PL/SQL

PL/SQL

Introduction

What is Oracle?

RELIANCE Tech Services
Anil Dhirubhai Ambani Group

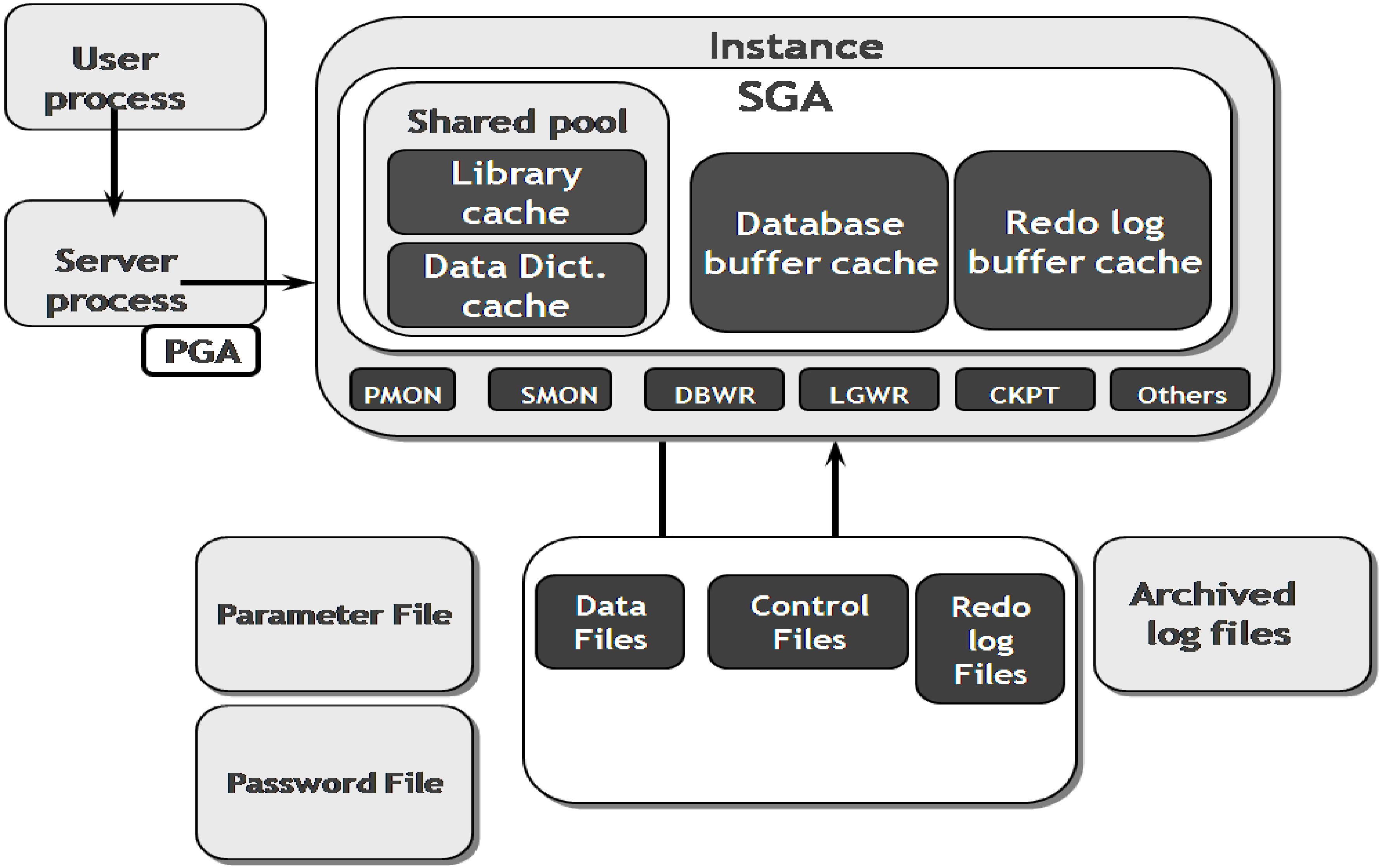
- Oracle is an Object Relational Database Management System (ORDBMS).
- Oracle uses:
  - Relational Data Model as well as Object Relational Data Model to store its database, and
  - SQL (commonly abbreviated as Structured Query Language) to process the stored data.
- Oracle 9i database is designed with improvements in the areas such as:
  - database performance
  - ease of management
  - scalability
  - security
  - availability, etc.

What is Oracle?

- The following features makes Oracle very powerful:
  - usage of XMLType, which is a new data type that lets you store native XML documents directly in the database
  - support of multimedia and large objects
  - support for Oracle Streams, which are a generic mechanism for sharing data that can be used as the basis of many processes including messaging, replication, and warehouse ETL processes

Oracle Architecture

RELIANCE Tech Services
Anil Dhirubhai Ambani Group



4

Oracle Architecture

- The Components of Oracle Architecture are as follows:
  - ➢ **Oracle server**: The "Oracle server" consists of an "Oracle instance" and an "Oracle database".
  - ➢ **Oracle instance:** An "Oracle instance" is the combination of the "background processes" and "memory structures".
    - ❖It is a means to access an Oracle database and always opens one and only one database.
    - ❖When a database is started on a database server (regardless of the type of computer), Oracle allocates a memory area called the "System Global Area (SGA)" and starts one or more "Oracle processes". This combination of the SGA and the Oracle processes is called an "Oracle instance".
    - ❖The memory and processes of an Oracle instance:
      - ✓**efficiently manage the data of the associated database, and**
      - ✓**serve one or multiple users of the database**
    - ❖Background processes perform functions on behalf of the invoking process. The background processes perform input / output (I/O), and monitor other Oracle processes to provide increased parallelism for "better performance" and "reliability".

Oracle Architecture

- **Oracle database**: An Oracle database consists of “Operating System files”, also known as “database files” that provide the actual physical storage for database information.
  - The database files are used to ensure that the data is kept consistent and can be recovered in the event of a failure of the instance.
- **Other key files**: Non-database files are used to configure the instance, authenticate privileged users, and recover the database in the event of a disk failure.
- **User and server processes**: The “user processes” and “server processes” are the primary processes involved when a SQL statement is executed. However, other processes may help the server complete the processing of the SQL statement.

Oracle Server

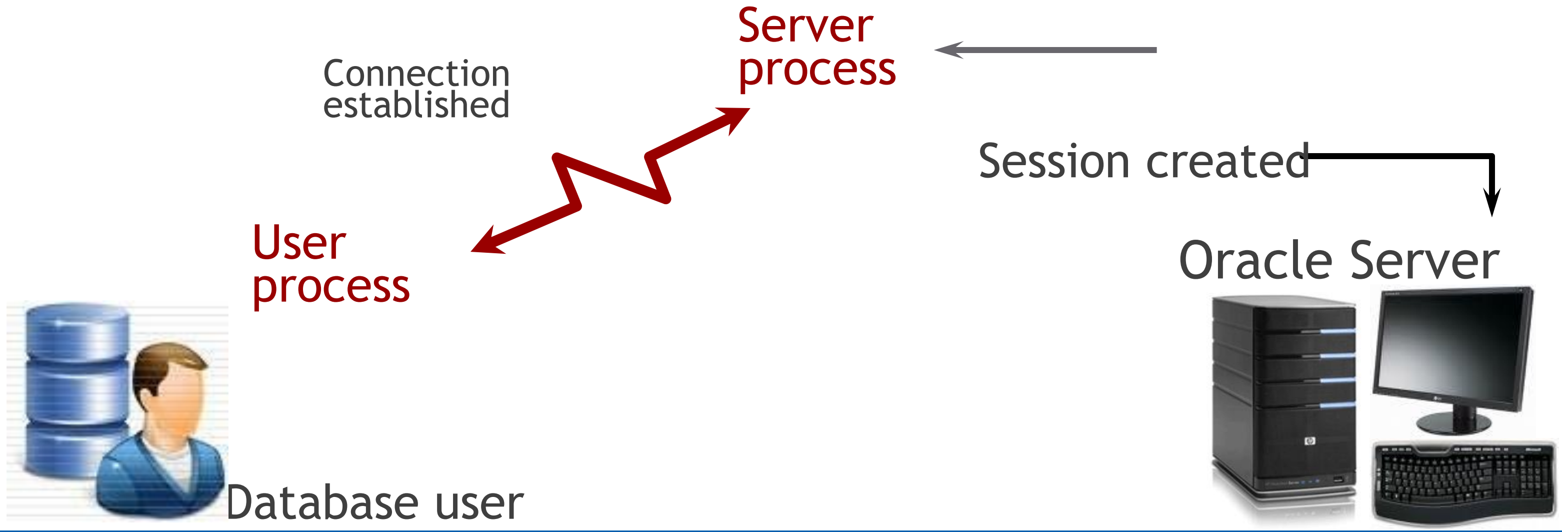
- An Oracle server:
  - is a “database management system (DBMS)” that provides an open, comprehensive, integrated approach to information management.
  - consists of
    - an “Oracle instance”
    - and an “Oracle database”.
- The Oracle Server can run on a number of different computers in one of the following environments:
  - Client-Application Server-Server
  - Client-Server
  - Host-Based

Oracle Instance

- When a database is started on a database server (regardless of the type of computer), Oracle allocates a memory area called the “System Global Area (SGA)” and starts one or more “Oracle processes”.
- This combination of the SGA and the Oracle processes is called an “Oracle instance”.
- The memory and processes of an Oracle instance:
  - ➢efficiently manage the data of the associated database, and
  - ➢serve one or multiple users of the database.
- An Oracle instance:
  - ➢is a means to access an Oracle database.
  - ➢always opens one and only one database.
  - ➢consists of memory and process structures.

