PL/SQL Case Study: Employee Management System

This step-by-step implementation will help you apply the concepts covered in Day 2 (PL/SQL Introduction, Variables, Collections, and Anonymous Blocks).

Case Study Overview

Scenario:  
A company wants to automate employee data processing using PL/SQL. The tasks include:

1. Storing and retrieving employee details.
2. Calculating bonuses based on department.
3. Tracking project assignments using collections.

Step-by-Step Implementation

Step 1: Database Setup

First, create a sample EMPLOYEES table.

CREATE TABLE employees (

    emp\_id NUMBER PRIMARY KEY,

    emp\_name VARCHAR2(100),

    salary NUMBER,

    dept\_id NUMBER,

    hire\_date DATE

);

INSERT INTO employees VALUES (1, 'John Doe', 5000, 10, '01-JAN-2020');

INSERT INTO employees VALUES (2, 'Jane Smith', 6000, 20, '15-MAR-2019');

INSERT INTO employees VALUES (3, 'Robert Brown', 4500, 10, '10-OCT-2021');

Step 2: PL/SQL Anonymous Block (Basic Example)

Task: Retrieve and display an employee’s details.

DECLARE

    v\_emp\_name employees.emp\_name%TYPE;  *-- Anchored datatype*

    v\_salary employees.salary%TYPE;

BEGIN

*-- Fetch employee details*

    SELECT emp\_name, salary INTO v\_emp\_name, v\_salary

    FROM employees

    WHERE emp\_id = 1;

*-- Display output*

    DBMS\_OUTPUT.PUT\_LINE('Employee: ' || v\_emp\_name || ', Salary: ' || v\_salary);

EXCEPTION

    WHEN NO\_DATA\_FOUND THEN

        DBMS\_OUTPUT.PUT\_LINE('Employee not found!');

END;

Output:

Employee: John Doe, Salary: 5000

Step 3: Using Collections (Associative Array)

Task: Store department-wise employee names in an associative array.

DECLARE

*-- Define an associative array (key: dept\_id, value: emp\_name)*

    TYPE dept\_emp\_type IS TABLE OF VARCHAR2(100) INDEX BY PLS\_INTEGER;

    v\_dept\_emps dept\_emp\_type;

BEGIN

*-- Fetch employees and store in the array*

    FOR emp\_rec IN (SELECT dept\_id, emp\_name FROM employees) LOOP

        v\_dept\_emps(emp\_rec.dept\_id) := emp\_rec.emp\_name;

    END LOOP;

*-- Display employees in department 10*

    DBMS\_OUTPUT.PUT\_LINE('Employees in Dept 10: ' || v\_dept\_emps(10));

END;

Output:

Employees in Dept 10: Robert Brown

What are the Steps for declaring Collection

Step 1 : DEfine the type

    TYPE dept\_emp\_type IS TABLE OF VARCHAR2(100) INDEX BY PLS\_INTEGER;

Step 2: Creating a variable of above type

    v\_dept\_emps dept\_emp\_type;

Step 4: Using VARRAY (Fixed-Size Collection)

Task: Store the last 3 hired employees in a VARRAY.

DECLARE

*-- Define a VARRAY type (max 3 employees)*

    TYPE last\_hires\_type IS VARRAY(3) OF VARCHAR2(100);

    v\_last\_hires last\_hires\_type := last\_hires\_type(); *-- Initialize*

BEGIN

*-- Fetch 3 most recent hires*

    FOR emp\_rec IN (

        SELECT emp\_name FROM employees

        ORDER BY hire\_date DESC

        FETCH FIRST 3 ROWS ONLY

    ) LOOP

        v\_last\_hires.EXTEND;

        v\_last\_hires(v\_last\_hires.LAST) := emp\_rec.emp\_name;

    END LOOP;

*-- Display the last hires*

    DBMS\_OUTPUT.PUT\_LINE('Last 3 hires:');

    FOR i IN 1..v\_last\_hires.COUNT LOOP

        DBMS\_OUTPUT.PUT\_LINE(i || ': ' || v\_last\_hires(i));

    END LOOP;

END;

Output:

Last 3 hires:

1: Robert Brown

2: Jane Smith

3: John Doe

Step 5: Using Nested Table (Dynamic Collection)

Task: Store all employees in a department using a nested table.

