Ch.No.	Topic	Page No.

Chapter 30

PL/SQL FUNDAMENTALS AND ITS STATEMENTS

CHAPTER OBJECTIVES

In this chapter you will learn:

- SQL Vs PL/SQL
- Advantages and Architecture of PL/SQL
- Features of Oracle
- Structures of PL/SQL Language PL/SQL Languages Elements
- Data types and Declaration
- Variable Attributes like %TYPE and %ROWTYPE
- Control Statements like Conditional Control, Iterative Control and Sequential Control
- Conditional Control statements like IF-THEN, IF- THEN – ELSE and IF-THEN-ELSIF
- Iterative Control statements like Simple Loop, While Loop and For Loop
- Sequential Control statements like GOTO and NULL

30.1 INTRODUCTION

PL/SQL stands for Procedural Language/Structured Query Language. PL/SQL is an extension of the SQL language. We can say it is the superset of the Structured Query Language specialized for use in the Oracle database. Because it is Procedural language, it eliminates many restrictions of the SQL Language. With the use of SQL user can only manipulate the information stored into database. User can perform very basic operations such as selecting the information from some prefabricated tables, inserting information into those tables, updating the information stored in table and also occasionally used to delete information from these tables. PL/SQL extends SQL by adding control structures found in the other procedural languages. Procedural constructs blend seamlessly with Oracle SQL, resulting in a structured, powerful language. PL/SQL combines the SQL's language's ease of data manipulation and the procedural language's ease of programming.

30.2 SQL VS PL/SQL

Some of the differences between SQL and PL/SQL are highlighted and shown below:

SQL	PL/SQL
SQL does not have any procedural capabilities. By procedural capabilities here we mean that there is no provision of conditional checking, looping and branching, which are very essential for filtration of data before entering it into database.	ORACLE has provided all procedural capabilities in PL/SQL to support data filtration.
SQL statements are passed to oracle engine (server) one at a time. Therefore, for each statement a call is made to the server resources and these resources are opened and closed every time for each of the statements. And this generates network traffic, resulting in slow processing.	In PL/SQL it sends the bundle of SQL statements to the oracle server in the form of BLOCK and hence calling the server resources only once for that block even if that block is having more than one SQL statement. After processing all the statements in a block ORACLE server closes the resources results in faster execution of SQL statements in a PL/SQL block.
In SQL there is no provision of handling errors and exceptions, which means that if any SQL statement fails to execute, then oracle gives its own error messages and error code which may not be user friendly.	Where as in PL/SQL we can program the block of statements to handle the errors in such a way that if any of the statement fails to execute then we can display user-friendly appropriate messages.
SQL does not support PL/SQL statements	PL/SQL supports SQL statements in its block.
We cannot store the intermediate results of a query in variables.	PL/SQL supports declaring the variables so as to store intermediate results of query for later use. These variables can be used any where in any other SQL statement of same PL/SQL block.

30.3 ADVANTAGES OF PL/SQL

PL/SQL is a completely portable, high performance transaction processing language, which offers the following advantages:

- Supports the declaration and manipulation of object types and collections.
- Allows the calling of external functions and procedures.
- Contains new libraries of built-in packages. A package is a file that group functions, cursors, stored procedures, and variables in one place.
- Triggers: A trigger is a PL/SQL program that is stored in the database and executed immediately before or after the INSERT, UPDATE, and DELETE commands.
- Cursors: Oracle uses workspaces to execute the SQL commands. Through PL/SQL cursors, it is possible to name the workspace and access its information.
- Support for SQL: PL/SQL allows us to use all the SQL data manipulation commands, transaction control commands, SQL functions (except group functions), operators

and pseudocolumns, thus allowing us to manipulate data values in a table more flexibly and effectively.

30.4 ARCHITECTURE OF PL/SQL

The PL/SQL engine executes the PL/SQL Blocks. The PL/SQL engine executes only the procedural statements and sends the SQL statements to the SQL statement executor in the Oracle Server. The PL/SQL engine resides in the Oracle Server. The call to the Oracle engine needs to be made only once to execute any number of SQL statements, if these SQL sentences are bundle inside a PL/SQL block. Since the oracle engine is called only once for each block, resulting increased speed of processing as compared to call for each SQL sentence.

