**Exercise 1: Control Structures**

**Scenario 1:**

**BEGIN**

**FOR cust\_rec IN (**

**SELECT CustomerID, Name, Age, InterestRate**

**FROM Customers**

**WHERE Age > 60**

**) LOOP**

**UPDATE Customers**

**SET InterestRate = InterestRate - 1**

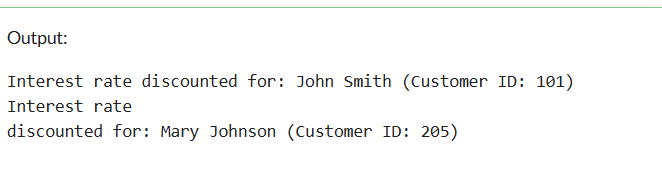
**WHERE CustomerID = cust\_rec.CustomerID;**

**DBMS\_OUTPUT.PUT\_LINE('Interest rate discounted for: ' || cust\_rec.Name || ' (Customer ID: ' || cust\_rec.CustomerID || ')');**

**END LOOP;**

**END;**

**/**

****

**Scenario 2:**

BEGIN

FOR cust\_rec IN (

SELECT CustomerID, Name, Balance

FROM Customers

WHERE Balance > 10000

) LOOP

UPDATE Customers

SET IsVIP = 'Y'

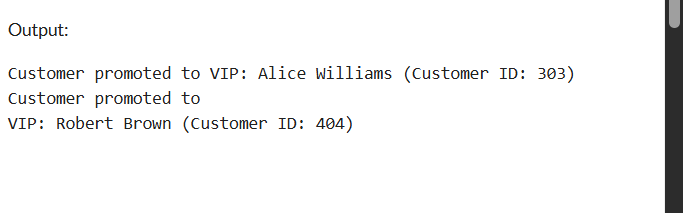
WHERE CustomerID = cust\_rec.CustomerID;

DBMS\_OUTPUT.PUT\_LINE('Customer promoted to VIP: ' || cust\_rec.Name || ' (Customer ID: ' || cust\_rec.CustomerID || ')');

END LOOP;

END;

/



**Scenario 3:**

BEGIN

FOR loan\_rec IN (

SELECT CustomerID, Name, LoanDueDate

FROM Customers

WHERE LoanDueDate BETWEEN SYSDATE AND SYSDATE + 30

) LOOP

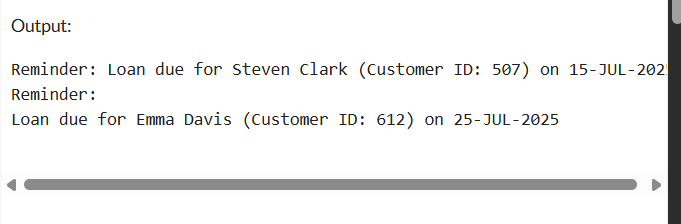
DBMS\_OUTPUT.PUT\_LINE('Reminder: Loan due for ' || loan\_rec.Name ||

' (Customer ID: ' || loan\_rec.CustomerID ||

') on ' || TO\_CHAR(loan\_rec.LoanDueDate, 'DD-MON-YYYY'));

END LOOP;

END;

/

**Exercise 2: Error Handling**

**Scenario 1:**

CREATE OR REPLACE PROCEDURE SafeTransferFunds (

p\_FromAccountID IN NUMBER,

p\_ToAccountID IN NUMBER,

p\_Amount IN NUMBER

)

AS

v\_Balance NUMBER;

BEGIN

-- Get balance of from account

SELECT Balance INTO v\_Balance

FROM Accounts

WHERE AccountID = p\_FromAccountID;

IF v\_Balance < p\_Amount THEN

RAISE\_APPLICATION\_ERROR(-20001, 'Insufficient funds in the source account.');

END IF;

-- Deduct from sender

UPDATE Accounts

SET Balance = Balance - p\_Amount

WHERE AccountID = p\_FromAccountID;

-- Add to receiver

UPDATE Accounts

SET Balance = Balance + p\_Amount

WHERE AccountID = p\_ToAccountID;

COMMIT;

DBMS\_OUTPUT.PUT\_LINE('Transfer successful.');

EXCEPTION

WHEN OTHERS THEN

ROLLBACK;

