What is PL/SQL block?

In PL/SQL, the code is not executed in single line format, but it is always executed by grouping the code into a single element called Blocks.

Blocks contain both PL/SQL as well as SQL instruction. All these instruction will be executed as a whole rather than executing a single instruction at a time.

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.

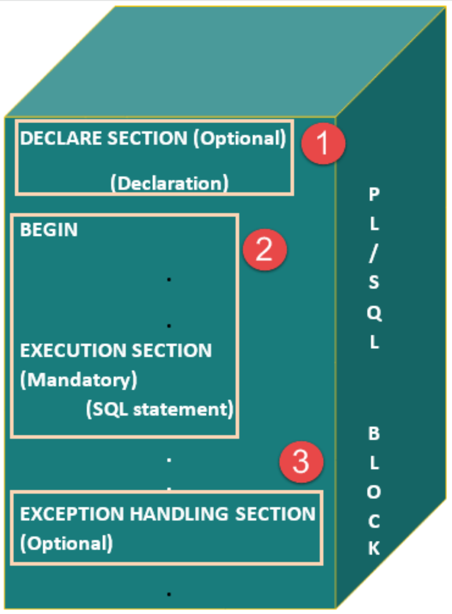
- Applications written in PL/SQL are fully portable.
- PL/SQL provides high security level.
- PL/SQL provides access to predefined SQL packages.
- PL/SQL provides support for Object-Oriented Programming.
- PL/SQL provides support for Developing Web Applications and Server Pages.

Block Structure

PL/SQL blocks have a pre-defined structure in which the code is to be grouped. Below are different sections of PL/SQL blocks.

1. Declaration section
2. Execution section
3. Exception-Handling section

The below picture illustrates the different PL/SQL block and their section order.



Declaration Section

This is the first section of the PL/SQL blocks. This section is an optional part. This is the section in which the declaration of variables, cursors, exceptions, subprograms, pragma instructions and collections that are needed in the block will be declared. Below are few more characteristics of this part.

- This particular section is optional and can be skipped if no declarations are needed.
- This should be the first section in a PL/SQL block, if present.
- This section starts with the keyword 'DECLARE' for triggers and anonymous block. For other subprograms, this keyword will not be present. Instead, the part after the subprogram name definition marks the declaration section.
- This section should always be followed by execution section.

Execution Section

Execution part is the main and mandatory part which actually executes the code that is written inside it. Since the PL/SQL expects the executable statements from this block this cannot be an empty block, i.e., it should have at least one valid executable code line in it. Below are few more characteristics of this part.

- This can contain both PL/SQL code and SQL code.
- This can contain one or many blocks inside it as a nested block.
- This section starts with the keyword 'BEGIN'.
- This section should be followed either by 'END' or Exception-Handling section (if present)

Exception-Handling Section:

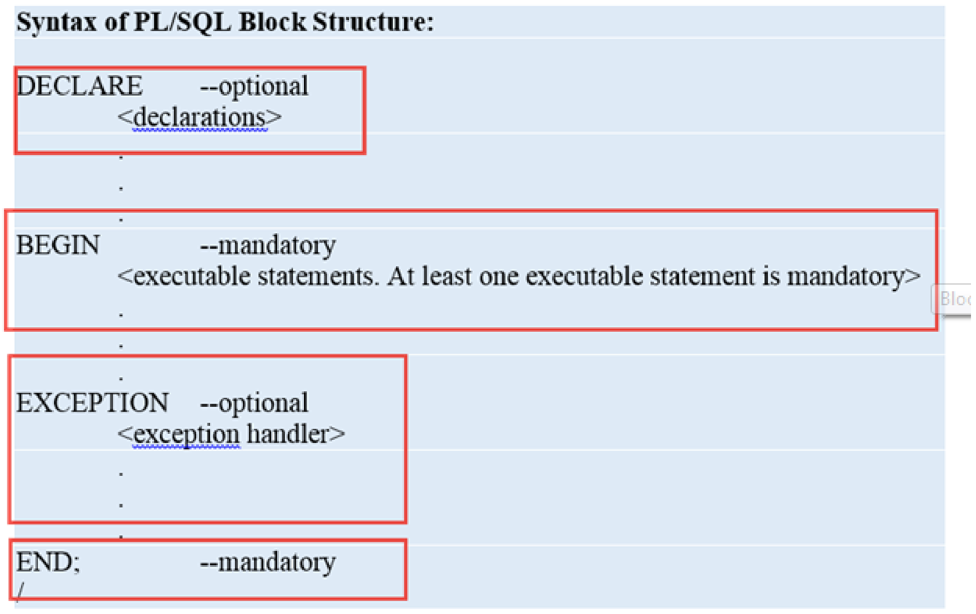
The exception is unavoidable in the program which occurs at run-time and to handle this Oracle has provided an Exception-handling section in blocks. This section can also contain PL/SQL statements. This is an optional section of the PL/SQL blocks.

