KARNAVATI UNIVERSITY | UIT

Unitedworld Institute Of Technology

B.Tech. Computer Science & Engineering

Semester : 3rd

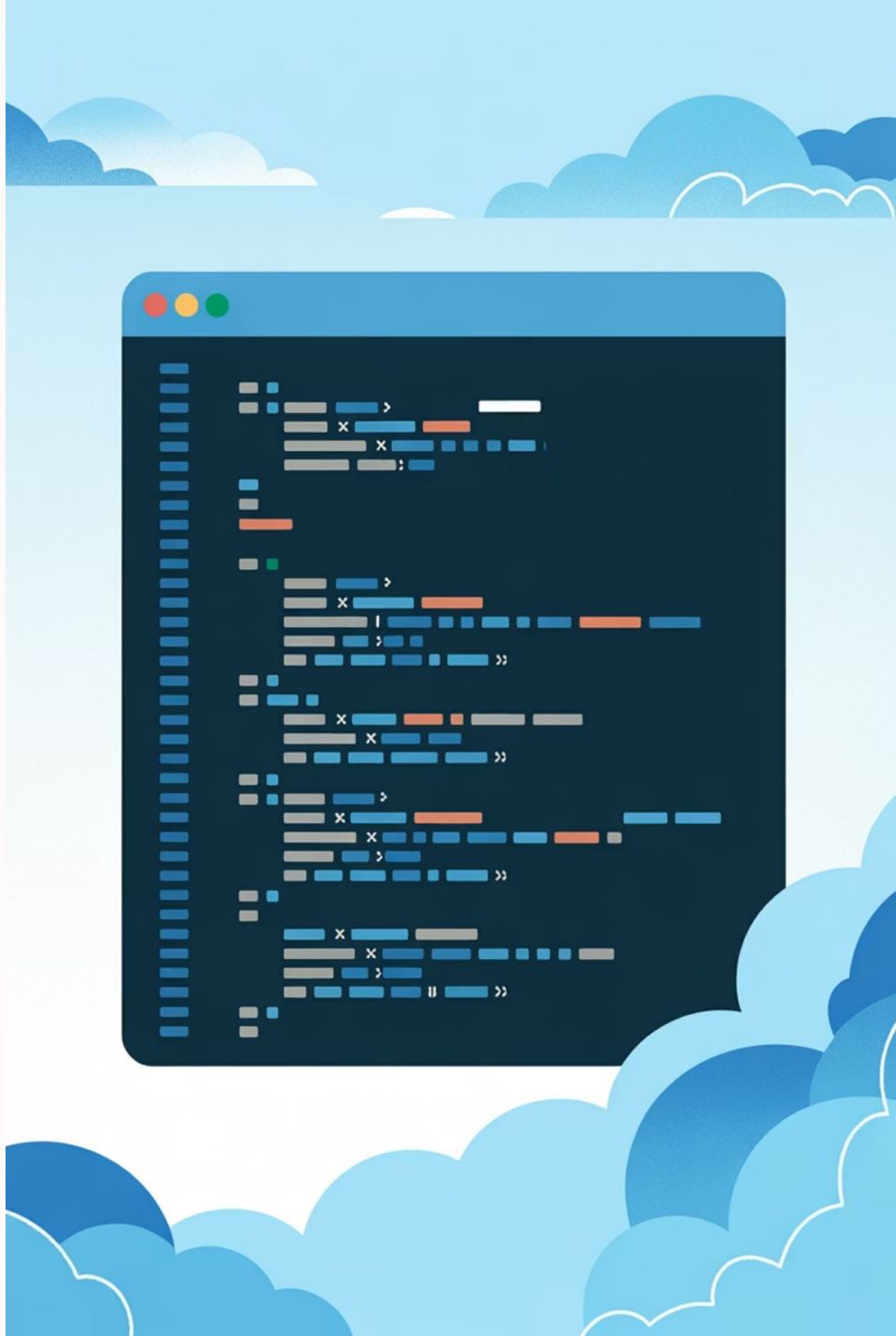
Introduction To Database Management System
Course Code: 71203002003

Unit – 3 : STRUCTURED QUERY LANGUAGE - SQL AND PL/SQL

Prepared by:
Mr. Utsav Kapadiya
Assistant Professor (UIT)

PL/SQL Concepts – Cursors in Detail

A comprehensive guide to understanding and implementing cursors in PL/SQL for effective database operations and row-by-row data processing.



