PL/SQL Variables and constants , Literals

Saturday, March 19, 2022 6:18 PM

- Block structured Language
- Conditional statements, loops, arrays, string, exceptions, collections, records, triggers, functions, procedures, cursors etc.
- Pl/SQL stands for "Procedural Language extension of SQL" that is used in Oracle. PL/SQL is integrated with Oracle database (since version 7).
- PL/SQL includes procedural language elements like conditions and loops. It allows declaration of constants and variables, procedures and functions, types and variable of those types and triggers. It can support Array and handle exceptions (runtime errors).
- The PL/SQL is known for its combination of data manipulating power of SQL with data processing power of procedural languages
- The PL/SQL is known for its combination of data manipulating power of SQL with data processing power of procedural languages

```
Radius Number := 5;

Date_of_birth date;
```

Declaration Restrictions:

In PL/SQL while declaring the variable some restrictions hold.

- Forward references are not allowed i.e. you must declare a constant or variable before referencing it in another statement even if it is a declarative statement.
  val number := Total - 200;
  Total number := 1000;
  The first declaration is illegal because the TOTAL variable must be declared before using it in an assignment expression.
- Variables belonging to the same datatype cannot be declared in the same statement.
  N1, N2, N3 Number;
  It is an illegal declaration.

The name of the variable must begin with ASCII letter. The PL/SQL is not case sensitive so it could be either lowercase or uppercase. For example: v_data and V_DATA refer to the same variables.

You should make your variable easy to read and understand, after the first character, it may be any number, underscore (_) or dollar sign ($).

NOT NULL is an optional specification on the variable.

Every time you declare a variable, PL/SQL defines a default value NULL to it. If you want to initialize a variable with other value than NULL value, you can do so during the declaration, by using any one of the following methods

- The DEFAULT keyword
- The assignment operator

```
counter binary_integer := 0;
greetings varchar2(20) DEFAULT 'Hello JavaTpoint';
```

Example:

```
DECLARE
  a integer := 30;
  b integer := 40;
  c integer;
  f real;
BEGIN
  c := a + b;
  dbms_output.put_line('Value of c: ' || c);
  f := 100.0/3.0;
  dbms_output.put_line('Value of f: ' || f);
END;
```

After the execution, this will produce the following result:

```
Value of c: 70
Value of f: 33.333333333333333333

PL/SQL procedure successfully completed.
```

```
DECLARE
 -- Global variables
  num1 number := 95;
  num2 number := 85;
BEGIN
  dbms_output.put_line('Outer Variable num1: ' || num1);
  dbms_output.put_line('Outer Variable num2: ' || num2);
  DECLARE
    -- Local variables
    num1 number := 195;
    num2 number := 185;
  BEGIN
    dbms_output.put_line('Inner Variable num1: ' || num1);
    dbms_output.put_line('Inner Variable num2: ' || num2);
  END;
END;
/
```

After the execution, this will produce the following result:

```
Outer Variable num1: 95
Outer Variable num2: 85
Inner Variable num1: 195
Inner Variable num2: 185

PL/SQL procedure successfully completed.
```

Variable Attributes:

- When you declare a PL/SQL variable to hold the column values, it must be of correct data types and precision, otherwise error will occur on execution. Rather than hard coding the data type and precision of a variable. PL/SQL provides the facility to declare a variable without having to specify a particular data type using %TYPE and %ROWTYPE attributes
- The %TYPE attribute is used to declare variables according to the already declared variable or database column. It is used when you are declaring an individual variable, not a record. The data type and precision of the variable declared using %TYPE attribute is the same as

that of the column that is referred from a given table

```
DECLARE
SALARY EMP.SAL % TYPE;
ECODE EMP.empno % TYPE;
BEGIN
Ecode :=&Ecode;
Select SAL into SALARY from EMP where EMPNO = ECODE;
dbms_output.put_line('Salary of ' || ECODE || 'is = || salary');
END;
```

