**COGNIZANT WEEK 2 PL/SQL**

**Exercise 1: Control Structures**

**Scenario 1:**

DECLARE CURSOR cust\_cursor IS SELECT c.CustomerID, l.LoanID, l.InterestRate FROM Customers c JOIN Loans l ON c.CustomerID = l.CustomerID WHERE EXTRACT(YEAR FROM SYSDATE) - EXTRACT(YEAR FROM c.DOB) > 60;

BEGIN FOR cust\_rec IN cust\_cursor LOOP UPDATE Loans SET InterestRate = InterestRate - (InterestRate \* 0.01) WHERE LoanID = cust\_rec.LoanID;

END LOOP;

COMMIT;

END;

/

**Scenario 2:**

ALTER TABLE Customers ADD IsVIP CHAR(1) DEFAULT 'N';

DECLARE CURSOR cust\_cursor IS SELECT CustomerID FROM Customers WHERE Balance > 10000;

BEGIN FOR cust\_rec IN cust\_cursor LOOP UPDATE Customers SET IsVIP = 'Y' WHERE CustomerID = cust\_rec.CustomerID;

END LOOP;

COMMIT;

END;

/

**Scenario 3:**

DECLARE CURSOR loan\_cursor IS SELECT l.LoanID, l.CustomerID, c.Name, l.EndDate FROM Loans l JOIN Customers c ON l.CustomerID = c.CustomerID WHERE l.EndDate BETWEEN SYSDATE AND SYSDATE + 30;

BEGIN FOR loan\_rec IN loan\_cursor LOOP DBMS\_OUTPUT.PUT\_LINE('Reminder: Dear ' || loan\_rec.Name || ', your loan with ID ' || loan\_rec.LoanID || ' is due on ' || TO\_CHAR(loan\_rec.EndDate, 'YYYY-MM-DD') || '.');

END LOOP;

END;

/

**Exercise 2: Error Handling**

**Scenario 1:**

CREATE OR REPLACE PROCEDURE SafeTransferFunds ( p\_SourceAccountID IN NUMBER, p\_TargetAccountID IN NUMBER, p\_Amount IN NUMBER ) IS e\_InsufficientFunds EXCEPTION; v\_SourceBalance NUMBER;

BEGIN -- Check source account balance SELECT Balance INTO v\_SourceBalance FROM Accounts WHERE AccountID = p\_SourceAccountID FOR UPDATE;

-- Raise exception if insufficient funds

IF v\_SourceBalance < p\_Amount THEN RAISE e\_InsufficientFunds;

END IF;

-- Deduct amount from source account

UPDATE Accounts SET Balance = Balance - p\_Amount WHERE AccountID = p\_SourceAccountID;

-- Add amount to target account

UPDATE Accounts SET Balance = Balance + p\_Amount WHERE AccountID = p\_TargetAccountID; COMMIT;

EXCEPTION WHEN e\_InsufficientFunds THEN ROLLBACK;

DBMS\_OUTPUT.PUT\_LINE('Error: Insufficient funds in source account.');

WHEN OTHERS THEN ROLLBACK;

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

END SafeTransferFunds;

/

**Scenario 2:**

CREATE OR REPLACE PROCEDURE UpdateSalary ( p\_EmployeeID IN NUMBER, p\_Percentage IN NUMBER ) IS v\_Salary NUMBER;

BEGIN -- Update employee salary UPDATE Employees SET Salary = Salary + (Salary \* p\_Percentage / 100) WHERE EmployeeID = p\_EmployeeID;

-- Check if update was successful

IF SQL%ROWCOUNT = 0 THEN RAISE\_APPLICATION\_ERROR(-20001, 'Employee ID not found.');

END IF;

COMMIT;

EXCEPTION WHEN OTHERS THEN ROLLBACK;

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

END UpdateSalary;

/

**Scenario 3:**

CREATE OR REPLACE PROCEDURE AddNewCustomer ( p\_CustomerID IN NUMBER, p\_Name IN VARCHAR2, p\_DOB IN DATE, p\_Balance IN NUMBER )

IS BEGIN

-- Insert new customer 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 ROLLBACK;

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

WHEN OTHERS THEN ROLLBACK;