What is a Cursor?

In PL/SQL, a cursor is a pointer or handle that allows you to fetch and process query results row by row. It acts as a critical bridge between SQL query results and your PL/SQL program logic.

When you execute a SELECT statement in PL/SQL that returns multiple rows, you cannot directly process them like you would in standard SQL. The cursor provides the mechanism to iterate through each row systematically, giving you precise control over data processing.

Think of a cursor as a navigation tool that points to one row at a time in your result set, allowing you to perform operations on each record individually before moving to the next one.

Formal Definition

Cursor Definition

A cursor is a temporary work area created in memory when a SQL statement is executed. This work area stores information about the rows returned by the query, including metadata and the current position in the result set.
The cursor maintains context information that allows you to retrieve one row at a time, process it, and then move to the next row efficiently.

Why Do We Need Cursors?

SQL Behavior

```
SELECT * FROM student;
```

In standard SQL, this query retrieves all rows at once and displays them immediately. This works perfectly for viewing data but doesn't allow for individual row manipulation.

PL/SQL Requirements

In PL/SQL, you often need to:

- Process each student's record individually
- Calculate grades or apply business logic
- Award bonus marks based on conditions
- Perform complex transformations
- Make decisions based on each row's data

The Solution

We use a cursor to fetch rows one by one, enabling row-level processing with full programmatic control. This gives us the flexibility to apply logic, perform calculations, and make decisions for each record.

Types of Cursors

Implicit Cursor

Automatically created by Oracle for simple queries like SELECT INTO, INSERT, UPDATE, and DELETE statements.
You don't need to declare or manage these cursors—Oracle handles everything behind the scenes, making them ideal for single-row operations.

Explicit Cursor

Declared and controlled manually by the programmer for multi-row SELECT queries that require iteration.
These give you complete control over opening, fetching, and closing operations, perfect for complex data processing scenarios.

Implicit Cursors

Understanding Oracle's automatic cursor management for single-row operations and DML statements.

Implicit Cursors Explained

Definition

An implicit cursor is automatically created by Oracle whenever you execute a SQL statement that does not return multiple rows requiring iteration.

These cursors are created, opened, fetched from, and closed automatically by Oracle without any explicit commands from the programmer. They're designed for simplicity and efficiency in straightforward database operations.

