# INTRODUCTION

PL/SQL stands for Procedural Language extension of SQL. PL/SQL is a combination of SQL along with the procedural features of programming languages. It was developed by Oracle Corporation in the early 90’s to enhance the capabilities of SQL.

# THE PL/SQL ENGINE

Oracle uses a PL/SQL engine to processes the PL/SQL statements. A PL/SQL code can be stored in the client system (client-side) or in the database (server-side).

# PL/SQL STRUCTURE

Each PL/SQL program consists of SQL and PL/SQL statements which from a PL/SQL block.

A PL/SQL Block consists of three sections:

1. The Declaration section (optional).
2. The Execution section (mandatory).
3. The Exception (or Error) Handling section (optional).
4. Declaration Section:

The Declaration section of a PL/SQL Block starts with the reserved keyword DECLARE. This section is optional and is used to declare any placeholders like variables, constants, records and cursors, which are used to manipulate data in the execution section. Placeholders may be any of Variables, Constants and Records, which stores data temporarily. Cursors are also declared in this section.

1. Execution Section:

The Execution section of a PL/SQL Block starts with the reserved keyword BEGIN and ends with END. This is a mandatory section and is the section where the program logic is written to perform any task. The programmatic constructs like loops, conditional statement and SQL statements form the part of execution section.

1. Exception Section:

The Exception section of a PL/SQL Block starts with the reserved keyword EXCEPTION. This section is optional. Any errors in the program can be handled in this section, so that the PL/SQL Blocks terminates gracefully. If the PL/SQL Block contains exceptions that cannot be handled, the Block terminates abruptly with errors.

Every statement in the above three sections must end with a semicolon (;). PL/SQL blocks can be nested within other PL/SQL blocks. Comments can be used to document code.

This is how a sample PL/SQL Block looks.

DECLARE

Variable declaration

BEGIN

Program Execution

EXCEPTION

Exception handling

END;

# ADVANTAGES OF PL/SQL

These are the advantages of PL/SQL

* ***Block Structures:*** PL SQL consists of blocks of code, which can be nested within each other. Each block forms a unit of a task or a logical module. PL/SQL Blocks can be stored in the database and reused.
* ***Procedural Language Capability:*** PL SQL consists of procedural language constructs such as conditional statements (if else statements) and loops like (FOR loops).
* ***Better Performance:*** PL SQL engine processes multiple SQL statements simultaneously as a single block, thereby reducing network traffic.
* ***Error Handling:*** PL/SQL handles errors or exceptions effectively during the execution of a PL/SQL program. Once an exception is caught, specific actions can be taken depending upon the type of the exception or it can be displayed to the user with a message.

# ORACLE PL/SQL

## Variables

These are placeholders that store the values that can change through the PL/SQL Block.

The General Syntax to declare a variable is:

*variable\_name datatype [NOT NULL := value ];*

* *variable\_name*is the name of the variable.
* *datatype*is a valid PL/SQL datatype.
* NOT NULL is an optional specification on the variable.
* *Value*or DEFAULT *value* is also an optional specification, where you can initialize a variable.
* Each variable declaration is a separate statement and must be terminated by a semicolon.
* All variables are declared in DECLARE block.
* When a variable is specified as NOT NULL, you must initialize the variable when it is declared.
* We can directly assign values to variables.
* We can assign values to variables directly from the database columns by using a SELECT.. INTO statement.

Syntax of assigning value to a variable is: *variable\_name:= value;*

* ***Local variables:*** These are declared in an inner block and cannot be referenced by outside Blocks.
* ***Global variables:*** These are declared in an outer block and can be referenced by itself and by its inner blocks.

## Constants

Constant is a value used in a PL/SQL Block that remains unchanged throughout the program. A constant is a user-defined literal value. You can declare a constant and use it instead of actual value.

The General Syntax to declare a variable is:

*constant\_name CONSTANT datatype := VALUE;*

* constant\_name is the name of the constant i.e. similar to a variable name.
* The word CONSTANT is a reserved word and ensures that the value does not change.
* VALUE - It is a value which must be assigned to a constant when it is declared. You cannot assign a value later.

## Records

Records are another type of datatypes which oracle allows to be defined as a placeholder. Records are composite datatypes, which means it is a combination of different scalar datatypes like char, varchar, number etc. Each scalar data types in the record holds a value. A record can be visualized as a row of data. It can contain all the contents of a row.

### Declaring a record

To declare a record, you must first define a composite datatype; then declare a record for that type.

