Syllabus Topic : Exception Handling

4.3 Exception Handling

➔ **(MSBTE - W-13, W-14, W-15, W-16, S-17, W-17)**

Q. 4.3.1 Explain the exception handling with its two types.

(Ans. : Refer section 4.3)

W-13, W-16, W-17, 8 Marks

Q. 4.3.2 Explain 'exception handling' in 'PL-SQL'. *(Refer section 4.3)* **W-14, 8 Marks**

Q. 4.3.3 Describe exception handling in brief. *(Refer section 4.3)* **W-15, 8 Marks**

Q. 4.3.4 Describe Exception handling. Explain with example. *(Refer section 4.3)* **S-17, 8 Marks**

- Exception is a warning or error condition which may interrupt the program execution.
- It can be defined as an error situation, arising during program execution.
- An exception is raised when an error occurs, the normal execution is stopped and control transfers to exception handler code.
- Exceptions are predefined or can be user defined.
- E.g. ZERO_DIVIDE and STORAGE_ERROR are predefined exceptions
- User defined Exceptions can be written in the declare section of any PL/SQL block, subprogram, or package.
- *Exception Handlers* is a block of code written to handle the raised exception. After an exception handler is executed the control passes back to the next statement after the statement where exception was raised.
- Handling the exceptions ensures that PL/SQL blocks do not exit unexpectedly
- Internal exceptions are raised automatically by the run-time system.
- User defined exceptions must be raised explicitly using *RAISE* statements.
- Predefined exceptions may also be explicitly raised using *RAISE* statements.
- Exceptions improve readability and reliability of code as it has a separate Error Handling blocks.

PL/SQL Exception message consists of following three parts:

- Type of Exception
- An Error Code
- A message

Structure for Exception Handling

The coding of Exception handling section in PLSQL is quite simple and easy to understand.

Syntax

```
DECLARE
Declaration section ....
BEGIN
Begin section.....
EXCEPTION
WHEN Exception_one THEN
Exception handler code to handle the errors
WHEN Exception_two THEN
Exception handler code to handle the errors
WHEN Others THEN
Exception handler code to handle the errors
END;
/
```

- When an Exception is raised in a PL/SQL block, Oracle searches for an appropriate exception handler in the exception section. E.g. in the above example, if the error raised is 'Exception_one', then the error is handled according to the statements under it. 'WHEN Others' exception is used to manage the exceptions that are not explicitly handled.
- When an exception is raised in inner PL/SQL block it should be handled in the exception block of the inner PL/SQL block, otherwise the control moves to the Exception block of the next upper PL/SQL block. If none of the blocks handle the exception the program ends abruptly with an error.

PL/SQL Exception Handling block

```
SQL>EXCEPTION
2 WHEN NO_DATA_FOUND THEN
3 v_msg := 'error found';
4 v_err := SQLCODE;
5 v_prog := 'fixdebt';
6 INSERT INTO errlog VALUES (v_err, v_msg, v_prog,
SYSDATE, USER);
7 WHEN OTHERS THEN
8 v_err := SQLCODE;
9 v_msg := SQLERRM;
10 v_prog := 'fixdebt';
11 INSERT INTO errlog VALUES (v_err, v_msg, v_prog,
   SYSDATE, USER);
12 RAISE;
```

Syllabus Topic : Predefined Exception

4.3.1 Predefined Exception

➔ (MSBTE - S-16)

Q. 4.3.5 What are predefined exception and user defined exceptions ?

(Refer sections 4.3.1 and 4.3.2) **S-16, 4 Marks**

(1) Named system exception

- The Named system exceptions are exceptions that have been already given names in the STANDARD package in PL/SQL.
- Named system exceptions need not be declared explicitly, they are raised implicitly when a predefined Oracle error occurs, also they are caught by referencing the standard name within an exception handling routine.
- Following is a standard set of Oracle exceptions :

