[Week 2] [PLSQL] HandsOn

# Exercise 1: Control Structures

## Scenario 1

The bank wants to apply a discount to loan interest rates for customers above 60 years old.

PL/SQL Block / Code:

BEGIN  
 FOR rec IN (SELECT LoanID, CustomerID, InterestRate  
 FROM Loans L  
 JOIN Customers C ON L.CustomerID = C.CustomerID  
 WHERE MONTHS\_BETWEEN(SYSDATE, C.DOB)/12 > 60) LOOP  
  
 UPDATE Loans  
 SET InterestRate = InterestRate - 1  
 WHERE LoanID = rec.LoanID;  
  
 DBMS\_OUTPUT.PUT\_LINE('Updated LoanID ' || rec.LoanID ||  
 ' for CustomerID ' || rec.CustomerID);  
 END LOOP;  
END;  
/

Sample Output:

## Scenario 2

Promote customers to VIP status based on balance > $10,000.

PL/SQL Block / Code:

BEGIN  
 FOR rec IN (SELECT CustomerID  
 FROM Customers  
 WHERE Balance > 10000) LOOP  
  
 UPDATE Customers  
 SET Balance = Balance  
 WHERE CustomerID = rec.CustomerID;  
  
 DBMS\_OUTPUT.PUT\_LINE('Customer ' || rec.CustomerID || ' set as VIP.');  
 END LOOP;  
END;  
/

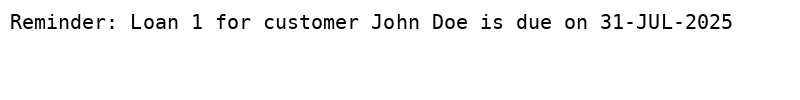
Sample Output:

## Scenario 3

Send reminders for loans due in the next 30 days.

PL/SQL Block / Code:

BEGIN  
 FOR rec IN (SELECT L.LoanID, C.Name, L.EndDate  
 FROM Loans L  
 JOIN Customers C ON L.CustomerID = C.CustomerID  
 WHERE L.EndDate BETWEEN SYSDATE AND SYSDATE + 30) LOOP  
  
 DBMS\_OUTPUT.PUT\_LINE('Reminder: Loan ' || rec.LoanID ||  
 ' for customer ' || rec.Name ||  
 ' is due on ' || rec.EndDate);  
 END LOOP;  
END;  
/

Sample Output:

# Exercise 2: Error Handling

## Scenario 1

Safe transfer of funds between accounts.

PL/SQL Block / Code:

CREATE OR REPLACE PROCEDURE SafeTransferFunds (  
 p\_from\_account\_id NUMBER,  
 p\_to\_account\_id NUMBER,  
 p\_amount NUMBER  
) AS  
 insufficient\_funds EXCEPTION;  
 from\_balance NUMBER;  
BEGIN  
 SELECT Balance  
 INTO from\_balance  
 FROM Accounts  
 WHERE AccountID = p\_from\_account\_id;  
  
 IF from\_balance < p\_amount THEN  
 RAISE insufficient\_funds;  
 END IF;  
  
 UPDATE Accounts  
 SET Balance = Balance - p\_amount  
 WHERE AccountID = p\_from\_account\_id;  
  
 UPDATE Accounts  
 SET Balance = Balance + p\_amount  
 WHERE AccountID = p\_to\_account\_id;  
  
 COMMIT;  
  
 DBMS\_OUTPUT.PUT\_LINE('Transfer successful.');  
  
EXCEPTION  
 WHEN insufficient\_funds THEN  
 ROLLBACK;  
 DBMS\_OUTPUT.PUT\_LINE('Error: Insufficient funds.');  
 WHEN OTHERS THEN  
 ROLLBACK;  
 DBMS\_OUTPUT.PUT\_LINE('Unexpected error: ' || SQLERRM);  
END;  
/

Sample Output:

## Scenario 2

Update salary and handle employee not found.

