SCSX1056 – ORACLE & SQL
UNIT IV - INTRODUCTION TO PL/SQL

Advantages – Architecture – PL/SQL Block – Data types and Usage – User – defined data types – Control Structures – Concept of Error Handing.

PL/SQL

PL/SQL stands for Procedural Language/SQL. PL/SQL extends SQL by adding control structures found on other procedural language. Procedural constructs blend seamlessly with Oracle SQL, resulting in a structured, powerful language. PL/SQL is unique as it combines the flexibility of SQL with the power and configurability of a third generation language. The procedural constructs and database access are present in PL/SQL.

PL/SQL can be used in the oracle relational database in the oracle server and in client side application development tools. Oracle8 supports PL/SQL version 8.0.3. PL/SQL8 and higher version of PL/SQL support many enhancements of oracle8, including large objects, object oriented design and development and collections (varying arrays and nested tables).

Advantages of PL/SQL

PL/SQL is a completely portable, high performance transaction processing language, which offers the following advantages.

- Supports the declaration and manipulation of object types and collections
- Allows the calling of external functions and procedures
- Contains new libraries of built-in packages.

PL/SQL has been upgraded to support directly most of the new features of the DBMS but not partitioned objects. There is an extensive support for the new type system and the support for tables has been further extended.

The other advantages are:

Support for SQL

PL/SQL allows us to use all SQL data manipulation commands, transaction control commands, SQL functions (except group functions), operators and pseudocolumns, thus allowing us to manipulate data values in a table more flexibly and effectively.

Higher productivity

PL/SQL can be used to include procedural constructs in non-procedural tools like oracle forms5 to build applications. Further, PL/SQL remains the same in all environments.

Better performance

Without PL/SQL oracle must process SQL statements one at a time. With PL/SQL, an entire block of statements can be processed in a single command line statement. This reduces the time taken to communicate between the application and the oracle server, thus enhancing performance.

Portability

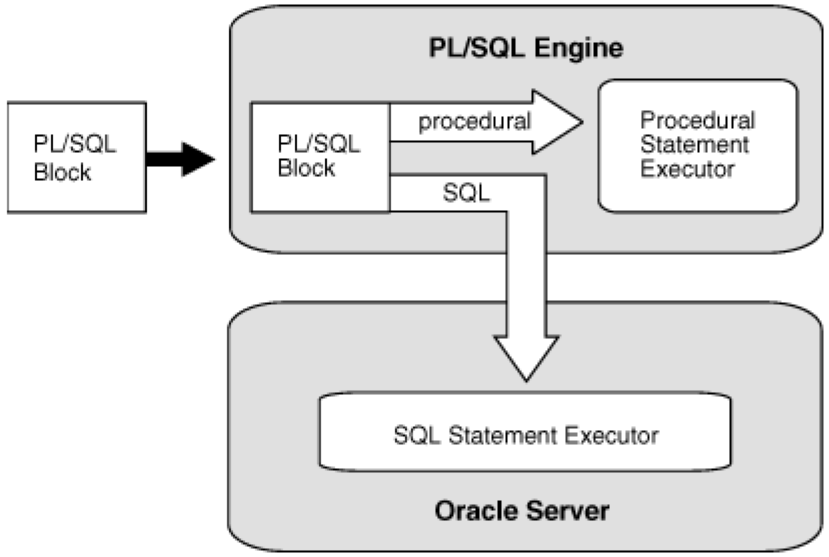
Applications written in PL/SQL are portable to any operating system or platform on which oracle ver 6.0 or higher are installed.

Integration with oracle

Both PL/SQL and oracle have their foundations in SQL. PL/SQL supports all the SQL data types and it integrates PL/SQL with the oracle data dictionary.

Architecture of PL/SQL

The PL/SQL engine executes PL/SQL blocks. The PL/SQL engine executes only the procedural statements and sends the SQL statements to the SQL statement executor in the oracle server. The PL/SQL engine can either reside in the oracle server or on oracle tools such as oracle forms 5, reports 3.



In the above block diagram, we find that the PL/SQL engine resides in the oracle server. It executes only the procedural statements in the block and sends the SQL statements to the SQL statement executor, which also resides in the oracle server.

