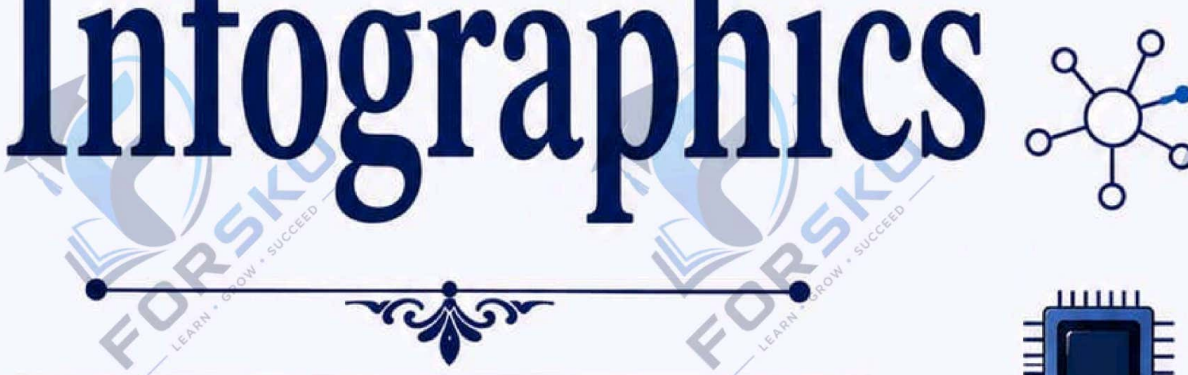
Infographics



Unit - III


DATA STRUCTURES
ALGORITHMS
COMPUTER NETWORKS
DATABASE SYSTEMS


#include <iostream>
using namespace std;
int main()
{
cout << "Hello, World!";
return 0;
}

INTRODUCTION TO PL/SQL

ORACLE PL/SQL

PL/SQL is Oracle's Procedural Language extension to SQL.

WHAT IS PL/SQL?

PL/SQL (Procedural Language/ Structured Query Language) is a high-level, block-structured language that integrates the power of SQL with the processing power of procedural programming.

For Data (Query Language) + For Logic (Procedural Language) → Powerful Database Programming Language

KEY FEATURES

- Block structured language.
- Combines SQL with procedural constructs.
- Supports variables, loops, conditions, exceptions and more.
- Increases application performance.
- Enhances security.
- Code reusability through stored program units.
- Handles complex business logic in the database.

ADVANTAGES OF PL/SQL

- Improved performance (reduced network traffic)
- Better security (code stays in the database)
- Modularity and reusability
- Easy maintenance
- Supports exception handling
- Reduces development time

PL/SQL IN THE ORACLE ENVIRONMENT

PL/SQL blocks are sent to the PL/SQL Engine, which processes the SQL statements and returns the result.

WHERE IS PL/SQL USED?

- Developing stored procedures
- Creating functions
- Writing triggers
- Building packages
- Developing applications with forms, reports and web interfaces
- Batch processing and automation

PL/SQL VS SQL

Feature	SQL	PL/SQL
Purpose	Data manipulation and retrieval	Process data and implement business logic
Type	Declarative Language	Procedural Language
Capability	Executes single SQL statements	Executes blocks of code with control structures
Error Handling	Limited	Robust (Exception Handling)
Use	Simple queries	Complex applications

EXAMPLE – SIMPLE PL/SQL BLOCK

```
BEGIN
  DBMS_OUTPUT.PUT_LINE('Hello PL/SQL!');
END;
/
```

Output:

```
Hello PL/SQL!
```

IN SHORT

PL/SQL is Oracle's powerful procedural extension to SQL that allows developers to write efficient, secure and modular programs to handle complex business logic within the database.

PL/SQL DATA TYPES

PL/SQL supports a rich set of data types to store different kinds of values in variables, constants, parameters and records.

1. SCALAR (SIMPLE) DATA TYPES

Store a single value.

