PL-SQL

The PL/SQL programming language was developed by Oracle Corporation in the late 1980s as procedural extension language for SQL and the Oracle relational database. Following are notable facts about PL/SQL:

- PL/SQL is a completely portable, high-performance transaction-processing language.
- PL/SQL provides a built-in interpreted and OS independent programming environment.
- PL/SQL can also directly be called from the command-line SQL*Plus interface.
- Direct call can also be made from external programming language calls to database.
- PL/SQL's general syntax is based on that of ADA and Pascal programming language.
- Apart from Oracle, PL/SQL is available in TimesTen in-memory database and IBM DB2.

Features of PL/SQL

PL/SQL has the following features:

- PL/SQL is tightly integrated with SQL.
- It offers extensive error checking.
- It offers numerous data types.
- It offers a variety of programming structures.
- It supports structured programming through functions and procedures.
- It supports object-oriented programming.
- It supports developing web applications and server pages.

Advantages of PL/SQL

PL/SQL has the following advantages:

- SQL is the standard database language and PL/SQL is strongly integrated with SQL. PL/SQL supports both static and dynamic SQL. Static SQL supports DML operations and transaction control from PL/SQL block. Dynamic SQL is SQL allows embedding DDL statements in PL/SQL blocks.
- PL/SQL allows sending an entire block of statements to the database at one time. This reduces network traffic and provides high performance for the applications.
- PL/SQL gives high productivity to programmers as it can query, transform, and update data in a database.
- PL/SQL saves time on design and debugging by strong features, such as exception handling, encapsulation, data hiding, and object-oriented data types.
- Applications written in PL/SQL are fully portable.
- PL/SQL provides high security level.
- PL/SQL provides access to predefined SQL packages.
- PL/SQL provides support for Object-Oriented Programming.
- PL/SQL provides support for Developing Web Applications and Server Pages.

BASIC PROGRAM STRUCTURE

PL/SQL is a block-structured language, meaning that PL/SQL programs are divided and written in logical blocks of code. Each block consists of three sub-parts:

S.N.	Sections & Description
1	**Declarations** This section starts with the keyword **DECLARE**. It is an optional section and defines all variables, cursors, subprograms, and other elements to be used in the program.
2	**Executable Commands** This section is enclosed between the keywords **BEGIN** and **END** and it is a mandatory section. It consists of the executable PL/SQL statements of the program. It should have at least one executable line of code, which may be just a NULL command to indicate that nothing should be executed.
3	**Exception Handling** This section starts with the keyword **EXCEPTION**. This section is again optional and contains exception(s) that handle errors in the program.

Every PL/SQL statement ends with a semicolon **(;)**. PL/SQL blocks can be nested within other PL/SQL blocks using **BEGIN** and **END**. Here is the basic structure of a PL/SQL block:

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

The PL/SQL Delimiters

A delimiter is a symbol with a special meaning. Following is the list of delimiters in PL/SQL:

Delimiter	Description
+, -, *, /	Addition, subtraction/negation, multiplication, division
%	Attribute indicator
'	Character string delimiter

.	Component selector
(,)	Expression or list delimiter
:	Host variable indicator
,	Item separator
"	Quoted identifier delimiter
=	Relational operator
@	Remote access indicator
;	Statement terminator
:=	Assignment operator
=>	Association operator
||	Concatenation operator
**	Exponentiation operator
<<, >>	Label delimiter (begin and end)
/*, */	Multi-line comment delimiter (begin and end)
--	Single-line comment indicator
..	Range operator
<, >, <=, >=	Relational operators
<>, '=, ~=, ^=	Different versions of NOT EQUAL

The PL/SQL Comments

Program comments are explanatory statements that you can include in the PL/SQL code that you write and helps anyone reading its source code. All programming languages allow for some form of comments.

The PL/SQL supports single-line and multi-line comments. All characters available inside any comment are ignored by PL/SQL compiler. The PL/SQL single-line comments start with the delimiter **--**(double hyphen) and multi-line comments are enclosed by /* and */.

```
DECLARE
   -- variable declaration
   message  varchar2(20):= 'Hello, World!';
BEGIN
   /*
    *  PL/SQL executable statement(s)
    */
   dbms_output.put_line(message);
END;
```

PL/SQL Scalar Data Types and Subtypes

PL/SQL Scalar Data Types and Subtypes come under the following categories:

Date Type	Description
Numeric	Numeric values on which arithmetic operations are performed.
Character	Alphanumeric values that represent single characters or strings of characters.
Boolean	Logical values on which logical operations are performed.
Datetime	Dates and times.

