**Exercise 7: Packages**

**Scenario 1:** Group all customer-related procedures and functions into a package.

**Question:** Create a package **CustomerManagement** with procedures for adding a new customer, updating customer details, and a function to get customer balance.

C**reate the CustomerManagement package specification:**

CREATE PACKAGE CustomerManagement AS

*-- Procedure to add a new customer*

PROCEDURE AddNewCustomer(p\_CustomerID IN NUMBER, p\_Name IN VARCHAR2, p\_DOB IN DATE);

*-- Procedure to update customer details*

PROCEDURE UpdateCustomerDetails(p\_CustomerID IN NUMBER, p\_Name IN VARCHAR2, p\_DOB IN DATE);

*-- Function to get customer balance*

FUNCTION GetCustomerBalance(p\_CustomerID IN NUMBER) RETURN NUMBER;

END CustomerManagement;

/

**Create the CustomerManagement package body:**

CREATE OR REPLACE PACKAGE BODY CustomerManagement AS

*-- Procedure to add a new customer*

PROCEDURE AddNewCustomer(p\_CustomerID IN NUMBER, p\_Name IN VARCHAR2, p\_DOB IN DATE) IS

BEGIN

*-- Insert a new customer into the Customers table*

INSERT INTO Customers (CustomerID, Name, DOB, Balance, LastModified)

VALUES (p\_CustomerID, p\_Name, p\_DOB, 0, SYSDATE);

EXCEPTION

WHEN DUP\_VAL\_ON\_INDEX THEN

*-- Handle the case when the customer ID already exists*

DBMS\_OUTPUT.PUT\_LINE('Error: Customer with ID ' || p\_CustomerID || ' already exists.');

END AddNewCustomer;

*-- Procedure to update customer details*

PROCEDURE UpdateCustomerDetails(p\_CustomerID IN NUMBER, p\_Name IN VARCHAR2, p\_DOB IN DATE) IS

BEGIN

*-- Update customer details in the Customers table*

UPDATE Customers

SET Name = p\_Name,

DOB = p\_DOB,

LastModified = SYSDATE

WHERE CustomerID = p\_CustomerID;

*-- Check if the update was successful*

IF SQL%ROWCOUNT = 0 THEN

DBMS\_OUTPUT.PUT\_LINE('Error: No customer found with ID ' || p\_CustomerID);

END IF;

END UpdateCustomerDetails;

*-- Function to get customer balance*

FUNCTION GetCustomerBalance(p\_CustomerID IN NUMBER) RETURN NUMBER IS

v\_Balance NUMBER;

BEGIN

*-- Retrieve the customer's balance*

SELECT Balance INTO v\_Balance

FROM Customers

WHERE CustomerID = p\_CustomerID;

RETURN v\_Balance;

EXCEPTION

WHEN NO\_DATA\_FOUND THEN

*-- Handle the case when no customer is found*

DBMS\_OUTPUT.PUT\_LINE('Error: No customer found with ID ' || p\_CustomerID);

RETURN NULL;

END GetCustomerBalance;

END CustomerManagement;

/

To use this package, you can call the procedures and functions as follows:

*-- Add a new customer*

BEGIN

CustomerManagement.AddNewCustomer(1, 'Alice Smith', TO\_DATE('1990-01-01', 'YYYY-MM-DD'));

END;

/

*-- Update customer details*

BEGIN

CustomerManagement.UpdateCustomerDetails(1, 'Alice Johnson', TO\_DATE('1990-01-01', 'YYYY-MM-DD'));

END;

/

*-- Get customer balance*

DECLARE

v\_Balance NUMBER;

BEGIN

v\_Balance := CustomerManagement.GetCustomerBalance(1);

DBMS\_OUTPUT.PUT\_LINE('Customer Balance: ' || v\_Balance);

END;

/

**Scenario 2:** Create a package to manage employee data.

**Question:** Write a package **EmployeeManagement** with procedures to hire new employees, update employee details, and a function to calculate annual salary.

