**Exercise 1: Control Structures**

**Scenario 1:**

BEGIN

    for c in (select \* from customers)

    LOOP

        if round(months\_between(sysdate,c.dob)/12)>60

        then

            update loans set interestrate=interestrate-1 where c.customerid=customerid;

        end if;

    end loop;

end;

**Scenario 2:**

ALTER TABLE Customers ADD IsVIP boolean;

update customers set isvip=false;

BEGIN

  FOR c IN (SELECT CustomerID, Balance FROM Customers)

  LOOP

    IF c.Balance > 10000 THEN

      UPDATE Customers SET IsVIP = true WHERE CustomerID = c.CustomerID;

    END IF;

  END LOOP;

END;

**Scenario 3:**

ALTER TABLE Customers ADD IsVIP boolean;

update customers set isvip=false;

BEGIN

  FOR c IN (SELECT CustomerID, Balance FROM Customers)

  LOOP

    IF c.Balance > 10000 THEN

      UPDATE Customers SET IsVIP = true WHERE CustomerID = c.CustomerID;

    END IF;

  END LOOP;

END;

**Exercise 2: Error Handling**

**Scenario 1:**

CREATE OR REPLACE PROCEDURE SafeTransferFunds(p\_from NUMBER, p\_to NUMBER, p\_amount NUMBER) IS  
 insufficient\_funds EXCEPTION;  
BEGIN  
 DECLARE  
 v\_balance NUMBER;  
 BEGIN  
 SELECT Balance INTO v\_balance FROM Accounts WHERE AccountID = p\_from;  
 IF v\_balance < p\_amount THEN  
 RAISE insufficient\_funds;  
 END IF;  
   
 UPDATE Accounts SET Balance = Balance - p\_amount WHERE AccountID = p\_from;  
 UPDATE Accounts SET Balance = Balance + p\_amount WHERE AccountID = p\_to;  
 COMMIT;  
 EXCEPTION  
 WHEN insufficient\_funds THEN  
 DBMS\_OUTPUT.PUT\_LINE('Error: Insufficient funds.');  
 ROLLBACK;  
 WHEN OTHERS THEN  
 DBMS\_OUTPUT.PUT\_LINE('Transfer failed: ' || SQLERRM);  
 ROLLBACK;  
 END;  
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%NOTFOUND THEN  
 RAISE\_APPLICATION\_ERROR(-20001, 'Employee ID not found.');  
 END IF;  
  
 COMMIT;  
EXCEPTION  
 WHEN OTHERS THEN  
 DBMS\_OUTPUT.PUT\_LINE('Error updating salary: ' || SQLERRM);  
 ROLLBACK;  
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 (CustomerID, Name, DOB, Balance, LastModified)  
 VALUES (p\_id, p\_name, p\_dob, p\_balance, SYSDATE);  
 COMMIT;  
EXCEPTION  
 WHEN DUP\_VAL\_ON\_INDEX THEN  
 DBMS\_OUTPUT.PUT\_LINE('Customer with ID ' || p\_id || ' already exists.');  
 ROLLBACK;  
 WHEN OTHERS THEN  
 DBMS\_OUTPUT.PUT\_LINE('Error: ' || SQLERRM);  
 ROLLBACK;  
END;

**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;

**Scenario 2:**

CREATE OR REPLACE PROCEDURE UpdateEmployeeBonus(p\_dept VARCHAR2, p\_bonus\_pct NUMBER) IS  
BEGIN  
 UPDATE Employees  
 SET Salary = Salary + (Salary \* p\_bonus\_pct / 100)  
 WHERE Department = p\_dept;  
 COMMIT;  
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;

    COMMIT;

  ELSE

    RAISE\_APPLICATION\_ERROR(-20002, 'Insufficient balance');

  END IF;

END;

**Exercise 4: Functions**

**Scenario 1:**

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  
 r NUMBER := p\_rate / (12 \* 100);  
 n NUMBER := p\_years \* 12;  
BEGIN  
 RETURN ROUND(p\_amount \* r / (1 - POWER(1 + r, -n)), 2);  
END;

**Scenario 3:**

CREATE OR REPLACE FUNCTION HasSufficientBalance(p\_accountid NUMBER, p\_amount NUMBER) RETURN BOOLEAN IS  
 v\_balance NUMBER;  
BEGIN  
 SELECT Balance INTO v\_balance FROM Accounts WHERE AccountID = p\_accountid;  
 RETURN v\_balance >= p\_amount;  
EXCEPTION  
 WHEN NO\_DATA\_FOUND THEN  
 RETURN FALSE;  
END;

**Exercise 5: Triggers**

**Scenario 1:**

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

**Scenario 2:**

CREATE TABLE AuditLog (  
 LogID NUMBER GENERATED ALWAYS AS IDENTITY,  
 AccountID NUMBER,  
 TransactionDate DATE,  
 Amount NUMBER,  
 TransactionType VARCHAR2(10)  
);  
  
CREATE OR REPLACE TRIGGER LogTransaction  
AFTER INSERT ON Transactions  
FOR EACH ROW  
BEGIN  
 INSERT INTO AuditLog (AccountID, TransactionDate, Amount, TransactionType)  
 VALUES (:NEW.AccountID, :NEW.TransactionDate, :NEW.Amount, :NEW.TransactionType);  
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(-20003, 'Withdrawal exceeds balance.');  
 ELSIF :NEW.TransactionType = 'Deposit' AND :NEW.Amount <= 0 THEN  
 RAISE\_APPLICATION\_ERROR(-20004, 'Deposit must be positive.');  
 END IF;  
END;

