## ASSIGNMENT NO: 3

**AIM : Write PL/SQL blocks for demonstrating triggers and cursors.**

**INDEX TERMS**: PL/SQL, Triggers, Cursors.

**THEORY**

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 (DML) statement (DELETE, INSERT, or UPDATE)
− A database definition (DDL) statement (CREATE, ALTER, or DROP).
− A database operation (SERVERERROR, LOGON, LOGOFF, STARTUP, or SHUTDOWN).

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

- 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 a trigger:**

```
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
END;
```

Here,

1. CREATE [OR REPLACE] TRIGGER trigger_name: It creates or replaces an existing trigger with the trigger_name.
2. {BEFORE | AFTER | INSTEAD OF} : This specifies when the trigger would be executed. The INSTEAD OF clause is used for creating trigger on a view.
3. {INSERT [OR] | UPDATE [OR] | DELETE}: This specifies the DML operation.
4. [OF col_name]: This specifies the column name that would be updated.
5. [ON table_name]: This specifies the name of the table associated with the trigger.
6. [REFERENCING OLD AS o NEW AS n]: This allows you to refer new and old values for various DML statements, like INSERT, UPDATE, and DELETE.
7. [FOR EACH ROW]: This specifies a row level trigger, i.e., the trigger would 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.
8. WHEN (condition): This provides a condition for rows for which the trigger would fire. This clause is valid only for row level triggers.
Following are the two very important point and should be noted carefully. OLD and NEW references are used for record level triggers these are not available for table 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.

**Cursor**

A cursor is a pointer to the context area. PL/SQL controls the context area through a cursor. A cursor holds the rows (one or more) returned by a SQL statement. The set of rows the cursor holds is referred to as the active set.You can name a cursor so that it could be referred to in a program to fetch and process the rows returned by the SQL statement, one at a time. There are two types of cursors Implicit cursors, Explicit cursors

**Implicit Cursors :**

Implicit cursors are automatically created by Oracle whenever an SQL statement is executed, when there is no explicit cursor for the statement. Programmers cannot control the implicit cursors and the information in it.Whenever a DML statement (INSERT, UPDATE and DELETE) is issued, an implicit cursor is associated with this statement. The following table provides the description of the most used attributes −

| Sr . No | Attribute & Description |
|---|---|
| 1 | %FOUND :Returns TRUE if an INSERT, UPDATE, or DELETE statement affected one or more rows or a SELECT INTO statement returned one or more rows. Otherwise, it returns FALSE. |
| 2 | %NOTFOUND: The logical opposite of %FOUND. It returns TRUE if an INSERT, UPDATE, or DELETE statement affected no rows, or a SELECT INTO statement returned no rows. Otherwise, it returns FALSE |
| 3 | %ISOPEN: Always returns FALSE for implicit cursors, because Oracle closes the SQL cursor automatically after executing its associated SQL statement. |
| 4 | %ROWCOUNT: Returns the number of rows affected by an INSERT, UPDATE, or DELETE statement, or returned by a SELECT INTO statement. |

**Example of Implicit Cursor:**

```
DECLARE
total_rows number(2);
BEGIN
UPDATE customers
SET salary = salary + 500;
IF sql%notfound THEN
dbms_output.put_line('no customers selected');
ELSIF sql%found THEN
total_rows := sql%rowcount;
dbms_output.put_line( total_rows || ' customers selected ');
END IF;
END;/
```

**Explicit Cursors**

Explicit cursors are programmer-defined cursors for gaining more control over the context area. An explicit cursor should be defined in the declaration section of the PL/SQL Block. It is created on a SELECT Statement which returns more than one row. Working with an explicit cursor includes the following steps −

1.Declaring the cursor for initializing the memory
2.Opening the cursor for allocating the memory
3.Fetching the cursor for retrieving the data
4.Closing the cursor to release the allocated memory

Declaring the Cursor

Declaring the cursor defines the cursor with a name and the associated SELECT statement. For example − CURSOR c_customers IS SELECT id, name, address FROM customers;

**Opening the Cursor**

Opening the cursor allocates the memory for the cursor and makes it ready for fetching the rows returned by the SQL statement into it.

For example, we will open the cursor as − OPEN c_customers;

**Fetching the Cursor**

Fetching the cursor involves accessing one row at a time. For example, fetch rows from the above- opened cursor as follows −

FETCH c_customers INTO c_id, c_name, c_addr;

**Closing the Cursor**

Closing the cursor means releasing the allocated memory. For example, we will close the opened cursor as −CLOSE c_customers;

**Example:**

```
DECLARE
c_id customers.id%type;
c_name customerS.No.ame%type;
CURSOR c_customers is SELECT id, name FROM customers;
BEGIN
OPEN c_customers;
LOOP
FETCH c_customers into c_id, c_name, c_addr;
EXIT WHEN c_customers%notfound;
dbms_output.put_line(c_id || ' ' || c_name || ' ' || c_addr);
END LOOP;
CLOSE c_customers;
END;/
```

**FAQs:**

1.What is the difference between execution of triggers and stored procedures?
2.Write query to disable a trigger?
3. Explain use of cursor?What is mean by % ROWTYPE and TYPE RECORD.

**Conclusion**:

Outcome of the experiment is understanding of

1. Execution of row level, statement level , Before and After Triggers.
2. Implement implicit,explicit and parameterized cursor.
3. Execute cursors with various types of cursor attributes.