Connecting to Oracle Instance

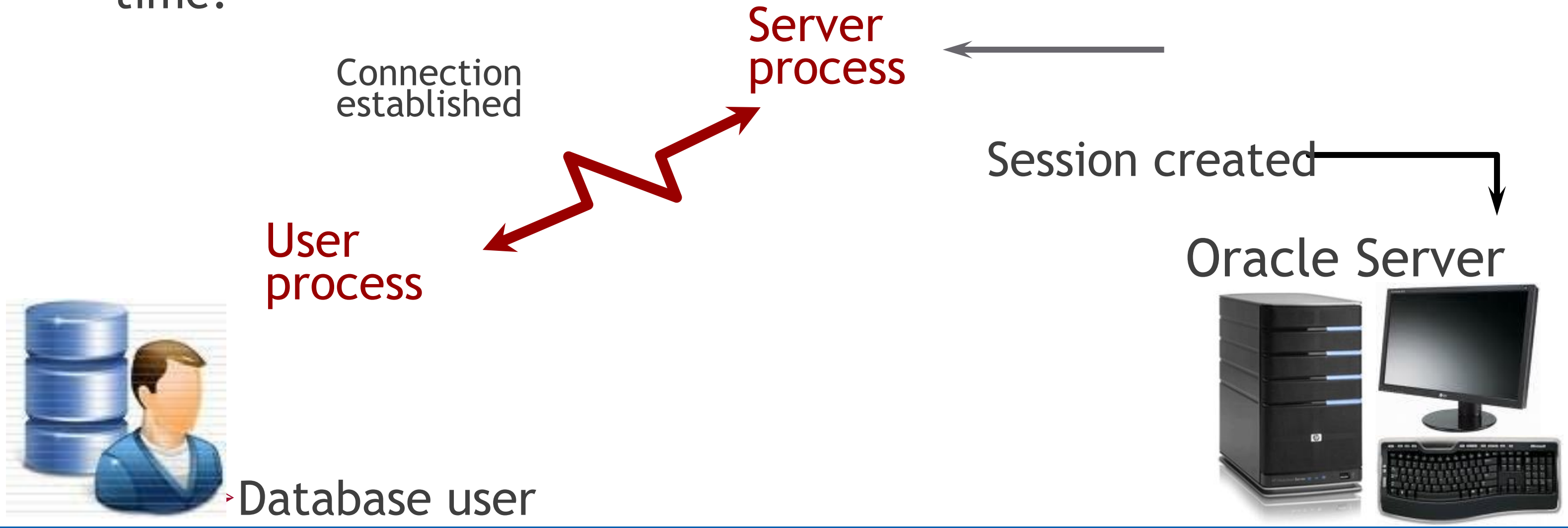
- Connecting to an Oracle instance consists of:
  - ➢ establishing a User Connection(tool such as SQL*Plus), and
    - ❖This application or tool is executed as a "user process".
    - ❖when a user logs on to the Oracle server, a process is created on the computer running the Oracle server. This process is called a "server process".
    - ❖The server process communicates with the Oracle instance on behalf of the "user process", which runs on the client. The "server process" executes SQL statements "on behalf of the user".



Connecting to Oracle Instance

RELIANCE Tech Services
Anil Dhirubhai Ambani Group

- Connecting to an Oracle instance consists of:
  - **Creating a Session**
    - A session is a "specific connection" of a user to an Oracle server.
      - ✓ The session starts when the user is validated by the Oracle server.
      - ✓ The session ends when the user logs out or when there is an abnormal termination.
    - For a given database user, many "concurrent sessions" are possible if the user logs on from many tools, applications, or terminals at the same time.



Oracle Database

- An Oracle database consists of three file types.
  - **Control files** that contain information necessary to maintain and verify database integrity.
  - **Data files** that contain the actual data in the database.
  - **Redo logs** that contain a record of changes made to the database to enable recovery of the data in case of failures.
- The Oracle server uses other files, as well, that are not part of the database.
  - **Parameter file** defines the characteristics of an Oracle instance. For example: The parameter file contains “parameters” that size some of the memory structures in the SGA.
    - ✓ There are two types of “initialization parameter files”:
      - **Static parameter file:**The parameter file is read-only during “instance startup
      - **Persistent parameter file:**SPFILE file is not meant to be manually modified and must always reside on the “server side”. It provides the ability to make changes to the database that are persistent across shutdown and startup.
  - **Password file** authenticates the users, who are privileged to start up and shut down an Oracle instance.
  - **Archived redo log files** are offline copies of the redo log files that may be necessary to recover from media failures

Memory Structure

- Memory Structure:
  - ➢ The Memory Structure of Oracle consists of two memory areas known as:
    - ❖**System Global Area (SGA)**: Allocated at instance startup, and is a fundamental component of an Oracle Instance.
    - ❖**Program Global Area (PGA)**: Allocated when the server process is started.

System Global Area (SGA)

- System Global Area (SGA):
  - ➢ System Global Area (SGA) is a group of “shared memory structures” that contain data and control information for one Oracle database instance. When “multiple users” are connected to the same instance, the data in the SGA is “shared by all users”. Hence, it is appropriately called “Shared Global Area”.
  - ➢ Oracle allocates memory for the SGA, when the database instance is started and returns the memory when the instance is shut down. The maximum size of the SGA is determined by sga_max_size initialization parameter in the initInstanceName.ora file or server parameter (SPFILE) file.
  - ➢ The SGA consists of several memory structures:
    - ❖Shared pool
    - ❖Database buffer cache
    - ❖Redo log buffer
    - ❖Other structures (E.g. lock and latch management, statistical data)

System Global Area (SGA)

- There are two optional memory structures that can be configured within the SGA:
  - Large pool
  - Java pool
- The size of the "variable portion", such as Shared Pool, Large Pool, and Java Pool, is variable. Hence, it is appropriately called "variable" because its size (which is measured in bytes) can be changed.
- A SGA is "Dynamic", which implements an infrastructure that allows the SGA configuration to change without shutting down the instance. As a result, it allows the sizes of the Database Buffer Cache, Shared Pool, and Large Pool to be changed without shutting down the instance.

SGA

- Shared Pool:
  - ➢ The shared pool is used to store the most recently executed SQL statements, and the most recently used Data Definitions.
  - ➢ It consists of two key performance-related memory structures:
    - ❖Library cache
    - ❖Data dictionary cache
  - ➢ It is sized by the parameter SHARED_POOL_SIZE.
- Library Cache :
  - ➢ The Library Cache stores information about the most recently used SQL and PL/SQL statements. The Library Cache enables sharing of commonly used statements.
  - ➢ It is managed by a least recently used (LRU) algorithm.
  - ➢ It consists of two structures:
    - ❖Shared SQL area
    - ❖Shared PL/SQL area

SGA

- **Shared area:** The shared area contains the parse tree and execution path for SQL statement.
- **Private area:** The private SQL area contains session specific information, such as bind variable, environment and session parameters, runtime stacks and buffers, etc.
    - The private SQL area is further divided into “persistent” and “runtime” areas.
        - → **Persistent area:** The persistent area contains information that is valid and applicable through multiple executions of the SQL statement.
        - → **Runtime area:** The runtime area contains data that is used only while the SQL statement is being executed.

- Data Dictionary Cache :
  - ➢ The Data Dictionary Cache is a collection of the most recently used definitions in the database.
    - ❖It includes information about database files, tables, Indexes, columns, users, privileges, and other database objects.
    - ❖During the "parse phase", the "server process" looks at the Data Dictionary for information to resolve "object names", and validate the access.
    - ❖Caching the Data Dictionary information into memory, improves the "response time" on queries.
  - ❖ The Data Dictionary Cache is also referred to as the "Dictionary Cache" or "Row Cache".

SGA

RELIANCE Tech Services
Anil Dhirubhai Ambani Group

- Database Buffer Cache:
  - The Database Buffer Cache stores copies of data blocks that have been retrieved from the data files.
    - It enables great "performance gains" when you obtain and update data.
    - It is managed through a least recently used (LRU) algorithm.
  - When a query is processed, the Oracle server process looks in the Database Buffer Cache for any blocks that may be required.
    - If the block is not found in the Database Buffer Cache, the server process reads the block from the data file, and places a copy in the Database Buffer Cache.
      - Because subsequent requests for the same block may find the block in memory, the requests may not require physical reads.
    - The Oracle server uses a Least Recently Used (LRU) algorithm to age out buffers that have not been recently accessed to make room for new blocks in the Database Buffer Cache.

