**The Islamic University of Gaza**
**Faculty of Engineering**
**Dept. of Computer Engineering**
**Database Lab (ECOM 4113)**

**Lab 7**

# Procedural Language Structured Query Language (PL/SQL)


**Eng. Ibraheem Lubbad**

Structured Query Language (SQL) is the primary language used to access and modify data in relational databases. There are only a few SQL commands you can easily learn and use them. However, if you want to alter any data that is retrieved in a conditional manner, you soon encounter the limitations of SQL.

**PL/SQL** is designed to meet more requirements than SQL. It provides a programming extension to already-existing SQL.

PL/SQL defines a block structure for writing code. Maintaining and debugging the code is made easier with such a structure. One can easily understand the flow and execution of the program unit.

PL/SQL offers modern software engineering features such as data encapsulation, exception handling, information hiding, and object orientation. It brings state-of-the-art programming to the Oracle server and toolset. PL/SQL provides all the procedural constructs that are available in any third-generation language (3GL).

## PL/SQL Block Structure:

A PL/SQL block consists of three sections:

- **Declarative (optional):** The declarative section begins with the keyword **DECLARE** and ends when the executable section starts.
- **Executable (required):** The executable section begins with the keyword **BEGIN** and ends with **END**. Observe that END is terminated with a semicolon. The executable section of a PL/SQL block can in turn include any number of PL/SQL blocks.
- **Exception handling (optional):** The exception section is nested within the executable section. This section begins with the keyword **EXCEPTION**

PL/SQL Block Structure

```
DECLARE (optional)
-- Variables, cursors, user-defined exceptions
BEGIN (mandatory)
-- SQL statements
-- PL/SQL statements
EXCEPTION (optional)
--Actions to perform when errors occur -- like try-catch  blocks in java
END; (mandatory)
```

## Block Types:

A PL/SQL program comprises one or more blocks. These blocks can be entirely separate or nested within another block. There are three types of blocks that make up a PL/SQL program. They are:

- **Anonymous blocks**
- **Procedures**
- **Functions**

Anonymous

```
[DECLARE]

BEGIN
--statements

[EXCEPTION]
END;
```

Procedures

```
PROCEDURE name
IS
BEGIN
--statements

[EXCEPTION]
END;
```

Functions

```
FUNCTION name
RETURN datatype
IS
BEGIN
--statements
RETURN value;
[EXCEPTION]
```

**Anonymous blocks:** Anonymous blocks are unnamed blocks. They are declared inline at the point in an application where they are to be executed and are compiled each time the application is executed. These blocks are not stored in the database. They are passed to the PL/SQL engine for execution at run time.

**Example:** create anonymous block to print “Hello World!” on the console

Example

```
SET SERVEROUTPUT ON;
DECLARE
BEGIN
  DBMS_OUTPUT.PUT_LINE(' Hello World! ');
END;
```

The command (**SET SERVEROUTPUT ON** ;) is used to enable output in SQL Developer.

### Declaring PL/SQL Variables

you can declare variables in the declarative part of any PL/SQL block, subprogram. Declarations allocate storage space for a value, specify its data type, and name the storage location so that you can reference it. In the executable section, the existing value of the variable can be replaced with the new value.

Syntax

```
IDENTIFIER [CONSTANT] DATATYPE [NOT NULL] [:= | DEFAULT EXPR];
```

Example 7.2

```
DECLARE
INST_ID  VARCHAR2(5) ;
INST_NAME  VARCHAR2(20) NOT NULL := 'IBRAHEEM';
SALARY  NUMBER(6,2) := 1400;
INST_HIREDATE DATE;
BUDGET CONSTANT NUMBER(12,2):=100000;
BEGIN
-- then use them in  executable section
END;
```

## Control Structures

### IF Statements:

Syntax of if statement

```
IF CONDITION THEN
  STATEMENTS;
[ELSIF CONDITION THEN
  STATEMENTS;]
[ELSE
  STATEMENTS;]
END IF;
```

