**PL/SQL - Triggers**

In this chapter, we will discuss Triggers in PL/SQL. Triggers are stored programs, which are automatically executed or fired when some events occur. Triggers are, in fact, written to be executed in response to any of the following events −

* A **database manipulation**DMLDML statement DELETE,INSERT,orUPDATEDELETE,INSERT,orUPDATE
* A **database definition**DDLDDL statement CREATE,ALTER,orDROPCREATE,ALTER,orDROP.
* A **database operation** SERVERERROR,LOGON,LOGOFF,STARTUP,orSHUTDOWNSERVERERROR,LOGON,LOGOFF,STARTUP,orSHUTDOWN.

Triggers can be defined on the table, view, schema, or database with which the event is associated.

**Benefits of Triggers**

Triggers can be written for the following purposes −

* Generating some derived column values automatically
* Enforcing referential integrity
* Event logging and storing information on table access
* Auditing
* Synchronous replication of tables
* Imposing security authorizations
* Preventing invalid transactions

**Creating Triggers**

The syntax for creating a trigger is −

CREATE [OR REPLACE ] TRIGGER trigger\_name

{BEFORE | AFTER | INSTEAD OF }

{INSERT [OR] | UPDATE [OR] | DELETE}

[OF col\_name]

ON table\_name

[REFERENCING OLD AS o NEW AS n]

[FOR EACH ROW]

WHEN (condition)

DECLARE

Declaration-statements

BEGIN

Executable-statements

EXCEPTION

Exception-handling-statements

END;

Where,

* CREATE [OR REPLACE] TRIGGER trigger\_name − Creates or replaces an existing trigger with the *trigger\_name*.
* {BEFORE | AFTER | INSTEAD OF} − This specifies when the trigger will be executed. The INSTEAD OF clause is used for creating trigger on a view.
* {INSERT [OR] | UPDATE [OR] | DELETE} − This specifies the DML operation.
* [OF col\_name] − This specifies the column name that will be updated.
* [ON table\_name] − This specifies the name of the table associated with the trigger.
* [REFERENCING OLD AS o NEW AS n] − This allows you to refer new and old values for various DML statements, such as INSERT, UPDATE, and DELETE.
* [FOR EACH ROW] − This specifies a row-level trigger, i.e., the trigger will be executed for each row being affected. Otherwise the trigger will execute just once when the SQL statement is executed, which is called a table level trigger.
* WHEN condition − This provides a condition for rows for which the trigger would fire. This clause is valid only for row-level triggers.

**Example**

To start with, we will be using the CUSTOMERS table we had created and used in the previous chapters −

Select \* from customers;

+----+----------+-----+-----------+----------+

| ID | NAME | AGE | ADDRESS | SALARY |

+----+----------+-----+-----------+----------+

| 1 | Ramesh | 32 | Ahmedabad | 2000.00 |

| 2 | Khilan | 25 | Delhi | 1500.00 |

| 3 | kaushik | 23 | Kota | 2000.00 |

| 4 | Chaitali | 25 | Mumbai | 6500.00 |

| 5 | Hardik | 27 | Bhopal | 8500.00 |

| 6 | Komal | 22 | MP | 4500.00 |

+----+----------+-----+-----------+----------+

The following program creates a **row-level** trigger for the customers table that would fire for INSERT or UPDATE or DELETE operations performed on the CUSTOMERS table. This trigger will display the salary difference between the old values and new values −

CREATE OR REPLACE TRIGGER display\_salary\_changes

BEFORE DELETE OR INSERT OR UPDATE ON customers

FOR EACH ROW

WHEN (NEW.ID > 0)

DECLARE

sal\_diff number;

BEGIN

sal\_diff := :NEW.salary - :OLD.salary;

dbms\_output.put\_line('Old salary: ' || :OLD.salary);

dbms\_output.put\_line('New salary: ' || :NEW.salary);

dbms\_output.put\_line('Salary difference: ' || sal\_diff);

END;

/

When the above code is executed at the SQL prompt, it produces the following result −

Trigger created.