Oracle Error	Exception Name	Description
ORA-00001	DUP_VAL_ON_INDEX	Duplicate value created in a field restricted by unique index
ORA-00051	TIMEOUT_ON_RESOURCE	Waiting for a resource which is timed out
ORA-00061	TRANSACTION_BACKED_OUT	Portion of transaction rolled back
ORA-01001	INVALID_CURSOR	Referencing cursor which did not exist
ORA-01012	NOT_LOGGED_ON	Executing calls to Oracle before logging in
ORA-01017	LOGIN_DENIED	Logging in with invalid username or password
ORA-01403	NO_DATA_FOUND	No rows returned as result of query
ORA-01422	TOO_MANY_ROWS	More than one row returned
ORA-01476	ZERO_DIVIDE	Executing Divide by zero
ORA-01722	INVALID_NUMBER	Trying to convert string to number
ORA-06500	STORAGE_ERROR	Memory insufficient or corrupted
ORA-06501	PROGRAM_ERROR	Internal program error
ORA-06502	VALUE_ERROR	Invalid character or numeric value
ORA-06511	CURSOR_ALREADY_OPEN	Opening a cursor that is already open

Example

Catching Named System Exception called DUP_VAL_ON_INDEX

```
SQL>CREATE OR REPLACE PROCEDURE new_dept
(d_no IN
2 CHAR, d_name IN VARCHAR2)
```

```
3 IS
4 BEGIN
5 INSERT INTO department (dept_no, dept_name)
VALUES
  ( d_no, d_name );
6 EXCEPTION
7 WHEN DUP_VAL_ON_INDEX THEN
8 raise_application_error (-25001,'Department No. alreading
  existing');
9 WHEN OTHERS THEN
10 raise_application_error (-25002,'Error inserting new
   Department.');
11 END;
12 /
```

Q. 4.3.6 Write a PL/SQL program which handles divide by zero exception. *(Refer section 4.3.1)*

S-15, 4 Marks

Example

ZERO_DIVIDE EXCEPTION

```
SQL> declare
2 num number;
3 begin
4 num :=78/0;
5 exception
6 when ZERO_DIVIDE then
7 dbms_output.put_line('Dividing by Zero Error');
8 end;
9 /
```

Dividing by Zero Error

PL/SQL procedure successfully completed.

(2) Unnamed System Exceptions

- System exceptions for which oracle do not provide names are known as unnamed system exception. These Exceptions have a code and an associated message.

There are two ways to handle unnamed system exceptions :

- Using the WHEN OTHERS exception handler, or
- Associating the exception code to a name and using it as a named exception.
- It is possible to assign a name to unnamed system exceptions using a Pragma called EXCEPTION_INIT. EXCEPTION_INIT will associate a predefined Oracle error number to a user_defined exception name.

Example

Declaring an unnamed system exception using EXCEPTION_INIT

Syntax

```
DECLARE
exception_name EXCEPTION;
PRAGMA
EXCEPTION_INIT (exception_name, Err_code);
BEGIN
Execution section
EXCEPTION
WHEN exception_name THEN
handle the exception
END;
/
```

Here, 'exception_name' is the name of a previously declared exception and the number is a negative value corresponding to an ORA- error number. The pragma must appear somewhere after the exception declaration in the same declarative section, as shown in the following example:

```
DECLARE
error_found EXCEPTION;
PRAGMA EXCEPTION_INIT(error_found, -45);
BEGIN
...// -- code that causes an ORA-00045 error
EXCEPTION
WHEN error_found THEN
// -- code to handle the error
END;
/
```

Example

When we try to delete a record from parent table while a child record still exist, an exception will be thrown with oracle code number -1517. User can provide a name to this exception and handle it in the exception as shown below:

```
SQL>DECLARE
2 Child_record_exception EXCEPTION;
3 PRAGMA
4 EXCEPTION_INIT (Child_record_exception, -1517);
5 BEGIN
6 Delete FROM department where dept_no = 'D001';
7 EXCEPTION
8 WHEN Child_record_exception
9 THEN dbms_output.put_line('Child records for this
department exist in employee table');
10 END;
11 /
```

Syllabus Topic : User defined Exception

4.3.2 User defined Exception

- Due to the complex structure of relational databases it is not always possible to handle all the unwanted exceptions using Named and Un-named exceptions. Therefore there is a need to name and catch exceptions that aren't defined by PL/SQL. These are called Named Programmer-defined exceptions or user defined exception.
- Named Programmer-defined exceptions should be explicitly declared in the declaration section.
- They should be explicitly raised in the execution section.
- They should be handled by referencing the user-defined exception name in the exception section.

