**PL/SQL EXERCISES**

**EXERCISE 1: CONTROL STRUCTURES**

**SCENARIO-1**

BEGIN

FOR customer\_rec IN (SELECT c.CustomerID, l.InterestRate

FROM Customers c

JOIN Loans l ON c.CustomerID = l.CustomerID

WHERE TRUNC(MONTHS\_BETWEEN(SYSDATE, c.DOB) / 12) > 60) LOOP

UPDATE Loans

SET InterestRate = customer\_rec.InterestRate - (customer\_rec.InterestRate \* 0.01)

WHERE CustomerID = customer\_rec.CustomerID;

-- Commit the changes

COMMIT;

END LOOP;

END;

/

**SCENARIO-2**

ALTER TABLE Customers ADD IsVIP VARCHAR2(5);

BEGIN

FOR customer\_rec IN (SELECT CustomerID, Balance

FROM Customers

WHERE Balance > 10000) LOOP

UPDATE Customers

SET IsVIP = 'TRUE'

WHERE CustomerID = customer\_rec.CustomerID;

-- Commit the changes

COMMIT;

END LOOP;

END;

/

**SCENARIO-3**

DECLARE

v\_due\_date Loans.EndDate%TYPE;

v\_customer\_id Loans.CustomerID%TYPE;

v\_message VARCHAR2(200);

BEGIN

FOR loan\_rec IN (SELECT CustomerID, EndDate

FROM Loans

WHERE EndDate BETWEEN SYSDATE AND SYSDATE + 30) LOOP

v\_due\_date := loan\_rec.EndDate;

v\_customer\_id := loan\_rec.CustomerID;

v\_message := 'Reminder: Your loan is due on ' || TO\_CHAR(v\_due\_date, 'YYYY-MM-DD') || '.';

DBMS\_OUTPUT.PUT\_LINE('Customer ID: ' || v\_customer\_id || ' - ' || v\_message);

END LOOP;

END;

/

**EXERCISE 2: ERROR HANDLING**

**SCENARIO-1**

CREATE OR REPLACE PROCEDURE SafeTransferFunds (

p\_from\_account\_id IN NUMBER,

p\_to\_account\_id IN NUMBER,

p\_amount IN NUMBER

) AS

BEGIN

-- Begin the transaction

SAVEPOINT before\_transfer;

-- Check if the from\_account has sufficient funds

DECLARE

v\_from\_balance NUMBER;

BEGIN

SELECT Balance INTO v\_from\_balance

FROM Accounts

WHERE AccountID = p\_from\_account\_id

FOR UPDATE;

IF v\_from\_balance < p\_amount THEN

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

END IF;

UPDATE Accounts

SET Balance = Balance - p\_amount, LastModified = SYSDATE

WHERE AccountID = p\_from\_account\_id;

UPDATE Accounts

SET Balance = Balance + p\_amount, LastModified = SYSDATE

WHERE AccountID = p\_to\_account\_id;

COMMIT;

EXCEPTION

WHEN OTHERS THEN

-- Rollback to the savepoint if any error occurs

ROLLBACK TO before\_transfer;

-- Log the error

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

END;

END SafeTransferFunds;

/

**SCENARIO-2**

CREATE OR REPLACE PROCEDURE UpdateSalary (

p\_employee\_id IN NUMBER,

p\_percentage IN NUMBER

) AS

BEGIN

DECLARE

v\_old\_salary NUMBER;

BEGIN

SELECT Salary INTO v\_old\_salary

FROM Employees

WHERE EmployeeID = p\_employee\_id

FOR UPDATE;

UPDATE Employees

SET Salary = Salary + (Salary \* p\_percentage / 100), LastModified = SYSDATE

WHERE EmployeeID = p\_employee\_id;

COMMIT;

EXCEPTION

WHEN NO\_DATA\_FOUND THEN

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

WHEN OTHERS THEN

-- Log any other errors

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

ROLLBACK;

END;

END UpdateSalary;