**Create the EmployeeManagement package specification:**

CREATE PACKAGE EmployeeManagement AS

*-- Procedure to hire a new employee*

PROCEDURE HireEmployee(p\_EmployeeID IN NUMBER, p\_Name IN VARCHAR2, p\_Position IN VARCHAR2, p\_Salary IN NUMBER, p\_Department IN VARCHAR2, p\_HireDate IN DATE);

*-- Procedure to update employee details*

PROCEDURE UpdateEmployeeDetails(p\_EmployeeID IN NUMBER, p\_Name IN VARCHAR2, p\_Position IN VARCHAR2, p\_Salary IN NUMBER, p\_Department IN VARCHAR2);

*-- Function to calculate annual salary*

FUNCTION CalculateAnnualSalary(p\_EmployeeID IN NUMBER) RETURN NUMBER;

END EmployeeManagement;

/

Create the EmployeeManagement package body:

CREATE OR REPLACE PACKAGE BODY EmployeeManagement AS

*-- Procedure to hire a new employee*

PROCEDURE HireEmployee(p\_EmployeeID IN NUMBER, p\_Name IN VARCHAR2, p\_Position IN VARCHAR2, p\_Salary IN NUMBER, p\_Department IN VARCHAR2, p\_HireDate IN DATE) IS

BEGIN

*-- Insert a new employee into the Employees table*

INSERT INTO Employees (EmployeeID, Name, Position, Salary, Department, HireDate)

VALUES (p\_EmployeeID, p\_Name, p\_Position, p\_Salary, p\_Department, p\_HireDate);

EXCEPTION

WHEN DUP\_VAL\_ON\_INDEX THEN

*-- Handle the case when the employee ID already exists*

DBMS\_OUTPUT.PUT\_LINE('Error: Employee with ID ' || p\_EmployeeID || ' already exists.');

END HireEmployee;

*-- Procedure to update employee details*

PROCEDURE UpdateEmployeeDetails(p\_EmployeeID IN NUMBER, p\_Name IN VARCHAR2, p\_Position IN VARCHAR2, p\_Salary IN NUMBER, p\_Department IN VARCHAR2) IS

BEGIN

*-- Update employee details in the Employees table*

UPDATE Employees

SET Name = p\_Name,

Position = p\_Position,

Salary = p\_Salary,

Department = p\_Department

WHERE EmployeeID = p\_EmployeeID;

*-- Check if the update was successful*

IF SQL%ROWCOUNT = 0 THEN

DBMS\_OUTPUT.PUT\_LINE('Error: No employee found with ID ' || p\_EmployeeID);

END IF;

END UpdateEmployeeDetails;

*-- Function to calculate annual salary*

FUNCTION CalculateAnnualSalary(p\_EmployeeID IN NUMBER) RETURN NUMBER IS

v\_Salary NUMBER;

BEGIN

*-- Retrieve the employee's salary*

SELECT Salary INTO v\_Salary

FROM Employees

WHERE EmployeeID = p\_EmployeeID;

*-- Calculate and return the annual salary*

RETURN v\_Salary \* 12;

EXCEPTION

WHEN NO\_DATA\_FOUND THEN

*-- Handle the case when no employee is found*

DBMS\_OUTPUT.PUT\_LINE('Error: No employee found with ID ' || p\_EmployeeID);

RETURN NULL;

END CalculateAnnualSalary;

END EmployeeManagement;

/

To use this package, you can call the procedures and functions as follows:

*-- Hire a new employee*

BEGIN

EmployeeManagement.HireEmployee(1, 'John Doe', 'Software Engineer', 5000, 'IT', TO\_DATE('2023-01-01', 'YYYY-MM-DD'));

END;

/

*-- Update employee details*

BEGIN

EmployeeManagement.UpdateEmployeeDetails(1, 'John Smith', 'Senior Software Engineer', 6000, 'IT');

END;

/

*-- Calculate annual salary*

DECLARE

v\_AnnualSalary NUMBER;

BEGIN

v\_AnnualSalary := EmployeeManagement.CalculateAnnualSalary(1);

DBMS\_OUTPUT.PUT\_LINE('Annual Salary: ' || v\_AnnualSalary);

END;

/

**Scenario 3:** Group all account-related operations into a package.

**Question:** Create a package **AccountOperations** with procedures for opening a new account, closing an account, and a function to get the total balance of a customer across all accounts.