Syntax

Named programmer-defined exception in a function

```
CREATE [OR REPLACE] FUNCTION function_name
[ (param1 [,param2]) ]
RETURN return_datatype
IS | AS
[declaration_section]
exception_name EXCEPTION;
BEGIN
executable_section
RAISE exception_name ;
EXCEPTION
WHEN exception_name THEN
[statements]
WHEN OTHERS THEN
[statements]
END [function_name];
/
```

Syllabus Topic : Cursor

4.4 Cursor

➔ **(MSBTE - W-13, S-15, W-15, S-16, W-16, S-17, W-17)**

Q. 4.4.1 Define cursor ? List the two types of cursor. *(Refer section 4.4)* **W-13, W-16, 2 Marks**

Q. 4.4.2 What is cursor ? *(Refer section 4.4)* **S-15, W-15, S-17, 2 Marks**

Q. 4.4.3 Explain implicit and explicit cursors. *(Refer sections 4.4.1 to 4.4.2)* **W-15, S-17, W-17, 4 Marks**

Q. 4.4.4 List types of cursor and explain each with example. *(Refer sections 4.4 to 4.4.2)* **S-16, 4 Marks**

DEFINITION

***Cursor** is a pointer to the private area in SQL that stores information about processing a specific DML statement.*

- The drawback with select statement is that it returns only one row at a time in a PL/SQL block.
- Cursors are capable of holding more than one row. It used for grouping a set of rows and implementing a similar logic to all the records of that group.
- Set of rows retrieved in such an area is called the *active set* and its size depends on the number of rows retrieved by the search condition of the query.
- Oracle reserves an area in memory called cursor, populates this area with appropriate data and frees the memory area when the task is completed.
- Cursors can be classified as :

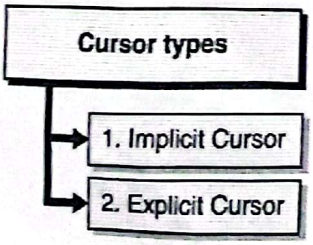


Fig. 4.4.1 : Cursor types

➔ **1. Implicit Cursor**

These cursors are also called as internal cursors and are managed by Oracle itself

➔ **2. Explicit Cursor**

These are User-defined Cursor used for external Processing

Syllabus Topic : Implicit Cursor

4.4.1 Implicit Cursor

- **Implicit cursor** is a session cursor which is opened every time you run a SELECT or DML statement in the PL/SQL block.
- An implicit cursor closes after its associated statement run but its attributes values remain available until another SELECT or DML statement is executed.
- Cursor attributes return information about the state of the cursor.
- The syntax for an implicit cursor attribute is **SQL*attribute***(e.g. SQL%FOUND).
- SQL*attribute* will always refer to the recently run DML or SELECT INTO statement.

☞ **Implicit cursor attributes**

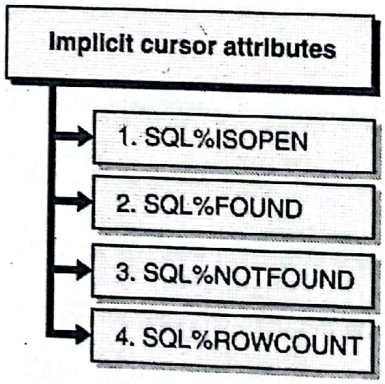


Fig. 4.4.2 : Implicit cursor attributes

➔ **1. SQL%ISOPEN**

Oracle automatically opens and closes implicit cursors associated to any DML or Select statement. Therefore the value for this attribute is always false.

2. SQL%FOUND

If any DML or Select into statement returned any row then the value of this attribute is true, if no rows were returned then the value is false otherwise when no recent statement is executed then this value is null.

3. SQL%NOTFOUND

It works like the SQL%FOUND but contrary results are obtained with SQL%NOTFOUND that is false if rows are returned, true if no rows are returned and null if no recent statement is executed.

4. SQL%ROWCOUNT

It stores null if no recent DML statements are executed and stores the count of the rows affected by the DML statement of Select into statement.

Sr. No.	Cursor Attributes	Description
1.	SQL%ISOPEN	Returns true if the cursor is open otherwise false
2.	SQL%FOUND	If any row is returned by DML or select into then true otherwise false
3.	SQL%NOTFOUND	Returns true when no row is returned by DML or select into statement
4.	SQL%ROWCOUNT	Returns the number of rows affected by DML statements

