**What is PL/SQL?**

PL/SQL (Procedural Language/Structured Query Language) is Oracle's procedural extension for SQL. It combines SQL's data manipulation capabilities with the procedural features of programming languages.

**Benefits Over SQL**

* **Procedural Logic**: PL/SQL supports procedural constructs like loops and conditional statements, allowing for more complex operations than standard SQL.
* **Error Handling**: PL/SQL provides robust error handling features through the use of exceptions, facilitating better management of runtime errors.
* **Modularity**: PL/SQL supports encapsulation in procedures and functions, promoting code reuse and maintainability.
* **Performance**: PL/SQL reduces network traffic because multiple SQL statements can be processed as a single block, improving performance in database operations.
* **Use of Variables**: PL/SQL allows the declaration of variables to hold data temporarily, enhancing program logic and functionality.

**PL/SQL Architecture**

PL/SQL architecture can be divided into several components:

1. **Anonymous Blocks**:
   * PL/SQL blocks can be executed without being stored in the database.
   * Structure:

sql

RunCopy code

1DECLARE

2 -- Variable declarations

3BEGIN

4 -- Executable statements

5EXCEPTION

6 -- Exception handling

7END;

1. **Syntax**:
   * Basic structure includes declarations, executable statements, and exception handling, with each section clearly specified.

**Variables & Data Types**

**Scalar Data Types**

* **Number**: Numeric values (e.g., INTEGER, FLOAT).
* **Character**: Alphanumeric data (e.g., CHAR, VARCHAR2).
* **Date**: Date and time values (e.g., DATE, TIMESTAMP).

**Anchored Data Types**

* Anchored types are based on existing database structures. They can change dynamically if the underlying type changes.

**Collections in PL/SQL**

Collections are data structures that allow for storage of multiple values.

1. **Associative Arrays**:
   * Also known as index-by tables, they consist of key-value pairs and can be indexed by strings or integers.
   * Syntax example:

sql

RunCopy code

1TYPE array\_type IS TABLE OF data\_type INDEX BY BINARY\_INTEGER;

1. **VARRAY**:
   * Fixed-size array of elements with a maximum limit.
   * Syntax example:

sql

RunCopy code

1TYPE varray\_type IS VARRAY(n) OF data\_type;

1. **Nested Tables**:
   * Unlike VARRAYs, they can grow dynamically and are useful for representing one-to-many relationships.
   * Syntax example:

sql

RunCopy code

1TYPE nested\_table\_type IS TABLE OF data\_type;

**Execution Plan Explanation**

The given execution plan shows how Oracle processes a SQL statement:

* **Operations**:
  + **SELECT STATEMENT**: Represents a SQL SELECT statement being executed.
  + **TABLE ACCESS BY INDEX ROWID**: Indicates access to a table using an index for faster data retrieval.
  + **INDEX RANGE SCAN**: Describes scanning an index for matching rows.
* **Details**:
  + **Id**: Unique identifier for the operation in the plan.
  + **Rows**: Estimated number of rows to be processed in this step.
  + **Bytes**: Number of bytes processed.
  + **Cost**: Estimated cost of the operation, providing insight into its efficiency.
  + **Time**: Estimated time taken for the operation.

**Conclusion**

PL/SQL is a powerful extension of SQL with features that enable complex procedural programming, error handling, and code modularity. Understanding its architecture and data types can significantly impact database interaction and application performance. The execution plan highlights the efficiency with which SQL queries are optimized and executed in Oracle databases.

## **Introducing PL/SQL block structure and anonymous block**

PL/SQL program units organize the code into blocks. A block without a name is known as an anonymous block. The anonymous block is the simplest unit in PL/SQL. It is called anonymous block because it is not saved in the Oracle database.

