CH-1 INTRODUCTION TO PL/SQL

SQL is the natural language of the DBA but suffers from inherent disadvantages. They are

- It does not have procedural capabilities
- SQL statements are passed to the oracle engine one at a time. This decreases the processing speed in a multi-user environment
- SQL has no facility of programmed handling of error messages.

Advantages of PL/SQL

1. PL/SQL is a development tool that not only supports SQL data manipulation but also provides facilities for condition checking, branching and looping
2. PL/SQL sends the entire block of SQL code to oracle engine. As the entire block of SQL code is passed to Oracle engine at one time for execution, all the changes made to the data in the table are done or undone in one go.
3. PL/SQL facilitates the displaying of user-friendly messages when errors are encountered.
4. PL/SQL allows declaration and use of variables in blocks of code. The variables are used to store intermediate results of a query for later processing
5. PL/SQL all sorts of calculations can be done efficiently and quickly
6. Applications written in PL/SQL are portable to any computer hardware system and operating system.

The Generic PL/SQL block

A single PL/SQL code block consists of a set of SQL statements.This block has to be logically grouped together for the engine to recognize it as singular block of code.

A PL/SQL block has a definite structure, which can be divided into sections. The sections of PL/SQL block are

a. Declare section

b. Master begin…end section

DECLARE	Declarations of memory variables, constants, cursors
BEGIN	SQL executable statements PL/ SQL executable statements
EXCEPTION	Handles errors that arise during execution of code block between BEGIN and EXCEPTION
END;	End of PL/SQL block

The Declare Section

Code block starts with the declaration section, in which memory variables and other oracle objects are declared, and initialized if necessary. Once declared they can be used in SQL statements for data manipulation

The Begin Section

It consists of a set of SQL and PL/SQL statements which describe processes to be applied to table data. Actual data manipulation, retrieval, branching and looping constructs are specified in this section.

The Exception Section

This section deals with handling of errors that arise during the execution of data manipulation statement, which make up the PL/SQL code block. Errors can arise due to syntax, logic or validation rule violation

The End Section : This marks the end of a PL/SQL block

Structure of PL/SQL block

```
DECLARE
  <declarations section>
BEGIN
  <executable command(s)>
EXCEPTION
  <exception handling>
END;
```

Example:

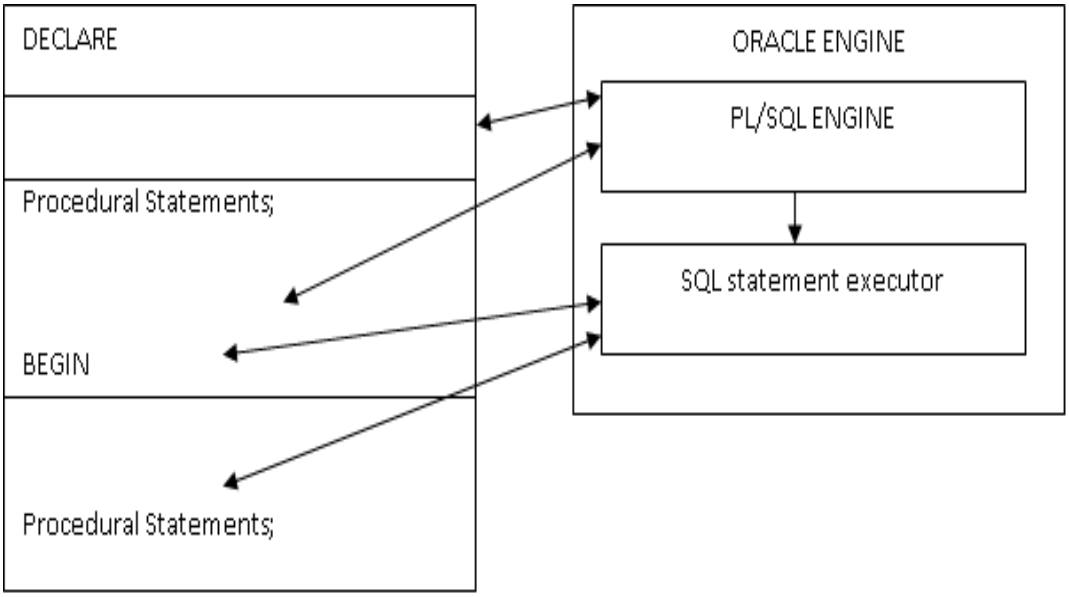
```
DECLARE
  message  varchar2(20):= 'Hello, World!';
BEGIN
  dbms_output.put_line(message);
END;
/
```

