:

|  |
| --- |
| Line |
| CREATE OR REPLACE TRIGGER emp\_logging\_TRG |
| BEFORE |
| INSERT OR |
| DELETE OR |
| UPDATE |
| ON emps |
| FOR EACH ROW |
| DECLARE |
| v\_op VARCHAR2(2); |
| BEGIN |
| IF **INSERTING** THEN |
| v\_op := 'I'; |
| ELSIF **DELETING** THEN |
| v\_op := 'D'; |
| ELSIF **UPDATING** THEN |
| v\_op := 'U'; |
| END IF; |
| INSERT INTO emp\_DML\_logs |
| (table\_name,user\_name,operation,curr\_date,curr\_emp\_no,new\_emp\_no) |
| VALUES('EMPS',USER,v\_op,SYSDATE,:OLD.employee\_id,:NEW.employee\_id); |
| END; |
| / |



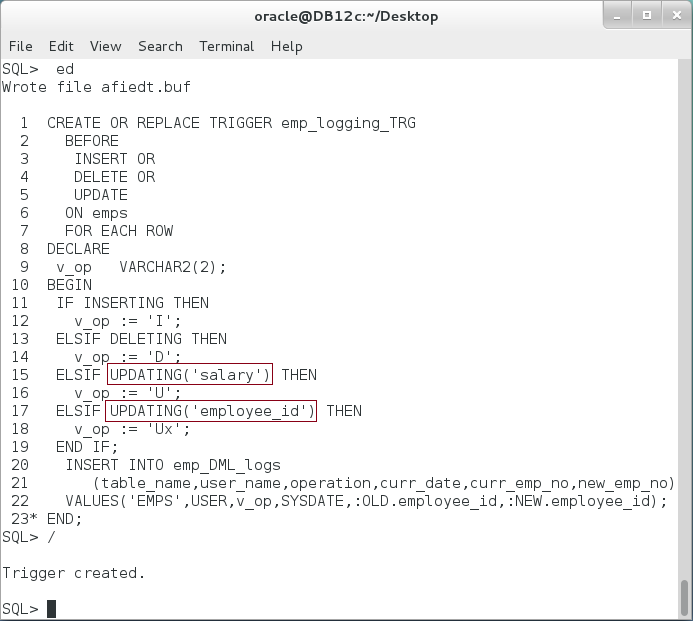
**Step 5:** Re-create the trigger as shown below:

|  |  |
| --- | --- |
| Line | Description |
| DELETE FROM emp\_DML\_logs | Remove all previous records from the log table. |
| / |
| DECLARE |
| v\_last\_emp NUMBER; |
| BEGIN |  |
| SELECT MAX(employee\_id) INTO v\_last\_emp FROM emps; |  |
| ------------ | Test INSERT on EMPS table |
| **INSERT** INTO emps(employee\_id,first\_name,salary, grade) |
| VALUES (v\_last\_emp+1, 'Ernesto',5000,100); |
| ------------ |  |
| **UPDATE** emps | Test UPDATE on EMPS table. |
| SET employee\_id= v\_last\_emp+2, salary = salary+1000 |
| WHERE employee\_id = v\_last\_emp+1; |
| ----------- |  |
| **DELETE** FROM emps | Test DELETE on EMPS table. |
| WHERE employee\_id = v\_last\_emp+2; |
| END; |
| / |  |
| SELECT \* FROM emp\_DML\_logs | Query the log table. |
| / |