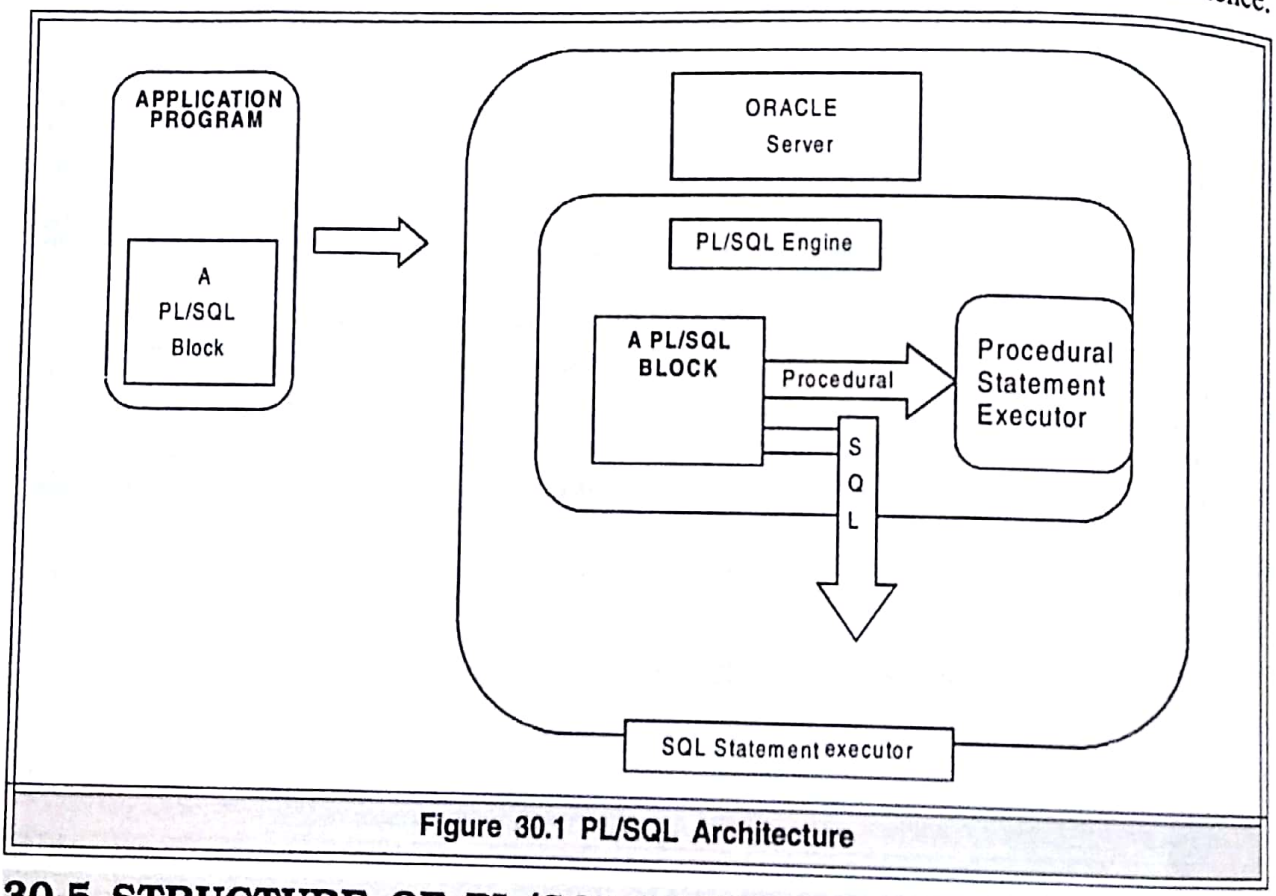

Figure 30.1 PL/SQL Architecture

30.5 STRUCTURE OF PL/SQL LANGUAGE

PL/SQL is a block structured language with procedural techniques with features like logic building, looping, error-handling mechanisms, data-types, variables, subroutines, procedural constructs. Block is smallest piece of PL/SQL code which groups logically related declarations and statements. Declarations are local to the blocks and cease to exist when block completes.

PL/SQL block consists of three sections:

Declare

- Used to declare variables and constants
- Is an optional section
- Is also used to declare type declarations, PL/SQL procedures and functions, which are local to module

Begin
- Is the executable section containing the code, which is executed when block is run
- Is compulsory

Exception
- Handles exceptions occurring during processing.
- Used to place predefined Error-handlers or user defined exceptions.
- Code contained in this section is executed only when an error occurs.
- Is an optional section

A PL/SQL statement is terminated with the END statement and a semicolon. Every PL/SQL program must consist of at least one block, which may contain any number of nested sub-blocks. Blocks can be nested in executable and exception-handling parts of a PL/SQL block or subprogram but not in declarative part.

```
Declare
        - Declare variables and constants
Begin
        - Process SQL statements
Exception
        - Error handlers
End;
```

30.6 PL/SQL LANGUAGES ELEMENTS

The PL/SQL essential language elements can be grouped as follows:

- ✦ Operators, indicators and punctuation
- ✦ Identifiers
- ✦ Comments
- ✦ Data types and Declarations
- ✦ Literals
- ✦ Expressions and comparisons

30.6.1 Operators, Indicators and Punctuation

In PL/SQL, these symbols are divided into a few groups:

Arithmetic operators

Used to perform arithmetic operations.

Examples: +, -, *, /, ** (Raises the first operand to the exponent of the second)

Expression operators

Used to create assignment, range and string catenation expressions.

Examples : =(Assignment operator, A:=3)

.. (range operator , 1..4)

|| (Concatenates two or more strings, 'dav'||'college')

30.6.2 Identifiers

Identifiers are named conventions used for variables, constants and oracle objects like tables, procedure, functions, packages, trigger and cursors etc.

30.6.3 Literals

It specifies an exact value in a program. It is an explicit numeric, character, string or Boolean value not represented by an identifier. Literals are used to initialize constants, variables and other data values.

Examples: 10,-19, 9.99 (Numeric literals)

'a', 'A' '?', ' ', ')' (Character literals)

'abc', 'jack', 'jill', '1234' (String literals)

TRUE and FALSE (Boolean literals)

30.6.4 Comments

Comments can be single line or multiple lines.

Single line comment:

Begins with – – can appear within a statement, at end of line.

Example : a number; – – variable declartion

Multi line comment

Begin with /* and end with a an asterisk-slash (*/).

Example : /* statements to select rate and quantity into variables and calculate value */

30.6.5 Expressions and Comparisons

Expressions are constructed using operands and operators. An operand is a variable, constant, literal or function call that contributes a value to an expression.

Operator Precedence

The operations within an expression are done in a order which depends on operator precedence. Operators with higher precedence are applied first, and those with same precedence are applied in no particular order. Parenthesis can be used to control the order of evaluation.

Table shows Operator Precedence:

Operator	Operation
**, NOT	Exponentiation, logical negation
+, -	Identity, negation
*, /	Multiplication, division
+, -, ||	Addition, subtraction , concatenation
=,!=,<,>,<=,>=,IS,NULL,LIKE, BETWEEN, IN, AND, OR	Conjunction Inclusion

30.6.6 Data types and Declaration

Every constant and variable has a data type, which specifies a storage format, constraints, and valid range of values.

Some commonly used data types of PL/SQL are:

- NUMBER
- VARCHAR
- BOOLEAN
- LONG RAW
- CHAR
- DATE
- LONG
- LOB

Number Type

We can use the NUMBER datatype to store fixed-point or floating-point numbers of virtually any size.

The syntax follows:

<variable_name> NUMBER[(precision,scale)];

To declare fixed-point numbers, for which you must specify scale, use the following form:

NUMBER(precision,scale)

To declare floating-point numbers, for which you cannot specify precision or scale because the decimal point can "float" to any position, use the following form:

NUMBER

To declare integers, which have no decimal point, use this form:

NUMBER(precision) – – same as NUMBER(precision,0)

How Scale Factors Affect Numeric Data Storage:

Input Data	Specified As	Stored As
1,456,123.89	NUMBER	1456123.89
1,456,123.89	NUMBER(*,1)	1456123.9
1,456,123.89	NUMBER(9)	1456124
1,456,123.89	NUMBER(9,2)	1456123.89
1,456,123.89	NUMBER(9,1)	1456123.9
1,456,123.89	NUMBER(6)	(not accepted, exceeds precision)
1,456,123.89	NUMBER(7,-2)	1456100

Character Types

Character types allow you to store alphanumeric data, represent words and text, and manipulate character strings.

CHAR

We can use the CHAR datatype to store fixed-length character data. The CHAR datatype takes an optional parameter that lets you specify a maximum length up to 32767 bytes.

The syntax follows:

<variable_name> CHAR[(maximum_length)];

We cannot use a constant or variable to specify the maximum length; you must use an integer literal in the range 1 .. 32767.

If you do not specify a maximum length, it defaults to 1.