PL/SQL Execution Environment:

The PL/SQL engine accepts any valid PL/SQL block as input

PL/SQL engine resides in the Oracle engine. Oracle engine can process both SQL statements and PL/SQL blocks.These blocks are sent to the PL/SQL engine, where the procedural statements are executed. SQL statements are sent to the SQL executor in the Oracle engine.

The call to the oracle engine needs to be made only once to execute any number of SQL statements if these SQL statements are bundled inside a PL/SQL block



PL/SQL Character set:

The basic character set in PL/SQL includes

Upper case letters {A..Z},Lower case letters {a..z}

Numerals {0..9},Symbols ( ) + - * / < > = ! @ %

Words used in PL/SQL blocks are called lexical units. Blank spaces can freely be inserted between lexical units in a PL/SQL block.

A literal is a numeric or character string used to represent itself. A numeric literal can be a integer or float value. E.g. 5, -20, 29.6. A string literal is a set of two or more characters enclosed in single quotes.E.g. 'Hello', 'Hi'.A character literal is a single character enclosed in single quotes. E.g. 'Y', 'N', 'm'. Boolean literals or logical literals are predetermined constants. The values these literals can hold are TRUE, FALSE, NULL

PL/SQL Data types

The default data types that can be declared in PL/SQL are

1. Number – Numeric values, on which arithmetic operations are performed.
2. Character – Alphanumeric values that represent single characters or strings of characters.
3. Boolean – Logical values, on which logical operations are performed.
4. Date/time – Date and time.

The above data types can have NULL values.

5. **The % type attribute is used to declare variables based on the definitions of columns in a table. It helps to declare a variable or constant to have same data type as that of a column declared in a table.**

Variables

A variable in PL/SQL is a named variable. A variable name must begin with a character and can be followed by a maximum of 29 characters. Reserved words cannot be used as variable names unless enclosed within double quotes.

A blank space cannot be embedded in a variable name. variable names are not case sensitive. Variables are separated from each other by a comma.

Values can be assigned to variables in two ways:

1. Using the assignment operator :=

2. Selecting or fetching table data values into variables

Constants: A constant is declared in the same way as how a variable is declared. The only difference is that we use a keyword constant and a value is assigned to it immediately. Thereafter no further assignments can be made.

Logical Comparisons

Comparisons in PL/SQL can be made using Boolean expressions. A Boolean expression consists of variables separated by relational operators{<,>,<=,>=,<>}.

A logical expression is a combination of two or more Boolean expressions separated by logical operators{AND,OR, NOT}.

A Boolean expression always evaluates to TRUE, FALSE or NULL.

Displaying user messages on the VDU:

DBMS_OUTPUT is a package that includes a number of functions and procedures that accumulate information in a buffer so that it can be retrieved later. These functions can also be used to display a user message on the screen.

PUT_LINE puts a piece of information in package buffer followed by end of the line marker. It expects a single parameter of character data type.

In order to display a message, SERVEROUTPUT should be set ON.

It is a SQL*Plus environment parameter that displays the information passed as parameter to the PUT_LINE function.

Syntax: **SET SERVEROUTPUT [ON/OFF]**

Comments:

A comment in PL/SQL can be written in two ways:

1. It can begin with a double hyphen(--). Here, the entire line can be treated as a comment.

2. A group of lines can be enclosed between /* and */. All the lines will be ignored for processing

Example:

a. --Single line comment

b. /* This is
a comment*/

Example of PL/SQL code

```
SET SERVEROUTPUT ON
DECLARE
  -- constant declaration
  pi constant number := 3.141592654;
  -- other declarations
  radius number(5,2);
  circumference number(7, 2);
  area number (10, 2);
BEGIN
  -- processing
  radius := 9.5;
  circumference := 2.0 * pi * radius;
  area := pi * radius * radius;
  -- output
  dbms_output.put_line('Radius: ' || radius);
  dbms_output.put_line('Circumference: ' || circumference);
  dbms_output.put_line('Area: ' || area);
END;
SET SERVEROUTPUT OFF
/
```

Control Structures

The flow of control statements can be classified as

•Conditional control

•Iterative control

Conditional control

PL/SQL allows the use of an IF statement to control the execution of a block of code.

