PL/SQL stands for **Procedural Language/Structured Query Language**. It’s Oracle Corporation’s procedural extension to SQL, and it's used primarily with Oracle databases

PL/SQL allows you to:

* Write **procedural code** (like loops, conditions, and variables) alongside SQL.
* Create **stored procedures**, **functions**, **packages**, and **triggers**.
* Handle **exceptions** (errors) gracefully.
* Improve performance by reducing the number of calls between applications and the database.

All OS supports it

Efficient

Exception handling

**1. Block Structure**

* PL/SQL programs are organized into **blocks** (Anonymous blocks, Procedures, Functions, Packages).
* Each block has **three sections**:
  + **Declaration** (optional): Variables, constants, cursors.
  + **Executable** (mandatory): SQL and PL/SQL statements.
  + **Exception Handling** (optional): Error handling logic.

**2. Tight Integration with SQL**

* PL/SQL supports **all SQL commands** (SELECT, INSERT, UPDATE, DELETE).
* Allows **embedding SQL inside procedural code** for better performance.

**3. Procedural Constructs**

* Supports **loops** (FOR, WHILE), **conditional statements** (IF-THEN-ELSE), and **modular programming** (procedures, functions).

**4. Exception Handling**

* Built-in mechanism to handle **runtime errors** using EXCEPTION block.
* Predefined exceptions (e.g., NO\_DATA\_FOUND, TOO\_MANY\_ROWS) and user-defined exceptions.

**5. Cursors**

* **Implicit cursors** for single-row queries.
* **Explicit cursors** for multi-row queries, giving fine control over fetching data.

**6. Packages**

* Group related procedures, functions, variables, and cursors into **packages** for better modularity and reusability.

**7. Triggers**

* PL/SQL supports **database triggers** that execute automatically on DML events (INSERT, UPDATE, DELETE).

**8. Portability**

* PL/SQL code runs on any Oracle database platform without modification.

**9. Security**

* Supports **privileges and roles** for secure execution.
* Code can be stored in the database, reducing exposure.

**10. Performance Features**

* **Bulk Collect** and **FORALL** for high-speed data processing.
* **Bind variables** to reduce parsing overhead.

Need for plsql is :

**✅ 1. SQL is Non-Procedural**

* SQL can **query and manipulate data**, but it **cannot handle procedural logic** like loops, conditions, or modular programming.
* PL/SQL adds **procedural capabilities** (IF-THEN-ELSE, loops, variables) to SQL.

**✅ 2. Better Performance**

* PL/SQL allows **sending blocks of code to the database in one go**, reducing **network traffic** compared to multiple SQL statements.
* Features like **Bulk Collect** and **FORALL** improve performance for large data operations.

**✅ 3. Error Handling**

* SQL has no built-in mechanism for handling runtime errors.
* PL/SQL provides **robust exception handling** to manage errors gracefully.

**✅ 4. Modularity and Reusability**

* PL/SQL supports **procedures, functions, and packages**, making code modular, reusable, and easier to maintain.

**✅ 5. Business Logic in Database**

* Instead of implementing business rules in the application layer, PL/SQL allows you to **embed business logic inside the database** for better security and consistency.

**✅ 6. Triggers and Automation**

* PL/SQL enables **triggers** that automatically execute on events like INSERT, UPDATE, DELETE, ensuring data integrity and automation.

**✅ 7. Security**

* PL/SQL programs can be stored in the database with **controlled access**, reducing exposure of SQL queries and sensitive logic.

Begin and end are compulsory statements

Declare and Exception are optional  
  
  
  
Code:

**BEGIN**

**DBMS\_OUTPUT.PUT\_LINE(‘Welcome to plsql’);**

**END;**

**/**

**Addition and subtraction code**

DECLARE

num1 NUMBER;

num2 NUMBER;

sum\_result NUMBER;

diff\_result NUMBER;

BEGIN

-- Accept user input via bind variables

num1 := :Enter\_the\_num1;

num2 := :Enter\_the\_num2;

-- Perform operations

sum\_result := num1 + num2;

diff\_result := num1 - num2;

-- Output results

DBMS\_OUTPUT.PUT\_LINE('Addition Result: ' || sum\_result);

DBMS\_OUTPUT.PUT\_LINE('Subtraction Result: ' || diff\_result);

END;

CODE 2

DECLARE

school\_name CONSTANT VARCHAR2(20) := 'DYP';

BEGIN

DBMS\_OUTPUT.PUT\_LINE('My college was : ' || school\_name);

END;

CODE 3 – if block

DECLARE

x int := 10;

y int := 20;