Example

Using CURSOR attributes

```
SQL>SET SERVEROUTPUT ON
2 BEGIN
3 UPDATE emp SET emp_sal  = emp_sal +1000 WHERE
  dept_no = 'D001';
4 IF SQL%FOUND THEN
5 DBMS_OUTPUT.PUT_LINE ('Salary Updated.....');
6 END IF;
7 IF SQL%NOTFOUND THEN
8 DBMS_OUTPUT.PUT_LINE ('Record not found.....');
9 END IF;
10 IF SQL%ROWCOUNT > 0 THEN
11 DBMS_OUTPUT.PUT_LINE (SQL%ROWCOUNT' ||
   'rows are updated');
12 END IF;
13 ELSE
14 DBMS_OUTPUT.PUT_LINE ('No records to be
updated');
15 END IF;
16 END;
17 /
```

```
3 rows are updated

PL/SQL procedure successfully completed.
```

Drawbacks of Implicit Cursors

The implicit cursor has the following drawbacks :

1. Implicit cursors are less efficient than an explicit cursor.
2. Implicit cursors are more vulnerable to data errors.
3. Implicit cursors provide less flexibility to implement programming control.

Syllabus Topic : Explicit Cursor - Declaring, Opening and closing a cursor, Fetching a Record from Cursor

4.4.2 Explicit Cursor

(MSBTE - W-14)

Q. 4.4.5 Write step by step syntax to create, open and close cursor in PL/SQL block. *(Refer section 4.4.2)* **W-14, 4 Marks**

- The Cursor which is declared by user is called as Explicit Cursor.
- Following are the steps of using an explicit Cursor :

Step1 : DECLARE Cursor in the declaration section.

☞ **Example**

```
CURSOR c_emp IS SELECT emp_no, emp_name, emp_sal,
emp_addr FROM emp;
```

In above example we created a cursor with name c_emp which is associated with emp table.

Step 2 : OPEN the cursor in the Execution Section.

General Syntax to open a cursor is:

```
OPEN cursor_name;
```

Step 3 : FETCH the data from cursor into PL/SQL variables or records in the Execution Section.

☞ **General Syntax to fetch records from a cursor is**

```
FETCH cursor_name INTO record_name;
```

OR

```
FETCH cursor_name INTO variable_list;
```

Step 4 : CLOSE the cursor in the Execution Section before we end the PL/SQL Block.

☞ **General Syntax to close a cursor is**

```
CLOSE cursor_name;
```

Following are the statements used and corresponding actions performed:

Statements	Action performed
OPEN c_emp;	Opens the cursor
FETCH c_emp INTO var_emo_no, var_emp_name, var_emp_sal, var_emp_addr;	Rows fetched from table into variables
CIOSEc_emp;	Closes the cursor

☞ **Explicit cursor variables**

Cursor variable	Description
c%ISOPEN	Returns true if the cursor is open otherwise false
c%FOUND	If any row is returned by DML or select into then true otherwise false
c%NOTFOUND	Returns true when no row is returned by DML or select into statement
c%ROWCOUNT	Returns the number of rows affected by DML statements

☞ **Example**

Using cursor variables

```
1 SQL> Declare
2 var_emp_no char(4);
3 var_emp_namevarchar(10);
4 var_emp_sal number(8,2);
5 var_emp_addrvarchar(10);
6 cursor c_emp is
7 select emp_no, emp_name, emp_sal, emp_addr from emp;
8 begin
9 open c_emp;
10 loop
11 fetch c_emp into
  var_emp_no,var_emp_name,var_emp_sal,var_emp_addr;
12 exit when c_emp%notfound;
13 var_new_sal := var_emp_sal +5000;
14 insert into emp
15
values(var_emp_no,var_emp_name,var_new_sal,var_emp_a
ddr);
16 end loop;
17 close c_emp;
18 end;
19 /
```

PL/SQL procedure successfully completed.

Syllabus Topic : Cursor for loop

4.4.3 Cursor for loop

When FOR LOOP is used in cursor we do not have to declare a record or variables to store the cursor values. It is not necessary to open, fetch and close the cursor as the FOR LOOP automatically performs these actions.

Syntax

```
FOR record IN cursor_name
LOOP
Action 1
Action 2
.
.
END LOOP;
```

Example

```
1 SQL> declare
2 cursor c_emp (esalnumber) is
3 select * from emp where emp_sal = esal;
4 begin
5 for i_emp in c_emp(10) loop
6 update emp
7 set emp_sal = emp_sal + 5000
8 where emp_sal  = i_emp.sal;
9 DBMS_OUTPUT.put_line('Employee Name: '||
i_emp.emp_name||' Employee New Salary:'10
||i_emp.emp_sal);
11 end loop;
12 end;
13 /
```

Output

```
Employee Name : Korth Employee
New Salary: 40560
PL/SQL procedure successfully completed
```

Syllabus Topic : Parameterized Cursor

4.4.4 Parameterized Cursor

➔ (MSBTE - W-15)

Q. 4.4.6 Explain parameterized cursor with example. *(Refer section 4.4.4)* **W-15, 4 Marks**

- Parameterized cursors are static cursors that can accept passed-in parameter values when they are opened.
- The following example includes a parameterized cursor. The cursor displays the name and salary of each employee in the EMP table whose salary is less than that specified by a passed-in parameter value.

