PL SQL programming

Exercise 1: Control Structures

Scenario 1: Apply 1% Discount for Customers Over 60

```
BEGIN
  FOR cust_rec IN (
    SELECT CustomerID, InterestRate
    FROM Loans
    JOIN Customers ON Loans.CustomerID = Customers.ID
    WHERE Customers.Age > 60
  ) LOOP
    UPDATE Loans
    SET InterestRate = InterestRate - 1
    WHERE CustomerID = cust_rec.CustomerID;
  END LOOP;

  COMMIT;
END;
/
```

Scenario 2: Promote to VIP if Balance > $10,000

```
BEGIN
  FOR cust_rec IN (
    SELECT ID, Balance FROM Customers WHERE Balance > 10000
  ) LOOP
    UPDATE Customers
    SET IsVIP = 'TRUE'
    WHERE ID = cust_rec.ID;
  END LOOP;

  COMMIT;
END;
/
```

Scenario 3: Loan Reminders for Next 30 Days

```
DECLARE
  v_due_date DATE;
BEGIN
  FOR loan_rec IN (
    SELECT CustomerID, LoanID, DueDate
    FROM Loans
    WHERE DueDate BETWEEN SYSDATE AND SYSDATE + 30
```

```
  ) LOOP
    SELECT DueDate INTO v_due_date FROM Loans WHERE LoanID = loan_rec.LoanID;

    DBMS_OUTPUT.PUT_LINE(
      'Reminder: Loan ID ' || loan_rec.LoanID ||
      ' for Customer ID ' || loan_rec.CustomerID ||
      ' is due on ' || TO_CHAR(v_due_date, 'DD-MON-YYYY')
    );
  END LOOP;
END;
/
```

Exercise 2: Error Handling

Scenario 1: SafeTransferFunds – Transfer with Rollback on Error

```
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 NUMBER;
BEGIN
    -- Check sender balance
    SELECT Balance INTO v_from_balance
    FROM Accounts
    WHERE AccountID = p_from_account_id;

    IF v_from_balance < p_amount THEN
        RAISE_APPLICATION_ERROR(-20001, 'Insufficient funds.');
    END IF;

    -- Deduct from sender
    UPDATE Accounts
    SET Balance = Balance - p_amount
    WHERE AccountID = p_from_account_id;

    -- Credit to receiver
    UPDATE Accounts
    SET Balance = Balance + p_amount
    WHERE AccountID = p_to_account_id;

    COMMIT;
EXCEPTION
    WHEN OTHERS THEN
        ROLLBACK;
        INSERT INTO ErrorLog(ErrorTime, ErrorMessage)
        VALUES (SYSDATE, 'Transfer failed: ' || SQLERRM);
        DBMS_OUTPUT.PUT_LINE('Error: ' || SQLERRM);
END;
/
```

Scenario 2: UpdateSalary – Handle Invalid Employee ID

```
CREATE OR REPLACE PROCEDURE UpdateSalary(
   p_emp_id     IN NUMBER,
   p_percent    IN NUMBER
) AS
BEGIN
   -- Try to update the salary
   UPDATE Employees
   SET Salary = Salary + (Salary * p_percent / 100)
   WHERE EmployeeID = p_emp_id;

   -- Check if any row was affected
   IF SQL%ROWCOUNT = 0 THEN
      RAISE_APPLICATION_ERROR(-20002, 'Employee ID not found.');
   END IF;

   COMMIT;
EXCEPTION
   WHEN OTHERS THEN
      ROLLBACK;
      INSERT INTO ErrorLog(ErrorTime, ErrorMessage)
      VALUES (SYSDATE, 'Salary update error: ' || SQLERRM);
      DBMS_OUTPUT.PUT_LINE('Error: ' || SQLERRM);
END;
/
```

Scenario 3: AddNewCustomer – Handle Duplicate Customer ID

```
CREATE OR REPLACE PROCEDURE AddNewCustomer(
   p_customer_id   IN NUMBER,
   p_name          IN VARCHAR2,
   p_age           IN NUMBER,
   p_balance       IN NUMBER
) AS
BEGIN
   INSERT INTO Customers(CustomerID, Name, Age, Balance)
   VALUES (p_customer_id, p_name, p_age, p_balance);

   COMMIT;
EXCEPTION
   WHEN DUP_VAL_ON_INDEX THEN
      ROLLBACK;
      INSERT INTO ErrorLog(ErrorTime, ErrorMessage)
      VALUES (SYSDATE, 'Duplicate Customer ID: ' || p_customer_id);
      DBMS_OUTPUT.PUT_LINE('Error: Customer already exists.');
   WHEN OTHERS THEN
      ROLLBACK;
      INSERT INTO ErrorLog(ErrorTime, ErrorMessage)
```

```
        VALUES (SYSDATE, 'Customer insert error: ' || SQLERRM);
        DBMS_OUTPUT.PUT_LINE('Error: ' || SQLERRM);
END;
/
```