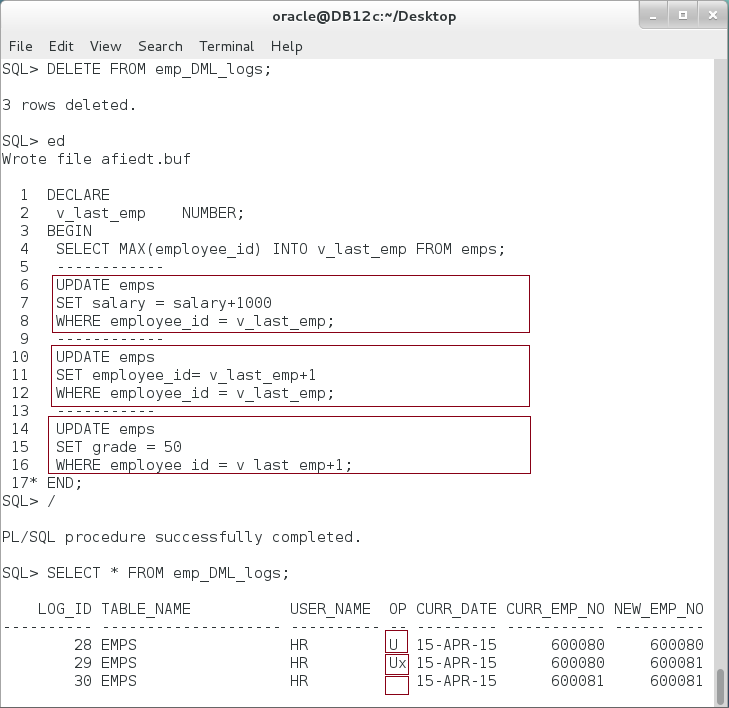
**Step 6:** Suppose you want to distinguish between when the use updates "**salary**" column and when he updates "**employee\_id**". Re-create a trigger as shown below :

|  |
| --- |
| Line |
| CREATE OR REPLACE TRIGGER emp\_logging\_TRG |
| BEFORE |
| INSERT OR |
| DELETE OR |
| UPDATE |
| ON emps |
| FOR EACH ROW |
| DECLARE |
| v\_op VARCHAR2(2); |
| BEGIN |
| IF INSERTING THEN |
| v\_op := 'I'; |
| ELSIF DELETING THEN |
| v\_op := 'D'; |
| ELSIF **UPDATING('salary')** THEN |
| v\_op := 'U'; |
| ELSIF **UPDATING('employee\_id')** THEN |
| v\_op := 'Ux'; |
| END IF; |
| INSERT INTO emp\_DML\_logs |
| (table\_name,user\_name,operation,curr\_date,curr\_emp\_no,new\_emp\_no) |
| VALUES('EMPS',USER,v\_op,SYSDATE,:OLD.employee\_id,:NEW.employee\_id); |
| END; |
| / |



**Step 7:** Test the previous trigger as shown below:

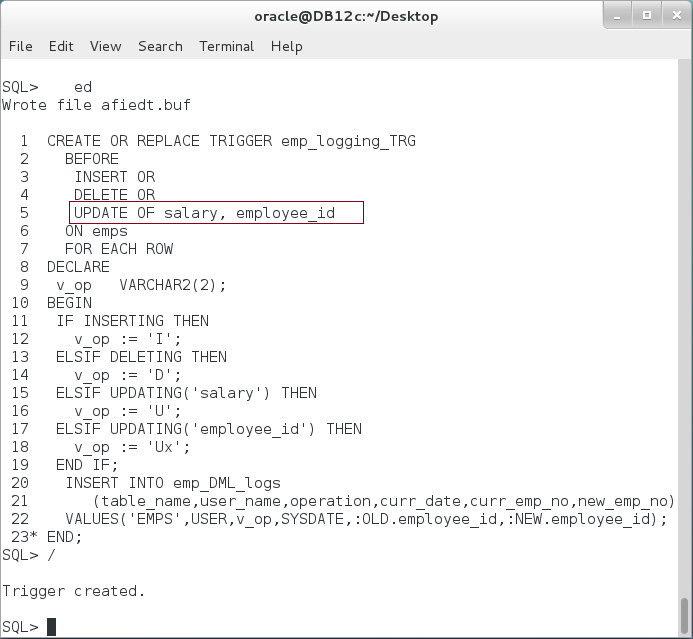
|  |  |
| --- | --- |
| Line | Description |
| DELETE FROM emp\_DML\_logs |  |
| / |  |
| DECLARE |  |
| v\_last\_emp NUMBER; |  |
| BEGIN |  |
| SELECT MAX(employee\_id) INTO v\_last\_emp FROM emps; |  |
| ------------ |  |
| **UPDATE** emps | Test UPDATE on "salary" column. |
| SET **salary** = salary+1000 |
| WHERE employee\_id = v\_last\_emp; |
| ------------ |  |
| **UPDATE** emps | Test UPDATE on "employee\_id" column |
| SET **employee\_id**= v\_last\_emp+1 |
| WHERE employee\_id = v\_last\_emp; |
| ----------- |  |
| **UPDATE** emps | Test UPDATE on "grade" column. |
| SET **grade** = 50 |
| WHERE employee\_id = v\_last\_emp+1; |
| END; |
| / |  |
| SELECT \* FROM emp\_DML\_logs |  |
| / |  |



