Introduction to PL/SQL, Flow Control Statements

Content

- Overview of PL/SQL
- Advantages of PL/SQL
- Architecture of PL/SQL
- PL/SQL Language Fundamentals
- PL/SQL Data Types
- PL/SQL Control Statements

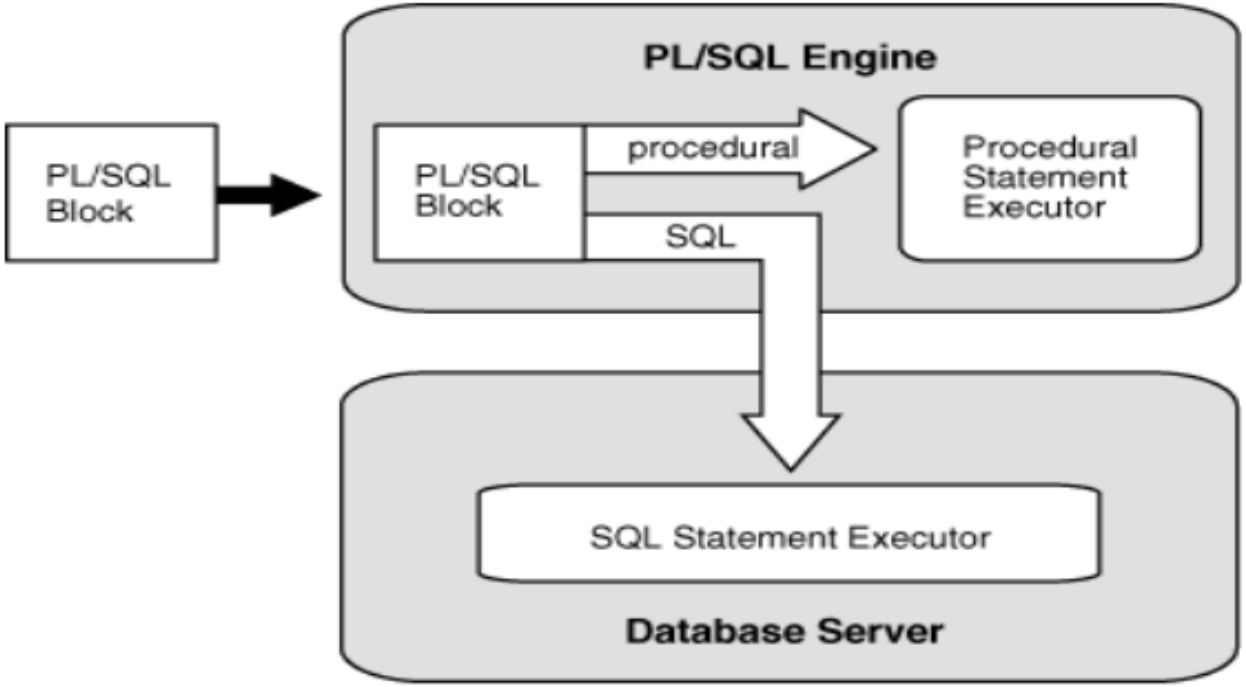
Overview of PL/SQL

- PL/SQL, the Oracle procedural extension of SQL, is a portable, high-performance transaction-processing language.

Advantages of PL/SQL:

- Tight Integration with SQL
- High Performance and Productivity
- Portability
- Scalability
- Manageability
- Support for Object-Oriented Programming

Architecture of PL/SQL



Architecture of PL/SQL

PL/SQL Engine

- The PL/SQL compilation and runtime system is an engine that compiles and runs PL/SQL units.
- The engine can be installed in the database or in an application development tool, such as Oracle Forms.
- In either environment, the PL/SQL engine accepts as input any valid PL/SQL unit. The engine runs procedural statements, but sends SQL statements to the SQL engine in the database

Architecture of PL/SQL

PL/SQL Units and Compilation Parameters

- PL/SQL units are affected by PL/SQL compilation parameters.
- Different PL/SQL units
- A PL/SQL unit is one of these:
  - PL/SQL anonymous block
  - FUNCTION
  - PACKAGE
  - PROCEDURE
  - TRIGGER

The PL/SQL program structure divides the code into blocks distinguished by the following keywords: DECLARE , BEGIN , EXCEPTION , and END . An unnamed PL/SQL code block (code not stored in the database as a procedure, function, or package) is known as an anonymous block.