**Create the AccountOperations package specification:**

CREATE OR REPLACE PACKAGE AccountOperations AS

*-- Procedure to open a new account*

PROCEDURE OpenAccount(p\_AccountID IN NUMBER, p\_CustomerID IN NUMBER, p\_AccountType IN VARCHAR2, p\_Balance IN NUMBER);

*-- Procedure to close an account*

PROCEDURE CloseAccount(p\_AccountID IN NUMBER);

*-- Function to get the total balance of a customer across all accounts*

FUNCTION GetTotalBalance(p\_CustomerID IN NUMBER) RETURN NUMBER;

END AccountOperations;

/

**Create the AccountOperations package body:**

CREATE OR REPLACE PACKAGE BODY AccountOperations AS

*-- Procedure to open a new account*

PROCEDURE OpenAccount(p\_AccountID IN NUMBER, p\_CustomerID IN NUMBER, p\_AccountType IN VARCHAR2, p\_Balance IN NUMBER) IS

BEGIN

*-- Insert a new account into the Accounts table*

INSERT INTO Accounts (AccountID, CustomerID, AccountType, Balance, LastModified)

VALUES (p\_AccountID, p\_CustomerID, p\_AccountType, p\_Balance, SYSDATE);

EXCEPTION

WHEN DUP\_VAL\_ON\_INDEX THEN

*-- Handle the case when the account ID already exists*

DBMS\_OUTPUT.PUT\_LINE('Error: Account with ID ' || p\_AccountID || ' already exists.');

END OpenAccount;

*-- Procedure to close an account*

PROCEDURE CloseAccount(p\_AccountID IN NUMBER) IS

BEGIN

*-- Delete the account from the Accounts table*

DELETE FROM Accounts

WHERE AccountID = p\_AccountID;

*-- Check if the deletion was successful*

IF SQL%ROWCOUNT = 0 THEN

DBMS\_OUTPUT.PUT\_LINE('Error: No account found with ID ' || p\_AccountID);

END IF;

END CloseAccount;

*-- Function to get the total balance of a customer across all accounts*

FUNCTION GetTotalBalance(p\_CustomerID IN NUMBER) RETURN NUMBER IS

v\_TotalBalance NUMBER;

BEGIN

*-- Retrieve the total balance of the customer across all accounts*

SELECT SUM(Balance) INTO v\_TotalBalance

FROM Accounts

WHERE CustomerID = p\_CustomerID;

RETURN v\_TotalBalance;

EXCEPTION

WHEN NO\_DATA\_FOUND THEN

*-- Handle the case when no accounts are found for the customer*

DBMS\_OUTPUT.PUT\_LINE('Error: No accounts found for customer ID ' || p\_CustomerID);

RETURN NULL;

END GetTotalBalance;

END AccountOperations;

/

To use this package, you can call the procedures and functions as follows:

*-- Open a new account*

BEGIN

AccountOperations.OpenAccount(1, 1, 'Savings', 1000);

END;

/

*-- Close an account*

BEGIN

AccountOperations.CloseAccount(1);

END;

/

*-- Get the total balance of a customer across all accounts*

DECLARE

v\_TotalBalance NUMBER;

BEGIN

v\_TotalBalance := AccountOperations.GetTotalBalance(1);

DBMS\_OUTPUT.PUT\_LINE('Total Balance: ' || v\_TotalBalance);

END;

/