Exercise 3: Stored Procedures

Scenario 1: ProcessMonthlyInterest – Apply 1% Interest to Savings Accounts

```
CREATE OR REPLACE PROCEDURE ProcessMonthlyInterest IS
BEGIN
  -- Apply 1% interest to all savings accounts
  UPDATE Accounts
  SET Balance = Balance + (Balance * 0.01)
  WHERE AccountType = 'SAVINGS';

  COMMIT;
EXCEPTION
  WHEN OTHERS THEN
    ROLLBACK;
    DBMS_OUTPUT.PUT_LINE('Error in processing monthly interest: ' || SQLERRM);
END;
/
```

Scenario 2: UpdateEmployeeBonus – Add Bonus to Employees in a Department

```
CREATE OR REPLACE PROCEDURE UpdateEmployeeBonus(
    p_department_id IN NUMBER,
    p_bonus_percent IN NUMBER
) IS
BEGIN
  -- Add bonus to all employees in the department
  UPDATE Employees
  SET Salary = Salary + (Salary * p_bonus_percent / 100)
  WHERE DepartmentID = p_department_id;

  COMMIT;
EXCEPTION
  WHEN OTHERS THEN
    ROLLBACK;
    DBMS_OUTPUT.PUT_LINE('Error updating employee bonus: ' || SQLERRM);
END;
/
```

Scenario 3: TransferFunds – Transfer Between Customer Accounts with Check

```
CREATE OR REPLACE PROCEDURE TransferFunds(
    p_from_account IN NUMBER,
    p_to_account   IN NUMBER,
    p_amount       IN NUMBER
) IS
    v_balance NUMBER;
BEGIN
  -- Get balance of the sender
  SELECT Balance INTO v_balance
  FROM Accounts
  WHERE AccountID = p_from_account;

  -- Check if balance is sufficient
  IF v_balance < p_amount THEN
    RAISE_APPLICATION_ERROR(-20003, 'Insufficient balance for transfer.');
  END IF;

  -- Debit from sender
  UPDATE Accounts
  SET Balance = Balance - p_amount
  WHERE AccountID = p_from_account;

  -- Credit to receiver
  UPDATE Accounts
  SET Balance = Balance + p_amount
  WHERE AccountID = p_to_account;

  COMMIT;
EXCEPTION
  WHEN OTHERS THEN
    ROLLBACK;
    DBMS_OUTPUT.PUT_LINE('Error during fund transfer: ' || SQLERRM);
END;
/
```

Exercise 4: Functions

Scenario 1: CalculateAge – Return Age from Date of Birth

```
CREATE OR REPLACE FUNCTION CalculateAge(
    p_dob DATE
) RETURN NUMBER IS
    v_age NUMBER;
BEGIN
    v_age := TRUNC(MONTHS_BETWEEN(SYSDATE, p_dob) / 12);
```

```
    RETURN v_age;
END;
/
```

Scenario 2: CalculateMonthlyInstallment – EMI Calculation

```
CREATE OR REPLACE FUNCTION CalculateMonthlyInstallment(
    p_loan_amount   IN NUMBER,
    p_annual_rate   IN NUMBER,
    p_years         IN NUMBER
) RETURN NUMBER IS
    v_monthly_rate  NUMBER;
    v_months        NUMBER;
    v_emi           NUMBER;
BEGIN
    v_monthly_rate := p_annual_rate / (12 * 100); -- Convert % annual to monthly decimal
    v_months := p_years * 12;

    IF v_monthly_rate = 0 THEN
        v_emi := p_loan_amount / v_months;
    ELSE
        v_emi := p_loan_amount * v_monthly_rate * POWER(1 + v_monthly_rate, v_months) /
                 (POWER(1 + v_monthly_rate, v_months) - 1);
    END IF;

    RETURN ROUND(v_emi, 2);
END;
/
```

Scenario 3: HasSufficientBalance – Boolean Check Before Transaction

```
CREATE OR REPLACE FUNCTION HasSufficientBalance(
    p_account_id IN NUMBER,
    p_amount     IN 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;
EXCEPTION
    WHEN NO_DATA_FOUND THEN
        RETURN FALSE;
    WHEN OTHERS THEN
        RETURN FALSE;
END;
/
```

Exercise 5: Triggers

Scenario 1: UpdateCustomerLastModified – Track Last Modified Timestamp

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

Scenario 2: LogTransaction – Insert Transaction Audit Record

Assumed AuditLog Table Structure:

```
CREATE TABLE AuditLog (
   LogID       NUMBER GENERATED BY DEFAULT AS IDENTITY,
   TransactionID NUMBER,
   Action      VARCHAR2(20),
   LogDate     DATE,
   Amount      NUMBER,
   AccountID   NUMBER
);
```