The General Syntax to define a composite datatype is:

TYPE record\_type\_name IS RECORD

(first\_col\_name column\_datatype,

second\_col\_name column\_datatype, ...);

* ***record\_type\_name:*** It is the name of the composite type you want to define.
* ***first\_col\_name, second\_col\_name, etc.:*** It is the names the fields/columns within the record.
* ***column\_datatype:*** It is the scalar datatype of the fields.

There are different ways you can declare the datatype of the fields.

* You can declare the fields in the same way as you declare the fields while creating the table.
* If a field is based on a column from database table, you can define the field\_type as follows:

col\_name table\_name.column\_name%type;

By declaring the field datatype using %type, the datatype of the column is dynamically applied to the field. This method is useful when you are altering the column specification of the table, because you do not need to change the code again.

**NOTE:** You can use also %type to declare variables and constants.

The General Syntax to declare a **record** of a user-defined datatype is:

record\_name record\_type\_name;

If all the fields of a record are based on the columns of a table, we can declare the record as follows:

record\_name table\_name%ROWTYPE;

### Defining a record

Defining (assigning values) to a record means assigning values to fields within a record, which can be done as follows:

* Assign value to a single field/column.

1. Assign directly:
   * + ***Syntax:*** record\_name.col\_name := value;
2. Assign through SELECT INTO statement:
   * + ***Syntax:*** SELECT col1, col2 INTO record\_name.col\_name1, record\_name.col\_name2 FROM table\_name [WHERE clause];

* Assign value to whole record.
  1. Assign directly:
     + ***Syntax:*** record\_name.col\_name := value;
  2. Assign through SELECT INTO statement:
     + ***Syntax:*** SELECT \* INTO record\_name FROM table\_name [WHERE clause];

We can also use a record column/field to get its value and assign it to a variable.

## Conditional Statements

PL/SQL supports programming language features like conditional statements. The programming constructs are similar to how you use in programming languages like **Java** and **C++**.

Syntax IF THEN ELSE STATEMENT (IF/ELSE):

IF condition

THEN

statement 1;

ELSE

statement 2;

END IF;

We can also use:

* IF/ELSE-ELSEIF
* NESTED IF
* NESTED IF/ELSE and
* IF within an ELSE block etc.

## Iterative Statements

An iterative control Statements are used when we want to repeat the execution of one or more statements for specified number of times. These are similar to those in

There are three types of loops in PL/SQL:

* Simple Loop
* While Loop
* For Loop

### Simple Loop

A Simple Loop is used when a set of statements is to be executed at least once before the loop terminates. An EXIT condition must be specified in the loop, otherwise the loop will get into an infinite number of iterations. When the EXIT condition is satisfied the process exits from the loop.

The General Syntax to write a Simple Loop is:

LOOP

statements;

EXIT;

{or EXIT WHEN condition;}

END LOOP;

These are the important steps to be followed while using Simple Loop.

* Initialize a variable before the loop body.
* Increment the variable in the loop.
* Use an EXIT WHEN statement to exit from the Loop. If you use an EXIT statement without WHEN condition, the statements in the loop is executed only once.

### **While Loop**

A WHILE LOOP is used when a set of statements has to be executed as long as a condition is true. The condition is evaluated at the beginning of each iteration. The iteration continues until the condition becomes false.

The General Syntax to write a WHILE LOOP is:

WHILE <condition>

LOOP statements;

END LOOP;

Important steps to follow when executing a while loop:

* Initialize a variable before the loop body.
* Increment the variable in the loop.
* EXIT WHEN statement and EXIT statements can be used in while loops but it's not done often.

### **FOR Loop**

A FOR LOOP is used to execute a set of statements for a predetermined number of times. Iteration occurs between the start and end integer values given. The counter is always incremented by 1. The loop exits when the counter reaches the value of the end integer.

The General Syntax to write a FOR LOOP is:

FOR counter IN val1..val2

LOOP statements;

END LOOP;

***val1:*** Start integer value.

***val2:*** End integer value.

Important steps to follow when executing a while loop:

* The counter variable is implicitly declared in the declaration section, so it's not necessary to declare it explicitly.
* The counter variable is incremented by 1 and does not need to be incremented explicitly.
* EXIT WHEN statement and EXIT statements can be used in FOR loops but it's not done often.

## Cursors

A cursor is a temporary work area created in the system memory when a SQL statement is executed. A cursor contains information on a select statement and the rows of data accessed by it. This temporary work area is used to store the data retrieved from the database, and manipulate this data. A cursor can hold more than one row, but can process only one row at a time. The set of rows the cursor holds is called the **active set**.

There are two types of cursors in PL/SQL:

### Implicit cursors

These are created by default when DML statements like, INSERT, UPDATE, and DELETE statements are executed. They are also created when a SELECT statement that returns just one row, is executed.