DECLARE

*-- Define a nested table type*

    TYPE emp\_list\_type IS TABLE OF VARCHAR2(100);

    v\_emp\_list emp\_list\_type := emp\_list\_type(); *-- Initialize*

BEGIN

*-- Fetch employees from department 10*

    SELECT emp\_name BULK COLLECT INTO v\_emp\_list

    FROM employees

    WHERE dept\_id = 10;

*-- Display employees*

    DBMS\_OUTPUT.PUT\_LINE('Employees in Dept 10:');

    FOR i IN 1..v\_emp\_list.COUNT LOOP

        DBMS\_OUTPUT.PUT\_LINE(i || ': ' || v\_emp\_list(i));

    END LOOP;

END;

Output:

Employees in Dept 10:

1: John Doe

2: Robert Brown

Step 6: Bonus Calculation (Using Procedures & Exception Handling)

Task: Calculate a 10% bonus for employees in department 10.

DECLARE

    v\_bonus NUMBER;

    v\_emp\_id NUMBER := 1; *-- Example: John Doe (Dept 10)*

BEGIN

*-- Check if employee exists*

    SELECT salary \* 0.10 INTO v\_bonus

    FROM employees

    WHERE emp\_id = v\_emp\_id AND dept\_id = 10;

    DBMS\_OUTPUT.PUT\_LINE('Bonus for Employee ' || v\_emp\_id || ': $' || v\_bonus);

EXCEPTION

    WHEN NO\_DATA\_FOUND THEN

        DBMS\_OUTPUT.PUT\_LINE('Employee not eligible for bonus!');

END;

Output:

Bonus for Employee 1: $500

Summary of Concepts Applied

|  |  |
| --- | --- |
| **PL/SQL Topic** | **Implementation** |
| Anonymous Blocks | Basic employee data retrieval |
| Anchored Data Types | %TYPE for variable declaration |
| Associative Arrays | Storing department-wise employees |
| VARRAY | Storing last 3 hires (fixed size) |
| Nested Tables | Dynamic list of employees in a department |
| Exception Handling | Bonus calculation with error handling |

PL/SQL Syntax Reference Table

|  |  |  |  |
| --- | --- | --- | --- |
| **Construct** | **Syntax** | **Example from Case Study** | **Purpose** |
| Anonymous Block | DECLARE [variables] BEGIN [logic] EXCEPTION [handlers] END; | Step 2 (Employee data retrieval) | Basic PL/SQL execution unit |
| Anchored Datatype | variable\_name table\_name.column\_name%TYPE; | v\_emp\_name employees.emp\_name%TYPE; | Inherits column datatype |
| Associative Array | TYPE type\_name IS TABLE OF datatype INDEX BY key\_type; | TYPE dept\_emp\_type IS TABLE OF VARCHAR2(100) INDEX BY PLS\_INTEGER; | Key-value storage |
| VARRAY | TYPE type\_name IS VARRAY(size) OF datatype; | TYPE last\_hires\_type IS VARRAY(3) OF VARCHAR2(100); | Fixed-size array |
| Nested Table | TYPE type\_name IS TABLE OF datatype; | TYPE emp\_list\_type IS TABLE OF VARCHAR2(100); | Dynamic-size array |
| EXTEND Method | collection\_name.EXTEND[(n)]; | v\_last\_hires.EXTEND; | Allocates space in collections |
| LAST Attribute | collection\_name.LAST | v\_last\_hires(v\_last\_hires.LAST) := emp\_rec.emp\_name; | Gets highest used index |
| COUNT Attribute | collection\_name.COUNT | FOR i IN 1..v\_last\_hires.COUNT LOOP | Gets number of elements |
| FOR LOOP | FOR index IN [REVERSE] start..end LOOP [statements] END LOOP; | Step 4 (Iterate through VARRAY) | Iterates through a range |
| BULK COLLECT | SELECT column\_name BULK COLLECT INTO collection\_name FROM table\_name; | SELECT emp\_name BULK COLLECT INTO v\_emp\_list FROM employees WHERE dept\_id = 10; | Fills collections efficiently |
| Cursor FOR LOOP | FOR record\_name IN (SELECT\_statement) LOOP [statements] END LOOP; | Step 3 (FOR emp\_rec IN (SELECT dept\_id, emp\_name FROM employees) LOOP) | Iterates through query results |