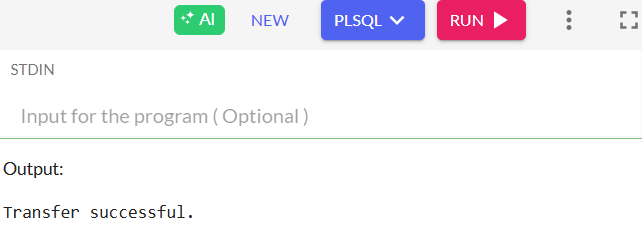
INSERT INTO ErrorLog (ProcedureName, ErrorMessage)

VALUES ('SafeTransferFunds', SQLERRM);

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

END;

/



**Scenario 2:**

CREATE OR REPLACE PROCEDURE UpdateSalary (

p\_EmployeeID IN NUMBER,

p\_Percent IN NUMBER

)

AS

BEGIN

UPDATE Employees

SET Salary = Salary + (Salary \* p\_Percent / 100)

WHERE EmployeeID = p\_EmployeeID;

IF SQL%ROWCOUNT = 0 THEN

RAISE\_APPLICATION\_ERROR(-20002, 'Employee not found.');

END IF;

COMMIT;

DBMS\_OUTPUT.PUT\_LINE('Salary updated successfully.');

EXCEPTION

WHEN OTHERS THEN

ROLLBACK;

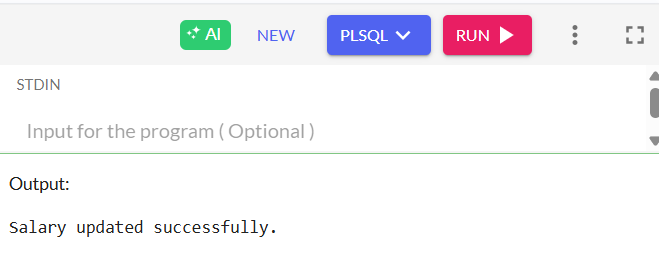
INSERT INTO ErrorLog (ProcedureName, ErrorMessage)

VALUES ('UpdateSalary', SQLERRM);

DBMS\_OUTPUT.PUT\_LINE('Salary update failed: ' || SQLERRM);

END;

/



**Scenario 3:**

CREATE OR REPLACE PROCEDURE AddNewCustomer (

p\_CustomerID IN NUMBER,

p\_Name IN VARCHAR2,

p\_Age IN NUMBER,

p\_Balance IN NUMBER

)

AS

BEGIN

INSERT INTO Customers (CustomerID, Name, Age, Balance)

VALUES (p\_CustomerID, p\_Name, p\_Age, p\_Balance);

COMMIT;

DBMS\_OUTPUT.PUT\_LINE('Customer added successfully.');

EXCEPTION

WHEN DUP\_VAL\_ON\_INDEX THEN

INSERT INTO ErrorLog (ProcedureName, ErrorMessage)

VALUES ('AddNewCustomer', 'Customer with ID ' || p\_CustomerID || ' already exists.');

DBMS\_OUTPUT.PUT\_LINE('Add customer failed: Duplicate customer ID.');

WHEN OTHERS THEN

ROLLBACK;

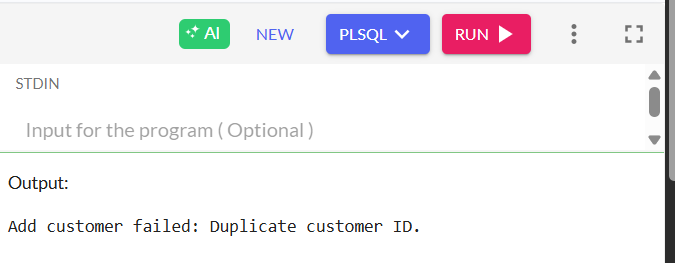
INSERT INTO ErrorLog (ProcedureName, ErrorMessage)

VALUES ('AddNewCustomer', SQLERRM);

DBMS\_OUTPUT.PUT\_LINE('Add customer failed: ' || SQLERRM);

END;

/



**Exercise 3: Stored Procedures**

**Scenario 1:**

CREATE OR REPLACE PROCEDURE ProcessMonthlyInterest

AS

BEGIN

FOR acc IN (SELECT AccountID, Balance FROM Accounts WHERE AccountType = 'Savings') LOOP