/

**SCENARIO-3**

CREATE OR REPLACE PROCEDURE AddNewCustomer (

p\_customer\_id IN NUMBER,

p\_name IN VARCHAR2,

p\_dob IN DATE,

p\_balance IN NUMBER

) AS

BEGIN

BEGIN

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

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

COMMIT;

EXCEPTION

WHEN DUP\_VAL\_ON\_INDEX THEN

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

WHEN OTHERS THEN

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

ROLLBACK;

END;

END AddNewCustomer;

/

**EXERCISE 3: STORED PROCEDURES**

**SCENARIO-1**

CREATE OR REPLACE PROCEDURE ProcessMonthlyInterest AS

BEGIN

FOR account\_rec IN (SELECT AccountID, Balance

FROM Accounts

WHERE AccountType = 'Savings') LOOP

UPDATE Accounts

SET Balance = Balance + (Balance \* 0.01), LastModified = SYSDATE

WHERE AccountID = account\_rec.AccountID;

COMMIT;

END LOOP;

END ProcessMonthlyInterest;

/

**SCENARIO-2**

CREATE OR REPLACE PROCEDURE UpdateEmployeeBonus (

p\_department IN VARCHAR2,

p\_bonus\_percentage IN NUMBER

) AS

BEGIN

UPDATE Employees

SET Salary = Salary + (Salary \* p\_bonus\_percentage / 100), LastModified = SYSDATE

WHERE Department = p\_department;

COMMIT;

EXCEPTION

WHEN OTHERS THEN

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

ROLLBACK;

END UpdateEmployeeBonus;

/

**SCENARIO-3**

CREATE OR REPLACE PROCEDURE TransferFunds (

p\_from\_account\_id IN NUMBER,

p\_to\_account\_id IN NUMBER,

p\_amount IN NUMBER

) AS

BEGIN

SAVEPOINT before\_transfer;

DECLARE

v\_from\_balance NUMBER;

BEGIN

SELECT Balance INTO v\_from\_balance

FROM Accounts

WHERE AccountID = p\_from\_account\_id

FOR UPDATE;

IF v\_from\_balance < p\_amount THEN

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

END IF;

UPDATE Accounts

SET Balance = Balance - p\_amount, LastModified = SYSDATE

WHERE AccountID = p\_from\_account\_id;

UPDATE Accounts

SET Balance = Balance + p\_amount, LastModified = SYSDATE

WHERE AccountID = p\_to\_account\_id;

COMMIT;

EXCEPTION

WHEN OTHERS THEN

ROLLBACK TO before\_transfer;

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

END;

END TransferFunds;

/

**EXERCISE 4: FUNCTIONS**

**SCENARIO-1**

CREATE OR REPLACE FUNCTION CalculateAge (

p\_dob DATE

) RETURN NUMBER AS

v\_age NUMBER;

BEGIN

v\_age := TRUNC(MONTHS\_BETWEEN(SYSDATE, p\_dob) / 12);

RETURN v\_age;

END CalculateAge;

/

**SCENARIO-2**

CREATE OR REPLACE FUNCTION CalculateMonthlyInstallment (

p\_loan\_amount NUMBER,

p\_interest\_rate NUMBER,

p\_duration\_years NUMBER

) RETURN NUMBER AS

v\_monthly\_installment NUMBER;

v\_monthly\_interest\_rate NUMBER;

v\_number\_of\_payments NUMBER;

BEGIN

v\_monthly\_interest\_rate := p\_interest\_rate / 12 / 100;

v\_number\_of\_payments := p\_duration\_years \* 12;

IF v\_monthly\_interest\_rate = 0 THEN

v\_monthly\_installment := p\_loan\_amount / v\_number\_of\_payments;

ELSE

v\_monthly\_installment := p\_loan\_amount \* v\_monthly\_interest\_rate \*

POWER(1 + v\_monthly\_interest\_rate, v\_number\_of\_payments) /

(POWER(1 + v\_monthly\_interest\_rate, v\_number\_of\_payments) - 1);

END IF;

