**Week-2 PL/SQL**

Solution by: Mahika Thakur, CSE 7TH Semester, MCKV Institute of Engineering

**Exercise 1: Control Structures**

**Scenario 1:**

**Answer:**

DECLARE

cust\_id Customers.CustomerID%TYPE;

cust\_dob Customers.DOB%TYPE;

cust\_age NUMBER;

loan\_interest\_rate Loans.InterestRate%TYPE;

BEGIN

FOR customer\_record IN (SELECT CustomerID, DOB FROM Customers) LOOP

cust\_id := customer\_record.CustomerID;

cust\_dob := customer\_record.DOB;

cust\_age := FLOOR(MONTHS\_BETWEEN(SYSDATE, cust\_dob) / 12);

IF cust\_age > 60 THEN

FOR loan\_record IN (SELECT LoanID, InterestRate FROM Loans WHERE CustomerID = cust\_id) LOOP

loan\_interest\_rate := loan\_record.InterestRate;

-- Apply 1% discount

UPDATE Loans

SET InterestRate = loan\_interest\_rate - 1

WHERE LoanID = loan\_record.LoanID;

END LOOP;

END IF;

END LOOP;

COMMIT;

END;

/

**Scenario 2:**

**Answer:**

DECLARE

balance Customers.Balance%TYPE;

BEGIN

FOR customer\_record IN (SELECT CustomerID, Balance FROM Customers) LOOP

balance := customer\_record.Balance;

IF balance > 10000 THEN

UPDATE Customers

SET IsVIP = 'TRUE' -- Ensure there is an IsVIP column in the Customers table

WHERE CustomerID = customer\_record.CustomerID;

END IF;

END LOOP;

COMMIT;

END;

/

**Scenario 3**

**Answer:**

dDECLARE

CURSOR loan\_cursor IS

SELECT l.LoanID, c.CustomerID, c.Name, l.EndDate

FROM Loans l

JOIN Customers c ON l.CustomerID = c.CustomerID

WHERE l.EndDate BETWEEN SYSDATE AND SYSDATE + 30;

customer\_name Customers.Name%TYPE;

end\_date Loans.EndDate%TYPE;

BEGIN

FOR loan\_record IN loan\_cursor LOOP

customer\_name := loan\_record.Name;

end\_date := loan\_record.EndDate;

-- Print reminder message

DBMS\_OUTPUT.PUT\_LINE('Reminder: Loan ID ' || loan\_record.LoanID || ' for customer ' || customer\_name || ' is due on ' || TO\_CHAR(end\_date, 'YYYY-MM-DD') || '.');

END LOOP;

END;

/

**Exercise 2: Error Handling**

**Scenario 1:**

**Answer:**

CREATE OR REPLACE PROCEDURE SafeTransferFunds (

p\_from\_account\_id IN NUMBER,

p\_to\_account\_id IN NUMBER,

p\_amount IN NUMBER

) AS

v\_from\_balance Accounts.Balance%TYPE;

v\_to\_balance Accounts.Balance%TYPE;

BEGIN

BEGIN

SELECT Balance INTO v\_from\_balance FROM Accounts WHERE AccountID = p\_from\_account\_id FOR UPDATE;

SELECT Balance INTO v\_to\_balance FROM Accounts WHERE AccountID = p\_to\_account\_id FOR UPDATE;

IF v\_from\_balance < p\_amount THEN

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

END IF;

-- Update balances

UPDATE Accounts

SET Balance = v\_from\_balance - p\_amount

WHERE AccountID = p\_from\_account\_id;

UPDATE Accounts

SET Balance = v\_to\_balance + p\_amount

WHERE AccountID = p\_to\_account\_id;

-- Commit transaction

COMMIT;

EXCEPTION

WHEN OTHERS THEN

-- Rollback transaction in case of error

ROLLBACK;

-- Log the error message

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

END;

END SafeTransferFunds;

/

**Scenario 2:**

**Answer:**

CREATE OR REPLACE PROCEDURE UpdateSalary (

p\_employee\_id IN NUMBER,

p\_percentage IN NUMBER

) AS

v\_current\_salary Employees.Salary%TYPE;

BEGIN

-- Start transaction

BEGIN

-- Retrieve current salary

SELECT Salary INTO v\_current\_salary FROM Employees WHERE EmployeeID = p\_employee\_id;

-- Update salary

UPDATE Employees

SET Salary = v\_current\_salary \* (1 + p\_percentage / 100)

WHERE EmployeeID = p\_employee\_id;