UPDATE Accounts

SET Balance = Balance + (Balance \* 0.01)

WHERE AccountID = acc.AccountID;

DBMS\_OUTPUT.PUT\_LINE('Interest added to Account ID: ' || acc.AccountID ||

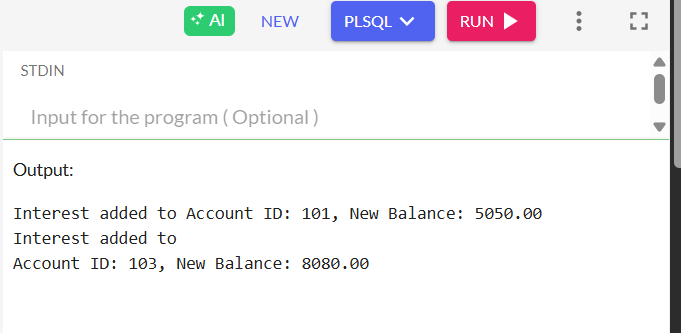
', New Balance: ' || TO\_CHAR(acc.Balance \* 1.01, '9999.99'));

END LOOP;

COMMIT;

END;

/



**Scenario 2:**

CREATE OR REPLACE PROCEDURE UpdateEmployeeBonus (

p\_Department IN VARCHAR2,

p\_BonusPct IN NUMBER

)

AS

BEGIN

FOR emp IN (SELECT EmployeeID, Name, Salary FROM Employees WHERE Department = p\_Department) LOOP

UPDATE Employees

SET Salary = Salary + (Salary \* p\_BonusPct / 100)

WHERE EmployeeID = emp.EmployeeID;

DBMS\_OUTPUT.PUT\_LINE('Bonus applied to ' || emp.Name ||

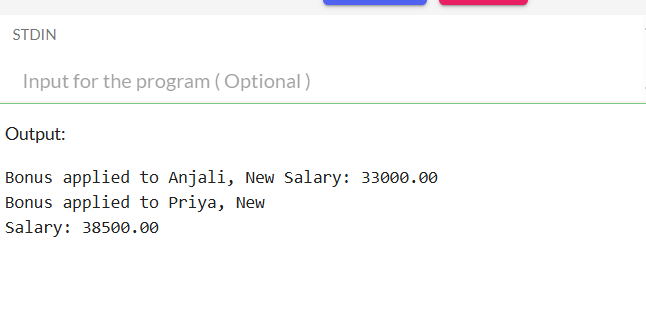
', New Salary: ' || TO\_CHAR(emp.Salary \* (1 + p\_BonusPct / 100), '99999.99'));

END LOOP;

COMMIT;

END;

/



**Scenario 3:**

CREATE OR REPLACE PROCEDURE TransferFunds (

p\_FromAccountID IN NUMBER,

p\_ToAccountID IN NUMBER,

p\_Amount IN NUMBER

)

AS

v\_Balance NUMBER;

BEGIN

-- Check source balance

SELECT Balance INTO v\_Balance

FROM Accounts

WHERE AccountID = p\_FromAccountID;

IF v\_Balance < p\_Amount THEN

DBMS\_OUTPUT.PUT\_LINE('Transfer failed: Insufficient balance in Account ID ' || p\_FromAccountID);

RETURN;

END IF;

-- Debit source

UPDATE Accounts

SET Balance = Balance - p\_Amount

WHERE AccountID = p\_FromAccountID;

-- Credit destination

UPDATE Accounts

SET Balance = Balance + p\_Amount

WHERE AccountID = p\_ToAccountID;

COMMIT;

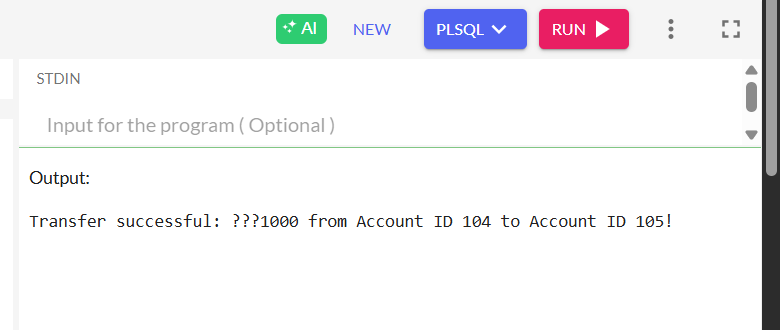
DBMS\_OUTPUT.PUT\_LINE('Transfer successful: ₹' || p\_Amount ||