Data Type	Description	Example	Declaration Example
NUMBER(p, s)	Stores numeric values. p = precision, s = scale	123.45	num NUMBER(5,2);
INTEGER	Stores whole numbers (no decimals).	100	i INTEGER;
BINARY_INTEGER	Stores binary integers. Range: -2,147,483,648 to 2,147,483,647	10101	b BINARY_INTEGER;
PLS_INTEGER	Stores signed integers. Faster than BINARY_INTEGER.	-500	p PLS_INTEGER;
BOOLEAN	Stores TRUE, FALSE or NULL.	TRUE	flag BOOLEAN;
CHAR(n)	Fixed length character string (n bytes).	'ABC'	ch CHAR(10);
VARCHAR2(n)	Variable length character string (max 32767 bytes).	'Hello'	name VARCHAR2(20);
NCHAR(n)	Fixed length Unicode character string.	N'ABC'	nch NCHAR(10);
NVARCHAR2(n)	Variable length Unicode character string.	N'Hello'	nname NVARCHAR2(20);
DATE	Stores date and time.	31-JUL-2025 14:30:00	d DATE;
TIMESTAMP	Stores date, time and fractional seconds.	31-JUL-2025 14:30:25.123	ts TIMESTAMP;

2. LOB (LARGE OBJECT) DATA TYPES

Store large amounts of data.

CLOB

Character Large Object. Stores large text data (up to 4 GB).

```
c CLOB;
```

BLOB

Binary Large Object. Stores binary data like images, audio, files (up to 4 GB).

```
b BLOB;
```

BFILE

Binary File. Stores a locator (pointer) to a binary file on the server.

```
f BFILE;
```

NCLOB

National Character Large Object. Stores large Unicode text data.

```
nc NCLOB;
```

3. COMPOSITE DATA TYPES

Store multiple values.

RECORD

Group of related items of possibly different data types.

```
TYPE emp_rec IS RECORD (
  id      NUMBER,
  name    VARCHAR2(20),
  sal     NUMBER(10,2)
);
```

TABLE (INDEX BY TABLE)

Collection of elements of the same data type indexed by PLS_INTEGER (associative array).

```
TYPE num_tab IS TABLE OF NUMBER
INDEX BY PLS_INTEGER;

nums num_tab;
```

4. REFERENCE DATA TYPES

Hold references (pointers).

REF CURSOR

Holds a pointer to the context area of a query.

```
rc SYS_REFCURSOR;
```

REF OBJECT

Holds a reference (OID) to an object in the database.

```
r REF my_object_type;
```

5. OTHER IMPORTANT POINTS

- PL/SQL is strongly typed.
- Variables must be declared before use.
- Default value for uninitialized variables is NULL.
- %TYPE and %ROWTYPE attributes can be used to declare variables/records based on table columns/rows.

Example using %TYPE

```
DECLARE
  v_name  emp.ename%TYPE;
  v_sal   emp.sal%TYPE;
BEGIN
    NULL;
END;
/
```

6. SPECIAL ATTRIBUTES

%TYPE → Declares a variable with the same data type as a table column.

```
v_empno  emp.empno%TYPE; -- same type as emp.empno
```

%ROWTYPE → Declares a record with the same structure as a table row.

```
v_emp  emp%ROWTYPE;      -- same structure as table emp
```

EXAMPLE – DECLARING VARIABLES

```
DECLARE
   v_id          NUMBER(5);
   v_name        VARCHAR2(30);
   v_join_date   DATE;
   v_active      BOOLEAN := TRUE;
   v_salary      NUMBER(10,2) := 0;

BEGIN
      v_id := 101;
      v_name := 'Aisha Khan';
      v_join_date := SYSDATE;
END;
/
```

NOTE

Choosing the right data type improves performance, saves memory and ensures data integrity.

BLOCK STRUCTURE OF PL/SQL

PL/SQL
ORACLE

A PL/SQL program is a collection of one or more SQL and PL/SQL statements grouped together in a logical block.

WHAT IS A PL/SQL BLOCK?

A PL/SQL block is the fundamental unit of execution in PL/SQL.
Every block must have the following structure:

1. **DECLARE** (Optional)
2. **BEGIN** (Mandatory)
3. **EXCEPTION** (Optional)
4. **END;** (Mandatory)

GENERAL SYNTAX

```
DECLARE
      -- Declaration section
BEGIN
      -- Executable section
EXCEPTION
      -- Exception handling section
END;
/
```

COMPONENTS OF A PL/SQL BLOCK

1 DECLARE SECTION (Optional)

- Used to declare variables, constants, cursors, user-defined exceptions, etc.
- It is executed only once when the block starts.

