# PL/SQL

* It stands for procedural language.
* By using this PL/SQL we can define our own data type.
* We can execute more than one statement at a time.
* By using PL/SQL we can execute procedure & functions.
* PL/SQL is having higher productivity (non PL/SQL block) like forms & reports.
* By using PL/SQL we can trap runtime errors.
* It has portability.
* Higher speed

**PL/SQL Data types:** It supports all SQL data types.

**(char, varchar, long, raw(l), longraw(l), number, date, blob, clob, bfile)**

**Const:** It is used to define constant variable . Example: const pie := 3.14;

**Boolean:** It returns either true or false, It occupies 1 byte of memory.

**%type:** It is used to retrieve only one column data type.

Example: b emp.empno%type;

**%rowtype:** It is used to retrieve entire row data type.

**Example:** var emp%rowtype;

**Block Structure:**

**Declare**

…declare block

All the declaration of the program will be done here.

**Begin**

…begin block

…main block

Main executable code will be written here.

**Exception**

…exception block

Error handling code will be written here.

**End;**

**/**

**Note:-** In PL/SQL program declare & exception is optional.

**Operators:**

**Arithmetic:**  +, -, \*, /

**Relational:**  =, >, <, <=, >=, <> (or) !=

**Logical:** and, or, not

**Concatenation:** ||

**Assignment:**  :=

**Example:** d := b+c; a := 25;

**Rules to write a program:**

1. It is not a case sensitive except with the data of the table.
2. Every statement has to be terminated by using semicolon (;) except conditional statement.
3. We have to follow the structure of PL/SQL program.

**To Open PL/SQL program:**

**Syntax:** edit a.sql (in Notepad)

**Example:** **To add two numbers**

**Syn: Edit a.sql** Declare

x number(4) := 25; y number(4) := 33; res number(5); Begin

res := x + y;

dbms\_output.put\_line(res);

End;

/

**Dbms\_output.put\_line ( ):**

 It is used to print variable values or messages on the screen.  It is similar to ‘C- Lang.’ Printf statement.

**Set Server Output On:** It is used to active the server to print the output on the screen.

**Set Server Output Off:** It doesn’t displays the output.

**Execution:**

**SQL>** @ a.sql; (or) **SQL>** get a.sql; (or) **SQL>** start a.sql; (or) **SQL>** /

**Note:** If we want to retrieve the values from key board at runtime use the following command.

1. number(4) := &A;
2. number(4) := &B;

**Control Statements:** The flow of statement execution in a program is called as Control Statement. These are three types.

1. **Conditional Statements**
2. **Iterative Control Statements**
3. **Sequential Control Statements**

**Conditional Statements:** These are again three types.

**Syntax’s:**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Simple If** |  | **If …. Else** |  | **Nested If …. Else** |
| If (Condition) then |  | If (Condition) then |  | If (Condition1) then |
| Statements |  | Statements |  | Statements |
| End if; |  | else |  | elsif (Condition2) then |
|  |  | Statements |  | Statements |
|  |  | end if; |  | else |
|  |  |  |  | Statements |
|  |  |  |  | end if; |
| **Iterative Control Statements:** | | |

* Simple Loop (or) loop …exit when (condition)
* While Loop
* For Loop

**Simple Loop:** It is used to execute same statement until the condition become true.

**Syntax:** Loop

Statements;

Exit when (condition);

End loop;

**Program:** WAP to print 1 to 10 numbers

Declare

A number(2) := 1;

Begin

Loop

dbms\_output.put\_line(A);

A := A+1;

Exit when (A>10);

End loop;

End;

/

**While Loop:** It is also looping control statement by using this same statements until the condition becomes false. **Syntax:** While (Condition) loop

Statements;

End loop;

**Program:** WAP to print 1 to 10 numbers

Declare

A number(2) := 1;

Begin

While (A <= 10) loop dbms\_output.put\_line(A);

A := A+1;

End loop;

End;

/

**For Loop:** It is also looping control statement it repeats the same statements until the given condition false.

**Syntax:** For <variablename> in lowerbound .. upperbound loop

Statements

End loop;

**Program:** WAP to print 1 to 10 numbers

Declare

A number(2) := 1;

Begin

For A in 1 to 10 loop dbms\_output.put\_line(A);

End loop;

End;

/

**Sub Programs:** Sub programs are PL/SQL programs which can be called in some other PL/SQL program.

**Types of Sub Programs:**

1) Procedures

2) Functions.

Except functions the behavior is to return a value to the calling program, there is no basic difference between a procedure and functions.

**Procedures:** Generally, Procedures are used to perform a specific task. It does not return any value.

* It contains block of statements.
* A procedure doesn’t return any value.
* Procedures support in parameters & out parameters.
* In procedures the formal parameters are reflecting on actual parameters.