SGA

- Redo Log Buffer Cache:
  - The Redo Log Buffer Cache records all changes made to the database data blocks.
    - It’s primary purpose is “recovery”.
    - Changes recorded within are called “redo entries”.
    - Redo entries contain information to “reconstruct” or “redo” changes.

SGA

- Large Pool:
  - ➢ The Large Pool is an optional area of memory in the SGA.
    - ❖It is configured only in a shared server environment.
  - ➢ When users connect through the “shared server”, Oracle needs to allocate “additional space” in the Shared Pool for storing information about the connections between the user processes, dispatchers, and servers. It relieves the burden placed on the “Shared Pool”.
  - ➢ This configured memory area is used for Session Memory (UGA), I/O slaves, and backup and restore operations.
  - ➢ Unlike Shared Pool, the Large Pool **does not** use an LRU list.
  - ➢ The performance gain is from the **reduction of overhead** from increasing and shrinkage of the shared SQL cache.

SGA

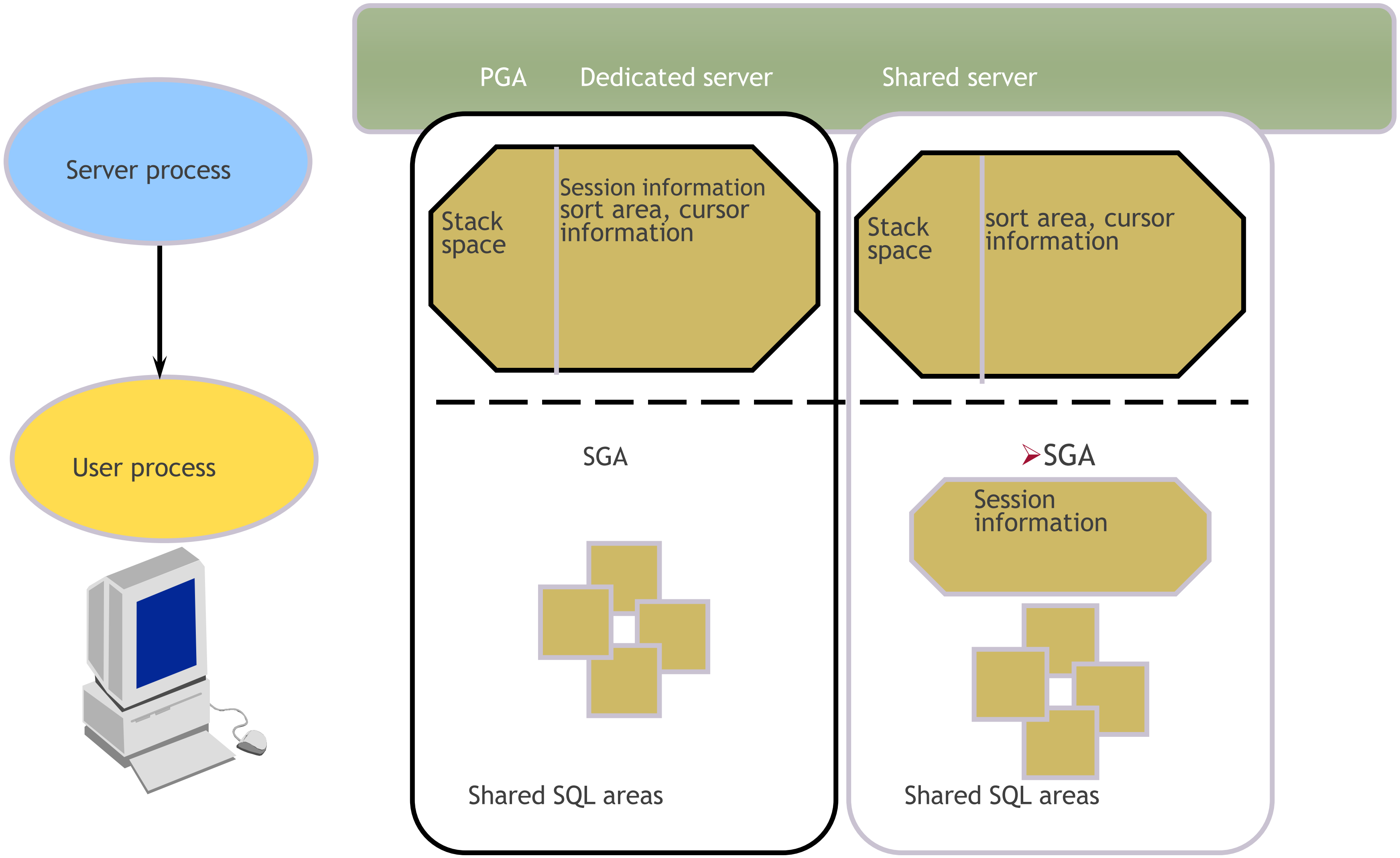
- Java Pool:
  - The Java Pool services the parsing requirements for Java commands.
  - It is required in case of installation and use of Java.
  - It is stored much the same way as PL/SQL in database tables.
  - It is sized by the JAVA_POOL_SIZE parameter.

PGA

RELIANCE Tech Services
Anil Dhirubhai Ambani Group

- Program Global Area (PGA):
  - The PGA is the memory reserved for each user process that connects to an Oracle database.
  - It is a “memory region” that contains data and control information for a “single server process” or a “single background process”.
  - The PGA is “allocated” when a process is created, and “de-allocated” when the process is terminated.
  - In contrast to the SGA, which is shared by several processes, the PGA is an area that is used by **only one process**.

PGA


Server process
User process
PGA
Dedicated server
Shared server
Stack space
Session information sort area, cursor information
SGA
Shared SQL areas
Stack space
sort area, cursor information
SGA
Session information
Shared SQL areas

PL/SQL

Introduction to PL/SQL

Introduction to PL/SQL

- PL/SQL is a procedural extension to SQL.
  - The “data manipulation” capabilities of “SQL” are combined with the “processing capabilities” of a “procedural language”.
  - PL/SQL provides features like conditional execution, looping and branching. PL/SQL supports subroutines as well.
  - PL/SQL program is of block type, which can be “sequential” or “nested” (one inside the other).
- PL/SQL is an “embedded language”. It was not designed to be used as a “standalone” language but instead to be invoked from within a “host” environment.
  - You cannot create a PL/SQL “executable” that runs all by itself.
  - It can run from within the database through SQL*Plus interface or from within an Oracle Developer Form (called client-side PL/SQL).

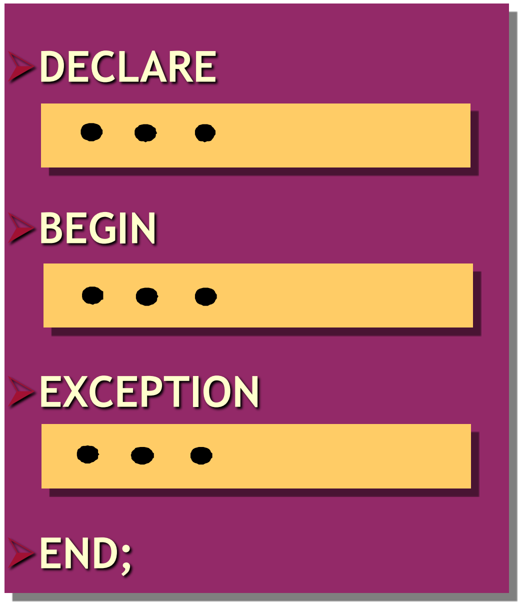
Features of PL/SQL

RELIANCE Tech Services
Anil Dhirubhai Ambani Group

- PL/SQL provides the following features:
  - ➢ Tight Integration with SQL :
    - ❖PL/SQL supports SQL data types, reducing the need to convert data passed between your application and database.
    - ❖PL/SQL lets you use all the SQL data manipulation, cursor control, transaction control commands, as well as SQL functions, operators, and pseudo columns.
  - ➢ Better performance
    - ❖Several SQL statements can be bundled together into one PL/SQL block and sent to the server as a single unit.
  - ➢ Standard and portable language
    - ❖Although there are a number of alternatives when it comes to writing software to run against the Oracle Database, it is easier to run highly efficient code in PL/SQL, to access the Oracle Database, than in any other language.