After the execution, this will produce the following result:

```
Enter value for ecode: 7499
Salary of 7499 is = 1600
PL/SQL procedure successfully completed.
```

```
DECLARE
EMPLOYEE EMP. % ROW TYPE;
BEGIN
EMPLOYEE.EMPNO := 2092;
5   EMPLOYEE.ENAME := 'Sanju';
Insert into EMP where (EMPNO, ENAME) Values (employee.empno, employee.ename);
dbms_output.put_line('Row Inserted');
END;
```

After the execution, this will produce the following result:

```
Row Inserted
PL/SQL procedure successfully completed.
```

Advantages:

If you don't know the data type at the time of declaration. The data type assigned to the associated variables will be determined dynamically at run time.

If the data type of the variable you are referencing changes the %TYPE or %ROWTYPE variable changes at run time without having to rewrite variable declarations. For example: if the ENAME column of an EMP table is changed from a VARCHAR2(10) to VRACHAR2(15) then you don't need to modify the PL/SQL code.

PL/SQL Constants

```
DECLARE
   -- constant declaration
   pi constant number := 3.141592654;
   -- other declarations
   radius number(5,2);
   dia number(5,2);
   circumference number(7, 2);
   area number (10, 2);
BEGIN
   -- processing
   radius := 9.5;
   dia := radius * 2;
   circumference := 2.0 * pi * radius;
   area := pi * radius * radius;
   -- output
   dbms_output.put_line('Radius: ' || radius);
   dbms_output.put_line('Diameter: ' || dia);
   dbms_output.put_line('Circumference: ' || circumference);
   dbms_output.put_line('Area: ' || area);
END;
/
```

After the execution of the above code at SQL prompt, it will produce the following result:.

```
Radius: 9.5
Diameter: 19
Circumference: 59.69
Area: 283.53

Pl/SQL procedure successfully completed.
```

Example of these different types of Literals:

Literals	Examples
Numeric	75125, 3568, 33.3333333 etc.
Character	'A' '%' '9' ' ' 'z' '('
String	Hello JavaTpoint!
Boolean	TRUE, FALSE, NULL etc.
Date and Time	'26-11-2002' , '2012-10-29 12:01:01'

PL/SQL Control Statements

Saturday, March 19, 2022 7:27 PM

- PL/SQL supports the programming language features like conditional statements and iterative statements. Its programming constructs are similar to how you use in programming languages like Java and C++

Syntax: (IF-THEN-ELSIF-ELSE statement):

```
IF condition1
THEN
  {...statements to execute when condition1 is TRUE...}
ELSIF condition2
THEN
  {...statements to execute when condition2 is TRUE...}
ELSE
  {...statements to execute when both condition1 and condition2 are FALSE...}
END IF;
```

```
DECLARE
  a number(3) := 500;
BEGIN
  -- check the boolean condition using if statement
  IF( a < 20 ) THEN
    -- if condition is true then print the following
    dbms_output.put_line('a is less than 20 ' );
  ELSE
    dbms_output.put_line('a is not less than 20 ' );
  END IF;
  dbms_output.put_line('value of a is : ' || a);
END;
```

After the execution of the above code in SQL prompt, you will get the following result:

```
a is not less than 20
value of a is : 500
PL/SQL procedure successfully completed.
```

PL/SQL Case Statement

- The PL/SQL CASE statement facilitates you to execute a sequence of statements based on a selector. A selector can be anything such as variable, function or an expression that the CASE statement checks to a Boolean value.

```
DECLARE
  grade char(1) := 'A';
BEGIN
  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('Average');
    when 'F' then dbms_output.put_line('Passed with Grace');
    else dbms_output.put_line('Failed');
  END CASE;
END;
```

After the execution of above code, you will get the following result:

```
Excellent
PL/SQL procedure successfully completed.
```

PL/SQL Exit Loop (Basic Loop)

- PL/SQL exit loop is used when a set of statements is to be executed at least once before the termination of the loop. There must be an EXIT condition specified in the loop, otherwise the loop will get into an infinite number of iterations. After the occurrence of EXIT condition, the process exits the loop.