**Please note:** The trigger is still catching UPDATE operations done on columns other than "salary" and "employee\_id". It records them with and empty (NULL) operation. Other than this issue, the trigger does not take in consideration UPDATE operation on both "salary" and "employee\_id" as once. **What would happen if you issue an UPDATE on both "salary" and "employee\_id" columns on the same query?**

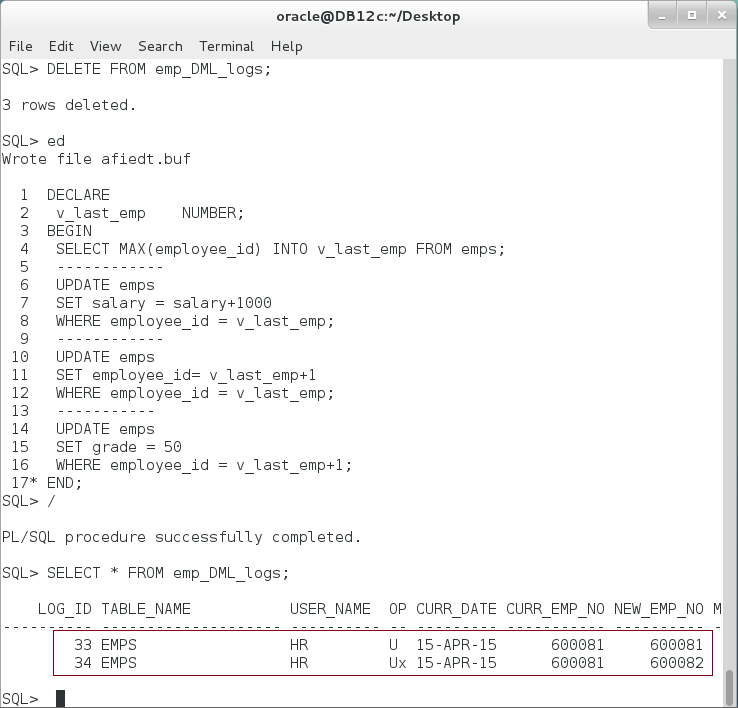
**Step 8:** Suppose now you don't want to track UPDATE operations other than "salary" and "employee\_id". In this case, you can instruct the trigger not to fire unless the change is on "salary" or "emplyee\_id" as shown below:

|  |
| --- |
| Line |
| CREATE OR REPLACE TRIGGER emp\_logging\_TRG |
| BEFORE |
| INSERT OR |
| DELETE OR |
| **UPDATE OF salary, employee\_id** |
| ON emps |
| FOR EACH ROW |
| DECLARE |
| v\_op VARCHAR2(2); |
| BEGIN |
| IF INSERTING THEN |
| v\_op := 'I'; |
| ELSIF DELETING THEN |
| v\_op := 'D'; |
| ELSIF UPDATING('salary') THEN |
| v\_op := 'U'; |
| ELSIF UPDATING('employee\_id') THEN |
| v\_op := 'Ux'; |
| END IF; |
| INSERT INTO emp\_DML\_logs |
| (table\_name,user\_name,operation,curr\_date,curr\_emp\_no,new\_emp\_no) |
| VALUES('EMPS',USER,v\_op,SYSDATE,:OLD.employee\_id,:NEW.employee\_id); |
| END; |
| / |