' from Account ID ' || p\_FromAccountID ||

' to Account ID ' || p\_ToAccountID);

END;

/



**Exercise 4: Functions**

Scenario 1:

CREATE OR REPLACE FUNCTION CalculateAge(p\_DOB DATE)

RETURN NUMBER

IS

v\_Age NUMBER;

BEGIN

v\_Age := FLOOR(MONTHS\_BETWEEN(SYSDATE, p\_DOB) / 12);

RETURN v\_Age;

END;

/

Scenario 2:

CREATE OR REPLACE FUNCTION CalculateMonthlyInstallment (

p\_LoanAmount IN NUMBER,

p\_AnnualRate IN NUMBER, -- Annual interest rate in percent

p\_LoanYears IN NUMBER

) RETURN NUMBER

IS

v\_MonthlyRate NUMBER;

v\_Months NUMBER;

v\_Installment NUMBER;

BEGIN

v\_MonthlyRate := p\_AnnualRate / (12 \* 100); -- Convert to monthly rate

v\_Months := p\_LoanYears \* 12;

v\_Installment := p\_LoanAmount \* v\_MonthlyRate /

(1 - POWER(1 + v\_MonthlyRate, -v\_Months));

RETURN ROUND(v\_Installment, 2);

END;

/

Scenario3:

CREATE OR REPLACE FUNCTION HasSufficientBalance (

p\_AccountID IN NUMBER,

p\_Amount IN NUMBER

) RETURN BOOLEAN

IS

v\_Balance NUMBER;

BEGIN

SELECT Balance INTO v\_Balance

FROM Accounts

WHERE AccountID = p\_AccountID;

RETURN v\_Balance >= p\_Amount;

EXCEPTION

WHEN NO\_DATA\_FOUND THEN

RETURN FALSE;

END;

/

Scenario3:

CREATE OR REPLACE FUNCTION HasSufficientBalance (

p\_AccountID IN NUMBER,

p\_Amount IN NUMBER

) RETURN BOOLEAN

IS

v\_Balance NUMBER;

BEGIN

SELECT Balance INTO v\_Balance

FROM Accounts

WHERE AccountID = p\_AccountID;

RETURN v\_Balance >= p\_Amount;

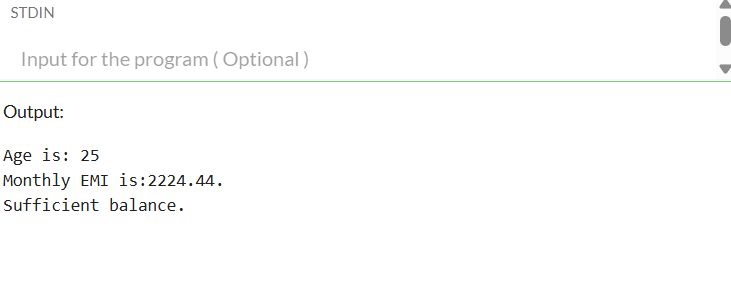
EXCEPTION

WHEN NO\_DATA\_FOUND THEN

RETURN FALSE;

END;

/



**Exercise 4:Triggers**

**Scenario 1:**

CREATE OR REPLACE TRIGGER UpdateCustomerLastModified

BEFORE UPDATE ON Customers

FOR EACH ROW

BEGIN

:NEW.LastModified := SYSDATE;

END;

/

Scenario2:

CREATE OR REPLACE TRIGGER LogTransaction

AFTER INSERT ON Transactions

FOR EACH ROW

BEGIN

INSERT INTO AuditLog (TransactionID, AccountID, Type, Amount)

VALUES (:NEW.TransactionID, :NEW.AccountID, :NEW.Type, :NEW.Amount);

END;

/

Scenario3:

CREATE OR REPLACE TRIGGER CheckTransactionRules

BEFORE INSERT ON Transactions

FOR EACH ROW

DECLARE

v\_balance NUMBER;