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

END AddNewCustomer;

/

**Exercise 3: Stored Procedures**

**Scenario 1:**

CREATE OR REPLACE PROCEDURE ProcessMonthlyInterest

IS BEGIN UPDATE Accounts SET Balance = Balance + (Balance \* 0.01) WHERE AccountType = 'Savings'; COMMIT;

END ProcessMonthlyInterest;

/

**Scenario 2:**

CREATE OR REPLACE PROCEDURE UpdateEmployeeBonus ( p\_Department IN VARCHAR2, p\_BonusPercentage IN NUMBER )

IS BEGIN UPDATE Employees SET Salary = Salary + (Salary \* p\_BonusPercentage / 100) WHERE Department = p\_Department;

COMMIT;

EXCEPTION WHEN OTHERS THEN ROLLBACK;

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

END UpdateEmployeeBonus;

/

**Scenario 3:**

CREATE OR REPLACE PROCEDURE TransferFunds ( p\_SourceAccountID IN NUMBER, p\_TargetAccountID IN NUMBER, p\_Amount IN NUMBER ) IS e\_InsufficientFunds EXCEPTION; v\_SourceBalance NUMBER; BEGIN

-- Check source account balance

SELECT Balance INTO v\_SourceBalance FROM Accounts WHERE AccountID = p\_SourceAccountID FOR UPDATE;

-- Raise exception if insufficient funds

IF v\_SourceBalance < p\_Amount THEN RAISE e\_InsufficientFunds;

END IF;

-- Deduct amount from source account

UPDATE Accounts SET Balance = Balance - p\_Amount WHERE AccountID = p\_SourceAccountID;

-- Add amount to target account

UPDATE Accounts SET Balance = Balance + p\_Amount WHERE AccountID = p\_TargetAccountID; COMMIT;

EXCEPTION WHEN e\_InsufficientFunds THEN ROLLBACK;

DBMS\_OUTPUT.PUT\_LINE('Error: Insufficient funds in source account.');

WHEN OTHERS THEN ROLLBACK;

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

END TransferFunds;

/

**Exercise 4: Functions**

**Scenario 1:**

CREATE OR REPLACE FUNCTION CalculateAge ( p\_DOB DATE ) RETURN NUMBER IS v\_Age NUMBER; BEGIN v\_Age := FLOOR((SYSDATE - p\_DOB) / 365.25);

RETURN v\_Age;

END CalculateAge;

/

**Scenario 2:**

CREATE OR REPLACE FUNCTION CalculateMonthlyInstallment ( p\_LoanAmount NUMBER, p\_InterestRate NUMBER, p\_LoanDurationYears NUMBER ) RETURN NUMBER IS v\_MonthlyInstallment NUMBER; v\_MonthlyRate NUMBER; v\_TotalPayments NUMBER;

BEGIN

-- Convert annual interest rate to monthly rate

v\_MonthlyRate := p\_InterestRate / 12 / 100;

-- Calculate the total number of monthly payments

v\_TotalPayments := p\_LoanDurationYears \* 12;

-- Calculate monthly installment using the formula for an amortizing loan payment

IF v\_MonthlyRate > 0 THEN v\_MonthlyInstallment := p\_LoanAmount \* v\_MonthlyRate / (1 - POWER(1 + v\_MonthlyRate, -v\_TotalPayments));

ELSE v\_MonthlyInstallment := p\_LoanAmount / v\_TotalPayments;

-- For zero interest rate case END IF;

RETURN v\_MonthlyInstallment;

END CalculateMonthlyInstallment;

/

**Scenario 3:**

CREATE OR REPLACE FUNCTION HasSufficientBalance ( p\_AccountID NUMBER, p\_Amount NUMBER ) RETURN BOOLEAN IS v\_Balance NUMBER;

BEGIN

-- Retrieve the balance for the specified account

SELECT Balance INTO v\_Balance FROM Accounts WHERE AccountID = p\_AccountID;

-- Check if the balance is sufficient

IF v\_Balance >= p\_Amount THEN RETURN TRUE;

ELSE RETURN FALSE;

END IF;

EXCEPTION WHEN NO\_DATA\_FOUND THEN RETURN FALSE;

