PL/SQL Advantages

#### **What is PL/SQL?**

PL/SQL is a Procedural Language extension of **Structured Query Language (SQL)**.  
PL/SQL is **specially designed** for Database oriented activities. Oracle PL/SQL allows you to perform data manipulation operation those are **safe and flexible**.

PL/SQL is a very secure **functionality tool** for manipulating, controlling, validating, and restricting unauthorized access data from the SQL database.

Using PL/SQL we can improve **application performance**. It also allows to **deal** with **errors** so we can provide **user friendly error messages**.

PL/SQL have a **great functionality** to display multiple records from the multiple tables at the same time.

PL/SQL is **capable** to **send** entire block of statements and execute it in the **Oracle engine** at once.

Advantages PL/SQL

Procedural language support : PL/SQL is a **development tools** not only for data manipulation futures but also provide the conditional checking, looping or branching operations same as like **other programming language**.

Reduces network traffic : This one is **great advantages** of PL/SQL. Because PL/SQL nature is **entire block** of SQL statements execute into **oracle engine** all at once so it's main benefit is **reducing** the **network traffic**.

Error handling : PL/SQL is dealing with **error handling**, It's permits the smart way **handling the errors** and giving**user friendly** error messages, when the errors are encountered.

Declare variable : PL/SQL gives you control to **declare variables** and access them **within the block**. The declared variables can be used at the time of **query processing**.

Intermediate Calculation : **Calculations** in PL/SQL done quickly and efficiently without using Oracle engines. This**improves** the transaction performance.

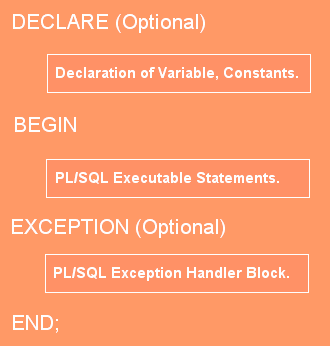
Portable application : Applications are written in PL/SQL are **portable** in any **Operating system**. PL/SQL applications are **independence program** to run any computer.

### PL/SQL Block Structure

PL/SQL is block structured language divided into three logical blocks.

BEGIN block and END; keyword are compulsory, and other two block DECLARE and EXCEPTION are optional block. END; is not a block only keyword to end of PL/SQL program.

PL/SQL block structure follows divide-and-conquer approach to solve the problem stepwise.



#### **Figure - PL/SQL block Structure**

### DECLARE

Variables and constants are declared, initialized within this section.

Variables and Constants : In this block, declare and initialize variables (and constants). You must have to declare variables and constants in declarative block before referencing them in procedural statement.

Declare Variables and Assigning values : You can define variable name, data type of a variable and its size. Date type can be: CHAR, VARCHAR2, DATE, NUMBER, INT or any other.

DECLARE -- DECLARE block, declare and initialize values

designation VARCHAR2(30);

eno number(5) := 5;

id BOOLEAN;

inter INTERVAL YEAR(2) TO MONTH;

BEGIN -- BEGIN block, also assign values

designation := UPPER('Web Developer');

id := TRUE;

inter := INTERVAL '45' YEAR;

END;

/

Declare Constants and Assigning values : Constants are declared same as variable but you have to add the CONSTANT keyword before defining data type. Once you define a constant value you can't change the value.

designation CONSTANT VARCHAR2(30) := 'Web Developer';

### BEGIN

BEGIN block is procedural statement block which will implement the actual programming logic. This section contains conditional statements (if...else), looping statements (for, while) and Branching Statements (goto) etc.

### EXCEPTION

PL/SQL easily detects user defined or predefined error condition. PL/SQL is famous for handling errors in smart way by giving suitable user friendly messages. Errors can be rise due to wrong syntax, bad logical or not passing a validation rules.

You can also define exception in your declarative block and later you can execute it by RAISE statement.

DECLARE

check\_exist EXCEPTION; -- declare exception type

...

BEGIN

....

RAISE check\_exist; -- Raise exception

....

EXCEPTION

WHEN check\_exist THEN -- execute raise exception

.....

END;

/

Note :

1. BEGIN block and END; keyword are compulsory of any PL/SQL program.
2. Where as DECLARE and EXCEPTION block are optional.

