# PL/SQL Assignment Solutions

**Exercise 1 - Control Structures**

-- Scenario 1

BEGIN

FOR c IN (SELECT \* FROM Customers) LOOP

IF MONTHS\_BETWEEN(SYSDATE, c.DOB)/12 > 60 THEN

UPDATE Loans SET InterestRate = InterestRate - 1 WHERE CustomerID = c.CustomerID; END IF;

## END LOOP;

END;

/

-- Scenario 2

BEGIN

FOR c IN (SELECT \* FROM Customers) LOOP IF c.Balance > 10000 THEN

UPDATE Customers SET IsVIP = TRUE WHERE CustomerID = c.CustomerID; END IF;

## END LOOP;

END;

/

-- Scenario 3

BEGIN

FOR l IN (SELECT \* FROM Loans WHERE EndDate <= SYSDATE + 30) LOOP

DBMS\_OUTPUT.PUT\_LINE('Reminder: Loan ID ' || l.LoanID || ' for Customer ' || l.CustomerID || ' is due soon.'); END LOOP;

END;

**Exercise 2 - Error Handling**

-- Scenario 1

CREATE OR REPLACE PROCEDURE SafeTransferFunds(p\_from NUMBER, p\_to NUMBER, p\_amount NUMBER) IS BEGIN

UPDATE Accounts SET Balance = Balance - p\_amount WHERE AccountID = p\_from AND Balance >= p\_amount; UPDATE Accounts SET Balance = Balance + p\_amount WHERE AccountID = p\_to;

COMMIT;

EXCEPTION

WHEN OTHERS THEN ROLLBACK;

INSERT INTO Transactions VALUES (NULL, p\_from, SYSDATE, p\_amount, 'Error'); DBMS\_OUTPUT.PUT\_LINE('Error in fund transfer: ' || SQLERRM);

## END;

/

-- Scenario 2

CREATE OR REPLACE PROCEDURE UpdateSalary(p\_empid NUMBER, p\_percent NUMBER) IS BEGIN

UPDATE Employees SET Salary = Salary + (Salary \* p\_percent / 100) WHERE EmployeeID = p\_empid; IF SQL%ROWCOUNT = 0 THEN

RAISE\_APPLICATION\_ERROR(-20001, 'Employee not found.'); END IF;

## EXCEPTION

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

## END;

/

-- Scenario 3

CREATE OR REPLACE PROCEDURE AddNewCustomer(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); EXCEPTION

## WHEN DUP\_VAL\_ON\_INDEX THEN

DBMS\_OUTPUT.PUT\_LINE('Error: Customer already exists.'); END;

/

CREATE OR REPLACE PROCEDURE ProcessMonthlyInterest IS BEGIN

UPDATE Accounts SET Balance = Balance \* 1.01 WHERE AccountType = 'Savings'; END;

/

-- Scenario 2

CREATE OR REPLACE PROCEDURE UpdateEmployeeBonus(p\_dept VARCHAR2, p\_bonus NUMBER) IS BEGIN

UPDATE Employees SET Salary = Salary + (Salary \* p\_bonus / 100) WHERE Department = p\_dept; END;

/

-- Scenario 3

CREATE OR REPLACE PROCEDURE TransferFunds(p\_from NUMBER, p\_to NUMBER, p\_amount NUMBER) IS v\_balance NUMBER;

## BEGIN

SELECT Balance INTO v\_balance FROM Accounts WHERE AccountID = p\_from; IF v\_balance >= p\_amount THEN

UPDATE Accounts SET Balance = Balance - p\_amount WHERE AccountID = p\_from; UPDATE Accounts SET Balance = Balance + p\_amount WHERE AccountID = p\_to; ELSE

RAISE\_APPLICATION\_ERROR(-20001, 'Insufficient funds'); END IF;

## END;

/

CREATE OR REPLACE FUNCTION CalculateAge(p\_dob DATE) RETURN NUMBER IS BEGIN

RETURN TRUNC(MONTHS\_BETWEEN(SYSDATE, p\_dob) / 12); END;

/

-- Scenario 2

CREATE OR REPLACE FUNCTION CalculateMonthlyInstallment(p\_amount NUMBER, p\_rate NUMBER, p\_years NUMBER) RETURN NUMBER IS

v\_months NUMBER := p\_years \* 12; v\_monthly\_rate NUMBER := p\_rate / 1200; BEGIN

RETURN (p\_amount \* v\_monthly\_rate) / (1 - POWER(1 + v\_monthly\_rate, -v\_months)); END;

/

-- Scenario 3

CREATE OR REPLACE FUNCTION HasSufficientBalance(p\_account\_id NUMBER, p\_amount NUMBER) RETURN BOOLEAN IS

v\_balance NUMBER;

## BEGIN

SELECT Balance INTO v\_balance FROM Accounts WHERE AccountID = p\_account\_id; RETURN v\_balance >= p\_amount;