**Step 9:** Test the previous trigger as shown:

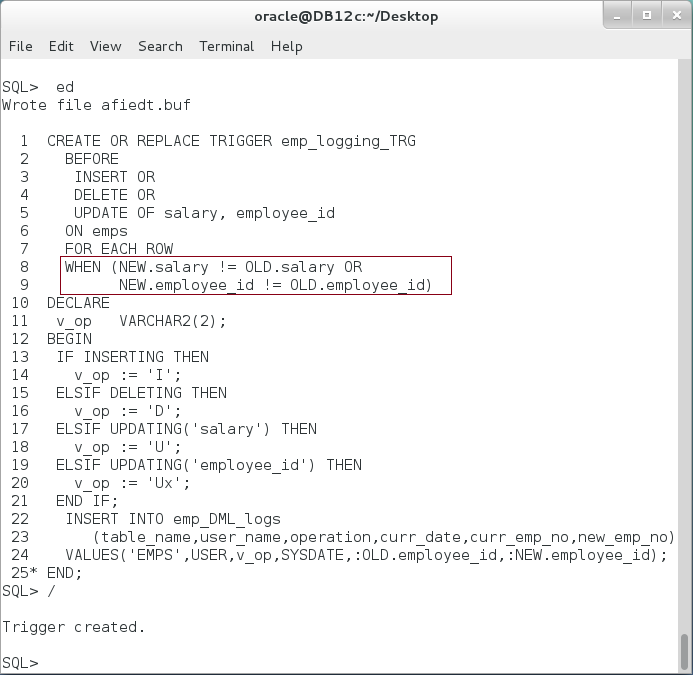
|  |  |
| --- | --- |
| Line | Description |
| DELETE FROM emp\_DML\_logs |  |
| / |  |
| DECLARE |  |
| v\_last\_emp NUMBER; |  |
| BEGIN |  |
| SELECT MAX(employee\_id) INTO v\_last\_emp FROM emps; |  |
| ------------ |  |
| **UPDATE** emps | Test UPDATE on "salary" column. |
| SET **salary** = salary+1000 |
| WHERE employee\_id = v\_last\_emp; |
| ------------ |  |
| **UPDATE** emps | Test UPDATE on "employee\_id" column |
| SET **employee\_id**= v\_last\_emp+1 |
| WHERE employee\_id = v\_last\_emp; |
| ----------- |  |
| **UPDATE** emps | Test UPDATE on "grade" column. |
| SET **grade** = 50 |
| WHERE employee\_id = v\_last\_emp+1; |
| END; |
| / |  |
| SELECT \* FROM emp\_DML\_logs |  |
| / |  |



**Please note:** the trigger is now working as your condition but it seems one little thing has been left behind: **what would happen if someone update "salary" value to its previous value (no real change is made)?**

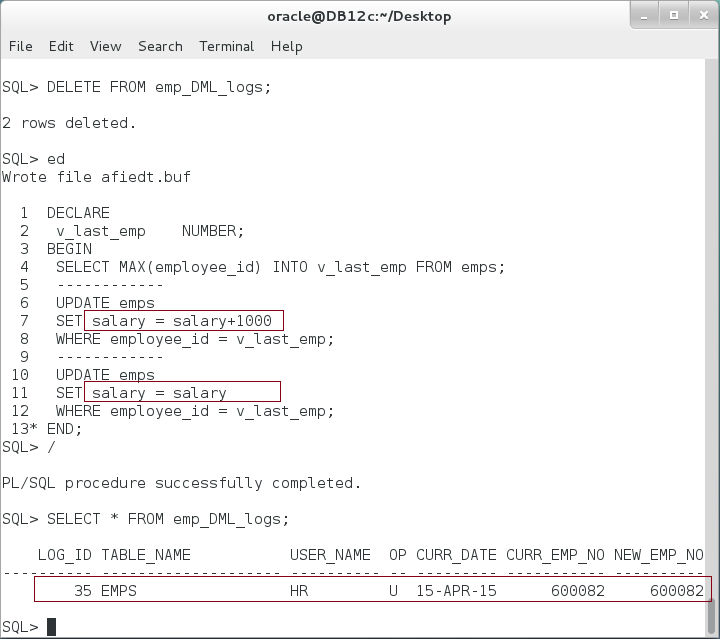
**Step 10:** To avoid saving unnecessary log data and fires the trigger for no reason, re-create the trigger as shown below:

|  |
| --- |
| Line |
| CREATE OR REPLACE TRIGGER emp\_logging\_TRG |
| BEFORE |
| INSERT OR |
| DELETE OR |
| UPDATE OF salary, employee\_id |
| ON emps |
| FOR EACH ROW |
| **WHEN (NEW.salary != OLD.salary OR** |
| **NEW.employee\_id != OLD.employee\_id)** |
| DECLARE |
| v\_op VARCHAR2(2); |
| BEGIN |
| IF INSERTING THEN |
| v\_op := 'I'; |
| ELSIF DELETING THEN |
| v\_op := 'D'; |
| ELSIF UPDATING('salary') THEN |
| v\_op := 'U'; |
| ELSIF UPDATING('employee\_id') THEN |
| v\_op := 'Ux'; |
| END IF; |
| INSERT INTO emp\_DML\_logs |
| (table\_name,user\_name,operation,curr\_date,curr\_emp\_no,new\_emp\_no) |
| VALUES('EMPS',USER,v\_op,SYSDATE,:OLD.employee\_id,:NEW.employee\_id); |
| END; |
| / |



