Basic Syntax of PL/SQL.

PL/SQL → block structured language i.e, programs are divided and written in logical blocks of code.

Each block consists of three subparts

1. Declarations.
2. Executable commands.
3. Exception Handling.

(1) - Starts with the keyword DECLARE
- optional
- Defines all variables, cursors, subprograms and other elements.

(2) - Enclosed between the keyword BEGIN and END.
- mandatory section.
- Should have atleast one executable line of code.
- may be just a NULL command.

(3) - Starts with the keyword EXCEPTION
- optional
- contains exception(s) that handle errors in the program.

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

Note

- Every PL/SQL statement ends with a semicolon (;) and last a slash (/) included.
- at first blank line after last line of code.
- to run the code from the command line.
- 'Hello world' program

```
DECLARE
   message varchar2(20) := 'Hello world';
```

```
BEGIN
   dbms_output.put_line(message);
END;
/
```

PL/SQL identifiers

- constants, variables, exceptions, procedures, cursors and reserved words
- consists of letter (a) followed by numerals, dollar signs, underscores and number signs
- Not exceed 30 characters
- Not case-sensitive

PL/SQL delimiters

a symbol with a special meaning

+, -, *, / → Addition, sub - - .

Example Program

```
set serveroutput on.

DECLARE
message varchar(20) := 'Hello World'.
BEGIN
dbms_output.put_line(message);
END;
/
```

O/P Hello World

PL/SQL procedure successfully completed.

Sum of two numbers.

```
DECLARE
 x number(5);
 y number(5);
 c number(5);
BEGIN
 x := 3
 y := 4
 c := x + y;
dbms_output.put_line('sum is ' || c);
END;
/
```

O/P Sum is 7.

PL/SQL procedure successfully completed.

```
DECLARE
 x number(5);
 y number(5);
 c number(5);
BEGIN
 x := &x;
 y := &y;
 c := x + y;
dbms_output.put_line('Sum is' || c);
END;
/
```

(64 line numbers)

O/P Enter value for x : 5
old 6 : x := &x;
new 6 : x := 5;
Enter value for y : 4
old 7 : y := &y;
new 7 : y := 4;
Sum is 9.

PL/SQL procedure successfully completed.

PL/SQL

- All SQL queries including select, insert, update - can be performed within the 'begin' and 'end' of PL/SQL block.

Assigning SQL queries results to PL/SQL variables.

- use SELECT INTO statement
- For each item in the 'SELECT' list, there must be a corresponding type-compatible variable in the 'INTO' list.
  Eg: customer (id, name, age, address, salary)
- To assign values from above table to PL/SQL variables.

```
DECLARE
cid customer.id %type;
cname customer.name %type;
caddr customer.address %type;
cage customer.age %type;
c sal customer.salary %type;
BEGIN
SELECT name, address, salary INTO cname, caddr, csal FROM
customer where id = cid;
dbms_output.put_line('customer' || cname || 'from' ||caddr ||
'earn' || csal);
END;
/
```

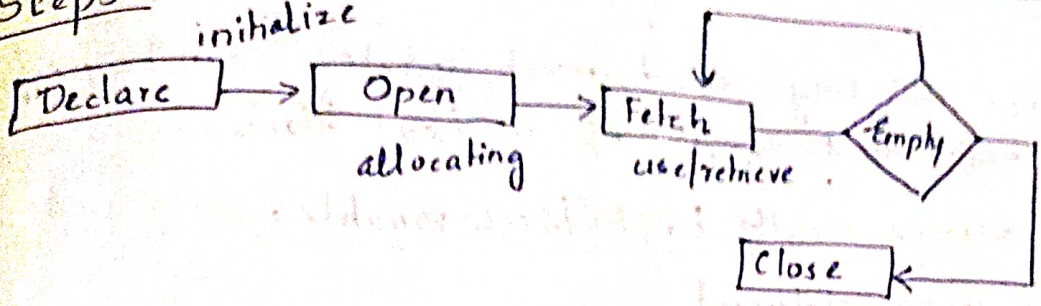
Cursor

- Oracle creates a memory area - 'context area' for processing an SQL statement.
- A cursor is a 'pointer' to this context area.
- Holds the rows returned by a SQL statement.
- The set of rows cursor holds ⟹ 'active set'
- To work with one row or a subset of result set of SQL statements cursors used.

