The Shirpur Education Society's

R. C. Patel College of Engineering & Polytechnic, Shirpur

Department of Computer Engineering and Computer Science & Engineering

---

Course Title - DATABASE MANAGEMENT SYSTEM (DMS)

Course Code - 313302

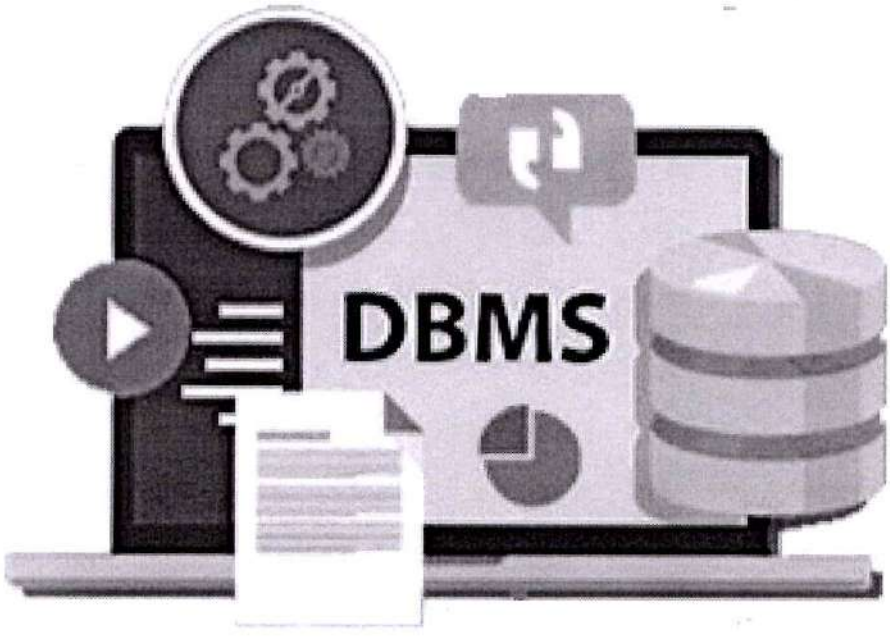
Programme Name - Computer Engineering and Computer Sci. & Engineering

Semester - Third

Unit - IV PL/SQL Programming

Total Marks: 18

Prepared By: Mr. Yogeshwar L. Patil



Unit-IV PL/SQL Programming

Marks-18

* Introduction —

PL/SQL stands for Procedural language / structured Query language.

- An extension to SQL with design feature of programming language
- It allows user to write program to do the various operations on database.
- PL/SQL is combination of SQL statements with programming language.
- PL/SQL offers a set of procedural commands CTE statement, loops, assignments, organized within blocks.
- PL/SQL integrates control statements and conditional statements with SQL.

* Advantages of PL/SQL —

1. Procedural language Capability - PL/SQL consist of procedural language such as conditional statement and loops like(FOR loop)
2. Modularized program development -
   - the basic unit of in PL/SQL program is block. All PL/SQL program consist of blocks.
   - These blocks can be thought as modules and can be 'modularized' in sequence or nested in other blocks.
   - You can break your application into smaller module
3. Improved performance —
   - PL/SQL allows you to logically combine multiple SQL statements as one unit or block.
   - PL/SQL engine processes multiple SQL statements simultaneously as single block, thereby reducing network traffic & provides high performance for the application
4. portability -
   - Application written in PL/SQL are portable to any computer hardware & operating system.
   - You can write portable packages and create libraries that can be reused on oracle databases in different environment

5. Exception Handling – an exception is an error that occur in the database or in user's program during runtime.
e.g hardware or network failures, application logic errors, data integrity error & so on.

- You can prepare for errors by writing exception handling code.
- Exception handling code tells your program what to do in the event of an exception.
- PL/SQL allows you to handle database and program exceptions efficiently. you can define seperate blocks for dealing with exceptions.

* PL/SQL Block Structure –

- PL/SQL is block structured language. Each PL/SQL program consist of SQL and PL/SQL statements. The basic unit in PL/SQL is block. A set of logically related statements forms a block.
- PL/SQL block contains three sections –
  1. Declare section
  2. Executable Section
  3. Exception Handling Section

```
DECLARE (optional)
/* variables, cursors, user
   defined exceptions.
BEGIN (Mandatory)
/* SQL statements
   PL/SQL statements
EXCEPTION (optional)
/* Action to perform when
   error occur
END (mandatory)
```

fig. PL/SQL Block structure.

