PL/SQL

- 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).

A PL/SQL Block:

- Each PL/SQL program consists of SQL and PL/SQL statements
- which form a PL/SQL block.
- A PL/SQL Block consists of three sections:
  - The Declaration section (optional).
  - The Execution section (mandatory).
  - The Exception (or Error) Handling section (optional).

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,

Declaration Section

- 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.

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.

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.

Exception Section

- 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.

PL/SQL Block

- This is how a sample PL/SQL Block looks.

```
DECLARE
    Variable declaration
BEGIN
    Program Execution
EXCEPTION
    Exception handling
END;
```

PL/SQL Placeholders

- Placeholders are temporary storage areas.
- Placeholders can be any of Variables, Constants and Records.
- Oracle defines placeholders to store data temporarily,
- which are used to manipulate data during the execution of a PL SQL block.

PL/SQL Placeholders

- Depending on the kind of data you want to store,
- you can define placeholders with a name and a datatype.
- Few of the datatypes used to define placeholders are as given below.
- Number (n,m) , Char (n) , Varchar2 (n) , Date , Long , Long raw, Raw, Blob, Clob, Nclob, Bfile

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.

PL/SQL Variables

- 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.

PL/SQL Variables

- For example,
- if you want to store the current salary of an employee,
- you can use a variable.
- DECLARE salary number (6);
- * "salary" is a variable of datatype number and of length 6.

PL/SQL Variables

- When a variable is specified as NOT NULL,
- you must initialize the variable when it is declared.
- For example: The below example declares two variables, one of which is a not null.
- DECLARE
  - salary number(4);
  - dept varchar2(10) NOT NULL := "HR Dept";

PL/SQL Variables

- The value of a variable can change in the execution or exception section of the PL/SQL Block.
- We can assign values to variables in two ways.
- We can directly assign values to variables.
  - The General Syntax is:
  - variable_name:= value;

PL/SQL Variables

- We can assign values to variables directly from the database columns by using a SELECT.. INTO statement.
- The General Syntax is:
  - SELECT column_name INTO variable_name FROM table_name [WHERE condition];

Example

- The below program will get the salary of an employee with id '1116' and display it on the screen.
- DECLARE
  - var_salary number(6);
  - var_emp_id number(6) = 1116;
- BEGIN SELECT salary INTO var_salary
- FROM employee

Example

- WHERE emp_id = var_emp_id;
- dbms_output.put_line(var_salary);
- dbms_output.put_line('The employee ' || var_emp_id || ' has salary ' || var_salary); END;

Scope of Variables

- PL/SQL allows the nesting of Blocks within Blocks
- i.e, the Execution section of an outer block can contain inner blocks.
- Therefore, a variable which is accessible to an outer Block is also accessible to all nested inner Blocks.

Scope of Variables

- The variables declared in the inner blocks are not accessible to outer blocks.
- Based on their declaration we can classify variables into two types.
- *Local* variables - These are declared in a inner block and cannot be referenced by outside Blocks.

Scope of Variables

- *Global* variables - These are declared in a outer block and can be referenced by its itself and by its inner blocks.
- For Example:
- creating two variables in the outer block and assigning their product to the third variable created in the inner block.

Scope of Variables

- 1> DECLARE
- 2> var_num1 number;
- 3> var_num2 number;
- 4> BEGIN
- 5> var_num1 := 100;
- 6> var_num2 := 200;
- 7> DECLARE
- 8> var_mult number;
- 9> BEGIN
- 10> var_mult := var_num1 * var_num2;
- 11> END; 12> END; 13> /

Scope of Variables

- The variable 'var_mult' is declared in the inner block,
- so cannot be accessed in the outer block
- i.e. it cannot be accessed after line 11.
- The variables 'var_num1' and 'var_num2' can be accessed anywhere in the block.

PL/SQL Constants

- As the name implies a *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.

PL/SQL Constants

- For example:
- If you want to write a program which will increase the salary of the employees by 25%,
- you can declare a constant and use it throughout the program.
- Next time when you want to increase the salary again you can change the value of the constant which will be easier than changing the actual value throughout the program.

PL/SQL Constants

- The General Syntax to declare a constant is:
- constant_name CONSTANT datatype := VALUE;
- *constant_name* is the name of the constant i.e. similar to a variable name.