1

Oracle Creates Cursor

System automatically allocates memory

2

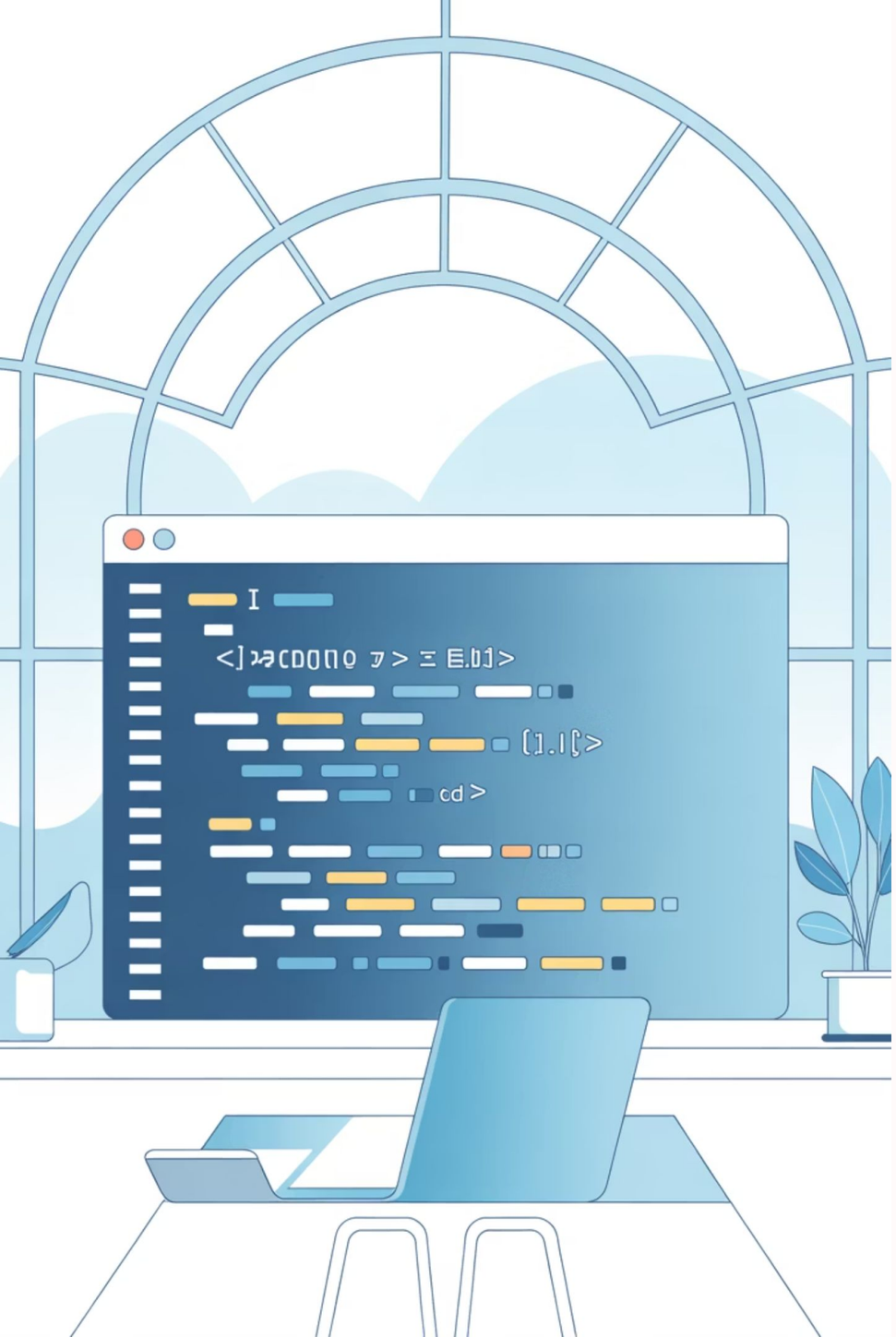
Executes Statement

Runs your SQL query or DML

3

Closes Automatically

Frees resources immediately



Example 1: Implicit Cursor with SELECT INTO

```
DECLARE  v_name student.name%TYPE;  v_marks student.marks%TYPE;BEGIN
SELECT name, marks INTO v_name, v_marks  FROM student  WHERE id = 101;
DBMS_OUTPUT.PUT_LINE('Name: ' || v_name ||                          ' Marks:
' || v_marks);END;/
```

Key Point: Oracle internally creates an implicit cursor for the SELECT INTO statement. If the query returns no rows, it raises a NO_DATA_FOUND exception. If it returns more than one row, it raises a TOO_MANY_ROWS exception. This automatic error handling ensures data integrity.

Example 2: Implicit Cursor with UPDATE

```
BEGIN  UPDATE student  SET marks = marks + 5  WHERE dept = 'CSE';    IF SQL%FOUND THEN    DBMS_OUTPUT.PUT_LINE('Some rows
updated.');  END IF;    IF SQL%NOTFOUND THEN    DBMS_OUTPUT.PUT_LINE('No rows updated.');  END IF;    DBMS_OUTPUT.PUT_LINE('Total
rows affected: ' ||                        SQL%ROWCOUNT);END;/
```

In this example, Oracle uses an implicit cursor automatically for the UPDATE statement. The cursor provides built-in attributes that allow you to check the operation's success and determine how many rows were affected. These attributes give you valuable feedback about your DML operations without requiring explicit cursor management.

Implicit Cursor Attributes

Oracle provides special attributes to query the status of the most recently executed implicit cursor. These attributes are essential for error checking and flow control.

SQL%FOUND

Returns TRUE if at least one row was affected by the SQL statement. Useful for confirming successful operations.