**Note 1:** Calling function parameters are called as actual parameters. **Note 2:** Called function parameters are called as formal parameters.

**In:** It parameter performs only read operations.

**Out:** It parameter performs only write operations.

**Syntax:** Create or replace procedure <procedure name> ( - , - ) is

Local variables;

Begin

Statements;

End <procedure name>;

**Example;** WAP to find the square value of given number by procedure.

Create or replace procedure sqr (a in out number) is

Begin

a := a\*a;

End;

/

**Execution:** If we want call above procedure gives the following command.  **Example: SQL>** exec sqr (5);

**Functions:** Functions will return a value.

* These are nothing but a block of statements.
* Functions always return the value.
* Function supports only parameters.
* In functions formal parameters are not affected by actual parameters.  We can define nested functions.

**Syntax:** Create or replace function <function name> ( - , - ) return data type is

Local variables;

Begin

Statements;

End function;

/

**Example:** WAP to find the sum of two numbers by using functions.

Declare

1. number :=&A;
2. number :=&B;
3. number;

Function f2 (x in number, y in number) return number is

Z number;

Begin Z := x+y; return Z; end f2;

Begin

C:=f2(A,B); dbms\_output.put\_line(C);

End;

/

**Package:** A package is an encapsulation of all related sub programs (Procedures & Functions) with Cursors, Exceptions, Variables, Constants.

By using a package, we can store all the related sub programs in a single location.

A Package will have two parts. (1) Package Specification. (2) Package Body. If we want to call any function or procedure of package we need to specify the package name preceded by function or procedure.

**Creation of Package Specification:**

**Syntax:** Create or replace package <package name> is

Procedure <procedure name> ( - , - )

Function <function name> ( - , - ) return data type

End <package name>;

/

**Example: SQL>** Create or replace package ram is

Procedure cal (a number, b number, c char);

End ram;

**Creation of Package Body:**

**Syntax:** Create or replace package body <package name> as

Procedure <procedure name> ( - , - ) is

Local variables;

Begin

Statements;

End <procedure name>;

Function <function name> ( - , - ) return data type is

Local variables;

Begin

Statements;

End <function name>;

End <package name>;

/

**Example: SQL>**

Create or replace package body ram as

Procedure cal (a number, b number, c char) is

r number;

Begin

If c = ‘+’ then

r := a + b;

elsif c := ‘-‘ then

r := a – b; else

dbms\_output.put\_line(‘Invalid’);

end if;

if c in (‘+’ , ‘-‘) then

dbms\_output.put\_line(a || c || b || ‘=’ || r); end if;

End cal;

End ram; /

**Calling the Package Procedure:**

**SQL>** exec ram.cal (8,2,’+’);

**Triggers:** A trigger is a PL/SQL program which fires (executes) automatically as soon as the triggering event (situation) occurs.

A trigger must be associated to a particular table.

In ORACLE 7.3 & older versions, a table can have a maximum of 12 triggers, from ORACLE 8i, there is no limitations in the no.of triggers on particular table.

* These are nothing but constraints by using this we can provide security to the databases.
* By using this we can restrict invalid data also.

**Syntax:** Create or replace trigger <trigger name> before/after insert/update/delete on

<table name> Begin

Statements;

End;

/

**Example:** WAP to raise the alert message before inserting the record into the table emp.

**SQL>** Create or replace trigger emp\_alert\_trig before insert on emp

Begin

Dbms\_output.put\_line(‘New Employees are added to the table’);

End;

/

**Note:** We can define the triggers.

Before Insert / Before Update / Before Delete After Insert / After Update / After Delete **Cursors:** A work area called private SQL area is used by the ORACLE Server to execute the SQL statements & to store the processed information temporally.

By using Cursors, we can name these private SQL areas & can manipulate the rows present in that area by getting them into the program individually.

When we are executing SQL statement it displays the result set for this we have to use SQL private area.

To manipulate the SQL private area we have to use cursors.

1. **Implicit Cursors:** For the SQL queries that fetches only one row By using this at a time we can fetch only one record, ORACLE declares it as implicit cursor automatically.

It does not supports attributes like %found, %notfound, %isopen, %rowcount.

1. **Explicit Cursors:** For the SQL queries that fetches more than one row, the Cursor needs to be declared explicitly by the user.

Explicit Cursor has to be declared in the declarative block by using following syntax.

By using this we can fetch more than one record at a time. It supports all attributes.