PL/SQL Block Structure

- PL/SQL is a block-structured language. Each basic programming unit that is written to build your application is (or should be) a “logical unit of work”.
- A PL/SQL block comprises of the following structures:
  - ➢ DECLARE – Optional
    - ❖Variables, cursors, user-defined exceptions
  - ➢ BEGIN – Mandatory
    - ❖SQL statements
    - ❖PL/SQL statements
  - ➢ EXCEPTION – Optional
    - ❖Actions to perform when errors occur
  - ➢ END; – Mandatory



PL/SQL Block Types

- The basic units (procedures and functions, also known as subprograms, and anonymous blocks) that make up a PL/SQL program are “logical blocks”, which can contain any number of nested sub-blocks.
- There are three types of blocks in PL/SQL:
  - ➢ Anonymous
    - ❖ Anonymous blocks are unnamed blocks. They are declared at the point in an application where they are to be executed and are passed to the PL/SQL engine for execution at runtime.
  - ➢ Named:
    - ❖Procedure
    - ❖Function

Variables in PL/SQL

- PL/SQL variables
  - Scalar
  - Composite
  - Reference(a pointer to another datatype.)
  - LOB (large objects)
- Guidelines for declaring variables:
  - initialize the variables designated as NOT NULL
  - initialize the identifiers by using the assignment operator (:=) or by using the DEFAULT reserved word
  - declare at most one Identifier per line
- While handling variables in PL/SQL:
  - declare and initialize variables within the declaration section
  - assign new values to variables within the executable section
  - pass values into PL/SQL blocks through parameters
  - view results through output variables

Variables in PL/SQL

- You need to declare all PL/SQL identifiers within the "declaration section" before referencing them within the PL/SQL block.
- Syntax

identifier [CONSTANT] datatype [NOT NULL] [:= | DEFAULT expr];

- ❖ **identifier** is the name of the variable.
- ❖ **CONSTANT** constrains the variable so that its value cannot change. Constants must be initialized.
- ❖ **datatype** is a scalar, composite, reference, or LOB datatype.
- ❖ **NOT NULL** constrains the variable so that it must contain a value. NOT NULL variables must be initialized.
- ❖ **expr** is any PL/SQL expression that can be a literal, another variable, or an expression involving operators and functions.

Data types in PL/SQL

- Base Scalar Datatypes:
  - Given below is a list of Base Scalar Datatypes:
    - VARCHAR2 (length)-The maximum length in PL/SQL is 32,767 bytes whereas VARCHAR2 database type can hold max 2000 bytes.
    - NUMBER [(precision, scale)]
    - DATE
    - CHAR [(maximum_length)]
    - LONG
    - LONG RAW
    - BOOLEAN
    - BINARY_INTEGER- *t can* store integers from -2147483647 to + 2147483647.
    - PLS_INTEGER

Example

```
v_job          VARCHAR2(9);
v_count        BINARY_INTEGER := 0;
v_total_sal    NUMBER(9,2) := 0;
v_orderdate    DATE := SYSDATE + 7;
c_tax_rate     CONSTANT NUMBER(3,2) := 8.25;
v_valid        BOOLEAN NOT NULL := TRUE;
```

%TYPE Attribute

- %TYPE Attribute:
  - %TYPE is used to declare a variable with the same datatype as a column of a specific table. This datatype is particularly used when declaring variables that will hold database values.
  - **Advantage:**
    - You need not know the exact datatype of a column in the table in the database.
    - If you change database definition of a column, it changes accordingly in the PL/SQL block at run time.

```
Var_Name        table_name.col_name%TYPE;
V_Empno         emp.empno%TYPE;
```

Example: %TYPE Attribute

```
declare
   nSalary emp.salary%type;
begin
       select sal into nsalary
       from employee
       where emp_code = 11;

       update employee set salary = salary + 101
       where emp_code = 11;
end;
```

%ROWTYPE Attribute

RELIANCE Tech Services
Anil Dhirubhai Ambani Group

- %ROWTYPE
  - It is used to declare a compound variable, whose type is same as that of a row of a table.
  - Columns in a row and corresponding fields in record should have same names and same datatypes. However, fields in a %ROWTYPE record do not inherit constraints, such as the NOT NULL, CHECK constraints, or default values.
  - Syntax:

```
Var_Name     table_name%ROWTYPE;
V_Emprec     emp%ROWTYPE
```

Example: %TYPE Attribute

```
DECLARE
   nRecord emp%rowtype;
BEGIN
     SELECT * into nRecord
     FROM emp
     WHERE empno = 11;

     UPDATE emp
     SET sal = sal + 101
     WHERE empno = 11;
END;
```

Example: Inserting and Updating Records

```
DECLARE
    dept_info dept%ROWTYPE;
BEGIN
-- deptno, dname, and loc are the table columns.
-- The record picks up these names from the %ROWTYPE.

    dept_info.deptno := 70;
    dept_info.dname := 'PERSONNEL';
    dept_info.loc := 'DALLAS';
/*Using the %ROWTYPE means we can leave out the column list
  (deptno, dname, loc) from the INSERT statement. */

    INSERT INTO dept VALUES dept_info;
END;
```

Composite Datatypes

- Composite Datatypes in PL/SQL:
  - Two composite datatypes are available in PL/SQL:
    - records
    - tables
  - A composite type contains components within it. A variable of a composite type contains one or more scalar variables.
  - Record Datatype:
    - A record is a collection of individual fields that represents a row in the table.
    - They are unique and each has its own name and datatype.
    - The record as a whole does not have value.

Composite Data types: Record

- Defining and declaring records:
  - Define a RECORD type, then declare records of that type.
  - Define in the declarative part of any block, subprogram, or package.
- Syntax:
  - **TYPE type_name IS RECORD (field_declaration [,field_ declaration] ...);**
  - where field_declaration stands for:
    - field_name field_type [[NOT NULL] {:= | DEFAULT} expression]
    - type_name is a type specifier used later to declare records. You can use %TYPE and %ROWTYPE to specify field types.
  - The value of a record is actually a collection of values, each of which is of some simpler type. The attribute %ROWTYPE lets you declare a record that represents a row in a database table.
  - After a record is declared, you can reference the record members directly by using the "." (Dot) notation. You can reference the fields in a record by indicating both the record and field names.

Example :Record Data type

```
DECLARE
    TYPE recname is RECORD
    (CUSTOMER_ID NUMBER,
     CUSTOMER_NAME VARCHAR2(20)
    );
    VAR_REC recname;
BEGIN
    var_rec.CUSTOMER_ID:=20;
    var_rec.CUSTOMER_name:=‘smith';
    dbms_output.put_line(var_rec.CUSTOMER_ID||'
    '||var_rec.CUSTOMER_name);
END;
```

Composite Datatype: table

RELIANCE Tech Services
Anil Dhirubhai Ambani Group

- A PL/SQL table :
  - They are objects of type TABLE, and look similar to database tables but with slight difference.
  - PL/SQL tables use a primary key to give you array-like access to rows.
    - Like the size of the database table, the size of a PL/SQL table is unconstrained. That is, the number of rows in a PL/SQL table can dynamically increase. So your PL/QSL table grows as new rows are added.
    - PL/SQL table can have one column and a primary key, neither of which can be named.
      - The column can have any datatype, but the primary key must be of the type BINARY_INTEGER.
    - Arrays are like temporary tables in memory. Thus they are processed very quickly.
    - Like the size of the database table, the size of a PL/SQL table is unconstrained.
    - The "column" can have any datatype. However, the "primary key" must be of the type BINARY_INTEGER.

Composite Datatype: PL/SQL Table

- Declaring a PL/SQL table:
  - There are two steps to declare a PL/SQL table:
    - Declare a TABLE type.
    - Declare PL/SQL tables of that type.
  - Syntax:

    TYPE type_name is TABLE OF

    {Column_type | table.column%type} [NOT NULL]

    INDEX BY BINARY_INTEGER;
  - Type_name is type specifier used in subsequent declarations to define PL/SQL tables and column_name is any datatype.
  - You can use %TYPE attribute to specify a column datatype. If the column to which table.
  - If the column is defined as NOT NULL, then PL/SQL table will reject NULLs.

Example: Composite Data type-table

- Example 1:
  - ➢ To create a PL/SQL table named as “emp_table” of char column.

```
DECLARE
TYPE emp_table is table of char(10)
INDEX BY BINARY_INTEGER;
```

- Example 2:
  - ➢ To create “Emp_table” based on the existing column of “Ename” of EMP table.

```
DECLARE
TYPE emp_table is table of emp.Ename%type
INDEX BY BINARY_INTEGER;
```

- Example 3:
  - ➢ To declare “Ename” column of the emp_table in PL/SQL as a NOT NULL.

```
DECLARE
TYPE emp_table is table of emp.ename%TYPE NOT NULL
INDEX BY BINARY_INTEGER;
```

Example: Composite Data type-table

- Step 2:
  - ➢ After defining type emp_table, define the PL/SQL tables of that type.
  - ➢ For example:

```
Ename_Tab emp_table;
```

  - ➢ These tables are unconstrained tables.
  - ➢ You cannot initialize a PL/SQL table in its declaration.
  - ➢ For example:

```
Ename_Tab  emp_tab :=('SMITH','JONES','BLAKE');          --Illegal
```

Referencing PL/SQL Tables

- To reference rows in a PL/SQL table, you specify the PRIMARY KEY value using the array-like syntax as shown below:

```
PL/SQL table_name (primary key value)
```

- Example:

```
DECLARE
        TYPE e_table is table of emp.Ename%type
        INDEX BY BINARY_INTEGER;
        e_tab e_table;
BEGIN
        e_tab(1) := 'SMITH'; --update SMITH's salary
        UPDATE EMP
        SET SAL = 1.1 * SAL
        WHERE ENAME = e_tab(1);
END;
```

Referencing Table

- Example 2:

```
DECLARE
        TYPE t_emp is table of emp%ROWtype
        INDEX BY BINARY_INTEGER;
        TYPE R_emp IS RECORD
                    (ENAME VARCHAR2(20), SAL NUMBER(4) );
        TYPE t_remp is table of R_EMP
        INDEX BY BINARY_INTEGER;
        V_Tremp         T_Remp ;
        V_Temp          T_Emp ;
BEGIN
        V_Tremp(1).ename := 'SMITH'; --assign values
        V_Temp(1).empno := 7566 ;
END;
```

Scope and Visibility of Variables

- Scope of Variables:
  - ➢ The scope of a variable is the portion of a program in which the variable can be accessed.
  - ➢ The scope of the variable is from the “variable declaration” in the block till the “end” of the block.
  - ➢ When the variable goes out of scope, the PL/SQL engine will free the memory used to store the variable, as it can no longer be referenced.