SQL%NOTFOUND

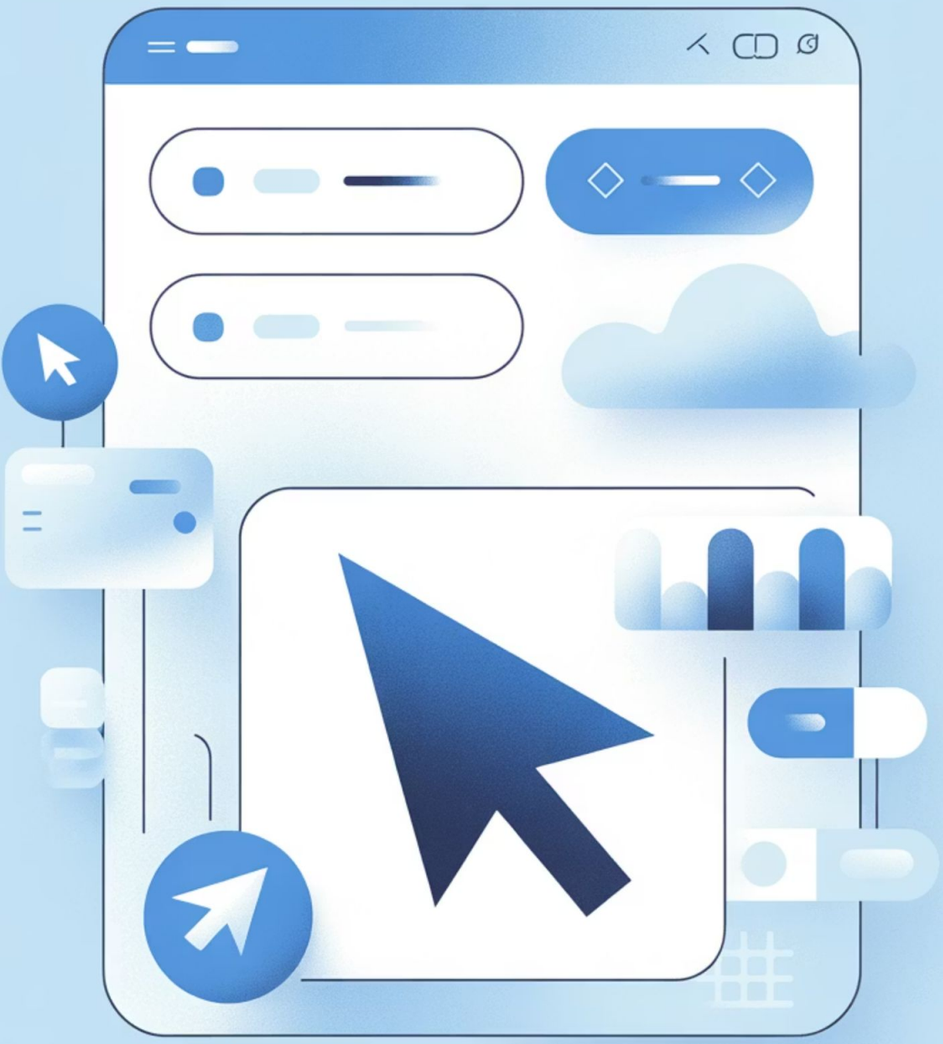
Returns TRUE if no rows were affected by the SQL statement. Helpful for detecting when operations had no impact.

SQL%ROWCOUNT

Returns the exact number of rows affected by the most recent SQL statement. Essential for logging and verification.

SQL%ISOPEN

Always returns FALSE for implicit cursors because Oracle automatically closes them immediately after execution.



Explicit Cursors

Taking full control of multi-row query processing with programmer-defined cursors.

Explicit Cursors Explained

Definition

An **explicit cursor** is manually declared and managed by the programmer. It provides complete control over the cursor lifecycle and is specifically designed for queries that return multiple rows requiring individual processing.

Unlike implicit cursors, you must explicitly declare, open, fetch from, and close these cursors. This gives you precise control over memory usage and data processing flow.

Steps to Use an Explicit Cursor

Step 1: Declare

Define the cursor with a name and associated SELECT query in the declaration section.

```
CURSOR c1 IS   SELECT name, marks FROM student;
```

Step 2: Open

Execute the query and establish the result set in memory.

```
OPEN c1;
```

Step 3: Fetch

Retrieve rows one at a time from the result set into variables.

```
FETCH c1 INTO v_name, v_marks;
```

Step 4: Close

Release the cursor and free associated memory resources.

```
CLOSE c1;
```

Complete Explicit Cursor Example

```
DECLARE  v_name student.name%TYPE;  v_marks student.marks%TYPE;    CURSOR c_student IS    SELECT name, marks FROM student    WHERE dept = 'CSE';BEGIN  OPEN c_student;  --
Step 1: Open cursor    LOOP    FETCH c_student INTO v_name, v_marks;  -- Step 2: Fetch    EXIT WHEN c_student%NOTFOUND;          -- Step 3: Exit check
DBMS_OUTPUT.PUT_LINE('Name: ' || v_name ||                          ' | Marks: ' || v_marks);  END LOOP;    CLOSE c_student;  -- Step 4: Close cursorEND;/
```