## END;

/

CREATE OR REPLACE TRIGGER UpdateCustomerLastModified BEFORE UPDATE ON Customers

## FOR EACH ROW BEGIN

:NEW.LastModified := SYSDATE; END;

/

-- Scenario 2

CREATE OR REPLACE TRIGGER LogTransaction AFTER INSERT ON Transactions

## FOR EACH ROW BEGIN

INSERT INTO AuditLog VALUES (NEW.TransactionID, SYSDATE, 'Inserted'); END;

/

-- 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; IF :NEW.TransactionType = 'Withdrawal' AND :NEW.Amount > v\_balance THEN RAISE\_APPLICATION\_ERROR(-20001, 'Withdrawal exceeds balance');

ELSIF :NEW.TransactionType = 'Deposit' AND :NEW.Amount <= 0 THEN RAISE\_APPLICATION\_ERROR(-20002, 'Deposit must be positive'); END IF;

## END;

/

# Exercise 6 - Cursors

-- Scenario 1 DECLARE

CURSOR txn\_cur IS SELECT \* FROM Transactions WHERE TransactionDate >= TRUNC(SYSDATE, 'MM'); BEGIN

FOR t IN txn\_cur LOOP

DBMS\_OUTPUT.PUT\_LINE('Transaction ID: ' || t.TransactionID || ', Amount: ' || t.Amount); END LOOP;

## END;

/

-- Scenario 2 DECLARE

CURSOR acc\_cur IS SELECT AccountID, Balance FROM Accounts; BEGIN

FOR a IN acc\_cur LOOP

UPDATE Accounts SET Balance = Balance - 100 WHERE AccountID = a.AccountID; END LOOP;

## END;

/

-- Scenario 3 DECLARE

CURSOR loan\_cur IS SELECT LoanID, InterestRate FROM Loans; BEGIN

FOR l IN loan\_cur LOOP

UPDATE Loans SET InterestRate = l.InterestRate \* 0.95 WHERE LoanID = l.LoanID; END LOOP;

## END;

/

CREATE OR REPLACE PACKAGE CustomerManagement IS

PROCEDURE AddCustomer(p\_id NUMBER, p\_name VARCHAR2, p\_dob DATE, p\_balance NUMBER); PROCEDURE UpdateCustomer(p\_id NUMBER, p\_name VARCHAR2);

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

/

CREATE OR REPLACE PACKAGE BODY CustomerManagement IS

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); END;

PROCEDURE UpdateCustomer(p\_id NUMBER, p\_name VARCHAR2) IS BEGIN

UPDATE Customers SET Name = p\_name WHERE CustomerID = p\_id; 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;

/

-- Scenario 2

CREATE OR REPLACE PACKAGE EmployeeManagement IS

PROCEDURE HireEmployee(p\_id NUMBER, p\_name VARCHAR2, p\_salary NUMBER, p\_dept VARCHAR2); PROCEDURE UpdateEmployee(p\_id NUMBER, p\_salary NUMBER);

FUNCTION GetAnnualSalary(p\_id NUMBER) RETURN NUMBER; END;

/

CREATE OR REPLACE PACKAGE BODY EmployeeManagement IS

PROCEDURE HireEmployee(p\_id NUMBER, p\_name VARCHAR2, p\_salary NUMBER, p\_dept VARCHAR2) IS BEGIN

INSERT INTO Employees VALUES (p\_id, p\_name, 'New', p\_salary, p\_dept, SYSDATE); END;

PROCEDURE UpdateEmployee(p\_id NUMBER, p\_salary NUMBER) IS BEGIN

UPDATE Employees SET Salary = p\_salary WHERE EmployeeID = p\_id; 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;

/

-- Scenario 3

CREATE OR REPLACE PACKAGE AccountOperations IS

PROCEDURE OpenAccount(p\_id NUMBER, p\_custid NUMBER, p\_type VARCHAR2, p\_balance NUMBER); PROCEDURE CloseAccount(p\_id NUMBER);

FUNCTION GetTotalBalance(p\_custid NUMBER) RETURN NUMBER; END;

/

CREATE OR REPLACE PACKAGE BODY AccountOperations IS

PROCEDURE OpenAccount(p\_id NUMBER, p\_custid NUMBER, p\_type VARCHAR2, p\_balance NUMBER) IS BEGIN

INSERT INTO Accounts VALUES (p\_id, p\_custid, p\_type, p\_balance, SYSDATE); END;

PROCEDURE CloseAccount(p\_id NUMBER) IS BEGIN

DELETE FROM Accounts WHERE AccountID = p\_id; END;

FUNCTION GetTotalBalance(p\_custid NUMBER) RETURN NUMBER IS v\_total NUMBER;

## BEGIN

SELECT SUM(Balance) INTO v\_total FROM Accounts WHERE CustomerID = p\_custid; RETURN v\_total;

## END;

END;

/