**Exercise 6: Cursors**

**Scenario 1:**

DECLARE  
 CURSOR trans\_cur IS  
 SELECT C.Name, A.AccountID, 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 TO\_CHAR(T.TransactionDate, 'MM-YYYY') = TO\_CHAR(SYSDATE, 'MM-YYYY');  
BEGIN  
 FOR t IN trans\_cur LOOP  
 DBMS\_OUTPUT.PUT\_LINE('Customer: ' || t.Name || ', Account: ' || t.AccountID ||  
 ', Date: ' || t.TransactionDate || ', Amount: ' || t.Amount || ', Type: ' || t.TransactionType);  
 END LOOP;  
END;

**Scenario 2:**

DECLARE  
 CURSOR acc\_cur IS SELECT AccountID, Balance FROM Accounts;  
BEGIN  
 FOR acc IN acc\_cur LOOP  
 UPDATE Accounts  
 SET Balance = Balance - 100  
 WHERE AccountID = acc.AccountID;  
 END LOOP;  
 COMMIT;  
END;

**Scenario 3:**

DECLARE  
 CURSOR loan\_cur IS SELECT LoanID, InterestRate FROM Loans;  
BEGIN  
 FOR l IN loan\_cur LOOP  
 UPDATE Loans  
 SET InterestRate = InterestRate + 0.5  
 WHERE LoanID = l.LoanID;  
 END LOOP;  
 COMMIT;  
END;

**Exercise 7: Packages**

**Scenario 1:**

CREATE OR REPLACE PACKAGE CustomerManagement AS  
 PROCEDURE AddCustomer(p\_id NUMBER, p\_name VARCHAR2, p\_dob DATE, p\_balance NUMBER);  
 PROCEDURE UpdateCustomer(p\_id NUMBER, p\_name VARCHAR2);  
 FUNCTION GetBalance(p\_id NUMBER) RETURN NUMBER;  
END CustomerManagement;  
/  
  
CREATE OR REPLACE PACKAGE BODY CustomerManagement AS  
 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, NULL);  
 END;  
  
 PROCEDURE UpdateCustomer(p\_id NUMBER, p\_name VARCHAR2) IS  
 BEGIN  
 UPDATE Customers SET Name = p\_name WHERE CustomerID = p\_id;  
 END;  
  
 FUNCTION GetBalance(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 CustomerManagement;

**Scenario 2:**

CREATE OR REPLACE PACKAGE EmployeeManagement AS  
 PROCEDURE HireEmployee(p\_id NUMBER, p\_name VARCHAR2, p\_pos VARCHAR2, p\_sal NUMBER, p\_dept VARCHAR2);  
 PROCEDURE UpdateEmployee(p\_id NUMBER, p\_name VARCHAR2);  
 FUNCTION AnnualSalary(p\_id NUMBER) RETURN NUMBER;  
END EmployeeManagement;  
/  
  
CREATE OR REPLACE PACKAGE BODY EmployeeManagement AS  
 PROCEDURE HireEmployee(p\_id NUMBER, p\_name VARCHAR2, p\_pos VARCHAR2, p\_sal NUMBER, p\_dept VARCHAR2) IS  
 BEGIN  
 INSERT INTO Employees VALUES (p\_id, p\_name, p\_pos, p\_sal, p\_dept, SYSDATE);  
 END;  
  
 PROCEDURE UpdateEmployee(p\_id NUMBER, p\_name VARCHAR2) IS  
 BEGIN  
 UPDATE Employees SET Name = p\_name WHERE EmployeeID = p\_id;  
 END;  
  
 FUNCTION AnnualSalary(p\_id NUMBER) RETURN NUMBER IS  
 v\_sal NUMBER;  
 BEGIN  
 SELECT Salary INTO v\_sal FROM Employees WHERE EmployeeID = p\_id;  
 RETURN v\_sal \* 12;  
 END;  
END EmployeeManagement;

**Scenario 3:**

CREATE OR REPLACE PACKAGE AccountOperations AS  
 PROCEDURE OpenAccount(p\_id NUMBER, p\_cust NUMBER, p\_type VARCHAR2, p\_balance NUMBER);  
 PROCEDURE CloseAccount(p\_id NUMBER);  
 FUNCTION TotalCustomerBalance(p\_custid NUMBER) RETURN NUMBER;  
END AccountOperations;  
/  
  
CREATE OR REPLACE PACKAGE BODY AccountOperations AS  
 PROCEDURE OpenAccount(p\_id NUMBER, p\_cust NUMBER, p\_type VARCHAR2, p\_balance NUMBER) IS  
 BEGIN  
 INSERT INTO Accounts VALUES (p\_id, p\_cust, p\_type, p\_balance, SYSDATE);  
 END;  
  
 PROCEDURE CloseAccount(p\_id NUMBER) IS  
 BEGIN  
 DELETE FROM Accounts WHERE AccountID = p\_id;  
 END;  
  
 FUNCTION TotalCustomerBalance(p\_custid NUMBER) RETURN NUMBER IS  
 v\_total NUMBER;  
 BEGIN  
 SELECT SUM(Balance) INTO v\_total FROM Accounts WHERE CustomerID = p\_custid;  
 RETURN NVL(v\_total, 0);  
 END;  
END AccountOperations;