VARCHAR2

Used to store variable length character data i.e. it stores a string upto the length of the variable unlike CHAR datatype, which stores sting variable upto the maximum length of the variable. The maximum length can be specified upto 32767 bytes. For example: If data type of address field is declared as VARCHAR(40) and address information of a particular record complete in 20 characters, then remaining 20 characters space is not padded with blank characters and memory space of 20 characters is used for some other purposes and not wasted as padded with blank characters.

LONG

The LONG datatype used to store variable-length character strings. The LONG datatype is like the VARCHAR2 datatype, except that the maximum length of a LONG value is 32760 bytes.

Syntax: <variable_name> LONG(maximum_length);

LONG columns can store text, arrays of characters, or even short documents.

BOOLEAN

The BOOLEAN datatype used to store the logical values TRUE and FALSE and the non-value NULL, which stands for a missing, inapplicable, or unknown value.

Syntax: <variable_name> BOOLEAN;

DATE

The DATE datatype used to store fixed-length date/time values. DATE values include the time of day in seconds since midnight. The default might be 'DD-MON-YY', which includes a two-digit number for the day of the month, an abbreviation of the month name, and the last two digits of the year.

Syntax : <variable_name> DATE;

RAW

The RAW datatype used to store binary data or byte strings. For example, a RAW variable might store a sequence of graphics characters or a digitized picture. Raw data is like VARCHAR2 data, except that PL/SQL does not interpret raw data.

Syntax : <variable_name> RAW(maximum_length);

The maximum width of a RAW database column is 2000 bytes.

LONG RAW

We can use the LONG RAW datatype to store binary data or byte strings. LONG RAW data is like LONG data, except that LONG RAW data is not interpreted by PL/SQL. The maximum length of a LONG RAW value is 32760 bytes.

LOB Types

The LOB (large object) datatypes BFILE, BLOB, CLOB, and NCLOB allow to store blocks of unstructured data (such as text, graphic images, video clips, and sound waveforms) up to four gigabytes in size. And, they allow efficient, random, piece-wise access to the data.

30.7 VARIABLES AND CONSTANTS

The PL/SQL language allows the declaration of variables and constants, which can be used in the SQL commands contained in the PL/SQL block. All the variables and constants used must be declared.

Variables

We can declare variables in the declaration section part and use elsewhere in the body of a PL/SQL block. The following example shows how to declare a variable

Example : age number (4) ;

A variable named age has been declared with a width of 4 bytes. Similarly we can declare a variable of Boolean data type. Example

Done Boolean ;

Variable Value Assignment

There are two ways to assign values to a variable. These are:

- With the use of assignment operator
- With the use of SELECT INTO clause to get value from the database item

With the use of assignment operator

In this the assignment operator " : =" is used to assign a value to a variable. As shown below:

```
a :          = b * c ;
increase :   = sal * 1.5 ;
OK :         = false ;
```

We can also use substitute variables for the assignment to variables. Substitute variables are those whose values are entered at run time during execution of the PL/SQL block. There is no need to declare the substitutable variables as shown below:

```
a: = &enter_number;
```

Here, enter_number is a substitute variable whose values is entered during execution, and substituted at the place of &enter_number as shown below:

Enter the value of enter_number: 3

Then, 3 is substituted at the place of &enter_number

```
a: = 3;
```

Commonly, we can use the same substitute variable name as main variable name, as shown below:

```
a: = &a;
b: = &b;
```

With the use of SELECT INTO clause

The second way to assign values to variables is to use the SELECT command to assign the contents of the fields of a table to a variable:

```
SELECT sal INTO s FROM emp where empno = 100;
```

In this case, variable 's' will get the value from sal column of emp table for empno 100.

NOTE: Select statement must return a single record; if it returns no record or more than one record then an error is raised.

Example 30.1: ***Write a PL/SQL code to calculate total sal of emp having empno 100. Table emp1 having following columns***

empno, ename, bp, da, hra, total.

Solution:

```
DECLARE
        E NUMBER(3);
        D NUMBER(3);
        H NUMBER(3);
        B NUMBER(3);
        T NUMBER(3);
BEGIN
        SELECT BP,DA, HRA INTO B, D, H FROM EMP1 WHERE EMPNO=100;
        T : =B+D+H;
        UPDATE EMP1 SET TOTAL=T WHERE EMPNO=100;
END;
```

Constant Declaration

It may be useful for you to declare constants in the declaration section of the PL/SQL blocks developed as well. Constants are named elements whose values do not change. For example, pi can be declared as a constant whose value 3.14 is never changed in the PL/SQL block.

The declaration of a constant is similar to that of declaration of a variable. We can declare constants in the declaration section and use it elsewhere in the executable part. To declare the constant we must make use of the keyword constant. This keyword must precede the data type as shown below.

```
pi constant number : = 3.14;
```

In the above example, we have assigned to the constant named pi. After this, no more assignment to the constant is allowed, i.e. 3.14 will be the initial and final value to the constant.

All the declaration must end with a semicolon (;).

30.7.1 Variable Attributes

Attributes allow us to refer to data types and objects from the database. PL/SQL variables and constants can have attributes. The following are the types of attributes, which are supported by PL/SQL.

- %TYPE
- %ROWTYPE

%TYPE : In general, the variables that deal with table columns should have the same data type and length as the column itself. %type attribute is used when declaring variables that refer to the database columns. When using the %type keyword, all you need to know is the name of the column and the table to which the variable will correspond.

Example

```
DECLARE
eno emp.empno%type; — eno of same data type and width as empno of emp table
salary emp.sal%type;
BEGIN
.....
.....
END;
```

The advantages of using %TYPE are as follows:

- We need not know the exact data type of the column empno and sal.
- If the database definition of empno and sal is changed, then, the data type of eno and salary changes accordingly at run time.

% ROWTYPE : %rowtype attribute provides a record type that represents a row in a table. For example, if the EMPLOYEE table contains four columns—EMPID, LASTNAME, FIRSTNAME, and SALARY—and you want to manipulate the values in each column of a row using only one referenced variable, the variable can be declared with the %rowtype keyword. Compare the use of %rowtype to manual record declaration:

```
DECLARE
        my_employee employee%ROWTYPE;
BEGIN
.....
END;
```

To refer empid of employee table we have to use my_employee.empid, similarly for lastname we have to use my_employee.lastname.

The other way is:

```
DECLARE
        my_empid employee.empid%TYPE;
        my_lastname employee.lastname%TYPE;
        my_firstname employee.firstname%TYPE,
        my_salary employee.salary%TYPE;
BEGIN
.....
END;
```

Example 30.2: ***Write a PL/SQL code to calculate total sal of emp having empno 100. Table emp1 having following columns***

empno, ename, bp, da, hra, total. Use %TYPE and %ROWTYPE for variable declaration.

