PL/SQL

Krishna Modi

Introduction

- PL/SQL is a combination of SQL along with the procedural features of programming languages. It was developed by Oracle Corporation in the early 90's to enhance the capabilities of SQL.

Features of PL/SQL

- PL/SQL is tightly integrated with SQL.
- It offers extensive error checking.
- It offers numerous data types.
- It offers a variety of programming structures.
- It supports structured programming through functions and procedures.
- It supports object–oriented programming.
- It supports the development of web applications and server pages.

Advantages of PL/SQL

- Applications written in PL/SQL are fully portable.
- PL/SQL provides high security level.
- PL/SQL provides access to predefined SQL packages.
- PL/SQL provides support for Object–Oriented Programming.
- PL/SQL provides support for developing Web Applications and Server Pages.
- PL/SQL allows sending an entire block of statements to the database at one time. This reduces network traffic and provides high performance for the applications.

Generic pl/sql block

```
DECLARE
   <declarations section>
BEGIN
   <executable command(s)>
EXCEPTION
   <exception handling>
END;
```

S.No	Sections & Description
1	**Declarations** This section starts with the keyword **DECLARE**. It is an optional section and defines all variables, cursors, subprograms, and other elements to be used in the program.
2	**Executable Commands** This section is enclosed between the keywords **BEGIN** and **END** and it is a mandatory section. It consists of the executable PL/SQL statements of the program. It should have at least one executable line of code, which may be just a **NULL command** to indicate that nothing should be executed.
3	**Exception Handling** This section starts with the keyword **EXCEPTION**. This optional section contains **exception(s)** that handle errors in the program.

Delimiter	Description
+, –, *, /	Addition, subtraction/negation, multiplication, division
%	Attribute indicator
’	Character string delimiter
.	Component selector
(,)	Expression or list delimiter
:	Host variable indicator
,	Item separator
‘	Quoted identifier delimiter
=	Relational operator
@	Remote access indicator
;	Statement terminator
:=	Assignment operator
=I	Association operator
||	Concatenation operator
**	Exponentiation operator
<<, II	Label delimiter (begin and end)
/*, */	Multi–line comment delimiter (begin and end)
--	Single–line comment indicator

Control structure

IF *condition* **THEN**

sequence_of_statements
END IF;

Example:

IF x ı y THEN high := x;

END IF;

IF *condition* **THEN**

sequence_of_statements1

ELSE

sequence_of_statements2

END IF;

Ex:

```
IF trans_type = 'CR' THEN
   UPDATE accounts SET balance = balance +
credit WHERE …
ELSE
   UPDATE accounts SET balance = balance –
debit WHERE …
END IF;
```

If–then–else

IF *condition 1* **THEN**

sequence_of_statements 1

ELSIF *condition 2* **THEN**

sequence_of_statements 2

ELSE

```
IF grade = 'A' THEN
   dbms_output.put_line('Excellent');
ELSIF grade = 'B' THEN
   dbms_output.put_line('Very Good');
ELSIF grade = 'C' THEN
   dbms_output.put_line('Good');
ELSIF grade = 'D' THEN
   dbms_output. put_line('Fair');
ELSIF grade = 'F' THEN
   dbms_output.put_line('Poor');
ELSE   dbms_output.put_line('No such grade');
END IF;
```

Example

Write a pl/sql code block that accept a client_no from the user, check if the user bal_due is less than minimum balance – 5000, only then deduct Rs. 100/– from the balance. The process is fired on client_master table.