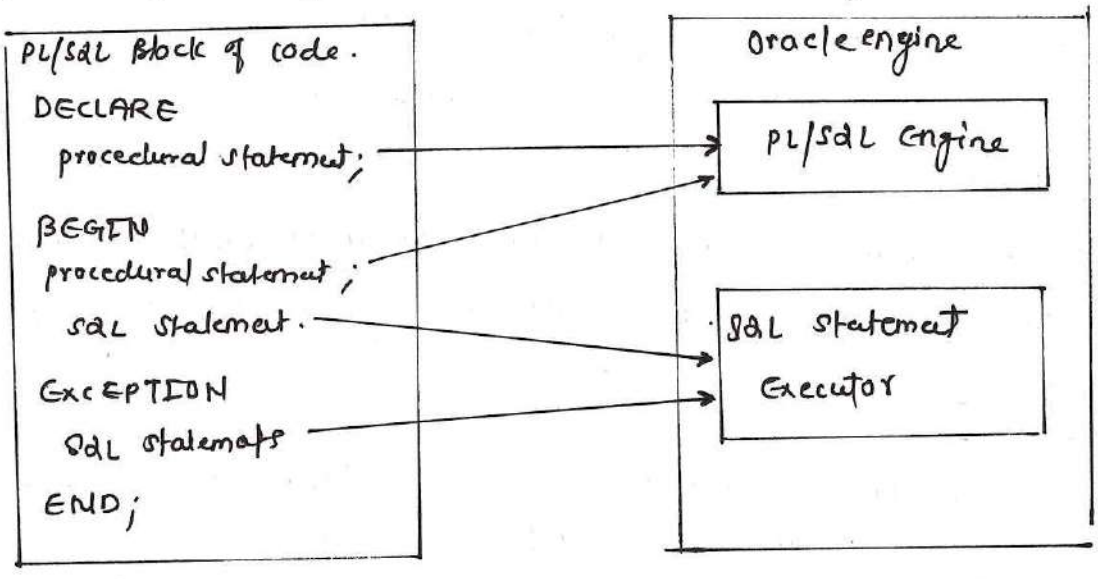
1. Declare section – This is optional section start with keyword 'DECLARE'. Declare section is used to declare variables, constants, cursor etc.
2. Executable Section – This section is compulsary. This section is starts with keyword 'BEGIN' and ends with END

The programs logic is written in this section.

3. Exception Handling Section - This section is optional and starts with the keyword 'EXCEPTION' It is used to write the code for handling the errors in the program.

PL/SQL execution Environment -

- the PL/SQL compilation and run-time system is an engine that compiles and executes PL/SQL blocks and subprograms.
- the engine can be installed in an Oracle server or in application development tool such as Oracle forms.



- Block of PL/SQL are passed to and processed by PL/SQL engine.
- It seperate out the SQL statements and sends them individualy to the SQL statement executor.
- PL/SQL engine execute procedural statements but sends SQL statements to the SQL statement executor in the oracle server.

* PL/SQL Datatypes -

* Scalar -

1 Char (size) -

- This datatype is used to store string values of fixed length.
- the size in brackets determines the number of character the cell can hold.
- It can hold maximum 255 characters

e.g char (50)

2. varchar (size) / varchar 2 (size) -

- This datatype is used to store variable length alphanumeric data.

- The maximum this datatype can hold upto 4000 characters
  e.g varchar2(88)

3. Date -
- This datatype is used to represent date and time
- The default format is - DD-MM-YY
  e.g '20-May-2026

4. Number (P,S) -
- This datatype is used to store fixed or floating point numbers
- P is precision & S specifies scale.
- The maximum precision is 38 digits
  e.g Number (8,2), Number(8)

5. Long -
- This datatype is used to store variables length character string containing upto 2GB. only one long column allowed per table.

* RAW -
- This datatype is used to store Binary data such as Digitized, picture or Image maximum storage up to 255 Bytes.

* RAW LONG -
- This data is used to store Binary data such as Digitized, picture or Image maximum storage up to 2GB

* Types of large objects -
  1. BLOB - This datatypes used to store Binary character like songs, videos, images maximum size up to 4GB
  2. CLOB - This datatype used to store character value with size up to 4GB

* Reference variables -
- Reference variables directly reference specific database column or row
- reference variable assume datatype of associated column or row

1. %type data declaration Syntax - variable_name table(field)name %type.

e.g. Empid emp.empno%type

- Empid is variable
- emp table name
- empno is unknown datatype column

%type is datatype to hold a value.