**Step 11:** Test the previous trigger as shown below:

|  |
| --- |
| Line |
| DELETE FROM emp\_DML\_logs |
| / |
| DECLARE |
| v\_last\_emp NUMBER; |
| BEGIN |
| SELECT MAX(employee\_id) INTO v\_last\_emp FROM emps; |
| ------------ |
| **UPDATE** emps |
| SET **salary = salary+1000** |
| WHERE employee\_id = v\_last\_emp; |
| ------------ |
| **UPDATE** emps |
| SET **salary = salary** |
| WHERE employee\_id = v\_last\_emp; |
| END; |
| / |
| SELECT \* FROM emp\_DML\_logs |
| / |

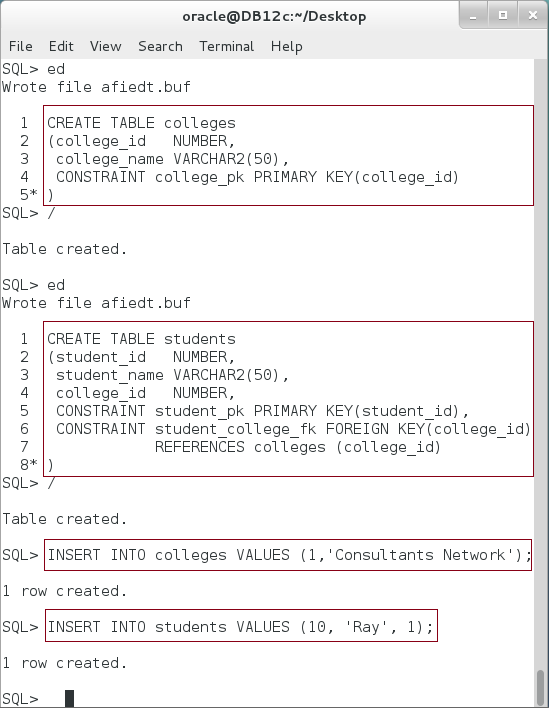
****

**Please note:** be careful when using WHEN condition, the previous trigger will not be executed on INSERT or DELETE. **Why?** **How can you fix it?**

## INSTEAD OF Trigger

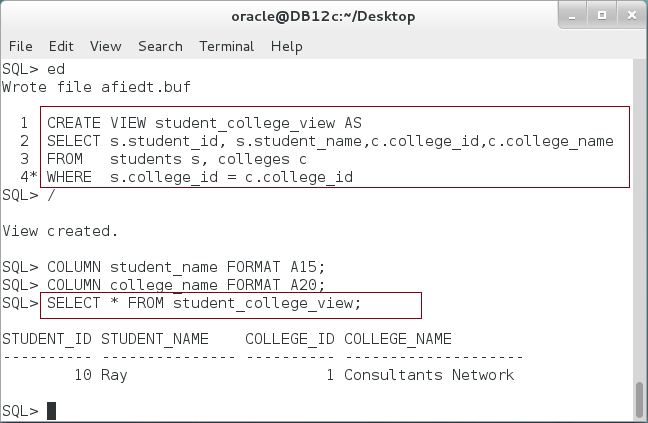
**Step 1:** Create "colleges" and "students" tables and populate them with one row for each as shown below:

|  |  |
| --- | --- |
| Line | Description |
| **CREATE TABLE colleges** | Create "colleges" table |
| (college\_id NUMBER, |
| college\_name VARCHAR2(50), |
| CONSTRAINT college\_pk PRIMARY KEY(college\_id) |
| ) |
| / |  |
|  |  |
| **CREATE TABLE students** | Create "students" table |
| (student\_id NUMBER, |
| student\_name VARCHAR2(50), |
| college\_id NUMBER, |
| CONSTRAINT student\_pk PRIMARY KEY(student\_id), |
| CONSTRAINT student\_college\_fk FOREIGN KEY(college\_id) |
| REFERENCES colleges (college\_id) |
| ) |  |
| / |  |
|  |  |
| **INSERT INTO colleges** VALUES (1,'Consultants Network'); | Insert some rows. |
| **INSERT INTO students** VALUES (10, 'Ray', 1); |