-- If account ID is not found, return false

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

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), LogDate DATE );

CREATE OR REPLACE TRIGGER LogTransaction AFTER INSERT ON Transactions FOR EACH ROW BEGIN INSERT INTO AuditLog ( LogID, TransactionID, AccountID, TransactionDate, Amount, TransactionType, LogDate ) 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

-- Retrieve the current balance of the account

SELECT Balance INTO v\_Balance FROM Accounts WHERE AccountID = :NEW.AccountID FOR UPDATE;

-- Check for positive deposits

IF :NEW.TransactionType = 'Deposit' AND :NEW.Amount <= 0 THEN RAISE\_APPLICATION\_ERROR(-20001, 'Deposit amount must be positive.');

END IF;

-- Check for sufficient balance for withdrawals

IF :NEW.TransactionType = 'Withdrawal' AND :NEW.Amount > v\_Balance THEN RAISE\_APPLICATION\_ERROR(-20002, 'Insufficient balance for withdrawal.');

END IF;

END CheckTransactionRules;

/

**Exercise 6: Cursors**

**Scenario 1:**

DECLARE CURSOR cur\_Transactions IS SELECT t.TransactionID, t.AccountID, t.TransactionDate, t.Amount, t.TransactionType, c.CustomerID, c.Name FROM Transactions t JOIN Accounts a ON t.AccountID = a.AccountID JOIN Customers c ON a.CustomerID = c.CustomerID WHERE TRUNC(t.TransactionDate, 'MM') = TRUNC(SYSDATE, 'MM');

v\_TransactionID Transactions.TransactionID%TYPE; v\_AccountID Transactions.AccountID%TYPE; v\_TransactionDate Transactions.TransactionDate%TYPE; v\_Amount Transactions.Amount%TYPE; v\_TransactionType Transactions.TransactionType%TYPE; v\_CustomerID Customers.CustomerID%TYPE; v\_CustomerName Customers.Name%TYPE; BEGIN OPEN cur\_Transactions;

LOOP FETCH cur\_Transactions INTO v\_TransactionID, v\_AccountID, v\_TransactionDate, v\_Amount, v\_TransactionType, v\_CustomerID, v\_CustomerName;

EXIT WHEN cur\_Transactions%NOTFOUND;

DBMS\_OUTPUT.PUT\_LINE('Customer ID: ' || v\_CustomerID); DBMS\_OUTPUT.PUT\_LINE('Customer Name: ' || v\_CustomerName);

DBMS\_OUTPUT.PUT\_LINE('Transaction ID: ' || v\_TransactionID);

DBMS\_OUTPUT.PUT\_LINE('Account ID: ' || v\_AccountID); DBMS\_OUTPUT.PUT\_LINE('Transaction Date: ' || v\_TransactionDate);

DBMS\_OUTPUT.PUT\_LINE('Amount: ' || v\_Amount); DBMS\_OUTPUT.PUT\_LINE('Transaction Type: ' || v\_TransactionType);

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

END LOOP;

CLOSE cur\_Transactions;

END;

/

**Scenario 2:**

DECLARE CURSOR cur\_Accounts IS SELECT AccountID, Balance FROM Accounts; v\_AccountID Accounts.AccountID%TYPE; v\_Balance Accounts.Balance%TYPE; v\_AnnualFee CONSTANT NUMBER := 100; -- Adjust the fee amount as needed

BEGIN OPEN cur\_Accounts; LOOP FETCH cur\_Accounts INTO v\_AccountID, v\_Balance;

EXIT WHEN cur\_Accounts%NOTFOUND;

-- Deduct the annual fee

UPDATE Accounts SET Balance = Balance - v\_AnnualFee WHERE AccountID = v\_AccountID; DBMS\_OUTPUT.PUT\_LINE('Applied annual fee to Account ID: ' || v\_AccountID);

END LOOP;

CLOSE cur\_Accounts;

COMMIT;

END;

/

**Scenario 3:**

DECLARE CURSOR cur\_Loans IS SELECT LoanID, InterestRate FROM Loans; v\_LoanID Loans.LoanID%TYPE; v\_InterestRate Loans.InterestRate%TYPE; v\_InterestRateIncrease CONSTANT NUMBER := 0.5; -- New policy: increase by 0.5%