### PL/SQL Data Types - Oracle

As you know about the variables and constants stores value in specific storage format. There are six built-in PL/SQL Data types

1. [Scalar data types](http://www.way2tutorial.com/plsql/plsql_data_types.php#scalartypes) - Scalar data types haven't internal components.
2. Composite data types - Composite data types have internal components to manipulate data easily.
3. Reference data types - This data types works like a pointer to hold some value.
4. [LOB data types](http://www.way2tutorial.com/plsql/plsql_data_types.php#lobtypes) - Stores large objects such as images, graphics, video.
5. [Unknown Column types](http://www.way2tutorial.com/plsql/plsql_data_types.php#unknowntype) - Identify columns when not know type of column.
6. [User Define data types](http://www.way2tutorial.com/plsql/plsql_data_types.php#userdefine) - Define your own data type that are inherit from predefined base data type.

### Scalar types

Scalar data type haven't internal components. It is like a linear data type. Scales data type divides into four different types character, numeric, boolean or date/time type.

#### **Numeric Data types**

Following are numeric data types in PL/SQL:

|  |  |
| --- | --- |
| Data types | Description, Storage(Maximum) |
| NUMBER(p,s) | NUMBER data type used to store numeric data. It's contain letters, numbers, and special characters. Storage Range : Precision range(p) : 1 to 38 and Scale range(s) : -84 to 127 NUMBER Subtypes : This sub type defines different types storage range.   |  |  |  | | --- | --- | --- | | Sub Data types | Maximum Precision | Description | | INTEGER | 38 digits | This data types are used to store fixed decimal points. You can use based on your requirements. | | INT | 38 digits | | SMALLINT | 38 digits | | DEC | 38 digits | | DECIMAL | 38 digits | | NUMERIC | 38 digits | | REAL | 63 binary digits | | DOUBLE PRECISION | 126 binary digits | | FLOAT | 126 binary digits | | |
| BINARY\_INTEGER | BINARY\_INTEGER data type store signed integer's value. Note : BINARY\_INTEGER values require less storage space compare of NUMBER data type values. Storage Range : from -2147483647 to 2147483647 BINARY\_INTEGER Subtypes : This sub type define constraint to store a value.   |  |  | | --- | --- | | Sub Data types | Description | | NATURAL | NATURAL/POSITIVE data type prevent to store negative value, allow only positive values. | | POSITIVE | | NATURALN | NATURALN/POSITIVEN data type prevent to assign a NULL value. | | POSITIVEN | | SIGNTYPE | SIGNTYPE allow only -1, 0, and 1 values. | | |
| PLS\_INTEGER | PLS\_INTEGER data type used to store signed integers data. Note : PLS\_INTEGER data type value require less storage space compare of NUMBER data type value. Storage Range : from -2147483647 to 2147483647 Performance : PLS\_INTEGER data type give you better performance on your data. PLS\_INTEGER perform arithmetic operation fast than NUMBER/BINARY\_INTEGER data type. | |

#### **Character Data types**

Character Data types used to store alphabetic/alphanumeric character. Following are some character data types in PL/SQL,

|  |  |  |
| --- | --- | --- |
| Data types | Description | Storage(Maximum) |
| CHAR | CHAR data type used to store character data within predefined length. | 32767 bytes |
| CHARACTER | CHARACTER data type same as CHAR type, is this another name of CHAR type. | 32767 bytes |
| VARCHAR2 | VARCHAR2 data type used to store variable strings data within predefined length. VARCHAR2 Subtypes : Following sub type defines same length value.   |  |  | | --- | --- | | Sub Data types | Description | | STRING | We can access this data type. | | VARCHAR | | 32767 bytes |
| NCHAR | NCHAR data type used to store national character data within predefined length. | 32767 bytes |
| NVARCHAR2 | NVARCHAR2 data type used to store Unicode string data within predefined length. | 32767 bytes |
| RAW | The RAW data type used to store binary data such as images, graphics etc. | 32767 bytes |
| LONG | LONG data type used to store variable string data within predefined length, This data type used for backward compatibility. Please use LONG data to the CLOB type. | 32760 bytes |
| LONG RAW | LONG RAW data type same as LONG type used to store variable string data within predefined length, This data type used for backward compatibility. Use LONG RAW data type for storing BLOB type data. | 32760 bytes |
| ROWID | The ROWID data type represents the actual storage address of a row. And table index identities as a logical rowid. This data type used to storing backward compatibility. We strongly recommended to use UROWID data type. | |
| UROWID[(size)] | The UROWID data type identifies as universal rowid, same as ROWID data type. Use UROWID data type for developing newer applications. Optional, You can also specify the size of UROWID column type. | 4000 bytes |

