────────────────────────────────────────────── **PL/SQL vs. SQL – Detailed Overview and Examples**

─────────────────── **1. PL/SQL vs. SQL**

**SQL (Structured Query Language):**

* **Nature:** Declarative language—you state *what* you want (e.g., retrieve or update data) without specifying the process.
* **Operations:** Works on sets of rows (set-based processing).
* **Control Structures:** Lacks traditional programming constructs (no loops, no conditionals).
* **Error Handling:** Does not have built-in exception handling.

**PL/SQL (Procedural Language/SQL):**

* **Nature:** Procedural language—you specify *how* the operations are performed using control statements like loops and conditionals.
* **Integration:** Integrates tightly with SQL, allowing you to embed SQL statements within procedural code.
* **Modularity:** Enables creation of procedures, functions, triggers, and packages for encapsulating and reusing code.
* **Error Handling:** Provides robust exception handling with try/catch-like blocks.

────────────────────────────────────────────── **2. Advantages of Using PL/SQL**

* **Performance:** Groups multiple SQL statements into one block, reducing network overhead.
* **Modularity and Reusability:** Code can be stored in the database as procedures or functions and reused by multiple applications.
* **Error Management:** Built-in exception handling allows for graceful error recovery.
* **Security:** Business logic can be encapsulated within stored procedures, limiting direct access to tables.
* **Maintainability:** Changes to business rules can be managed in one place rather than in multiple application layers.

────────────────────────────────────────────── **3. PL/SQL Block and Its Types**

A **PL/SQL block** is the basic unit of a PL/SQL program. It has three parts:

* **Declaration Section:** (Optional) Declare variables, constants, types, etc.
* **Execution Section:** (Mandatory) Contains the executable code.
* **Exception Section:** (Optional) Contains code to handle runtime errors.

**Types of PL/SQL Blocks:**

* **Anonymous Block:**
  + Has no name and is executed immediately.
  + Typically used for one-time scripts or testing.
* **Named Blocks:**
  + **Procedures:** Perform actions and can accept parameters but do not return a value.
  + **Functions:** Return a single value and can be used in SQL queries (subject to purity levels).
  + **Triggers:** Automatically executed when a specified database event occurs.
  + **Packages:** Groups related procedures, functions, variables, and cursors.

────────────────────────────────────────────── **4. Anonymous Block vs. Procedure vs. Function**

| **Feature** | **Anonymous Block** | **Procedure** | **Function** |
| --- | --- | --- | --- |
| **Name** | No name | Has a name | Has a name |
| **Storage** | Not stored in the database | Stored in the database as a schema object | Stored in the database as a schema object |
| **Return Value** | Does not return a value | Does not return a value (but can have OUT parameters) | Must return a value |
| **Invocation** | Executed immediately, once | Can be called repeatedly from applications or PL/SQL blocks | Can be called repeatedly and even used in SQL statements (if permitted) |
| **Usage Scenario** | One-time scripts or testing | Encapsulating actions or business logic | Calculations and returning results |

────────────────────────────────────────────── **5. PL/SQL Data Types**

* **Scalar Types:**
  + NUMBER – Numeric values.
  + VARCHAR2, CHAR – String data.
  + DATE, TIMESTAMP – Date and time values.
  + BOOLEAN – Logical values (only in PL/SQL).
* **Composite Types:**
  + **Records:** Group related items, similar to a row in a table.
  + **Collections:**
    - Associative Arrays (Index-by Tables)
    - Nested Tables
    - VARRAYs (Variable-size Arrays)
* **LOBs (Large Objects):**
  + BLOB – Binary data.
  + CLOB – Character large objects.
* **Reference Types:**
  + REF CURSOR – Pointers to query result sets.

────────────────────────────────────────────── **6. %TYPE and %ROWTYPE**

* **%TYPE:**
  + Allows a variable to inherit the data type of a table column or another variable.
  + **Example:**

DECLARE

v\_employee\_name employees.last\_name%TYPE;

BEGIN

-- v\_employee\_name now has the same data type as employees.last\_name.

END;

* **%ROWTYPE:**
  + Declares a record that represents an entire row from a table or cursor.
  + **Example:**

sql

Copy

DECLARE

v\_employee\_record employees%ROWTYPE;

BEGIN

-- v\_employee\_record now contains all the columns from the employees table.

END;

────────────────────────────────────────────── **7. Types of Loops in PL/SQL**

* **Basic Loop:**

sql

Copy

LOOP

-- Statements

EXIT WHEN condition;

END LOOP;

* **WHILE Loop:**

sql

Copy

WHILE condition LOOP

-- Statements

END LOOP;

* **FOR Loop:**

sql

Copy

FOR counter IN lower\_bound..upper\_bound LOOP

-- Statements

END LOOP;

*Note:* In a FOR loop, the loop variable is automatically declared and is read-only.

────────────────────────────────────────────── **8. Exceptions in PL/SQL**

An **exception** is an error condition that disrupts the normal flow of execution. PL/SQL handles exceptions using a dedicated exception section.

**Types of Exceptions:**

* **Predefined Exceptions:**
  + Examples: NO\_DATA\_FOUND, ZERO\_DIVIDE, TOO\_MANY\_ROWS
* **User-Defined Exceptions:**
  + Declared in the declaration section and raised explicitly with the RAISE statement.
* **System-Defined Exceptions:**
  + Handled internally by the PL/SQL engine (e.g., arithmetic overflow).