- This is the section where the exception raised in the execution block is handled.
- This section is the last part of the PL/SQL block.
- Control from this section can never return to the execution block.
- This section starts with the keyword 'EXCEPTION'.
- This section should always be followed by the keyword 'END'.

The Keyword 'END' marks the end of PL/SQL block.

PL/SQL Block Syntax

Below is the syntax of the PL/SQL block structure.



```
Syntax of PL/SQL Block Structure:

DECLARE       --optional
          <declarations>
          .
          .
          .
BEGIN         --mandatory
          <executable statements. At least one executable statement is mandatory>
          .
          .
          .
EXCEPTION     --optional
          <exception handler>
          .
          .
          .
END;          --mandatory
/
```

```
DECLARE --optional
    <declarations>

BEGIN   --mandatory
    <executable statements. At least one executable statement is mandatory>

EXCEPTION --optional
    <exception handles>

END;   --mandatory
/
```

Note: A block should always be followed by '/' which sends the information to the compiler about the end of the block.

The 'Hello World' Example

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

The end; line signals the end of the PL/SQL block. To run the code from the SQL command line, you may need to type / at the beginning of the first blank line after the last line of the code. When the above code is executed at the SQL prompt, it produces the following result –

```
Hello World
```

```
PL/SQL procedure successfully completed.
```

Note: A block should always be followed by ‘/’ which sends the information to the compiler about the end of the block.

The PL/SQL Identifiers

PL/SQL identifiers are constants, variables, exceptions, procedures, cursors, and reserved words. The identifiers consist of a letter optionally followed by more letters, numerals, dollar signs, underscores, and number signs and should not exceed 30 characters.

By default, identifiers are not case-sensitive. So you can use integer or INTEGER to represent a numeric value. You cannot use a reserved keyword as an identifier.

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 can be included in the PL/SQL code that you write and helps anyone reading its source code. All programming languages allow some form of comments.

The PL/SQL supports single-line and multi-line comments. All characters available inside any comment are ignored by the 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;
/
```

When the above code is executed at the SQL prompt, it produces the following result –

```
Hello World

PL/SQL procedure successfully completed.
```

PL/SQL Variables

A variable is a meaningful name which facilitates a programmer to store data temporarily during the execution of code. It helps you to manipulate data in PL/SQL programs. It is nothing except a name given to a storage area. Each variable in the PL/SQL has a specific data type which defines the size and layout of the variable's memory.

A variable should not exceed 30 characters. Its letter optionally followed by more letters, dollar signs, numerals, underscore etc.

How to declare variable in PL/SQL

You must declare the PL/SQL variable in the declaration section or in a package as a global variable. After the declaration, PL/SQL allocates memory for the variable's value and the storage location is identified by the variable name.

Syntax for declaring variable:

Following is the syntax for declaring variable:

```
variable_name [CONSTANT] datatype [NOT NULL] [:= | DEFAULT initial_value]
```

Here, variable_name is a valid identifier in PL/SQL and datatype must be valid PL/SQL data type. A data type with size, scale or precision limit is called a constrained declaration. The constrained declaration needs less memory than unconstrained declaration.

Example:

Radius Number := 5;

Date_of_birth date;

Declaration Restrictions:

In PL/SQL while declaring the variable some restrictions hold.

- Forward references are not allowed i.e. you must declare a constant or variable before referencing it in another statement even if it is a declarative statement.
  val number := Total - 200;

Total number := 1000;

The first declaration is illegal because the TOTAL variable must be declared before using it in an assignment expression.

- Variables belonging to the same datatype cannot be declared in the same statement.

  N1, N2, N3 Number;

  It is an illegal declaration.

Naming rules for PL/SQL variables

The variable in PL/SQL must follow some naming rules like other programming languages.

- The variable_name should not exceed 30 characters.
- Variable name should not be the same as the table table's column of that block.
- The name of the variable must begin with ASCII letter. The PL/SQL is not case sensitive so it could be either lowercase or uppercase. For example: v_data and V_DATA refer to the same variables.
- You should make your variable easy to read and understand, after the first character, it may be any number, underscore (_) or dollar sign ($).
- NOT NULL is an optional specification on the variable.

Initializing Variables in PL/SQL (Assigning Values to Variables)

Everytime you declare a variable, PL/SQL defines a default value NULL to it. If you want to initialize a variable with other value than NULL value, you can do so during the declaration, by using any one of the following methods.

- The DEFAULT keyword
- The assignment operator

```
counter binary_integer := 0;
greetings varchar2(20) DEFAULT 'Hello JavaTpoint';
```

You can also specify NOT NULL constraint to avoid NULL value. If you specify the NOT NULL constraint, you must assign an initial value for that variable.

You must have a good programming skill to initialize variable properly otherwise, sometimes program would produce unexpected result.

Example of initilizing variable

Let's take a simple example to explain it well:

```
DECLARE
  a integer := 30;
  b integer := 40;
  c integer;
  f real;
BEGIN
  c := a + b;
  dbms_output.put_line('Value of c: ' || c);
  f := 100.0/3.0;
  dbms_output.put_line('Value of f: ' || f);
END;
```

After the execution, this will produce the following result:

```
Value of c: 70
Value of f: 33.33333333333333333

PL/SQL procedure successfully completed.
```

Variable Scope in PL/SQL:

PL/SQL allows nesting of blocks. A program block can contain another inner block. If you declare a variable within an inner block, it is not accessible to an outer block. There are two types of variable scope:

- Local Variable: Local variables are the inner block variables which are not accessible to outer blocks.
- Global Variable: Global variables are declared in outermost block.

Example of Local and Global variables

Let's take an example to show the usage of Local and Global variables in its simple form:

```
DECLARE
 -- Global variables
   num1 number := 95;
   num2 number := 85;
BEGIN
   dbms_output.put_line('Outer Variable num1: ' || num1);
   dbms_output.put_line('Outer Variable num2: ' || num2);
   DECLARE
      -- Local variables
      num1 number := 195;
      num2 number := 185;
   BEGIN
      dbms_output.put_line('Inner Variable num1: ' || num1);
      dbms_output.put_line('Inner Variable num2: ' || num2);
   END;
END;
```

After the execution, this will produce the following result:

```
Outer Variable num1: 95
Outer Variable num2: 85
Inner Variable num1: 195
Inner Variable num2: 185