Types of Statements

- Given below are some of the SQL statements that are used in PL/SQL:
  - **INSERT statement**
    - The syntax for the INSERT statement remains the same as in SQL-INSERT.
    - For example:

```
DECLARE
  V_Dname VARCHAR2(15) := 'FINANCE';
BEGIN
  INSERT INTO dept VALUES (50, V_Dname, 'BOMBAY');
END;
```

Types of Statements

- **DELETE statement**
  - For example:

```
DECLARE
  V_Sal_Cutoff  NUMBER  := 2000;
BEGIN
  DELETE FROM emp  WHERE  sal < V_Sal_Cutoff;
END;
/
```

Types of Statements

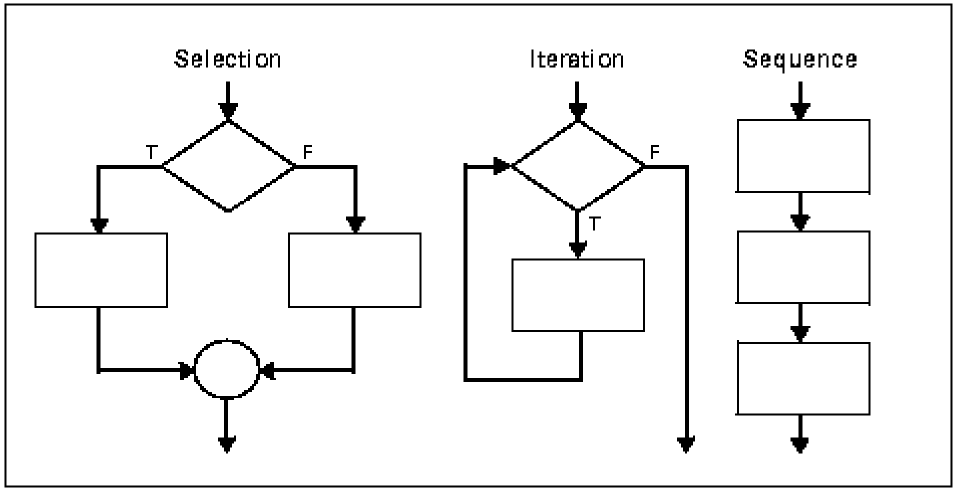
RELIANCE Tech Services
Anil Dhirubhai Ambani Group

- **UPDATE statement**
  - For example:

```
DECLARE
  V_Sal_Incr  NUMBER(5) := 1000;
BEGIN
  UPDATE emp SET sal = sal + V_Sal_Incr WHERE          ename='SMITH';
END;
```

Programmatic Constructs

- Programmatic Constructs are of the following types:
  - Selection structure:
    - condition, then executes one sequence of statements instead of another, depending on whether the condition is TRUE or FALSE.
    - A condition is any variable or expression that returns a Boolean value (TRUE or FALSE).
  - Iteration structure
  - Sequence structure



Conditional Execution

- **Conditional Execution:**
  - This construct is used to execute a set of statements only if a particular condition is TRUE or FALSE.
  - Syntax:

```
IF Condition_Expr
THEN
        PL/SQL_Statements
END IF;
```

  - To take alternate action if condition is FALSE, use the following syntax:

```
IF Condition_Expr THEN
        PL/SQL_Statements_1 ;
ELSE
        PL/SQL_Statements_2 ;
END IF;
```

Conditional Execution

❖To check for multiple conditions, use the following syntax.:

```
IF Condition_Expr_1
        THEN
                PL/SQL_Statements_1 ;
        ELSIF Condition_Expr_2
        THEN
                PL/SQL_Statements_2 ;
        ELSIF Condition_Expr_3
        THEN
                PL/SQL_Statements_3 ;
        ELSE
                PL/SQL_Statements_n ;
        END IF;
```

Listing

- **Looping**
  - A LOOP is used to execute a set of statements more than once.
  - Syntax:

```
LOOP
        PL/SQL_Statements;
END LOOP ;
```

- Example:

```
DECLARE
        V_Counter  NUMBER := 50 ;
BEGIN
LOOP
        INSERT INTO dept
                VALUES(V_Counter,'NEW
DEPT','SOMEWHERE');                          V_Counter :=
V_Counter + 10 ;
END LOOP;
        COMMIT ;
END ;
/
```

Exit

- **EXIT**
  - Exit path is provided by using EXIT or EXIT WHEN commands.
  - EXIT is an unconditional exit. Control is transferred to the statement following END LOOP, when the execution flow reaches the EXIT statement.

```
BEGIN
          .....
          LOOP
                IF <Condition> THEN


                      . . . . .
                      EXIT ;
                END IF ;

          END LOOP;
          LOOP
                ..........
                EXIT WHEN <condition>
          END LOOP;
          . .
          COMMIT ;
          END ;
```

Exit

❖For example:

```
DECLARE
 V_Counter NUMBER := 50 ;
BEGIN
 LOOP
      INSERT INTO dept
         VALUES(V_Counter,'NEWDEPT','SOMEWHERE');
 DELETE  FROM emp WHERE deptno = V_Counter;
         V_Counter := V_Counter + 10 ;
         EXIT WHEN  V_Counter >100 ;
 END LOOP;
 COMMIT ;
END ;
```

FOR Loop

- **FOR Loop:**
  - Syntax:

```
FOR Variable IN [REVERSE] Lower_Bound..Upper_Bound
LOOP
  PL/SQL_Statements
END LOOP ;
```

- Variables in FOR loop need not be explicitly declared.
  - Variables take values starting at a Lower_Bound and ending at a Upper_Bound.
  - The variable value is incremented by 1, every time the loop reaches the bottom.
  - When the variable value becomes equal to the Upper_Bound, then the loop executes and exits.
- When REVERSE is used, then the variable takes values starting at Upper_Bound and ending at Lower_Bound.
- Value of the variable is decremented each time the loop reaches the bottom.

For Loop

❖For Example:

```
DECLARE
V_Counter   NUMBER := 50 ;
BEGIN
  FOR  Loop_Counter  IN 2..5
  LOOP
  INSERT INTO dept
  VALUES(V_Counter,'NEW DEPT','SOMEWHERE') ;
  V_Counter := V_Counter + 10 ;
  END LOOP;
  COMMIT ;
END ;
```

While Loop

- **WHILE Loop**
  - The WHILE loop is used as shown below.
  - Syntax:

```
WHILE Condition
LOOP
        PL/SQL  Statements;
END LOOP;
```

  - EXIT OR EXIT WHEN can be used inside the WHILE loop to prematurely exit the loop.

SELECT CASE

- The syntax for the **case** statement is:

```
CASE  [ expression ]
   WHEN condition_1 THEN result_1
   WHEN condition_2 THEN result_2
   ...
   WHEN condition_n THEN result_n
   ELSE result
  END
```

- ❖*expression* is optional. It is the value that you are comparing to the list of conditions. (ie: condition_1, condition_2, ... condition_n)
- ❖*condition_1* to *condition_n* must all be the same datatype. Conditions are evaluated in the order listed. Once a *condition* is true, the **case** statement will return the result and stop evaluating the conditions.
- ❖*result_1* to *result_n* must all be the same datatype. This is the value returned once a *condition* is found to be true.