PL/SQL provides subtypes of data types. For example, the data type NUMBER has a subtype called INTEGER. You can use subtypes in your PL/SQL program to make the data types compatible with data types in other programs while embedding PL/SQL code in another program, such as a Java program.

PL/SQL Numeric Data Types and Subtypes

Following is the detail of PL/SQL pre-defined numeric data types and their sub-types:

Data Type	Description
PLS_INTEGER	Signed integer in range -2,147,483,648 through 2,147,483,647, represented in 32 bits
BINARY_INTEGER	Signed integer in range -2,147,483,648 through 2,147,483,647, represented in

	32 bits
BINARY_FLOAT	Single-precision IEEE 754-format floating-point number
BINARY_DOUBLE	Double-precision IEEE 754-format floating-point number
NUMBER(prec, scale)	Fixed-point or floating-point number with absolute value in range 1E-130 to (but not including) 1.0E126. A NUMBER variable can also represent 0.
DEC(prec, scale)	ANSI specific fixed-point type with maximum precision of 38 decimal digits.
DECIMAL(prec, scale)	IBM specific fixed-point type with maximum precision of 38 decimal digits.
NUMERIC(pre, secale)	Floating type with maximum precision of 38 decimal digits.
DOUBLE PRECISION	ANSI specific floating-point type with maximum precision of 126 binary digits (approximately 38 decimal digits)
FLOAT	ANSI and IBM specific floating-point type with maximum precision of 126 binary digits (approximately 38 decimal digits)
INT	ANSI specific integer type with maximum precision of 38 decimal digits
INTEGER	ANSI and IBM specific integer type with maximum precision of 38 decimal digits
SMALLINT	ANSI and IBM specific integer type with maximum precision of 38 decimal digits
REAL	Floating-point type with maximum precision of 63 binary digits (approximately 18 decimal digits)

PL/SQL Character Data Types and Subtypes

Following is the detail of PL/SQL pre-defined character data types and their sub-types:

Data Type	**Description**
CHAR	Fixed-length character string with maximum size of 32,767 bytes

VARCHAR2	Variable-length character string with maximum size of 32,767 bytes
RAW	Variable-length binary or byte string with maximum size of 32,767 bytes, not interpreted by PL/SQL
NCHAR	Fixed-length national character string with maximum size of 32,767 bytes
NVARCHAR2	Variable-length national character string with maximum size of 32,767 bytes
LONG	Variable-length character string with maximum size of 32,760 bytes
LONG RAW	Variable-length binary or byte string with maximum size of 32,760 bytes, not interpreted by PL/SQL
ROWID	Physical row identifier, the address of a row in an ordinary table
UROWID	Universal row identifier (physical, logical, or foreign row identifier)

PL/SQL Boolean Data Types

The **BOOLEAN** data type stores logical values that are used in logical operations. The logical values are the Boolean values TRUE and FALSE and the value NULL.

However, SQL has no data type equivalent to BOOLEAN. Therefore, Boolean values cannot be used in:

- SQL statements
- Built-in SQL functions (such as TO_CHAR)
- PL/SQL functions invoked from SQL statements

PL/SQL Datetime and Interval Types

The **DATE** datatype to store fixed-length datetimes, which include the time of day in seconds since midnight. Valid dates range from January 1, 4712 BC to December 31, 9999 AD.

The default date format is set by the Oracle initialization parameter NLS_DATE_FORMAT. For example, 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, for example, 01-OCT-12.

Each DATE includes the century, year, month, day, hour, minute, and second. The following table shows the valid values for each field:

Field Name	**Valid Datetime Values**	**Valid Interval Values**
YEAR	-4712 to 9999 (excluding year 0)	Any nonzero integer

MONTH	01 to 12	0 to 11
DAY	01 to 31 (limited by the values of MONTH and YEAR, according to the rules of the calendar for the locale)	Any nonzero integer
HOUR	00 to 23	0 to 23
MINUTE	00 to 59	0 to 59
SECOND	00 to 59.9(n), where 9(n) is the precision of time fractional seconds	0 to 59.9(n), where 9(n) is the precision of interval fractional seconds
TIMEZONE_HOUR	-12 to 14 (range accommodates daylight savings time changes)	Not applicable
TIMEZONE_MINUTE	00 to 59	Not applicable
TIMEZONE_REGION	Found in the dynamic performance view V$TIMEZONE_NAMES	Not applicable
TIMEZONE_ABBR	Found in the dynamic performance view V$TIMEZONE_NAMES	Not applicable