PL/SQL Constants

- 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.
- For example, to declare salary_increase, you can write code as follows:

PL/SQL Constants

- DECLARE
- salary_increase CONSTANT number (3) := 10;
- You *must* assign a value to a constant at the time you declare it.
- If you do not assign a value to a constant while declaring it and try to assign a value in the execution section, you will get a error.

PL/SQL Constants

- If you execute the below Pl/SQL block you will get error.
- DECLARE
- salary_increase CONSTANT number(3);
- BEGIN
  - salary_increase := 100;
  - dbms_output.put_line (salary_increase);
- END;

PL/SQL 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.

PL/SQL Records

- 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:

Declaring a record

- 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.

Declaring a record

- *first_col_name, second_col_name, etc.,*
- - *it is the* names of the fields/columns within the record.
- *column_datatype* defines the scalar datatype of the fields.

Declaring a record

- There are different ways you can declare the datatype of the fields.
- 1) You can declare the field in the same way you declare the fields when creating a table.
- 2) 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;

Declaring a record

- By declaring the field datatype in the above method,
- 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.

Declaring a record

- The General Syntax to declare a record of a user-defined datatype is:
  - record_name record_type_name;
- The following code shows how to declare a record called *employee_rec* based on a user-defined type

Declaring a record

- DECLARE
- TYPE employee_type IS RECORD
- (employee_id number(5),
- employee_first_name varchar2(25),
- employee_last_name employee.last_name%type,
- employee_dept employee.dept%type);
- employee_salary employee.salary%type;
- employee_rec employee_type;

Declaring a record

- 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;
- For example, the above declaration of employee_rec can be as follows:
- DECLARE employee_rec employee%ROWTYPE;

Declaring a record

- The advantages of declaring the record as a ROWTYPE are:
- You do not need to explicitly declare variables for all the columns in a table.
- If you alter the column specification in the database table, you do not need to update the code.

Declaring a record

- The disadvantage of declaring the record as a ROWTYPE is:
- When u create a record as a ROWTYPE, fields will be created for all the columns in the table and memory will be used to create the datatype for all the fields.
- So use ROWTYPE only when you are using all the columns of the table in the program.

Declaring a record

- **NOTE:** When you are creating a record,
- you are just creating a datatype,
- similar to creating a variable.
- You need to assign values to the record to use them.
- The following table consolidates the different ways in which you can define and declare a pl/sql record

Declaring a record

Syntax	Usage
TYPE record_type_name IS RECORD (column_name1 datatype, column_name2 datatype, ...);	Define a composite datatype, where each field is scalar.
col_name table_name.column_name%type;	Dynamically define the datatype of a column based on a database column.
record_name record_type_name;	Declare a record based on a user-defined type.
record_name table_name%ROWTYPE;	Dynamically declare a record based on an entire row of a table. Each column in the table corresponds to a field in the record.

Passing Values To and From a Record

- When you assign values to a record,
- you actually assign values to the fields within it.
- The General Syntax to assign a value to a column within a record direclty is:
- record_name.col_name := value;

Passing Values To and From a Record

- If you used %ROWTYPE to declare a record, you can assign values as shown:
  - record_name.column_name := value;
- We can assign values to records using SELECT Statements as shown:

Passing Values To and From a Record

  - SELECT col1, col2
  - INTO record_name.col_name1, record_name.col_name2 FROM table_name [WHERE clause];
- If %ROWTYPE is used to declare a record then you can directly assign values to the whole record instead of each columns separately.

Passing Values To and From a Record

- In this case, you must SELECT all the columns from the table into the record as shown:
- SELECT *
- INTO record_name
- FROM table_name
- [WHERE clause];

Passing Values To and From a Record

- The General Syntax to retrieve a value from a specific field into another variable is:
  - var_name := record_name.col_name;
- The following table consolidates the different ways you can assign values to and from a record:

Passing Values To and From a Record

Syntax	Usage
record_name.col_name := value;	To directly assign a value to a specific column of a record.
record_name.column_name := value;	To directly assign a value to a specific column of a record, if the record is declared using %ROWTYPE.
SELECT col1, col2 INTO record_name.col_name1, record_name.col_name2 FROM table_name [WHERE clause];	To assign values to each field of a record from the database table.
SELECT * INTO record_name FROM table_name [WHERE clause];	To assign a value to all fields in the record from a database table.
variable_name := record_name.col_name;	To get a value from a record column and assigning it to a variable.