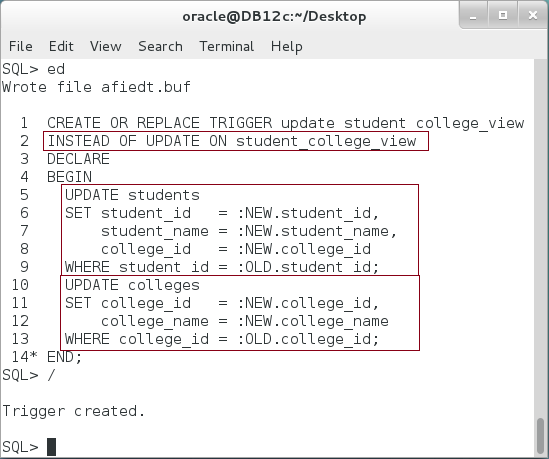
**Step 2:** Create a view "student\_college\_view" based on the previous tables as shown in the following:

|  |  |
| --- | --- |
| Line | Description |
| **CREATE VIEW** student\_college\_view AS | Create view |
| SELECT s.student\_id, s.student\_name,c.college\_id,c.college\_name |
| FROM students s, colleges c |
| WHERE s.college\_id = c.college\_id |
| / |  |
| COLUMN student\_name FORMAT A15; |  |
| COLUMN college\_name FORMAT A20; |  |
|  | Query the view. |
| **SELECT \* FROM** student\_college\_view |
| / |



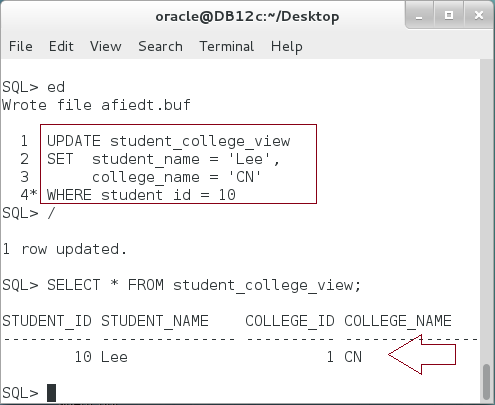
**Step 3:** Suppose you want to allow your users to execute UPDATE command on this view. Oracle allows you to create an INSTEAD OF trigger to achieve this goal. Execute the following command:

|  |  |
| --- | --- |
| Line | Description |
| CREATE OR REPLACE TRIGGER update\_student\_college\_view |  |
| **INSTEAD OF UPDATE** **ON student\_college\_view** |  |
| DECLARE |  |
| BEGIN |  |
| **UPDATE students** | Update students table. |
| SET student\_id = :NEW.student\_id, |
| student\_name = :NEW.student\_name, |
| college\_id = :NEW.college\_id |
| WHERE student\_id = :OLD.student\_id; |
| **UPDATE** **colleges** | Update colleges table. |
| SET college\_id = :NEW.college\_id, |
| college\_name = :NEW.college\_name |
| WHERE college\_id = :OLD.college\_id; |
| END; |  |
| / |  |

****

**Step 4:** Test the previous trigger by updating the view and then query its content:

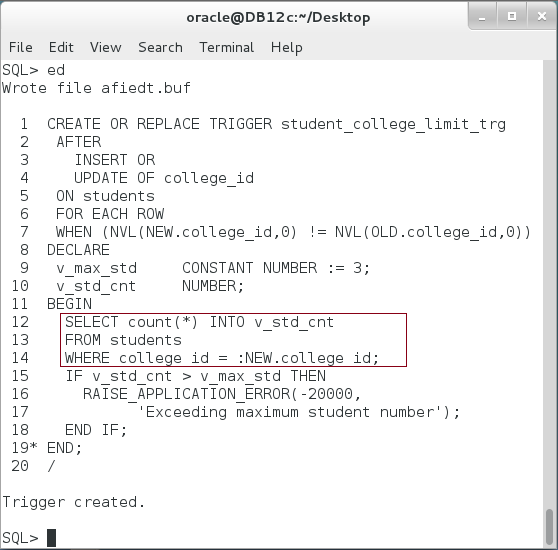
|  |  |
| --- | --- |
| Line | Description |
| UPDATE student\_college\_view | Update "student\_college\_view" view. |
| SET student\_name = 'Lee', |
| college\_name = 'CN' |
| WHERE student\_id = 10 |
| / |  |
| SELECT \* FROM student\_college\_view | Query the view content. |
| / |