PL/SQL procedure successfully completed.
```

PL/SQL Constants

A constant is a value used in a PL/SQL block that remains unchanged throughout the program. It is a user-defined literal value. It can be declared and used instead of actual values.

Let's take an example to explain it well:

Suppose, you have to write a program which will increase the salary of the employees upto 30%, you can declare a constant and use it throughout the program. Next time if you want to increase the salary again you can change the value of constant than the actual value throughout the program.

Syntax to declare a constant:

1. constant_name CONSTANT datatype := VALUE;
    - **Constant_name:**it is the name of constant just like variable name. The constant word is a reserved word and its value does not change.
    - **VALUE:** it is a value which is assigned to a constant when it is declared. It can not be assigned later.

Example of PL/SQL constant

Let's take an example to explain it well:

```
DECLARE
  -- constant declaration
  pi constant number := 3.141592654;
  -- other declarations
  radius number(5,2);
  dia number(5,2);
  circumference number(7, 2);
  area number (10, 2);
BEGIN
  -- processing
  radius := 9.5;
  dia := radius * 2;
  circumference := 2.0 * pi * radius;
  area := pi * radius * radius;
  -- output
  dbms_output.put_line('Radius: ' || radius);
  dbms_output.put_line('Diameter: ' || dia);
  dbms_output.put_line('Circumference: ' || circumference);
  dbms_output.put_line('Area: ' || area);
END;
/
```

After the execution of the above code at SQL prompt, it will produce the following result:.

```
Radius: 9.5
Diameter: 19
Circumference: 59.69
Area: 283.53

Pl/SQL procedure successfully completed.
```

PL/SQL Literals

Literals are the explicit numeric, character, string or boolean values which are not represented by an identifier. For example: TRUE, NULL, etc. are all literals of type boolean. PL/SQL literals are case-sensitive. There are following kinds of literals in PL/SQL:

- Numeric Literals
- Character Literals
- String Literals
- BOOLEAN Literals
- Date and Time Literals

Example of these different types of Literals:

Literals	Examples
Numeric	75125, 3568, 33.3333333 etc.
Character	'A' '%' '9' ' ' 'z' '('
String	Hello JavaTpoint!
Boolean	TRUE, FALSE, NULL etc.
Date and Time	'26-11-2002' , '2012-10-29 12:01:01'

PL/SQL If

PL/SQL supports the programming language features like conditional statements and iterative statements.

Syntax for IF Statement:

There are different syntaxes for the IF-THEN-ELSE statement.

Syntax: (IF-THEN statement):

```
IF condition
THEN
Statement: {It is executed when condition is true}
END IF;
```

This syntax is used when you want to execute statements only when condition is TRUE.

Syntax: (IF-THEN-ELSE statement):

```
IF condition
THEN
   {...statements to execute when condition is TRUE...}
ELSE
   {...statements to execute when condition is FALSE...}
END IF;
```

This syntax is used when you want to execute one set of statements when condition is TRUE or a different set of statements when condition is FALSE.

Syntax: (IF-THEN-ELSIF statement):

```
IF condition1
THEN
   {...statements to execute when condition1 is TRUE...}
ELSIF condition2
THEN
   {...statements to execute when condition2 is TRUE...}
END IF;
```

This syntax is used when you want to execute one set of statements when condition1 is TRUE or a different set of statements when condition2 is TRUE.

Syntax: (IF-THEN-ELSIF-ELSE statement):

```
IF condition1
THEN
  {...statements to execute when condition1 is TRUE...}
ELSIF condition2
THEN
  {...statements to execute when condition2 is TRUE...}
ELSE
  {...statements to execute when both condition1 and condition2 are FALSE...}
END IF;
```

It is the most advance syntax and used if you want to execute one set of statements when condition1 is TRUE, a different set of statement when condition2 is TRUE or a different set of statements when both the condition1 and condition2 are FALSE.

When a condition is found to be TRUE, the IF-THEN-ELSE statement will execute the corresponding code and not check the conditions any further.

If there no condition is met, the ELSE portion of the IF-THEN-ELSE statement will be executed.

ELSIF and ELSE portions are optional.

Example of PL/SQL If Statement

Let's take an example to see the whole concept:

```
DECLARE
  a number(3) := 500;
BEGIN
  -- check the boolean condition using if statement
  IF( a < 20 ) THEN
    -- if condition is true then print the following
    dbms_output.put_line('a is less than 20 ' );
  ELSE
    dbms_output.put_line('a is not less than 20 ' );
  END IF;
  dbms_output.put_line('value of a is : ' || a);
END;
```

After the execution of the above code in SQL prompt, you will get the following result:

```
a is not less than 20
value of a is : 500
PL/SQL procedure successfully completed.
```

PL/SQL Case Statement

The PL/SQL CASE statement facilitates you to execute a sequence of statements based on a selector. A selector can be anything such as variable, function or an expression that the CASE statement checks to a boolean value.

The CASE statement works like the IF statement, only using the keyword WHEN. A CASE statement is evaluated from top to bottom. If it get the condition TRUE, then the corresponding THEN clause is executed and the execution goes to the END CASE clause.

Syntax for the CASE Statement:

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

Example of PL/SQL case statement

Let's take an example to make it clear:

```
DECLARE
  grade char(1) := 'A';
BEGIN
  CASE grade
    when 'A' then dbms_output.put_line('Excellent');
    when 'B' then dbms_output.put_line('Very good');
```

```
      when 'C' then dbms_output.put_line('Good');
      when 'D' then dbms_output.put_line('Average');
      when 'F' then dbms_output.put_line('Passed with Grace');
      else dbms_output.put_line('Failed');
   END CASE;
END;
```

After the execution of above code, you will get the following result:

```
Excellent
PL/SQL procedure successfully completed.
```

User Defined Record

PL/SQL provides a user-defined record type that allows you to define the different record structures. These records consist of different fields. Suppose you want to keep track of your books in a library. You might want to track the following attributes about each book –

- Title
- Author
- Subject
- Book ID

Defining a Record

The record type is defined as –

```
TYPE
type_name IS RECORD
  ( field_name1  datatype1  [NOT NULL]  [:= DEFAULT EXPRESSION],
   field_name2   datatype2   [NOT NULL]  [:= DEFAULT EXPRESSION],
   ...
   field_nameN  datatypeN  [NOT NULL]  [:= DEFAULT EXPRESSION);
record-name  type_name;
```

The Book record is declared in the following way –

```
DECLARE
TYPE books IS RECORD
(title  varchar(50),
   author  varchar(50),
   subject varchar(100),
   book_id   number);
book1 books;
book2 books;
```

Accessing Fields

To access any field of a record, we use the dot **(.)** operator. The member access operator is coded as a period between the record variable name and the field that we wish to access. Following is an example to explain the usage of record –

```
DECLARE
   type books is record
      (title varchar(50),
      author varchar(50),
      subject varchar(100),
      book_id number);
   book1 books;
   book2 books;
BEGIN
   -- Book 1 specification
   book1.title  := 'C Programming';
   book1.author := 'Nuha Ali ';
   book1.subject := 'C Programming Tutorial';
   book1.book_id := 6495407;
   -- Book 2 specification
   book2.title := 'Telecom Billing';
   book2.author := 'Zara Ali';
   book2.subject := 'Telecom Billing Tutorial';
   book2.book_id := 6495700;

  -- Print book 1 record
   dbms_output.put_line('Book 1 title : '|| book1.title);
   dbms_output.put_line('Book 1 author : '|| book1.author);
   dbms_output.put_line('Book 1 subject : '|| book1.subject);
   dbms_output.put_line('Book 1 book_id : ' || book1.book_id);
   -- Print book 2 record
   dbms_output.put_line('Book 2 title : '|| book2.title);
   dbms_output.put_line('Book 2 author : '|| book2.author);
   dbms_output.put_line('Book 2 subject : '|| book2.subject);
   dbms_output.put_line('Book 2 book_id : '|| book2.book_id);
END;
/
```