The following points need to be considered here −

* OLD and NEW references are not available for table-level triggers, rather you can use them for record-level triggers.
* If you want to query the table in the same trigger, then you should use the AFTER keyword, because triggers can query the table or change it again only after the initial changes are applied and the table is back in a consistent state.
* The above trigger has been written in such a way that it will fire before any DELETE or INSERT or UPDATE operation on the table, but you can write your trigger on a single or multiple operations, for example BEFORE DELETE, which will fire whenever a record will be deleted using the DELETE operation on the table.

**Triggering a Trigger**

Let us perform some DML operations on the CUSTOMERS table. Here is one INSERT statement, which will create a new record in the table −

INSERT INTO CUSTOMERS (ID,NAME,AGE,ADDRESS,SALARY)

VALUES (7, 'Kriti', 22, 'HP', 7500.00 );

When a record is created in the CUSTOMERS table, the above create trigger, **display\_salary\_changes** will be fired and it will display the following result −

Old salary:

New salary: 7500

Salary difference:

Because this is a new record, old salary is not available and the above result comes as null. Let us now perform one more DML operation on the CUSTOMERS table. The UPDATE statement will update an existing record in the table −

UPDATE customers

SET salary = salary + 500

WHERE id = 2;

When a record is updated in the CUSTOMERS table, the above create trigger, **display\_salary\_changes** will be fired and it will display the following result −

Old salary: 1500

New salary: 2000

Salary difference: 500

CREATE OR REPLACE TRIGGER EMP\_TRG

INSTEAD OF INSERT OR UPDATE OR DELETE ON EMP\_VW

BEGIN

IF TO\_CHAR(SYSDATE,'HH24') BETWEEN 9 AND 24 THEN

DBMS\_OUTPUT.PUT\_LINE('USER CANNOT PERFORM DML AT THIS TIME');

END IF;

END;

/

DBMS\_OUTPUT –

BEGIN

--DBMS\_OUTPUT.DISABLE;

DBMS\_OUTPUT.ENABLE;

DBMS\_OUTPUT.PUT\_LINE('HELLO');

END;

/

BEGIN

DBMS\_OUTPUT.PUT('THIS IS JUST PUT');

--DBMS\_OUTPUT.PUT\_LINE('THIS IS PUT\_LINE');

DBMS\_OUTPUT.NEW\_LINE();

DBMS\_OUTPUT.PUT('THIS IS JUST PUT AGAIN');

DBMS\_OUTPUT.PUT\_LINE('THIS IS PUT LINE AGAIN');

END;

DECLARE

lines dbms\_output.chararr;

num\_lines number;

BEGIN

-- enable the buffer with default size 20000

dbms\_output.enable;

dbms\_output.put\_line('Hello Reader!');

dbms\_output.put\_line('Hope you have enjoyed the tutorials!');

dbms\_output.put\_line('Have a great time exploring pl/sql!');

num\_lines := 3;

dbms\_output.get\_lines(lines, num\_lines);

FOR i IN 1..num\_lines LOOP

dbms\_output.put\_line(lines(i));

END LOOP;

END;

/

UTL\_FILE PACKAGE

===============================

**STEP 1. CREATE DIRECTORY FROM SYSDBA USER – UTLDIR**

**sQL> CREATE DIRECTORY UTLDIR AS ‘D:\SACHIN\’**

**SQL> GRANT READ,WRITE ON DIRECTORY UTLDIR TO SCOTT**

**NOW LOG IN USING SCOTT AND TYPE FOLLOWING PROGRAM -**

**DECLARE**

LH UTL\_FILE.FILE\_TYPE;

L\_TXT **VARCHAR2**(1000);

**BEGIN**

LH := UTL\_FILE.FOPEN('UTLDIR','TESTFILE.TXT','R');

**LOOP**

UTL\_FILE.GET\_LINE(LH, L\_TXT);

DBMS\_OUTPUT.PUT\_LINE(L\_TXT);