****

## Compound DML Trigger

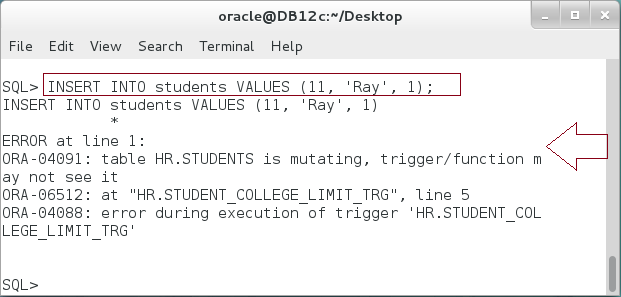
**Step 1:** Based on "colleges" and "students" tables created on the previous section, suppose you want to enforce a business rule so that the number of students in each college does not exceed a specified number, say 3. We will first make it simple by creating the following trigger:

|  |  |
| --- | --- |
| Line | Description |
| **CREATE** OR REPLACE **TRIGGER** student\_college\_limit\_trg |  |
| **AFTER** |  |
| INSERT OR |  |
| UPDATE OF college\_id |  |
| **ON students** |  |
| FOR EACH ROW |  |
| WHEN (NVL(NEW.college\_id,0) != NVL(OLD.college\_id,0)) |  |
| DECLARE |  |
| v\_max\_std CONSTANT NUMBER := 3; |  |
| v\_std\_cnt NUMBER; |  |
| BEGIN |  |
| **SELECT count(\*)** INTO v\_std\_cnt | Get the number of students in the selected college. |
| **FROM students** |
| **WHERE college\_id = :NEW.college\_id;** |
| IF v\_std\_cnt> v\_max\_std THEN | Raise exception if the total number of students is greater than 3 |
| RAISE\_APPLICATION\_ERROR(-20000, |
| 'Exceeding maximum student number'); |
| END IF; |
| END; |  |
| / |  |

****

**Step 2:** To test the trigger, try to insert new rows in "students" table:

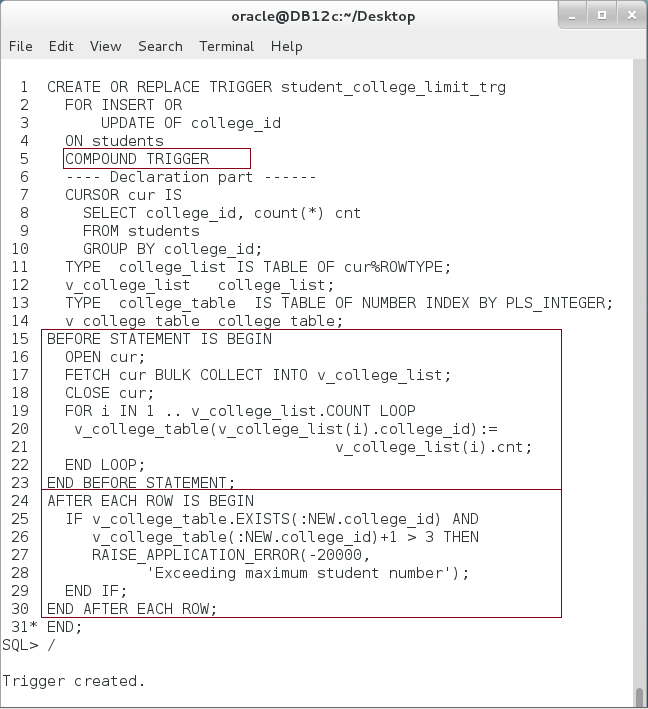
|  |  |
| --- | --- |
| Line | Description |
| INSERT INTO students VALUES (11, 'Ray', 1) |  |
| / |  |

****

**Please note:** the previous insert statement failed. The error shows that the table is mutating. You are not allowed to query the table on which the trigger is created. This condition is applied on AFTER FOR EACH trigger. Oracle prevents such access to avoid getting inconsistence data of the table.