Introduction to PL/SQL block

A PL/SQL block can be divided into three parts, namely, a declarative part, an executable part and an exception handling part. The order is as shown below:

```
DECLARE
   Declarations
BEGIN
   Executable statements
EXCEPTION
   Handlers
END;
```

Objects can be declared in the declarative part, which can be used, in the executable part for further manipulations. All procedural statements are included between the BEGIN and END statements. Errors that occur during execution are dealt in the exception handling part.

Before proceeding to learn about the above three parts, we need to have a brief idea on the character set and lexical units used in the PL/SQL block.

The PL/SQL character set includes the following:

- Upper and lower case letters. They are not case sensitive except within strings and character literals.
- Numerals from 0 to 9.
- All special symbols and characters.
- Tab, space and carriage return.

PL/SQL text can contain groups of characters known as lexical units. The following are the lexical units.

- Identifiers
- Literals
- Comments
- Delimiters (simple and compound symbols)

The following example illustrates the features discussed above

Example

Total:= salary * 0.90; -- to compute total

In the above example

```
Total and salary  ---- identifiers
* and ;             ----- simple symbols
:=                  ------ compound symbols
0.90                  ------ numeric literals
    --              ----- represents comment
```

some of the simple symbols are +,-,*,/,=,<,>,%(attribute indicator), ;(statement terminator) and : (host variable indicator). The compound symbols consist of two characters like <>, !=, :=(assignment), || (concatenation), --(single line comment), **(exponentiation) /* */ (multiple comment), ..(range operator), << >>(label delimeter).

Data types and their usage

PL/SQL data types can be classified into scalar and composite data types.

Scalar data types

Scalar data types include all SQL data types and ANSI standard data types. The data types include char, varchar2, long, long raw, raw, date, Boolean and binary_integer. Scalar data types comprise of Boolean, binary_integer and defined below.

Boolean

Boolean data types can be used to store the values TRUE, FALSE or NULL. They do not take any parameters. We cannot insert a Boolean data type in a database column. We cannot fetch column values into a Boolean variable.

Binary_integer

Binary_integer is used to store singed integers. The magnitude range of a binary_integer value is -2^{31-1} .. 2^{31-1}. For convenience PL/SQL predefines the following binary_integer subtypes.

- Natural (0--2^{31-1})
- Positive (0--2^{31-1})

We can use these subtypes to restrict a variable to positive integer values.

Number

It is similar to the SQL number data type. It also includes ANSI standard types, which comprise of the following data types.

- Dec/decimal
- Int/integer
- Real

Variables

We can declare variables in the declarative part and use them elsewhere in the body of PL/SQL block, i.e. we can use them in the SQL and procedural statements. To declare a variable named in_stock with a width of 4 bytes we can issue the following statement:

```
In_strock number(4);
```

Similarly we can assign a Boolean data type to a variable as shown below.

```
Done Boolean;
```

Constants

We can declare constants in the declarative part and can use them elsewhere in the executable part. To declare a constant we must make use of the keyword constant. This keyword must precede the data type as shown below:

Creditlimit constant real:=7000.00;

In the above example, we have assigned 7000.00 to the constant named creditlimit. After this, no more assignment to the constant is allowed i.e., 7000.00 will be the initial and final value to the constant.

Character

Variables of the type and used to store strings or character data. The various subtypes include:

- Varchar2 – this behaves similar to the varchar2 database type. Variables of this type can hold variable length character strings. In orcle8 a varchar2 database column can hold 4000 bytes

- Char – variables of this type ore of fixed length. The length is not optional. If the length is not specified it defaults to 1. in oracle8, a char database column can hold upto 2000 bytes.

- Long – unlike the long type of SQL, the PL/SQL long type is a variable length string with a maximum length of 32,760 bytes. Long data type is similar to varchar2 variables.

Raw

Raw types are sued to store binary data. Character variables are automatically converted between character sets by oracle, if necessary. These are similar to char variables, except that they are not converted between character sets. It is used to store fixed length binary data. The maximum length of a raw variable is 32,767 bytes. However, the maximum length of a database raw column is 255 bytes.
Long raw is similar to long data, except that PL/SQL will not convert between character sets. The maximum length of long raw variable is 32,760 bytes. The maximum length of a long raw column is 2 GB.