PL/SQL Block / Code:

CREATE OR REPLACE PROCEDURE UpdateSalary (  
 p\_employee\_id NUMBER,  
 p\_percentage NUMBER  
) AS  
 e\_not\_found EXCEPTION;  
 PRAGMA EXCEPTION\_INIT(e\_not\_found, -01403);  
BEGIN  
 UPDATE Employees  
 SET Salary = Salary \* (1 + p\_percentage/100)  
 WHERE EmployeeID = p\_employee\_id;  
  
 IF SQL%ROWCOUNT = 0 THEN  
 RAISE e\_not\_found;  
 END IF;  
  
 COMMIT;  
  
 DBMS\_OUTPUT.PUT\_LINE('Salary updated.');  
  
EXCEPTION  
 WHEN e\_not\_found THEN  
 DBMS\_OUTPUT.PUT\_LINE('Error: Employee not found.');  
 ROLLBACK;  
 WHEN OTHERS THEN  
 DBMS\_OUTPUT.PUT\_LINE('Error: ' || SQLERRM);  
 ROLLBACK;  
END;  
/

Sample Output:

## Scenario 3

Add new customer, check for duplicate ID.

PL/SQL Block / Code:

CREATE OR REPLACE PROCEDURE AddNewCustomer (  
 p\_customer\_id NUMBER,  
 p\_name VARCHAR2,  
 p\_dob DATE,  
 p\_balance NUMBER  
) AS  
BEGIN  
 INSERT INTO Customers (CustomerID, Name, DOB, Balance, LastModified)  
 VALUES (p\_customer\_id, p\_name, p\_dob, p\_balance, SYSDATE);  
  
 COMMIT;  
  
 DBMS\_OUTPUT.PUT\_LINE('Customer added successfully.');  
  
EXCEPTION  
 WHEN DUP\_VAL\_ON\_INDEX THEN  
 DBMS\_OUTPUT.PUT\_LINE('Error: Customer ID already exists.');  
 ROLLBACK;  
 WHEN OTHERS THEN  
 DBMS\_OUTPUT.PUT\_LINE('Error: ' || SQLERRM);  
 ROLLBACK;  
END;  
/

Sample Output:



# Exercise 3: Stored Procedures

## Scenario 1

Process monthly interest for all savings accounts.

PL/SQL Block / Code:

CREATE OR REPLACE PROCEDURE ProcessMonthlyInterest AS  
BEGIN  
 FOR rec IN (SELECT AccountID, Balance  
 FROM Accounts  
 WHERE AccountType = 'Savings') LOOP  
  
 UPDATE Accounts  
 SET Balance = Balance \* 1.01,  
 LastModified = SYSDATE  
 WHERE AccountID = rec.AccountID;  
  
 DBMS\_OUTPUT.PUT\_LINE('Interest applied to Account ' || rec.AccountID);  
 END LOOP;  
  
 COMMIT;  
END;  
/

Sample Output:

## Scenario 2

Update employee bonus.

PL/SQL Block / Code:

CREATE OR REPLACE PROCEDURE UpdateEmployeeBonus (  
 p\_department VARCHAR2,  
 p\_bonus\_percent NUMBER  
) AS  
BEGIN  
 UPDATE Employees  
 SET Salary = Salary \* (1 + p\_bonus\_percent/100)  
 WHERE Department = p\_department;  
  
 DBMS\_OUTPUT.PUT\_LINE('Bonuses applied for department: ' || p\_department);  
 COMMIT;  
END;  
/

Sample Output:

## Scenario 3

Customers should be able to transfer funds between their accounts.

PL/SQL Block / Code:

CREATE OR REPLACE PROCEDURE TransferFunds (  
 p\_from\_account\_id NUMBER,  
 p\_to\_account\_id NUMBER,  
 p\_amount NUMBER  
) AS  
 from\_balance NUMBER;  
BEGIN  
 SELECT Balance  
 INTO from\_balance  
 FROM Accounts  
 WHERE AccountID = p\_from\_account\_id;  
  
 IF from\_balance < p\_amount THEN  
 RAISE\_APPLICATION\_ERROR(-20001, 'Insufficient balance.');  
 END IF;  
  
 UPDATE Accounts  
 SET Balance = Balance - p\_amount  
 WHERE AccountID = p\_from\_account\_id;  
  
 UPDATE Accounts  
 SET Balance = Balance + p\_amount  
 WHERE AccountID = p\_to\_account\_id;  
  
 COMMIT;  
  
 DBMS\_OUTPUT.PUT\_LINE('Transfer completed.');  
END;  
/

Sample Output:

# Exercise 4: Functions

## Scenario 1

Calculate customer age.