Key Notes:

1. 1-Based Indexing: All PL/SQL collections start at index 1 (not 0).
2. Initialization Requirement: Collections must be initialized before use:
3. v\_last\_hires last\_hires\_type := last\_hires\_type(); *-- Empty constructor*
4. Exception Handling: Always handle NO\_DATA\_FOUND when using SELECT INTO.

Flow Comparison: EXTEND + LAST vs. BULK COLLECT

|  |  |  |
| --- | --- | --- |
| **Approach** | **When to Use** | **Example** |
| EXTEND + LAST | Manual element-by-element addition | Step 4 (VARRAY population) |
| BULK COLLECT | Batch loading from queries | Step 5 (Nested Table fill) |

1. Variable Operations

|  |
| --- |
|  |

|  |  |  |
| --- | --- | --- |
| **Operation** | **Example** | **Practice Scenario** |
| Declaration | v\_name VARCHAR2(100);  v\_salary employees.salary%TYPE; | Declare variables for employee data. |
| Assignment | v\_name := 'John Doe';  v\_salary := 5000; | Store hardcoded or fetched values. |
| Anchored Datatypes | v\_emp\_name employees.emp\_name%TYPE; | Dynamically match column datatypes. |
| Default Values | v\_bonus NUMBER := 0; | Initialize variables before use. |
| NULL Handling | IF v\_name IS NULL THEN ... | Check for uninitialized variables. |
| Scope Rules | Nested block with DECLARE v\_local\_var ... | Experiment with global vs. local variables. |

2. Collection Operations

Common to All Collections (VARRAY, Nested Table, Associative Array)

|  |  |  |
| --- | --- | --- |
| **Operation** | **Example** | **Practice Scenario** |
| Initialization | v\_employees emp\_list\_type := emp\_list\_type(); | Initialize empty collections. |
| Element Access | v\_employees(1) := 'John'; | Add/modify elements by index. |
| COUNT | IF v\_employees.COUNT > 0 THEN ... | Check the number of elements. |
| FIRST/LAST | FOR i IN v\_employees.FIRST..v\_employees.LAST LOOP | Iterate through valid indices. |
| EXISTS | IF v\_employees.EXISTS(3) THEN ... | Check if an index exists. |
| DELETE | v\_employees.DELETE(2); | Remove elements by index. |

Specific to VARRAY/Nested Tables

|  |  |  |
| --- | --- | --- |
| **Operation** | **Example** | **Practice Scenario** |
| EXTEND | v\_employees.EXTEND;  v\_employees.EXTEND(2); | Allocate space for new elements. |
| TRIM | v\_employees.TRIM;  v\_employees.TRIM(2); | Remove elements from the end. |
| LIMIT (VARRAY only) | IF v\_employees.COUNT < v\_employees.LIMIT THEN ... | Check remaining capacity in VARRAY. |

Specific to Associative Arrays

|  |  |  |
| --- | --- | --- |
| **Operation** | **Example** | **Practice Scenario** |
| String Indexing | v\_dept\_emps('IT') := 'Alice'; | Use strings as keys. |
| Sparse Collections | v\_employees(10) := 'Bob';  v\_employees(20) := 'Charlie'; | Non-sequential indices. |

3. Advanced Operations

|  |  |  |
| --- | --- | --- |
| **Operation** | **Example** | **Practice Scenario** |
| BULK COLLECT | SELECT emp\_name BULK COLLECT INTO v\_employees FROM employees; | Populate collections from queries. |
| FORALL | FORALL i IN 1..v\_ids.COUNT<br> UPDATE employees SET salary = v\_salaries(i)... | Batch DML operations. |
| Collection Methods | v\_employees.TRIM;  v\_employees.EXTEND(3); | Chain methods for complex logic. |
| Nested Collections | TYPE dept\_type IS TABLE OF emp\_list\_type; |  |