Rowid

This is the same as the database rowid pseudocolumn type. It can hold a rowid, which can be considered as a uniaque key for every row in the database. Rowids are stored internally as a fixed length binary quantity, whose length varies depending on the operating system.

Composite types

The three composite types available in PL/SQL are records, tables, and varrays. A composite type is one that has components within it. A variable of a composite type contains one or more scalar variables.

LOB types

The LOB types are used to store large objects. A large object can be either a binary or a character value upto 4 GB in size. Large objects can contain unstructured data, which is accessed more efficiently than long or long raw data, with fewer restrictions. LOB types are manipulated using the DBMS_LOB package. There are four types of LOBs and they are:

- BLOB (Binary LOB) – this stores unstructured binary data up to 4 GB in length. A blob could contain video or picture information.
- CLOB (Character LOB) – this stores single byte character up to 4GB in length. This might be used to store documents.
- BFILE (Binary File) – it stores read only binary data as an external file outside the database.

Storage of LOB data

Create table vendor_master(v_code varchar2(5), v_name varchar2(15), v_add1 varchar2(20), v_add2 varchar2(20), v_add3 varchar2(20), tel_no number(10), msg clob);

Initializing LOB values

Insert into vendor_master values(‘v201’,’arkay’, ‘10’, ‘first st’, ‘mds’ 4343434, ‘this customer is a instant pay customer. Pays money as soon as the goods are delivered can be trusted’);

User defined data types

There are two different types of objects namely, persistent and transient. Persistent objects are those that are stored in the database. These can be used both with SQL commands and also in PL/SQL blocks. Persistent objects reside in the data dictionary. Persistent objects are available to the user until they are explicitly deleted. Persistent objects could be implemented as tables, columns or attributes. Transient objects exist only within the scope of the PL/SQL block. These get automatically deallocated once they go out of the scope of the PL/SQL block. Examples of transient objects include PL/SQL variables.

Let us assume that the user has a user-defined data types address_ty. This user defined data type has a structure as shown below:

Name	Type
STREET_NO	NUMBER(3)
STREET_NAME	VARCHAR2(20)
CITY	VARCHAR2(20)
STATE	VARCHAR2(20)

A persistent object address_ty is used in a PL/SQL block and the value that is initialized become transient as the object is out of scope as soon as the PL/SQL object goes out of scope.

Declare

```
    a address_ty := address_ty(10.'first st' 'chenai','tn');
Begin
    Dbms_output.put_line('The street number' || a.street_no);
    Dbms_output.put_line('The street name' || a.street_name);
    Dbms_output.put_line('The city is' || a.city);
    Dbms_output.put_line('The state is' || a.state);
End;
```

The above PL/SQL procedure highlights two important things. One is the use of the constructor method to initialize the object type. The other is the usage of the Dbms_output.put_line. The dbms_output package allows displaying of messages and variable values on the console.

Attributes

Attributes allow us to refer to data types and objects from the database. PL/SQL variables and constants can have attributes. The following are the types of attributes supported by PL/SQL.

- %type
- %rowtype

%type

%type attribute is used when declaring variables that refer to the database columns. Consider the following example where a variable called vcode is declared to be of type vencode in the item table using %type attribute.

Declare

```
vcode vendor_master.vencode %type;
```

Where vcode is the variable name, vendor_master is the name of the table and vencode is the name of the column.

The advantages of using %type is,

- We need not know the exact data type of the column 'vencode'.
- If the database definition of 'vencode' is changed, then, the data type of 'vcode' changes accordingly at run time.

%rowtype

%rowtype attribute provides a record type that represents a row in a table. The record can store an entire row of data selected from the table or fetched by a cursor. In the following example, a record named 'vend_inf' will have the same names and data types as the columns in the customer table.

```
Declare
vend_inf  vendor_master%rowtype;
```

Logical comparisons

PL/SQL supports the comparison of variables and constants in SQL and PL/SQL statements. These comparisons, called 'Boolean expressions', are often connected by logical operators AND, OR and NOT. The following are the three kinds of Boolean expressions.