BEGIN

if (y > x) then

DBMS\_OUTPUT.PUT\_LINE('Result: ' || y || ' is greater than ' || x);

end if;

END;

**CODE if else block**

DECLARE

x INT := 20;

y INT := 10;

BEGIN

IF y > x THEN

DBMS\_OUTPUT.PUT\_LINE('Result: ' || y || ' is greater than ' || x);

ELSE

DBMS\_OUTPUT.PUT\_LINE('Result: ' || y || ' is not greater than ' || x);

END IF;

END;

**CODE odd/even**

DECLARE

num NUMBER;

BEGIN

-- Accept input from user

num := :Enter\_the\_number;

-- Check if number is even or odd

IF MOD(num, 2) = 0 THEN

DBMS\_OUTPUT.PUT\_LINE('The number ' || num || ' is EVEN.');

ELSE

DBMS\_OUTPUT.PUT\_LINE('The number ' || num || ' is ODD.');

END IF;

END;

**Loop Code**

BEGIN

  FOR i IN 1 .. 10 LOOP

    IF MOD(i, 2) = 0 THEN

      DBMS\_OUTPUT.PUT\_LINE(i || ' is EVEN');

    ELSE

      DBMS\_OUTPUT.PUT\_LINE(i || ' is ODD');

    END IF;

  END LOOP;

END;

**SWITCH CASE**

DECLARE

a INT := 2; -- Assign a value to 'a'

b INT;

BEGIN

b := MOD(a, 2); -- Calculate remainder when 'a' is divided by 2

CASE b

WHEN 0 THEN

DBMS\_OUTPUT.PUT\_LINE('Even Number');

WHEN 1 THEN

DBMS\_OUTPUT.PUT\_LINE('Odd Number');

END CASE;

END;

ANOTHER EXAMPLES USING SWITCH CASE

DECLARE

day\_num INT;

BEGIN

day\_num := :day\_num; -- Bind variable input

CASE day\_num

WHEN 1 THEN

DBMS\_OUTPUT.PUT\_LINE('Sunday');

WHEN 2 THEN

DBMS\_OUTPUT.PUT\_LINE('Monday');

WHEN 3 THEN

DBMS\_OUTPUT.PUT\_LINE('Tuesday');

WHEN 4 THEN

DBMS\_OUTPUT.PUT\_LINE('Wednesday');

WHEN 5 THEN

DBMS\_OUTPUT.PUT\_LINE('Thursday');

WHEN 6 THEN

DBMS\_OUTPUT.PUT\_LINE('Friday');

WHEN 7 THEN

DBMS\_OUTPUT.PUT\_LINE('Saturday');

ELSE

DBMS\_OUTPUT.PUT\_LINE('Invalid day number');

END CASE;

END;

CODE for loop

DECLARE

i INT;

BEGIN

i := 0; -- Bind variable input

LOOP

if i>10 then

exit;

end if;

DBMS\_OUTPUT.PUT\_LINE(i);

i := i+1;

END LOOP;

END;

CODE FOR tables

DECLARE

i INT;

n int;

BEGIN

i := 1;

n := :n;

LOOP

if i>10 then

exit;

end if;

DBMS\_OUTPUT.PUT\_LINE(i\*n);

i := i+1;

END LOOP;

END;

WHILE LOOP

DECLARE

i INT:=1;

BEGIN

WHILE (i <=10) LOOP

DBMS\_OUTPUT.PUT\_LINE(''||i);

i := i+2;

END LOOP;

END;

DECLARE                          -- Start of the declaration section of an anonymous PL/SQL block

    a NUMBER;                    -- Declare variable 'a' of type NUMBER (NULL by default)

    b NUMBER;                    -- Declare variable 'b' of type NUMBER

    c NUMBER;                    -- Declare variable 'c' of type NUMBER (will hold the result)

    -- Local function declaration and definition (a nested subprogram inside this block)

    FUNCTION findmax(X IN NUMBER, y IN NUMBER)  -- Function named 'findmax' with two IN parameters, returns NUMBER

    RETURN NUMBER

    IS

        z NUMBER;                -- Local variable inside the function to hold the computed result

    BEGIN                        -- Start of the function body

        IF X > y THEN            -- If X is greater than y

            z := y;              -- (Bug) Assign y to z (this is not "max" logic; both branches assign y)

        ELSE

            z := y;              -- Assign y to z

        END IF;                  -- End of IF statement

        RETURN z;                -- Return z from the function

    END;                         -- End of function 'findmax'

BEGIN                            -- Start of the executable section of the anonymous block

    a := 10;                     -- Assign 10 to variable 'a'

    b := 100;                    -- Assign 100 to variable 'b'

    c := findmax(a, b);          -- Call the function with a and b; assign the returned value to 'c'