Can contain:

- Variables
- Constants
- Cursors
- User-defined exceptions
- Types, Subtypes, Records

Example:

```
DECLARE
   v_empno NUMBER(4);
   v_name VARCHAR2(20);
   v_salary NUMBER(10,2);
BEGIN
   -- executable statements
END;
/
```

2 BEGIN SECTION (Mandatory)

- Contains the executable statements.
- Executes the logic sequentially.
- At least one statement is required.

Example:

```
DECLARE
   v_num NUMBER := 10;
BEGIN
   v_num := v_num + 5;
   DBMS_OUTPUT.PUT_LINE
   ('Value: ' || v_num);
END;
/
```

3 EXCEPTION SECTION (Optional)

- Handles run-time errors that may occur during execution.
- If an exception occurs, control is transferred to this section.
- Can have multiple exception handlers.

Example:

```
EXCEPTION
   WHEN ZERO_DIVIDE THEN
      DBMS_OUTPUT.PUT_LINE
      ('Cannot divide by zero.');
   WHEN OTHERS THEN
      DBMS_OUTPUT.PUT_LINE
      ('Some other error
      occurred.');
```

4 END; SECTION (Mandatory)

- Marks the end of the PL/SQL block.
- **The semicolon (;)** is mandatory.
- **The slash (/)** is used to execute the block in SQL*Plus/SQLcl.

Example:

```
BEGIN
   NULL;
END;
/
```

Note:
/ is not part of the block. It is a SQL*Plus command to execute the **block**.

FLOW OF EXECUTION

Start

- **DECLARE** (Executed once) → Memory is allocated and variables, cursors, etc., are initialized.
- **BEGIN** (Execute statements sequentially) → All statements inside BEGIN...END are executed in order.
- **EXCEPTION** (If any error occurs) → If an error occurs in BEGIN section, control transfers here. If no error, this section is skipped.
- **END;** (Block Terminates) → Control comes out of the block and execution continues.

COMPLETE EXAMPLE

```
DECLARE
   v_a NUMBER := 10;
   v_b NUMBER := 0;
   v_result NUMBER;
BEGIN
   v_result := v_a / v_b;
   DBMS_OUTPUT.PUT_LINE('Result: ' || v_result);
EXCEPTION
   WHEN ZERO_DIVIDE THEN
      DBMS_OUTPUT.PUT_LINE('Division by zero is not allowed.');
   WHEN OTHERS THEN
      DBMS_OUTPUT.PUT_LINE('An error occurred.');
END;
/
```

OUTPUT

```
Division by zero is not allowed.
```

IMPORTANT POINTS

- **DECLARE** and **EXCEPTION** sections are optional, but **BEGIN** and **END;** are mandatory.
- A block must have at least one executable statement in the **BEGIN** section.
- If an exception occurs and is not handled, it is propagated to the calling environment.
- Blocks can be anonymous (not stored) or part of stored programs.

TYPES OF BLOCKS

- **Anonymous Block** – Not stored in the database.
- **Named Block** – Stored in the database as Procedures, Functions, Packages, Triggers.

EXCEPTION HANDLING IN PL/SQL

PL/SQL
ORACLE

Exception handling manages run-time errors and ensures normal termination of the program.

WHAT IS EXCEPTION?

An exception is an event that disrupts the normal flow of program execution. PL/SQL provides a way to handle such errors gracefully without terminating the program.

Examples:

- Division by zero
- No data found
- Too many rows returned
- Invalid value
- Constraint violation
- User-defined errors

SYNTAX

```
BEGIN
    -- Executable statements
EXCEPTION
   WHEN exception1 THEN
      -- Statements to handle exception1
   WHEN exception2 THEN
      -- Statements to handle exception2
   ...
   WHEN OTHERS THEN
      -- Statements to handle all other exceptions
END;
/
```

Key Points

- The EXCEPTION section is optional.
- It is executed only when an error occurs in the BEGIN section.
- If no exception occurs, the EXCEPTION part is skipped.
- WHEN OTHERS is used to handle any other unexpected errors.

TYPES OF EXCEPTIONS

1. PREDEFINED (SYSTEM) EXCEPTIONS

Raised automatically by Oracle.

Examples:

- ZERO_DIVIDE
- NO_DATA_FOUND
- TOO_MANY_ROWS
- DUP_VAL_ON_INDEX
- VALUE_ERROR
- INVALID_NUMBER
- LOGIN_DENIED

2. USER-DEFINED EXCEPTIONS

Created by the programmer using RAISE statement.