BEGIN OPEN cur\_Loans; LOOP FETCH cur\_Loans INTO v\_LoanID, v\_InterestRate;

EXIT WHEN cur\_Loans%NOTFOUND;

-- Update the interest rate based on the new policy

UPDATE Loans SET InterestRate = v\_InterestRate + v\_InterestRateIncrease WHERE LoanID = v\_LoanID; DBMS\_OUTPUT.PUT\_LINE('Updated interest rate for Loan ID: ' || v\_LoanID);

END LOOP;

CLOSE cur\_Loans;

COMMIT;

END;

/

**Exercise 7: Packages**

**Scenario 1:**

CREATE OR REPLACE PACKAGE CustomerManagement AS PROCEDURE AddNewCustomer ( p\_CustomerID IN NUMBER, p\_Name IN VARCHAR2, p\_DOB IN DATE, p\_Balance IN NUMBER ); PROCEDURE UpdateCustomerDetails ( p\_CustomerID IN NUMBER, p\_Name IN VARCHAR2, p\_DOB IN DATE );

FUNCTION GetCustomerBalance ( p\_CustomerID IN NUMBER ) RETURN NUMBER;

END CustomerManagement;

/

CREATE OR REPLACE PACKAGE BODY CustomerManagement AS PROCEDURE AddNewCustomer ( p\_CustomerID IN NUMBER, p\_Name IN VARCHAR2, p\_DOB IN DATE, p\_Balance IN NUMBER ) IS BEGIN INSERT INTO Customers (CustomerID, Name, DOB, Balance, LastModified) VALUES (p\_CustomerID, p\_Name, p\_DOB, p\_Balance, SYSDATE); EXCEPTION WHEN DUP\_VAL\_ON\_INDEX THEN DBMS\_OUTPUT.PUT\_LINE('Error: Customer with ID ' || p\_CustomerID || ' already exists.');

END AddNewCustomer;

PROCEDURE UpdateCustomerDetails ( p\_CustomerID IN NUMBER, p\_Name IN VARCHAR2, p\_DOB IN DATE ) IS BEGIN UPDATE Customers SET Name = p\_Name, DOB = p\_DOB, LastModified = SYSDATE WHERE CustomerID = p\_CustomerID; EXCEPTION WHEN NO\_DATA\_FOUND THEN DBMS\_OUTPUT.PUT\_LINE('Error: Customer with ID ' || p\_CustomerID || ' does not exist.');

END UpdateCustomerDetails;

FUNCTION GetCustomerBalance ( p\_CustomerID IN NUMBER ) RETURN NUMBER IS v\_Balance NUMBER; BEGIN SELECT Balance INTO v\_Balance FROM Customers WHERE CustomerID = p\_CustomerID;

RETURN v\_Balance;

EXCEPTION WHEN NO\_DATA\_FOUND THEN DBMS\_OUTPUT.PUT\_LINE('Error: Customer with ID ' || p\_CustomerID || ' does not exist.');

RETURN NULL;

END GetCustomerBalance;

END CustomerManagement;

/

**Scenario 2:**

CREATE OR REPLACE PACKAGE EmployeeManagement AS PROCEDURE HireEmployee ( p\_EmployeeID IN NUMBER, p\_Name IN VARCHAR2, p\_Position IN VARCHAR2, p\_Salary IN NUMBER, p\_Department IN VARCHAR2, p\_HireDate IN DATE );

PROCEDURE UpdateEmployeeDetails ( p\_EmployeeID IN NUMBER, p\_Name IN VARCHAR2, p\_Position IN VARCHAR2, p\_Salary IN NUMBER, p\_Department IN VARCHAR2 );

FUNCTION CalculateAnnualSalary ( p\_EmployeeID IN NUMBER ) RETURN NUMBER;

END EmployeeManagement;