Syntax of basic loop:

```
LOOP
  Sequence of statements;
END LOOP;
```

Syntax of exit loop:

```
LOOP
  statements;
  EXIT;
  {or EXIT WHEN condition;}
END LOOP;
```

```
DECLARE
i NUMBER := 1;
BEGIN
LOOP
EXIT WHEN i>10;
DBMS_OUTPUT.PUT_LINE(i);
i := i+1;
END LOOP;
END;
```

After the execution of the above code, you will get the following result:

```
1
2
3
4
5
6
7
8
9
10
```

PL/SQL EXIT Loop Example 2

```
DECLARE
VAR1 NUMBER;
VAR2 NUMBER;
BEGIN
VAR1:=100;
VAR2:=1;
LOOP
DBMS_OUTPUT.PUT_LINE (VAR1*VAR2);
IF (VAR2=10) THEN
EXIT;
END IF;
VAR2:=VAR2+1;
END LOOP;
END;
```

Output:

```
100
200
300
400
500
600
700
800
900
1000
```

PL/SQL While Loop

```
DECLARE
i INTEGER := 1;
BEGIN
WHILE i <= 10 LOOP
DBMS_OUTPUT.PUT_LINE(i);
i := i+1;
END LOOP;
END;
```

```
1
2
3
4
5
6
7
8
9
10
```

PL/SQL FOR Loop

```
DECLARE
VAR1 NUMBER;
BEGIN
VAR1:=10;
FOR VAR2 IN 1..10
LOOP
DBMS_OUTPUT.PUT_LINE (VAR1*VAR2);
END LOOP;
END;
```

Output:

```
10
20
30
40
50
60
70
80
90
100
```

PL/SQL For Loop REVERSE

```
DECLARE
VAR1 NUMBER;
BEGIN
VAR1:=10;
FOR VAR2 IN REVERSE 1..10
LOOP
DBMS_OUTPUT.PUT_LINE (VAR1*VAR2);
END LOOP;
END;
```

Output:

```
100
90
80
70
60
50
40
30
20
10
```

PL/SQL Continue Statement

- For example: If a continue statement exits a cursor FOR LOOP prematurely then it exits an inner loop and transfer control to the next iteration of an outer loop, the cursor closes (in this context, CONTINUE works like GOTO).

```
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
```

PL/SQL GOTO Statement

```
DECLARE
  a number(2) := 30;
BEGIN
  <<loopstart>>
  -- while loop execution
  WHILE a < 50 LOOP
    dbms_output.put_line ('value of a: ' || a);
    a := a + 1;
    IF a = 35 THEN
      a := a + 1;
      GOTO loopstart;
    END IF;
  END LOOP;
END;
```

```
value of a: 30
value of a: 31
value of a: 32
value of a: 33
value of a: 34
value of a: 36
value of a: 37
value of a: 38
value of a: 39
value of a: 40
value of a: 41
value of a: 42
value of a: 43
value of a: 44
value of a: 45
value of a: 46
value of a: 47
value of a: 48
value of a: 49

Statement processed.
```

Restriction on GOTO statement

Following is a list of some restrictions imposed on GOTO statement.

- Cannot transfer control into an IF statement, CASE statement, LOOP statement or sub-block.
- Cannot transfer control from one IF statement clause to another or from one CASE statement WHEN clause to another.
- Cannot transfer control from an outer block into a sub-block.
- Cannot transfer control out of a subprogram.
- Cannot transfer control into an exception handler.

- Oracle provides some attributes known as Implicit cursor's attributes to check the status of DML operations. Some of them are: %FOUND, %NOTFOUND, %ROWCOUNT and %ISOPEN.