★ Variables –

- Variables is nothing but a name given to storage area that our program can manipulate. information is transmitted between PL/SQL program and the database through variables.
- Each variable in PL/SQL has specific datatype, which determines the size & layout of the variables memory.

Syntax –

```
variable_name  datatype [NOT NULL := value];
```

Where,

variable_name is the name of the variable
datatype is valid PL/SQL datatype
NOTNULL is optional specification on the variable
value or DEFAULT value also an optional specification, where you can initialize the variable

- Each variable declaration is seperate statement & must be terminated by semicolon.

e.g. a NUMBER := 8;

★ Variable Scope in PL/SQL –

- PL/SQL allows the nesting of blocks i.e block may contain another inner block.
- if variable is declared within an inner block, it is not accessible to the outer block
- or if variable is declared & accessible to an outer blocks, it is also accessible to all nested inner blocks

1. Local variables – variables declared in an inner block & not accesible to outer Block.
2. Global variables – variables declared in the outermost block or package.

* Generating output - To display message to the user the SERVEROUTPUT should set to ON

e.g SET SERVEROUTPUT ON ;

- put the command at the beginning of the program, right before the declaration section.

* Displaying output -

- DBMS_OUTPUT - It is a package that includes number of procedure and functions that accumulate information in buffer so that it can be retrived later.
this function can also be used to display mosages to the user.

- PUT_LINE - It is the procedure to generate the output on the screen

e.g DBMS_OUTPUT.PUT_LINE (X);

or DBMS_OUTPUT.PUT_LINE ('The Addition is :-' || X);

* Accepting Input from user -

- To accept input from user you can use '&' operator. To run queries on SQL prompt use -

variable-name := & variable-name ;

e.g. A := &a
B := &b

* Constants -

- constant is a value used in PL/SQL Block that remains unchanged throughout the program.
- A constant holds a value that once declared, does not change in the program.
- A constant declaration specified its name, data-type & value & allocates storage for it.

Syntax -

constant-name CONSTANT datatype := value ;

Where, constant_name is the name of constant i.e similar to variable name.

- the word CONSTANT is keyword which does not change its value.
- value which must be assigned to a constant when it is declared.

e.g. PI CONSTANT NUMBER := 3.14

* Control Structure –

- Control Structure is used for decision making and changing flow of the program.

1. Conditional Control –

1. IF-THEN statement -

- An IF-THEN statement allows you to specify only one group of action to take.
- the simplest form of IF statement associated a condition with sequence of statement enclosed by keyword THEN and END IF
- if condition is true then statement get execute, if false then if statement does nothing.

Syntax -

```
IF <condition> then
    Statements;
END IF;
```

program -

```
1. declare
2. a number := 10;
3  b number := 20;
4  begin
5  if b>a then
6  dbms_output.put_line ("B is greater');
7  end if;
8  end;
```

2. IF-THEN-ELSE Statement –

- A sequence of IF-THEN statements can be followed by an optional sequence of ELSE statement, which executes when the condition is false.

- The IF clause check a condition; the THEN clause defines what to do if the condition is true; the ELSE clause defines what to do if the condition is false or null.

Syntax -

```
IF <condition> THEN
    Statements;
ELSE
    Statements;
END IF;
```

Program -

```
A number(3);
B number(3);
Begin
  A:=&A;
  B:=&B;
  If (A>B) Then
  dbms_output.put_line ('greater no is:-' ||A);
  else
  dbms_output.put_line ('greater no is:-'||B);
  end if;
  end;
```

3. IF-THEN-ELSEIF Statement -

- If we want a multiway branch, we can use IF-THEN-ELSEIF
- It allows you to choose between several alternatives
- an IF-THEN can be followed by an optional ELSIF else Statement
- The ELSIF clause lets you add additional conditions.

Syntax -

```
IF <condition> THEN
    Statements;
ELSIF <condition> THEN
    Statements;
    :
ELSIF <condition> THEN
    Statement;
ELSE
    END IF
```

2. Iterative control –

An iterative control statements are used when we want to repeat the execution of one or more statements for specified number of times.