- Numeric
- Character
- Date

Numeric Boolean expression

We can compare numbers using numeric Boolean expressions. The table given below illustrates relational operators and their meanings when used with numeric values.

OPERATOR	MEANING	EXAMPLE
=	is equal to	A=434
!=	Is not equal to	C!=5656
<	Is lesser than	A<1

>	Is greater than	B>4
<=	Is lesser than or equal to	A<=b
>=	Is greater than or equal to	a>=c

Character Boolean expressions

A sequence of characters or individual characters enclosed in a string can be compared using character Boolean expressions. Their evaluation is based on the alphabetical ordering of the character strings. The table given below illustrates relational operators and their meanings when used with character strings.

OPERATOR	MEANING	EXAMPLE
=	is same as	Name='vivek'
!=	Is not same as	Product != 'computer'
<	Comes alphabetically before	Name1<name2
>	Comes alphabetically after	Name1>name2
<=	Comes alphabetically before or is the same as	Name2<=name3
>=	Comes alphabetically afteror is the same as	Namee3>=name4

Date Boolean expressions

We can compare two dates using date Boolean expressions. The table illustrates the relational operators and their meanings when used with dates.

OPERATOR	MEANING	EXAMPLE
=	is same as	Odate = '12-jan-74'
!=	Is not same as	Odate != '14-feb-88'
<	Is earlier than	Odate<'19-jun-88'
>	Is later than	Odate>'30-jun-77'
<=	Is earlier than or the same as	Next_day(odate,'friday')<='1-jan-97'
>=	Is later than or the same as	Odate>='25-dec-67'

Control structures

PL/SQL can also process data using flow of control statements. The flow of control statements can be classified under the following categories:

- Conditional control

- Iterative control
- Sequential control

Conditional control

Sequence of statements can be executed based on a certain condition using the **if** statement. There are three forms of if statements, namely, **if then**, **if then else** and **if then elsif**. The simplest form of an if statement is the if then statement. The syntax is

```
                  If condition then
                  Sequence of statements:
                  End if;
```

The sequence of statements is executed only if the condition evaluates to true. If it is false or null, then, the control passes to the statement after 'end if'. An 'else' clause in the 'if then' statement defines the steps or processes to be performed if the condition is false or null. An 'if then elsif' statement can be used to select one of several mutually exclusive alternatives. The following example illustrates the if then else statement.

```
Declare
                  Orderstatus order_master.ostatus%type;
Begin
                  Select ostatus into orderstatus from order_master where
orderno='0001';
                  If orderstatus='p' then
                  Update order_master set odate='9-jan-99' where orderno='0001';
                  Else
                  Update order_master set odate='26-jan-99' where orderno='0001';
                  End if;
End;
```

The output of the above command is

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

Iterative control

A sequence of statements can be executed any number of times using loop constructs. Loops can be broadly classified into:

- Simple loop
- For loop
- While loop

Simple Loop

Syntax:

```
 Loop
Sequence of statements;
Exit when condition
End loop;
```

example

```
declare
   a number:=100;
begin
loop
  a:=a+25;
exit when a =250;
end loop;
dbms_output.put_line(to_char(a));
end;
```

the exit when statement allows to complete a loop if further processing is undesirable or imposible.

The output PL/SQL command is

PL/SQL procedure successfully completed.

While Loop

Syntax:

```
 While <condition>
Loop
Sequence of statements
End loop;
```

While loop

The while loop statement indicates a condition associated with a sequence of statement. If the condition evaluates to true, then the sequence of statements will be executed, and again control resumes at the beginning of the loop. If the condition evaluates to false, then the loop is bypassed and the control passes to the next statement.

Example

```
Declare
I number:=0;
J number:=0;
Begin
While I<=100 loop
J:=J+I;
I:=I+2;
End loop;
dbms_output.put_line('the value of j is'|| J);
end;
```

the output of the above PL/SQL is given below

```
the value of j is 2550
PL/SQL procedure successfully completed.
```

For Loop

__**Syntax:**__

```
  For counter in [reverse] lower bound…upper bound
Loop
    Sequence of statements
End Loop;
```

The number of iterations for a while loop is unknown until the loop terminates, whereas the number of iterations in a **for loop** is known before the loop gets executed. The **for loop** statement specifies a range of integers, to execute the sequence of statements once for each integer.
By default, iteration proceeds from lowerbound to upperbound. If we use the optional keyword reverse, then, iteration proceeds downwards from upperbound to lowerbound. The following example executes the update statement once for each integer specified in the 'for loop'.
Example

```
Begin
       For I in 1..2
       Loop
               Update order_master set ostatus='p' where odate<sysdate;
       End loop;
       End;
```

The output of the above PL/SQL procedure is given below:

PL/SQL procedure successfully completed.