Example:

```
DECLARE
   my_exception EXCEPTION;
  -- Use my_exception in exception handling
```

COMMON PREDEFINED EXCEPTIONS

Exception Name	Occurs When
ZERO_DIVIDE	A number is divided by zero.
NO_DATA_FOUND	A SELECT INTO statement returns no rows.
TOO_MANY_ROWS	A SELECT INTO statement returns more than one row.
DUP_VAL_ON_INDEX	Duplicate value violates a UNIQUE constraint.
VALUE_ERROR	A value is of wrong data type or size.
INVALID_NUMBER	A string cannot be converted to a number.
LOGIN_DENIED	Invalid username or password.
STORAGE_ERROR	Insufficient memory for the operation.
CURSOR_ALREADY_OPEN	An attempt to open an already open cursor.
INVALID_CURSOR	An invalid cursor operation is attempted.
OTHERS	Catches all exceptions not explicitly handled.

EXAMPLE – HANDLING PREDEFINED EXCEPTION

```
DECLARE
   a NUMBER := 10;
   b NUMBER := 0;
   c NUMBER;
BEGIN
   c := a / b;
   DBMS_OUTPUT.PUT_LINE('Result: ' || c);
EXCEPTION
   WHEN ZERO_DIVIDE THEN
      DBMS_OUTPUT.PUT_LINE('Error: Division by zero is not allowed.');
END;
/
```

OUTPUT

```
Error: Division by zero is not allowed.
```

EXAMPLE – USING WHEN OTHERS

```
DECLARE
   v_num NUMBER;
BEGIN
   v_num := TO_NUMBER('ABC'); -- Invalid conversion
EXCEPTION
   WHEN VALUE_ERROR THEN
      DBMS_OUTPUT.PUT_LINE('Value error occurred.');
   WHEN OTHERS THEN
      DBMS_OUTPUT.PUT_LINE('Some other error occurred.');
END;
/
```

RAISE STATEMENT

Used to raise an exception manually.

RAISE predefined exception

```
RAISE ZERO_DIVIDE;
RAISE NO_DATA_FOUND;
...
```

RAISE user-defined exception

```
my_exception EXCEPTION;
...
RAISE my_exception;
```

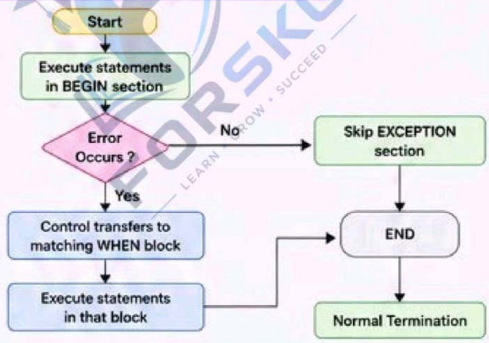
Example:

```
DECLARE
   invalid_age EXCEPTION;
   age NUMBER := -5;
BEGIN
   IF age < 0 THEN
      RAISE invalid_age;
   END IF;
EXCEPTION
   WHEN invalid_age THEN
      DBMS_OUTPUT.PUT_LINE('Age cannot be negative.');
END;
/
```

Note

RAISE initiates an exception. It immediately transfers control to the appropriate exception handler.

FLOW OF EXCEPTION HANDLING



BEST PRACTICES

- Handle specific exceptions first, then use WHEN OTHERS.
- Provide meaningful error messages to users.
- Do not ignore exceptions silently.
- Test exception handling thoroughly.
- Log errors for later analysis.

IMPORTANCE

- Prevents abnormal termination of the program.
- Improves reliability and robustness.
- Helps in debugging and maintenance.
- Provides user-friendly error messages.
- Ensures data integrity and consistency.

CURSOR IN PL/SQL

A cursor is a database object that allows you to retrieve and manipulate rows returned by a SQL query one row at a time.

PL/SQL

ORACLE

PURPOSE OF CURSOR

- Process multiple rows returned by a SQL query.
- Fetch and handle data row by row.
- Provide control over the processing of rows.

TYPES OF CURSOR

1 IMPLICIT CURSOR *(System-Defined Cursor)*

Automatically created and managed by PL/SQL for DML statements (INSERT, UPDATE, DELETE, SELECT INTO).

FEATURES

- Created automatically.
- Cannot be explicitly opened.
- Used for single-row operations.
- Use WHERE clause to restrict multiple rows.
- Attributes: SQL%FOUND, SQL%NOTFOUND, SQL%ROWCOUNT, SQL%ISOPEN.