Types of cursors.

* Implicit cursors → Automatically created by oracle, when an SQL statement is executed.

Steps:



Declare - syntax:

CURSOR cursor_name [([parameter1 . . . [. . .]])]
[RETURN returntype specification] IS SQL_select_statements
[FOR update [of [column list]];

Open - syntax.

OPEN cursor_name [(argument, [argument2] . .)];

Fetch - syntax.

FETCH cursor_name INTO record or variables.

- to check if fetched or succeeded or not use.
" %FOUND" or "%NOT FOUND" attributes.
- to loop through all records use "LOOP" statement.

Syntax
Loop
fetch cursor_name into record_name;
exit when cursor_name %NOT FOUND;
dbms_output.put_line (record_name / fieldname)
end loop;

Close - Syntax

close cursor_name;

PL/SQL function.

- named block that returns a single value.
(Procedure multiple values returns)

Syntax

```
CREATE [OR REPLACE] FUNCTION fn_name
[ (parameter1 [IN] [OUT] datatype,
   - - - - - - - - - - - - - ] )
RETURN return_datatype IS/AS
BEGIN
   - - -
   return return ddatatype;
   EXCEPTION
      - - -
END;
/
```

IN → to pass parameters into function.

OUT → return value back to the calling program.

IN OUT → same parameter as i/p and o/p

Eg of PL/SQL function.

```
create or replace function [n1 IN number
n2 IN number]
RETURN number
is
n3 number (8);
begin
n3 = n1+n2;
return n3;
END;
/
```

To call the function.

```
DECLARE
    n3 number(2);
BEGIN
    n3 := add (11, 22);
dbms_output.put_line (n3)
end;
```

Triggers

- stored programs, which are automatedlly executed or fired when some event occur.
- They are written to be executed in response to any of the following events.

(1) DML statement

(2) DDL statement

(3) A database operation like startup, shutdown, logon, logoff, server etc.

Syntax

```
Create [or replace] trigger triggername
{BEFORE/AFTER/INSTEAD OF}  creating trigger on a view.
{INSERT [OR] | UPDATE [OR] | DELETE}
  [OF col_name]
  ON tablename
  [REFERENCING OLD AS O. NEW AS ]
  [FOR EACH ROW]        \ refer new & old value from DML

  WHEN (condtion)

  DECLARE
     Declaration stt.

  Begin
     Executable stts.

  Exception
     exception_handling stts.

  end;
```

Example.

```
create or replace trigger sqlchange
Before delete or insert or update on
customers
for each row
when (new.id >0)
declare
   sal_diff number;
Begin
   sal_diff := :NEW.salary - :OLD.salary,
   o/p  (:OLD.salary);
    "   (:new.salary);
        (sal_diff).
end;
/
```

To check if it is executed

```
DECLARE
  tot_rows number (2);
Begin
   update customers
   set salary = salary + 5000;
  If sql % not found then
   o/p ('not updated').
  else if sql % found then
   tot_rows := sql % row count;
   o/p (tot_rows || 'customers update)
   end if;
   end;
   /
```

view → virtual tables

- An SQL statement that is stored in the db with an associated name.
- created using 'CREATE VIEW stl.
- created from a single table, multiple tables or another view.
- If can be droped with DROP VIEW <view_name>

```
CREATE VIEW viewname as
SELECT column1, column2 - -
FROM table (c) name
WHERE [condition]
```

- can be updated, insert/delete values to/from it.

Eg; CUSTOMERS (ID, NAME, AGE, ADDRESS, SALARY)

```
CREATE VIEW CUST_VIEW AS
SELECT name, age FROM CUSTOMERS;
```