PL/SQL Language Fundamentals

- A PL/SQL language fundamental components are:
  - Declarations
  - Scope and Visibility of Identifiers
  - Assigning Values to Variables
  - Expressions

PL/SQL Language Fundamentals

Declarations:

- A declaration allocates storage space for a value of a specified data type, and names the storage location so that you can reference it.
- You must declare objects before you can reference them.
- Declarations can appear in the declarative part of any block, subprogram, or package.

```
DECLARE
        product_name VARCHAR2( 100 ) := 'Laptop';
BEGIN
        NULL;
END;
```

PL/SQL Language Fundamentals

- A variable declaration always specifies the name and data type of the variable.
- For most data types, a variable declaration can also specify an initial value.
- The variable name must be a valid user-defined identifier .

PL/SQL Language Fundamentals

Example of variable Declarations:

```
DECLARE
        acct_id INTEGER(4) NOT NULL := 9999;  -- Variable Declaration with NOT NULL Constraint
        name    VARCHAR(25) NOT NULL := 'Smith'; -- Variable Declaration with NOT NULL
Constraint
        counter INTEGER;  -- Variable is initialized to NULL by Default
        credit_limit    CONSTANT REAL   := 5000.00;  -- Constant variable declaration
        hours_worked    INTEGER := 40; -- Initial Value set for the variable
        in_stock        BOOLEAN;      -- variable of type boolean
        surname employees.last_name%TYPE; -- Declaring Items using the %TYPE Attribute
BEGIN
        NULL;
END;
/
```

PL/SQL Language Fundamentals

Scope and Visibility of Identifiers:

- The scope of an identifier is the region of a PL/SQL unit from which you can reference the identifier.
- The visibility of an identifier is the region of a PL/SQL unit from which you can reference the identifier without qualifying it.
- An identifier is local to the PL/SQL unit that declares it.
- If that unit has subunits, the identifier is global to them.

PL/SQL Language Fundamentals

Example for Scope and Visibility of Identifiers:

```
-- Outer block:
DECLARE
  a CHAR;  -- Scope of a (CHAR) begins
  b REAL;    -- Scope of b begins
BEGIN
  -- Visible: a (CHAR), b

  -- First sub-block:
  DECLARE
    a INTEGER;  -- Scope of a (INTEGER) begins
    c REAL;       -- Scope of c begins
  BEGIN
    -- Visible: a (INTEGER), b, c
    NULL;
  END;          -- Scopes of a (INTEGER) and c end

  -- Second sub-block:
  DECLARE
    d REAL;     -- Scope of d begins
  BEGIN
    -- Visible: a (CHAR), b, d
    NULL;
  END;          -- Scope of d ends

-- Visible: a (CHAR), b
END;            -- Scopes of a (CHAR) and b end
/
```

Assigning values to Variables:

After declaring a variable, you can assign a value to it in these ways:

- Use the assignment statement to assign it the value of an expression.
- Use the SELECT INTO or FETCH statement to assign it a value from a table.

PL/SQL Language Fundamentals

Example for Assigning values to the variables:

```
DECLARE  -- You can assign initial values here
 hours_worked   NUMBER := 40;
 wages        NUMBER;
 hourly_salary  NUMBER := 22.50;
 bonus   NUMBER(8,2);
BEGIN  -- You can assign values here too
 wages := (hours_worked * hourly_salary) + bonus;
 SELECT salary * 0.10 INTO bonus FROM employees WHERE employee_id = 100;
END;
/
```

PL/SQL Data Types

- Every PL/SQL constant, variable, parameter, and function return value has a data type.
- The PL/SQL data types are:
    - The SQL data types
    - BOOLEAN,
    - PLS_INTEGER
    - BINARY_INTEGER
    - REF CURSOR

PL/SQL Control Statements

PL/SQL :categories of control statements:

- Conditional selection statements
- Loop statements

Conditional selection statements

- The conditional selection statements run different statements for different data values.
- The conditional selection statements are:
  - ✓ IF
  - ✓ CASE

PL/SQL Control Statements

Conditional selection statements

- The IF statement has these forms:
  - IF THEN
  - IF THEN ELSE
  - IF THEN ELSIF
- The CASE statement has these forms:
  - Simple, which evaluates a single expression and compares it to several potential values.
  - Searched, which evaluates multiple conditions and chooses the first one that is true.