EXAMPLE

```
UPDATE emp SET sal = sal + 1000
WHERE deptno = 10;

IF SQL%FOUND THEN
   DBMS_OUTPUT.PUT_LINE('Record updated');
END IF;

DBMS_OUTPUT.PUT_LINE('Rows affected: ' ||
SQL%ROWCOUNT);
```

2 EXPLICIT CURSOR *(User-Defined Cursor)*

Declared, opened, fetched, and closed explicitly by the programmer to process multiple rows returned by a SELECT statement.

FEATURES

- Must be declared in the DECLARE section.
- Must be opened, fetched, and closed explicitly.
- Used for multi-row operations.
- Gives more control to the programmer.

EXAMPLE

```
DECLARE
   CURSOR c_emp IS SELECT empno, ename FROM emp;
   v_empno emp.empno%TYPE;
   v_ename emp.ename%TYPE;
BEGIN
   OPEN c_emp;
   LOOP
      FETCH c_emp INTO v_empno, v_ename;
      EXIT WHEN c_emp%NOTFOUND;
      DBMS_OUTPUT.PUT_LINE(v_empno || ' - ' || v_ename);
   END LOOP;
   CLOSE c_emp;
END;
/
```

3 REF CURSOR *(Reference Cursor)*

A pointer to a result set. The result set can be opened and passed as a parameter to a procedure or function.

FEATURES

- Uses the REF CURSOR data type.
- Can return multiple rows.
- Useful for dynamic SQL and returning result sets from procedures/functions.
- Must be opened and closed.

EXAMPLE

```
DECLARE
   TYPE emp_cur IS REF CURSOR;
   c_emp emp_cur;
   v_emp emp%ROWTYPE;
BEGIN
   OPEN c_emp FOR SELECT * FROM emp;
   LOOP
      FETCH c_emp INTO v_emp;
      EXIT WHEN c_emp%NOTFOUND;
      DBMS_OUTPUT.PUT_LINE(v_emp.ename);
   END LOOP;
   CLOSE c_emp;
END;
/
```

4 DYNAMIC CURSOR *(Dynamic SQL Cursor)*

Used with dynamic SQL statements where the query is not known until runtime.

FEATURES

- Uses EXECUTE IMMEDIATE or DBMS_SQL package.
- The query is built and executed at runtime.
- Useful for flexible applications.

EXAMPLE

```
DECLARE
   TYPE ref_cur IS REF CURSOR;
   c ref_cur;
   v_query VARCHAR2(200);
BEGIN
   v_query := 'SELECT ename FROM emp WHERE sal > 3000';
   OPEN c FOR v_query;
   LOOP
      FETCH c INTO v_query; -- Example placeholder
      EXIT WHEN c%NOTFOUND;
   END LOOP;
   CLOSE c;
END;
/
```

COMPARISON OF CURSOR TYPES

Type	Created By	Open/Close	Used For	Rows
Implicit Cursor	PL/SQL (System)	Automatic	DML, Single-row operations	Single row
Explicit Cursor	Programmer	Manual	SELECT, Multi-row operations	Multiple rows
Ref Cursor	Programmer	Manual	Returning result sets, procedures/functions	Multiple rows
Dynamic Cursor	Programmer	Manual	Dynamic SQL (statements at runtime)	Multiple rows

KEY POINTS

- Cursors allow row-by-row processing of query results.
- Choose the right cursor type based on the requirement.
- Implicit cursor for simple DML.
- Explicit cursor for controlled multi-row processing.
- Ref cursor for returning result sets.
- Dynamic cursor for runtime queries.

CURSOR ATTRIBUTES IN PL/SQL

PL/SQL

ORACLE

Cursor attributes provide information about the status of a cursor during its operation.

WHAT ARE CURSOR ATTRIBUTES?

Cursor attributes are predefined properties associated with a cursor.

They return values that describe the state or result of cursor operations.