Oracle provides few attributes called as implicit cursor attributes to check the status of DML operations. The cursor attributes available are:

* %FOUND
  + The return value is TRUE, if the DML statements like INSERT, DELETE and UPDATE affect at least one row and if SELECT ….INTO statement return at least one row. False Otherwise.
* %NOTFOUND
  + The return value is TRUE, if DML statements like INSERT, DELETE and UPDATE do not affect even one row and if SELECT ….INTO statement does not return a row. False Otherwise.
* %ROWCOUNT
  + Return the number of rows affected by the DML operations INSERT, DELETE, UPDATE, SELECT.
* %ISOPEN
  + Returns TRUE if the cursor is open, FALSE if the cursor is closed.

### Explicit cursors

They must be created when you are executing a SELECT statement that returns more than one row. Even though the cursor stores multiple records, only one record can be processed at a time, which is called as **current row**. When you fetch a row the current row position moves to next row.

An explicit cursor is defined in the declaration section of the PL/SQL Block. It is created on a SELECT Statement which returns more than one row. We can provide a suitable name for the cursor.

The General Syntax for creating a cursor is as given below:

CURSOR cursor\_name IS select\_statement;

***cursor\_name:*** A suitable name for the cursor.

***select\_statement:*** A select query which returns multiple rows.

There are four steps in using an Explicit Cursor.

* DECLARE the cursor in the declaration section.
* OPEN the cursor in the Execution Section.
* FETCH the data from cursor into PL/SQL variables or records in the Execution Section.
* CLOSE the cursor in the Execution Section before you end the PL/SQL Block.

#### Declare Cursor

DECLARE

CURSOR emp\_cur IS

SELECT \*

FROM emp\_tbl

WHERE salary > 5000;

#### Open Cursor

OPEN cursor\_name;

#### Fetch the records in the cursor one at a time

FETCH cursor\_name INTO record\_name;

#### Close Cursor

CLOSE cursor\_name;

#### Important Points Regarding Explicit Cursors

* When a cursor is opened, the first row becomes the current row.
* On every fetch statement, the pointer moves to the next row.
* When there is more than one row in a cursor we can use loops along with explicit cursor attributes to fetch all the records.
* Cursor will throw an error when pointing after last row.
* We can use attributes like %FOUND, %NOTFOUND, %ROWCOUNT, %ISOPEN to check status of an explicit cursor.

**NOTE:**

* Both implicit and explicit cursors have the same functionality, but they differ in the way they are accessed.
* PL/SQL returns an error when no data is selected.
* Cursor Returns INVALID\_CURSOR if cursor is declared, but not open; or if cursor has been closed.
* Cursor Returns NULL if cursor is open, but fetch has not been executed.

## Procedures

A stored procedure is a named PL/SQL block which performs one or more specific task. This is similar to a procedure in other programming languages. A procedure has a header and a body. The header consists of the name of the procedure and the parameters or variables passed to the procedure. The body consists of declaration section, execution section and exception section similar to a general PL/SQL Block. A procedure is similar to an anonymous PL/SQL Block but it is named for repeated usage.

### Creating Procedure

General Syntax to create a procedure is:

CREATE [OR REPLACE] PROCEDURE proc\_name [list of parameters]

IS

Declaration section

BEGIN

Execution section

EXCEPTION

Exception section

END;

***IS*** marks the beginning of the body of the procedure and is similar to DECLARE in anonymous PL/SQL Blocks. The code between IS and BEGIN forms the Declaration section.

### Parameters Types and Usage

We can pass parameters to procedures in three ways.

#### IN-parameters

* These types of parameters are used to send values to stored procedures.
* This type of parameter is a read only parameter.

#### OUT-parameters

* + These types of parameters are used to get values from stored procedures. This is similar to a return type in functions.
  + This is a write-only parameter i.e. we cannot pass values to OUT parameters while executing the stored procedure, but we can assign values to OUT parameter inside the stored procedure and the calling program can receive this output value.

#### IN OUT-parameters

* + The IN OUT parameter allows us to pass values into a procedure and get output values from the procedure.
  + This is possible only if the value passed to the procedure and output value have a same datatype.

**NOTE:**

* Syntax for Passing Parameters:

CREATE [OR REPLACE] PROCEDURE proc\_name (param\_name PARAM\_TYPE datatype)

***PARAM\_TYPE:*** IN/OUT/IN OUT

* A procedure may or may not return any value.
* If a parameter is not explicitly defined a parameter type, then by default it is an IN type parameter.

### Executing procedure

There are two ways to execute a procedure.

1) From the SQL prompt.