**Example Exception Handling:**

sql

Copy

BEGIN

-- Code that may raise an exception

EXCEPTION

WHEN NO\_DATA\_FOUND THEN

DBMS\_OUTPUT.PUT\_LINE('No data found!');

WHEN OTHERS THEN

DBMS\_OUTPUT.PUT\_LINE('An unexpected error occurred: ' || SQLERRM);

END;

────────────────────────────────────────────── **9. Cursors in PL/SQL**

A **cursor** is a pointer to the result set of a query, allowing row-by-row processing.

* **Implicit Cursors:**
  + Automatically created by Oracle for DML statements.
  + Best for single-row queries or simple operations.
* **Explicit Cursors:**
  + Declared explicitly by the programmer for handling multiple rows.
  + Provides full control (open, fetch, close).

**Example of an Explicit Cursor:**

sql

Copy

DECLARE

CURSOR emp\_cursor IS

SELECT employee\_id, last\_name FROM employees;

v\_employee\_id employees.employee\_id%TYPE;

v\_last\_name employees.last\_name%TYPE;

BEGIN

OPEN emp\_cursor;

LOOP

FETCH emp\_cursor INTO v\_employee\_id, v\_last\_name;

EXIT WHEN emp\_cursor%NOTFOUND;

DBMS\_OUTPUT.PUT\_LINE('Employee ID: ' || v\_employee\_id || ', Name: ' || v\_last\_name);

END LOOP;

CLOSE emp\_cursor;

END;

────────────────────────────────────────────── **10. Implicit vs. Explicit Cursors – Detailed Comparison**

| **Feature** | **Implicit Cursor** | **Explicit Cursor** |
| --- | --- | --- |
| **Declaration** | Created automatically by Oracle for DML. | Must be declared explicitly by the programmer. |
| **Usage** | Ideal for single-row queries. | Used when detailed row-by-row processing is needed. |
| **Control** | Limited (uses attributes like %FOUND). | Full control over cursor operations (open, fetch, close). |
| **Error Handling** | Managed automatically (less granular). | Offers detailed, row-specific error handling. |

────────────────────────────────────────────── **PL/SQL Program Examples**

Below are several sample PL/SQL programs. You can copy and paste each of these blocks into your Oracle environment or Word document for documentation and testing.

────────────────────────────────────────────── **I. Factorial of a Number**

DECLARE

n NUMBER := 5; -- Change this value as needed

fact NUMBER := 1;

BEGIN

FOR i IN 1..n LOOP

fact := fact \* i;

END LOOP;

DBMS\_OUTPUT.PUT\_LINE('Factorial of ' || n || ' is: ' || fact);

END;

/

────────────────────────────────────────────── **II. Check if a Number is Prime**

DECLARE

n NUMBER := 17; -- Number to check

is\_prime BOOLEAN := TRUE;

BEGIN

IF n <= 1 THEN

is\_prime := FALSE;

ELSE

-- Loop from 2 to the square root of n (rounded down)

FOR i IN 2..TRUNC(SQRT(n)) LOOP

IF MOD(n, i) = 0 THEN

is\_prime := FALSE;

EXIT;

END IF;

END LOOP;

END IF;

IF is\_prime THEN

DBMS\_OUTPUT.PUT\_LINE(n || ' is a prime number.');

ELSE

DBMS\_OUTPUT.PUT\_LINE(n || ' is not a prime number.');

END IF;

END;

/

────────────────────────────────────────────── **III. Fibonacci Series**

DECLARE

n NUMBER := 10; -- Number of terms to display

a NUMBER := 0;

b NUMBER := 1;

c NUMBER;

BEGIN

DBMS\_OUTPUT.PUT\_LINE('Fibonacci Series:');

DBMS\_OUTPUT.PUT\_LINE(a);

DBMS\_OUTPUT.PUT\_LINE(b);

FOR i IN 3..n LOOP

c := a + b;

DBMS\_OUTPUT.PUT\_LINE(c);

a := b;

b := c;

END LOOP;

END;

/

────────────────────────────────────────────── **IV. Check if a Number is Positive, Negative, or Zero**

DECLARE

num NUMBER := -5; -- Change this value to test different cases

BEGIN

IF num > 0 THEN

DBMS\_OUTPUT.PUT\_LINE('The number is positive.');

ELSIF num < 0 THEN

DBMS\_OUTPUT.PUT\_LINE('The number is negative.');

ELSE

DBMS\_OUTPUT.PUT\_LINE('The number is zero.');

END IF;

END;

/

────────────────────────────────────────────── **V. Check if a Number is Odd or Even**

DECLARE

num NUMBER := 7; -- Change this value to test different cases

BEGIN

IF MOD(num, 2) = 0 THEN

DBMS\_OUTPUT.PUT\_LINE(num || ' is even.');

ELSE

DBMS\_OUTPUT.PUT\_LINE(num || ' is odd.');

END IF;

END;

/

────────────────────────────────────────────── **VI. Function to Find Sum and Average of the First N Numbers**

*Create the function:*

CREATE OR REPLACE FUNCTION sum\_avg(n IN NUMBER)

RETURN VARCHAR2 AS

total NUMBER := 0;

avg\_val NUMBER;

BEGIN

FOR i IN 1..n LOOP

total := total + i;

END LOOP;

avg\_val := total / n;

RETURN 'Sum: ' || total || ', Average: ' || avg\_val;

END;

/

*Call the function in an anonymous block:*

sql

Copy

DECLARE

result VARCHAR2(100);

BEGIN

result := sum\_avg(10); -- Change the input as needed

DBMS\_OUTPUT.PUT\_LINE(result);

END;

/