Cursor Operation Flow

STANDARD CURSOR ATTRIBUTES

Attribute	Data Type	Description	Value / Use	When to Use	Example
%ISOPEN	BOOLEAN	TRUE if the cursor is open.	TRUE – Cursor is open FALSE – Cursor is closed	To check whether a cursor is currently open before performing operations.	`IF c_emp%ISOPEN THEN CLOSE c_emp; END IF;`
%FOUND	BOOLEAN	TRUE if the most recent FETCH retrieves a row.	TRUE – Row fetched FALSE – No row fetched	To check if the FETCH statement was successful.	`FETCH c_emp INTO v_emp; IF c_emp%FOUND THEN -- process the row END IF;`
%NOTFOUND	BOOLEAN	TRUE if the most recent FETCH does not retrieve any row.	TRUE – No row fetched FALSE – Row fetched	To detect the end of result set or when no row is returned.	`FETCH c_emp INTO v_emp; EXIT WHEN c_emp%NOTFOUND;`
%ROWCOUNT	NUMBER	Returns the number of rows affected by DML operations or fetched so far.	0, 1, 2, ... n (Number of rows)	• After DML: rows affected • After FETCH: total rows fetched so far	`UPDATE emp SET sal = sal + 500 WHERE deptno = 10; DBMS_OUTPUT.PUT_LINE ('Rows updated: ' || c_emp%ROWCOUNT);`
%BULK_ROWCOUNT	PLS_INTEGER TABLE	Returns the number of rows affected by each iteration in FORALL statement.	Collection of numbers (one for each iteration)	With FORALL to know rows affected for each iteration.	`FORALL i IN 1..emp_ids.COUNT UPDATE emp SET sal = sal + 100 WHERE empno = emp_ids(i); DBMS_OUTPUT.PUT_LINE (c%BULK_ROWCOUNT(i));`
%BULK_EXCEPTIONS	PLS_INTEGER TABLE	Provides details of errors for each iteration in FORALL when SAVE EXCEPTIONS is used.	Collection of error information (index & error code)	With FORALL ... SAVE EXCEPTIONS to handle row-level errors.	`EXCEPTION WHEN OTHERS THEN FOR i IN 1..SQL%BULK_EXCEPTIONS. COUNT LOOP DBMS_OUTPUT.PUT_LINE (SQL%BULK_EXCEPTIONS(i).ERROR_INDEX || ' - ' || SQL%BULK_EXCEPTIONS(i).ERROR_CODE); END LOOP;`

NOTES

- %ISOPEN, %FOUND and %NOTFOUND are mainly used with FETCH operations.
- %ROWCOUNT is commonly used with DML statements and also with cursor FETCH.
- Bulk attributes (%BULK_ROWCOUNT and %BULK_EXCEPTIONS) are used with bulk processing (FORALL).
- Attributes are read-only and updated automatically by PL/SQL engine.

QUICK EXAMPLE

```
DECLARE
  CURSOR c_emp IS SELECT empno, ename FROM emp WHERE deptno = 10;
   v_emp c_emp%ROWTYPE;
BEGIN
  OPEN c_emp;
  DBMS_OUTPUT.PUT_LINE('Is Open : ' || c_emp%ISOPEN);
  LOOP
    FETCH c_emp INTO v_emp;
    EXIT WHEN c_emp%NOTFOUND;
    DBMS_OUTPUT.PUT_LINE('Emp: ' || v_emp.ename);
    DBMS_OUTPUT.PUT_LINE('Rows fetched so far: ' || c_emp%ROWCOUNT);
  END LOOP;
  CLOSE c_emp;
  DBMS_OUTPUT.PUT_LINE('Is Open after close : ' || c_emp%ISOPEN);
END;
/
```

OUTPUT (Example)

```
Is Open : TRUE
Emp: KING
Rows fetched so far: 1
Emp: CLARK
Rows fetched so far: 2
Emp: MILLER
Rows fetched so far: 3
Is Open after close : FALSE
```

KEY POINTS

- Cursor attributes give real-time information about cursor status.
- They help in writing efficient and error-free PL/SQL programs.
- Always check %NOTFOUND to exit the FETCH loop.
- Use %ROWCOUNT to know the impact of DML operations.

AT A GLANCE

Attribute	Tells You...
%ISOPEN	Whether cursor is open or not
%FOUND	If the last FETCH returned a row
%NOTFOUND	If the last FETCH returned no row
%ROWCOUNT	Number of rows affected or fetched
%BULK_ROWCOUNT	Rows affected in each FORALL iteration
%BULK_EXCEPTIONS	Error details in FORALL with SAVE EXCEPTIONS

TIP

Use cursor attributes wisely to control cursor flow and handle data efficiently!