SELECT CASE

- Example:

```
        DECLARE
         score number :=&score;
        BEGIN
         CASE
           WHEN score BETWEEN 80 AND 100 THEN dbms_output.put_line('A');
           WHEN score BETWEEN 65 AND 79  THEN dbms_output.put_line('B');
           WHEN score BETWEEN 50 AND 64  THEN dbms_output.put_line('C');
           WHEN score BETWEEN 40 AND 49  THEN dbms_output.put_line('D');
           WHEN score BETWEEN 0  AND 39  THEN dbms_output.put_line('F');
           ELSE dbms_output.put_line( 'Invalid score');
         END CASE;
      END;
```

Labelling of Loops

- **Labeling of Loops:**
  - The label can be used with the EXIT statement to exit out of a particular loop.

```
BEGIN
 <<Outer_Loop>>
 LOOP
         PL/SQL
         << Inner_Loop>>
         LOOP
                   PL/SQL  Statements ;
                   EXIT Outer_Loop WHEN <Condition Met>
         END LOOP  Inner_Loop
 END LOOP  Outer_Loop
END ;
 /
```

PL/SQL

Cursors

Introduction to Cursor

- ORACLE allocates memory on the Oracle server to process SQL statements. It is called as “**context area**”. Context area stores information like number of rows processed, the set of rows returned by a query, etc.
- A Cursor is a “handle” or “pointer” to the context area. Using this cursor the PL/SQL program can control the context area, and thereby access the information stored in it.
- There are two types of cursors - “explicit” and “implicit”.
  - ➢ In an explicit cursor, a cursor name is explicitly assigned to a SELECT statement through CURSOR IS statement.
  - ➢ An implicit cursor is used for all other SQL statements.
  - ➢ Processing an explicit cursor involves four steps. In case of implicit cursors, the PL/SQL engine automatically takes care of these four steps.

Explicit Cursors

RELIANCE Tech Services
Anil Dhirubhai Ambani Group

- Explicit Cursor:
  - ➢ The set of rows returned by a query can consist of zero, one, or multiple rows, depending on how many rows meet your search criteria.
  - ➢ When a query returns multiple rows, you can explicitly declare a cursor to process the rows.
  - ➢ You can declare a cursor in the declarative part of any PL/SQL block, subprogram, or package.
  - ➢ Processing has to be done by the user.
- Implicit Cursor:
  - ➢ The PL/SQL engine takes care of automatic processing.
  - ➢ PL/SQL implicitly declares cursors for all DML statements.
  - ➢ They are simple SELECT statements and are written in the BEGIN block (executable section) of the PL/SQL.
  - ➢ They are easy to code, and they retrieve exactly one row

Processing Explicit Cursors

RELIANCE Tech Services
Anil Dhirubhai Ambani Group

- While processing Explicit Cursors you have to perform the following four steps:
    1. Declare the cursor
    2. Open the cursor for a query
    3. Fetch the results into PL/SQL variables
    4. Close the cursor

Cursor

1. Declaring a Cursor:
   - Syntax:

     CURSOR Cursor_Name IS Select_Statement
   - Any SELECT statements are legal including JOINS, UNION, and MINUS clauses.
     - SELECT statement should not have an INTO clause.
   - Cursor declaration can reference PL/SQL variables in the WHERE clause.
     - The variables (bind variables) used in the WHERE clause must be visible at the point of the cursor.

2. Opening a Cursor

- Syntax:

```
OPEN  Cursor_Name;
```

- When a cursor is opened, the following events occur:
  1. The values of bind variables are examined.
  2. The active result set is determined.
  3. The active result set pointer is set to the first row.
- "Bind variables" are evaluated only once at Cursor open time.
  - Changing the value of Bind variables after the Cursor is opened will not make any changes to the active result set.
  - The query will see changes made to the database that have been committed prior to the OPEN statement.
- You can open more than one Cursor at a time.

3. Fetching from a Cursor

- Syntax:

```
FETCH Cursor_Name  INTO List_Of_Variables;
FETCH  Cursor_Name INTO PL/SQL _Record;
```

- The “list of variables” in the INTO clause should match the “column names list” in the SELECT clause of the CURSOR declaration, both in terms of count as well as in datatype.
- After each FETCH, the active set pointer is increased to point to the next row.
  - ✓ The end of the active set can be found out by using %NOTFOUND attribute of the cursor.
- For each column value returned by the query associated with the cursor, there must be a corresponding, type-compatible variable in the INTO list.

4. Closing a Cursor

- Syntax

  CLOSE Cursor_Name;

  - Closing a Cursor frees the resources associated with the Cursor.
    - ✓ You cannot FETCH from a closed Cursor.
    - ✓ You cannot close an already closed Cursor.

Cursor Attributes

RELIANCE Tech Services
Anil Dhirubhai Ambani Group

- Cursor Attributes:
  - Explicit cursor attributes return information about the execution of a multi-row query.
  - When an “Explicit cursor” or a “cursor variable” is opened, the rows that satisfy the associated query are identified and form the result set.
  - Attributes are:
    - %ISOPEN,
    - %FOUND,
    - %NOTFOUND,
    - %ROWCOUNT, etc.

Cursor Attributes

- **%ISOPEN**
  - %ISOPEN returns TRUE if its cursor or cursor variable is open. Otherwise it returns FALSE.
  - Oracle closes the SQL cursor automatically after executing its associated SQL statement. As a result, %ISOPEN always yields FALSE for Implicit cursor.
  - Syntax:

```
        Cur_Name%ISOPEN
```

  - Exampe:

```
        DECLARE
                CURSOR C1 IS
                SELECT_Statement ;
         BEGIN
             IF C1%ISOPEN THEN
                PL/SQL_Statements ;
             END IF;
          END ;
```

Cursor Attributes

- %FOUND
  - %FOUND yields NULL after a cursor or cursor variable is opened but before the first fetch.
  - Thereafter, it yields:
    - ✓TRUE if the last fetch has returned a row, or
    - ✓FALSE if the last fetch has failed to return a row
  - Syntax:

```
Cur_Name%FOUND
```

  - Example:

```
DECLARE Section;
OPEN C1 ;
FETCH C1 INTO Var_List ;
IF C1%FOUND THEN
        PL/SQL_Statements ;
END IF ;
```

Cursor Attributes

- **%NOTFOUND**
  - %NOTFOUND is the logical opposite of %FOUND.
  - %NOTFOUND yields:
    - ✓FALSE if the last fetch has returned a row, or
    - ✓TRUE if the last fetch has failed to return a row
  - It is mostly used as an exit condition.
  - Syntax:

Cur_Name%NOTFOUND

Types of Cursor Attributes

- **%ROWCOUNT**
  - %ROWCOUNT returns number of rows fetched from the cursor area using FETCH command.
  - %ROWCOUNT is zeroed when its cursor or cursor variable is opened.
    - ✓Before the first fetch, %ROWCOUNT yields 0.
    - ✓Thereafter, it yields the number of rows fetched at that point of time.
  - The number is incremented if the last FETCH has returned a row.
  - Syntax:

    Cur_Name%NOTFOUND

- Processing Implicit Cursors:
  - Oracle implicitly opens a cursor to process each SQL statement that is not associated with an explicitly declared cursor.
  - This implicit cursor is known as SQL cursor.
    - Program cannot use the OPEN, FETCH, and CLOSE statements to control the SQL cursor. PL/SQL implicitly does those operations .
    - You can use cursor attributes to get information about the most recently executed SQL statement.
    - Implicit Cursor is used to process INSERT, UPDATE, DELETE, and single row SELECT INTO statements.

Implicit cursors

- Example 1:

```
BEGIN
UPDATE dept SET dname ='SYSTEMS‘ WHERE deptno= 50;
IF SQL%NOTFOUND THEN
   INSERT INTO dept(deptno,dname) VALUES ( 50, 'SYSTEMS');
END IF;
END;
```

- Example 2:

```
BEGIN
UPDATE dept
   SET dname ='SYSTEMS'
   WHERE deptno = 50;
  IF SQL%ROWCOUNT = 0 THEN
       INSERT INTO dept(deptno,dname)VALUES ( 50, 'SYSTEMS');
END IF;
END;
```

FETCH loops

- They are examples of simple loop statements.
- The FETCH statement should be followed by the EXIT condition to avoid infinite looping.
- Condition to be checked is cursor%NOTFOUND.
- Examples: LOOP .. END LOOP, WHILE LOOP, etc

```
DECLARE
  CURSOR C1 IS ……..
BEGIN
  OPEN CURSOR C1;  /* Open the cursor and identify the active
           result set.*/
LOOP
  FETCH C1 INTO  Variable_List ;
   -- Exit out of the loop when there are no more rows.
  /* Exit is done before processing to prevent handling of NULL rows.*/
           EXIT WHEN C1%NOTFOUND ;
```

FETCH

```
        END LOOP;
        CLOSE  C1;
        COMMIT;
    END;
```

- Example: Fetch using WHILE Loop

```
    DECLARE
            CURSOR C1  IS ........
    BEGIN
        OPEN CURSOR  C1 ;
        FETCH C1 INTO  Variable_List ;
        WHILE C1%FOUND
        LOOP
            FETCH C1 INTO Variable_List;
        END LOOP;
        CLOSE C1;
        COMMIT;
    END;
```

FOR Cursor Loop

- FOR Cursor Loop:
  - For all other loops we had to go through all the steps of OPEN, FETCH, AND CLOSE statements for a cursor.
  - PL/SQL provides a shortcut via a CURSOR FOR Loop. The CURSOR FOR Loop implicitly handles the cursor processing.
    - You can pass parameters to the cursor in a CURSOR FOR loop.

```
FOR Variable in Cursor_Name ( PARAM1 , PARAM 2 ….);
  FOR V_Get_Det in C_Select_Emp (800,5000)
  LOOP
   Process the variables
  END LOOP;
```