BEGIN

-- Get current balance

SELECT Balance INTO v\_balance FROM Customers WHERE CustomerID = :NEW.AccountID;

-- Check deposit

IF :NEW.Type = 'Deposit' AND :NEW.Amount <= 0 THEN

RAISE\_APPLICATION\_ERROR(-20001, 'Deposit amount must be positive.');

END IF;

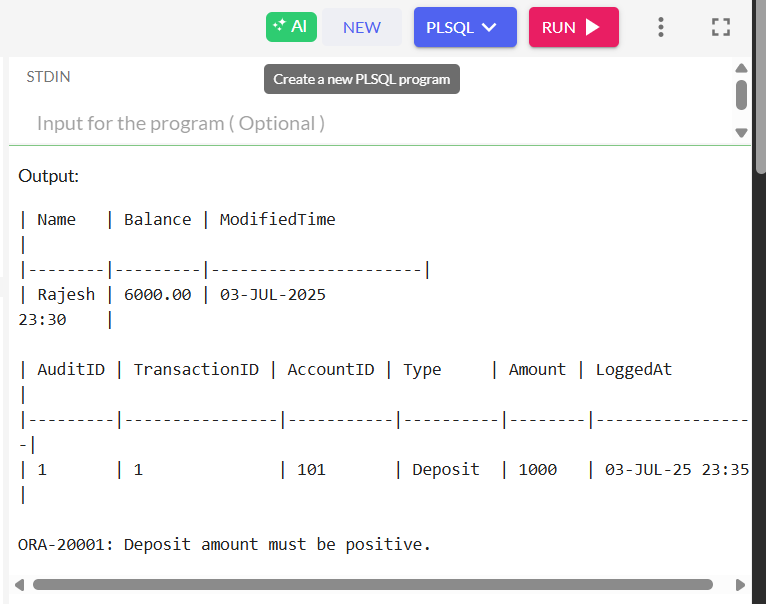
-- Check withdrawal

IF :NEW.Type = 'Withdrawal' AND :NEW.Amount > v\_balance THEN

RAISE\_APPLICATION\_ERROR(-20002, 'Insufficient balance for withdrawal.');

END IF;

END;

/

**Exercise 6: Cursors:**

**Scenario1:**

**SET SERVEROUTPUT ON;**

DECLARE

CURSOR txn\_cursor IS

SELECT c.CustomerID, c.Name, t.Amount, t.TxnType, t.TxnDate

FROM Customers c

JOIN Transactions t ON c.CustomerID = t.CustomerID

WHERE TRUNC(t.TxnDate, 'MM') = TRUNC(SYSDATE, 'MM');

v\_custID Customers.CustomerID%TYPE;

v\_name Customers.Name%TYPE;

v\_amount Transactions.Amount%TYPE;

v\_type Transactions.TxnType%TYPE;

v\_date Transactions.TxnDate%TYPE;

BEGIN

OPEN txn\_cursor;

LOOP

FETCH txn\_cursor INTO v\_custID, v\_name, v\_amount, v\_type, v\_date;

EXIT WHEN txn\_cursor%NOTFOUND;

DBMS\_OUTPUT.PUT\_LINE('Customer: ' || v\_name ||

', Type: ' || v\_type ||

', Amount: ₹' || v\_amount ||

', Date: ' || TO\_CHAR(v\_date, 'DD-MON-YYYY'));

END LOOP;

CLOSE txn\_cursor;

END;

/

Scenario2:

SET SERVEROUTPUT ON;

DECLARE

CURSOR acc\_cursor IS

SELECT AccountID, Balance FROM Accounts;

v\_accID Accounts.AccountID%TYPE;

v\_balance Accounts.Balance%TYPE;

v\_fee CONSTANT NUMBER := 500;

BEGIN

OPEN acc\_cursor;

LOOP

FETCH acc\_cursor INTO v\_accID, v\_balance;

EXIT WHEN acc\_cursor%NOTFOUND;

IF v\_balance >= v\_fee THEN

UPDATE Accounts SET Balance = Balance - v\_fee WHERE AccountID = v\_accID;

DBMS\_OUTPUT.PUT\_LINE('₹' || v\_fee || ' fee deducted from Account ID: ' || v\_accID);