Solution:


```
DECLARE
        B EMP1.BP%TYPE;
        D EMP1.DA%TYPE;
        H EMP1.HRA%TYPE;
        T EMP1.TOTAL%TYPE;
BEGIN
        SELECT BP,DA, HRA INTO B, D, H FROM EMP1 WHERE EMPNO=100;
        T:=B+D+H;
        UPDATE EMP1 SET TOTAL=T WHERE EMPNO=100;
END;
```

With the use of %ROWTYPE:

```
DECLARE
        REC EMP1%ROWTYPE;
BEGIN
        SELECT * INTO REC FROM EMP WHERE EMPNO=100;
        REC.TOTAL:=REC.BP+REC.HRA+REC.DA;
        UPDATE EMP1 SET TOTAL=REC.TOTAL WHERE EMPNO=100;
END;
```

30.8 DISPLAYING USER MESSAGES ON THE SCREEN

PL/SQL require a method through which messages can be displayed to the user on the monitor.

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

PUT_LINE is a procedure used to display message to the user on monitor. It accepts a single parameter of character data type. If used to display a message, it is the message 'string'.

Example:

DBMS_OUTPUT.PUT_LINE('ENTER THE NUMBER');

DBMS_OUTPUT.PUT_LINE('Result of Sum operation is: ' || sum);

To display messages to the user the SERVEROUTPUT should be set to ON. SERVEROUTPUT is a SQL*PLUS environment parameter that displays the information passed as a parameter to the PUT_LINE function.

Syntax:

SET SERVEROUTPUT [ON/OFF];

30.9 CONTROL STATEMENTS

We can change the logical flow of statements within the PL/SQL block with a number of control structures.

Control structures can be:

- Conditional Control
- Sequential Control
- Iterative Control

Conditional Control : It allows testing the truth of a condition and executing sections of program depending on the condition that may be true or false.

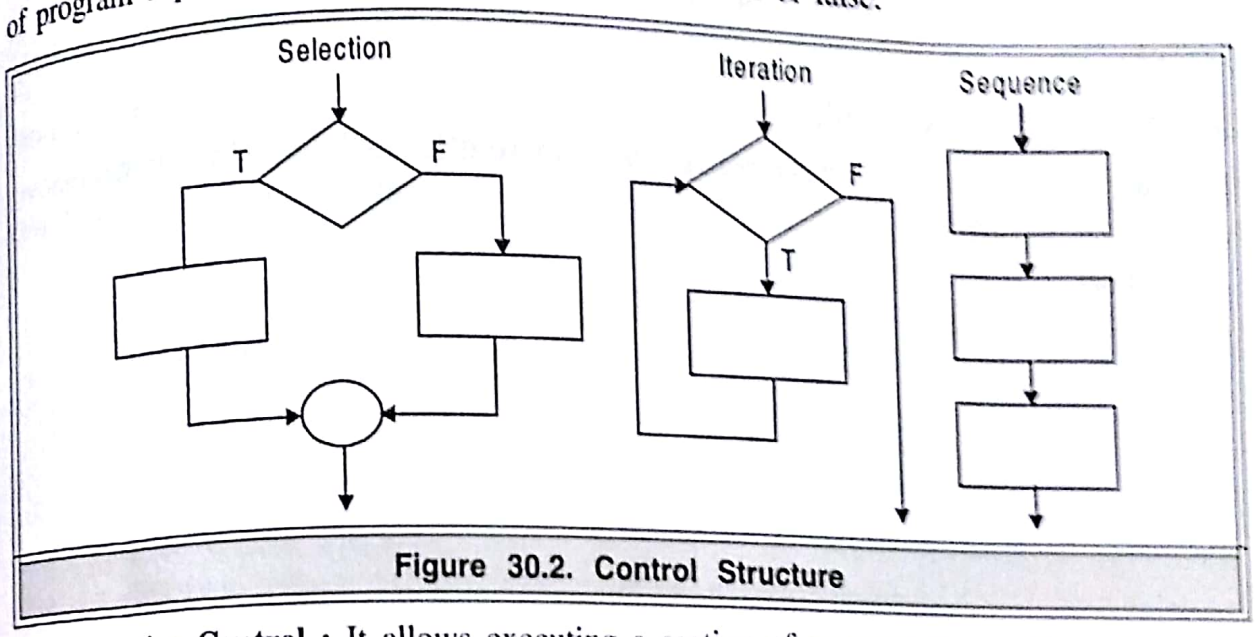

Figure 30.2. Control Structure

Iterative Control : It allows executing a section of program repeatedly as long as a specified condition remains true.

Sequential Control : It allows ordering the sequence of processing sections of program.

30.10 CONDITIONAL CONTROL

There are following conditional control statements:

- IF-THEN STATEMENT
- IF- THEN – ELSE STATEMENT
- IF-THEN-ELSIF STATEMENT (LADDER IF)

IF Statements

It is used to take alternative actions depending on circumstances. It executes a sequence of statements conditionally.

There are three forms of IF statements: IF-THEN, IF-THEN-ELSE, and IF-THEN-ELSIF.

30.10.1 IF-THEN

The simplest form of IF statement associates a condition with a sequence of statements enclosed by the keywords THEN and END IF (not ENDIF), as follows:

Syntax:

```
IF condition THEN
    sequence_of_statements;
END IF;
```

The sequence of statements is executed only if the condition yields TRUE. In either case, control passes to the next statement.

Example:

```
IF a > b THEN
    dbms_ouput.put_line('a is greater');
END IF;
```

30.10.2 IF-THEN-ELSE

The second form of IF statement adds the keyword ELSE followed by an alternative sequence of statements.

Syntax:

```
IF condition THEN
    sequence_of_statements1;
ELSE
    sequence_of_statements2;
END IF;
```