    DBMS\_OUTPUT.PUT\_LINE(        -- Call Oracle's DBMS\_OUTPUT package to print a line to the output buffer

        'Maximum number is: ' || c

    );

END;                             -- End of the anonymous PL/SQL block

PROCEDURE

CREATE TABLE

CREATE TABLE STUDENT (

    STUDENT\_ID     NUMBER GENERATED BY DEFAULT AS IDENTITY PRIMARY KEY,

    FIRST\_NAME     VARCHAR2(50),

    LAST\_NAME      VARCHAR2(50),

    DOB            DATE,

    GENDER         CHAR(1),

    EMAIL          VARCHAR2(100),

    PHONE\_NUMBER   VARCHAR2(15),

    ENROLL\_DATE    DATE DEFAULT SYSDATE

);

TOTAL CODE FOR CURSOR - first table creation, then adding data, then cursor

=========================================================================

CREATE TABLE STUDENT (

STUDENT\_ID NUMBER GENERATED BY DEFAULT AS IDENTITY PRIMARY KEY,

NAME VARCHAR2(50),

COURSE VARCHAR2(50),

STUDNUM NUMBER

);

INSERT INTO STUDENT (NAME, COURSE, STUDNUM)

VALUES ('Ram Kumar', 'Computer Science', 101)

INSERT INTO STUDENT (NAME, COURSE, STUDNUM)

VALUES ('Anjali Mehta', 'Mechanical Engineering', 102);

DECLARE

a STUDENT.NAME%TYPE;

b STUDENT.COURSE%TYPE;

d STUDENT.STUDENT\_ID%TYPE;

CURSOR showRec(p\_id STUDENT.STUDENT\_ID%TYPE) IS

SELECT NAME, COURSE FROM STUDENT WHERE STUDENT\_ID = p\_id;

BEGIN

-- Replace 1 with the actual student ID you want to search

d := 1;

OPEN showRec(d);

LOOP

FETCH showRec INTO a, b;

EXIT WHEN showRec%NOTFOUND;

DBMS\_OUTPUT.PUT\_LINE('Name: ' || a || ', Course: ' || b);

END LOOP;

CLOSE showRec;

END;

/

Trigger full code

CREATE TABLE customer (

ID INT NOT NULL,

NAME VARCHAR2(40) NULL,

AGE INT NOT NULL,

ADDRESS VARCHAR2(40) NULL,

SALARY INT NULL,

PRIMARY KEY(ID));

INSERT INTO customer (ID, NAME, AGE, ADDRESS, SALARY)

VALUES (1, 'John Doe', 30, '123 Elm Street', 50000);

INSERT INTO customer (ID, NAME, AGE, ADDRESS, SALARY)

VALUES (2, 'Jane Smith', 25, '456 Oak Avenue', 60000);

INSERT INTO customer (ID, NAME, AGE, ADDRESS, SALARY)

VALUES (3, 'Mike Johnson', 40, '789 Pine Road', 75000);

INSERT INTO customer (ID, NAME, AGE, ADDRESS, SALARY)

VALUES (6, 'Amit Verma', 32, '12 Lotus Street', 52000);

INSERT INTO customer (ID, NAME, AGE, ADDRESS, SALARY)

VALUES (7, 'Priya Nair', 29, '88 Jasmine Avenue', 58000);

INSERT INTO customer (ID, NAME, AGE, ADDRESS, SALARY)

VALUES (8, 'Rahul Mehta', 45, '77 Rose Garden', 82000);

INSERT INTO customer (ID, NAME, AGE, ADDRESS, SALARY)

VALUES (9, 'Sneha Kulkarni', 38, '34 Tulip Lane', NULL); -- Salary is optional

INSERT INTO customer (ID, NAME, AGE, ADDRESS, SALARY)

VALUES (10, NULL, 50, '90 Orchid Road', 47000); -- Name is optional

SELECT \* FROM customer;

CREATE OR REPLACE TRIGGER Display\_salary\_change

BEFORE DELETE OR INSERT OR UPDATE ON customer

FOR EACH ROW

WHEN (NEW.ID > 0)

DECLARE

sal\_diff number;

BEGIN

sal\_diff := :NEW.SALARY - :OLD.SALARY;

dbms\_output.put\_line('Old salary: ' || :OLD.SALARY);

dbms\_output.put\_line('New salary: ' || :NEW.SALARY);

dbms\_output.put\_line('Difference in salary: ' || sal\_diff);

END;

UPDATE customer

SET SALARY = 55000

WHERE ID = 1;