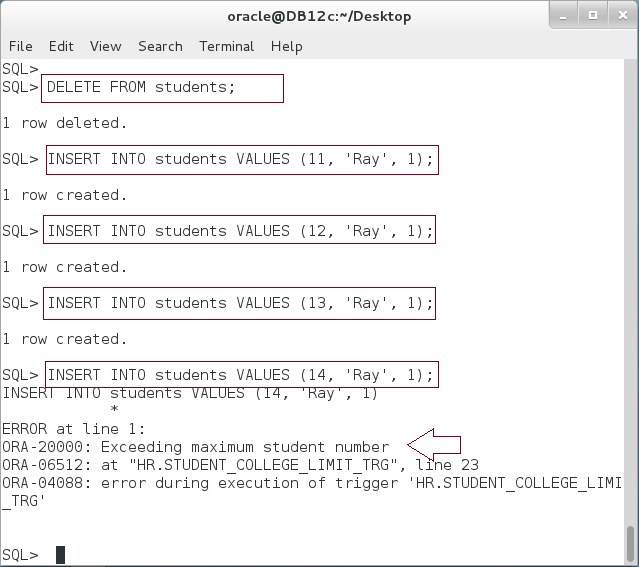
**Step 3:** Starting from Oracle 11g, Oracle presents Compound Trigger to avoid mutating table exception. The idea of Compound Trigger is to run some code in different levels (Statement or Row level), in different timing (Before or After) the event but all of them shared a common variables. To solve the previous issue using a Compound Trigger, execute the following:

|  |  |
| --- | --- |
| Line | Description |
| **CREATE** OR REPLACE **TRIGGER** student\_college\_limit\_trg |  |
| **FOR** INSERT OR |  |
| UPDATE OF college\_id |  |
| ON students |  |
| **COMPOUND TRIGGER** |  |
| ---- Declaration part ------ | Declare a common variable right after "COMPOUND TRIGGER" clause. There is no "DECLARE" keyword allowed here. |
| CURSOR cur IS |
| SELECT college\_id, count(\*) cnt |
| FROM students |
| GROUP BY college\_id; |
| TYPE college\_list IS TABLE OF cur%ROWTYPE; |
| v\_college\_list college\_list; |
| TYPE college\_table IS TABLE OF NUMBER INDEX BY PLS\_INTEGER; |
| v\_college\_table college\_table; |  |
| **BEFORE STATEMENT IS BEGIN** | Use "BEFORE STATEMENT" section to query the table and put the result in a common variable. |
| OPEN cur; |
| FETCH cur BULK COLLECT INTO v\_college\_list; |
| CLOSE cur; |
| FOR i IN 1 .. v\_college\_list.COUNT LOOP |
| v\_college\_table(v\_college\_list(i).college\_id) := v\_college\_list(i).cnt; |
| END LOOP; |
| **END BEFORE STATEMENT;** |  |
| **AFTER EACH ROW IS BEGIN** | Use "AFTER EACH ROW" section to check your business rule. |
| IF v\_college\_table.EXISTS(:NEW.college\_id) AND |
| v\_college\_table(:NEW.college\_id)+1 > 3 THEN |
| RAISE\_APPLICATION\_ERROR(-20000, |
| 'Exceeding maximum student number'); |
| END IF; |
| **END AFTER EACH ROW;** |
| END; |  |
| / |  |

****

**Step 4:** Test the previous trigger as shown below:

|  |  |
| --- | --- |
| Line | Description |
| DELETE FROM students |  |
| / |  |
| INSERT INTO students VALUES (11, 'Ray', 1) |  |
| / |  |
| INSERT INTO students VALUES (12, 'Ray', 1) |  |
| / |  |
| INSERT INTO students VALUES (13, 'Ray', 1) |  |
| / |  |
| INSERT INTO students VALUES (14, 'Ray', 1) |  |
| / |  |



# SUMMARY

DML Triggers involve many features. You are able to fire DML trigger in statement level (executing once after or before DML statement), or in row-level (executing after or before DML for each row). Moreover, the trigger may involve more conditions to narrow down the cases in which the trigger will be fired. Additionally, the developer may use conditional predicates (UPDATING, INSERTING, or DELETING) to know what exactly the event firing the trigger. INSTEAD OF trigger is the only option available to execute trigger for non-editioning view or a nested. Compound DML trigger is used to void failing in mutating-table exception, ORA-04091.

After completing this lab exercise, you should be able to use DML Trigger in different situations including Statement-level, Row-level, or Compound.

# REFERENCES

* https://docs.oracle.com/database/121/LNPLS/triggers.htm#LNPLS20041

# INDEX

Compound DML trigger 27

DML trigger 2, 27

INSTEAD OF trigger 2, 27