**END** **LOOP**;

**EXCEPTION**

**WHEN** NO\_DATA\_FOUND **THEN**

UTL\_FILE.FCLOSE(LH);

**END**;

/

APPEND OPERATION

**DECLARE**

LH UTL\_FILE.FILE\_TYPE;

L\_TXT **VARCHAR2**(1000);

**BEGIN**

LH := UTL\_FILE.FOPEN('UTLDIR','TESTFILE.TXT','A');

UTL\_FILE.PUT\_LINE(LH, 'Appended from PL/SQL');

UTL\_FILE.FCLOSE(LH);

**END**;

APPEND – NEW FILE CREATED

DECLARE

LH UTL\_FILE.FILE\_TYPE;

L\_TXT VARCHAR2(1000);

BEGIN

LH := UTL\_FILE.FOPEN('UTLDIR','Demo\_write\_file.TXT','A');

UTL\_FILE.PUT(LH, 'UTL\_FILE is a great utility from Oracle to access OS files');

UTL\_FILE.FCLOSE(LH);

END;

/

Create table Leave

(Empno number(4),

S\_date date,

E\_date date,

snap blob,

msg clob)

===============

Insert into Leave values

(7900, '17-APR-98', '20-APR-98', NULL,'The LC and Amendments entry Forms have been completed. All the validat Insert into leave values

(7439,'12-APR-98', '17-APR-98', empty\_blob(),

'The assignments regarding Oracle 8 have

been completed. I''ll be back on 17th')ions have been incorporated and passed for testing.')

alter table Leave

add(b\_file bfile)

Create or Replace Directory L\_DIR as 'd:\sachin\'

CREATE OR REPLACE PROCEDURE b\_file(Eno in number) IS

LOC BFILE;

V\_FILEEXISTS INTEGER;

v\_FILEISOPEN INTEGER;

NUM NUMBER;

OFFSET NUMBER;

LEN NUMBER;

DIR\_ALIAS VARCHAR2(5);

NAME VARCHAR2(15);

CONTENTS LONG;

BEGIN

SELECT B\_FILE INTO LOC FROM LEAVE WHERE EMPNO=Eno;

-- Check to see if file exists

V\_FILEEXISTS := DBMS\_LOB.FILEEXISTS(LOC);

IF v\_FILEEXISTS = 1 THEN

DBMS\_OUTPUT.PUT\_LINE('The file exists');

ELSE

GoTo E;

END IF;

-- Check if file open

v\_FILEISOPEN := DBMS\_LOB.FILEISOPEN( LOC );

--Determine actions if file is opened or not

IF v\_FILEISOPEN = 1 THEN

DBMS\_OUTPUT.PUT\_LINE('The file is open');

ELSE

DBMS\_OUTPUT.PUT\_LINE('Opening the file');

DBMS\_LOB.FILEOPEN(LOC);

LEN := DBMS\_LOB.GETLENGTH( LOC );

DBMS\_OUTPUT.PUT\_LINE('Length of the file : ' ||

TO\_CHAR(LEN));

NUM := 40; OFFSET := 1;

DBMS\_LOB.READ(LOC, NUM, OFFSET, CONTENTS);

DBMS\_OUTPUT.PUT\_LINE('Contents of the file : '|| CONTENTS);

END IF;

DBMS\_LOB.FILEGETNAME(LOC, DIR\_ALIAS, NAME);

DBMS\_OUTPUT.PUT\_LINE ('Opening ' || dir\_alias ||

name);

DBMS\_LOB.FILECLOSE(LOC); -- Close the BFILE

<<E>>

DBMS\_OUTPUT.PUT\_LINE('The file cannot be found');

END;

/

SQLLDR

CREATE TABLE EMP1

AS SELECT EMPNO,ENAME,SAL

FROM EMP

/

TRUNCATE TABLE EMP1

load data

infile 'd:\sachin\mydata.txt'

into table emp1 fields terminated by ","

optionally enclosed by '"' ( empno, ename, sal, deptno )