#### **Boolean Data types**

Boolean Data types stores logical values either TRUE or FALSE. Let's see Boolean data types in PL/SQL:

|  |  |
| --- | --- |
| Data types | Description |
| Boolean | Boolean data type stores logical values. Boolean data types doesn't take any parameters. Boolean data type store either TRUE or FALSE. Also store NULL, Oracle treats NULL as an unassigned boolean variable. You can not fetch boolean column value from another table. |

#### **Date/Time Data types**

Date/time variable can holds value, we can say date/time data type. PL/SQL automatically converts character value in to default date format ('DD-MM-YY') value. Following are the Date/Time data types in PL/SQL:

|  |  |  |
| --- | --- | --- |
| Data types | Description | Range |
| DATE | DATE data type stores valid date-time format with fixed length. Starting date from Jan 1, 4712 BC to Dec 31, 9999 AD. | Jan 1, 4712 BC to Dec 31, 9999 AD |
| TIMESTAMP | TIMESTAMP data type stores valid date (year, month, day) with time (hour, minute, second).   |  |  | | --- | --- | | Type | TIMESTAMP Type | | 1 | Syntax : TIMESTAMP [(fractional\_seconds\_precision)] Example : TIMESTAMP '2014-04-13 18:10:52.124' fractional\_seconds\_precision optionally specifies the number of digits in the fractional part of the second precision. Range from 0 to 9. The default is 6. | | 2 | Syntax : TIMESTAMP [(fractional\_seconds\_precision) ] WITH TIME ZONE Example : TIMESTAMP '2014-04-13 18:10:52.124 +05:30' WITH TIME ZONE specify the UTC time zone. Following two values represent the same instant in UTC. TIMESTAMP '1999-04-15 8:00:00 -8:00' (8.00 AM Pacific Standard Time) or  TIMESTAMP '1999-04-15 11:00:00 -5:00' (11:00 AM Eastern Standard Time) both are same. | | 3 | Syntax : TIMESTAMP [(fractional\_seconds\_precision) ] WITH LOCAL TIME ZONE Example : COL\_NAME TIMESTAMP(3) WITH LOCAL TIME ZONE; WITH LOCAL TIME ZONE specifies when you insert values into the database column, value is stored with the time zone of the database. The time-zone displacement is not stored in the column. When you retrieve value from Oracle database, returns it according to your UTC local time zone. | | |

Following are the Interval data types in PL/SQL:

Syntax : INTERVAL YEAR [ (precision) ] TO MONTH  
Note : precision specifies number of digits in years field range range from 0 to 4 and default is 2.  
Example : following example declares variable data type INTERVAL YEAR TO MONTH and assign interval 45 years and 7 month.

DECLARE

inter INTERVAL YEAR(2) TO MONTH;

BEGIN

inter := INTERVAL '45-7' YEAR TO MONTH;

inter := '45-7'; -- assign from character type (implicit conversion)

inter := INTERVAL '7' MONTH; -- Specify Months

inter := INTERVAL '45' YEAR; -- Specify years

END;

|  |  |
| --- | --- |
| Data types | Description |
| INTERVAL YEAR TO MONTH | INTERVAL YEAR TO MONTH data type is used to store and manipulate intervals of year and month. |
| INTERVAL DAY TO SECOND | INTERVAL DAY TO SECOND data type is used to store and manipulate intervals of days, hours, minutes, and seconds. Syntax : INTERVAL DAY [ (leading\_precision ) ] TO SECOND[(fractional\_seconds\_precision)] Note : leading\_precision and fractional\_seconds\_precisionspecifies number of digits in days field range from 0 to 9. The defaults are 2 and 6. Example : following example declare variable data type INTERVAL DAY TO SECOND.  DECLARE  inter INTERVAL DAY(3) TO SECOND(3);  BEGIN  IF inter > INTERVAL '6' DAY ...  ...  END; |