An anonymous block is an only one-time use and useful in certain situations such as creating test units. The following illustrates anonymous block syntax:

[DECLARE]

Declaration statements;

BEGIN

Execution statements;

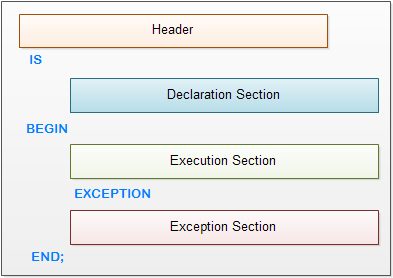
[EXCEPTION]

Exception handling statements;

END;

/Code language: SQL (Structured Query Language) (sql)

Let’s examine the PL/SQL block structure in greater detail.

PL/SQL Block Structure

The anonymous block has three basic sections that are the declaration, execution, and exception handling. Only the execution section is mandatory and the others are optional.

* The declaration section allows you to define data types, structures, and [variables](https://plsqltutorial.com/plsql-variables/). You often declare variables in the declaration section by giving them names, data types, and initial values.
* The execution section is required in a block structure and it must have at least one statement. The execution section is the place where you put the execution code or business logic code. You can use both procedural and SQL statements inside the execution section.
* The exception handling section is starting with the [EXCEPTION](https://plsqltutorial.com/plsql-exception/)keyword. The exception section is the place that you put the code to handle exceptions. You can either catch or handle exceptions in the exception section.

Notice that the single forward slash (/) is a signal to instruct SQL\*Plus to execute the PL/SQL block.

SQL\*Plus is an Oracle database client tool that executes PL/SQL statements and outputs the query’s results. SQL\*Plus provides administrators and programmers with command-line interface to work with Oracle database. SQL\*Plus is commonly referred as SQLPLUS.

## **PL/SQL block structure example**

Let’s take a look at the simplest PL/SQL block that does nothing.

BEGIN

NULL;

END;Code language: SQL (Structured Query Language) (sql)

If you execute the above anonymous block in SQL\*Plus you will see that it issues a message saying:

PL/SQL procedure successfully completed.

Because the NULL statement does nothing.

To display database’s output on the screen, you need to:

* First, use the SET SERVEROUTPUT ON command to instruct SQL\*Plus to echo database’s output after executing the PL/SQL block. The SET SERVEROUTPUT ONis SQL\*Plus command, which is not related to PL/SQL.
* Second, use the DBMS\_OUTPUT.PUT\_LINE procedure to output a string on the screen.

The following example displays a message Hello PL/SQL on a screen using SQL\*Plus:

SET SERVEROUTPUT ON SIZE 1000000

BEGIN

DBMS\_OUTPUT.PUT\_LINE('Hello PL/SQL');

END;

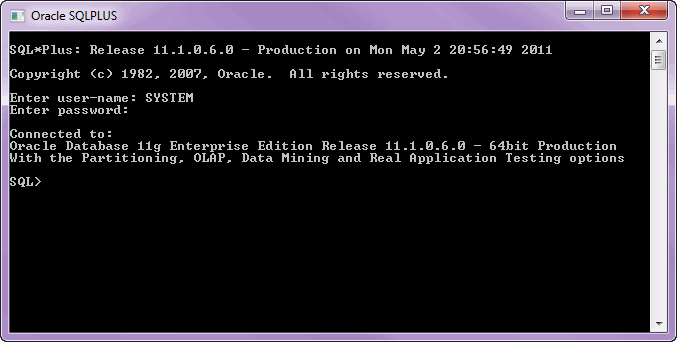
/Code language: SQL (Structured Query Language) (sql)

In this example, you just use the execution part to execute code. You will learn [how to declare variables](https://plsqltutorial.com/plsql-variables/) and [handle exceptions](https://plsqltutorial.com/plsql-exception/) in the next tutorials.

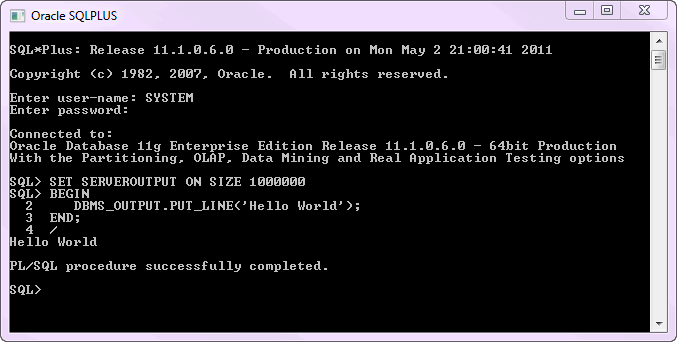
## **Exercise on anonymous block structure**

Now, it is your turn to create a PL/SQL block and execute it in SQL\*Plus that display a greeting message “Hello World” on the screen.

