Chapter 11
PL/SQL (part 1)

About PL/SQL

- PL/SQL is an extension to SQL with design features of programming languages.
- Data manipulation and query statements of SQL are included within procedural units of code.

Benefits of PL/SQL

- Integration
- Improved Performance
- Modularize program development
- It is portable.
- You can declare identifiers.
- You can program with procedural language control structures.
- It can handle errors.

PL/SQL Block Structure

- DECLARE – Optional
  - Variables, cursors, user-defined exceptions
- BEGIN – Mandatory
  - SQL statements
  - PL/SQL statements
- EXCEPTION – Optional
  - Actions to perform when errors occur
- END; – Mandatory

```
DECLARE
  v_variable  VARCHAR2(5);
BEGIN
  SELECT     column_name
    INTO     v_variable
    FROM     table_name;
EXCEPTION
  WHEN exception_name THEN
  ...
END;
```

Block Types

Anonymous

```
[DECLARE]
BEGIN
  --statements
[EXCEPTION]
END
```

Procedure

```
PROCEDURE name
IS
BEGIN
  --statements
[EXCEPTION]
END;
```

Function

```
FUNCTION name
RETURN datatype
IS
BEGIN
  --statements
  RETURN value;
[EXCEPTION]
END;
```

Program Constructs

- Anonymous block
- Stored procedure/function
- Application trigger
- Application procedure/function
- Database trigger
- Packaged procedure/function

Use of Variables

- Use variables 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 (varchar2, number, date, char, long, boolean)
  - Composite (PL/SQL tables, PL/SQL records)
  - Reference (%TYPE, %ROWTYPE)
  - LOB (large objects)
- Non-PL/SQL variables: Bind and host variables

Declaring Variables

Syntax

```
identifier [CONSTANT] datatype [NOT NULL]
          [:= | DEFAULT expr];
```

Examples

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

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

Naming Rules

- Two variables can have the same name, provided they are in different blocks.
- The variable name (identifier) should not be the same as the name of table columns used in the block.

```
DECLARE
  empno   NUMBER(4);
BEGIN
  SELECT        empno
  INTO    empno
  FROM    emp
  WHERE         ename = 'SMITH';
END;
```

Assigning Values to Variables

Syntax

- *identifier* := *expr*;

Examples

Set a predefined hiredate for new employees.

```
v_hiredate := '31-OCT-01';
```

Set employee name to Weston

```
v_ename := 'Weston';
```

Variable Initialization and Keywords

- Using:
  - Assignment operator (:=)
  - DEFAULT keyword
  - NOT NULL constraint

Scalar Datatypes

- Hold a single value
- Have no internal components

Base Scalar Datatypes

–VARCHAR2 (*maximum_length*)
–NUMBER [(*precision, scale*)]
–DATE
–CHAR [(*maximum_length*)]
–LONG
–LONG RAW
–BOOLEAN
–BINARY_INTEGER
–PLS_INTEGER

Scalar Variable Definitions

Examples

```
v_job                   VARCHAR2(9);
v_count             BINARY_INTEGER := 0;
v_total_sal         NUMBER(9,2) := 0;
v_orderdate         DATE := SYSDATE + 7;
c_tax_rate          CONSTANT NUMBER(3,2) := 8.25;
v_valid             BOOLEAN NOT NULL := TRUE;
```

The %TYPE Attribute

- Declare a variable according to:
  - A database column definition
  - Another previously declared variable
- Prefix %TYPE with:
  - The database table and column
  - The previously declared variable name

Declaring Variables with the %TYPE Attribute

Examples

```
...  v_ename                  emp.ename%TYPE;
     v_balance                NUMBER(7,2);
     v_min_balance            v_balance%TYPE := 10;
...
```

Declaring Boolean Variables

- Only the values TRUE, FALSE, and NULL can be assigned to a Boolean variable.
- The variables are connected by the logical operators AND, OR, and NOT.
- The variables always yield TRUE, FALSE, or NULL.
- Arithmetic, character, and date expressions can be used to return a Boolean value.

Creating Bind Variables

- To declare a bind variable in the SQL*Plus environment, you can use the command VARIABLE.
- For example, you declare a variable of type NUMBER and VARCHAR2 as follows:

```
SQL> VARIABLE my_number NUMBER
SQL> VARIABLE my_char   VARCHAR2
```

Using the Bind Variable

```
SQL> VARIABLE return_num NUMBER

CREATE OR REPLACE PROCEDURE testing
(v_in_char IN VARCHAR2, v_out_num NUMBER)
AS
BEGIN
    ... do something with v_in_char
    ... do something to produce v_out_num
END;
/
SQL> EXECUTE testing ('my varchar', :return_num)
   /* 'my varchar' will be assigned to v_in_char
      v_out_num will be assigned to :return_num
         which is then printed using the statement
          below */
SQL> PRINT return_num
```

Bind Variable Example - Procedure

```
CREATE OR REPLACE PROCEDURE determine_salary
(v_ename varchar2, v_sal out number)
AS
BEGIN
 SELECT sal
 INTO v_sal
 FROM emp
 WHERE upper(ename) = upper(v_ename);
END;
/
VARIABLE var_sal NUMBER;
EXECUTE determine_salary('ford',:var_sal);
PRINT var_sal
```

Bind Variable Example - Function

```
CREATE OR REPLACE FUNCTION double_num
  (v_number IN NUMBER)
  return number
AS
  con_multiplier CONSTANT NUMBER := 2;
  v_result number;
BEGIN
  v_result := con_multiplier * v_number;
  return v_result;
END;
/

SELECT double_num(4) FROM DUAL;
```

Referencing Non-PL/SQL Variables

- Store the annual salary into a SQL*Plus host variable.
  - Reference non-PL/SQL variables as host variables.
  - Prefix the references with a colon (:).