ELSE

DBMS\_OUTPUT.PUT\_LINE('Insufficient balance in Account ID: ' || v\_accID || '. Fee not deducted.');

END IF;

END LOOP;

CLOSE acc\_cursor;

COMMIT;

END;

/

Scenario3:

SET SERVEROUTPUT ON;

DECLARE

CURSOR loan\_cursor IS

SELECT LoanID, LoanAmount FROM Loans;

v\_loanID Loans.LoanID%TYPE;

v\_amount Loans.LoanAmount%TYPE;

v\_newRate NUMBER(5,2);

BEGIN

OPEN loan\_cursor;

LOOP

FETCH loan\_cursor INTO v\_loanID, v\_amount;

EXIT WHEN loan\_cursor%NOTFOUND;

IF v\_amount < 100000 THEN

v\_newRate := 10;

ELSIF v\_amount <= 500000 THEN

v\_newRate := 8;

ELSE

v\_newRate := 7;

END IF;

UPDATE Loans SET InterestRate = v\_newRate WHERE LoanID = v\_loanID;

DBMS\_OUTPUT.PUT\_LINE('Loan ID ' || v\_loanID || ' updated to interest rate: ' || v\_newRate || '%');

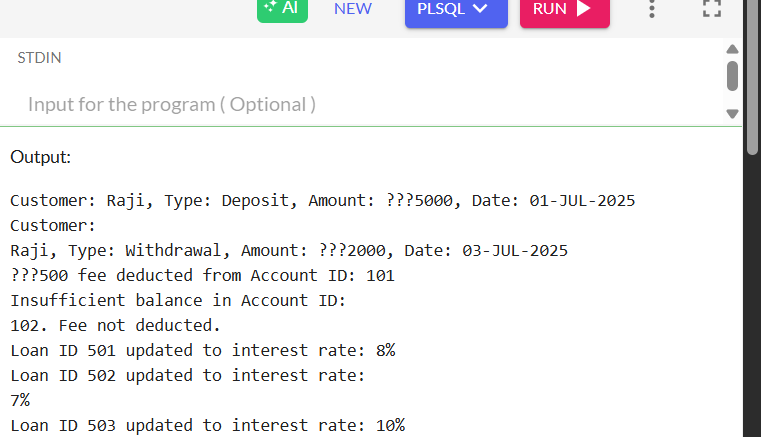
END LOOP;

CLOSE loan\_cursor;

COMMIT;

END;

/



**Exercise 7: Packages:**

CREATE OR REPLACE PACKAGE CustomerManagement AS

PROCEDURE AddCustomer(p\_ID NUMBER, p\_Name VARCHAR2, p\_DOB DATE, p\_Balance NUMBER);

PROCEDURE UpdateCustomer(p\_ID NUMBER, p\_Name VARCHAR2, p\_Balance NUMBER);

FUNCTION GetCustomerBalance(p\_ID NUMBER) RETURN NUMBER;

END CustomerManagement;

/

CREATE OR REPLACE PACKAGE BODY CustomerManagement AS

PROCEDURE AddCustomer(p\_ID NUMBER, p\_Name VARCHAR2, p\_DOB DATE, p\_Balance NUMBER) IS

BEGIN

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

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

DBMS\_OUTPUT.PUT\_LINE('Customer added: ' || p\_Name);

END;

PROCEDURE UpdateCustomer(p\_ID NUMBER, p\_Name VARCHAR2, p\_Balance NUMBER) IS

BEGIN

UPDATE Customers

SET Name = p\_Name, Balance = p\_Balance, LastModified = SYSDATE

WHERE CustomerID = p\_ID;

DBMS\_OUTPUT.PUT\_LINE('Customer updated: ' || p\_Name);

END;

FUNCTION GetCustomerBalance(p\_ID NUMBER) RETURN NUMBER IS

v\_balance NUMBER;

BEGIN

SELECT Balance INTO v\_balance FROM Customers WHERE CustomerID = p\_ID;

RETURN v\_balance;

END;

END CustomerManagement;

/

SET SERVEROUTPUT ON;

BEGIN

CustomerManagement.AddCustomer(3, 'Meena Kumari', TO\_DATE('1995-12-01', 'YYYY-MM-DD'), 2000);