In PL/SQL datetime data type or interval data type fields values show the valid values for each field.

|  |  |  |
| --- | --- | --- |
| Field Name | Valid Value | Valid Interval Value |
| YEAR | -4712 to 9999 | Integer Value exclude 0 |
| MONTH | 01 to 12 | 0 to 11 |
| DAY | 01 to 31 | Integer Value exclude 0 |
| HOUR | 00 to 23 | 0 to 23 |
| MINUTE | 00 to 59 | 0 to 59 |
| SECOND | 00 to 59.9(n) here n is precision of time fractional seconds | 0 to 59.9(n) |

### LOB types

LOB data types used to store large objects such as image, video, graphics, text or audio. Maximum size is up to 4 Gigabytes. Following are the LOB data types in PL/SQL:

|  |  |  |
| --- | --- | --- |
| Data types | Description | Storage(Maximum) |
| BFILE | BFILE data type is used to store large binary objects into Operating System file. BFILE stores full file locator's path which are points to a stored binary object with in server. BFILE data type is read only, you can't modify them. | Size: up to 4GB (232 - 1 bytes) Directory name: 30 character File name: 255 characters |
| BLOB | BLOB data type is same as BFILE data type used to store unstructured binary object into Operating System file. BLOB type fully supported transactions are recoverable and replicated. | Size: 8 TB to 128 TB (4GB - 1) \* DB\_BLOCK\_SIZE |
| CLOB | CLOB data type is used to store large blocks of character data into Database. Store single byte and multi byte character data. CLOB type is fully supported transactions, also that are recoverable and replicated. | Size: 8 TB to 128 TB (4GB - 1) \* DB\_BLOCK\_SIZE |
| NCLOB | NCLOB data type to store large blocks of NCHAR data into Database. Store single byte and multi byte character data. NCLOB type fully supported transactions are recoverable and replicated. | NCLOB data type is used to store large blocks of NCHAR data into Database. Store single byte and multi byte character data. NCLOB type fully supported transactions, also that are recoverable and replicated. | Size: 8 TB to 128 TB (4GB - 1) \* DB\_BLOCK\_SIZE |

#### **Unknown Column types**

PL/SQL this data type is used when column type is not know.

|  |  |
| --- | --- |
| Data types | Description |
| %Type | This data type is used to store value unknown data type column in a table. Column is identified by %type data type. Eg. emp.eno%type emp name is table,  eno is a unknown data type column and %Type is data type to hold the value. | |
| %RowType | This data type is used to store values unknown data type in all columns in a table. All columns are identified by %RowType datatype. Eg. emp%rowtype emp name is table, all column type is %rowtype. | |
| %RowID | RowID is data type. RowID is two type extended or restricted. Extended return 0 and restricted return 1 otherwise return the row number. Function of Row ID:   |  |  | | --- | --- | | Function RowID | Description | | ROWID\_Verify | Verify if the rowid can be extended. | | ROWID\_Type | 0 = rowid, 1 = extended. | | ROWID\_Block\_Number | Block number that contain the record return 1 extended. | | ROWID\_Object | Object number of the object that contain record. | | ROWID\_Relative\_FNumber | Relative file number that contain record. | | ROWID\_Row\_Number | Row number of the Record. | | ROWID\_To\_Absolute\_FNumber | Return the absolute file number. | | ROWID\_To\_Extended | Convert the extended format. | | ROWID\_To\_Restricted | Convert the restricted format. | | |

### User-Defined Subtypes

PL/SQL gives you the control to create your own sub data type that are inherit from predefined base type. Sub types can increase reliability and provide compatibility with ANSI/ISO type. Several predefined subtypes are in STANDARD package.

#### **Defining Subtypes**

You can define your own subtypes in declarative part of PL/SQL block using the following syntax,

SUBTYPE subtype\_name IS base\_type[(constraint)] [NOT NULL];

Following example, predefined data type inherit from CHARACTER and INTEGER data type to make a new sub types,

SUBTYPE CHARACTER IS CHAR;

SUBTYPE INTEGER IS NUMBER(10,4); -- allows for numbers

#### **Example**

DECLARE

SUBTYPE message IS varchar2(25);

SUBTYPE age IS INTEGER(2,0);

description message;

ages age;

BEGIN

description := 'Web Developer';

ages := 22;

dbms\_output.put\_line('I am ' || description || ' and I am ' || ages || ' years Old.');

END;

/

#### **Result**

I am Web Developer and I am 22 years Old.   
PL/SQL procedure successfully completed.