**Syn:** Cursor <Cursor name> is <SQL statement>; **Example:** Cursor abc is select \* from emp;

**Explicit Cursor manipulation can be done by using the following Commands:**

1. **Open:** Used to open the Cursor. **Syn:** Open <Cursor name>;
2. **Fetch:** Used to get the individual rows into the program by storing them into the local variables. **Syn:** fetch <Cursor name> into <Local Variables>;
3. **Close:** Used to close the Cursor. **Syn:** Close <Cursor name>;

**Attributes:**

**%FOUND:** It returns either true or false value. If the fetched record is available in the result set it result true other wise it returns false value.

**%NOTFOUND:** It also returns either true or false value. If the record is available it returns false value, if the record is not available it returns true value.

**%ISOPEN:** It is used to check whether the cursor is open or not.

**%ROWCOUNT:** It counts no. of fetched records.

While working with cursors we have to follow the following statements.

* + Creating Cursor
  + Open Cursor
  + Fetch the Records
  + Close Cursor

**Example:** WAP by using cursor to display the empno & sal

Declare

Cursor c1 is select \* from emp;

X c1%rowtype;

E emp.empno%type;

N emp.sal%type;

Begin

Open c1;

Loop

Fetch c1 into X;

Exit when c1%notfound;

E := X.empno;

N := X.sal;

Dbms\_output.put\_line (E || ‘ ‘ || N);

End Loop;

Close c1;

End;

/

**Exception Handling:** Exception is nothing but error when the exception is raised automatically the program gets abnormal termination by exception handling. We can terminate the program even if the exception rose also. We have two types.

**Defined Exception**

 **Predefined Exceptions**

## Undefined Exception

 **User defined Exceptions**

**Defined Exception:** These are having no and names.

**Undefined Exception:** These are having only no.

 **SQL CODE:** It is used to display the exception no. of predefined exceptions.

 **SQL ERRM:** It is used to display the exception name of the predefined exceptions.

**User defined Exception:** These are raised by users according to their requirements.

**Note:**

* Enter the raise statement names the appropriate exception.
* A raise statement stops the exception of procedure & control passes to the exception handler block.
* User defined error number range be -20000 to 20999.
* Call the raise\_application\_error procedure to returns the user specified error no & message.

**Embedded SQL:** The SQL standard defines embedding of SQL in variety of programming languages, such as Pascal, PL/I, C and Cobol. A language in which SQL queries are embedded is referred to as host languages, and the SQL structures permitted in the host languages constitute embedded SQL.

**Features of Embedded SQL:**

* Programs written in the host languages can use the embedded SQL system to access and update data stored in a database.
* In embedded SQL, all query processing is performed by the database system.
* It allows the results of the query to be available to the program one record at a time.
* The embedded SQL statements appear in the middle of host programming language.
* These statements are prefixed by an EXEC SQL to differ from host languages statements.

**Mr. I.S.Raghuram**

**MYSQL DATA TYPES**

**TEXT TYPES**

|  |  |
| --- | --- |
| CHAR ( ) | A fixed section from 0 to 255 characters long |
| VARCHAR ( ) | A variable section from 0 to 255 characters long |
| TINYTEXT | A string with a maximum length of 255 characters |
| TEXT | A string with a maximum length of 65535 characters |
| BLOB | A string with a maximum length of 65535 characters |
| MEDIUMTEXT | A string with a maximum length of 16777215 characters |
| MEDIUMBLOB | A string with a maximum length of 16777215 characters |
| LONGTEXT | A string with a maximum length of 4294967295 characters |
| LONGBLOB    **NUMBER TYPES** | A string with a maximum length of 4294967295 characters |
| TINYINT( ) | -128 to 127 normal  0 to 255 UNSIGNED. |
| SMALLINT( ) | -32768 to 32767 normal  0 to 65535 UNSIGNED. |
| MEDIUMINT( ) | -8388608 to 8388607 normal  0 to 16777215 UNSIGNED. |
| INT( ) | -2147483648 to 2147483647 normal  0 to 4294967295 UNSIGNED. |
| BIGINT( ) | -9223372036854775808 to 9223372036854775807 normal 0 to 18446744073709551615 UNSIGNED. |
| FLOAT | A small number with a floating decimal point. |
| DOUBLE( , ) | A large number with a floating decimal point. |
| DECIMAL( , )    **DATE TYPES** | A DOUBLE stored as a string , allowing for a fixed decimal point. |
| DATE | YYYY-MM-DD |
| DATETIME | YYYY-MM-DD HH:MM:SS |
| TIMESTAMP | YYYYMMDDHHMMSS |
| TIME | HH:MM:SS |

**MISLINIOUS TYPES**

ENUM ( )

Short for ENUMERATION which means that each column may have one of a specified possible values

SET

Similar to ENUM except each column may have more than one of the specified possible values.

**Page**