In PL/SQL, IF-THEN-ELSE-END structure allows to check certain conditions under which a specific block of code should be executed.

The different forms of IF statements are as follows:

1) IF condition THEN

<action>

ELSIF<condition> THEN

<action>

Else

<action>

END IF;

2) IF condition THEN

statement 1;

ELSE

statement 2;

END IF;

3)IF condition 1 THEN

statement 1;

statement 2;

ELSIF condtion2 THEN

statement 3;

ELSE

statement 4;

END IF;

Example:

```
set serveroutput on
declare
 x number(2);
```

```
 y number(2);
begin
 x:=&x;
 y:=&y;
 if x>y then
   dbms_output.put_line (x || 'is bigger');
 else
   dbms_output.put_line (y || 'is bigger');
 end if;
end;
```

Output:

```
Enter value for x: 35
Enter value for y: 25
35is bigger
PL/SQL procedure successfully completed.
```

Iterative control

Iterative control indicates the ability to repeat sections of code. A loop marks the sequence of statements that has to be repeated. 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:

- Simple Loop
- While loop
- For loop

Simple 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, otherwise the loop will get into an infinite number of iterations. When the EXIT condition is satisfied the process exits from the loop.

Syntax of Simple Loop:

LOOP

<sequence of statements>;

[EXIT WHEN condition;]

END LOOP;

Example**:**

```
set serveroutput on
declare
        i number:=0;
begin
        dbms_output.put_line('Even numbers from 1 to 10 are:');
        loop
        i:= i+2;
        dbms_output.put_line (i);
        exit when i>=10;
        end loop;
end;
```

Output:

```
Even numbers from 1 to 10 are:
2
4
6
8
10
PL/SQL procedure successfully completed.
```

While loop

A WHILE LOOP is used when a set of statements has to be executed as long as a condition is true. The condition is evaluated at the beginning of each iteration. The iteration continues until the condition becomes false.

Syntax of WHILE LOOP is:

WHILE <condition>

LOOP

<Action>

END LOOP;

Example:

```
set serveroutput on
declare
        pi constant number(5,2):=3.14;
        r number(2);
        c number(5,2);
begin
        r:=3;
        while r<=7
        loop
        c:=2*pi*r;
        insert into circle values(r,c);
        r:=r+1;
        end loop;
end;
```

Output:

PL/SQL procedure successfully completed.

SQL> select * from circle;

RADIUS	CIRCUM
3	18.84
4	25.12
5	31.4
6	37.68
7	43.96

FOR Loop

A FOR LOOP is used to execute a set of statements for a predetermined number of times. Iteration occurs between the start and end integer values given. The counter is always incremented by 1. The loop exits when the counter reachs the value of the end integer.

Syntax of FOR LOOP is:

FOR variable IN [REVERSE] start...end

LOOP

<action>

END LOOP;

Example:

```
set serveroutput on
declare
        i number;
        n number;
        f number:=1;
begin
        n:=&n;
        for i in 1..n
        loop
        f:=f*i;
        end loop;
        dbms_output.put_line('Factorial is '|| f);
end;
```

Output:

```
Enter value for n: 5
Factorial is 120
PL/SQL procedure successfully completed.
```

CH-2 PL/SQL Transactions

A series of one or more SQL statements that are logically related or a series of operations performed on the Oracle table data is called as a **transaction**.

Oracle treats changes made to data in a table as a two step process.

- First, the requested changes are done. In order to make these changes permanent, a **COMMIT** statement is issued at the SQL prompt.
- **A ROLLBACK** statement issued at the SQL prompt can be used to undo a part of or the entire transaction.

A transaction begins with the first executable statement after a commit, rollback or a connection made to the Oracle engine.