### PL/SQL Variable Declaration | Variable Scope

PL/SQL variable declaration always specifies the variable name, data type of variable and size. In PL/SQL variable declaration you can also specifies initial value of declared variables.

#### **Variable Declaration Syntax**

The general syntax to declare a variable is:

variable\_name Datatype[Size] [NOT NULL] := [ value ];

Explanation:

* variable\_name is the predefined name of the variable.
* Data type is a valid PL/SQL data type.
* Size is an optional specification of data type size to hold the maximum size value.
* NOT NULL is an optional specification of the variable value can't be accept NULL.
* value is also an optional specification, where you can initialize the initial value of variable.
* Each variable declaration is terminated by a semicolon.

#### **Variable Declaration Example**

In this example variable defining employee number (eno) is NOT NULL(compulsory), employee name and initializing initial value to a variable,

Example Code:

DECLARE

eno number(5) NOT NULL := 2 -- NOT NULL (value can't be blank), Assign initial value

ename varchar2(15) := 'Branson Devs'; -- intialize value at the time of declaration

BEGIN

dbms\_output.put\_line('Declared Value:');

dbms\_output.put\_line(' Employee Number: ' || eno || ' Employee Name: ' || ename);

END;

/

Backward slash '/' indicates to execute the above PL/SQL Program.

Example Result :

Declared Value:  
Employee Number: 2 Employee Name: Branson Devs

#### **PL/SQL Placeholders**

Placeholders are any of variables, constants or records to store temporary in storage area. Later you can use it to manipulate data during the execution of a PL/SQL block. You can define placeholders with the name and data type. Here are some data types to define placeholders.

NUMBER(p,s), NUMBER(n), CHAR, VARCHAR2, DATE, LONG, LONG RAW, BLOB, CBLOB, BFILE

#### **Variables Scope**

PL/SQL variable scope is identified the region range which you can reference the variable. PL/SQL have two type scopes local scope and global scope,

Local variables - Variables declared in inner block and can't be referenced by the outside blocks.  
Global variables - Where as variables declared in a outer block and can be referencing by itself in inner blocks.

#### **Variable Scope Example**

In this example declaration two variables (num1 and num2) are in the outer block (Global variable) and another one third variable declared (num\_addition) into the inner block (local variable). Variable 'num\_addition' declared inner block so can't access in the outer block. But num1 and num2 can be accessed anywhere in the block.

Example Code:

DECLARE

num1 number := 10;

num2 number := 20;

BEGIN

DECLARE

num\_addition number;

BEGIN

num\_addition := num1 + num2;

dbms\_output.put\_line('Addition is: ' || num\_addition);

END; -- End of access num\_addition variable

END;

/

Example Result :

Addition is: 30

#### **Variable Scope Identifier (OUTER keyword)**

This example is also showing a difference between inner block and outer block variable scope. You can use OUTER keyword to access outer block variable inside the inner block. It's called global qualified name space.

Example Code:

DECLARE

num number := 10;

BEGIN

DECLARE

num number := 10;

BEGIN

IF num = OUTER.num THEN

DBMS\_OUTPUT.PUT\_LINE('Both are same value');

ELSE

DBMS\_OUTPUT.PUT\_LINE('Different value');

END IF;

END; -- End of scope to access num variable

END;

/

Example Result :

Both are same value

### PL/SQL Constants Variable | Declare PL/SQL Constants

You can declare PL/SQL constants along with the value and can not change them throughout the program block. The constants declaration functionality is available in almost all programming languages.

#### **PL/SQL Constant Declaration Syntax**

The general syntax for declaring a constant variable is:

Constant\_name CONSTANT Datatype[Size] := Value;

Explanation:

1. Constant\_name is a predefined name of the constant (similar to a variable name).
2. CONSTANT is a reserved keyword.
3. Data type is a valid PL/SQL data type.
4. Size is an optional specification of data type. It holds maximum capacity value for the particular variable.
5. Value must be assigned to a constant when it is declared. You can not assign or change it later.
6. Each constant declaration is terminated by a semicolon.