Example

```
DECLARE
  my_record emp%ROWTYPE;
  CURSOR c1 (max_wage NUMBER) IS
SELECT * FROM emp WHERE sal < max_wage;
BEGIN
  OPEN c1(5000);
  LOOP
    FETCH c1 INTO my_record;
    EXIT WHEN c1%NOTFOUND;
    DBMS_OUTPUT.PUT_LINE('Name = ' ||
my_record.ename || ', salary = '
      || my_record.sal);
  END LOOP;
  CLOSE c1;
END;
```

If 5000 is passed in as the value of *max_wage*, only the name and salary data for those employees whose salary is less than 5000 is returned.

Output

```
Name = Ram, salary = 1500.00
Name = Krishna, salary = 1600.00
Name = Varad, salary = 1550.00
Name = Manthan, salary = 3950.00
```

Syllabus Topic : Procedures

4.5 Procedures

➔ **(MSBTE - W-13, W-14, S-17)**

Q. 4.5.1 Explain following with syntax.
(i) Function (ii) Procedure.
(Refer section 4.5) **W-13, 4 Marks**

Q. 4.5.2 List advantages of Procedures.
(Refer section 4.5.1) **W-14, 4 Marks**

Q. 4.5.3 Define following term with example
(i) Procedure *(Refer section 4.5)*
(ii) Function *(Refer section 4.6)* **S-17, 4 Marks**

- Stored procedure is a group of SQL statements which can be executed repeatedly. It allows variable declarations, flow control and other useful programming techniques. Parameters are the important part of procedures.
- The parameters make the stored procedure more flexible and useful. Stored procedures can increase productivity by writing once and using it many times.

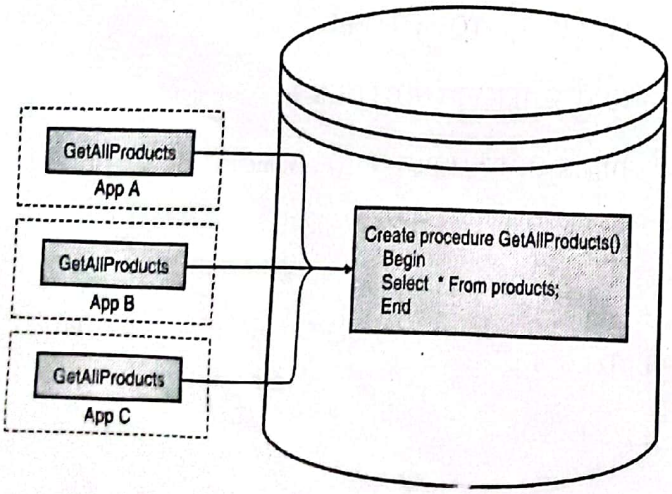


Fig. 4.5.1

Syllabus Topic : Advantages of Stored Procedure

4.5.1 Advantages of Stored Procedure

- Stored procedure can Share logic with different applications.
- Stored procedures can **isolate** users from data tables. This gives you the ability to grant access to the stored procedures that manipulate the data but not directly to the tables.
- Stored procedure can provide security that means users can input data and also they can change it but they could not write procedure because of data access controls on them.
- Stored procedure can be used to improve performance because it reduces network traffic. With a stored procedure, multiple calls can be kept into one.
- Stored procedure is fast because it is precompiled.
- It is easy to maintain.
- It improves reliability and reduced development time.
- Increased security-database administrator can control the users who access the stored procedure.
- Reduce network usage among servers and clients.

Syllabus Topic : Creating Procedure

4.5.2 Creating Procedure

☞ **Syntax**

```
CREATE [OR REPLACE] PROCEDURE procedure_name
[ (param1 [,param2]) ]
IS
[declaration_section]
BEGIN
[executable_section]
EXCEPTION
[ exception_section]
```

```
END [procedure_name];
/
```

☞ **Example**

```
SQL> CREATE OR REPLACE PROCEDURE DEMO AS
2 BEGIN
3 DBMS_OUTPUT.PUT_LINE('Welcome to PL/SQL');
4 END;
5 /
```

Procedure created.