- There are three types of loops in PL/SQL –

1. Loop –

- a simple loop is used when a set of statements is to be executed at least once before the loop terminates.
- An EXIT condition must be specified in the loop, when the EXIT condition is satisfied the process exits from loop

Syntax –

```
LOOP
    Statements;
    EXIT when <condition>;
    END LOOP
```

program –

```
1  declare
2  i number:=1;
3  begin
4  loop
5  dbms_output.put_line(i);
6  i:=i+1;
7  EXIT when i>5;
8  END loop;
9  end;
```

2. WHILE LOOP –

- the WHILE LOOP statement associates with sequence of statements before each iteration of the loop, the condition is evaluated.
- If the condition is true, the sequence of statements is executed, then control resumes at the top of the loop.
- If the condition is false or null, the loop is bypassed & control passes to the next statement

Syntax -

```
WHILE <condition> LOOP
    sequence of statements ;
END LOOP ;
```

program -

```
Declare
i number := 0
Begin
  WHILE i<5 LOOP
    i := i+1
    dbms_output.put_line (i);
  end loop ;
  end ;
```

3. FOR LOOP -

- a FOR LOOP is used to execute a set of statements for predetermined number of times.
- Operation occurs between the start and end integer value given

Syntax -

```
FOR counter IN val1..val2 LOOP
    statements ;
END LOOP ;
```

program -

```
declare
~~in~~ i number~~(10)~~ ;
i number := 5 ;
FOR I in 1..5 LOOP
  dbms_output.put_line (i) ;
END LOOP ;
END ;
```

3 Sequential Control –

1. GOTO Statement –

- The GO TO statement immediately transfer program control unconditionally to label i.e jump location.
- Go to statement can not branch into if statement, loop statement
- changes the flow of control within a block

Syntax –

```
Begin
  Statements;
GOTO Label_1;
  Statement-s
<< label_1 >> Statements;
END;
```

2. NULL Statement –

- NULL statement does nothing passes control to the next statement some languages refer to such an instruction as a no-operation.

* Exception Handling –

- Exception is nothing but an error
- when an error occurs exception is raised, normal execution is stopped and control transfers to exception handling part.
- Exception handlers are routines written to handle the exception.
- PL/SQL provides a feature to handle exceptions which occur in PL/SQL block known as exception Handling.
- using exception Handling we can test code and avoid it from exiting unexpectedly. When an exception occurs a message.

Syntax –

```
DECLARE
    Declaration section
BEGIN
    Executable section
EXCEPTION
WHEN ex_name1 THEN
    - error Handling statement
WHEN ex-name2 THEN
    - error Handling statement
END;
```

Types of exception —

1. predefined Exception –

- System exception are automatically raised by oracle, when program violates a RDBMS rules.
- the most common errors that can occur during the execution of PL/SQL
  i.e NO-DATA-FOUND, ZERO_DIVIDE and too many rows etc.

following are Named system exception –

1. ZERO_DIVIDE — It occur when a program attempts to divide a number by zero.
2. NO-DATA-FOUND – It is occur when select_into statement returns no rows.
3. TOO-MANY_ROWS – When we select or fetch more than one row into a record.
4. CURSOR_ALREADY-OPEN — when we open a cursor that is already open.

- Those system exception for which oracle does not provide a name is known as an unnamed system exception.

program –

Eg –

```
DECLARE
    a number;
BEGIN
    A:=10/0;
EXCEPTION
    WHEN ZERO_DIVIDE THEN
        DBMS_OUTPUT.PUT_LINE ("Divide by zero error");
END;
```

2. User defined Exception –

- A user defined exception has to be defined by programmer
- It is declared in the declaration section with their type as exception.
- There are methods to define user defined exception
  1. RAISE
  2. RAISE_APPLICATION_ERROR

- They must be raised explicitly using raise statement, unlike pre-defined exceptions that are raised implicitly.