PL/SQL Control Statements

Conditional selection statements – IF THEN STATEMENT

- The IF THEN statement either runs or skips a sequence of one or more statements, depending on a condition.
- **Syntax:**

```
IF condition THEN
 statements
END IF;
```

Example

```
IF new_balance < minimum_balance THEN
        overdrawn := TRUE;
END IF;
```

PL/SQL Control Statements

Conditional selection statements – IF THEN ELSE STATEMENT

- **Syntax:**

```
IF condition THEN
 statements
ELSE
 else statements
END IF;
```

Example

```
IF new_balance < minimum_balance THEN
        overdrawn := TRUE;
ELSE
        overdrawn :=FALSE
END IF;
```

PL/SQL Control Statements

Conditional selection statements – IF THEN ELSIF STATEMENT

- **Syntax:**

```
IF condition_1 THEN
        statements_1
ELSIF condition_2 THEN
        statements_2
[ ELSIF condition_3 THEN
        statements_3
]...
[ ELSE
        else_statements
]
END IF;
```

- **Example:**

```
IF sales > 50000 THEN
        bonus := 1500;
ELSIF sales > 35000 THEN
        bonus := 500;
ELSE
        bonus := 100;
END IF;
```

PL/SQL Control Statements

Conditional selection statements – Simple CASE STATEMENT

- **Syntax:**

```
CASE selector
WHEN selector_value_1 THEN statements_1
WHEN selector_value_2 THEN statements_2
...
WHEN selector_value_n THEN statements_n
[ ELSE
else_statements ]
END CASE;]
```

- **Example:**

```
CASE grade
WHEN 'A' THEN DBMS_OUTPUT.PUT_LINE('Excellent');
WHEN 'B' THEN DBMS_OUTPUT.PUT_LINE('Very Good');
WHEN 'C' THEN DBMS_OUTPUT.PUT_LINE('Good');
WHEN 'D' THEN DBMS_OUTPUT.PUT_LINE('Fair');
WHEN 'F' THEN DBMS_OUTPUT.PUT_LINE('Poor');
ELSE DBMS_OUTPUT.PUT_LINE('No such grade');
END CASE;
```

PL/SQL Control Statements

Conditional selection statements – Searched CASE STATEMENT

- Syntax:

```
CASE
WHEN condition_1 THEN statements_1
WHEN condition_2 THEN statements_2
...
WHEN condition_n THEN statements_n
[ ELSE
 else_statements ]
END CASE;]
```

- Example:

```
CASE
WHEN grade = 'A' THEN DBMS_OUTPUT.PUT_LINE('Excellent');
WHEN grade = 'B' THEN DBMS_OUTPUT.PUT_LINE('Very Good');
WHEN grade = 'C' THEN DBMS_OUTPUT.PUT_LINE('Good');
WHEN grade = 'D' THEN DBMS_OUTPUT.PUT_LINE('Fair');
WHEN grade = 'F' THEN DBMS_OUTPUT.PUT_LINE('Poor');
ELSE DBMS_OUTPUT.PUT_LINE('No such grade');
END CASE;
```

PL/SQL Control Statements

LOOP statements

- Loop statements run the same statements with a series of different values. The loop statements are:
  - Basic LOOP
  - FOR LOOP
  - Cursor FOR LOOP
  - WHILE LOOP

PL/SQL Control Statements

LOOP statements

- The statements that exit a loop are:
  - EXIT
  - EXIT WHEN
- The statements that exit the current iteration of a loop are:
  - CONTINUE
  - CONTINUE WHEN

PL/SQL Control Statements

LOOP statements – Basic LOOP STATEMENT

- **Syntax:**

```
[ label ] LOOP
        statements
END LOOP [ label ];
```

LOOP statements – EXIT STATEMENT

The EXIT statement exits the current iteration of a loop unconditionally and transfers control to the end of either the current loop.

- **Syntax:**

```
EXIT;
```

PL/SQL Control Statements

Example– Basic LOOP statement with EXIT statement

- **Example:**

```
DECLARE
        x NUMBER := 0;
BEGIN
        LOOP
        DBMS_OUTPUT.PUT_LINE ('Inside loop:  x = ' || TO_CHAR(x));
        x := x + 1;
        IF x > 3 THEN
        EXIT;
        END IF;
        END LOOP;
        DBMS_OUTPUT.PUT_LINE(' After loop:  x = ' || TO_CHAR(x));
END;
/
```

```
Inside loop:  x = 0
Inside loop:  x = 1
Inside loop:  x = 2
Inside loop:  x = 3
 After loop:  x = 4
```

PL/SQL Control Statements

LOOP statements – EXIT WHEN STATEMENT

- The EXIT WHEN statement exits the current iteration of a loop when the condition in its WHEN clause is true, and transfers control to the end of either the current loop or an enclosing labeled loop.
- Each time control reaches the EXIT WHEN statement, the condition in its WHEN clause is evaluated.
- If the condition is not true, the EXIT WHEN statement does nothing.