Attribute	Description
%FOUND	Its return value is TRUE if DML statements like INSERT, DELETE and UPDATE affect at least one row or more rows or a SELECT INTO statement returned one or more rows. Otherwise it returns FALSE.
%NOTFOUND	Its return value is TRUE if DML statements like INSERT, DELETE and UPDATE affect no row, or a SELECT INTO statement return no rows. Otherwise it returns FALSE. It is a just opposite of %FOUND.
%ISOPEN	It always returns FALSE for implicit cursors, because the SQL cursor is automatically closed after executing its associated SQL statements.
%ROWCOUNT	It returns the number of rows affected by DML statements like INSERT, DELETE, and UPDATE or returned by a SELECT INTO statement.

PL/SQL Implicit Cursor Example

Create customers table and have records:

ID	NAME	AGE	ADDRESS	SALARY
1	Ramesh	23	Allahabad	20000
2	Suresh	22	Kanpur	22000
3	Mahesh	24	Ghaziabad	24000
4	Chandan	25	Noida	26000
5	Alex	21	Paris	28000
6	Sunita	20	Delhi	30000

Create procedure:

```
DECLARE
  total_rows number(2);
BEGIN
  UPDATE  customers
  SET salary = salary + 5000;
  IF sql%notfound THEN
    dbms_output.put_line('no customers updated');
  ELSIF sql%found THEN
    total_rows := sql%rowcount;
    dbms_output.put_line( total_rows || ' customers updated ');
  END IF;
END;
/
```

Output:

```
6 customers updated
PL/SQL procedure successfully completed.
```

2) PL/SQL Explicit Cursors

The Explicit cursors are defined by the programmers to gain more control over the context area. These cursors should be defined in the declaration section of the PL/SQL block. It is created on a SELECT statement which returns more than one row.

Steps:

You must follow these steps while working with an explicit cursor.

1. Declare the cursor to initialize in the memory.
2. Open the cursor to allocate memory.
3. Fetch the cursor to retrieve data.
4. Close the cursor to release allocated memory.

PL/SQL Explicit Cursor Example

Explicit cursors are defined by programmers to gain more control over the context area. It is defined in the declaration section of the PL/SQL block. It is created on a SELECT statement which returns more than one row.

Let's take an example to demonstrate the use of explicit cursor. In this example, we are using the already created CUSTOMERS table.

Create customers table and have records:

ID	NAME	AGE	ADDRESS	SALARY
1	Ramesh	23	Allahabad	20000
2	Suresh	22	Kanpur	22000
3	Mahesh	24	Ghaziabad	24000
4	Chandan	25	Noida	26000
5	Alex	21	Paris	28000
6	Sunita	20	Delhi	30000

```
DECLARE
  c_id customers.id%type;
  c_name customers.name%type;
  c_addr customers.address%type;
  CURSOR c_customers is
    SELECT id, name, address FROM customers;
BEGIN
  OPEN c_customers;
  LOOP
    FETCH c_customers into c_id, c_name, c_addr;
    EXIT WHEN c_customers%notfound;
    dbms_output.put_line(c_id || ' ' || c_name || ' ' || c_addr);
  END LOOP;
  CLOSE c_customers;
END;
/
```

```
1  Ramesh  Allahabad
2  Suresh  Kanpur
3  Mahesh  Ghaziabad
4  Chandan  Noida
5  Alex  Paris
6  Sunita  Delhi
PL/SQL procedure successfully completed.
```

PL/SQL Exception Handling

20 March 2022 14:10

- An error occurs during the program execution is called Exception in PL/SQL
- There are two type of exceptions:
  - System-defined Exceptions
  - User-defined Exceptions
- Basic Syntax

```
DECLARE
  <declarations section>
BEGIN
  <executable command(s)>
EXCEPTION
  <exception handling goes here >
  WHEN exception1 THEN
    exception1-handling-statements
  WHEN exception2  THEN
    exception2-handling-statements
  WHEN exception3 THEN
    exception3-handling-statements
  ........
  WHEN others THEN
    exception3-handling-statements
END;
```

