- A procedure is a logically grouped set of SQL and PL/SQL statements that perform a specific task.
- They are the named PL/SQL block that can be compiled and stored in one of the Oracle engine's system tables.

Procedures and Functions are made up of:

- **A declarative part:** May contain the declaration of cursor, variables, exceptions or any other sub programs. These objects are local to the procedure or function.
- **An executable part:** Contains statements that assign values, control execution and manipulate data.
- **An optional exception-handling part:** Deals with exceptions that may be raised during execution of code.

Procedures and Functions are stored in Oracle database but before storing they are compiled and parsed by Oracle engine.

Where do stored procedure and functions reside?

- Procedure and function is stored in the oracle database. They can be invoked or called by any PL/SQL block.
- Before a procedure or function is stored, the oracle engine parses and compiles the procedure or function.

The following steps are performed automatically by oracle engine while creating procedure.

1. Compiles the procedure and functions.
2. Stores the procedure or functions in the database.

- The oracle engine complies the pl/sql code block. If **error occurs** during the **compilation of the procedure and function**, an invalid procedure and function get created.

The oracle engine displays a message after creation that the procedure or function was created with **compilation error.**

- For displaying error using select query
  SELECT * FROM USER_ERRORS;
- When the procedure and function is invoked, the oracle engine loads the compiled procedure and function in memory area is called the **SYSTEM GLOBAL AREA(SGA).** This allows the code to be executed quickly.

- **How does the oracle engine execute procedures/functions?**
  - Verifies user access
  - Verifies procedure or function validity.
  - Execute the procedure or function.
- To check status of procedure and function is shown by use of select statement as follows:
  - SELECT object_name, object_type, status FROM USER_OBJECTS WHERE object_type = 'PROCEDURE';
  - SELECT object_name, object_type, status FROM USER_OBJECTS WHERE object_type = 'FUNCTION';

If status is valid, will a procedure and function be executed. Once found valid, the oracle engine loads a procedure or function into memory.

- **Advantages Of Using A Procedure or Function**
  - **Security :** Store procedure and functions can help enforce data security. for eg by giving permission to a procedure and function with grant option.
  - **Performance :** it improve the database performance. When procedure and function is present in the shared pool of the SGA retrieval from disk is not required every time different user call the procedure or function.
  - **Memory Allocation :** the amount of memory used reduces as stored procedure or function have shared memory capabilities. Only one copy of procedure needs to be loaded for execution by multiple users.
  - **Productivity :** by writing procedure and function redundant coding can be avoided, increasing productivity.
  - **Integrity :** a procedure and function needs to be tested only once to guarantee that it returns an accurate result. The oracle engine has high level of in-built security and integrity of procedure and function.

- **Procedures Vs Functions**
  - A function must return a value back to the caller.
  - By defining multiple OUT parameters in a procedure, multiple values can be Passed to the caller.

❖ **Creating stored procedure**

Syntax : **CREATE [OR REPLACE] PROCEDURE procedure_name**
[(argument [IN | OUT | IN OUT] data type [, ...])]
{IS | AS}
<DECLARATION SECTION>
BEGIN
< procedure_body >
EXCEPTION
<exception handler>]
END [procedure name];

Keywords and parameters

REPLACE	Recreates the procedure if already exists. This option is used to change the definition of an existing procedure without dropping, recreating.
PROCEDURE	Is the name of procedure to be created.
ARGUMNET	is the name of argument to the procedure.
IN	Indicates that the parameter will **accept** a value from the user.
OUT	Indicates that the parameter will **return** a value to the user.
INOUT	Indicates that the parameter will either accept a value from the user or return a value to the user.
DATA TYPE	Is the data type of an argument. It supports any data type supported by PL/SQL

Example :

```
CREATE OR REPLACE PROCEDURE sample
AS
BEGIN
        dbms_output.put_line('Hello World!');
END;/
```

OUTPUT:
Procedure created.

❖ **Executing a Standalone Procedure**

A standalone procedure can be called in two ways:

- Using the EXECUTE keyword
- Calling the name of the procedure from a PL/SQL block

➢ The above procedure named 'sample' can be called with the EXECUTE keyword as:

```
EXECUTE sample;
```

The above call would display:
Hello World
PL/SQL procedure successfully completed.

➢ The procedure can also be called from another PL/SQL block:

```
BEGIN
        sample;
END;
/
```

The above call would display:

Hello World
PL/SQL procedure successfully completed.