EXECUTE [or EXEC] procedure\_name;

2) Within another procedure – simply use the procedure name.

procedure\_name;

## Functions

A function is a named PL/SQL Block which is similar to a procedure. **The major difference between a procedure and a function is, a function must always return a value, but a procedure may or may not return a value.**

### Creating Function

The General Syntax to create a function is:

CREATE [OR REPLACE] FUNCTION function\_name [parameters]

RETURN return\_datatype;

IS

Declaration\_section

BEGIN

Execution\_section

Return return\_variable;

EXCEPTION

exception section

Return return\_variable;

END;

***Return Type:*** The header section defines the return type of the function. The return datatype can be any of the oracle datatype like varchar, number etc.

The execution and exception section both should return a value which is of the datatype defined in the header section.

### Executing Function

A function can be executed in the following ways:

1) Since a function returns a value we can assign it to a variable.

employee\_name:= employer\_details\_func;

2) As a part of a SELECT statement

SELECT employer\_details\_func FROM dual;

3) In a PL/SQL Statements like:

dbms\_output.put\_line (employer\_details\_func);

## Exception Handling

PL/SQL provides a feature to handle the Exceptions which occur in a PL/SQL Block known as exception Handling. Using Exception Handling we can test the code and avoid it from exiting abruptly. When an exception occurs; a message which explains its cause is received.

PL/SQL Exception message consists of three parts.

* Type of Exception
* An Error Code
* A message

### Structure of Exception Handling

The General Syntax for coding the exception section

DECLARE

Declaration section

BEGIN

Exception section

EXCEPTION

WHEN ex\_name1 THEN

-Error handling statements

WHEN ex\_name2 THEN

-Error handling statements

WHEN Others THEN

-Error handling statements

END;

General PL/SQL statements can be used in the Exception Block.

### Types of Exception

There are 3 types of Exceptions.

* + - Named System Exceptions
      * System exceptions are automatically raised by Oracle, when a program violates a RDBMS rule.
      * They are pre-defined and given a name in Oracle.
      * For example: NO\_DATA\_FOUND and ZERO\_DIVIDE
    - Unnamed System Exceptions
      * Those system exception for which oracle does not provide a name.
      * These exceptions do not occur frequently. These Exceptions have a code and an associated message.
    - User-defined Exceptions
    - These exceptions are based on business rules.
    - They should be explicitly declared in the declaration section.
    - They should be explicitly raised in the Execution Section.
    - They should be handled by referencing the user-defined exception name in the exception section.

## Triggers

A trigger is a pl/sql block structure which is fired when a DML statements like Insert, Delete, Update is executed on a database table. A trigger is triggered automatically when an associated DML statement is executed.

The Syntax for creating a trigger is:

CREATE [OR REPLACE ] TRIGGER trigger\_name

{BEFORE | AFTER | INSTEAD OF }

{INSERT [OR] | UPDATE [OR] | DELETE}

[OF col\_name]

ON table\_name

[REFERENCING OLD AS o NEW AS n]

[FOR EACH ROW]

WHEN (condition)

BEGIN

sql statements

END;

***CREATE [OR REPLACE] TRIGGER trigger\_name:*** This clause creates a trigger with the given name or overwrites an existing trigger with the same name.

***{BEFORE | AFTER | INSTEAD OF}:*** This clause indicates at what time should the trigger get fired i.e. for example: before or after updating a table. INSTEAD OF is used to create a trigger on a view. Before and after cannot be used to create a trigger on a view.

***{INSERT [OR] | UPDATE [OR] | DELETE}:*** This clause determines the triggering event. More than one triggering events can be used together separated by OR keyword. The trigger gets fired at all the specified triggering event.

***[OF col\_name]:*** This clause is used with update triggers. This clause is used when you want to trigger an event only when a specific column is updated.

***[ON table\_name]:*** This clause identifies the name of the table or view to which the trigger is associated.

***[REFERENCING OLD AS o NEW AS n]:*** This clause is used to reference the old and new values of the data being changed. By default, you reference the values as [:old.column\_name/:new.column\_name]. The reference names can also be changed from old (or new) to any other user-defined name. You cannot reference old values when inserting a record, or new values when deleting a record, because they do not exist.

***[FOR EACH ROW]:*** This clause is used to determine whether a trigger must fire when each row gets affected (i.e. a Row Level Trigger) or just once when the entire sql statement is executed (i.e. Statement level Trigger).

***WHEN (condition):*** This clause is valid only for row level triggers. The trigger is fired only for rows that satisfy the condition specified.

Triggers are executed in following hierarchy:

1. BEFORE statement level
2. BEFORE row level
3. AFTER row level
4. AFTER statement level