A transaction is a group of events that occur between any of the following events:

•Connecting to Oracle

•Disconnecting from Oracle

•Committing changes to database table

•Rollback

Closing Transactions – A transaction can be closed using either a commit or rollback statement. By using these statements, table data can be changes or all the changes made to the table data can be undone.

COMMIT – A commit ends the current transaction and makes permanent any changes made to the table data during a transaction. Syntax: **commit;**

SAVEPOINT – It marks and saves the current point in the processing of a transaction. When a savepoint is used with a rollback statement, parts of the transaction can be undone. An active savepoint is one that is specified since the last COMMIT or ROLLBACK.

Syntax: **savepoint <savepointname>;**

ROLLBACK – A rollback does exactly the opposite of COMMIT. It ends the transaction but undoes any changes made during the transaction. Rollback can be fired at the SQL prompt with or without savepoint clause.

Syntax: **rollback [work] [to [savepoint]<savepointname>];**

CURSORS

The Oracle engine uses a work area for its internal processing in order to execute an SQL statement. This work area is private to SQL operations and is called a **Cursor.**

The data that is stored in the cursor is called the **Active Data Set**. The size of the cursor in the memory is the size required to hold the number of rows in the active data set. Oracle has a predefined area in the main memory set aside, within which cursors are opened.

A cursor contains information on a select statement and the rows of data accessed by it. A cursor can hold more than one row, but can process only one row at a time.

Cursors are classified depending on the circumstances under which they are opened.Both implicit and explicit cursors have the same functionality, but they differ in the way they are accessed

Implicit cursors

If the oracle engine opened a cursor for its internal processing, it is known as **implicit cursor**. These are created by default when DML statements like, INSERT, UPDATE, and DELETE statements are executed. They are also created when a SELECT statement that returns just one row is executed.

Implicit Cursor Attributes:

Attributes	Return Value	Example
%FOUND	The return value is TRUE, if the DML statements like INSERT, DELETE and UPDATE affect at least one row and if SELECT ….INTO statement return at least one row.	SQL%FOUND
	The return value is FALSE, if DML statements like INSERT, DELETE and UPDATE do not affect row and if SELECT….INTO statement do not return a row.	
%NOTFOUND	The return value is FALSE, if DML statements like INSERT, DELETE and UPDATE at least one row and if SELECT ….INTO statement return at least one row.	SQL%NOTFOUND
	The return value is TRUE, if a DML statement like INSERT, DELETE and UPDATE do not affect even one row and if SELECT ….INTO statement does not return a row.	
%ROWCOUNT	Return the number of rows affected by the DML operations INSERT, DELETE, UPDATE, SELECT	SQL%ROWCOUNT

Example: Consider the PL/SQL Block that uses implicit cursor attributes as shown below:

```
/*increase the salary of all employees by 1000*/
DECLARE  var_rows number(5);
BEGIN
  UPDATE employ
  SET salary = salary + 1000;
  IF SQL%NOTFOUND THEN
    dbms_output.put_line('None of the salaries where updated');
  ELSIF SQL%FOUND THEN
    var_rows := SQL%ROWCOUNT;
    dbms_output.put_line('Salaries for ' || var_rows || 'employees are updated');
  END IF;
END;
```

Output:

Salaries for 7employees are updated

PL/SQL procedure successfully completed.

Explicit cursors

A cursor can be opened They must be created when you are executing a SELECT statement that returns more than one row. Even though the cursor stores multiple records, only one record can be processed at a time, which is called as **current row**.

When you fetch a row the current row position moves to next row. When individual records in a table have to be processed inside a PL/SQL block explicit cursor is used. This cursor will be declared and mapped to an SQL query in the declare section of the PL/SQL block and used with the executable section of PL/SQL block.

