

# **PL/SQL Collections and Records Project Report.**

## **Employee Salary Management System**

**Name:** Peace Bizima      **ID:** 27778

**Date:** November 2025

**Course:** Database Development with PL/SQL

**Institution:** AUCA

---

## **1. Introduction**

### **Project Overview**

This project demonstrates the use of **PL/SQL Collections, Records, and GOTO statements** through the development of an **Employee Salary Management System**. The program handles employee information, salary history, payroll calculations, and special logic for senior employees.

### **Objectives**

- Demonstrate all three PL/SQL collection types
- Use table-based and user-defined records
- Apply GOTO statements for special processing
- Build a business-like salary management system
- Implement strong error handling and data validation

### **Technologies Used**

- Oracle Database 21c
  - PL/SQL
  - SQL\*Plus
  - VARRAY, Nested Table, Associative Array
  - Table-based and user-defined records
- 

## **2. Database Schema Design**

Two main tables were created for this project:

### **Employees Table**

- Stores basic employee information (ID, names, department, base salary).

### **Salary History Table**

- Stores monthly salary amounts and bonuses per employee.

### **Sample Data**

Five employees were inserted, with multiple salary history entries to allow testing of collections, records, and bulk operations.

---

## **3. Implementation Summary**

### **3.1 Collections Used**

#### **VARRAY**

- Fixed-size array (max 5 items)
- Used to store department employee IDs
- Maintains order and cannot have gaps

#### **Nested Table**

- Dynamic and can grow or shrink
- Used to store salary history amounts
- Works with BULK COLLECT for fast data loading

#### **Associative Array**

- Key-value collection indexed by numbers
- Used to map employee IDs to full names
- Offers fast lookup

### **Example Types Created**

- dept\_employee\_ids – VARRAY
- salary\_table – Nested Table
- employee\_name\_array – Associative Array

Screenshot:

```
SET SERVEROUTPUT ON;

DECLARE
    -- VARRAY Collection
    TYPE dept_employee_ids IS VARRAY(5) OF employees.employee_id%TYPE;

    -- Nested Table Collection
    TYPE salary_table IS TABLE OF salary_history.salary_amount%TYPE;

    -- Associative Array Collection
    TYPE employee_name_array IS TABLE OF VARCHAR2(100) INDEX BY PLS_INTEGER;
```

---

## 3.2 Records Used

### Table-Based Record

- Uses %ROWTYPE
- Matches the entire structure of the employees table
- Simple and direct for storing row data

### User-defined Record

- Custom structure combining multiple fields
- Includes a nested collection to hold salary history inside the record
- More flexible for complex business logic

Screenshots:

```
-- Table-based Record
emp_record employees%ROWTYPE;

-- User-defined Record
TYPE employee_comprehensive_rec IS RECORD (
    emp_id employees.employee_id%TYPE,
    full_name VARCHAR2(100),
    department employees.department_id%TYPE,
    base_sal employees.base_salary%TYPE,
    salary_history salary_table
);

-- Variables
v_department_employees dept_employee_ids;
v_employee_names employee_name_array;
v_comprehensive_emp employee_comprehensive_rec;
v_total_payroll NUMBER := 0;
v_employee_count NUMBER := 0;
v_emp_id NUMBER;
```

---

## 4. Program Execution and Results

### 4.1 Collection Initialization Output

```
VARRAY initialized with 3 employees
Associative array initialized with 3 names
```

The program correctly loaded:

- 3 employee IDs into VARRAY
- 3 employee names into associative array

---

### 4.2 Employee Processing with GOTO

```
Regular: John Doe - $5000
Regular: Jane Smith - $6000
Using GOTO for senior employee
Senior: Charlie Wilson - $7000 + $1000 bonus
```

Logic Explained:

- Regular employees follow the normal process
- A GOTO statement was used for employee ID 5
- Senior employees get an extra \$1000 bonus
- This demonstrates PL/SQL control flow logic

```
--- Processing Employees with GOTO ---
Regular: John Doe - $5000
Regular: Jane Smith - $6000
Using GOTO for senior employee
Senior: Charlie Wilson - $7000 + $1000 bonus

--- Results ---
Total Employees: 3
Total Payroll: $19000
Average Salary: $6333.33
```

