About PL/SQL

- PL/SQL is the PROCEDURAL EXTENSION to SQL with design features of programming languages.

  What design features?
  ~~~~~~~~~~~~~~~~~~~~
  Variables & Constants
  Data Types
  Control Structures
    - Selection
    - Iterative
    - Branching
  Exception Handling
  Sub-Programming / Subroutines
    - Procedures
    - Functions
    - Packages
  :

- Data manipulation and query statements of SQL are included within procedural units of code.

- PL/SQL provides features like conditional execution, looping and branching.
- PL/SQL supports subroutines, as well.
- PL/SQL program is of block type, which can be “sequential” or “nested” (one inside the other).

PL/SQL Environment

A PL/SQL block of code can have TWO types of statements
(1) PL/SQL Statements
(2) SQL Statements

The PL/SQL engine parses the PL/SQL block of code and segregates the two type of statements.

The PL/SQL statements are executed by the PROCEDURAL STATEMENT EXECUTOR.

However, the SQL statements are executed by the SQL STATEMENT EXECUTOR.

The PL/SQL engine is part of the Oracle Database Server.

Featues of PL/SQL

PL/SQL provides the following features:
* Tight Integration with SQL
* Better performance
* Several SQL statements can be bundled together into one PL/SQL block and sent to the server as a single unit.
* Standard and portable language
* Although there are a number of alternatives when it comes to writing software to run against the Oracle Database, it is easier to run highly efficient code in PL/SQL, to access the Oracle Database, than
  ~~~~~~~~~~~~~~~~~~~~

  in any other programming language.
* Modularized Program Development
* You can program with procedural language control structures
* PL/SQL handle errors - Exception Handling

PL/SQL Block Structure
======================
A PL/SQL block comprises of the following blocks

```
DECLARE (Optional)
  -- All varibales, constants, cursors
  -- User-defined exception
BEGIN (Mandatory)
  -- We will have the PL/SQL and SQL
  -- statements.
EXCEPTION (Optional)
  -- Action to perform when error occurs
END; (Mandatory)
```

Example:

```
BEGIN
      DBMS_OUTPUT.PUT_LINE('Hello World - PL/SQL');
END;
/
```

Type the above code in any file which ends with .SQL

Now, to execute the file as follows

```
SQL> @D:\Dec3-CG\ex_01.sql

PL/SQL procedure successfully completed.
```

If we are using the DBMS_OUTPUT.PUT_LINE() we need to set the server output
as ON

Observe below:

```
SQL> SET SERVEROUTPUT ON
SQL> @D:\Dec3-CG\ex_01.sql
Hello World - PL/SQL

PL/SQL procedure successfully completed.
```

Now observe the output.

Comments
========
PL/SQL supports TWO types of comments

(1) Single Line Comment
    ~~~~~~~~~~~~~~~~~~~
    Done using -- (Double dash/hyphen) characters

(2) Multi-Line Comment
    ~~~~~~~~~~~~~~~~~~
    Same like that of C/C++/Java

    i.e.

```
    /*
      This is our multi-line
      comment
    */
```

PL/SQL Block Types
==========
PL/SQL supports THREE types of blocks
(a) Anonymous PL/SQL block
(b) Procedure
(c) Function

Both, Procedure & Function are NAMED blocks.

Use of Variables
==========
Variables can be used for:
• Temporary storage of data
• Manipulation of stored values
• Reusability
• Ease of maintenance

Handling Variables in PL/SQL
==========
• Declare and initialize variables in the declaration section.
• Assign new values to variables in the executable section.
• Pass values into PL/SQL blocks through parameters.
• View results through output variables.

Types of Variables
==========
• PL/SQL variables:
  – Scalar
  – Composite
  – Reference
  – LOB (large objects)
• Non-PL/SQL variables: Bind or host variables

Declaring PL/SQL Variables
==========
To use the variables or constants we need to declare them.