Example

```
DECLARE
GRADE NUMBER(3) := 95;
RESULT VARCHAR2(1) ;
BEGIN
  IF GRADE >= 90 THEN
     RESULT := 'A' ;
  ELSIF GRADE >= 80 THEN
     RESULT := 'B' ;
  ELSIF GRADE >= 70 THEN
     RESULT := 'C' ;
  ELSIF GRADE >= 60 THEN
     RESULT := 'D' ;
  ELSE
     RESULT := 'F' ;
  END IF;
  DBMS_OUTPUT. PUT_LINE ('Result: ' || RESULT);
END;
```

## CASE Expression:

A CASE expression selects a result and returns it. To select the result, the CASE expression uses expressions. The value returned by these expressions is used to select one of several alternatives.

Syntax of CASE Expression

```
CASE SELECTOR
   WHEN EXPRESSION1 THEN RESULT1
   WHEN EXPRESSION2 THEN RESULT2
   ...
   WHEN EXPRESSIONN THEN RESULTN
   [ELSE RESULTN+1]
END;
```

Example

```
DECLARE
GRADE CHAR(1) := 'A' ;
APPRAISAL VARCHAR2(20);
BEGIN
APPRAISAL :=
  CASE GRADE
     WHEN 'A' THEN 'Excellent'
     WHEN 'B' THEN 'Very Good'
     WHEN 'C' THEN 'Good'
     ELSE 'No such grade'
  END;
  DBMS_OUTPUT.PUT_LINE ('Grade: '||GRADE||' - Appraisal '|| APPRAISAL);
END;
```

## Searched CASE Expressions:

In searched CASE statements, you do not have a test expression. Instead, the WHEN clause contains an expression that results in a Boolean value. The same example is rewritten in this slide to show searched CASE statements.

Example

```
DECLARE
GRADE CHAR(1) := 'B' ;
APPRAISAL VARCHAR2(20);
BEGIN
APPRAISAL :=
  CASE
     WHEN GRADE = 'A' THEN 'Excellent'
     WHEN GRADE IN ('B' , 'C' ) THEN 'Good'
     ELSE 'No such grade'
  END;
  DBMS_OUTPUT.PUT_LINE('Grade: ' || GRADE || ' - Appraisal ' ||APPRAISAL);
END;
```

## Loops:

### Basic Loops:

The syntax of cursor

```
LOOP
STATEMENT1;
...
EXIT [WHEN CONDITION];
END LOOP;
```

**Example:** write a PL/SQL code to print number from 1 to 15 to the console.

Example

```
DECLARE
I NUMBER(2) := 1;
BEGIN
  LOOP
     DBMS_OUTPUT. PUT_LINE (I);
     I := I + 1;
     EXIT WHEN I > 15;
  END LOOP;
END;
```

### WHILE Loops:

The syntax of while loops

```
WHILE CONDITION LOOP
  STATEMENT1;
  STATEMENT2;
  ...
END LOOP;
```

**Example:** write a PL/SQL code to print number from 1 to 15 to the console.

Example

```
DECLARE
I NUMBER(2) := 1;
BEGIN
  WHILE I <= 15 LOOP
  DBMS_OUTPUT. PUT_LINE (I);
  I := I + 1;
  END LOOP;
END;
```

## FOR Loops:

You can use a FOR loop to shortcut the test for the number of iterations. You do not have to declare the counter;it is declared implicitly. **'lower bound ..upper_bound'** is required syntax.

The syntax of for loop

```
FOR COUNTER IN [REVERSE] LOWER_BOUND.. UPPER_BOUND LOOP
  STATEMENT1;
  STATEMENT2;
  ...
END LOOP;
```

**Example:** write a PL/SQL code to print number from 1 to 15 to the console.

Example

```
DECLARE
BEGIN
  FOR I IN 1.. 15 LOOP
    DBMS_OUTPUT. PUT_LINE (I);
  END LOOP;
END;
```

## The "SELECT INTO" Clause:

It's used to retrieves data from one or more database tables, and assigns the selected values to variables .It used to retrieves one or more columns from **only one row**.