/

CREATE OR REPLACE PACKAGE BODY EmployeeManagement AS PROCEDURE HireEmployee ( p\_EmployeeID IN NUMBER, p\_Name IN VARCHAR2, p\_Position IN VARCHAR2, p\_Salary IN NUMBER, p\_Department IN VARCHAR2, p\_HireDate IN DATE ) IS BEGIN INSERT INTO Employees (EmployeeID, Name, Position, Salary, Department, HireDate) VALUES (p\_EmployeeID, p\_Name, p\_Position, p\_Salary, p\_Department, p\_HireDate);

EXCEPTION WHEN DUP\_VAL\_ON\_INDEX THEN DBMS\_OUTPUT.PUT\_LINE('Error: Employee with ID ' || p\_EmployeeID || ' already exists.');

END HireEmployee;

PROCEDURE UpdateEmployeeDetails ( p\_EmployeeID IN NUMBER, p\_Name IN VARCHAR2, p\_Position IN VARCHAR2, p\_Salary IN NUMBER, p\_Department IN VARCHAR2 ) IS BEGIN UPDATE Employees SET Name = p\_Name, Position = p\_Position, Salary = p\_Salary, Department = p\_Department WHERE EmployeeID = p\_EmployeeID;

EXCEPTION WHEN NO\_DATA\_FOUND THEN DBMS\_OUTPUT.PUT\_LINE('Error: Employee with ID ' || p\_EmployeeID || ' does not exist.');

END UpdateEmployeeDetails;

FUNCTION CalculateAnnualSalary ( p\_EmployeeID IN NUMBER ) RETURN NUMBER IS v\_Salary NUMBER; v\_AnnualSalary NUMBER;

BEGIN SELECT Salary INTO v\_Salary FROM Employees WHERE EmployeeID = p\_EmployeeID; v\_AnnualSalary := v\_Salary \* 12; RETURN v\_AnnualSalary; EXCEPTION WHEN NO\_DATA\_FOUND THEN DBMS\_OUTPUT.PUT\_LINE('Error: Employee with ID ' || p\_EmployeeID || ' does not exist.');

RETURN NULL;

END CalculateAnnualSalary;

END EmployeeManagement;

/

**Scenario 3:**

CREATE OR REPLACE PACKAGE AccountOperations AS PROCEDURE OpenNewAccount ( p\_AccountID IN NUMBER, p\_CustomerID IN NUMBER, p\_AccountType IN VARCHAR2, p\_Balance IN NUMBER ); PROCEDURE CloseAccount ( p\_AccountID IN NUMBER );

FUNCTION GetTotalBalance ( p\_CustomerID IN NUMBER ) RETURN NUMBER;

END AccountOperations;

/

CREATE OR REPLACE PACKAGE BODY AccountOperations AS PROCEDURE OpenNewAccount ( p\_AccountID IN NUMBER, p\_CustomerID IN NUMBER, p\_AccountType IN VARCHAR2, p\_Balance IN NUMBER ) IS BEGIN INSERT INTO Accounts (AccountID, CustomerID, AccountType, Balance, LastModified) VALUES (p\_AccountID, p\_CustomerID, p\_AccountType, p\_Balance, SYSDATE);

EXCEPTION WHEN DUP\_VAL\_ON\_INDEX THEN DBMS\_OUTPUT.PUT\_LINE('Error: Account with ID ' || p\_AccountID || ' already exists.');

END OpenNewAccount;

PROCEDURE CloseAccount ( p\_AccountID IN NUMBER ) IS BEGIN DELETE FROM Accounts WHERE AccountID = p\_AccountID;

EXCEPTION WHEN NO\_DATA\_FOUND THEN DBMS\_OUTPUT.PUT\_LINE('Error: Account with ID ' || p\_AccountID || ' does not exist.');

END CloseAccount; FUNCTION GetTotalBalance ( p\_CustomerID IN NUMBER ) RETURN NUMBER IS v\_TotalBalance NUMBER;

BEGIN SELECT SUM(Balance) INTO v\_TotalBalance FROM Accounts WHERE CustomerID = p\_CustomerID; RETURN v\_TotalBalance;

EXCEPTION WHEN NO\_DATA\_FOUND THEN DBMS\_OUTPUT.PUT\_LINE('Error: Customer with ID ' || p\_CustomerID || ' does not exist.');

RETURN NULL;

END GetTotalBalance;

END AccountOperations;

/