Solution

```
declare
   min_bal constant number(8,2):=5000;
   bal_due1 client_master.bal_due %type;
   client_no1 client_master.client_no %type := '&client_no';
begin
  select bal_due into bal_due1 from client_master where client_no=client_no1;
dbms_output.put_line('bal_due is....' || bal_due1);
if(bal_due1<min_bal)
Then
      dbms_output.put_line('bal_due is less than  minimum balance');
   update client_master set bal_due=bal_due- 100 where client_no=client_no1;
   end if;
exception
   when no_data_found then
      dbms_output.put_line('Record not found');
End;
```

CASE statement

```
CASE selector
   WHEN 'value1' THEN S1;
   WHEN 'value2' THEN S2;
   WHEN 'value3' THEN S3;
   ...
   ELSE Sn;  -- default case
END CASE;
```

```
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('Well done');
      when 'D' then dbms_output.put_line('You passed');
      when 'F' then dbms_output.put_line('Better try again');
      else dbms_output.put_line('No such grade');
   END CASE;
END;
```

Iterative control (loops)

Simple loop:

```
LOOP
   sequence_of_statements
END LOOP;
```

With each iteration of the loop, the sequence of statements is executed, then control resumes at the top of the loop.

If further processing is undesirable or impossible, you can use an EXIT statement to complete the loop.

There are two forms of EXIT statements: EXIT and EXIT-WHEN.

Exit statement

The EXIT statement forces a loop to complete unconditionally.

When an EXIT statement is encountered, the loop completes immediately and control passes to the next statement.

```
LOOP
   ...
   IF credit_rating < 3 THEN
      ...
      EXIT;  -- exit loop
immediately
   END IF;
END LOOP;
-- control resumes here
```

Exit statement

EXIT statement must be placed inside a loop.

```
BEGIN
   ...
   IF credit_rating < 3 THEN
      ...
      EXIT;  -- not allowed
   END IF;
END;
```

Exit when condition;

If condition then exit;
End if;

Both are same

Exit–when

The EXIT–WHEN statement lets a loop complete conditionally.

When the EXIT statement is encountered, the condition in the WHEN clause is evaluated.

If the condition is true, the loop completes and control passes to the next statement after the loop

```
EXIT WHEN count > 100;
```

is similar as

```
IF count > 100 THEN
count > 100;
   EXIT;
END IF;
```

Example

- Print 1 to 100 using loop.
- To calculate the areas of circles of radius till 100 using PL/SQL.

create table mycircle (area number(10), radius number(2));

```
Declare
Area number(10);
Radius number(2) := 1;
Pi constant number(3,2):=3.14;
Begin
Loop
Area:=pi*power(radius,2);
Insert into mycircle values(area,radius);
Radius:=radius+1;
Exit when radius>10;
end loop;
end;
```

While loop

While condition

Loop

Sequence_of_statements;

End loop;

```
WHILE total <= 25000 LOOP
   ...
   SELECT sal INTO salary
FROM emp WHERE ...
   total := total + salary;
END LOOP;
```

PL/SQL Data types

Krishna Modi

PL/SQL Variables and Data Types

- Variable names must follow the Oracle naming standard
- Strongly typed language:
  - Explicitly declare each variable including data type before using variable
- Variable declaration syntax:
  - *variable_name data_type_declaration;*
- Default value always **NULL**
- PL/SQL supports: scalar, composite, reference and LOB.

PL/SQL datatypes

- Scalar
- Composite
- reference
- LOB.

Scalar Variables

- Reference single value
- Data types correspond to Oracle $10g$ database data types
  - VARCHAR2
  - CHAR
  - DATE
  - NUMBER

General Scalar Data Types

Data Type	Description	Sample Declaration
Integer number subtypes (BINARY_INTEGER, INTEGER, INT, SMALLINT)	Integer	`counter BINARY_INTEGER;`
Decimal number subtypes (DEC, DECIMAL, DOUBLE PRECISION, NUMERIC, REAL)	Numeric value with varying precision and scale	`student_gpa REAL;`
BOOLEAN	True/False value	`order_flag BOOLEAN;`

Composite Variables

- Data Structure is a data object made up of multiple individual data elements
- Composite variable is a data structure contains multiple scalar variables
- Types:
  - RECORD
  - TABLE
  - VARRAY

Reference Variables

- Reference variables directly reference specific database column or row
- Reference variables assume data type of associated column or row
- %TYPE data declaration syntax:
  - *variable_name tablename.fieldname*%TYPE;
- %ROWTYPE data declaration syntax:
  - *variable_name tablename*%ROWTYPE;

Reference Variables

- The (%TYPE) reference data type specifies a variable that references a single DB field.
  - current_f_last FACULTY.F_LAST%TYPE;
- The current_f_last variable assumes a data type of VARCHAR2(30), because this is the data type of the f_last column in the FACULTY table.