remainder_of_query contains the list of tables or views

```
SELECT select_list INTO variable_list FROM remainder_of_query;
```

```
DECLARE
  INST_Name VARCHAR2(20);
  INST_SAL NUMBER(8,2);
BEGIN
  SELECT NAME, SALARY
  INTO INST_Name, INST_SAL
  FROM INSTRUCTOR
  WHERE id = '12121';
  DBMS_OUTPUT. PUT_LINE(INST_Name || ' ' || INST_SAL);
END;
```

### The %TYPE Attribute:

The %TYPE attribute is used to declare a variable according to:

- A database column definition.
- Another declared variable.

It is a prefixed with:

- The database table and column.
- The name of the declared variable.

Use it to declare a new variable with the same data type of a predefined variable or a column in a table.

Use Conversion function

```
IDENTIFIER TABLE_NAME.COLUMN_NAME%TYPE;
```

Use Conversion function

```
DECLARE
  INST_NAME INSTRUCTOR.NAME%TYPE;
  BALANCE NUMBER(7, 2);
  MIN_BALANCE BALANCE%TYPE := 1000;
  STD_NAME    INST_NAME%TYPE;
BEGIN
...
END;
```

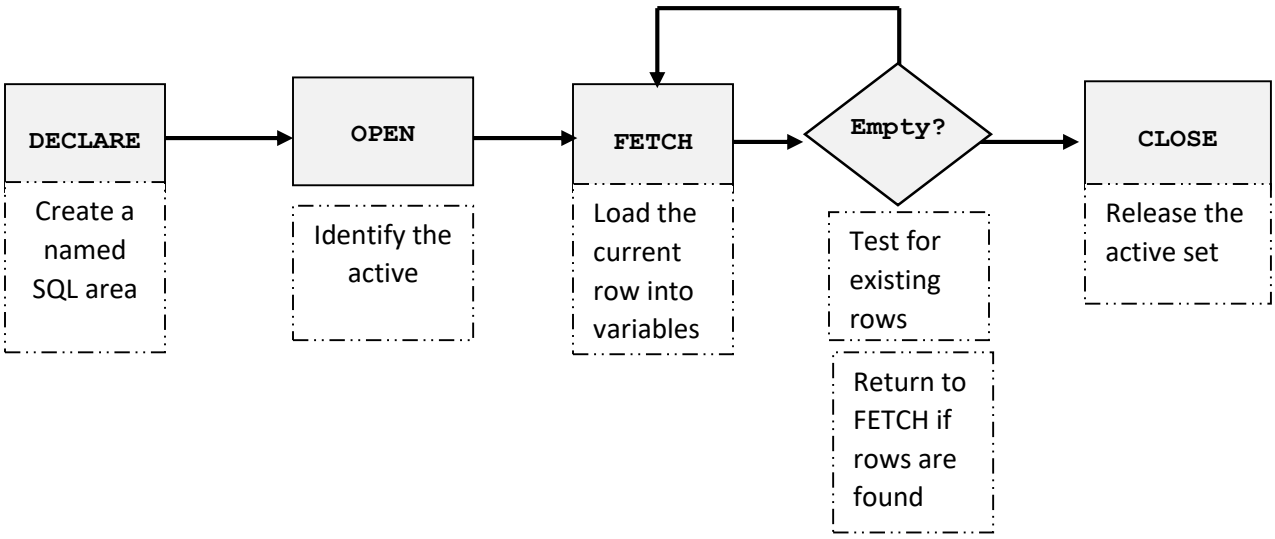
## Explicit Cursor:

A cursor is a SELECT statement that is defined within the declaration section of your PL/SQL code. It is created on a SELECT Statement which returns more than one row.