RETURN v\_monthly\_installment;

END CalculateMonthlyInstallment;

/

**SCENARIO-3**

CREATE OR REPLACE FUNCTION HasSufficientBalance (

p\_account\_id NUMBER,

p\_amount NUMBER

) RETURN BOOLEAN AS

v\_balance NUMBER;

BEGIN

SELECT Balance INTO v\_balance

FROM Accounts

WHERE AccountID = p\_account\_id;

IF v\_balance >= p\_amount THEN

RETURN TRUE;

ELSE

RETURN FALSE;

END IF;

EXCEPTION

WHEN NO\_DATA\_FOUND THEN

RETURN FALSE;

END HasSufficientBalance;

/

**EXERCISE 5: TRIGGERS**

**SCENARIO-1**

CREATE OR REPLACE TRIGGER UpdateCustomerLastModified

BEFORE UPDATE ON Customers

FOR EACH ROW

BEGIN

:NEW.LastModified := SYSDATE;

END UpdateCustomerLastModified;

/

**SCENARIO-2**

CREATE TABLE AuditLog (

LogID NUMBER PRIMARY KEY,

TransactionID NUMBER,

AccountID NUMBER,

TransactionDate DATE,

Amount NUMBER,

TransactionType VARCHAR2(10),

LoggedAt DATE

);

CREATE OR REPLACE TRIGGER LogTransaction

AFTER INSERT ON Transactions

FOR EACH ROW

BEGIN

INSERT INTO AuditLog (LogID, TransactionID, AccountID, TransactionDate, Amount, TransactionType, LoggedAt)

VALUES (AuditLog\_SEQ.NEXTVAL, :NEW.TransactionID, :NEW.AccountID, :NEW.TransactionDate, :NEW.Amount, :NEW.TransactionType, SYSDATE);

END LogTransaction;

/

**SCENARIO-3**

CREATE OR REPLACE TRIGGER CheckTransactionRules

BEFORE INSERT ON Transactions

FOR EACH ROW

DECLARE

v\_balance NUMBER;

BEGIN

SELECT Balance INTO v\_balance

FROM Accounts

WHERE AccountID = :NEW.AccountID

FOR UPDATE;

IF :NEW.TransactionType = 'Withdrawal' THEN

IF :NEW.Amount > v\_balance THEN

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

END IF;

END IF;

IF :NEW.TransactionType = 'Deposit' THEN

IF :NEW.Amount <= 0 THEN

RAISE\_APPLICATION\_ERROR(-20003, 'Deposit amount must be positive.');

END IF;

END IF;

END CheckTransactionRules;

/

**EXERCISE 6: CURSORS**

**SCENARIO-1**

DECLARE

CURSOR transactions\_cursor IS

SELECT c.CustomerID, c.Name, t.TransactionDate, t.Amount, t.TransactionType

FROM Customers c

JOIN Accounts a ON c.CustomerID = a.CustomerID

JOIN Transactions t ON a.AccountID = t.AccountID

WHERE EXTRACT(MONTH FROM t.TransactionDate) = EXTRACT(MONTH FROM SYSDATE)

AND EXTRACT(YEAR FROM t.TransactionDate) = EXTRACT(YEAR FROM SYSDATE);

v\_customer\_id Customers.CustomerID%TYPE;

v\_name Customers.Name%TYPE;

v\_transaction\_date Transactions.TransactionDate%TYPE;

v\_amount Transactions.Amount%TYPE;

v\_transaction\_type Transactions.TransactionType%TYPE;

BEGIN

OPEN transactions\_cursor;

LOOP

FETCH transactions\_cursor INTO v\_customer\_id, v\_name, v\_transaction\_date, v\_amount, v\_transaction\_type;

EXIT WHEN transactions\_cursor%NOTFOUND;

DBMS\_OUTPUT.PUT\_LINE('Customer ID: ' || v\_customer\_id);

DBMS\_OUTPUT.PUT\_LINE('Name: ' || v\_name);