Sequential control

The goto statement allows us to branch to a label unconditionally. The label, which is enclosed within double angle brackets, must precede an executable SQL statement or a PL/SQL block. When executed, the goto statement transfers control to the labeled statement or block.

Example

```
Declare
            Qtyhand itemfile.qty_hand%type;
            relevel itemfile.re_level%type;
begin
            select qty_hand,re_level into qtyhand, relevel from itemfile where
            itemcode='i201';
            if qtyhand<relevel then
            goto upadation;
            end if;
<<updation>>
            update itemfile set qty_hand =qyt_hand + re_level where
            itemcode='i201';
end;
```

the output of the above command is

PL/SQL procedure successfully completed.

Concept of error handling

Error condition in PL/SQL is termed as an exception. There are two types of exception. They are

- Predefined exception
- User-defined exception

An exception is raised when an error occurs. In case of an error, normal execution stops and the control is immediately transferred to the exception handling part of the PL/SQL block. Predefined exceptions are raised automatically by the system during run time, whereas user-defined exceptions must be raised explicitly using RAISE statements. Exceptions are explained in detail in the following sections.

Predefined exception

An exception is raised implicitly when a PL/SQL program violates oracle rule. The following are the predefined exceptions supported by PL/SQL

Predefined exception	Explanation
No_data_found	This exception is raised when select statement returns no rows.
Cursor_already_open	This exception is raised when we try to open a cursor which is already opened.
Dup_val_on_index	This exception is raised when we insert duplicate values in a column, which is defined as unique index.
Storage_cursor	This exception is raised if PL/SQL runs out of memory or if the memory is corrupted.
Program_error	This exception is raised if PL/SQL has an internal problem.
Zero_divide	This exception is raised when we try to divide a number by zero.
Invalid_cursor	This exception is raised when we violate cursor operation. For example, when we try to close a cursor which is not opened.
Login_denied	This exception is raised when we try to enter oracle using invalid username/password
Invalid_number	This exception is raised if the conversion of a character string to number fails because the string does not represent a valid number. For example, inserting 'john' for a column of type number will raise this exception.
Too_many_rows	Raised when the select into statement returns more than one row.

Syntax for predefined exception is as follows:

```
Begin
Sequence_of_statements;
Exception
            When <exception_name> then
            Sequence_of_statements;
            When others then
            Sequence_of_statements;
End;
```

Where ‘others’ handler guarantees that no exception will go unhandled. A PL/SQL block can have only one ‘others’ handler.
Example

```
Declare

                Qtyhand itemfile.qty_hand%type;
                relevel itemfile.re_level%type;
begin
                select qty_hand,re_level into qtyhand, relevel from itemfile where
                itemcode=’i201’;
Exception
When no_data_found then
dbms_output.put_line(‘such an item number not availanble’);
end;
```

the output of the above command is

```
such an item number not availanble
PL/SQL procedure successfully completed.
```

User-defined exception

A user-defined exception should be declared and raised explicitly by a ‘raise’ statements. It can be declared only in the declarative part of the PL/SQL block. The syntax is

```
<exception_name> Exception;
```

the syntax for a ‘raise’statement is as follows:

```
raise < Exception_name>;
```

example

```
declare
                lo_value exception;
                Qtyhand itemfile.qty_hand%type;

begin
                select qty_hand,re_level into qtyhand, relevel from itemfile where
                itemcode=’i201’;
                if qty_hand < 200 then
                raise lo_value;
                end if;
```

```
                 exception
                 when lo_value then
                 dbms_output.put_line('quantity not enough  … reorder');
end;
```

the output of the above command is

```
quantity not enough  … reorder
PL/SQL procedure successfully completed.
```