The syntax to declare variables or constants are:
identifier [CONSTANT] datatype [NOT NULL] [:= | DEFAULT expr];

Examples
```
DECLARE
  v_hiredate     DATE;
  v_deptno  NUMBER(2) NOT NULL := 10;
  v_location     VARCHAR2(13) := 'Delhi';
  c_comm    CONSTANT NUMBER := 1400;
```

NOTE: Observe the coding convention/style

* Let variable name start with the letter 'v'
* Let constant name start with the letter 'c'

Guidelines to declaring Variables

==========
• Follow naming conventions.
• Initialize variables designated as NOT NULL and CONSTANT.
• Declare one identifier per line.
• Initialize identifiers by using the assignment
  operator (:=) or the DEFAULT reserved word.

REMEMBER, the assignment operator is a COLON followed by EQUAL sign
No space in between.

Basic Scalar Data Types
==========
Scalar = single value

The different scalar data types supported by PL/SQL are:
• VARCHAR2 (maximum_length)
• NUMBER [(precision, scale)]
• DATE
• CHAR [(maximum_length)]
• LONG
• LONG RAW
• BOOLEAN
• BINARY_INTEGER
• PLS_INTEGER

Declaring Datatype with %TYPE Attribute
==========
What should be done if we want a variable to be of a particular column
type
of a table?

1st method - Look into the description of the table
             and identify the different column & its data type
             Use the same for the variables.

2nd method - Use the %TYPE attribute

Example:
```
      v_empname         emp.ename%TYPE;
      v_salary          emp.sal%TYPE;
```

We can also use %TYPE to declare variable of another previously declared
variable.

Example:
```
      v_count                NUMBER(2) := 1;
      v_total                v_count%TYPE;
      -- v_total is of the same data type as that of v_count
```

Declaring Datatype with %ROWTYPE Attribute
==========
To access the entire row of a database table the %ROWTYPE attribute can
be
used.

Example
```
      v_deptRecord           dept%ROWTYPE;
```

Composite Data Types
====================
TBD

==
======
Control Structures
==================
What are Control Structures?
~~~~~~~~~~~~~~~~~~~~~~~~~~~
Control Structures CONTROL THE FLOW of execution of the statement.

By default the statements are executed in a SEQUENTIAL manner.
i.e. 1st statement first, 2nd statement second and so on.

Types
[1] Selection / Decision Making

[2] Looping / Iterative

[3] Branch / Jump

Controlling PL/SQL Flow of Execution
====================================
• You can change the logical execution of statements using conditional
  IF statements and loop control structures.

• Conditional IF statements:
  – IF-THEN-END IF
  – IF-THEN-ELSE-END IF
  – IF-THEN-ELSIF-END IF

The IF Statement
================
Syntax:
```
IF condition THEN
  statements;
[ELSIF condition THEN
  statements;]
[ELSE
  statements;]
END IF;
```

Example:
If the employee name is Gietz, set the Manager ID to 102.

```
IF UPPER(v_last_name) = 'GIETZ' THEN
   v_mgr := 102;
END IF;
```

NOTE: Using ELSE is optional

Example
Set a Boolean flag to TRUE if the hire date is greater than five years;
otherwise, set the Boolean flag to FALSE.

```
DECLARE
v_hire_date DATE := '12-Dec-1990';
```
~~~~~~~~~~~~~~~~~~~~~~~~~~~

```
v_five_years BOOLEAN;
BEGIN
:
IF MONTHS_BETWEEN(SYSDATE,v_hire_date)/12 > 5 THEN
      v_five_years := TRUE;
ELSE
      v_five_years := FALSE;
END IF;
:
```

Compound IF Statement
==========
If the last name is Vargas and the salary is more than 6500:

Set department number to 60.

Example:

```
:
IF v_ename = 'Vargas' AND salary > 6500 THEN
      v_deptno := 60;
END IF;
:
```

Multi-Way Branching
==========
Multi-way branching is done using IF...ELSEIF statement.
Check for MULTIPLE conditions.

Syntax:

```
IF Condition_Expr_1
THEN
      PL/SQL_Statements_1 ;
ELSIF Condition_Expr_2
THEN
      PL/SQL_Statements_2 ;
ELSIF Condition_Expr_3
THEN
      PL/SQL_Statements_3 ;
ELSE
      PL/SQL_Statements_n
END IF;
```

Given the day of the week (as a number), display the weekday.
Care should be taken for INVALID weekday.