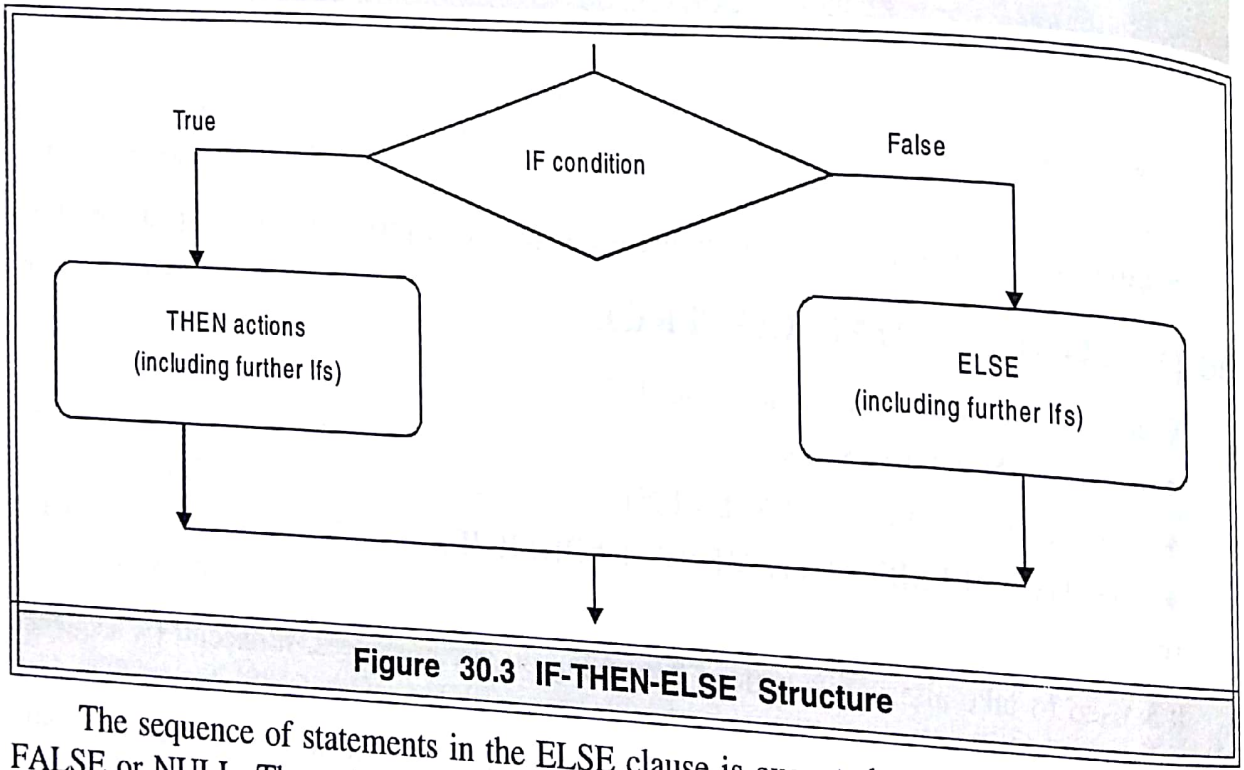

Figure 30.3 IF-THEN-ELSE Structure

The sequence of statements in the ELSE clause is executed only if the condition yields FALSE or NULL. Thus, the ELSE clause ensures that a sequence of statements is executed.

Example 30.3: ***To illustrate of If-ELSE construct PL/SQL block to find greater of two numbers. User will enter any two numbers and IF statement will check for the greater among the entered number.***

Solution:

```
DECLARE
   A NUMBER := &ENTER_A;
   B NUMBER := &ENTER_B;
```

```
BEGIN
  IF A > B THEN
  DBMS_OUTPUT.PUT_LINE(' A IS GREATER ');
  ELSE
  DBMS_OUTPUT.PUT_LINE(' B IS GREATER ');
  END IF;
END;
```


The THEN and ELSE clauses can include IF statements. That is, IF statements can be nested, as the following example shows:

Example 30.4: To illustrate of Nested If-Else construct PL/SQL block to find greatest of three numbers

Solution:

```
DECLARE
        A NUMBER := &ENTER_A;
        B NUMBER := &ENTER_B;
        C NUMBER := &ENTER_C;
BEGIN
      IF A>B THEN
          IF A>C THEN
              DBMS_OUTPUT.PUT_LINE('A IS GREATEST');
      ELSE
              DBMS_OUTPUT.PUT_LINE('C IS GREATEST');
      END IF;
ELSE
       IF B>C THEN
          DBMS_OUTPUT.PUT_LINE('B IS GREATEST');
 ELSE
          DBMS_OUTPUT.PUT_LINE('C IS GREATEST');
      END IF;
   END IF;
END;
```

30.10.3 IF-THEN-ELSIF

Sometimes we want to select an action from several mutually exclusive alternatives.

The third form of IF statement uses the keyword ELSIF (not ELSEIF) to introduce additional conditions, as follows:

Syntax:

```
IF condition1 THEN
    sequence_of_statements1;
ELSIF condition2 THEN
    sequence_of_statements2;
ELSE
    sequence_of_statements3;
END IF;
```

If the first condition yields FALSE or NULL, the ELSIF clause tests another condition. An IF statement can have any number of ELSIF clauses; the final ELSE clause is optional. Conditions are evaluated one by one from top to bottom.

If any condition yields TRUE, its associated sequence of statements is executed and control passes to the next statement. If all conditions yield FALSE or NULL, the sequence in the ELSE clause is executed. Consider the following example:

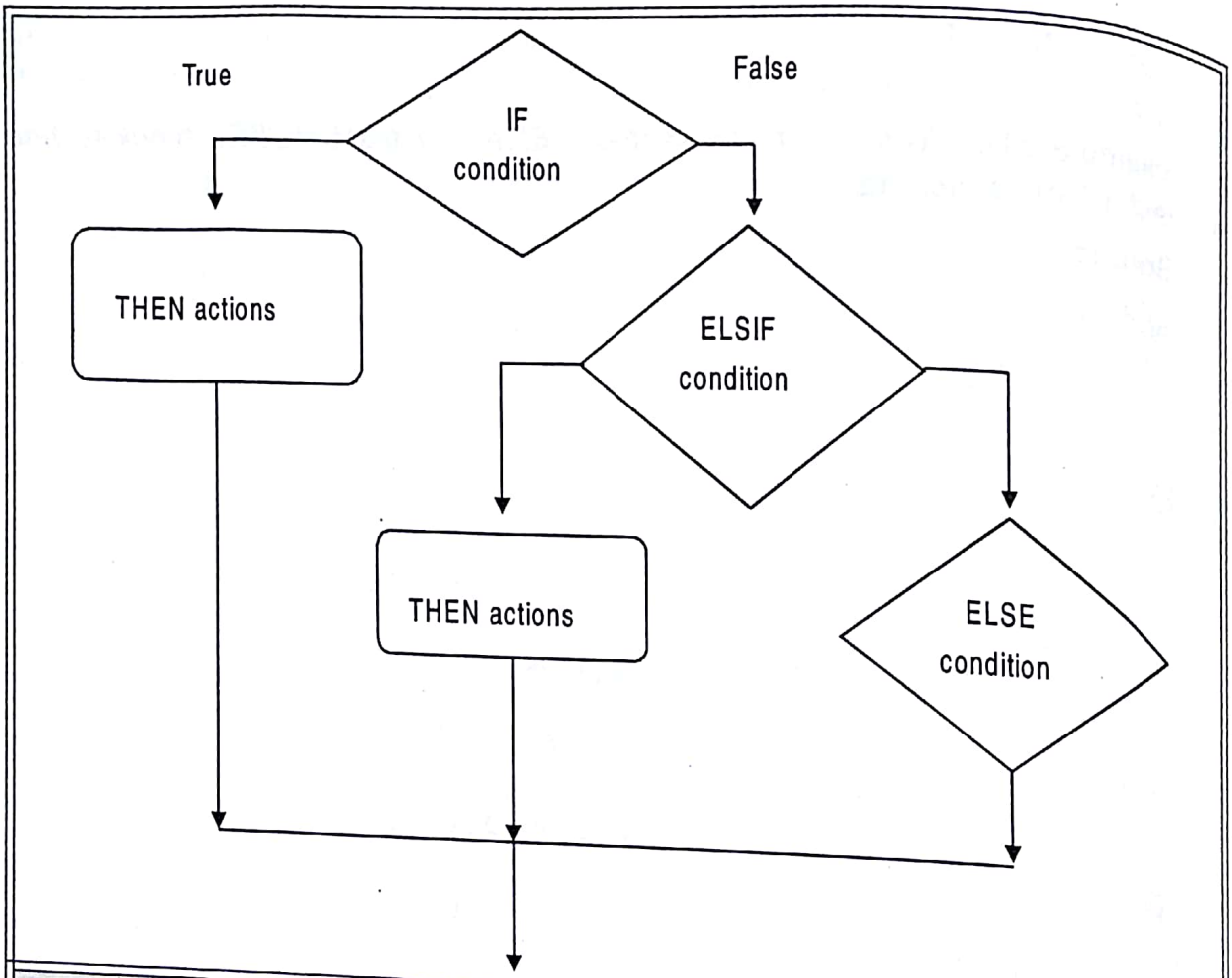

Figure 30.4 IF-THEN-ELSIF Structure

Example 30.5: ***To illustrate IF-THEN-ELSIF a PL/SQL block to calculate addition, subtraction, multiplication and division of two numbers according to user choice.***

Solution:

```
DECLARE
        A NUMBER:=&A;
        B NUMBER:=&B;
        C NUMBER;
        X NUMBER;
BEGIN
        X:=&ENTER_CHOICE;
        IF X=1 THEN
              C:=A+B;
```

```
        ELSIF X=2 THEN
        C:=A-B;
        ELSIF X=2 THEN
        C:=A-B;
        ELSIF X=3 THEN
        C:=A*B;
        ELSIF X=4 THEN
        C:=A/B;
  ELSE
      DBMS_OUTPUT.PUT_LINE('NOT A VALID OPTION');
   END IF;
DBMS_OUTPUT.PUT_LINE('RESULT IS' || C);
END;
```


Description of the PL/SQL block code

If the value of X is 1 then addition is performed, if the value of x is 2 then subtraction is performed, for X=3 multiplication is performed and for X=4 division operation is performed, otherwise not a valid option message is displayed. Instead of using four IF statements, this operation is performed with a single IF THEN ELSIF statement.

30.11 ITERATIVE CONTROL

A sequence of statements can be executed any number of times using the LOOP constructs. The LOOP command initializes a group of commands indefinitely, or until a condition forces a "break" in the LOOP and detours the execution of the program to another place. The command is used with the EXIT command, which is responsible for interrupting the LOOP execution. The LOOPS can be broadly classified into:

- ✦ Simple Loop Statement
- ✦ While Loop Statement
- ✦ For Loop Statement

30.11.1 Simple LOOP Statement

The simplest form of LOOP statement is the basic (or infinite) loop, which encloses a sequence of statements between the keywords LOOP and END LOOP, as follows:

```
LOOP
    sequence_of_statements;
END LOOP;
```

With each iteration of the loop, the sequence of statements is executed, and then control resumes at the top of the loop. If further processing is undesirable or impossible, you can use an EXIT statement to complete the loop.

There are two forms of EXIT statements:

- ✦ EXIT
- ✦ EXIT-WHEN

EXIT Statement : The EXIT statement forces a loop to complete unconditionally. When an EXIT statement is encountered, the loop completes immediately and control passes to the next statement.

Example:

```
LOOP
 ...
 IF credit_rating < 3 THEN
 ...
 EXIT; – – exit loop immediately
  END IF;
END LOOP;
...control resumes here
```


EXIT-WHEN Statement : The EXIT-WHEN statement allows a loop to complete conditionally. When the EXIT statement is encountered, the condition in the WHEN clause is evaluated. If the condition yields TRUE, the loop completes and control passes to the next statement after the loop.

An example follows:

```
LOOP
    EXIT WHEN c>5; – – exit loop if condition is true
...
END LOOP;
```

Until the condition yields TRUE, the loop cannot complete. So, statements within the loop must change the value of the condition. The EXIT-WHEN statement replaces a simple IF statement. For example, compare the following statements:

```
IF c > 5 THEN  | EXIT WHEN c > 5;
EXIT;          |
END IF;        |
```

These statements are logically equivalent, but the EXIT-WHEN statement is easier to read and understand.

Example 30.6: ***To illustrate a LOOP-EXIT a PL/SQL block to Print the numbers from 1 to N.***

Solution:

```
DECLARE
 VAR1 NUMBER :=1;
 N NUMBER :=&ENTER_N;
BEGIN
 LOOP
 DBMS_OUTPUT.PUT_LINE( VAR1);
 VAR1 := VAR1+1;
 EXIT WHEN VAR1 > N ;
 END LOOP;
END;
```

30.11.2 WHILE-LOOP Statement

The WHILE-LOOP statement associates a condition with a sequence of statements enclosed by the keywords LOOP and END LOOP, as follows:

Syntax:

```
WHILE condition LOOP
    sequence_of_statements;
END LOOP;
```

Before each iteration of the loop, the condition is evaluated. If the condition yields TRUE, the sequence of statements is executed, then control resumes at the top of the loop. If the condition yields FALSE or NULL, the loop is bypassed and control passes to the next statement. An example follows:

```
WHILE i <= 10 LOOP
 a:=n*i;
 i:=i+1;
END LOOP;
```

The number of iterations depends on the condition and is unknown until the loop completes. Since the condition is tested at the top of the loop, the sequence might execute zero times. In the last example, if the initial value of i is greater than 10, the condition yields FALSE and the loop is bypassed. To ensure that a WHILE loop executes at least once, use an initialized Boolean variable. A statement inside the loop must assign a new value to the Boolean variable. Otherwise, you have an infinite loop.