Example of Exception Handling

ID	NAME	AGE	ADDRESS	SALARY
1	Ramesh	23	Allahabad	20000
2	Suresh	22	Kanpur	22000
3	Mahesh	24	Ghaziabad	24000
4	Chandan	25	Noida	26000
5	Alex	21	Paris	28000
6	Sunita	20	Delhi	30000

```
DECLARE
  c_id customers.id%type := 8;
  c_name  customers.name%type;
  c_addr customers.address%type;
BEGIN
  SELECT  name, address INTO  c_name, c_addr
  FROM customers
  WHERE id = c_id;
DBMS_OUTPUT.PUT_LINE ('Name: '||  c_name);
 DBMS_OUTPUT.PUT_LINE ('Address: ' || c_addr);
EXCEPTION
  WHEN no_data_found THEN
    dbms_output.put_line('No such customer!');
  WHEN others THEN
    dbms_output.put_line('Error!');
END;
```

```
No such customer!
PL/SQL procedure successfully completed.
```

```
DECLARE
  c_id customers.id%type := 5;
  c_name  customers.name%type;
  c_addr customers.address%type;
BEGIN
  SELECT  name, address INTO  c_name, c_addr
  FROM customers
  WHERE id = c_id;
DBMS_OUTPUT.PUT_LINE ('Name: '||  c_name);
 DBMS_OUTPUT.PUT_LINE ('Address: ' || c_addr);
EXCEPTION
  WHEN no_data_found THEN
    dbms_output.put_line('No such customer!');
  WHEN others THEN
    dbms_output.put_line('Error!');
END;
```

```
Name: alex
Address: paris
PL/SQL procedure successfully completed.
```

- We can raise the exception explicitly other than the database server.
- We can declare our own user define exceptions

User defined Exceptions

- PL/SQL facilitates their users to define their own exceptions according to the need of the

program. A user-defined exception can be raised explicitly, using either a RAISE statement or the procedure DBMS_STANDARD.RAISE_APPLICATION_ERROR

Syntax for user define exceptions

```
DECLARE
my-exception EXCEPTION;
```

Pre defined exceptions

- We already have some pre defined exceptions which are already declared and called when the database rule have been violated by the program.
- We can call them in exception handling or else the compiler calls them automatically when the error have been triggered.

Exception	Oracle Error	SQL Code	Description
ACCESS_INTO_NULL	06530	-6530	It is raised when a NULL object is automatically assigned a value.
CASE_NOT_FOUND	06592	-6592	It is raised when none of the choices in the "WHEN" clauses of a CASE statement is selected, and there is no else clause.
COLLECTION_IS_NULL	06531	-6531	It is raised when a program attempts to apply collection methods other than exists to an uninitialized nested table or varray, or the program attempts to assign values to the elements of an uninitialized nested table or varray.
DUP_VAL_ON_INDEX	00001	-1	It is raised when duplicate values are attempted to be stored in a column with unique index.
INVALID_CURSOR	01001	-1001	It is raised when attempts are made to make a cursor operation that is not allowed, such as closing an unopened cursor.
INVALID_NUMBER	01722	-1722	It is raised when the conversion of a character string into a number fails because the string does not represent a valid number.
LOGIN_DENIED	01017	-1017	It is raised when s program attempts to log on to the database with an invalid username or password.
NO_DATA_FOUND	01403	+100	It is raised when a select into statement returns no rows.
NOT_LOGGED_ON	01012	-1012	It is raised when a database call is issued without being connected to the database.
PROGRAM_ERROR	06501	-6501	It is raised when PL/SQL has an internal problem.
ROWTYPE_MISMATCH	06504	-6504	It is raised when a cursor fetches value in a variable having incompatible data type.
SELF_IS_NULL	30625	-30625	It is raised when a member method is invoked, but the instance of the object type was not initialized.
STORAGE_ERROR	06500	-6500	It is raised when PL/SQL ran out of memory or memory was corrupted.
TOO_MANY_ROWS	01422	-1422	It is raised when a SELECT INTO statement returns more than one row.
VALUE_ERROR	06502	-6502	It is raised when an arithmetic, conversion, truncation, or size-constraint error occurs.
ZERO_DIVIDE	01476	1476	It is raised when an attempt is made to divide a number by zero.