Syntax -

```
DECLARE
BEGIN
  <exception_name> exception;
    <SQL sentenses>
  if <condition> Then
  RAISE <exception_name>
  END if ;
  EXCEPTION
  WHEN < exception_name> THEN
  - error Handling statements
  END ;
```

Program - To check whether salary is negative, if negative raise the exception.

```
DECLARE
    sal number (7,2);
    negative_sal EXCEPTION;
BEGIN
select salary into sal from emp where
                                empno=8888;
if sal<0 Then
   RAISE negative_sal ;
else
  update emp set salary=10000 where empno=8888
end if;
EXCEPTION
 WHEN NEGA- negative_sal Then
    Dbms_output.put_line ("Salary is negative");
END;
```

* Cursors —

- PL/SQL cursor is pointer that points the result set of the SQL Query Against database table
- Cursor is temperory work area which is used by SQL server used to sto store the result of query.
- It is used to store the data retrived from the database & manipulate this data.
- The data stored in the cursor is called as Active Data set
- a cursor can hold more than one row but process only one row at a time.
- cursor is used with select statement & store result.

Types of cursor —

1. Implicit cursors —

- Implicit cursors created for every query made in oracle like Delete, update, insert, select.
- They are also created when select statement that returns just one row is executed.
- When we write commands, we use implicit cursors
- They are System generated cursors.
- ~~Orad.~~ Oracle provides few attributes called as implicit cursors attributes to check the Status of DML operations.

1. % ROWCOUNT — count the row affected by DML operations
2. % FOUND - If SELECT or DML statement affects one or more rows it returns TRUE otherwise FALSE
3. % NOTFOUND - no rows found then returns TRUE else returns FALSE
4. % ISOPEN — True if cursor is open or false if cursor has not been opened or has been closed

e.g - count the no. of rows Deleted from table wing row count

```
begin
delete from Student where name = "PATIL";
dbms_output.put_line("Number of rows deleted:="||SQL%
                                                rowcount);
end;
```

2. Explicit cursors –

- Explicit cursors can be declared by programmer within PL/SQL
- They must be created when you are executing a SELECT statement that returns more that one row.
- Even through the cursor store multiple records, only one record can be processed at a time, which is called as current row.
- when you fetch row the current row position moves to next row
- cursor is first @declared then Open, once the cursor open fetch data from cursor, finaly close the cursor.
- Explicit cursor can be controlled using following three control Statement–

1 Declare cursors – Define name and structure of cursor

Syntax – cursor <cursor_name> IS select statement;

e.g – cursor c1 is select rname, marks from Student.

2. Open cursor – the open statement execute the query associated with the cursor, identifies the result set and the position the cursor before the first row.

Syntax – open <cursor-name>

e.g open c1;

3. Fetch @ata from cursor – Fetching the cursor involves accessing one row at a time. when data is fetched it is copied to the record or variables & logical pointer moves to the next row & it become the current row.

- If we are fetching cursors to a list of variables, the variables Should be listed in same order in the fetch statement as the columns present in the cursor

Syntax –

FETCH <cursor-name> into <variable-name>

e.g. fetch c1 into var1;

4. close a cursor – When the last row has been processed, the close statement disables the cursor

synta – CLOSE <cursor-name>;

General syntax –

```
declare
    variable declaration;
    cursor <cursor_name> Is select Statement;
    begin
        open <cursor_name>;
        fetch <cursor_name> into <variable_name>;
        close <cursor_name>
    end;
```

Program –

e.g.

```
declare
    emp_rec emp_tbl%.rowtype;
    cursor emp_cur Is select * from emp_tbl where salary>10;
Begin.
    Open emp_cur
    fetch emp_cur into emp_rec;
    dbms_output.put_line (emp_rec.first_name||" "||emp_rec.last_name);
    close emp_cur;
End;
```

* Cursor for loops –

- In most situations that require an explicit cursor, you can simplify by using cursor FOR loop instead of the OPEN, FETCH, & CLOSE statements
- A cursor FOR loop implicitly declares its loop index as a record that represents a row fetched from the database.
- next, it's opens a cursor, ~~rep~~ repeatedly fetches rows of values from the result set into fields in the record, then closes the cursor when all rows have been processed.

e.g.

```
Declare
    cursor c1 is select * from emp;
Begin
    FOR outvariable in c1
    LOOP
    EXIT WHEN c1%.Notfound;
    If outvariable.sal < 20000 THEN
```

```
dbms_output.put_line (outvariable.sal || " " || outvariable.lastname)
END if;
END LOOP;
END;
```

* parameterized cursor –

- PL/SQL parameterized cursor pass the parameters into a cursor & use them in to query.
- Default values is assigned to the cursor parameters & scope of the parameter is locally.
- PL/SQL parameterized cursor ~~defined~~ defines only datatype.