First, you need to login to the Oracle database via SQL\*Plus by providing a username and password as shown the following picture.

[](https://plsqltutorial.com/wp-content/uploads/2011/04/PLSQL_SQLPLUS_LOGIN.gif)Figure 1. SQL\*Plus Login

Second, type the following code into the SQL\*Plus and execute it as the following picture:

[](https://plsqltutorial.com/wp-content/uploads/2011/04/PLSQL_HELLO_WORLD.gif)Figure 2.PL/SQL Hello World

Congratulation, you’ve finished the first PL/SQL program!

In this tutorial, you have learned how PL/SQL organizes the code using block structure, and how to create the first PL/SQL and execute it using SQL\*PLUS.

In PL/SQL, a variable is a meaningful name of a temporary storage location that supports a particular data type in a program. Before using a variable, you need to declare it first in the declaration section of a [PL/SQL block](https://plsqltutorial.com/plsql-block-structure/).

## **PL/SQL variables naming rules**

Like other programming languages, a variable in PL/SQL must follow the naming rules as follows:

* The variable name must be less than 31 characters. Try to make it as meaningful as possible within 31 characters.
* The variable name must begin with an ASCII letter. It can be either lowercase or uppercase. Notice that PL/SQL is case-insensitive, which means v\_data and V\_DATA refer to the same variable.
* Followed by the first character are any number, underscore ( \_), and dollar sign ( $) characters. Once again, do not make your variables hard to read and difficult to understand.

## **PL/SQL variables naming convention**

It is highly recommended that you should follow the naming conventions listed in the following table to make the variables obvious in PL/SQL programs:

| **Prefix** | **Data Type** |
| --- | --- |
| v\_ | VARCHAR2 |
| n\_ | NUMBER |
| t\_ | TABLE |
| r\_ | ROW |
| d\_ | DATE |
| b\_ | BOOLEAN |

Each organization has its own development naming convention guidelines. Make sure that you comply with your organization’s naming convention guidelines.

For example, if you want to declare a variable that holds the first name of the employee with the VARCHAR2 data type, the variable name should be v\_first\_name.

## **PL/SQL Variables Declaration**

To declare a variable, you use a variable name followed by the data type and terminated by a semicolon ( ;). You can also explicitly add a length constraint to the data type within parentheses. The following illustrates some examples of declaring variables in a PL/SQL [anonymous block](https://plsqltutorial.com/plsql-block-structure/):

DECLARE

v\_first\_name varchar2(20);

v\_last\_name varchar2(20);

n\_employee\_id number;

d\_hire\_date date;

BEGIN

NULL;

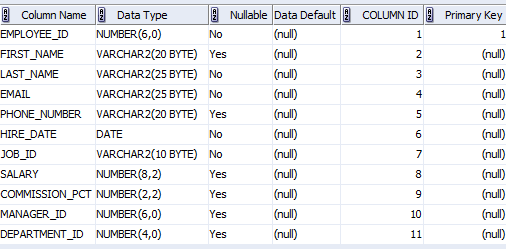
END;Code language: SQL (Structured Query Language) (sql)

## **PL/SQL variable anchors**

In PL/SQL program, one of the most common tasks is to select values from columns in a table into a set of variables. In case the data types of columns of the table changes, you have to change the PL/SQL program to make the types of the variables compatible with the new changes.

PL/SQL provides you with a very useful feature called **variable anchors**. It refers to the use of the  %TYPE  keyword to declare a variable with the data type is associated with a column’s data type of a particular column in a table.