Passing Parameters to Cursor

- Example:

```
DECLARE
 V_Empno emp.empno%TYPE;
 CURSOR C(Job VARCHAR2,Sal NUMBER)
 IS  SELECT empno FROM emp WHERE job=Job  AND sal>Sal ;
BEGIN
 OPEN C('CLERK',1000);
 FETCH C INTO V_Empno;
 WHILE C%FOUND
 LOOP
        UPDATE emp SET job ='SR.CLERK‘ WHERE empno = V_Empno ;
        FETCH C INTO V_Empno;
 END LOOP ;
 CLOSE C;
END;
```

Passing Parameters to Cursor – FOR LOOP

- Example:

```
DECLARE
CURSOR C(Job VARCHAR,Sal  NUMBER ) IS
SELECT empno,job FROM emp WHERE job = Job AND sal>Sal
FOR UPDATE  OF sal NOWAIT ;
BEGIN
FOR V_Emprec IN C( 'CLERK',1000)
LOOP
     UPDATE emp SET JOB='SR.CLERK',        sal = 1.1 * sal
     WHERE CURRENT OF C;
 END LOOP ;
END;
/
```

Reliance Tech Services
Anil Dhirubhai Ambani Group

PL/SQL

Exception Handling

Introduction

- Error Handling:
  - ➢ 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:
      - ✓division by zero
      - ✓out of memory
    - ❖Some common internal exceptions have predefined names, namely:
      - ✓ZERO_DIVIDE
      - ✓STORAGE_ERROR
    - ❖The other exceptions can be given user-defined names.
    - ❖Exceptions can be defined in the declarative part of any PL/SQL block, subprogram, or package. These are user-defined exceptions.

Declaring Exceptions

- Exception is an error that is defined by the program.
  - ➢ It could be an error with the data, as well.
- There are three types of exceptions in Oracle:
  - ➢ Predefined exceptions
  - ➢ User defined exceptions
  - ➢ Numbered exceptions
- Exceptions are declared in the Declaration section, raised in the Executable section, and handled in the Exception section.

Predefined Exceptions

- Predefined Exceptions correspond to the most common Oracle errors.
  - They are always available to the program. Hence there is no need to declare them.
  - They are automatically raised by ORACLE whenever that particular error condition occurs.
  - Examples:
    - **NO_DATA_FOUND:**This exception is raised when SELECT INTO …. statement does not return any rows.
    - **TOO_MANY_ROWS:**This exception is raised when SELECT INTO …. statement returns more than one row.
    - **INVALID_CURSOR:**This exception is raised when an illegal cursor operation is performed such as closing an already closed cursor.
    - **VALUE_ERROR:**This exception is raised when an arithmetic, conversion, truncation, or constraint error occurs in a procedural statement.
    - **DUP_VAL_ON_INDEX:**This exception is raised when the UNIQUE CONSTRAINT is violated.

Handling User Defined Exceptions

- Declaring a User Defined Exception:

```
DECLARE
  E_Balance_Not_Sufficient EXCEPTION;
  E_Comm_Too_Large EXCEPTION;
  …
BEGIN
  NULL;
END;
```

- Raising Exceptions:
  - ➢ Internal exceptions are raised implicitly by the run-time system, as are user-defined exceptions that are associated with an Oracle error number using EXCEPTION_INIT.
  - ➢ Other user-defined exceptions must be raised explicitly by RAISE statements.
    - ❖The syntax is:

```
RAISE  Exception_Name;
```

Example

- An exception is defined and raised as shown below:

```
DECLARE
  ...
  Retired_Emp EXCEPTION ;
BEGIN
  PL/SQL_Statements ;
  IF Error_Condition THEN
            RAISE Retired_Emp ;
            PL/SQL_Statements ;
EXCEPTION
  WHEN Retired_Emp THEN
            PL/SQL_Statements ;
END ;
/
```

Exception Handling

- Control passing to Exception Handler :
  - When an exception is raised, normal execution of your PL/SQL block or subprogram stops, and control passes to its exception-handling part.
  - To catch the raised exceptions, you write "exception handlers".
    - Each exception handler consists of a WHEN clause, which specifies an exception, followed by a sequence of statements to be executed when that exception is raised.
    - These statements complete execution of the block or subprogram, however, the control does not return to where the exception was raised. In other words, you cannot resume processing where you left off.

Example

```
DECLARE
        Dup_Deptno EXCEPTION;
        V_Counter BINARY_INTEGER;
        V_Department NUMBER(2) := 50;
BEGIN
        SELECT COUNT(*) INTO V_Counter  FROM Dept WHERE  deptno
  = 50;
        IF V_Counter > 0 THEN
                RAISE Dup_Deptno ;
        END IF;
  INSERT INTO dept VALUES (V_Department ,'NEW NAME', 'NEW LOC');
EXCEPTION
        WHEN Dup_Deptno THEN INSERT INTO Error_Log VALUES
  ('Department ‘  || V_Department   ||' IS ALREADY PRESENT');
END ;
```

- OTHERS Exception Handler:
  - The optional OTHERS exception handler, which is always the last handler in a block or subprogram, acts as the handler for all exceptions that are not specifically named in the Exception section.
  - A block or subprogram can have only one OTHERS handler.
  - To handle a specific case within the OTHERS handler, predefined functions SQLCODE and SQLERRM are used.
    - SQLCODE returns the current error code. And SQLERRM returns the current error message text.
    - The values of SQLCODE and SQLERRM should be assigned to local variables before using it within a SQL statement.

Associating an Exception Name to Oracle Errors

- An exception name can be associated with an ORACLE error.
  - This gives us the ability to trap the error specifically rather than via the OTHERS handlers.
  - This is done with the help of “compiler directives”.
  - For example: PRAGMA EXCEPTION_INIT
    - A PRAGMA is a compiler directive that is processed at compile time, not at run time. It is used to name an exception.
    - In PL/SQL, the PRAGMA EXCEPTION_INIT tells the compiler to associate an exception name with an Oracle error number.
      - ✓This arrangement lets you refer to any internal exception by name, and to write a specific handler for it.
    - When you see an error stack, or sequence of error messages, the one on top is the one that you can trap and handle.

PRAGMA_INIT

- User defined exceptions can be named with error number between -20000 and -20999.
- The naming is declared in Declaration section.
- It is valid within the PL/SQL blocks only.
- Syntax is:

```
PRAGMA EXCEPTION_INIT(Exception Name,Error_Number);
```

- Example:

```
DECLARE
  V_Deptno NUMBER := 10;
  Child_Record_Found EXCEPTION;
--All ORACLE ERRORS ARE NEGATIVE
--Error-2292 OCCURS WHEN REFERENTIAL INTEGRITY RULE IS VIOLATED
          PRAGMA EXCEPTION_INIT (Child_Record_Found, -2292);
BEGIN
  DELETE FROM dept WHERE deptno = V_Deptno;
EXCEPTION
  WHEN Child_Record_Found THEN
/* Error_Log is a table containing one varchar2 column created by the user to
  store error messages */
```

Example

```
INSERT INTO Error_Log
        VALUES ('Employees Exist for Department No' || V_Deptno);
        WHEN OTHERS THEN
                INSERT INTO Error_Log VALUES('Unable to delete dept No'
|| V_Deptno || 'because of Unknown Reasons');
END;
```

RELIANCE Tech Services
Anil Dhirubhai Ambani Group

PL/SQL

Subprograms in PL/SQL

Subprograms in PL/SQL

- A subprogram is a named block of PL/SQL.
- There are two types of subprograms in PL/SQL, namely: Procedures and Functions.
- Each subprogram has:
  - A declarative part
  - An executable part or body, and
  - An exception handling part (which is optional)
- A function is used to perform an action and return a single value.
- When a client executes a procedure or function, the processing is done in the server. This reduces the network traffic.
- The subprograms are compiled and stored in the Oracle database as “stored programs”, and can be invoked whenever required. As the subprograms are stored in a compiled form, when called they only need to be executed. [illegible]

Procedure

- A procedure is a subprogram used to perform a specific action.
- A procedure contains two parts:
  - the specification, and
  - the body
- It is illegal to constrain datatypes.
- Syntax:

```
CREATE PROCEDURE  Proc_Name
    (Parameter {IN | OUT | IN OUT} datatype := value,...) AS
    Variable_Declaration ;
    Cursor_Declaration ;
    Exception_Declaration ;
BEGIN
     PL/SQL_Statements ;
EXCEPTION
     Exception_Definition ;
END Proc_Name ;
```

Example:

```
CREATE OR REPLACE PROCEDRE PROC1 AS
BEGIN
   DBMS_OUTPUT.PUT_LINE('Hello from procedure …');
END;
/
```

- To execute the procedure type:
  - Execute PROC1

Parameter Modes

IN	OUT	IN OUT
The default	Must be specified	Must be specified
Used to pass values to the procedure.	Used to return values to the caller.	Used to pass initial values to the procedure and return updated values to the caller.
Formal parameter acts like a constant.	Formal parameter acts like an uninitialized variable.	Formal parameter acts like an uninitialized variable.
Formal parameter cannot be assigned a value.	Formal parameter cannot be used in an expression, but should be assigned a value.	Formal parameter should be assigned a value.
Actual parameter can be a constant, literal, initialized variable, or expression.	Actual parameter must be a variable.	Actual parameter must be a variable.
Actual parameter is passed by reference (a pointer to the value is passed in).	Actual parameter is passed by value (a copy of the value is passed out) unless NOCOPY is specified.	Actual parameter is passed by value (a copy of the value is passed in and out) unless NOCOPY is specified.