e.g. cursor displays employee information from Emp table whose emp_id is 10

```
Declare
  cursor c (n number) is select * from emp where emp_id = n;
  emp1 emp%rowtype;
Begin
    open c(10);
    for emp1 IN c(10) LOOP
    dbms_output.put_line('emp_id:' || Emp1.emp_id);
    dbms_output.put_line('emp_name:' || Emp1.emp_name);
    dbms_output.put_line('Salary' || Emp1.Salary);
  END LOOP;
  Close c;
END;
```

* Procedures –

- A stored procedure is named PL/SQL Block perform one or more specific task. This is similar to procedure in other programming language.
- store ~~proc~~ procedure offer advantages in the area of development, integrity, security, performance and memory allocation.

* Advantages –

1. Security - procedures offers more security.
2. productivity - Avoid redundant code for common procedures in multiple application.

3. memory savings - Requires only one copy of the code for multiple users.

4. performance - Precompiled code hence no compilation is required to execute a code.

5. Intigrity and accuracy - As procedure is needed to be tested only once hence guarantee of accurate result.

- Creating a Procedure -

- procedure consist header and body.
- The section before the keyword IS is called procedure's header
- header ~~consist~~ contains name of procedure & parameter passed to it
- Everything after the keyword Is is known as procedure's body.
- body contains declaration, execution, exception section same as PL/SQL

Syntax -

```
CREATE [OR REPLACE] PROCEDURE procedure_name
[ parameter_name [ IN | OUT | IN OUT ] Datatype [ IS | AS ]
variable declaration;
constant declaration;

Begin
   PL/SQL Subprograms / procedure body;
END <procedure_name>;
```

- procedure_name Specifies the name of procedure
- [OR REPLACE) option allows modifying an existing procedure
- procedure body contains the executable part
- The AS keyword is used instead of the IS keyword for creating Standalone procedure.
- the optional parameter list contains name, mode & type of parameter
- Passing parameters to procedure by following way -

1. IN - This parameter are used to send the value to procedure
2. OUT - This parameter are used to Get the value from procedure
3. IN OUT - This parameter are used to get and send value to procedure if IN & OUT datatype is same.

program - to insert the record into table through procedure.

```
create or replace procedure ins2 (no1 in number, name1 in
                                  varchar2, marks1 in number)
as
begin
  insert into student1 values (no1, name1, marks1);
  dbms_output.put_line ("record inserted");
  end ins21;
```

- Executing procedures - from sql prompt

  Syntax - execute procedure_name (argument value);

  e.g. exec ins21 (12, 'PATIL', 75);

- Deleting procedure - to delete procedure DROP PROCEDURE command is used.

  syntax - DROP PROCEDURE procedure_name;

  e.g DROP PROCEDURE ins2;

* Functions -

- A function is logically grouped set of SQL & PL/SQL statements that perform a specific task
- A function is named PL/SQL code block that accept value & return only one value.
- various advantages of functions are given below -

- Advantages -

1. code reusability feature can be used
2. It saves time & cost
3. Increase flexibility of the program
4. memory space required less
5. It can return value to the calling program

- Creating a function -

Syntax -

```
CREATE OR REPLACE FUNCTION function_name (Parameter in
                  Datatype, ...) return Datatype [IS/AS]
```

```
    Variable declaration;
    constant declaration;
Begin
    PL/SQL subprogram body;
    return value;
Exception
    Exception PL/SQL block
END <function-name>;
```

where,

function_name specifies name of the function, [OR REPLACE] option allows modifying an existing function

- argument - is the name of an argument to the function. parantheses can be omitted if no arguments are present.
- RETURN - clause specifies that datatype you are going to return from the function.

Datatype - is the datatype of argument.

IN - Specifies that value for the argument must be specified when calling the procedure or function.

OUT - Specifies that the procedure passes a value for this argument. back to its calling environment after execution.

program - find square of number.

```
create or replace function sqrt(no in number) return number AS
begin
  return no * no;
end sqrt;
```

- Executing & Deleting function -

- To use a function, you will have to call that function to performed defined task. when a program calls a function, program control is transferred to the called function.
- To call a function we simply need to pass the required parameters along with function name & if function returns value then we can store returned value.

calling function through program -

```
declare
 val1 number;
begin
val1 := &val1;
dbms_output.put_line("square" || sqrt(val1));
end;
```

- Deleting function —
  - to delete function, we use DROP FUNCTION command

Syntax - DROP FUNCTION function_name.

e.g DROP FUNCTION sqrt;

* Database Triggers —

- A database trigger is stored subprogram associated with database table, view, or event for instance, we can have oracle fire a trigger automatically before or after an INSERT, UPDATE OR DELETE statement affects a table.
- trigger is stored procedure which is called implicitly by oracle engine whenever an insert, update or delete statement is fired.
- It is procedure that are stored in the database & are implicitly run or fired, when a DML event occur in the database.