-- Commit transaction

COMMIT;

EXCEPTION

WHEN NO\_DATA\_FOUND THEN

-- Handle case where employee ID does not exist

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

ROLLBACK;

WHEN OTHERS THEN

-- Handle other errors

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

ROLLBACK;

END;

END UpdateSalary;

/

**Scenario 3:**

**Answer:**

CREATE OR REPLACE PROCEDURE AddNewCustomer (

    p\_customer\_id IN NUMBER,

    p\_name IN VARCHAR2,

    p\_dob IN DATE,

    p\_balance IN NUMBER

) AS

BEGIN

    -- Start transaction

    BEGIN

        -- Attempt to insert new customer

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

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

        -- Commit transaction

        COMMIT;

    EXCEPTION

        WHEN DUP\_VAL\_ON\_INDEX THEN

            -- Handle case where customer ID already exists

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

            ROLLBACK;

        WHEN OTHERS THEN

            -- Handle other errors

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

            ROLLBACK;

    END;

END AddNewCustomer;

/

**Exercise 3: Stored Procedures**

**Scenario 1:**

**Answer:**

CREATE OR REPLACE PROCEDURE ProcessMonthlyInterest AS

BEGIN

    -- Update balances for savings accounts

    UPDATE Accounts

    SET Balance = Balance \* 1.01

    WHERE AccountType = 'Savings';

    -- Commit changes

    COMMIT;

END ProcessMonthlyInterest;

/

**Scenario 2:**

**Answer:**

CREATE OR REPLACE PROCEDURE UpdateEmployeeBonus (

    p\_department IN VARCHAR2,

    p\_bonus\_percentage IN NUMBER

) AS

BEGIN

    -- Update employee salaries

    UPDATE Employees

    SET Salary = Salary \* (1 + p\_bonus\_percentage / 100)

    WHERE Department = p\_department;

    -- Commit changes

    COMMIT;

END UpdateEmployeeBonus;

/

**Scenario 3:**

**Answer:**

CREATE OR REPLACE PROCEDURE TransferFunds (

    pfrom\_account\_id IN NUMBER,

    pto\_account\_id IN NUMBER,

    pamount IN NUMBER

) AS

    vfrom\_balance Accounts.Balance%TYPE;

    vto\_balance Accounts.Balance%TYPE;

BEGIN

    -- Start transaction

    BEGIN

        -- Lock the accounts to prevent concurrent modifications

        SELECT Balance INTO vfrom\_balance FROM Accounts WHERE AccountID = pfrom\_account\_id FOR UPDATE;

        SELECT Balance INTO vto\_balance FROM Accounts WHERE AccountID = pto\_account\_id FOR UPDATE;

        -- Check for sufficient funds

        IF vfrom\_balance < pamount THEN

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

        END IF;

        -- Update balances

        UPDATE Accounts

        SET Balance = vfrom\_balance - pamount

        WHERE AccountID = pfrom\_account\_id;

        UPDATE Accounts

        SET Balance = vto\_balance + pamount

        WHERE AccountID = pto\_account\_id;

        -- Commit transaction

        COMMIT;

    EXCEPTION

        WHEN OTHERS THEN

            -- Rollback transaction in case of error

            ROLLBACK;

            -- Log the error message

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

    END;

END TransferFunds;

/

**Exercise 4: Functions**

**Scenario 1:**

**Answer:**

CREATE OR REPLACE FUNCTION CalculateAge (

    pdob IN DATE

) RETURN NUMBER AS

    vage NUMBER;

BEGIN

    -- Calculate age

    vage := FLOOR(MONTHS\_BETWEEN(SYSDATE, pdob) / 12);

    RETURN vage;

END CalculateAge;

/

**Scenario 2:**

**Answer:**

CREATE OR REPLACE TRIGGER CheckTransactionRules

BEFORE INSERT ON Transactions

FOR EACH ROW

DECLARE

    v\_balance Accounts.Balance%TYPE;

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(-20002, 'Insufficient funds for withdrawal.');

    ELSIF :NEW.TransactionType = 'Deposit' AND :NEW.Amount <= 0 THEN

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

    END IF;

END CheckTransactionRules;

/

**Scenario 3:**

**Answer:**

CREATE OR REPLACE FUNCTION HasSufficientBalance (

    p\_account\_id IN NUMBER,

    p\_amount IN NUMBER

) RETURN BOOLEAN AS

    v\_balance Accounts.Balance%TYPE;