Trigger Code:

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

Scenario 3: CheckTransactionRules – Validate Deposits and Withdrawals

Assumed Transactions Table Columns:

```
CREATE OR REPLACE TRIGGER CheckTransactionRules
BEFORE INSERT ON Transactions
FOR EACH ROW
DECLARE
```

```
    v_balance NUMBER;
BEGIN
  -- Validate deposit
  IF :NEW.Type = 'DEPOSIT' AND :NEW.Amount <= 0 THEN
    RAISE_APPLICATION_ERROR(-20001, 'Deposit amount must be positive.');
  END IF;

  -- Validate withdrawal
  IF :NEW.Type = 'WITHDRAWAL' THEN
    SELECT Balance INTO v_balance
    FROM Accounts
    WHERE AccountID = :NEW.AccountID;

    IF :NEW.Amount > v_balance THEN
      RAISE_APPLICATION_ERROR(-20002, 'Withdrawal exceeds available balance.');
    END IF;
  END IF;
END;
/
```

Exercise 6: Cursors

Scenario 1: GenerateMonthlyStatements – Print Monthly Transactions for Customers

```
DECLARE
  CURSOR trans_cursor IS
    SELECT CustomerID, TransactionID, Amount, TransactionDate
    FROM Transactions
    WHERE TRUNC(TransactionDate, 'MM') = TRUNC(SYSDATE, 'MM')
    ORDER BY CustomerID;
 v_customer_id    NUMBER;
  v_transaction_id NUMBER;
  v_amount         NUMBER;
  v_date           DATE;
BEGIN
  OPEN trans_cursor;
  LOOP
    FETCH trans_cursor INTO v_customer_id, v_transaction_id, v_amount, v_date;
    EXIT WHEN trans_cursor%NOTFOUND;

    DBMS_OUTPUT.PUT_LINE('Customer ID: ' || v_customer_id ||
                         ', Transaction ID: ' || v_transaction_id ||
                         ', Amount: ' || v_amount ||
                         ', Date: ' || TO_CHAR(v_date, 'DD-MON-YYYY'));
  END LOOP;
  CLOSE trans_cursor;
END;
/
```

Scenario 2: ApplyAnnualFee – Deduct Maintenance Fee from All Accounts

```
DECLARE
  CURSOR account_cursor IS
    SELECT AccountID, Balance
    FROM Accounts;
  v_account_id NUMBER;
  v_balance    NUMBER;
  v_fee CONSTANT NUMBER := 100; -- Annual maintenance fee
BEGIN
  OPEN account_cursor;
  LOOP
    FETCH account_cursor INTO v_account_id, v_balance;
    EXIT WHEN account_cursor%NOTFOUND;
    UPDATE Accounts
    SET Balance = Balance - v_fee
    WHERE AccountID = v_account_id;
  END LOOP;
  CLOSE account_cursor;

  COMMIT;
END;
/
```

Scenario 3: UpdateLoanInterestRates – Apply New Interest Policy

```
DECLARE
  CURSOR loan_cursor IS
    SELECT LoanID, InterestRate
    FROM Loans;
  v_loan_id     NUMBER;
  v_interest    NUMBER;
BEGIN
  OPEN loan_cursor;
  LOOP
    FETCH loan_cursor INTO v_loan_id, v_interest;
    EXIT WHEN loan_cursor%NOTFOUND;
    IF v_interest > 10 THEN
      UPDATE Loans
      SET InterestRate = InterestRate - 1
      WHERE LoanID = v_loan_id;
    ELSIF v_interest >= 7 THEN
      UPDATE Loans
      SET InterestRate = InterestRate - 0.5
      WHERE LoanID = v_loan_id;
    END IF;
  END LOOP;
  CLOSE loan_cursor;
  COMMIT;
END;
/
```

Exercise 7: Packages

Scenario 1: CustomerManagement Package

Package Specification:

```
CREATE OR REPLACE PACKAGE CustomerManagement AS
  PROCEDURE AddCustomer(p_id NUMBER, p_name VARCHAR2, p_age NUMBER,
p_balance NUMBER);
  PROCEDURE UpdateCustomer(p_id NUMBER, p_name VARCHAR2, p_age NUMBER);
  FUNCTION GetCustomerBalance(p_id NUMBER) RETURN NUMBER;
END CustomerManagement;
/
```