PL/SQL Large Object (LOB) Data Types

Large object (LOB) data types refer large to data items such as text, graphic images, video clips, and sound waveforms. LOB data types allow efficient, random, piecewise access to this data. Following are the predefined PL/SQL LOB data types:

Data Type	Description	Size
BFILE	Used to store large binary objects in operating system files outside the database.	System-dependent. Cannot exceed 4 gigabytes (GB).
BLOB	Used to store large binary objects in the database.	8 to 128 terabytes (TB)

CLOB	Used to store large blocks of character data in the database.	8 to 128 TB
NCLOB	Used to store large blocks of NCHAR data in the database.	8 to 128 TB

NULLs in PL/SQL

PL/SQL NULL values represent missing or unknown data and they are not an integer, a character, or any other specific data type. Note that NULL is not the same as an empty data string or the null character value '\0'. A null can be assigned but it cannot be equated with anything, including itself.

The 'Hello World' Example:

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

OR

Dbms_output.put_line('hello world');
```

NOTE :-- To display output message , the SERVEROUTPUT should be set ON .

Syntax :

SET SERVEROUTPUT ON;

Q : WAP To add 2 numbers .

Declare

A number(2);

B number(2);

C number(2);

```
Begin
        A := 10;
        B:= 20;
        C := A + B;
        Dbms_output.put_line('sum of '||  A ||  ' and ' ||  B  || ' is ' ||   C);
End;
```

Q : WAP To add 2 numbers given by the user

```
Declare
        A number(2);
        B number(2);
        C number(2);
Begin
        A := &A;
        B:= &B;
        C := A + B;
        Dbms_output.put_line('sum of '||  A ||  ' and ' ||  B  || ' is ' ||   C);
End;
```

WAP code to calculate total amount (TA+DA) of an employee (EID= 20) , Also update the amount in the table .

Emp (EID, Ename, TA,DA,total)

```
Declare
        a number(10);
        b number (10);
        c number(10);
Begin
        Select TA,DA into a,b from Emp where EID =20;
        C := a+b;
        Update Emp set total = c where EID = 20;
End;
```

%TYPE

FOR acquiring the same datatype for a declared variable as that was in the database.

Syntax :

V_name TABLENAME.COLUMNNAME%TYPE,

Example :

```
Declare  A Emp.TA%type;
```

CONDITIONAL STATEMENTS

IF – THEN
IF – THEN – ELSE
IF – THEN –ELSIF

WAP to find the largest number

```
Declare
        A number(2);
Begin   B number(2);
        A := &A;
        B := &B;
        If A > B THEN
                DBMS_OUTPUT.PUT_LINE('A IS LARGER');
        ELSE
                DBMS_OUTPUT.PUT_LINE('B IS LARGER');
        END IF;
END;
```

WAP TO CHECK EVEN OR ODD

```
Declare
        A number (2);
Begin
        A = &A;
        If A mod 2 = 0 then
                Dbms_output.put_line('even');
        Else
                Dbms_output.put_line('odd');
        End if;
End;
```

LOOPING

- **LOOP**
- **WHILE-LOOP**
- **FOR- LOOP**

WAP display first 10 natural numbers .

```
Declare
        i number(2);

begin

        i := 1;

        LOOP
                Dbms_output.put_line(i);
                i:=i+1;
                Exit when i > 10;
        END LOOP;
END;
```

Q – wap to print square of numbers from 1 to 10;
Q – WAP to find the sum of first 10 numbers
Q – WAP to print the table of 2.
Q – WAP to find the factorial

FOR loop

```
Declare
        A number (4);
        i number(4);
begin
        for i in 1..10
        loop
                A : = 2 * i ;
        Dbms_output.put_line(‘ 2  *  ’ || i  || ‘ = ’ || A  );

End loop;

End;
```

For reverse

```
for i in reverse 1..10
```

WHILE LOOP :

```
DECLARE
   a number(2) := 10;
BEGIN
   WHILE a < 20 LOOP
      dbms_output.put_line('value of a: ' || a);
      a := a + 1;
   END LOOP;
END;
```