❖ **IN & OUT Mode Example 1**

This program **finds the minimum of two values**, here procedure takes two numbers using IN mode and returns their minimum using OUT paramters.

```
SQL> create or replace PROCEDURE findMin(x IN number, y IN number, z OUT number) IS
    BEGIN
    IF x < y THEN
    z:= x;
    ELSE
    z:= y;
    END IF;
    END;
   /

Procedure created.

SQL>  DECLARE
 2   a number;
 3   b number;
 4   c number;
 5    BEGIN
 6    a:= 23;
 7    b:= 45;
 8    findMin(a, b, c);
 9    dbms_output.put_line(' Minimum of (23, 45) : ' || c);
10    END;
11  /
Minimum of (23, 45) : 23

PL/SQL procedure successfully completed.
```

❖ **Methods for Passing Parameters**

Actual parameters could be passed in three ways:

- Positional notation
- Named notation
- Mixed notation
- **POSITIONAL NOTATION**
  In positional notation, you can call the procedure as:
  Example : findMin(a, b, c, d);
- **NAMED NOTATION**

In named notation, the actual parameter is associated with the formal parameter using the arrow symbol (=>). So the procedure call would look like:
Example : findMin(x=>a, y=>b, z=>c, m=>d);

- **MIXED NOTATION**

In mixed notation, you can mix both notations in procedure call; however, the **positional notation should precede the named notation.**
The following call is legal:

```
findMin(a, b, c, m=>d);
```

But this is not legal:

```
findMin(x=>a, b, c, d);
```

❖ **Deleting a Standalone Procedure**

A standalone procedure is deleted with the DROP PROCEDURE statement. Syntax for deleting a procedure is:

```
DROP PROCEDURE procedure-name;
```

So you can drop greetings procedure by using the following statement:

```
BEGIN
    DROP PROCEDURE greetings;
END;/
```

❖ **PL/SQL Functions**

A PL/SQL function is same as a procedure except that it returns a value.

Creating a Function

A standalone function is created using the CREATE FUNCTION statement.

```
CREATE [OR REPLACE] FUNCTION function_name
    [argument IN data type [, ...])]
RETURN return_datatype
{IS | AS}
<DECLARATION SECTION>
BEGIN
    < function_body >
[EXCEPTION
    <exception code>]
END [function name];
```

Keywords and parameters

REPLACE	Recreates the procedure if already exists. This option is used to change the definition of an existing procedure without dropping, recreating.
FUNCTION	Is the name of function to be created.
ARGUMNET	is the name of argument to the function.
IN	Indicates that the parameter will **accept** a value from the user.
RETURN	Is the data type of the function's return values. Because every function must return a value, this clause is required.
PROGRAM BODY	Is the definition of function consisting of pl/sql statements.

Example:
The following example illustrates creating and calling a standalone function. This function returns the total number of CUSTOMERS in the customers table. We will use the CUSTOMERS table which we had created in PL/SQL Variables chapter:

```
Select * from customers;
```

```
+----+----------+-----+-----------+----------+
| ID | NAME | AGE | ADDRESS | SALARY |
+----+----------+-----+-----------+----------+
| 1 | Ramesh | 32 | Ahmedabad | 2000.00 |
| 2 | Khilan | 25 | Delhi | 1500.00 |
| 3 | kaushik | 23 | Kota | 2000.00 |
| 4 | Chaitali | 25 | Mumbai | 6500.00 |
| 5 | Hardik | 27 | Bhopal | 8500.00 |
| 6 | Komal | 22 | MP | 4500.00 |
+----+----------+-----+-----------+----------+
CREATE OR REPLACE FUNCTION totalCustomers
RETURN number IS
total number(2) := 0;
BEGIN
SELECT count(*) into total
FROM customers;
RETURN total;
END;
/
```

When above code is executed using SQL prompt, it will produce the following result:
Function created.

❖ **Calling a Function**

- While creating a function, you give a definition of what the function has to do. To use a function, you will have to call that function to perform the defined task.
- When a program calls a function, program control is transferred to the called function.
- A called function performs defined task and when its return statement is executed or when it last end statement is reached, it returns program control back to the main program.

- To call a function you simply need to pass the required parameters along with
- function name and if function returns a value then you can store returned value.

Following program calls the function totalCustomers from an anonymous block:

```
DECLARE
    c number(2);
BEGIN
    c := totalCustomers();
    dbms_output.put_line('Total no. of Customers: ' || c);
END;/
```

When the above code is executed at SQL prompt, it produces the following result:
Total no. of Customers: 6
PL/SQL procedure successfully completed.

Example :

```
CREATE OR REPLACE PROCEDURE raise_salary(emp_id INTEGER,increase REAL) IS
        curr_salary real;
        sal_missing EXCEPTION;
 BEGIN
        SELECT salary INTO curr_salary from emp
        WHERE id = emp_id;
        IF curr_salary IS NULL THEN
                RAISE sal_missing;
        ELSE
                UPDATE emp SET SALary = SALary+INCREASE
                WHERE id = emp_id;
        END IF;
        EXCEPTION
                WHEN NO_DATA_FOUND THEN
                        INSERT INTO EMP_AUDIT VALUES(emp_id,'no such number');
                WHEN SAL_MISSING THEN
                        INSERT INTO EMP_AUDIT VALUES (emp_id,'salary is null');
END RAISE_SALARY;
SQL> /

Procedure created.
```

- **Calling procedure**

```
declare
        emp number := &emp;
        incr number := &incr;
begin

        raise_salary(emp,incr);

end;
```