The syntax of cursor

```
Declare CURSOR C IS
  SELECT_STATEMENT;
BEGIN
  OPEN EMP_CURSOR;
  ...
  FETCH ...
  ...
  CLOSE EMP_CURSOR;
END;
```

**Controlling Cursors:**




**Explicit Cursor Attributes:**

Attributes that Obtain status information about a cursor. The following table illustrates explicit cursors you can use with cursors:

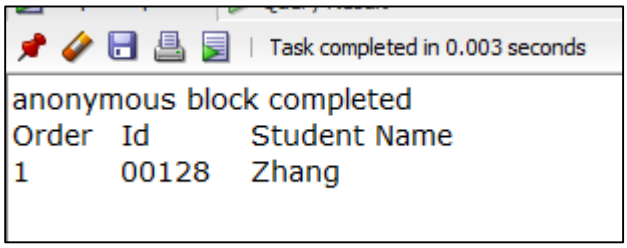
| Attribute | Type | Description |
|---|---|---|
| **%ISOPEN** | Boolean | Return TRUE if the cursor is open |
| **%NOTFOUND** | Boolean | Returns `FALSE` if the last fetch returned a row |
| **%FOUND** | Boolean | Returns `TRUE` if the last fetch returned a row |
| **%ROWCOUNT** | Number | Returns the number of rows fetched |

Any Explicit cursor attribute will be accessed as

**cursor_name%attribute_name** as shown below in the example.

Example

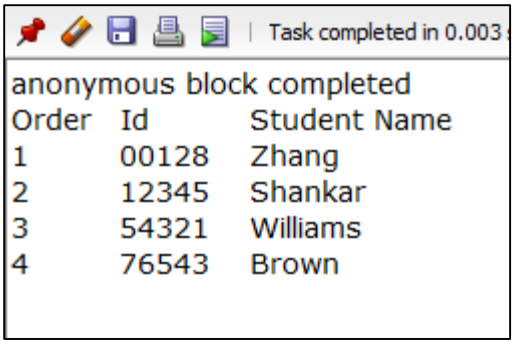
```
DECLARE
  CURSOR STD_CURSOR IS
  SELECT ID, NAME FROM STUDENT
  WHERE DEPT_NAME = 'Comp. Sci.';
  STD_NAME STUDENT.NAME%TYPE;
  STD_ID STUDENT.ID%TYPE;
BEGIN
  OPEN STD_CURSOR;
  DBMS_OUTPUT. PUT_LINE('Order   Id          Student Name ' );
  FETCH STD_CURSOR INTO STD_ID,STD_NAME;
  DBMS_OUTPUT. PUT_LINE(STD_CURSOR%ROWCOUNT || '          ' || STD_ID || '
' || STD_NAME);
  CLOSE STD_CURSOR;
END;
```




The previous code will print the first record returned from the query. You can use loop to print all records:

Example

```
DECLARE
  CURSOR STD_CURSOR IS
  SELECT ID, NAME FROM STUDENT
  WHERE DEPT_NAME = 'Comp. Sci.';
  STD_NAME STUDENT.NAME%TYPE;
  STD_ID STUDENT.ID%TYPE;
BEGIN
  OPEN STD_CURSOR;
  DBMS_OUTPUT. PUT_LINE('Order   Id          Student Name ' );
  IF STD_CURSOR%ISOPEN THEN
    LOOP
      FETCH STD_CURSOR INTO STD_ID,STD_NAME;
      EXIT WHEN STD_CURSOR%NOTFOUND;
      DBMS_OUTPUT. PUT_LINE(STD_CURSOR%ROWCOUNT || '          ' || STD_ID
|| '          ' || STD_NAME);
    END LOOP;
  END IF;
  CLOSE STD_CURSOR;
END;
```




Note that you can process the rows of the active set by fetching values into a PL/SQL record using **%ROWTYPE** Attribute:

**%ROWTYPE:**

The %ROWTYPE attribute provides a record type that represents a row in a database table, you can use the %ROWTYPE attribute in variable declarations as a datatype