BEGIN

    -- Retrieve account balance

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

    -- Check if balance is sufficient

    RETURN v\_balance >= p\_amount;

EXCEPTION

    WHEN NO\_DATA\_FOUND THEN

        -- Handle case where account does not exist

        RETURN FALSE;

END HasSufficientBalance;

/

**Exercise 5: Triggers**

**Scenario 1:**

**Answer:**

CREATE OR REPLACE TRIGGER UpdateCustomerLastModified

BEFORE UPDATE ON CUSTOMERS

FOR EACH ROW

BEGIN

    :NEW.LastModified:=SYSDATE;

END UpdateCustomerLastModified

/

**Scenario 2:**

**Answer:**

CREATE TABLE AUDITLOG(

AuditID NUMBER PRIMARY KEY,

TransactionID NUMBER,

ActionTime DATE,

ActionType VARCHAR2(10),

Amount NUMBER);

CREATE SEQUENCE AuditLog\_seq START ITH 1 INCREMENT BY 1;

CREATE OR REPLACE TRIGGER LogTransaction

AFTER INSERT ON Transactions

FOR EACH ROW

BEGIN

    INSERT INTO AuditLog(AuditID,TransactionID,ActionTime,ActionType,Amount)

    VALUES(AuditLog\_seq.NEXTVAL, :NEW.TransactionID,'INSERT',:NEW.Amount);

END LogTransaction

/

**Scenario 3:**

**Answer:**

CREATE OR REPLACE TRIGGER CheckTransactionRules

BEFORE INSERT ON Transactions

FOR EACH ROW

DECLARE

    v\_balance Accounts.Balance%TYPE;

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(-20002, 'Insufficient funds for withdrawal.');

    ELSIF :NEW.TransactionType = 'Deposit' AND :NEW.Amount <= 0 THEN

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

    END IF;

END CheckTransactionRules;

/

**Exercise 6: Cursors**

**Scenario 1:**

**Answer:**

DECLARE

    CURSOR transaction\_cursor IS

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

        FROM Transactions t

        JOIN Accounts a ON t.AccountID = a.AccountID

        JOIN Customers c ON a.CustomerID = c.CustomerID

        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 transaction\_cursor;

    LOOP

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

        EXIT WHEN transaction\_cursor%NOTFOUND;

        -- Print the statement (or use another method to generate the report)

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

                             ', Name: ' || v\_name ||

                             ', Date: ' || TO\_CHAR(v\_transaction\_date, 'YYYY-MM-DD') ||

                             ', Amount: ' || v\_amount ||

                             ', Type: ' || v\_transaction\_type);

    END LOOP;

    CLOSE transaction\_cursor;

END;

**Scenario 2:**

**Answer:**

DECLARE

    CURSOR account\_cursor IS

        SELECT AccountID, Balance

        FROM Accounts;

    v\_account\_id Accounts.AccountID%TYPE;

    v\_balance Accounts.Balance%TYPE;

    v\_annual\_fee CONSTANT NUMBER := 50; -- Example fee amount

BEGIN

    OPEN account\_cursor;

    LOOP

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

        EXIT WHEN account\_cursor%NOTFOUND;

        -- Deduct annual maintenance fee

        UPDATE Accounts

        SET Balance = v\_balance - v\_annual\_fee

        WHERE AccountID = v\_account\_id;

        -- Check if balance is negative

        IF (v\_balance - v\_annual\_fee) < 0 THEN

            DBMS\_OUTPUT.PUT\_LINE('Warning: Account ' || v\_account\_id || ' has insufficient funds after fee deduction.');

        END IF;

    END LOOP;

    CLOSE account\_cursor;

    -- Commit changes

    COMMIT;

END;

/

**Scenario 3:**

**Answer:**

DECLARE

    CURSOR loan\_cursor IS

        SELECT LoanID, InterestRate

        FROM Loans;

    v\_loan\_id Loans.LoanID%TYPE;

    v\_current\_rate Loans.InterestRate%TYPE;

    v\_new\_rate NUMBER;

BEGIN

    OPEN loan\_cursor;

    LOOP

        FETCH loan\_cursor INTO v\_loan\_id, v\_current\_rate;

        EXIT WHEN loan\_cursor%NOTFOUND;

        v\_new\_rate := v\_current\_rate + 0.5; -- Example increase of 0.5%

        UPDATE Loans

        SET InterestRate = v\_new\_rate

        WHERE LoanID = v\_loan\_id;

    END LOOP;

    CLOSE loan\_cursor;

    COMMIT;

END;