The steps involved in using explicit cursor and manipulating data are

- Declare a cursor in the declare section of PL/SQL block
- Open the cursor

•Fetch the data from the cursor one row at a time into the memory variables

•Process the data that was fetched

•Exit from the loop after the processing is complete

•Close the cursor

Explicit cursor attributes

Attributes	Return Value	Example
%ISOPEN	Evaluates to TRUE, if an explicit cursor is open. It returns FALSE if the cursor is closed	Cursor_name%ISOPEN
%FOUND	The return value is TRUE, if the DML statements like INSERT, DELETE and UPDATE affect at least one row and if SELECT ….INTO statement return at least one row.	Cursor_name%FOUND
	The return value is FALSE, if DML statements like INSERT, DELETE and UPDATE do not affect row and if SELECT….INTO statement do not return a row.	
%NOTFOUND	The return value is FALSE, if DML statements like INSERT, DELETE and UPDATE at least one row and if SELECT ….INTO statement return at least one row.	Cursor_name%NOTFOUND
	The return value is TRUE, if a DML statement like INSERT, DELETE and UPDATE do not affect even one row and if SELECT ….INTO statement does not return a row.	
%ROWCOUNT	Return the number of rows affected by the DML operations INSERT, DELETE, UPDATE, SELECT	Cursor_name%ROWCOUNT

Functionality of Open, Fetch and Close commands

Declaring a cursor – A cursor is declared in the declare section of the PL/SQL block. This is done to create an active data set. The cursor name is used to reference the active data set that is held within the cursor

Syntax:

CURSOR cursor-name IS SELECT statement;

Example:

```
DECLARE
  CURSOR emp_cur IS
  SELECT *
  FROM employ
  WHERE salary > 5000;
```

In the above example we are creating a cursor 'emp_cur' on a query which returns the records of all the employees with salary greater than 5000.

Opening a cursor – Opening a cursor executes a query and creates an active data set that contains all rows which meet the search criteria of the query. An open statement retrieves the records from the table and places the records in the cursor

Syntax:

OPEN cursor_name;

Fetching records from the cursor – The fetch statement retrieves the rows from the active data set into the memory variables declared in the PL/SQL block one row at a time. Each time a fetch statement is executed, the cursor pointer is advances to the next row in the active data set.

Syntax:

FETCH cursor-name INTO variable list;

Closing a cursor – The close statement disables the cursor and the active data set becomes undefined. This will release the memory occupied by the cursor and the active data set.

Syntax:

CLOSE cursor-name;

General Form of using an explicit cursor is:

```
DECLARE
   variables;
   records;
```

```
   create a cursor;
 BEGIN
  OPEN cursor;
  FETCH cursor;
    process the records;
  CLOSE cursor;
 END;
```

Example:

```
/*Display employee name and salary whose salary is more than 6000*/
SET SERVEROUTPUT ON
 DECLARE
   emp_rec employ%rowtype;
   CURSOR emp_cur IS
   SELECT *
   FROM employ
   WHERE sal > 6000;
 BEGIN
   OPEN emp_cur;
   FETCH emp_cur INTO emp_rec;
     dbms_output.put_line (emp_rec.ename || '  ' || emp_rec.sal);
  CLOSE emp_cur;
 END;
/
```

Output:

```
SQL> @EXPCUR.SQL
Gita  7000
PL/SQL procedure successfully completed.
```

Cursor FOR LOOP

When using FOR LOOP you need not declare a record or variables to store the cursor values, need not open, fetch and close the cursor. These functions are accomplished by the FOR LOOP automatically.

General Syntax for using FOR LOOP:

```
FOR record_name IN cursor_name
LOOP
   process the row...
END LOOP;
```

Example:

```
/*To display all employees whose salary is greater than 9000*/
SET SERVEROUTPUT ON
 DECLARE
   CURSOR emp_cur IS
   SELECT *
   FROM employ
   WHERE sal > 9000;
 BEGIN
   for emp_rec in emp_cur
   loop
    dbms_output.put_line (emp_rec.ename || '  ' || emp_rec.sal);
   end loop;
 END;
/
```

Output:

```
Rani  10000
Ranbir  11000
PL/SQL procedure successfully completed.
```

Parameterized Cursors

Oracle permits the data that is retrieved from the table be allowed to change according to need. The contents of parameterized cursor will constantly change depending on the value that is passed as parameter. Since the cursor accepts user-defined values into its parameters, thus changing the result set extracted, it is called parameterized cursor.

