Oracle Database 11g

PL SQL – Part I

Overview

What is PL/SQL?

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

Benefits of PL/SQL:

- Modularize program development
- A procedural language with control structures
- Handle errors

PL/SQL Block Structure

DECLARE (Optional)

Variables, constants, cursors, user defined exceptions

BEGIN (Mandatory)

SQL statements

PL/SQL control statements

EXCEPTION (Optional)

Actions to perform when errors occur

END; (Mandatory)

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;
```

Developing a Simple PL/SQL Block

Handling Variables in PL/SQL

- **Declare and initialize variables within the declaration section.**
- **Assign new values to variables within the executable section.**

Declaring Variables and Constants: Syntax

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

Guidelines

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

Declaring Scalar Variables

- **Have no internal components.**
- **Hold a single value.**
- **Base Types:**
  - **BINARY_INTEGER**
  - **NUMBER [(*precision, scale*)]**
  - **CHAR [(*maximum_length*)]**
  - **VARCHAR2(*maximum_length*)**
  - **DATE**
  - **BOOLEAN**

Scalar Variable Declarations: Examples

```
v_gender      CHAR(1);
v_count       BINARY_INTEGER := 0;
v_total_sal   NUMBER(9) := 0;
v_order_date  DATE := SYSDATE + 7;
c_tax_rate    CONSTANT NUMBER(3,2) := 8.25;
v_valid       BOOLEAN NOT NULL := TRUE;
```

Operators in PL/SQL: Examples

- **Set the value of a Boolean flag.**

```
v_equal        := (v_n1 = v_n2);
```

- **Validate an employee number if it contains a value.**

```
v_valid        := (v_emp_id IS NOT NULL);
```

Nested Blocks and Variable Scope

- Statements can be nested wherever an executable statement is allowed.
- Nested block becomes a statement.
- Exception section can contain nested blocks.
- Scope of an object is the region of the program that can refer to the object.
- Identifier is visible in the regions in which you can reference the unqualified identifier.
  - A block can look up to the enclosing block.
  - A block cannot look down to enclosed blocks.

Nested Blocks and Variable Scope: Example

```
DECLARE
    x  BINARY_INTEGER;
BEGIN
   ...
   DECLARE
       y  NUMBER;
   BEGIN
    ...
   END;
 ...
END;
```

Scope of x

Scope of y

Commenting Code

Comment code by

- Prefixing the comment with two dashes (- -).
- Placing the comment between /* and */.

Example

```
...
  v_sal NUMBER (9,2);
BEGIN
  /* Compute the annual salary based on the
monthly salary input from the user */
  v_sal := v_sal * 12;
END;
```

DataType Conversion

Datatype Conversion

- Convert data to comparable datatypes.
- Mixed datatypes can result in an error and affect performance.
- Conversion functions:
  - TO_CHAR
  - TO_DATE
  - TO_NUMBER

Datatype Conversion: Example

- This statement produces a compile error.

```
v1 := USER||SYSDATE;
```

- To correct the error, the TO_CHAR conversion function is used.

```
v1 := USER||TO_CHAR(SYSDATE);
```

%TYPE Attribute

The %TYPE Attribute

- Declare a variable according to
  - Another previously declared variable.
  - A database column definition.
- Prefix %TYPE with
  - The database table and column.
  - The previously declared variable name.
- PL/SQL determines the datatype and size of the variable.

The %TYPE Attribute: Examples

```
...
  v_last_name          emp.last_name%TYPE;
  v_first_name         emp.first_name%TYPE;
  v_balance            NUMBER(7);
  v2_balance           v_balance%TYPE := 10;
...
```

Advantages of using the %TYPE attribute

- **The datatype of the underlying database column may be unknown.**
- **The datatype of the underlying database column may change at runtime.**

%ROWTYPE Attribute

The %ROWTYPE Attribute

- Declare a variable according to a collection of columns in a database table or view.
- Prefix %ROWTYPE with the database table or view.
- Fields in the record take their names and data types from the columns of the table or view.

- Syntax:

```
DECLARE
identifier        reference%ROWTYPE;
```

```
DECLARE
Dept_record       dept%ROWTYPE;
```

Advantages of Using `%ROWTYPE`

- The number and data types of the underlying database columns may be unknown.
- The number and data types of the underlying database column may change at run time.
- The attribute is useful when retrieving a row with the `SELECT *` statement.

Interacting with the Database

SQL Commands in PL/SQL

- Extract a row of data from the database by using the SELECT command.
- Make changes to rows in the database by using DML commands.
- Control a transaction with COMMIT or ROLLBACK commands.

Retrieving Data: Syntax

Retrieve data from the database with SELECT.

```
SELECT    select_list
INTO      variable_name |  record_name
FROM      table
WHERE     condition;
```

- **INTO clause is required.**
- **Exactly one row must be returned.**
- **Full SELECT syntax is available.**

Retrieving Data: Example

Retrieve the order date and the ship date for the specified order.

```
DECLARE
  v1 orders.date_ordered%TYPE;
  v2 orders.date_shipped%TYPE;
BEGIN
    SELECT  date_ordered, date_shipped
    INTO    v1, v2
    FROM    orders
    WHERE order_id = 102;
END;
```

Retrieving Data: Example

Return the sum of the salaries for all employees in the specified department.

```
DECLARE
  v_sum_salary  emp.salary%TYPE;
BEGIN
    SELECT  SUM(salary)      --group function
    INTO    v_sum_salary
    FROM    emp
    WHERE   dept_id = 30;

END;
```

Retrieving Data: Example

Retrieve all information about the specified department

```
DECLARE
  dept_record   dept%ROWTYPE;
BEGIN
    SELECT   *
    INTO  dept_record   --PL/SQL RECORD
    FROM  dept
    WHERE dept_id = 30;
  ...
END;
```

SELECT Exceptions

- SELECT statements in PL/SQL must retrieve exactly one row.
- If zero or more than one row is retrieved, an exception is raised.
- SELECT exceptions:
  - TOO_MANY_ROWS
  - NO_DATA_FOUND

TOO_MANY_ROWS Exception: Example

Retrieve the order for customer number 208.

```
BEGIN
    SELECT order_id
    INTO  v1
    FROM  orders
    WHERE customer_id = 208;
end;
```

TOO_MANY_ROWS

```
Customer 208 has several orders.
```

NO_DATA_FOUND Exception: Example

Retrieve the order for customer number 999.

```
BEGIN
    SELECT order_id
    INTO  v1
    FROM  orders
    WHERE customer_id = 999;
end;
```

NO_DATA_FOUND