Syntax:

```
EXIT WHEN condition;
```

PL/SQL Control Statements

Example– Basic LOOP statement with EXIT WHEN statement

- Example:

```
DECLARE
x NUMBER := 0;
BEGIN
LOOP
DBMS_OUTPUT.PUT_LINE ('Inside loop:  x = ' || TO_CHAR(x));
x := x + 1;
EXIT WHEN x>3;
END LOOP;
DBMS_OUTPUT.PUT_LINE(' After loop:  x = ' || TO_CHAR(x));
END;
```

```
Inside loop:  x = 0
Inside loop:  x = 1
Inside loop:  x = 2
Inside loop:  x = 3
 After loop:  x = 4

Statement processed.
```

PL/SQL Control Statements

LOOP statements – CONTINUE STATEMENT

- The CONTINUE statement exits the current iteration of a loop unconditionally and transfers control to the next iteration of either the current loop or an enclosing labeled loop.
- **Syntax:**

```
CONTINUE;
```

PL/SQL Control Statements

Example– Basic LOOP statement with CONTINUE statement

```
DECLARE
 x NUMBER := 0;
BEGIN
LOOP -- After CONTINUE statement, control resumes here
DBMS_OUTPUT.PUT_LINE ('Inside loop:  x = ' || TO_CHAR(x));
x := x + 1;
IF x < 3 THEN
CONTINUE;
END IF;
DBMS_OUTPUT.PUT_LINE('Inside loop, after CONTINUE:  x = ' || TO_CHAR(x));
EXIT WHEN x = 5;
END LOOP;
DBMS_OUTPUT.PUT_LINE (' After loop:  x = ' || TO_CHAR(x));
END;
/
```

```
Inside loop:  x = 0
Inside loop:  x = 1
Inside loop:  x = 2
Inside loop, after CONTINUE:  x = 3
Inside loop:  x = 3
Inside loop, after CONTINUE:  x = 4
Inside loop:  x = 4
Inside loop, after CONTINUE:  x = 5
 After loop:  x = 5

Statement processed.
```

PL/SQL Control Statements

LOOP statements – CONTINUE WHEN STATEMENT

- The CONTINUE WHEN statement exits the current iteration of a loop when the condition in its WHEN clause is true, and transfers control to the next iteration of either the current loop or an enclosing labeled loop.
- Each time control reaches the CONTINUE WHEN statement, the condition in its WHEN clause is evaluated. If the condition is not true, the CONTINUE WHEN statement does nothing.
- **Syntax:**

```
CONTINUE WHEN condition;
```

PL/SQL Control Statements

Example– Basic LOOP statement with CONTINUE WHEN statement

```
DECLARE
x NUMBER := 0;
BEGIN
LOOP -- After CONTINUE statement, control resumes here
DBMS_OUTPUT.PUT_LINE ('Inside loop:  x = ' || TO_CHAR(x));
x := x + 1;
CONTINUE WHEN x < 3;
DBMS_OUTPUT.PUT_LINE('Inside loop, after CONTINUE:  x = ' || TO_CHAR(x));
EXIT WHEN x = 5;
END LOOP;
DBMS_OUTPUT.PUT_LINE (' After loop:  x = ' || TO_CHAR(x));
END;
/
```

```
Inside loop:  x = 0
Inside loop:  x = 1
Inside loop:  x = 2
Inside loop, after CONTINUE:  x = 3
Inside loop:  x = 3
Inside loop, after CONTINUE:  x = 4
Inside loop:  x = 4
Inside loop, after CONTINUE:  x = 5
 After loop:  x = 5

Statement processed.
```

PL/SQL Control Statements

LOOP statements – FOR LOOP STATEMENT

- The FOR LOOP statement runs one or more statements while the loop index is in a specified range.
- The statement has this structure:
- **Syntax:**

```
[ label ] FOR index IN [ REVERSE ] lower_bound..upper_bound
LOOP
        statements
END LOOP [ label ];
```

PL/SQL Control Statements

Example– FOR LOOP statement

```
BEGIN
 DBMS_OUTPUT.PUT_LINE ('lower_bound < upper_bound');
 FOR i IN 1..3 LOOP
  DBMS_OUTPUT.PUT_LINE (i);
 END LOOP;
 DBMS_OUTPUT.PUT_LINE ('lower_bound = upper_bound');
 FOR i IN 2..2 LOOP
  DBMS_OUTPUT.PUT_LINE (i);
 END LOOP;
 DBMS_OUTPUT.PUT_LINE ('lower_bound > upper_bound');
 FOR i IN 3..1 LOOP
  DBMS_OUTPUT.PUT_LINE (i);
 END LOOP;
END;
/
```

```
lower_bound < upper_bound
1
2
3
lower_bound = upper_bound
2
lower_bound > upper_bound

Statement processed.
```

THANK YOU