PL/SQL Block / Code:

CREATE OR REPLACE FUNCTION CalculateAge (  
 p\_dob DATE  
) RETURN NUMBER IS  
BEGIN  
 RETURN FLOOR(MONTHS\_BETWEEN(SYSDATE, p\_dob) / 12);  
END;  
/

Sample Output:

## Scenario 2

Calculate monthly installment for a loan.

PL/SQL Block / Code:

CREATE OR REPLACE FUNCTION CalculateMonthlyInstallment (  
 p\_loan\_amount NUMBER,  
 p\_interest\_rate NUMBER,  
 p\_years NUMBER  
) RETURN NUMBER IS  
 monthly\_rate NUMBER;  
 months NUMBER;  
BEGIN  
 monthly\_rate := p\_interest\_rate / 12 / 100;  
 months := p\_years \* 12;  
  
 RETURN (p\_loan\_amount \* monthly\_rate) /  
 (1 - POWER(1 + monthly\_rate, -months));  
END;  
/

Sample Output:

## Scenario 3

Check sufficient balance.

PL/SQL Block / Code:

CREATE OR REPLACE FUNCTION HasSufficientBalance (  
 p\_account\_id NUMBER,  
 p\_amount NUMBER  
) RETURN BOOLEAN IS  
 balance NUMBER;  
BEGIN  
 SELECT Balance  
 INTO balance  
 FROM Accounts  
 WHERE AccountID = p\_account\_id;  
  
 RETURN balance >= p\_amount;  
EXCEPTION  
 WHEN NO\_DATA\_FOUND THEN  
 RETURN FALSE;  
END;  
/

Sample Output:

# Exercise 5: Triggers

## Scenario 1

Update LastModified when customer's record is updated.

PL/SQL Block / Code:

CREATE OR REPLACE TRIGGER UpdateCustomerLastModified  
BEFORE UPDATE ON Customers  
FOR EACH ROW  
BEGIN  
 :NEW.LastModified := SYSDATE;  
END;  
/

Sample Output:

## Scenario 2

Maintain an audit log for all transactions.

PL/SQL Block / Code:

CREATE OR REPLACE TRIGGER LogTransaction  
AFTER INSERT ON Transactions  
FOR EACH ROW  
BEGIN  
 INSERT INTO AuditLog (TransactionID, LogDate, Message)  
 VALUES (:NEW.TransactionID, SYSDATE, 'Transaction logged.');  
END;  
/

Sample Output:

## Scenario 3

Enforce business rules on deposits and withdrawals.

PL/SQL Block / Code:

CREATE OR REPLACE TRIGGER CheckTransactionRules  
BEFORE INSERT ON Transactions  
FOR EACH ROW  
DECLARE  
 v\_balance NUMBER;  
BEGIN  
 IF :NEW.TransactionType = 'Withdrawal' THEN  
 SELECT Balance  
 INTO v\_balance  
 FROM Accounts  
 WHERE AccountID = :NEW.AccountID;  
  
 IF v\_balance < :NEW.Amount THEN  
 RAISE\_APPLICATION\_ERROR(-20001, 'Insufficient balance.');  
 END IF;  
 ELSIF :NEW.TransactionType = 'Deposit' THEN  
 IF :NEW.Amount <= 0 THEN  
 RAISE\_APPLICATION\_ERROR(-20002, 'Deposit amount must be positive.');  
 END IF;  
 END IF;  
END;  
/

Sample Output:

# Exercise 6: Cursors

## Scenario 1

Generate monthly statements.

PL/SQL Block / Code:

DECLARE  
 CURSOR cur IS  
 SELECT AccountID, TransactionDate, Amount, TransactionType  
 FROM Transactions  
 WHERE TO\_CHAR(TransactionDate, 'MM-YYYY') = TO\_CHAR(SYSDATE, 'MM-YYYY');  
  
BEGIN  
 FOR rec IN cur LOOP  
 DBMS\_OUTPUT.PUT\_LINE('AccountID: ' || rec.AccountID ||  
 ', Date: ' || rec.TransactionDate ||  
 ', Amount: ' || rec.Amount ||  
 ', Type: ' || rec.TransactionType);  
 END LOOP;  
END;  
/

Sample Output:

## Scenario 2

Apply annual fee to all accounts.