DBMS\_OUTPUT.PUT\_LINE('Transaction Date: ' || v\_transaction\_date);

DBMS\_OUTPUT.PUT\_LINE('Amount: ' || v\_amount);

DBMS\_OUTPUT.PUT\_LINE('Transaction Type: ' || v\_transaction\_type);

DBMS\_OUTPUT.PUT\_LINE('---------------------------');

END LOOP;

CLOSE transactions\_cursor;

END;

/

**SCENARIO-2**

DECLARE

CURSOR accounts\_cursor IS

SELECT AccountID, Balance

FROM Accounts;

v\_account\_id Accounts.AccountID%TYPE;

v\_balance Accounts.Balance%TYPE;

v\_annual\_fee CONSTANT NUMBER := 50;

BEGIN

OPEN accounts\_cursor;

LOOP

FETCH accounts\_cursor INTO v\_account\_id, v\_balance;

EXIT WHEN accounts\_cursor%NOTFOUND;

UPDATE Accounts

SET Balance = Balance - v\_annual\_fee, LastModified = SYSDATE

WHERE AccountID = v\_account\_id;

DBMS\_OUTPUT.PUT\_LINE('Annual fee applied to Account ID: ' || v\_account\_id);

END LOOP;

CLOSE accounts\_cursor;

-- Commit the changes

COMMIT;

END;

/

**SCENARIO-3**

DECLARE

CURSOR loans\_cursor IS

SELECT LoanID, InterestRate

FROM Loans;

v\_loan\_id Loans.LoanID%TYPE;

v\_interest\_rate Loans.InterestRate%TYPE;

v\_new\_interest\_rate Loans.InterestRate%TYPE;

BEGIN

OPEN loans\_cursor;

LOOP

FETCH loans\_cursor INTO v\_loan\_id, v\_interest\_rate;

EXIT WHEN loans\_cursor%NOTFOUND;

v\_new\_interest\_rate := v\_interest\_rate + 0.5;

UPDATE Loans

SET InterestRate = v\_new\_interest\_rate

WHERE LoanID = v\_loan\_id;

DBMS\_OUTPUT.PUT\_LINE('Interest rate updated for Loan ID: ' || v\_loan\_id || ' to ' || v\_new\_interest\_rate || '%');

END LOOP;

CLOSE loans\_cursor;

COMMIT;

END;

/

**EXERCISE 7: PACKAGES**

**SCENARIO-1**

CREATE OR REPLACE PACKAGE CustomerManagement AS

PROCEDURE AddNewCustomer(p\_customer\_id NUMBER, p\_name VARCHAR2, p\_dob DATE, p\_balance NUMBER);

PROCEDURE UpdateCustomerDetails(p\_customer\_id NUMBER, p\_name VARCHAR2, p\_dob DATE);

FUNCTION GetCustomerBalance(p\_customer\_id NUMBER) RETURN NUMBER;

END CustomerManagement;

/

CREATE OR REPLACE PACKAGE BODY CustomerManagement AS

PROCEDURE AddNewCustomer(p\_customer\_id NUMBER, p\_name VARCHAR2, p\_dob DATE, p\_balance NUMBER) IS

BEGIN

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

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

END AddNewCustomer;

PROCEDURE UpdateCustomerDetails(p\_customer\_id NUMBER, p\_name VARCHAR2, p\_dob DATE) IS

BEGIN

UPDATE Customers

SET Name = p\_name, DOB = p\_dob, LastModified = SYSDATE

WHERE CustomerID = p\_customer\_id;

END UpdateCustomerDetails;

FUNCTION GetCustomerBalance(p\_customer\_id NUMBER) RETURN NUMBER IS

v\_balance NUMBER;

BEGIN

SELECT Balance INTO v\_balance

FROM Customers

WHERE CustomerID = p\_customer\_id;

RETURN v\_balance;

END GetCustomerBalance;

END CustomerManagement;