How It Works

- The cursor fetches one student record at a time from the CSE department
- Each iteration processes a single row

When no more rows are available, `c_student%NOTFOUND` becomes TRUE

- The loop exits and the cursor is properly closed

Explicit Cursor Attributes

Just like implicit cursors, explicit cursors provide attributes to monitor their status. These attributes use the cursor name as a prefix.

cursor_name%FOUND

Returns TRUE if the most recent FETCH operation successfully retrieved a row. Useful for confirming data availability.

cursor_name%NOTFOUND

Returns TRUE if the most recent FETCH found no more rows. Essential for loop termination logic.

cursor_name%ROWCOUNT

Returns the total number of rows fetched so far. Perfect for progress tracking and logging.

cursor_name%ISOPEN

Returns TRUE if the cursor is currently open. Prevents errors from double-opening or closing already-closed cursors.

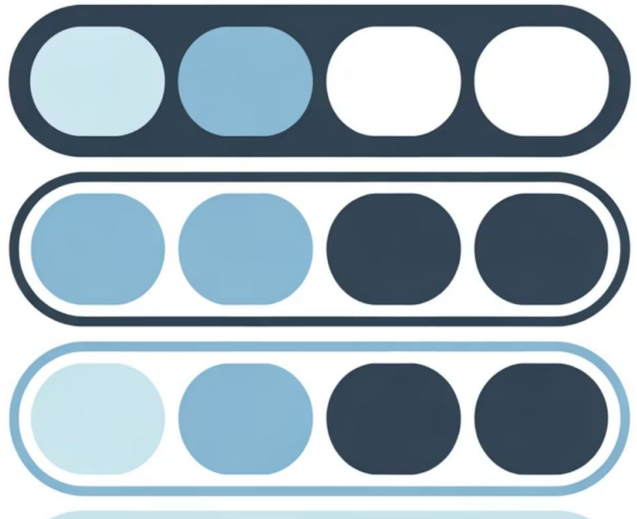
Example with Cursor Attributes

```
DECLARE  v_name student.name%TYPE;  v_marks student.marks%TYPE;    CURSOR c_student IS    SELECT name, marks FROM student;BEGIN  OPEN c_student;    LOOP    FETCH c_student INTO v_name, v_marks;    EXIT WHEN c_student%NOTFOUND;        DBMS_OUTPUT.PUT_LINE('Row ' ||
c_student%ROWCOUNT ||                         ': ' || v_name || ' - ' || v_marks);  END LOOP;    CLOSE c_student;END;/
```

What This Demonstrates

This example shows how `%ROWCOUNT` tracks the number of rows processed, giving you a running count during iteration. This is valuable for:

- Progress monitoring
- Debugging and logging
- Performance analysis



Cursor FOR Loop: The Simplified Approach

PL/SQL provides an elegant shortcut for explicit cursors that eliminates the need to manually open, fetch, or close the cursor. The Cursor FOR Loop handles all cursor management automatically.

```
BEGIN  FOR rec IN (SELECT name, marks FROM student              WHERE dept = 'CSE')   LOOP    DBMS_OUTPUT.PUT_LINE('Name: ' || rec.name ||
', Marks: ' || rec.marks);  END LOOP;END;/
```

What Oracle Does Automatically

- Opens the cursor before the first iteration
- Fetches each record into the loop variable
- Closes the cursor when the loop completes or exits

Why Use This Approach

This is the most common and easiest way to use cursors in practice. It reduces code complexity, eliminates common errors like forgetting to close cursors, and improves code readability significantly.

Advantages of Cursors

Row-by-Row Processing

Enables detailed processing of each individual record with custom logic, perfect for complex business rules that vary by row.