Example 30.7: ***To illustrate a PL/SQL block to Print the desired multiplication table.***

Solution:

```
DECLARE
        TABLE_OF NUMBER :=&ENTER_TABLEOF ;
        COUNT NUMBER := 1;
        RESULT NUMBER;
BEGIN
        WHILE COUNT <= 10
LOOP
        RESULT = TABLE_OF * COUNT ;
        DBMS_OUTPUT.PUT_LINE(TABLE_OF|| ' * '||COUNT || ' = '||RESULT);
        COUNT := COUNT +1 ;
    END LOOP;
END;
```

Example 30.8: ***To illustrate of WHILE LOOP with the use of SQL statements PL/SQL block to calculate the SIMPLE INTEREST for same PRINCIPAL and TIME but with different RATE OF INTEREST starting from 5% and to store each result in a Simple_Interest table with following structure: (Principal, Rate, Time, Si).***

Solution:

```
DECLARE
        PRINCIPAL       NUMBER := &ENTER_P;
        RATE_OF_INT NUMBER := 5 ;
        TIME_IN_YEAR    NUMBER := 2;
        SIMPLE_INT      NUMBER(6,2);
BEGIN
        WHILE RATE_OF_INT < =15
        LOOP
         SIMPLE_INT := (PRINCIPAL*RATE_OF_INT*TIME_IN_YEAR)/100;
         INSERT INTO SIMPLE_INTEREST(PRINCIPAL,RATE,TIME,SI)
         VALUES(PRINCIPAL,RATE_OF_INT,TIME_IN_YEAR,SIMPLE_INT);
         RATE_OF_INT := RATE_OF_INT + 1 ;
 END LOOP;
END;
```


30.11.3 FOR-LOOP Statement

Whereas the number of iterations through a WHILE loop is unknown until the loop completes, the number of iterations through a FOR loop is known before the loop is entered. FOR loops iterate over a specified range of integers. The range is part of an iteration scheme, which is enclosed by the keywords FOR and LOOP. A double dot (..) serves as the range operator. The syntax follows:

Syntax:

```
FOR counter IN [REVERSE] lower_bound..higher_bound LOOP
    sequence_of_statements;
END LOOP;
```

The range is evaluated when the FOR loop is first entered and is never re-evaluated. As the next example shows, the sequence of statements is executed once for each integer in the range. After each iteration, the loop counter is incremented.

```
FOR i IN 1..3 LOOP – – assign the values 1,2,3 to I
    sequence_of_statements; – – executes three times
END LOOP;
```

The following example shows that if the lower bound equals the higher bound, the sequence of statements is executed once:

```
FOR i IN 3..3 LOOP – – assign the value 3 to i
    sequence_of_statements; – – executes one time
END LOOP;
```

FOR-LOOP in Reverse Order

By default, iteration proceeds upward from the lower bound to the higher bound. However, if you use the keyword REVERSE, iteration proceeds downward from the higher bound to

the lower bound, as the below example shows. After each iteration, the loop counter is decremented.

```
FOR i IN REVERSE 1..3 LOOP – – assign the values 3,2,1 to i
    sequence_of_statements; – – executes three times
END LOOP;
```


NOTE: However, the loop counter increment (or decrement) must be 1. Some languages provide a STEP clause, which lets you specify a different increment.

PL/SQL has no such structure, but you can easily build one. Consider the following example:

```
FOR j IN 5..15 LOOP          – – assign values 5,6,7,... to j
IF MOD(j, 5) = 0 THEN        – – pass multiples of 5
sequence_of_statements;      – – j has values 5,10,15
END IF;
END LOOP;
```

This loop is logically equivalent to one, which has step value of 5. Within the sequence of statements, the loop counter has only the values 5, 10, and 15.

Scope Rules

The loop counter is defined only within the loop. You cannot reference it outside the loop. After the loop is exited, the loop counter is undefined, as the following example shows:

```
FOR ctr IN 1..10 LOOP
    ...
END LOOP;
sum := ctr - 1; – – illegal
```

We need not explicitly declare the loop counter because it is implicitly declared as a local variable of type INTEGER.

Using the EXIT Statement

The EXIT statement allows a FOR loop to complete prematurely. For example, the following loop normally executes ten times, but as soon as the b becomes greater than 999, the loop completes no matter how many times it has executed:

```
FOR i IN 1..10 LOOP
 b:=a*i;
EXIT WHEN b>999;
 ...
END LOOP;
```

30.12 SEQUENTIAL CONTROL

GOTO statement and NULL are commonly used to change the sequence of statements.

30.12.1 GOTO Statement

The GOTO statement branches to a label unconditionally. The label must be unique within its scope and must precede an executable statement or a PL/SQL block. When

executed, the GOTO statement transfers control to the labeled statement or block. In the following example, you go to an executable statement farther down in a sequence of statements:


```
BEGIN
    ...
    GOTO insert_row;
    ...
    <<insert_row>>
    INSERT INTO emp VALUES ...
END;
```

The GOTO statement must be followed by an executable statement.

For example:

```
LOOP
.....
.......
IF A>B then
 Go to <<abc>>;
END IF;
.......
......
<<abc> – – it is illegal because it does not precede the executable statement.
END LOOP:
End;
```

30.12.2 NULL Statement

The NULL statement explicitly specifies inaction; it does nothing other than pass control to the next statement.

Uses of NULL Statement:

It can improve readability.

```
IF A>B THEN
  DBMS_OUTPUT.PUTLINE(A);
ELSE
  NULL;
END IF;
```

The solution of illegal use of GOTO statement is with the NULL statement as shown below:

```
LOOP
.....
      IF A>B THEN
          GO TO <<ABC>>;
      END IF;
      .......
      <<abc>> /* It is correct now because it is followed by an executable statement
                 NULL without changing the meaning of the block.*/
      NULL;
END LOOP;
```

Flash Back

SQL stands for Procedural Language/SQL. PL/SQL is an extension of the SQL language. We can say it is the superset of the Structured Query Language specialized for use in the Oracle database. Because it is Procedural language, it eliminates many restrictions of the SQL Language. PL/SQL is a completely portable, high performance transaction processing language, which Supports the declaration and manipulation of object types and collections, external functions and procedures, packages, Triggers and Cursors.

The PL/SQL engine executes the PL/SQL Blocks. The PL/SQL engine executes only the procedural statements and sends the SQL statements to the SQL statement executor in the Oracle Server. PL/SQL block consists of three sections:

Declare
– Declare variables and constants

Begin
– Process SQL statements

Exception
– Error handlers

End;

The PL/SQL essential language elements can be grouped as follows :

- Operators, indicators and punctuation
- Literals
- Expressions and comparisons
- Identifiers
- Comments

Data types and Declarations Some commonly used data types of PL/SQL are:

- NUMBER
- VARCHAR
- BOOLEAN
- LONG RAW
- CHAR
- DATE
- LONG
- LOB

The following are the types of attributes, which are supported by PL/SQL.