Let’s take a look at the employeestable in HRsample database provided by Oracle:

[](https://plsqltutorial.com/wp-content/uploads/2011/05/Employees_Table.gif)Employees Table

DECLARE

v\_first\_name EMPLOYEES.FIRST\_NAME%TYPE;

v\_last\_name EMPLOYEES.LAST\_NAME%TYPE;

n\_employee\_id EMPLOYEES.EMPLOYEE\_ID%TYPE;

d\_hire\_date EMPLOYEES.HIRE\_DATE%TYPE;

BEGIN

NULL;

END;

/Code language: SQL (Structured Query Language) (sql)

The v\_first\_name variable has a data type that is the same as the data type of the first\_namecolumn in the  emloyees  table. In case the data type of the first\_namecolumn changes, the type of the v\_first\_namevariable automatically inherits the new data type of the column.

## **PL/SQL variable assignment**

In PL/SQL, to assign a value or a variable to another, you use the assignment operator ( := ) which is a colon( :) followed by the equal sign( = ).

Please see the code listing below to get a better understanding:

DECLARE

v\_first\_name EMPLOYEES.FIRST\_NAME%TYPE;

v\_last\_name EMPLOYEES.LAST\_NAME%TYPE;

n\_employee\_id EMPLOYEES.EMPLOYEE\_ID%TYPE;

d\_hire\_date EMPLOYEES.HIRE\_DATE%TYPE;

BEGIN

v\_first\_name := 'Mary';

v\_last\_name := 'Jane';

d\_hire\_date := to\_date('19700101','YYYYMMDD');

END;

/Code language: SQL (Structured Query Language) (sql)

In the example above, we assigned Mary to v\_first\_namevariable, Janeto v\_last\_namevariable, and result of the to\_datefunction to d\_hire\_datevariable.

You can use INTOof the SELECT statement to assign a value to a variable. The INTOclause moves the values from the SELECTquery’s column list into corresponding PL/SQL variables.

SET SERVEROUTPUT ON SIZE 1000000;

DECLARE

v\_first\_name EMPLOYEES.FIRST\_NAME%TYPE;

v\_last\_name EMPLOYEES.LAST\_NAME%TYPE;

n\_employee\_id EMPLOYEES.EMPLOYEE\_ID%TYPE;

d\_hire\_date EMPLOYEES.HIRE\_DATE%TYPE;

BEGIN

SELECT employee\_id,

first\_name,

last\_name,

hire\_date

INTO n\_employee\_id,

v\_first\_name,

v\_last\_name,

d\_hire\_date

FROM employees

WHERE employee\_id = 200;

DBMS\_OUTPUT.PUT\_LINE(v\_first\_name);

DBMS\_OUTPUT.PUT\_LINE(v\_last\_name);

DBMS\_OUTPUT.PUT\_LINE(d\_hire\_date);

END;

/Code language: SQL (Structured Query Language) (sql)

## **Initializing variables**

When you declare a variable, its value is uninitialized and hence is NULL. You can initialize variable a value in declaration section by using variable assignment.

See the following example:

DECLARE

n\_employee\_id EMPLOYEES.EMPLOYEE\_ID%TYPE :=200;

d\_hire\_date EMPLOYEES.HIRE\_DATE%TYPE:=to\_date('19700101','YYYYMMDD');

BEGIN

NULL;

END;

/Code language: SQL (Structured Query Language) (sql)

In PL/SQL, NULLmeans an unknown value so it has some special characteristics as follows:

* NULLis not equal to anything, even itself NULL.
* NULLis not greater than or less than anything else, even NULL.
* You cannot use logical operator equal ( =) or ( <>) with NULL. You must use the SQL [IS NULL](http://www.zentut.com/sql-tutorial/sql-is/) or [IS NOT NULL](http://www.zentut.com/sql-tutorial/sql-is/) to test the NULL values.

In this tutorial, we have shown you how to declare, assign and initialize PL/SQL variables. We also walked you through how to declare PL/SQL variables using variable anchors to make your code more flexible and adaptable to the changes in columns of the database tables.