Declaring Parameterized cursor:

Syntax: CURSOR cursor_name IS <SELECT statement>;

Opening a Parameterized Cursor:

Syntax: OPEN cursor_name(value/variable/expression);

Example:/*DISPLAY DETAILS OF STUDENT WITH ROLLNUMBER */

```
SET SERVEROUTPUT ON
DECLARE
        cursor c(no number) is select * from student
        where rollno = no;
        tmp student%rowtype;
BEGIN
        FOR tmp IN c(14)
        LOOP
        dbms_output.put_line('ROLLNO:       '||tmp.rollno);
        dbms_output.put_line('STUDENT_Name:  '||tmp.name);
        dbms_output.put_line('TOTAL:        '||tmp.total);
        dbms_output.put_line('COURSE:       '||tmp.course);
        END Loop;
END;
/
```

Output:

```
ROLLNO:       14
STUDENT_Name:  Gita
TOTAL:        145
COURSE:       bca
PL/SQL procedure successfully completed
```

CH-3 PL/SQL Database Objects

PROCEDURES AND FUNCTIONS

Procedures or Functions is a logically grouped set of SQL and PL/SQL statements that perform a specific task.

A stored procedure or function is a named PL/SQL code block that has been compiled and stored in one of the oracle engine's system tables .

Procedure and functions consists of

1. A declarative part
2. An Executable part
3. An exceptional –handling part (optional)

Oracle engine performs the following steps automatically while creating a procedure or function

- **Compiles the procedure or function**
- **Stores the procedure or function in the database**

The oracle engine compiles the PL/SQL block. If errors occur during the compilation of procedure or function, an invalid procedure or function gets created. The oracle engine displays the message after creation of that the procedure or function was created with compilation errors. The compilation process does not display any errors. These errors can be viewed by giving the following statement at the SQL prompt

SQL> SELECT * FROM USER_ERRORS;

Advantages of using procedure or function

The following are the advantages of using functions or procedures

a. **Security** : Stored procedures and functions can help to enforce data security.

For example, we can give permission to a stored procedures or function to query a table and grant permission to the user to use the stored procedures or function, rather than the table itself.

b. **Performance**: it improves database performance in the following ways:

- Amount of information sent over network is less
- No compilation step to execute the code

c. **Memory allocation**: the amount of memory used reduces as stored procedures and functions have shared memory capability. Only one copy of stored procedure or function needs to be loaded for execution by multiple users.

d. **Productivity** – redundant coding can be avoided thereby increasing the productivity

e. **Integrity**: Stored procedures and functions need to be tested only once to guarantee that it returns accurate result.

Procedures:

A stored procedure or in simple a proc is a named PL/SQL block which performs one or more specific task. This is similar to a procedure in other programming languages. A procedure has a **header and a body.**

The header consists of the name of the procedure and the parameters or variables passed to the procedure. The body consists or declaration section, execution section and exception section similar to a general PL/SQL Block. A procedure is similar to an anonymous PL/SQL Block but it is named for repeated usage.

General Syntax to create a procedure is:

```
CREATE OR REPLACE PROCEDURE procedure-name
(<Argument> {IN, OUT, IN OUT}<data type>...)
IS
  Declaration section
BEGIN
  Execution section
EXCEPTION
  Exception section
END;
```

We can pass parameters to procedures in three ways.

1. IN – indicates that the parameter will accept a value from the user
2. OUT – indicates that the parameter will return a value to the user
3. IN OUT - indicates that the parameter will either accept a value from the user or return a value to the user

Example:

/*procedure */

```
CREATE or REPLACE PROCEDURE pro1(no in number,temp out student%rowtype)
IS
BEGIN
SELECT * INTO temp FROM student WHERE rollno = no;
END;
/
```

SQL> @proc1;

Procedure created.

SQL> ed sqlproc.sql

/*PL-SQL code for implementing procedure*/

```
SET SERVEROUTPUT ON
DECLARE
        temp student%rowtype;
        no number :=&no;
BEGIN
        pro1(no,temp);
        dbms_output.put_line(temp.rollno||'    '|| temp.name||'  '||temp.total);
END;
/
```

Output:

Enter value for no: 12

12 Sriram 235

PL/SQL procedure successfully completed.

Functions: A function is a named PL/SQL Block which is similar to a procedure. The major difference between a procedure and a function is, a function must always return a value, but a procedure may or may not return a value. Here, return type in the header section defines the return type of the function. The return data type can be any of the oracle data type like varchar, number etc.