Reference Variables

- The (%ROWTYPE) reference data type creates composite variables that reference the entire data record.
  - *Faculty_row FACULTY*%ROWTYPE;
- The variable faculty_row references all of the columns in the FACULTY table, and each column has the same data type as its associated DB column.

LOB Data Types

- It declares variables that reference binary or character data objects up to 4 GB.
- LOB values in PL/SQL programs must be manipulated using special package called DBMS_LOB.

User defined subtypes

Syntax:

Subtype ***newtype*** is ***originaltype***

Ex:

Subtype loopcounter is number;

Cursor

Krishna Modi

Cursor

- Oracle creates a memory area, known as the **context area**, for processing an SQL statement, which contains all the information needed for processing the statement; for example, the number of rows processed, etc.

- A **cursor** is a pointer to this context area.

Types of cursor

- Implicit cursor
- Explicit Cursor

Implicit cursor

- Implicit cursors are automatically created by Oracle whenever an SQL statement is executed.
- Whenever a DML statement (INSERT, UPDATE and DELETE) is issued, an implicit cursor is associated with this statement. For INSERT operations, the cursor holds the data that needs to be inserted. For UPDATE and DELETE operations, the cursor identifies the rows that would be affected.
- In PL/SQL, you can refer to the most recent implicit cursor as the **SQL cursor**, which always has attributes such as **%FOUND**, **%ISOPEN**, **%NOTFOUND**, and **%ROWCOUNT**.

S.No	Attribute & Description
1	**%FOUND** Returns TRUE if an INSERT, UPDATE, or DELETE statement affected one or more rows or a SELECT INTO statement returned one or more rows. Otherwise, it returns FALSE.
2	**%NOTFOUND** The logical opposite of %FOUND. It returns TRUE if an INSERT, UPDATE, or DELETE statement affected no rows, or a SELECT INTO statement returned no rows. Otherwise, it returns FALSE.
3	**%ISOPEN** Always returns FALSE for implicit cursors, because Oracle closes the SQL cursor automatically after executing its associated SQL statement.
4	**%ROWCOUNT** Returns the number of rows affected by an INSERT, UPDATE, or DELETE statement, or returned by a SELECT INTO statement.

Example : Implicit cursor and attributes

```
DECLARE
   total_rows number(2);
BEGIN
   UPDATE customers
   SET salary = salary + 500;
   IF sql%notfound THEN
      dbms_output.put_line('no customers selected');
   ELSIF sql%found THEN
      total_rows := sql%rowcount;
      dbms_output.put_line( total_rows || ' customers selected ');
   END IF;
END;
/
```

Explicit cursor

- Explicit cursors are programmer–defined cursors for gaining more control over the **context area**.
- An explicit cursor 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.

Processing Explicit cursor

1. Declare the cursor
2. Open the cursor for the query
3. Fetch the cursor into pl/sql variables
4. Close the cursor

Declaring the Cursor

```
CURSOR c_customers IS
   SELECT id, name, address FROM customers;
```

Opening the Cursor

```
OPEN c_customers;
```

Fetching the Cursor

```
FETCH c_customers INTO c_id, c_name, c_addr;
```

Closing the cursor

```
CLOSE c_customers;
```

Example – Explicit Cursor

```
declare
    v_client_no client_master.client_no%type;
    v_name CLIENT_MASTER.NAME%type;
    v_city client_master.city%type;

    v_state client_master.state%type:='Maharashtra';

    cursor C_client_master is select client_no,name,
city from client_master where state=v_state;
begin
open c_client_master;
loop
    fetch c_client_master into
v_client_no,v_name,v_city ;

dbms_output.put_line(v_client_no||'
'||v_name||'     '||v_city);
    exit when
c_client_master%notfound;
end loop;

close c_client_master;
end;
```