PL/SQL Block / Code:

DECLARE  
 CURSOR c IS SELECT AccountID, Balance FROM Accounts;  
BEGIN  
 FOR rec IN c LOOP  
 UPDATE Accounts  
 SET Balance = Balance - 100  
 WHERE AccountID = rec.AccountID;  
  
 DBMS\_OUTPUT.PUT\_LINE('Annual fee deducted from Account ' || rec.AccountID);  
 END LOOP;  
 COMMIT;  
END;  
/

Sample Output:

## Scenario 3

Update loan interest rates.

PL/SQL Block / Code:

DECLARE  
 CURSOR c IS SELECT LoanID, InterestRate FROM Loans;  
BEGIN  
 FOR rec IN c LOOP  
 UPDATE Loans  
 SET InterestRate = InterestRate + 0.5  
 WHERE LoanID = rec.LoanID;  
  
 DBMS\_OUTPUT.PUT\_LINE('Updated LoanID ' || rec.LoanID);  
 END LOOP;  
 COMMIT;  
END;  
/

Sample Output:

# Exercise 7: Packages

## Scenario 1

Package CustomerManagement.

PL/SQL Block / Code:

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 VALUES (p\_id, p\_name, p\_dob, p\_balance, SYSDATE);  
 COMMIT;  
 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;  
 COMMIT;  
 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;  
/

Sample Output:

## Scenario 2

Package EmployeeManagement.

PL/SQL Block / Code:

CREATE OR REPLACE PACKAGE EmployeeManagement AS  
 PROCEDURE HireEmployee(p\_id NUMBER, p\_name VARCHAR2, p\_position VARCHAR2, p\_salary NUMBER, p\_department VARCHAR2);  
 PROCEDURE UpdateEmployee(p\_id NUMBER, p\_salary NUMBER);  
 FUNCTION GetAnnualSalary(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\_department VARCHAR2) IS  
 BEGIN  
 INSERT INTO Employees VALUES (p\_id, p\_name, p\_position, p\_salary, p\_department, SYSDATE);  
 COMMIT;  
 END;  
  
 PROCEDURE UpdateEmployee(p\_id NUMBER, p\_salary NUMBER) IS  
 BEGIN  
 UPDATE Employees SET Salary = p\_salary WHERE EmployeeID = p\_id;  
 COMMIT;  
 END;  
  
 FUNCTION GetAnnualSalary(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;  
/

Sample Output:

## Scenario 3

Package AccountOperations.

PL/SQL Block / Code:

CREATE OR REPLACE PACKAGE AccountOperations AS  
 PROCEDURE OpenAccount(p\_id NUMBER, p\_cust\_id NUMBER, p\_type VARCHAR2, p\_balance NUMBER);  
 PROCEDURE CloseAccount(p\_id NUMBER);  
 FUNCTION GetTotalBalance(p\_cust\_id NUMBER) RETURN NUMBER;  
END AccountOperations;  
/  
  
CREATE OR REPLACE PACKAGE BODY AccountOperations AS  
 PROCEDURE OpenAccount(p\_id NUMBER, p\_cust\_id NUMBER, p\_type VARCHAR2, p\_balance NUMBER) IS  
 BEGIN  
 INSERT INTO Accounts VALUES (p\_id, p\_cust\_id, p\_type, p\_balance, SYSDATE);  
 COMMIT;  
 END;  
  
 PROCEDURE CloseAccount(p\_id NUMBER) IS  
 BEGIN  
 DELETE FROM Accounts WHERE AccountID = p\_id;  
 COMMIT;  
 END;  
  
 FUNCTION GetTotalBalance(p\_cust\_id NUMBER) RETURN NUMBER IS  
 v\_total NUMBER;  
 BEGIN  
 SELECT SUM(Balance) INTO v\_total FROM Accounts WHERE CustomerID = p\_cust\_id;  
 RETURN NVL(v\_total, 0);  
 END;  
END AccountOperations;  
/

Sample Output:

