Oracle PL/SQL Introduction

Oracle Academy Study Materials

Introduction to PL/SQL

PL/SQL:

- Stands for Procedural Language extension to SQL
- Is Oracle Corporation’s standard data access language for relational databases
- Seamlessly integrates procedural constructs with SQL
- Provides a block structure for executable units of code. Maintenance of code is made easier with such a well-defined structure.
- Provides procedural constructs such as:
  - Variables, constants, and types
  - Control structures such as conditional statements and loops
  - Reusable program units that are written once and executed many times

Benefits of PL/SQL

- Integration of procedural constructs with SQL
  - SQL is a nonprocedural language,
  - PL/SQL integrates control statements and conditional statements with SQL
- Improved performance
  - With PL/SQL, you can combine all these SQL statements into a single program unit. The application can send the entire block to the database instead of sending the SQL statements one at a time. This significantly reduces the number of database calls.
  - you can use PL/SQL blocks to group SQL statements before sending them to the Oracle database server for execution.

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)

Introduction to PL/SQL

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;
```

Use of Variables

<u>**Variables can be used for:**</u>

- Temporary storage of data
- Manipulation of stored values
- Reusability

<u>**A variable name:**</u>

- Must start with a letter
- Can include letters or numbers
- Can include special characters (such as $, _, and #)
- Must contain no more than 30 characters
- Must not include reserved words

<u>**Variables are:**</u>

- Declared and initialized in the declarative section
- Used and assigned new values in the executable section
- Passed as parameters to PL/SQL subprograms
- Used to hold the output of a PL/SQL subprogram

Declaring and Initializing PL/SQL Variables

Syntax

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

Examples

```
DECLARE
  emp_hiredate      DATE;
  emp_deptno        NUMBER(2) NOT NULL := 10;
  location          VARCHAR2(13) := 'Atlanta';
  c_comm            CONSTANT NUMBER := 1400;
```

Declaring and Initializing PL/SQL Variables

Types of PL/SQL variables:

– Scalar
– Composite
– Reference
– Large object (LOB)
– Non-PL/SQL variables: Bind variables

Guidelines for declaring and initializing PL/SQL variables

- Use meaningful identifiers for variables.
- Initialize variables designated as `NOT NULL` and `CONSTANT`.
- Initialize variables with the assignment operator (:=) or the `DEFAULT` keyword
- Declare one identifier per line for better readability and code maintenance.
- Avoid using column names as identifiers

%TYPE Attribute

The %TYPE attribute

- Is used to declare a variable according to:
  - A database column definition
  - Another declared variable
- Is prefixed with:
  - The database table and column
  - The name of the declared variable

Syntax

```
identifier              table.column_name%TYPE;
```

%TYPE Attribute

Advantages of the %TYPE Attribute

- You can avoid errors caused by data type mismatch or wrong precision.
- You can avoid hard coding the data type of a variable.
- You need not change the variable declaration if the column definition changes.
  - If you have already declared some variables for a particular table without using the %TYPE attribute, the PL/SQL block may throw errors if the column for which the variable is declared is altered.
  - When you use the %TYPE attribute, PL/SQL determines the data type and size of the variable when the block is compiled. This ensures that such a variable is always compatible with the column that is used to populate it.

SQL Functions in PL/SQL

- Available in procedural statements:
  - Single-row functions: character functions, data type conversion functions, and date and time-stamp functions
- Not available in procedural statements:
  - DECODE
  - Group functions

Operators in PL/SQL

- Logical
- Arithmetic
- Concatenation
- Parentheses to control order of operations

Same as in SQL

- Exponential operator (**)

Programming Guidelines

Make code maintenance easier by:

- Documenting code with comments
- Developing a case convention for the code
- Developing naming conventions for identifiers and other objects
- Enhancing readability by indenting
- For clarity, indent each level of code.

SQL Statements in PL/SQL

- Retrieve a row from the database by using the SELECT command.
- Make changes to rows in the database by using DML commands.
- Control a transaction with the COMMIT, ROLLBACK, or SAVEPOINT command.
- The END keyword signals the end of a PL/SQL block, not the end of a transaction
  - a block can span multiple transactions,
  - a transaction can span multiple blocks.
- PL/SQL does not directly support data definition language (DDL) statements
  - DDL statements cannot be directly executed.
  - These statements are dynamic SQL statements.
- PL/SQL does not directly support data control language (DCL) statements
  - You can use dynamic SQL to execute them.

SELECT Statements in PL/SQL

- Retrieve data from the database with a SELECT statement.
- Syntax:

```
SELECT   select_list
INTO     {variable_name[, variable_name]...| record_name}
FROM     table
[WHERE   condition];
```

- Terminate each SQL statement with a semicolon (;).
- Every value retrieved must be stored in a variable by using the INTO clause.
- The WHERE clause is optional and can be used to specify input variables, constants, literals, and PL/SQL expressions.

SELECT Statements in PL/SQL

- The INTO clause is mandatory and occurs between the SELECT and FROM clauses.
  - It is used to specify the names of variables that hold the values that SQL returns from the SELECT clause.
  - You must specify one variable for each item selected, and the order of the variables must correspond with the items selected.
- When you use the SELECT statement with the INTO clause, queries must return **only one row**
  - A query that returns more than one row or no row generates an error.
  - PL/SQL manages these errors by raising standard exceptions: NO_DATA_FOUND and TOO_MANY_ROWS.
  - Include a WHERE condition in the SQL statement so that the statement returns a single row.
- Use group functions, such as SUM, in a SQL statement, because group functions apply to groups of rows in a table.

Retrieving Data in PL/SQL: Example

```
DECLARE
 v_fname VARCHAR2(25);
BEGIN
 SELECT first_name INTO v_fname FROM employees WHERE employee_id=200;
 DBMS_OUTPUT.PUT_LINE(' First Name is : '||v_fname);
END;
```

```
DECLARE
 v_emp_hiredate   employees.hire_date%TYPE;
 v_emp_salary     employees.salary%TYPE;
BEGIN
  SELECT hire_date, salary INTO v_emp_hiredate, v_emp_salary
  FROM employees WHERE employee_id = 100;
END;
```

```
DECLARE
   v_sum_sal   NUMBER(10,2);
   v_deptno    NUMBER NOT NULL := 60;
BEGIN
  SELECT  SUM(salary) INTO v_sum_sal
  FROM employees WHERE department_id = v_deptno;
  DBMS_OUTPUT.PUT_LINE ('The sum of salary is ' || v_sum_sal);
END;
```

Naming Conventions

- Use a naming convention to avoid ambiguity in the WHERE clause.
- Avoid using database column names as identifiers.
- Syntax errors can arise because PL/SQL checks the database first for a column in the table.
- The names of local variables and formal parameters take precedence over the names of database tables.
- The names of database table columns take precedence over the names of local variables.

Manipulating Data Using PL/SQL

Make changes to database tables by using DML commands:

- `INSERT`
- `UPDATE`
- `DELETE`
- `MERGE`

Writing Executable Statements

%ROWTYPE Attribute

```
...
DEFINE employee_number = 124
 DECLARE
  emp_rec   employees%ROWTYPE;
 BEGIN
  SELECT * INTO emp_rec FROM employees
  WHERE  employee_id = &employee_number;
  INSERT INTO retired_emps(empno, ename, job, mgr, hiredate,
              leavedate, sal, comm, deptno)
  VALUES (emp_rec.employee_id, emp_rec.last_name, emp_rec.job_id,
          emp_rec.manager_id,emp_rec.hire_date, SYSDATE,
          emp_rec.salary, emp_rec.commission_pct,
          emp_rec.department_id);
END;
/
```

Writing Executable Statements

IF Statements

Syntax:

```
IF condition THEN
  statements;
[ELSIF condition THEN
  statements;]
[ELSE
  statements;]
END IF;
```

Writing Executable Statements

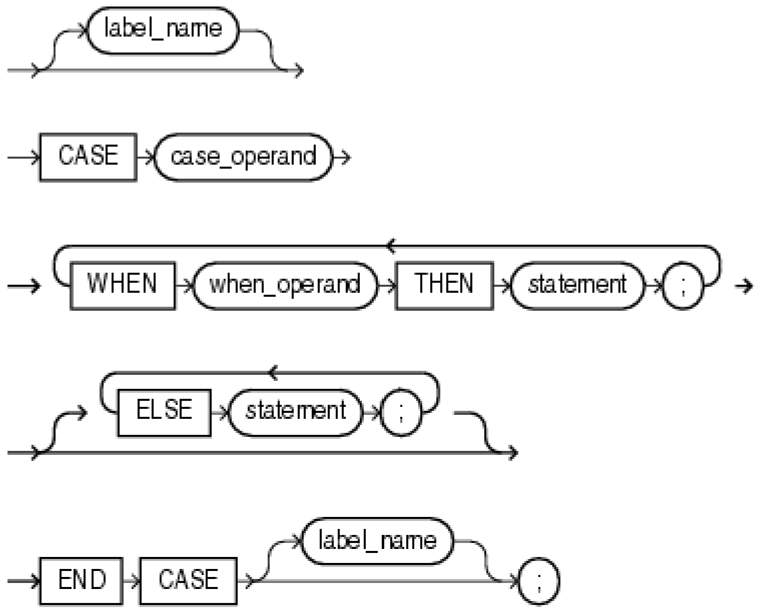
CASE Expressions

– A CASE expression selects a result and returns it.
– To select the result, the CASE expression uses expressions. The value returned by these expressions is used to select one of several alternatives.

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

CASE Statement

- The CASE statement is more readable compared to multiple IF and ELSIF statements.
- How a CASE Expression Differs from a CASE Statement
  - A CASE expression evaluates the condition and returns a value, whereas a CASE statement evaluates the condition and performs an action. A CASE statement can be a complete PL/SQL block.
  - CASE statements end with END CASE;
  - CASE expressions end with END;



Handling Nulls

- When working with nulls, you can avoid some common mistakes by keeping in mind the following rules:
  - Simple comparisons involving nulls always yield `NULL`.
  - Applying the logical operator `NOT` to a null yields `NULL`.
  - If the condition yields `NULL` in conditional control statements, its associated sequence of statements is not executed.

Logic Tables

- FALSE takes precedence in an AND condition.
- TRUE takes precedence in an OR condition
- AND returns TRUE only if both of its operands are TRUE
- OR returns FALSE only if both of its operands are FALSE
- NULL AND TRUE always evaluates to NULL because it is not known whether the second operand evaluates to TRUE
- The negation of NULL (NOT NULL) results in a null value because null values are indeterminate.

AND	*TRUE*	*FALSE*	*NULL*
TRUE	TRUE	FALSE	NULL
FALSE	FALSE	FALSE	FALSE
NULL	NULL	FALSE	NULL

OR	*TRUE*	*FALSE*	*NULL*
TRUE	TRUE	TRUE	TRUE
FALSE	TRUE	FALSE	NULL
NULL	TRUE	NULL	NULL

NOT	
TRUE	FALSE
FALSE	TRUE
NULL	NULL

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
- An EXIT statement can be used to terminate loops. A basic loop must have an EXIT.

- Basic loop syntax:

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

WHILE Loops

WHILE Loop syntax:

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

FOR Loops

- Use a `FOR` loop to shortcut the test for the number of iterations.
- Do not declare the counter; it is declared implicitly.
- Reference the counter within the loop only; it is undefined outside the loop.
- Do not reference the counter as the target of an assignment.
- '`lower_bound .. upper_bound`' is required syntax.
- Neither loop bound should be NULL.

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

PL/SQL CONTINUE Statement

Definition

- Adds the functionality to begin the next loop iteration
- Provides programmers with the ability to transfer control to the next iteration of a loop
- Uses parallel structure and semantics to the EXIT statement

Benefits

- Eases the programming process
- May see a small performance improvement over the previous programming workarounds to simulate the CONTINUE statement
- Commonly used to filter data inside a loop body before the main processing begins.