Example
For a given value, calculate a percentage of that value based on a condition.

```
:
IF v_start > 100 THEN
      v_start := 0.2 * v_start;
ELSIF v_start >= 50 THEN
      v_start := 0.5 * v_start;
ELSE
```

```
      v_start := 0.1 * v_start;
END IF;
:
```

CASE Expressions
==========
Similar to SWITCH...CASE in C/C++/Java

• A CASE expression selects a result and returns it.
• To select the result, the CASE expression uses an expression whose value
  is used to select one of several alternatives.

```
Syntax:
CASE selector
  WHEN expression1 THEN result1
  WHEN expression2 THEN result2
  ...
  WHEN expressionN THEN resultN
  [ELSE resultN+1;]
END;
```

[2] Loop / Iterative
• Loops repeat a statement or sequence of statements multiple times.
• There are three loop types:
  – Basic loop
  – FOR loop
  – WHILE loop

Basic Loops AKA Simple Loops
==========
```
Syntax:
LOOP                              <-- Delimiter
      statement1;                 <-- Statement(s)
      :
      :
      EXIT [WHEN condition];            <-- EXIT condition
END LOOP;                         <-- Delimiter
```

The loop terminates when the condition become TRUE.

EXIT Statement
==========
Exit path is provided by using EXIT or EXIT WHEN statements.

A plain EXIT is termed as an UNCONDITIONAL exit.
However, EXIT WHEN is a CONDITIONAL exit.

```
Example:
DECLARE
      v_counter        NUMBER(2) := 1;
BEGIN
      DBMS_OUTPUT.PUT_LINE('Start');
      LOOP
            DBMS_OUTPUT.PUT_LINE( v_counter );
            -- Incrment the counter by 1
            v_counter := v_counter + 1;
```

```
            EXIT WHEN v_counter > 10;
      END LOOP;
      DBMS_OUTPUT.PUT_LINE('End');
END;
/
```

The above PL/SQL code will generate integer from 1 to 10 in between
'Start'
and 'End'

FOR Loop
========
Syntax:

```
FOR counter IN [REVERSE] lower_bound..upper_bound LOOP
   statement1;
   statement2;
   . . .
END LOOP;
```

- Use a FOR loop to shortcut the test for the number of iterations.
- Do not declare the 'counter' it is declared implicitly.
- 'lower_bound .. upper_bound' is required syntax.

Example:

```
BEGIN
      DBMS_OUTPUT.PUT_LINE('The FOR Loop');
      FOR i IN 1..10 LOOP
            DBMS_OUTPUT.PUT_LINE( i );
      END LOOP;
      DBMS_OUTPUT.PUT_LINE('End');
      DBMS_OUTPUT.PUT_LINE('The FOR Loop - REVERSE clause');
      FOR n IN REVERSE 1..5 LOOP
            DBMS_OUTPUT.PUT_LINE( n );
      END LOOP;
END;
```

NOTE: Observe the REVERSE clause.
      FOR n IN REVERSE 1..5 LOOP will generate numbers from
      5 to 1 in reverse order.

The WHILE Loop
==============
It is a TOP-TESTED loop. i.e. Condition is check first and statements are executed later.

Syntax:

```
WHILE condition LOOP
  statement1;
  statement2;
  . . .
END LOOP;
```

The statement(s) will be executed as long as the condtion is TRUE.
Once the condition becomes FALSE, the loop terminates.

Example:

```
DECLARE
      v_counter        NUMBER(2) := 1;
```

```
BEGIN
      DBMS_OUTPUT.PUT_LINE('The WHILE Loop');
      WHILE v_counter <= 10 LOOP
            DBMS_OUTPUT.PUT_LINE( 'Welcome ' || v_counter || ' time(s).'
);
            -- Incrment the counter by 1
            v_counter := v_counter + 1;
      END LOOP;
      DBMS_OUTPUT.PUT_LINE('End');
END;
```

Nested Loops
============
When we can have one loop inside another loop, it is termed as NESTED loops.

However, when we use EXIT statment to come out of nested loops, it comes out
or terminates ONLY from the inner most loop.

To exit all the loops in nested loops, we need to give label to the EXIT statement as follows:

```
      EXIT <loop_label>;
```

LABELs are specified in the PL/SQL code as INDENTIFERs enclosed in DOUBLE ANGLE Brackets as follows:

```
      <<LABEL_1>>
```