**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.

CREATE OR REPLACE PACKAGE CustomerManagement AS

-- Procedure to add a new customer

PROCEDURE AddNewCustomer(

p\_CustomerID IN Customers.CustomerID%TYPE,

p\_Name IN Customers.Name%TYPE,

p\_DOB IN Customers.DOB%TYPE,

p\_Balance IN Customers.Balance%TYPE

);

-- Procedure to update customer details

PROCEDURE UpdateCustomerDetails(

p\_CustomerID IN Customers.CustomerID%TYPE,

p\_Name IN Customers.Name%TYPE,

p\_DOB IN Customers.DOB%TYPE,

p\_Balance IN Customers.Balance%TYPE

);

-- Function to get customer balance

FUNCTION GetCustomerBalance(

p\_CustomerID IN Customers.CustomerID%TYPE

) RETURN NUMBER;

END CustomerManagement;

/

CREATE OR REPLACE PACKAGE BODY CustomerManagement AS

-- Implementation of AddNewCustomer procedure

PROCEDURE AddNewCustomer(

p\_CustomerID IN Customers.CustomerID%TYPE,

p\_Name IN Customers.Name%TYPE,

p\_DOB IN Customers.DOB%TYPE,

p\_Balance IN Customers.Balance%TYPE

) IS

BEGIN

BEGIN

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

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

COMMIT;

EXCEPTION

WHEN DUP\_VAL\_ON\_INDEX THEN

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

WHEN OTHERS THEN

DBMS\_OUTPUT.PUT\_LINE('Error: ' || SQLERRM);

END;

END AddNewCustomer;

-- Implementation of UpdateCustomerDetails procedure

PROCEDURE UpdateCustomerDetails(

p\_CustomerID IN Customers.CustomerID%TYPE,

p\_Name IN Customers.Name%TYPE,

p\_DOB IN Customers.DOB%TYPE,

p\_Balance IN Customers.Balance%TYPE

) IS

BEGIN

BEGIN

UPDATE Customers

SET Name = p\_Name,

DOB = p\_DOB,

Balance = p\_Balance,

LastModified = SYSDATE

WHERE CustomerID = p\_CustomerID;

IF SQL%ROWCOUNT = 0 THEN

DBMS\_OUTPUT.PUT\_LINE('Error: Customer with ID ' || p\_CustomerID || ' does not exist.');

ELSE

COMMIT;

END IF;

EXCEPTION

WHEN OTHERS THEN

DBMS\_OUTPUT.PUT\_LINE('Error: ' || SQLERRM);

END;

END UpdateCustomerDetails;

-- Implementation of GetCustomerBalance function

FUNCTION GetCustomerBalance(

p\_CustomerID IN Customers.CustomerID%TYPE

) RETURN NUMBER IS

v\_Balance NUMBER;

BEGIN

BEGIN

SELECT Balance INTO v\_Balance

FROM Customers

WHERE CustomerID = p\_CustomerID;

EXCEPTION

WHEN NO\_DATA\_FOUND THEN

v\_Balance := NULL;

DBMS\_OUTPUT.PUT\_LINE('Error: Customer with ID ' || p\_CustomerID || ' does not exist.');

WHEN OTHERS THEN

DBMS\_OUTPUT.PUT\_LINE('Error: ' || SQLERRM);

END;

RETURN v\_Balance;

END GetCustomerBalance;

END CustomerManagement;

/

-- Add a new customer

BEGIN

CustomerManagement.AddNewCustomer(3, 'Alice Cooper', TO\_DATE('1995-03-22', 'YYYY-MM-DD'), 2000);

END;

/

-- Update customer details

BEGIN

CustomerManagement.UpdateCustomerDetails(1, 'Johnathan Doe', TO\_DATE('1985-05-15', 'YYYY-MM-DD'), 1200);

END;

/

-- Get customer balance

DECLARE

v\_Balance NUMBER;

BEGIN

v\_Balance := CustomerManagement.GetCustomerBalance(2);

DBMS\_OUTPUT.PUT\_LINE('Balance for customer ID 2: ' || 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 OR REPLACE PACKAGE EmployeeManagement AS