/

**SCENARIO-2**

CREATE OR REPLACE PACKAGE EmployeeManagement AS

PROCEDURE HireEmployee(p\_employee\_id NUMBER, p\_name VARCHAR2, p\_position VARCHAR2, p\_salary NUMBER, p\_department VARCHAR2, p\_hire\_date DATE);

PROCEDURE UpdateEmployeeDetails(p\_employee\_id NUMBER, p\_name VARCHAR2, p\_position VARCHAR2, p\_salary NUMBER, p\_department VARCHAR2);

FUNCTION CalculateAnnualSalary(p\_employee\_id NUMBER) RETURN NUMBER;

END EmployeeManagement;

/

CREATE OR REPLACE PACKAGE BODY EmployeeManagement AS

PROCEDURE HireEmployee(p\_employee\_id NUMBER, p\_name VARCHAR2, p\_position VARCHAR2, p\_salary NUMBER, p\_department VARCHAR2, p\_hire\_date DATE) IS

BEGIN

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

VALUES (p\_employee\_id, p\_name, p\_position, p\_salary, p\_department, p\_hire\_date);

END HireEmployee;

PROCEDURE UpdateEmployeeDetails(p\_employee\_id NUMBER, p\_name VARCHAR2, p\_position VARCHAR2, p\_salary NUMBER, p\_department VARCHAR2) IS

BEGIN

UPDATE Employees

SET Name = p\_name, Position = p\_position, Salary = p\_salary, Department = p\_department

WHERE EmployeeID = p\_employee\_id;

END UpdateEmployeeDetails;

FUNCTION CalculateAnnualSalary(p\_employee\_id NUMBER) RETURN NUMBER IS

v\_salary NUMBER;

BEGIN

SELECT Salary INTO v\_salary

FROM Employees

WHERE EmployeeID = p\_employee\_id;

RETURN v\_salary \* 12; -- Assuming monthly salary

END CalculateAnnualSalary;

END EmployeeManagement;

/

**SCENARIO-3**

CREATE OR REPLACE PACKAGE AccountOperations AS

PROCEDURE OpenAccount(p\_account\_id NUMBER, p\_customer\_id NUMBER, p\_account\_type VARCHAR2, p\_balance NUMBER);

PROCEDURE CloseAccount(p\_account\_id NUMBER);

FUNCTION GetTotalBalance(p\_customer\_id NUMBER) RETURN NUMBER;

END AccountOperations;

/

CREATE OR REPLACE PACKAGE BODY AccountOperations AS

PROCEDURE OpenAccount(p\_account\_id NUMBER, p\_customer\_id NUMBER, p\_account\_type VARCHAR2, p\_balance NUMBER) IS

BEGIN

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

VALUES (p\_account\_id, p\_customer\_id, p\_account\_type, p\_balance, SYSDATE);

END OpenAccount;

PROCEDURE CloseAccount(p\_account\_id NUMBER) IS

BEGIN

DELETE FROM Accounts

WHERE AccountID = p\_account\_id;

END CloseAccount;

FUNCTION GetTotalBalance(p\_customer\_id NUMBER) RETURN NUMBER IS

v\_total\_balance NUMBER;

BEGIN

SELECT SUM(Balance) INTO v\_total\_balance

FROM Accounts

WHERE CustomerID = p\_customer\_id;

RETURN v\_total\_balance;

END GetTotalBalance;

END AccountOperations;

/

**SCHEMA NEEDED:**

CREATE TABLE Customers (

CustomerID NUMBER PRIMARY KEY,

Name VARCHAR2(100),

DOB DATE,

Balance NUMBER,

LastModified DATE

);

CREATE TABLE Accounts (

AccountID NUMBER PRIMARY KEY,

CustomerID NUMBER,

AccountType VARCHAR2(20),

Balance NUMBER,

LastModified DATE,

FOREIGN KEY (CustomerID) REFERENCES Customers(CustomerID)

);

CREATE TABLE Transactions (

TransactionID NUMBER PRIMARY KEY,

AccountID NUMBER,

TransactionDate DATE,

Amount NUMBER,

TransactionType VARCHAR2(10),

FOREIGN KEY (AccountID) REFERENCES Accounts(AccountID)

);

CREATE TABLE Loans (

LoanID NUMBER PRIMARY KEY,

CustomerID NUMBER,

LoanAmount NUMBER,

InterestRate NUMBER,

StartDate DATE,

EndDate DATE,

FOREIGN KEY (CustomerID) REFERENCES Customers(CustomerID)

);

CREATE TABLE Employees (

EmployeeID NUMBER PRIMARY KEY,

Name VARCHAR2(100),

Position VARCHAR2(50),

Salary NUMBER,

Department VARCHAR2(50),

HireDate DATE

);

CREATE TABLE AuditLog (

LogID NUMBER PRIMARY KEY,

TransactionID NUMBER,

AccountID NUMBER,

TransactionDate DATE,

Amount NUMBER,

TransactionType VARCHAR2(10),

LoggedAt DATE

);

**DATA INSERTED:**

INSERT INTO Customers (CustomerID, Name, DOB, Balance, LastModified) VALUES (1, 'Mitra Sri', TO\_DATE('1960-05-15', 'YYYY-MM-DD'), 5000, SYSDATE);

INSERT INTO Customers (CustomerID, Name, DOB, Balance, LastModified) VALUES (2, 'Aisha Ameen', TO\_DATE('1980-08-20', 'YYYY-MM-DD'), 10000, SYSDATE);

INSERT INTO Customers (CustomerID, Name, DOB, Balance, LastModified) VALUES (3, 'Deborah Jacob', TO\_DATE('1990-12-25', 'YYYY-MM-DD'), 7500, SYSDATE);

INSERT INTO Accounts (AccountID, CustomerID, AccountType, Balance, LastModified) VALUES (1, 1, 'Savings', 5000, SYSDATE);

INSERT INTO Accounts (AccountID, CustomerID, AccountType, Balance, LastModified) VALUES (2, 2, 'Savings', 10000, SYSDATE);

INSERT INTO Accounts (AccountID, CustomerID, AccountType, Balance, LastModified) VALUES (3, 3, 'Checking', 7500, SYSDATE);

INSERT INTO Transactions (TransactionID, AccountID, TransactionDate, Amount, TransactionType) VALUES (1, 1, SYSDATE-5, 100, 'Deposit');

INSERT INTO Transactions (TransactionID, AccountID, TransactionDate, Amount, TransactionType) VALUES (2, 2, SYSDATE-10, 200, 'Withdrawal');

INSERT INTO Transactions (TransactionID, AccountID, TransactionDate, Amount, TransactionType) VALUES (3, 3, SYSDATE-20, 300, 'Deposit');

INSERT INTO Loans (LoanID, CustomerID, LoanAmount, InterestRate, StartDate, EndDate) VALUES (1, 1, 10000, 5.5, SYSDATE-365, SYSDATE+730);

INSERT INTO Loans (LoanID, CustomerID, LoanAmount, InterestRate, StartDate, EndDate) VALUES (2, 2, 20000, 6.0, SYSDATE-730, SYSDATE+365);

INSERT INTO Loans (LoanID, CustomerID, LoanAmount, InterestRate, StartDate, EndDate) VALUES (3, 3, 15000, 5.0, SYSDATE-1095, SYSDATE+1095);

INSERT INTO Employees (EmployeeID, Name, Position, Salary, Department, HireDate) VALUES (1, 'Martha Stewart', 'Manager', 80000, 'Finance', TO\_DATE('2010-02-15', 'YYYY-MM-DD'));

INSERT INTO Employees (EmployeeID, Name, Position, Salary, Department, HireDate) VALUES (2, 'Joyce David', 'Analyst', 60000, 'IT', TO\_DATE('2015-06-20', 'YYYY-MM-DD'));

INSERT INTO Employees (EmployeeID, Name, Position, Salary, Department, HireDate) VALUES (3, 'Mary Clark', 'Clerk', 40000, 'HR', TO\_DATE('2020-01-05', 'YYYY-MM-DD'));