Syllabus Topic : Executing Stored Procedure

4.5.3 Executing Stored Procedure

You can call procedure in three ways-

☞ **Using EXECUTE**

```
SQL> EXECUTE DEMO
Welcome to PL/SQL
```

PL/SQL procedure successfully completed.

☞ **Using CALL**

```
SQL> call DEMO();
Welcome to PL/SQL
```

Call completed.

☞ **Using PL/SQL block**

```
1 SQL> begin
2 DEMO();
3 end;
4 /
Welcome to PL/SQL
```

PL/SQL procedure successfully completed.

Syllabus Topic : Deleting Stored Procedure

4.5.4 Deleting Stored Procedure

DROP is used for destroying the Procedure.

☞ **Syntax**

```
DROP PROCEDURE procedure_name;
```

☞ **Example**

```
SQL>DROP PROCEDURE DEMO;
Procedure dropped.
```

☞ **Example**

➔ **(MSBTE - S-15, W-15)**

Q. 4.5.4 Write PL/SQL procedure to calculate factorial of a given number. *(Refer section 4.5.4)* **S-15, 4 Marks**

Q. 4.5.5 Write PL/SQL program to display factorial of any number. *(Refer section 4.5.4)* **W-15, 4 Marks**

```
SQL> create or replace procedure factorial as
declare
  n integer;
  fact integer:=1;
  i integer;
begin
  n:=&n;
  for i in 1..n
  loop
    fact:=fact*i;
  end loop;
  dbms_output.put_line('factorial='||fact);
end factorial;/
```

Q. 4.5.6 What statement is used in PL/SQL to display the output ? *(Refer section 4.5.4)* **W-15, 2 Marks**

Use a procedure called dbms_output.put_line.

Syllabus Topic : Functions In PL/SQL

4.6 Function

→ (MSBTE - W-16)

Q. 4.6.1 Explain function in PL/SQL with suitable example.
(Refer section 4.6) W-16, 4 Marks

- **Stored function** or **user defined function** are very similar to procedures, except that a function returns a value to the environment from where it is called.
- When we want a unit of code to perform a specific task and return result, we should use a function. Procedures should be used when the code should only perform tasks like insertion, updation etc. then it need not return any result.

☞ Function Parameters

The Parameter modes define the behavior of formal parameters. The three parameter modes, IN (Default), OUT, and IN OUT, can be used with any subprogram.

1. **IN** - The parameter can be referenced by the procedure or function. The value of the parameter cannot be overwritten by the procedure or function.
2. **OUT** - The parameter cannot be referenced by the procedure or function, but the value of the parameter can be overwritten by the procedure or function.
3. **IN OUT** - The parameter can be referenced by the procedure or function and the value of the parameter can be overwritten by the procedure or function.

Syllabus Topic : Creating Function

4.6.1 Creating Function

☞ Syntax for creating function

```
CREATE [OR REPLACE] FUNCTION function_name
[ (param1 [,param2]) ]
RETURN return_datatype
IS | AS
[declaration_section]
BEGIN
[ executable_section]
EXCEPTION
[ exception_section]
END [function_name];
/
```

☞ Example 1

```
SQL> set serveroutput on
SQL> CREATE OR REPLACE FUNCTION SUM (A INT, B
INT) RETURN INT IS
2 BEGIN
3 RETURN (A + B);
4 END;
5 /
```

Function created.

☞ Example 2

```
SQL> CREATE OR REPLACE FUNCTION JOINS (s1
STRING, s2 STRING) RETURN STRING IS
2 BEGIN
3 str3 := s1 + s2;
4 RETURN (str3);
5 END;
6 /
```

Function created.

Syllabus Topic : Executing Function

4.6.2 Executing Function

☞ **FUNCTION CALL**

```
SQL> BEGIN
2 DBMS_OUTPUT.PUT_LINE ('RESULT IS:' || SUM
(652,310));
3 END;
4 /
RESULT IS: 962
```

PL/SQL function successfully completed.

Syllabus Topic : Deleting a Function

4.6.3 Deleting a Function

DROP is used to destroy a function.

☞ **Syntax**

```
DROP FUNCTION function_name;
```

☞ **Example**

```
SQL>DROP FUNCTION SUM;
```

Function dropped.

Syllabus Topic : Advantages of functions

4.6.4 Advantages of functions

- Functions help to extend the functionality PL/SQL language.
- Functions help to break a program into manageable, well-defined modules.
- Functions promote reusability. Once created, a function can be called any number of times.
- Functions supports maintainability. It is possible to change code of a function without affecting other functions.
- Functions help to differentiate tasks in a program.

4.6.5 Difference between Procedures and Functions

➔ **(MSBTE - S-15)**

Q. 4.6.2 Differentiate between function and procedure. *(Refer section 4.6.4)* **S-15, 4 Marks**

Sr. No.	FUNCTION	PROCEDURE
1.	A function must always return a value	A procedure cannot return a value
2.	The return statement in a function returns control to the calling program and returns the results of the function	The return statement of a procedure returns control to the calling program but no value
3.	Functions can return data in OUT and IN OUT parameters	Procedures can also return data in OUT and IN OUT parameters
4.	Functions can be called from SQL	Procedure cannot be called from SQL
5.	Functions are considered expressions	Procedure cannot be considered as expressions

Syllabus Topic : Database Trigger

4.7 Database Trigger

➔ **(MSBTE - W-13, S-14, W-14, S-15, S-16, W-16, S-17, W-17)**

Q. 4.7.1 What is database trigger ? Compare database trigger and procedures and explain use of database trigger. *(Refer sections 4.7, 4.7.1 and 4.7.4)* **W-13, S-16, 4 Marks**

Q. 4.7.2 Explain trigger with suitable example. *(Refer section 4.7)* **S-14, 4 Marks**

Q. 4.7.3 What is trigger ? List its types. *(Refer sections 4.7 and 4.7.2)* **W-14, 4 Marks**

Q. 4.7.4 How to create trigger ? State any two advantages of trigger. *(Refer sections 4.7.1 and 4.7.3)* **S-15, 4 Marks**

Q. 4.7.5 What is database Trigger ? How to create Trigger ? *(Refer sections 4.7 and 4.7.3)* **W-16, 4 Marks**

Q. 4.7.6 Explain the concept of trigger.
(Refer section 4.7) **S-17, 4 Marks**

Q. 4.7.7 Define database trigger, Compare database trigger and procedures.
(Refer sections 4.7 and 4.7.4) **W-17, 4 Marks**

DEFINITION

A trigger is a PL/SQL block structure or stored procedure that defines an action the database should take when some data base related event occurs.

- It plays automatically depending on the particular condition. A trigger is fired or executed when a DML statements like Insert, Delete, Update is executed on a database table.
- A trigger is triggered automatically at predefined timing and event, when an associated DML statement is executed.
- Triggers are useful in achieving security and auditing. It also maintains referential integrity and data integrity.

Syllabus Topic : Use of Trigger

4.7.1 Uses of Trigger

- To check the integrity of data.
- To catch errors in business logic.
- SQL triggers run the scheduled tasks
- SQL triggers are very useful to audit the changes of data in tables.

Syllabus Topic : Types of Triggers, How to apply database Trigger

4.7.2 Types of Triggers

There are various types of triggers.

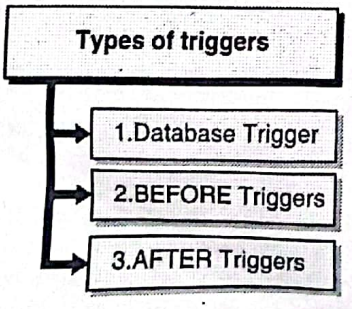


Fig. 4.7.1 : Types of triggers

➜ **(1) Database Trigger**

Database trigger consist of four parts:

(i) The trigger type

(ii) The triggering event

(iii) Trigger condition

(iv) Trigger body

☞ **Database Trigger is applied as follows**

- The Database triggers are activated on any event occurring in the database, while application trigger are restricted to an application.
- Database triggers can be created on top of table, view, schema or database.
- The events can be Before Insert, After Insert, Before Update, After Update, Before Delete and After Delete.

➜ **(2) BEFORE Triggers**

BEFORE triggers are also used to derive specific column values before completing a triggering INSERT or UPDATE statement.

➜ **(3) AFTER Triggers**

The Oracle-defined *AFTER ROW* trigger, We can create as many triggers of the preceding different types as we need for each type of DML statement (INSERT,UPDATE, or DELETE). For example, suppose we have a table result, and we want to know when the table is being accessed and the types of queries being issued.