Syntax for creating a function:

CREATE OR REPLACE FUNCTION <function_name>

(<Argument> {IN }<data type>...)

RETURN return_datatype;

IS

Declaration_section

BEGIN

Execution_section

Return return_variable;

EXCEPTION

exception section

Return return_variable;

END;

Example:

```
SQL> ED STUDFUN.SQL
/*create a function*/
CREATE or REPLACE FUNCTION STUDFUN(no in number) RETURN varchar2
IS
name varchar2(20);
BEGIN
select name into name from student where rollno = no;
return name;
```

```
END;
SQL> @STUDFUN.SQL;
Function created.
SQL> ed plfun.sql

/*PL-SQL block to call the function*/
SET SERVEROUTPUT ON
DECLARE
        no number :=&no;
        name varchar2(20);
BEGIN
        name := studfun(no);
        dbms_output.put_line('Name:'||'  '||name);
END;
```

Output:

```
Enter value for no: 12
old   2:       no number :=&no;
new   2:       no number :=12;
Name:   Sriram

PL/SQL procedure successfully completed.
```

Procedures versus Functions:

The differences between a procedure and a function are listed below:

•A function must return a value back to the caller. A function can return only one value to the called PL/SQL block

•By defining multiple OUT parameters in a procedure, multiple values can be passed to the caller. The OUT variable is global in nature. Hence, it can be accessible by any PL/SQL code block.

Deleting a stored procedure or function

DROP PROCEDURE <PROCEDURENAME?

ORACLE PACKAGES

A package is an Oracle object, which holds other objects within it. PL/SQL packages are schema objects that groups logically related **procedures, functions, constants, variables and cursor**s. It is a way of creating generic, encapsulated and reusable code. A package once written and debugged is compiled and stored in Oracle's system tables held in Oracle database. Packages contain PL/SQL blocks of code, which have been written to perform some process entirely on their own. These PL/SQL blocks do not require any input from other PL/SQL blocks of code.

Components of an Oracle Package:

A package has usually two components – **specification and a body**.

Package Specification:

The specification is the interface to the package. It just DECLARES the types, variables, constants, exceptions, cursors, and subprograms that can be referenced from outside the package. In other words, it contains all information about the content of the package, but excludes the code for the subprograms. All objects placed in the specification are called public objects. Any subprogram not in the package specification but coded in the package body is called a private object.

Package specification contains

- Name of the package
- Names of the data types of any arguments
- This declaration is local to the database and global to the package

The syntax for package specification is as follows:

CREATE PACKAGE <package-specification-name> IS

<function declaration>

<procedure declaration >

END<package-specification-name>;

Package Body:

The body of a package contains the definition of the public objects that are declared in the specification. The package body has the codes for various methods declared in the package The CREATE PACKAGE BODY Statement is used for creating the package body. The syntax for creating package body is as follows:

CREATE PACKAGE BODY <package-name> AS

<function definition >

<procedure definition>

END<package-name>;

Advantages of Packages

Packages offer the following advantages

- Packages enable organizing applications as efficient modules
- It allows granting of privileges efficiently
- They enable overloading of procedures and functions, if required
- They promote reusability of code
- They improve performance by loading multiple objects into the memory at once.

Creating a Package:

The first step in creating a package is to create its specification. The specification declares the objects that are contained in the body of the package. A package can include functions and procedures. After the specification is created, the body of the package needs to be created. The body of the package is a collection of detailed definitions of the objects that were declared in the specification.

Example:

Display the student name, course and total by implementing package

/*Creating package specification*/

SQL> ed sampkg

```
CREATE or REPLACE PACKAGE pkg1
AS
PROCEDURE janpro1
        (no in number);
FUNCTION janfun1
        (no in number)
        RETURN number;
END pkg1;
/
SQL> @sampkg;
Package created.

/*Creating body of package*/
SQL> ed pkgbody
CREATE or REPLACE PACKAGE BODY pkg1
AS
        FUNCTION janfun1(no in number) return number
        IS
        tot number;
        BEGIN
                select total into tot from student where rollno = no;
        return tot;
        END;

        PROCEDURE janpro1(no in number)
        IS
        temp1 student.name%type;
        temp2 student.course%type;
```

```
BEGIN
        select name,course into temp1, temp2 from student where rollno = no;
        dbms_output.put_line('Student Name:' || temp1);
        dbms_output.put_line('Course Name:' || temp2);
END;
```