#### **Constant Example**

In this example, we will store the employee number which is NOT NULL (compulsory), employee Name and employee department which is constant,

Example Code :

DECLARE

eno number(5) NOT NULL := 2

ename varchar2(15) := 'Branson Devs';

edept CONSTANT varchar2(15) := 'Web Developer';

BEGIN

dbms\_output.put\_line('Declared Value:');

dbms\_output.put\_line(' Employee number: ' || eno || ' Employee Name: ' || ename);

dbms\_output.put\_line('Constant Declared:');

dbms\_output.put\_line(' Employee Department: ' || edept);

END;

/

Backward slash '/' is indicated to execute the above PL/SQL Block Program.

Example Result :

Declared Value:  
Employee number: 2 Employee Name: Branson Devs  
Constant Declared:  
Employee Department: Web Developer  
  
PL/SQL procedure successfully operation.

#### **Variable/Constant Declarations Example**

In this example, we will store the pi which is constant real number, radius and area which are real number,

Example Code :

DECLARE

pi CONSTANT REAL := 3.14159;

radius REAL := 3;

area REAL := (pi \* radius\*\*2);

BEGIN

dbms\_output.put\_line(' PI: ' || pi || ' Radius: ' || radius);

dbms\_output.put\_line(' Area: ' || area);

END;

/

Example Result :

PI: 3.14159 Radius: 1  
Area: 28.27431

### PL/SQL SET Serveroutput ON

Whenever you start Oracle SQL (PL/SQL) at that time you must have to write the "SET Serveroutput ON" command.

PL/SQL program execution into Oracle engine so we always required to get serveroutput result and display into the screen otherwise result can't be display.

#### **Set PL/SQL serveroutput Result ON**

SQL> set serveroutput on

#### **Set serveroutput on Example**

In this example showing you how to turn on serveroutput result, Here first line turn on serveroutput. After define variables and constants to print defined variable value using dbms\_output.put\_line command.

Example Code :

SQL> set serveroutput on

SQL> DECLARE

eno number(5) NOT NULL := 2

ename varchar2(15) := 'Branson Devs';

edept CONSTANT varchar2(15) := 'Web Developer';

BEGIN

dbms\_output.put\_line('Declared Value:');

dbms\_output.put\_line(' Employee Number: ' || eno || ' Employee Name: ' || ename);

dbms\_output.put\_line('Constant Declared:');

dbms\_output.put\_line(' Employee Department: ' || edept);

END;

/

Result display only if you execute "set serveroutput on" command.

Example Result :

Declared Value:  
Employee number: 2 Employee Name: Branson Devs  
Constant Declared:  
Employee Department: Web Developer

### PL/SQL Comments

PL/SQL Comments you can write single line comments or either multiple line comments,

1. Multi-line comments and
2. Single line comments

Multi-line comments are delimited with /\*..COMMENT TEXT..\*/ and  
single line comments starts with two dashes --.

PL/SQL Comments can begin in any column on any line. If you are embedding comments in SQL that will be embedded in PL/SQL you need to be careful for writing a column.

#### **PL/SQL Comments Syntax**

SQL>-- Single Line Comment

SQL>/\* Multiline Comment line 1

Multiline Comment line2 \*/

#### **Comments Example**

This example is nothing any new same as last lesson only new for showing a how to write single line comment or multi line comment in PL/SQL program,

Example Code :

SQL> set serveroutput on

SQL> DECLARE

eno number(5) NOT NULL := 2; --assign value into eno variable

ename varchar2(15) := 'Branson Devs';

/\* Constant value is fixed for edept value is "Web Deloper"

is fixed all program not required declare all times. \*/

edept CONSTANT varchar2(15) := 'Web Developer';

BEGIN

dbms\_output.put\_line('Declared Value:');

dbms\_output.put\_line(' Employee Number: ' || eno || ' Employee Name: ' || ename);

dbms\_output.put\_line('Constant Declared:');

dbms\_output.put\_line(' Employee Department: ' || edept);

END;

/

Example Result :

Declared Value:  
Employee number: 2 Employee Name: Branson Devs  
Constant Declared:  
Employee Department: Web Developer