- %TYPE
- %ROWTYPE

%type attribute is used when declaring variables that refer to the database columns and %rowtype attribute provides a record type that represents a row in a table. DBMS_OUPUT.PUT_LINE is used to Display user Messages on the Screen.

Control structures can be:

- Conditional Control
- Sequential Control
- Iterative Control

Conditional Control: It allows testing the truth of a condition and executing sections of program depending on the condition that may be true or false.

Iterative Control: It allows executing a section of program repeatedly as long as a specified condition remains true.

Sequential Control: It allows ordering the sequence of processing sections of program.

There are following conditional control statements:

- IF-THEN STATEMENT
- IF- THEN – ELSE STATEMENT
- IF-THEN-ELSIF STATEMENT (LADER IF)

Iterative Control: A sequence of statements can be executed any number of times using the LOOP constructs. The LOOPS can be broadly classified into:

- Simple Loop Statement
- While Loop Statement
- For Loop Statement

Sequential Control: GOTO statement and NULL are commonly used to change the sequence of statements. The GOTO statement branches to a label unconditionally. The NULL statement explicitly specifies inaction; it does nothing other than pass control to the next statement.

REVIEW QUESTIONS

1. What are the problems of SQL and how they are solved by PL/SQL?
2. What are the advantages of PL/SQL?
3. Explain the architecture of PL/SQL?
4. What are the features of Oracle?
5. What are the main blocks of a PL/SQL code? Explain the importance of each?
6. What is operator precedence in PL/SQL?
7. How can we issue comments in PL/SQL code?
8. What are the data type of PL/SQL? Explain with examples.
9. How can we declare variables and constants?
10. How can we assign the values to PL/SQL variables? Explain with examples.
11. How can we display the output in PL/SQL?
12. What are the Control Statements in PL/SQL? Explain with examples.
13. What are the Conditional control statements in PL/SQL? Explain with examples.
14. What are the Looping statements in PL/SQL? Explain with examples.
15. What are the Sequential statements in PL/SQL? Explain with examples.
16. Write a PL/SQL code to find the factorial of any number.
17. Write a PL/SQL code to check even or odd of a number.
18. Write a PL/SQL code to calculate the Simple Interest.
19. Write a PL/SQL code to calculate the Electricity bill.
20. Write a PL/SQL code to calculate the Net Salary of an employee.
21. Write a PL/SQL code to calculate the Telephone bill and store the result in a database.

HANDS ON SESSION

1. PL/SQL is a _____ of SQL statements
   (a) Subset (b) Superset (c) Powerset (d) None of these
2. A PL/SQL statement is terminated with
   (a) END statement (b) STOP Statement
   (c) BREAK statement (d) None

3. The VARCHAR2 datatype is used to store
   (a) Variable length character data (b) Fixed length character data
   (c) Depends (d) a & b
4. Which operator is used for declaring variables that refer to the database columns
   (a) + (b) % (c) – (d) *
5. % ROWTYPE attribute provides a record type that represents
   (a) Row in a Table (b) Columns in a Table
   (c) Field in a Table (d) All
6. The two forms of EXIT statements
   (a) EXIT & EXIT WHY (b) EXIT WHEN & EXIT
   (c) EXIT HOW & EXIT (d) All
7. Which statement are used to change the sequence of statements
   (a) GOTO (b) NULL (c) None of these (d) Both a & b
8. LONGRAW datatype is used to store
   (a) Char Data & strings (b) Binary Data & strings
   (c) int Data & strings (d) float Data & strings
9. Which of the following is optional in PL/SQL Block
   (a) Begin (b) Declare (c) Exception (d) End
   (e) b ,c (f) None
10. Which of the following is having problem
    (a) temp1, temp2 number(5,2) ; (b) temp1 number(5,2), temp2 number(5,2);
    (c) temp1 number(5,2); temp2 number(5,2); (d) None of the above has problem
    (e) All of above has problem
11. Which of the following is not an oracle Data type
    (a) CHAR (b) NCHAR (c) VARCHAR (d) VARCHAR2
    (e) UROWID (f) TIME
12. Which of the following is not a right way of comments
    (a) start with double hyphen – (b) start with /* and end with */
    (c) Start with /* and end with /* (d) Start with */ and end with */
13. Which of the following is not a Oracle reserve word.
    (a) EXCLUSIVE (b) INCREMENT (c) CASE (d) PRIOR
14. Which of the following are the variable attributes
    (a) %type (b) %Rowtype (c) %rowcount (d) %count
15. Which of the following is the assignment operator in Oracle
    (a) = (b) := (c) == (d) none
16. DBMS_OUTPUT is a
    (a) Function (b) Procedure
    (c) User defined Package (d) System Supplied Package
17. What is default format of DATE type data in Oracle
    (a) dd-mon-yyyy (b) dd/mm/yyyy (c) mon-dd-yy (d) yyyy-dd-mon

18. Which of the following is a correct value that can be stored in varible declared as below
A number(5,1)
(a) 34567.2 (b) 3456.4 (c) 3434.34 (d) 1.3232

19. Which section of the PL/SQL block handles errors and abnormal conditions?
(a) Declaration (b) Exception (c) Executable (d) Anonymous block

20. What is the mandatory clause in a select statement when used inside a PL/SQL block.
(a) INTO (b) Where (c) Order By (d) Group by

21. In which section of PL/SQL block is a constant assigned value?
(a) Executable (b) declaration
(c) Exception (d) declaration or executable sections.

22. Why does the following statement in the declaration section fail :
PRODUCT_IN_STOCK BOOLEAN: = 'TRUE';
(a) Assignment operation is not permitted in the declaration section.
(b) The size/width for the variable is not defined.
(c) Boolean is not a valid Data type supported by PL/SQL.
(d) A Boolean variable cannot be assigned a character string value.

23. What type of constant can be defined when you declare a variable?
(a) Check (b) Not Null
(c) Check and Not Null (d) no constraint

24. Choose two answers that are true. A variable is defined as %type.
(a) if the underlying table column data type changes, PL/SQL code needs to be changed.
(b) You do not have to know the data type or the precision of the column.
(c) Only character variables can be defined using %type.
(d) You need not be concerned about change that may be made to column definitions

Solution Keys

1. (b)	2. (a)	3. (a)	4. (b)	5. (a)	6. (b)	7. (d)	8. (a&b)	9. (e)
10. (c&d)	11. (f)	12. (c&d)	13. (c)	14. (a&b)	15. (b)	16. (d)	17. (a)	18. (b)
19. (b)	20. (a)	21. (b)	22. (d)	23. (b)	24. (b&d)			

●●●