ERROR HANDLING IN PL/SQL

An exception handler is nothing but a code block in memory that will attempt to resolve the current exception condition. The oracle engine can recognize every exception condition that occurs in the memory. To handle very common and repetitive exception conditions, the oracle engine uses **Named Exception Handlers.**

PL/SQL Errors are pre-defined and are automatically raised by Oracle whenever an error is encountered. Each error is assigned a unique number and a message. In PL/SQL, a warning or error condition is called an **exception**. Exceptions can be internally defined (by the run-time system) or user defined.

Examples of internally defined exceptions include division by zero and out of memory. Some common internal exceptions have predefined names, such as ZERO_DIVIDE and STORAGE_ERROR. The other internal exceptions can be given names.

When an error occurs, an exception is raised. That is, normal execution stops and control transfers to the exception-handling part of your PL/SQL block or subprogram. Internal exceptions are raised implicitly (automatically) by the run-time system. User-defined exceptions must be raised explicitly by RAISE statements, which can also raise predefined exceptions. To handle raised exceptions, you write separate routines called **exception handlers**.

Syntax:

EXCEPTION

WHEN<Exception Name> THEN

<User Defined Action to be carried out>

Oracle's Named Exception Handlers

The Oracle engine has a set of predefined Oracle error handlers called Named Exceptions. These error handlers are referenced by their names. Some of the pre-defined Oracle named exception handlers are listed below

DUP_VAL_ON_INDEX	Raised when an insert or update attempts to create two rows with duplicate values in columns constrained by unique index
LOGIN_DENIED	Raised when an invalid user name/password is used to log onto Oracle
NO_DATA_FOUND	Raised when a select statement returns zero rows
NOT_LOGGED_ON	Raised when PL/SQL issues an Oracle call without being logged onto Oracle
PROGRAM_ERROR	Raised when PL/SQL has an internal problem
TOO_MANY_ROWS	Raised when a select statement returns more than one row
VALUE_ERROR	Raised when the data type or data size is invalid
OTHERS	Stands for all other exceptions not explicitly named

User-Named Exception Handlers

This technique is used to bind a numbered exception handler to a name using **Pragma Exception_init().** This binding of a numbered exception handler, to a name (String) is done in the declare section of the PL/SQL block. The Pragma action word is a call to a pre-compiler, which immediately binds the numbered exception handler to the name when encountered. The function Exception_init() takes two parameters – the first is the user defined exception name and the second is the Oracle engine's exception number. These lines will be included in the declare section of the PL/SQL block.

Syntax:

DECLARE

<Exception Name> EXCEPTION

PRAGMA EXCEPTION_INIT(<Exception Name>,<Error Code>);

BEGIN

.....

EXCEPTION

WHEN<Exception Name> THEN

<Action>

END;

User Defined Exception Handlers

User Defined Exception Handlers for Business Rule Validation

In commercial applications, data that is being manipulated needs to be validated against business rules. If the data violates a business rule, the entire record must be rejected. To trap business rules being violated the technique of raising user-defined exceptions and then handling them is used.

User defined error conditions are declared in the DECLARE section of PL/SQL block. In the executable part, a check for the condition is made. If the condition exists, the call to the user defined exception is made using RAISE statement. The exception once raised is then handled in the exception handling section of the PL/SQL code block.

Syntax:

DECLARE

<<Exception Name> Exception;

BEGIN

SQL statements;

IF <condition>THEN

RAISE <Exception Name>;

END IF;

EXCEPTION

WHEN <Exception Name> THEN {user-defined action}

END

Example:

SET SERVEROUTPUT ON

DECLARE

```
        myex EXCEPTION;
        i NUMBER;
BEGIN
        FOR i IN (SELECT * FROM student)
         LOOP
                IF i.rollno = 15 THEN
                        RAISE myex;
                END IF;
        END LOOP;
EXCEPTION
        WHEN myex THEN
                dbms_output.put_line('ROLLNO is Filled Already');
END;
/
```