CustomerManagement.UpdateCustomer(3, 'Meena Kumari', 2500);

DBMS\_OUTPUT.PUT\_LINE('Balance: ₹' || CustomerManagement.GetCustomerBalance(3));

END;

/

Scenario 2:

CREATE OR REPLACE PACKAGE EmployeeManagement AS

PROCEDURE HireEmployee(p\_ID NUMBER, p\_Name VARCHAR2, p\_Position VARCHAR2, p\_Salary NUMBER, p\_Dept VARCHAR2, p\_HireDate DATE);

PROCEDURE UpdateEmployee(p\_ID NUMBER, p\_Salary NUMBER);

FUNCTION CalculateAnnualSalary(p\_ID NUMBER) RETURN NUMBER;

END EmployeeManagement;

/

CREATE OR REPLACE PACKAGE BODY EmployeeManagement AS

PROCEDURE HireEmployee(p\_ID NUMBER, p\_Name VARCHAR2, p\_Position VARCHAR2, p\_Salary NUMBER, p\_Dept VARCHAR2, p\_HireDate DATE) IS

BEGIN

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

VALUES (p\_ID, p\_Name, p\_Position, p\_Salary, p\_Dept, p\_HireDate);

DBMS\_OUTPUT.PUT\_LINE('Employee hired: ' || p\_Name);

END;

PROCEDURE UpdateEmployee(p\_ID NUMBER, p\_Salary NUMBER) IS

BEGIN

UPDATE Employees SET Salary = p\_Salary WHERE EmployeeID = p\_ID;

DBMS\_OUTPUT.PUT\_LINE('Employee salary updated.');

END;

FUNCTION CalculateAnnualSalary(p\_ID NUMBER) RETURN NUMBER IS

v\_salary NUMBER;

BEGIN

SELECT Salary INTO v\_salary FROM Employees WHERE EmployeeID = p\_ID;

RETURN v\_salary \* 12;

END;

END EmployeeManagement;

/

BEGIN

EmployeeManagement.HireEmployee(3, 'Ravi Kumar', 'Analyst', 55000, 'Finance', TO\_DATE('2020-08-01', 'YYYY-MM-DD'));

EmployeeManagement.UpdateEmployee(3, 58000);

DBMS\_OUTPUT.PUT\_LINE('Annual Salary: ₹' || EmployeeManagement.CalculateAnnualSalary(3));

END;

/

Scenario 3:

CREATE OR REPLACE PACKAGE AccountOperations AS

PROCEDURE OpenAccount(p\_ID NUMBER, p\_CustomerID NUMBER, p\_Type VARCHAR2, p\_Balance NUMBER);

PROCEDURE CloseAccount(p\_ID NUMBER);

FUNCTION GetTotalBalance(p\_CustomerID NUMBER) RETURN NUMBER;

END AccountOperations;

/

CREATE OR REPLACE PACKAGE BODY AccountOperations AS

PROCEDURE OpenAccount(p\_ID NUMBER, p\_CustomerID NUMBER, p\_Type VARCHAR2, p\_Balance NUMBER) IS

BEGIN

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

VALUES (p\_ID, p\_CustomerID, p\_Type, p\_Balance, SYSDATE);

DBMS\_OUTPUT.PUT\_LINE('Account opened with ID: ' || p\_ID);

END;

PROCEDURE CloseAccount(p\_ID NUMBER) IS

BEGIN

DELETE FROM Accounts WHERE AccountID = p\_ID;

DBMS\_OUTPUT.PUT\_LINE('Account closed: ' || p\_ID);

END;

FUNCTION GetTotalBalance(p\_CustomerID NUMBER) RETURN NUMBER IS

v\_total NUMBER;

BEGIN

SELECT NVL(SUM(Balance), 0) INTO v\_total FROM Accounts WHERE CustomerID = p\_CustomerID;

RETURN v\_total;

END;

END AccountOperations;

/

BEGIN

AccountOperations.OpenAccount(3, 1, 'Fixed', 7000);

AccountOperations.CloseAccount(2);

DBMS\_OUTPUT.PUT\_LINE('Total Balance of Customer 1: ₹' || AccountOperations.GetTotalBalance(1));

END;

/