Multi-Row Query Handling

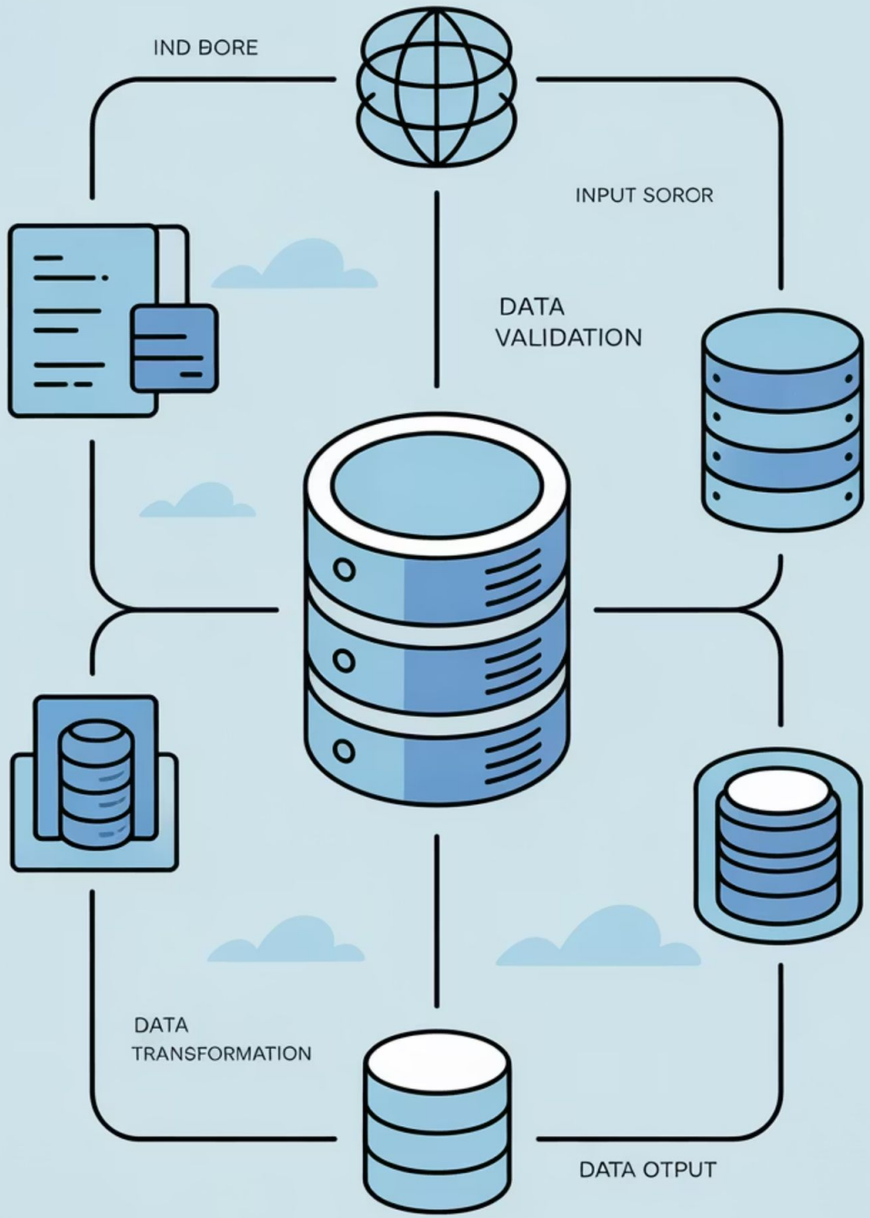
Simplifies the management and processing of queries returning multiple rows, making complex iterations straightforward.

Precise Control

Gives developers complete control over how data is fetched, when it's processed, and how resources are managed.

Logic Integration

Can be seamlessly combined with conditional statements, loops, and complex business logic for sophisticated data processing.



Disadvantages of Cursors

Performance Impact

Processing one row at a time can be significantly slower for very large datasets compared to set-based operations. Each fetch operation incurs overhead.

Memory Consumption

Cursors use more memory resources because they maintain state information and context for the entire result set during processing.

Not for Bulk Operations

When dealing with large-scale data operations, cursors are inefficient. For bulk data processing, use BULK COLLECT and FORALL statements instead, which can be orders of magnitude faster.

Best Practice: Use cursors when you need row-level processing with complex logic. For simple bulk operations, prefer set-based SQL or bulk collection techniques.

Real-Life Analogy & Summary

Think of a Cursor Like a Waiter 🍽️

The waiter (cursor) brings one dish (row) at a time from the kitchen (database).

The chef (database) has prepared all the dishes (records), but the waiter delivers them sequentially to your table.

Summary Comparison

Type	Created By	Used For	Auto-Close	Example
Implicit Cursor	Oracle	Single-row ops	Yes ✓	SELECT INTO
Explicit Cursor	Programmer	Multi-row ops	No ✗	CURSOR c1 IS
FOR Loop Cursor	Oracle (implicit)	Simplified fetch	Yes ✓	FOR rec IN

Cursors are fundamental to PL/SQL programming, enabling sophisticated row-level data processing with precise control and flexibility.

Thank You!