Character Strings

- Concatenation

  SFirstName Sarah
  SLastName Miller

  SFullName := SFirstName || ' ' || SLastName;

  ⟶ Sarah Miller

- Literals: are unchanging data values coded directly into a program
  - Eg. 3.14159 for pi
- Always enclose a string literal in single quotes
  - Eg. ‘Marla Weston’

- Some functions useful with strings
  - RTRIM(): removes all trailing spaces
  - LENGTH(): returns the length of a string
  - UPPER(): converts string to uppercase
  - LOWER(): converts string to lowercase
  - INSTR(): search for matching substring
  - SUBSTR(): extract a substring from string

DBMS_OUTPUT.PUT_LINE

- An Oracle-supplied packaged procedure
- An alternative for displaying data from a PL/SQL block
- Must be enabled in SQL*Plus with
- SET SERVEROUTPUT ON

IF/THEN/ELSE

- Programming structure

```
IF <condition> THEN
   <statements that execute if TRUE>;
ELSE
   <statements that execute if FALSE>;
END IF;
```

IF/ELSIF

```
IF <condition1> THEN
   < statements that execute when condition1 is TRUE >;
ELSIF <condition2> THEN
   < statements that execute when condition2 is TRUE >;
ELSIF <condition3> THEN
    < statements that execute when condition3 is TRUE >;
…
ELSE
   <statements that execute if none of the above are TRUE>;
END IF;
```

Iterative Control: LOOP Statements

– Loops repeat a statement or sequence of statements multiple times.
– There are three loop types:
  - Basic loop
  - FOR loop
  - WHILE loop

●Syntax

```
LOOP
  statement1;
  . . .
  EXIT [WHEN condition];
END LOOP;

where: condition          is a Boolean variable or
                          expression (TRUE, FALSE,
                          or NULL);
```

●Example

```
DECLARE
  v_ordid        item.ordid%TYPE := 601;
  v_counter         NUMBER(2) := 1;
BEGIN
  LOOP
    INSERT INTO item(ordid, itemid)
      VALUES(v_ordid, v_counter);
    v_counter := v_counter + 1;
    EXIT WHEN v_counter > 10;
  END LOOP;
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 index; it is declared implicitly.

●Guidelines

– Reference the counter within the loop only; it is undefined outside the loop.
– Use an expression to reference the existing value of a counter.
– Do *not* reference the counter as the target of an assignment.

- Insert the first 15 new line items for order number 411.
- Example

```
DECLARE
  v_ordid        item.ordid%TYPE := 411;
BEGIN
  FOR i IN 1..15 LOOP
    INSERT INTO item(ordid, itemid)
      VALUES(v_ordid, i);
  END LOOP;
END;
```

WHILE Loop

- Syntax

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

- Use the WHILE loop to repeat statements while a condition is TRUE.

- Example

```
ACCEPT p_new_order PROMPT 'Enter the order number: '
ACCEPT p_items PROMPT 'Enter the number of items in this order: '
DECLARE
v_count          NUMBER(2) := 1;
BEGIN
  WHILE v_count <= &p_items LOOP
    INSERT INTO item (ordid, itemid)
    VALUES (&p_new_order, v_count);
    v_count := v_count + 1;
  END LOOP;
  COMMIT;
END;
/
```

Nested Loops and Labels

- Nest loops to multiple levels.
- Use labels to distinguish between blocks and loops.
- Exit the outer loop with the EXIT statement referencing the label.

```
...
BEGIN
  <<Outer_loop>>
  LOOP
    v_counter := v_counter+1;
  EXIT WHEN v_counter>10;
    <<Inner_loop>>
    LOOP
      ...
      EXIT Outer_loop WHEN total_done = 'YES';
      -- Leave both loops
      EXIT WHEN inner_done = 'YES';
      -- Leave inner loop only
      ...
    END LOOP Inner_loop;
    ...
  END LOOP Outer_loop;
END;
```

Summary

- PL/SQL blocks are composed of the following sections:
  - Declarative (optional)
  - Executable (required)
  - Exception handling (optional)
- A PL/SQL block can be an anonymous block, procedure, or function.
- PL/SQL identifiers:
  - Are defined in the declarative section
  - Can be of scalar, composite, reference, or LOB datatype
  - Can be based on the structure of another variable or database object
  - Can be initialized

- Change the logical flow of statements by using control structures.
  - Conditional (IF statement)
  - Loops:
    - Basic loop
    - FOR loop
    - WHILE loop
    - EXIT statement