-- Procedure to hire a new employee

PROCEDURE HireEmployee(

p\_EmployeeID IN Employees.EmployeeID%TYPE,

p\_Name IN Employees.Name%TYPE,

p\_Position IN Employees.Position%TYPE,

p\_Salary IN Employees.Salary%TYPE,

p\_Department IN Employees.Department%TYPE,

p\_HireDate IN Employees.HireDate%TYPE

);

-- Procedure to update employee details

PROCEDURE UpdateEmployeeDetails(

p\_EmployeeID IN Employees.EmployeeID%TYPE,

p\_Name IN Employees.Name%TYPE,

p\_Position IN Employees.Position%TYPE,

p\_Salary IN Employees.Salary%TYPE,

p\_Department IN Employees.Department%TYPE,

p\_HireDate IN Employees.HireDate%TYPE

);

-- Function to calculate annual salary

FUNCTION CalculateAnnualSalary(

p\_EmployeeID IN Employees.EmployeeID%TYPE

) RETURN NUMBER;

END EmployeeManagement;

/

CREATE OR REPLACE PACKAGE BODY EmployeeManagement AS

-- Implementation of HireEmployee procedure

PROCEDURE HireEmployee(

p\_EmployeeID IN Employees.EmployeeID%TYPE,

p\_Name IN Employees.Name%TYPE,

p\_Position IN Employees.Position%TYPE,

p\_Salary IN Employees.Salary%TYPE,

p\_Department IN Employees.Department%TYPE,

p\_HireDate IN Employees.HireDate%TYPE

) IS

BEGIN

BEGIN

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

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

COMMIT;

EXCEPTION

WHEN DUP\_VAL\_ON\_INDEX THEN

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

WHEN OTHERS THEN

DBMS\_OUTPUT.PUT\_LINE('Error: ' || SQLERRM);

END;

END HireEmployee;

-- Implementation of UpdateEmployeeDetails procedure

PROCEDURE UpdateEmployeeDetails(

p\_EmployeeID IN Employees.EmployeeID%TYPE,

p\_Name IN Employees.Name%TYPE,

p\_Position IN Employees.Position%TYPE,

p\_Salary IN Employees.Salary%TYPE,

p\_Department IN Employees.Department%TYPE,

p\_HireDate IN Employees.HireDate%TYPE

) IS

BEGIN

BEGIN

UPDATE Employees

SET Name = p\_Name,

Position = p\_Position,

Salary = p\_Salary,

Department = p\_Department,

HireDate = p\_HireDate

WHERE EmployeeID = p\_EmployeeID;

IF SQL%ROWCOUNT = 0 THEN

DBMS\_OUTPUT.PUT\_LINE('Error: Employee with ID ' || p\_EmployeeID || ' does not exist.');

ELSE

COMMIT;

END IF;

EXCEPTION

WHEN OTHERS THEN

DBMS\_OUTPUT.PUT\_LINE('Error: ' || SQLERRM);

END;

END UpdateEmployeeDetails;

-- Implementation of CalculateAnnualSalary function

FUNCTION CalculateAnnualSalary(

p\_EmployeeID IN Employees.EmployeeID%TYPE

) RETURN NUMBER IS

v\_Salary NUMBER;

BEGIN

BEGIN

SELECT Salary INTO v\_Salary

FROM Employees

WHERE EmployeeID = p\_EmployeeID;

RETURN v\_Salary \* 12; -- Assuming salary is monthly

EXCEPTION

WHEN NO\_DATA\_FOUND THEN

DBMS\_OUTPUT.PUT\_LINE('Error: Employee with ID ' || p\_EmployeeID || ' does not exist.');

RETURN NULL;

WHEN OTHERS THEN

DBMS\_OUTPUT.PUT\_LINE('Error: ' || SQLERRM);

RETURN NULL;

END;

END CalculateAnnualSalary;

END EmployeeManagement;

/

-- Hire a new employee

BEGIN

EmployeeManagement.HireEmployee(3, 'Alice Johnson', 'Analyst', 5000, 'Finance', TO\_DATE('2024-08-01', 'YYYY-MM-DD'));