Parameter Modes

Using IN mode:

- While using an IN parameter, you can pass values to the subprogram being called. Inside the subprogram, an IN parameter acts like a “constant”. It cannot be assigned a value.
- You can pass a constant, literal, initialized variable, or expression as an IN parameter.
- IN parameters can be initialized to default values, which are used if those parameters are omitted from the subprogram call.

Using OUT mode:

- An OUT parameter returns a value to the caller of a subprogram.
- Inside the subprogram, an OUT parameter acts like a “variable”. You can change its value, and reference the value after assigning it.
- Example:

Example:

•Example:

```
CREATE OR REPLACE PROCEDURE PROC2(N1 IN

NUMBER, N2 IN NUMBER, TOT OUT NUMBER) IS
BEGIN
        TOT := N1 + N2;
END;
/
```

•To execute the procedure:

```
SQL > VARIABLE T NUMBER
SQL > EXEC PROC2(33, 66, :T)

PL/SQL procedure successfully completed.

SQL > PRINT T
```

Example:

```
PROCEDURE split_name(
  phrase IN VARCHAR2, first OUT VARCHAR2, last OUT VARCHAR2) IS
  first := SUBSTR(phrase, 1, INSTR(phrase, ' ')-1);
  last := SUBSTR(phrase, INSTR(phrase, ' ')+1);
  IF first = 'John' THEN
    DBMS_OUTPUT.PUT_LINE('That is a common first name.');
  END IF;END;
/
```

Example:

```
CREATE OR REPLACE PROCEDURE Raise_Salary
( P_Empno IN NUMBER, raise in NUMBER) IS
    V_Cur_Salary   NUMBER ;
    Missing_Salary EXCEPTION;
BEGIN
    SELECT sal into V_Cur_Salary FROM emp WHERE empno=P_Empno;
    IF  V_Cur_Salary IS NULL THEN
            RAISE  Missing_Salary;
    END IF ;
  UPDATE emp SET sal = sal + V_Cur_Salary   WHERE empno =
  P_Empno ;
EXCEPTION
  WHEN NO_DATA_FOUND THEN
INSERT INTO Emp_Audit VALUES( P_Empno, 'No Employee with id ' `
            || P_Empno);
  WHEN  Missing_Salary THEN
INSERT INTO Emp_Audit VALUES( P_Empno, 'SALARY IS MISSING');
END;
/
```

Example:

- Example 2:

```
CREATE OR REPLACE PROCEDURE Get_Emp_Details
(P_Empno IN NUMBER, P_Ename OUT VARCHAR2,P_Sal OUT NUMBER )
    IS
BEGIN
      SELECT ename, sal into P_Ename, P_Sal
      from emp
      WHERE empno=P_Empno;
EXCEPTION
      WHEN NO_DATA_FOUND  THEN
      INSERT INTO Emp_Audit
      VALUES( P_Empno, 'No Employee with id ' || P_Empno);
      P_Ename := NULL;
      P_Sal := NULL;
END ;
```

Example:

- Example 3: Passing DEFAULT values to parameters

```
PROCEDURE Create_Dept( New_Deptno IN NUMBER,
   New_Dname IN  VARCHAR2  DEFAULT 'TEMP'
   New_Loc  IN VARCHAR2  DEFAULT  'TEMP' ) IS
BEGIN
     INSERT INTO  dept VALUES ( New_Deptno, New_Dname, New_Loc) ;
END ;
```

Functions

Reliance Tech Services
Anil Dhirubhai Ambani Group

- A function is similar to a procedure and is used to compute a value.
  - ➢ A function can return multiple values by using OUT parameters.
  - ➢ A function is used as part of an expression, and can be called as Lvalue = Function_Name(Param1, Param2, …….)
  - ➢ Functions returning a single value for a row can be used with SQL statements.
- Syntax:

```
CREATE  FUNCTION Func_Name(Param datatype := value,..)
RETURN datatype1 AS
    Variable_Declaration ;
    Cursor_Declaration ;
    Exception_Declaration ;
BEGIN
      PL/SQL_Statements ;
      RETURN Variable_Or_Value_Of_Type_Datatype1 ;
EXCEPTION
  Exception_Definition ;
END Func_Name ;
```

Example:

- Example 1:

```
CREATE FUNCTION Crt_Dept(Dno NUMBER, Dname VARCHAR2, Dloc
  VARCHAR2) RETURN BOOLEAN AS
BEGIN
    INSERT INTO dept
    VALUES(Dno,Dname,Dloc) ;
    RETURN TRUE ;
EXCEPTION
    WHEN OTHERS THEN
    RETURN FALSE ;
END Crt_Dept ;
```

Example:

RELIANCE Tech Services
Anil Dhirubhai Ambani Group

- Example 2:

```
CREATE OR REPLACE FUNCTION Get_Avg_Sal(P_Deptno IN NUMBER)
RETURN NUMBER AS
   V_Sal NUMBER;
BEGIN
   SELECT Trunc(Avg(sal)) INTO V_Sal from emp
   Where deptno=P_Deptno;
   IF V_Sal IS NULL THEN
           V_Sal := -1 ;
  END IF;
  RETURN V_Sal;
EXCEPTION
  WHEN OTHERS THEN RETURN -2;
END Get_Avg_Sal;
```

Packages

- Packages are PL/SQL constructs that allow related objects to be stored together.
- A Package consists of two parts, namely "Package Specification" and "Package Body" and are stored separately in a "Data Dictionary".
  - **Package Specification:** It is used to declare functions and procedures that are part of the package. Package Specification also contains variable and cursor declarations, which are used by the functions and procedures.
    - Any object declared in a Package Specification can be referenced from other PL/SQL blocks. So Packages provide global variables to PL/SQL.
  - **Package Body:** It contains the function and procedure definitions, which are declared in the Package Specification.
    - The Package Body is optional.
    - If the Package Specification does not contain any procedures or functions and contains only variable and cursor declarations then the body need not be present.

Packages

RELIANCE Tech Services
Anil Dhirubhai Ambani Group

- All functions and procedures declared in the Package Specification are accessible to all users who have permissions to access the Package.
- Users cannot access subprograms, which are defined in the Package Body but not declared in the Package Specification. They can only be accessed by the subprograms within the Package Body. This facility is used to hide unwanted or sensitive information from users.
- A Package generally consists of functions and procedures, which are required by a specific application or a particular module of an application.
- Note that:
  - Packages variables ~ global variables
  - Functions and Procedures ~ accessible to users having access to the package
  - Private Subprograms ~ not accessible to users

Packages

- Syntax package Declaration:

```
CREATE PACKAGE Package_Name  AS
    Variable_Declaration ;
    Cursor_Declaration ;
    Function_Declaration ;
    Procedure_Declaration ;
  END Package_Name ;
```

- Where Function_Declaration is of the form

```
FUNCTION Func_Name(Param datatype,...)
        RETURN datatype1 ;
```

- Procedure_Declaration is of the form

```
PROCEDURE Proc_Name(Param {IN|OUT|IN OUT}datatype,...);
```

Packages

- Syntax of Package Body:

```
➢ CREATE PACKAGE BODY Package_Name AS
    Variable_Declaration ;
    Cursor_Declaration ;
    PROCEDURE Proc_Name(Param {IN|OUT|IN OUT} datatype,...} IS
    BEGIN
      PL/SQL_Statements ;
    END Proc_Name ;
    FUNCTION Func_Name(Param datatype,...) IS
    BEGIN
      PL/SQL_Statements ;
    END Func_Name ;
➢ END Package_Name ;
```

Example

Reliance Tech Services
Anil Dhirubhai Ambani Group

- Example :

```
CREATE OR REPLACE PACKAGE PACK1 AS
       PROCEDURE PROC1;
       FUNCTION FUN1 RETURN VARCHAR2;
     END PACK1;
```

- Package Body:

```
CREATE OR REPLACE PACKAGE BODY PACK1 AS
     PROCEDURE PROC1 IS
     BEGIN
     DBMS_OUTPUT.PUT_LINE(‘Hi a message frm prcedure’);
     END PROC1;
FUNCTION FUN1 RETURN VARCHAR2 IS
     BEGIN
     RETURN (‘HELLO FROM FUN1’);
     END FUN1;
END PACK1;
```

Example

- Executing Procedure and function from a package:
  - To execute the procedure

    ```
    EXEC PACK1.PROC1;
    ```

  - To execute the function

    ```
    SELECT PACK1.FUN1 FROM DUAL;
    ```