LIFE CYCLE OF A CURSOR IN PL/SQL

A cursor goes through a series of states from creation to closing. These steps are executed in order.

PL/SQL

ORACLE

WHAT IS A CURSOR?

A cursor is a database object that allows you to retrieve and manipulate rows returned by a SQL query, one row at a time.

CURSOR LIFE CYCLE OVERVIEW

DECLARE (Define Cursor) → OPEN (Execute Query) → FETCH (Retrieve Rows) → PROCESS (Use Data) → CLOSE (Release Cursor)

STEPS IN THE LIFE CYCLE

1 DECLARE (Define Cursor)

- The cursor is declared in the DECLARE section.
- It defines the SQL query but does not execute it.

2 OPEN (Execute Query)

- The cursor is opened using the OPEN statement.
- The SQL query associated with the cursor is executed.

After OPEN, the cursor is positioned before the first row of the result set.

3 FETCH (Retrieve Rows)

- FETCH retrieves rows from the result set one at a time.
- After each FETCH, the cursor moves to the next row.

When no more rows are available, %NOTFOUND becomes TRUE.

4 PROCESS (Use Data)

- The fetched row is processed in the PL/SQL block.
- This step can be repeated for all rows using a loop.

Processing continues until all rows are fetched.

5 CLOSE (Release Cursor)

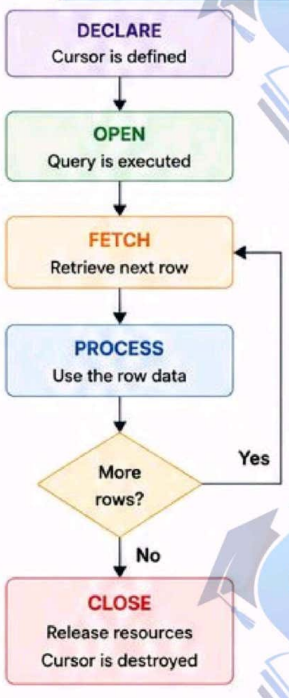
- The cursor is closed using the CLOSE statement.
- It releases the resources (memory, locks) held by the cursor.

Aftet CLOSE, the cursor becomes invalid loss cannot be used until reopened.

EXAMPLE – EXPLICIT CURSOR LIFE CYCLE

```
DECLARE
   CURSOR c_emp IS SELECT empno, ename FROM emp;
   v_empno   emp.empno%TYPE;
   v_ename   emp.ename%TYPE;
BEGIN
   OPEN c_emp;                                  -- Step 2: OPEN
 LOOP
    FETCH c_emp INTO v_empno, v_ename;          -- Step 3: FETCH
    EXIT WHEN c_emp%NOTFOUND;  -- No more rows
                                                -- Step 4: PROCESS
    DBMS_OUTPUT.PUT_LINE('Emp: ' || v_empno || ' - ' || v_ename);
END LOOP;
   CLOSE c_emp;                                 -- Step 5: CLOSE
END;
/
```

DETAILED FLOW DIAGRAM



DECLARE
Cursor structure and the query are defined.

OPEN
The query is sent to the database. Result set is created.

FETCH
The next row is retrieved into variables.
Cursor pointer moves forward.

PROCESS
Business logic is applied to the fetched data.

CLOSE
Result set and resources are released.
Cursor is no longer active.

CURSOR STATES

- **DECLARED** – Cursor is defined.
- **OPEN** – Cursor is open and active.
- **FETCHING** – Rows are being retrieved.
- **CLOSED** – Cursor is closed.

IMPORTANT NOTES

- Always OPEN a cursor before FETCH.
- Always CLOSE a cursor after use.
- Use EXIT WHEN cursor%NOTFOUND to end the fetch loop.
- Not closing a cursor may lead to memory loss and performance issues.

CURSOR ATTRIBUTES USED IN LIFE CYCLE

Attribute	Meaning
%ISOPEN	TRUE if cursor is open
%FOUND	TRUE if the last FETCH returned a row
%NOTFOUND	TRUE if the last FETCH did not return a row
%ROWCOUNT	Number of rows fetched so far

KEY BENEFITS

- Efficient row-by-row processing of query results.
- Provides control over data manipulation.
- Useful for complex processing and validations.
- Improves performance for large result sets.

REMEMBER

DECLARE → OPEN → FETCH → PROCESS → CLOSE

Follow this sequence for safe and effective cursor handling in PL/SQL.