— Uses -

1. Data is generated on its own
2. Replicate table can be maintained
3. To enforce complex integrity constraints
4. To edit Data modification
5. To auto increment a field.

* Types of triggers —

1. Row level trigger — A row trigger is fired once for each row that is affected by DML command.
   - if an update command updates 100 rows then row level trigger is fired 100 times.
   - if a triggering statement affect no rows, row trigger is not run.

2. Statement level trigger – Statement trigger is fired only for once for a DML Statement irrespective of the number of rows affected by the Statement.

- It is default type of trigger.

3. Before trigger – while defining trigger you can specify whether the trigger is to be fired before the command i.e insert, delete, update is executed.

4. AFTER trigger – AFTER Trigger are fired after the triggering action is completed.

- for example if after trigger is associated with INSERT command then it is fired after the row is inserted into the table.

* Creating trigger –

Syntax –

```
CREATE [OR REPLACE] TRIGGER <trigger_name>
{BEFORE|AFTER}
{INSERT|DELETE|UPDATE [OF column..]}
ON TABLE_NAME
[FOR EACH ROW [WHEN condition]]
BEGIN
  sql statements;
END;
```

Where,

CREATE [OR REPLACE] TRIGGER trigger_name.

- This clause creates a trigger with the given name & overwrites an existing trigger with the same name.

- [BEFORE|AFTER]
  - This clause indicates at what time should trigger get fired

[INSERT | DELETE | UPDATE]

INSERT – If you want database to fire trigger whenever INSERT statement adds row to a table

DELETE – If you want database to fire trigger whenever DELETE

statement removes row

UPDATE – If you want the database to fire trigger whenever UPDATE statement changes values.

- [OF col_name]
- This clause is used when you want only when a specific column is updated.

[ON table_name]

This clause identified the name of the table to which the trigger is associated.

[FOR EACH ROW]

This clause used to determine whether a trigger must fire when each row get affected

- [WHEN (condition)
- This clause is valid only for row level triggers. the trigger is fired only for rows that satisfy the conditing specified.

program – display pass or fail after insert, update of new marks into student table using trigger.

```
create trigger studtrigger
after insert or update on student
for each row
begin
  if (: new-marks < 40) then
  dbms_output.put_line ("fail");
  else
  dbms_output.put_line ("pass");
end if;
end;
```

- Enable / Disable trigger -

- To enable / disable trigger ALTER TRIGGER comand is used

syntax -

```
ALTER TRIGGER trigger_name Enable/ Disable;
```

- when trigger is created, it is automatically enabled. To disable all triggers of the EMP table we use

```
Alter table EMP Disable all triggers
```

- Deleting a triggers

- To delete trigger, we use DROP TRIGGER command

syntax -

```
DROP TRIGGER trigger_name;
```

e.g. `DROP TRIGGER tr1;`

Unit - IV PL/SQL Programming

Question Bank

2 Marks

1. Draw PL/SQL Block structure. **(W-25)**
2. State various data types in PL/SQL. **(W-25)**
3. Enlist any four string functions in SQL. **(S-25)**
4. List the type of exception in PL/SQL. **(S-25)**
5. State any two advantages of PL/SQL. **(S-25)**

4 Marks

1. Differentiate between PL/SQL function and procedure. **(W-25)**
2. Write a PL/SQL program that asks the user to input two numbers and divide the first number by the second. Handle the predefined exception. **(W-25)**
3. Write PL/SQL program to print largest of three numbers **(W-25, S-25)**
4. Write PL/SQL procedure to print numbers between 20 to 30. **(W-25)**
5. Define cursor. Explain steps to use explicit cursor with example **(W-25)**
6. Define cursor. Write step by step syntax to declare open, fetch and close cursor in PL/SQL. **(S-25)**
7. Explain trigger with suitable example. **(S-25)**
8. Write a PL/SQL program to print odd numbers in 1 to 10 **(S-25)**