|  |  |
| --- | --- |
| **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.   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.  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”;*

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 the two ways given below.

1) We can directly assign values to variables.   
    The General Syntax is:

*variable\_name:= value;*

2) 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*

*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;*

*/*

**NOTE: The backward slash '/' in the above program indicates to execute the above PL/SQL Block.**

**Scope of PS/SQL 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. 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.
* *Global* variables - These are declared in a outer block and can be referenced by its itself and by its inner blocks.

For Example: In the below example we are creating two variables in the outer block and assigning thier product to the third variable created in the inner block. 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.

|  |
| --- |
| *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> /* |

## **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.  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. 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. * 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:

*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. 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**

|  |
| --- |
| What are 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* defines the scalar datatype of the fields.   There are different ways you can declare the datatype of the fields.  1) You can declare the field in the same way as you declare the fieds while creating the 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; | |  |

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.   
  
The General Syntax to declare a record of a uer-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.

|  |
| --- |
| 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; |

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 as follows:

|  |
| --- |
| DECLARE   employee\_rec employee%ROWTYPE; |

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

The disadvantage of declaring the record as a ROWTYPE is:  
1) 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.

**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.

|  |  |
| --- | --- |
| **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; |

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:

|  |
| --- |
| 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. 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]; |

Lets see how we can get values 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:

|  |  |
| --- | --- |
| **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. |

## **Conditional Statements in PL/SQL**

As the name implies, PL/SQL supports programming language features like conditional statements, iterative statements.

The programming constructs are similar to how you use in programming languages like Java and C++.

|  |  |
| --- | --- |
| In this section I will provide you syntax of how to use conditional statements in PL/SQL programming. **Conditional Statements in PL/SQL** **IF THEN ELSE STATEMENT**  1)  *IF condition*  *THEN*  *statement 1;*  *ELSE*  *statement 2;*  *END IF;*    2)  *IF condition 1*  *THEN*  *statement 1;*  *statement 2;*  *ELSIF condtion2 THEN*  *statement 3;*  *ELSE*  *statement 4;*  *END IF* |  |

3)

*IF condition 1*

*THEN*

*statement 1;*

*statement 2;*

*ELSIF condtion2 THEN*

*statement 3;*

*ELSE*

*statement 4;*

*END IF;*

4)

*IF condition1 THEN*

*ELSE*

*IF condition2 THEN*

*statement1;*

*END IF;*

*ELSIF condition3 THEN*

*statement2;*

*END IF;*

## **Iterative Statements in PL/SQL**

|  |  |
| --- | --- |
| Iterative control Statements are used when we want to repeat the execution of one or more statements for specified number of times. There are three types of loops in PL/SQL: • Simple Loop • While Loop • For Loop | |
| 1) 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. |  |

**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.

1) Initialise a variable before the loop body.  
2) Increment the variable in the loop.  
3) Use a EXIT WHEN statement to exit from the Loop. If you use a EXIT statement without WHEN condition, the statements in the loop is executed only once.

2) 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: 

1) Initialise a variable before the loop body.  
2) Increment the variable in the loop.  
3) EXIT WHEN statement and EXIT statements can be used in while loops but it's not done oftenly.

3) 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 reachs 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: 

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