/

**Exercise 7: Packages**

**Scenario 1:**

**Answer:**

CREATE OR REPLACE PACKAGE CustomerManagement AS

    PROCEDURE AddNewCustomer(p\_CustomerID NUMBER, p\_Name VARCHAR2, p\_DOB DATE, p\_Balance NUMBER);

    PROCEDURE UpdateCustomerDetails(p\_CustomerID NUMBER, p\_Name VARCHAR2, p\_DOB DATE, p\_Balance NUMBER);

    FUNCTION GetCustomerBalance(p\_CustomerID NUMBER) RETURN NUMBER;

END CustomerManagement;

/

CREATE OR REPLACE PACKAGE BODY CustomerManagement AS

    PROCEDURE AddNewCustomer(p\_CustomerID NUMBER, p\_Name VARCHAR2, p\_DOB DATE, p\_Balance NUMBER) IS

    BEGIN

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

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

    END AddNewCustomer;

    PROCEDURE UpdateCustomerDetails(p\_CustomerID NUMBER, p\_Name VARCHAR2, p\_DOB DATE, p\_Balance NUMBER) IS

    BEGIN

        UPDATE Customers

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

        WHERE CustomerID = p\_CustomerID;

    END UpdateCustomerDetails;

    FUNCTION GetCustomerBalance(p\_CustomerID NUMBER) RETURN NUMBER IS

        v\_Balance NUMBER;

    BEGIN

        SELECT Balance INTO v\_Balance

        FROM Customers

        WHERE CustomerID = p\_CustomerID;

        RETURN v\_Balance;

    END GetCustomerBalance;

END CustomerManagement;

/

**Scenario 2:**

**Answer:**

CREATE OR REPLACE PACKAGE EmployeeManagement AS

    PROCEDURE HireEmployee(p\_EmployeeID NUMBER, p\_Name VARCHAR2, p\_Position VARCHAR2, p\_Salary NUMBER, p\_Department VARCHAR2, p\_HireDate DATE);

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

    FUNCTION CalculateAnnualSalary(p\_EmployeeID NUMBER) RETURN NUMBER;

END EmployeeManagement;

/

CREATE OR REPLACE PACKAGE BODY EmployeeManagement AS

    PROCEDURE HireEmployee(p\_EmployeeID NUMBER, p\_Name VARCHAR2, p\_Position VARCHAR2, p\_Salary NUMBER, p\_Department VARCHAR2, p\_HireDate 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);

    END HireEmployee;

    PROCEDURE UpdateEmployeeDetails(p\_EmployeeID 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\_EmployeeID;

    END UpdateEmployeeDetails;

    FUNCTION CalculateAnnualSalary(p\_EmployeeID NUMBER) RETURN NUMBER IS

        v\_Salary NUMBER;

    BEGIN

        SELECT Salary INTO v\_Salary

        FROM Employees

        WHERE EmployeeID = p\_EmployeeID;

        RETURN v\_Salary \* 12;

    END CalculateAnnualSalary;

END EmployeeManagement;

/

**Scenario 3:**

**Answer:**

CREATE OR REPLACE PACKAGE AccountOperations AS

PROCEDURE OpenAccount(p\_AccountID NUMBER, p\_CustomerID NUMBER, p\_AccountType VARCHAR2, p\_Balance NUMBER);

    PROCEDURE CloseAccount(p\_AccountID NUMBER);

    FUNCTION GetTotalBalance(p\_CustomerID NUMBER) RETURN NUMBER;

END AccountOperations;

/

CREATE OR REPLACE PACKAGE BODY AccountOperations AS

PROCEDURE OpenAccount(p\_AccountID NUMBER, p\_CustomerID NUMBER, p\_AccountType VARCHAR2, p\_Balance NUMBER) IS

BEGIN

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

        VALUES (p\_AccountID, p\_CustomerID, p\_AccountType, p\_Balance, SYSDATE);

    END OpenAccount;

    PROCEDURE CloseAccount(p\_AccountID NUMBER) IS

    BEGIN

        DELETE FROM Accounts

        WHERE AccountID = p\_AccountID;

    END CloseAccount;

    FUNCTION GetTotalBalance(p\_CustomerID NUMBER) RETURN NUMBER IS

        v\_TotalBalance NUMBER;

    BEGIN

        SELECT SUM(Balance) INTO v\_TotalBalance

        FROM Accounts

        WHERE CustomerID = p\_CustomerID;

        RETURN v\_TotalBalance;

    END GetTotalBalance;

END AccountOperations;

/