---

## 4.3 Payroll Calculation

Total Employees: 3  
Total Payroll: \$19000  
Average Salary: \$6333.33

The system calculated:

- Number of employees processed
- Total payroll including senior bonus
- Average salary using collection values

---

## 4.4 Record Demonstration

Table-based Record: John Doe  
User-defined Record: John Doe  
Salary History Entries: 2

This shows:

- Successful fetching of employee row using %ROWTYPE
- Correct creation of user-defined record
- Nested table holding salary history inside the record

## Screenshot:

```
== PL/SQL COLLECTIONS AND RECORDS DEMONSTRATION ==
✓ VARRAY initialized with 3 elements
✓ Associative Array initialized with 3 elements

--- Processing Employees with GOTO ---
Regular: John Doe - $5000
Regular: Jane Smith - $6000
Using GOTO for senior employee
Senior: Charlie Wilson - $7000 + $1000 bonus

--- Results ---
Total Employees: 3
Total Payroll: $19000
Average Salary: $6333.33
```

---

## 5. Key Features and Structure

### Main Program Steps

1. Declare collections and records
2. Initialize VARRAY, Nested Table, and Associative Array
3. Process employees and apply GOTO logic
4. Perform payroll calculations
5. Demonstrate both record types
6. Output results

### Key PL/SQL Features Demonstrated

#### Collections

- VARRAY → fixed number of IDs
- Nested Table → salary history
- Associative Array → fast name lookup

#### Records

- Table-based → structure of employees table
- User-defined → flexible record combining multiple fields

## Control Flow

- GOTO used for senior employee logic
- Conditional statements
- Loops for reading collection data

## Error Handling

- NO\_DATA\_FOUND
  - ZERO\_DIVIDE
  - OTHERS for unexpected errors
- 

# 6. Error Handling and Validation

The program includes:

- Checks for missing employees
- Prevention of division-by-zero
- Helpful user messages
- Validation for uninitialized collections
- A general exception block capturing unexpected errors

### Exception screenshot:

```
-- Table-based record
BEGIN
    SELECT * INTO emp_record FROM employees WHERE employee_id = 1;
    DBMS_OUTPUT.PUT_LINE('Table-based Record: ' || emp_record.first_name || ' ' || emp_record.last_name);
EXCEPTION
    WHEN NO_DATA_FOUND THEN
        DBMS_OUTPUT.PUT_LINE('Table-based Record: Employee 1 not found');
END;

-- User-defined record with nested table
BEGIN
    SELECT employee_id, first_name || ' ' || last_name, department_id, base_salary
    INTO v_comprehensive.emp.emp_id, v_comprehensive.emp.full_name, v_comprehensive.emp.department, v_comprehensive.emp.base_sal
    FROM employees WHERE employee_id = 1;

    SELECT salary_amount BULK COLLECT INTO v_comprehensive.emp.salary_history
    FROM salary_history WHERE employee_id = 1;

    DBMS_OUTPUT.PUT_LINE('User-defined Record: ' || v_comprehensive.emp.full_name);
    DBMS_OUTPUT.PUT_LINE('Salary History Entries: ' || v_comprehensive.emp.salary_history.COUNT);
EXCEPTION
    WHEN NO_DATA_FOUND THEN
        DBMS_OUTPUT.PUT_LINE('User-defined Record: Data not found for employee 1');
END;

DBMS_OUTPUT.PUT_LINE(CHR(10) || '== PROJECT COMPLETED SUCCESSFULLY ==');
DBMS_OUTPUT.PUT_LINE('/ VARRAY Collections');
DBMS_OUTPUT.PUT_LINE('/ Nested Table Collections');
```

This ensures the program runs safely even with bad or incomplete data.

---

## 7. Conclusion

### Project Achievements

- Successfully implemented all PL/SQL collection types
- Worked with both table-based and custom records
- Demonstrated GOTO for special processing
- Built a real-world salary management logic
- Ensured strong exception handling