END;

/

-- Update employee details

BEGIN

EmployeeManagement.UpdateEmployeeDetails(1, 'John Smith', 'Senior Developer', 6500, 'IT', TO\_DATE('2015-06-15', 'YYYY-MM-DD'));

END;

/

-- Calculate annual salary

DECLARE

v\_AnnualSalary NUMBER;

BEGIN

v\_AnnualSalary := EmployeeManagement.CalculateAnnualSalary(2);

DBMS\_OUTPUT.PUT\_LINE('Annual Salary for employee ID 2: ' || 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 OR REPLACE PACKAGE AccountOperations AS

-- Procedure to open a new account

PROCEDURE OpenAccount(

p\_AccountID IN Accounts.AccountID%TYPE,

p\_CustomerID IN Accounts.CustomerID%TYPE,

p\_AccountType IN Accounts.AccountType%TYPE,

p\_Balance IN Accounts.Balance%TYPE

);

-- Procedure to close an account

PROCEDURE CloseAccount(

p\_AccountID IN Accounts.AccountID%TYPE

);

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

FUNCTION GetTotalBalance(

p\_CustomerID IN Accounts.CustomerID%TYPE

) RETURN NUMBER;

END AccountOperations;

/

CREATE OR REPLACE PACKAGE BODY AccountOperations AS

-- Implementation of OpenAccount procedure

PROCEDURE OpenAccount(

p\_AccountID IN Accounts.AccountID%TYPE,

p\_CustomerID IN Accounts.CustomerID%TYPE,

p\_AccountType IN Accounts.AccountType%TYPE,

p\_Balance IN Accounts.Balance%TYPE

) IS

BEGIN

BEGIN

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

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

COMMIT;

EXCEPTION

WHEN DUP\_VAL\_ON\_INDEX THEN

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

WHEN OTHERS THEN

DBMS\_OUTPUT.PUT\_LINE('Error: ' || SQLERRM);

END;

END OpenAccount;

-- Implementation of CloseAccount procedure

PROCEDURE CloseAccount(

p\_AccountID IN Accounts.AccountID%TYPE

) IS

BEGIN

BEGIN

DELETE FROM Accounts

WHERE AccountID = p\_AccountID;

IF SQL%ROWCOUNT = 0 THEN

DBMS\_OUTPUT.PUT\_LINE('Error: Account with ID ' || p\_AccountID || ' does not exist.');

ELSE

COMMIT;

END IF;

EXCEPTION

WHEN OTHERS THEN

DBMS\_OUTPUT.PUT\_LINE('Error: ' || SQLERRM);

END;

END CloseAccount;

-- Implementation of GetTotalBalance function

FUNCTION GetTotalBalance(

p\_CustomerID IN Accounts.CustomerID%TYPE

) RETURN NUMBER IS

v\_TotalBalance NUMBER;

BEGIN

BEGIN

SELECT SUM(Balance) INTO v\_TotalBalance

FROM Accounts

WHERE CustomerID = p\_CustomerID;

-- Handle the case where the customer has no accounts

IF v\_TotalBalance IS NULL THEN

v\_TotalBalance := 0;

END IF;

RETURN v\_TotalBalance;

EXCEPTION

WHEN NO\_DATA\_FOUND THEN

DBMS\_OUTPUT.PUT\_LINE('Error: Customer with ID ' || p\_CustomerID || ' does not exist.');

RETURN 0;

WHEN OTHERS THEN

DBMS\_OUTPUT.PUT\_LINE('Error: ' || SQLERRM);

RETURN 0;

END;

END GetTotalBalance;

END AccountOperations;

/

-- Open a new account

BEGIN

AccountOperations.OpenAccount(3, 1, 'Checking', 5000);

END;

/

-- Close an account

BEGIN

AccountOperations.CloseAccount(3);

END;

/

-- Get total balance for a customer

DECLARE

v\_TotalBalance NUMBER;

BEGIN

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

DBMS\_OUTPUT.PUT\_LINE('Total Balance for customer ID 1: ' || v\_TotalBalance);

END;

/