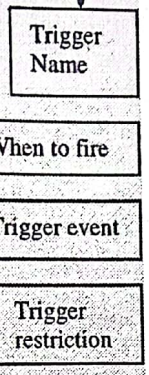
Syllabus Topic : Syntax for Creating Trigger

4.7.3 Syntax for Creating Trigger

☞ **Syntax**

To create trigger

```
CREATE [OR REPLACE] TRIGGER trigger_name
{BEFORE | AFTER | INSTEAD OF}
{INSERT [OR] | UPDATE [OR] | DELETE}
[OF column_name]
ON table_name
[for each row[when<trigger condition>]
<trigger body>
Begin
---statements
[REFERENCING OLD AS o NEW AS n]
End;
```



The following example shows how to create database trigger:-

- Assume two files, temp1 and temp2, having single column.
- Create table temp1(description varchar2(50));
- Create table temp2(description varchar2(50));

```
SQL>Create or replace trigger  ABC
2 Before insert
3 On temp1
4 Begin
5 Insert into temp2 values('test trigger');
6 End;
7 /
SQL> insert into temp1 values('learning tringger');
SQL> Select * from temp2;
```

Output

```
in temp2 file
Test trigger
```

On executing above example, the database trigger abc will be executed and trigger will be stored for further use. Once you will insert record in temp1 file , the trigger ABC will insert a record automatically in temp2 file

Syllabus Topic : Deleting Trigger

4.7.4 Deleting Trigger

Deleting Triggers

Triggers can be deleted using DROP common.

Syntax

```
DROP TRIGGER trigger_name;
```

Difference between trigger and stored procedure

Parameters	Procedures	Triggers
Execute	They can be executed whenever required.	They are automatically executed on occurrence of specified event.
Parameters	We can pass parameters to procedures.	We cannot pass parameters to triggers.
Return value	May return values on execution	No return value
Calling	We can call a procedure inside a trigger.	We can't call trigger inside a procedure.

Q. 4.7.8 Write PL/SQL program to display the largest of three number. *(Refer section 4.7.4)*

W-13, 4 Marks

```
declare
a integer;
b integer;
c integer;
begin
a:=&a;
```

```
b:=&b;
c:=&c;
if a=b and b=c and c=a
then dbms_output.put_line('ALL ARE EQUAL');
elsif a>b and a>c then
dbms_output.put_line('A IS GREATER');
elsif b>c then dbms_output.put_line('B IS GREATER');
else dbms_output.put_line('C IS GREATER');
end if;
end;
```

Output

```
OUTPUT: SQL> @ GREATESTOF3.sql
17 /
Enter value for a: 8
old 6: a:=&a;
new 6: a:=8;
Enter value for b: 9
old 7: b:=&b;
new 7: b:=9;
Enter value for c: 7
old 8: c:=&c;
new 8: c:=7;
B IS GREATER
PL/SQL procedure successfully completed.
```

Q. 4.7.9 Write PL/SQL program to display square of any number. *(Refer section 4.7.4)* **W-14, 4 Marks**

```
declare
n integer;
sqr integer;
begin
n:=:n;
sqr:=n*n;
dbms_output.put_line('Given Number is '||n);
dbms_output.put_line(' ');
dbms_output.put_line('Square of Number is '||sqr);
end;
```

Output

```
Given Number is 7
Square of Number is 49
```

Q. 4.7.10 Write PL/SQL program to print even or odd numbers from given range. (Accept number from user) *(Refer section 4.7.4)* **W-15, 4 Marks**

```
Declare
 n integer(4):=&n;
 Begin
 if mod(n,2)=0
 then
   dbms_output.put_line(n||' even number');
else
    dbms_output.put_line(n||' odd number');
 end if;
end;
/
```

Output

```
SQL>@e:\plsql\even.sql
Enter value for n: 5
old  2:  n number(4):=&n;
new  2:  n number(4):=5;
5 odd number
```

Q. 4.7.11 Write a PL/SQL program using while loop to display n even numbers.

(Refer section 4.7.4) W-17, 4 Marks

```
DECLARE
N integer(3) :=0;
        I integer(4):=&I;
BEGIN
WHILE N<=I
LOOP
N :=N+2;
DBMS_OUTPUT.PUT_LINE(N);
END
LOOP;
END;
```

❑❑❑