Package Body:

```
CREATE OR REPLACE PACKAGE BODY CustomerManagement AS

  PROCEDURE AddCustomer(p_id NUMBER, p_name VARCHAR2, p_age NUMBER,
p_balance NUMBER) IS
  BEGIN
    INSERT INTO Customers(CustomerID, Name, Age, Balance)
    VALUES (p_id, p_name, p_age, p_balance);
  END;

  PROCEDURE UpdateCustomer(p_id NUMBER, p_name VARCHAR2, p_age NUMBER)
IS
  BEGIN
    UPDATE Customers
    SET Name = p_name, Age = p_age
    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;
  EXCEPTION
    WHEN NO_DATA_FOUND THEN
      RETURN 0;
  END;

END CustomerManagement;
/
```

Scenario 2: EmployeeManagement Package

Package Specification:

```
CREATE OR REPLACE PACKAGE EmployeeManagement AS
  PROCEDURE HireEmployee(p_id NUMBER, p_name VARCHAR2, p_salary NUMBER,
p_dept NUMBER);
  PROCEDURE UpdateEmployee(p_id NUMBER, p_name VARCHAR2, p_salary
NUMBER);
  FUNCTION CalculateAnnualSalary(p_id NUMBER) RETURN NUMBER;
END EmployeeManagement;
/
```

Package Body:

```
CREATE OR REPLACE PACKAGE BODY EmployeeManagement AS

  PROCEDURE HireEmployee(p_id NUMBER, p_name VARCHAR2, p_salary NUMBER,
p_dept NUMBER) IS
  BEGIN
    INSERT INTO Employees(EmployeeID, Name, Salary, DepartmentID)
    VALUES (p_id, p_name, p_salary, p_dept);
  END;

  PROCEDURE UpdateEmployee(p_id NUMBER, p_name VARCHAR2, p_salary
NUMBER) IS
  BEGIN
    UPDATE Employees
    SET Name = p_name, Salary = p_salary
    WHERE EmployeeID = p_id;
  END;

  FUNCTION CalculateAnnualSalary(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;
  EXCEPTION
    WHEN NO_DATA_FOUND THEN
      RETURN 0;
  END;

END EmployeeManagement;
/
```

Scenario 3: AccountOperations Package

Package Specification:

```
CREATE OR REPLACE PACKAGE AccountOperations AS
  PROCEDURE OpenAccount(p_account_id NUMBER, p_customer_id NUMBER, p_type
VARCHAR2, p_balance NUMBER);
  PROCEDURE CloseAccount(p_account_id NUMBER);
  FUNCTION GetTotalBalance(p_customer_id NUMBER) RETURN NUMBER;
END AccountOperations;
/
```

Package Body:

```
CREATE OR REPLACE PACKAGE BODY AccountOperations AS

  PROCEDURE OpenAccount(p_account_id NUMBER, p_customer_id NUMBER, p_type
VARCHAR2, p_balance NUMBER) IS
  BEGIN
    INSERT INTO Accounts(AccountID, CustomerID, AccountType, Balance)
    VALUES (p_account_id, p_customer_id, p_type, p_balance);
  END;

  PROCEDURE CloseAccount(p_account_id NUMBER) IS
  BEGIN
    DELETE FROM Accounts
    WHERE AccountID = p_account_id;
  END;

  FUNCTION GetTotalBalance(p_customer_id NUMBER) RETURN NUMBER IS
    v_total NUMBER;
  BEGIN
    SELECT NVL(SUM(Balance), 0) INTO v_total
    FROM Accounts
    WHERE CustomerID = p_customer_id;

    RETURN v_total;
  END;

END AccountOperations;
/
```

Tables:

1.Table Customers:

CustomerID	Name	DOB	Balance	LastModified
Filter...	Filter...	Filter...	Filter...	Filter...
1	John Doe	1985-05-15	1000	2025-06-28
2	Jane Smith	1990-07-20	1500	2025-06-28
3	Alice Brown	1992-03-12	2000	2025-06-28
4	Bob Gray	1988-11-01	1750	2025-06-28

2. TABLE Accounts

ACCOUNTID	CUSTOMERID	ACCOUNTTYPE	BALANCE	LASTMODIFIED
1	1	Savings	1000	20/03/25
2	2	Checking	1500	15/05/24

3. TABLE Transactions

TransactionID	AccountID	TransactionDate	Amount	TransactionType
Filter...	Filter...	Filter...	Filter...	Filter...
1	1	2025-05-25	200.00	Deposit
2	2	2025-06-23	300.00	Withdrawal

4. TABLE Loans

LoanID	CustomerID	LoanAmount	InterestRate	StartDate
Filter...	Filter...	Filter...	Filter...	Filter...
1	1	5000.00	5.00	2025-05-28

5. TABLE Employees

EmployeeID	Name	Position	Salary	Department	HireDate
Filter...	Filter...	Filter...	Filter...	Filter...	Filter...
1	Alice Johnson	Manager	70000.00	HR	2015-06-15
2	Bob Brown	Developer	60000.00	IT	2017-03-20