Using ROWTYPE Attribute

```
DECLARE
  CURSOR STD_CURSOR IS
    SELECT ID, NAME FROM STUDENT
    WHERE DEPT_NAME = 'Comp. Sci.';
  STD_RECORD STD_CURSOR%ROWTYPE;
BEGIN
  OPEN STD_CURSOR;
  DBMS_OUTPUT. PUT_LINE('Order   Id          Student Name ' );
  IF STD_CURSOR%ISOPEN THEN
    LOOP
      FETCH STD_CURSOR INTO STD_RECORD;
      EXIT WHEN STD_CURSOR%NOTFOUND;
      DBMS_OUTPUT. PUT_LINE(STD_CURSOR%ROWCOUNT || '          ' ||
STD_RECORD.ID || '    ' || STD_RECORD.NAME);
   END LOOP;
  END IF;
  CLOSE STD_CURSOR;
END;
```

**Example:** Uses %ROWCOUNT to fetch the names with id and department name of the three highest-paid instructor.

Example use %rowcount

```
DECLARE
  CURSOR INST_CURSOR is
    SELECT id , name, dept_name FROM instructor
    ORDER BY salary DESC;   -- start with highest-paid instructor
  INST_RECORD     INST_CURSOR%ROWTYPE;
BEGIN
  OPEN INST_CURSOR;
    LOOP
      FETCH INST_CURSOR INTO INST_RECORD;
      EXIT WHEN (INST_CURSOR%ROWCOUNT > 3) OR (INST_CURSOR%NOTFOUND);
      dbms_output.put_line('Instructor ' || INST_RECORD.name || ' (' ||
INST_RECORD.id || ') work on  ' || INST_RECORD.dept_name);
    END LOOP;
  CLOSE INST_CURSOR;
END;
```

## Cursor's FOR Loops:

The cursor FOR LOOP statement implicitly declares its loop index as a record variable of the row type, and then opens a cursor. With each iteration, the cursor FOR LOOP statement fetches a row from the result set into the record

Cursor's FOR Loops

```
DECLARE
CURSOR STD_CURSOR IS
  SELECT ID, NAME FROM STUDENT
  WHERE DEPT_NAME = 'Comp. Sci.';
STD_RECORD STD_CURSOR%ROWTYPE;
BEGIN
  DBMS_OUTPUT. PUT_LINE('Order    Id          Student Name ' );
  FOR STD_RECORD IN STD_CURSOR  LOOP
    DBMS_OUTPUT. PUT_LINE(STD_CURSOR%ROWCOUNT || '          ' ||
STD_RECORD.ID || '     ' || STD_RECORD.NAME);
  END LOOP;
END;
```

## Explicit Cursor (SQL):

A SQL (implicit) cursor is automatically opened by the database to process each SQL statement is executed.

- 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.

| Attribute | Type | Description |
|---|---|---|
| **%ISOPEN** | Boolean | Always returns FALSE, since SQL cursor closed automatically after executing its associated SQL statement. |
| **%NOTFOUND** | Boolean | returns TRUE if an INSERT, UPDATE, or DELETE statement affected no rows, or a SELECTINTO statement returned no rows |
| **%FOUND** | Boolean | Returns TRUE if an INSERT, UPDATE, or DELETE statement affected one or more rows or a SELECT INTO statement returned one or more rows |
| **%ROWCOUNT** | Number | Returns the number of rows affected by an INSERT, UPDATE or DELETE statement, or returned by a SELECT INTO statement |

Any SQL cursor attribute will be accessed as **sql%attribute_name** as shown below in the example.

**Example:** Increase salary of each instructor in Computer Sciences by 5% and use the SQL%ROWCOUNT attribute to determine the number of rows affected:

Example

```
BEGIN
  UPDATE INSTRUCTOR SET SALARY =SALARY+ SALARY * .05
  WHERE DEPT_NAME='Comp. Sci.';

  IF SQL%FOUND THEN
  DBMS_OUTPUT.PUT_LINE(' THE NUMBER OF INSTRUCTOR, AFFECTED  BY RISING
'||  SQL%ROWCOUNT );
  END IF;
END;
```

**Note:** You can use cursor attributes in procedural statements but not in SQL statements

```
anonymous block completed
The number of instructor, affected  by rising  3
```

**END**