PL/SQL Exercises with Full Answers

Exercise 1

Scenario 1 - Answer:

```
BEGIN
   FOR cust IN (SELECT c.CustomerID, c.DOB, l.LoanID, l.InterestRate
             FROM Customers c JOIN Loans l ON c.CustomerID =
l.CustomerID) LOOP
      IF MONTHS_BETWEEN(SYSDATE, cust.DOB) / 12 > 60 THEN
         UPDATE Loans SET InterestRate = InterestRate - 1 WHERE LoanID
= cust.LoanID;
      END IF;
   END LOOP;
   COMMIT;
END;
```

Scenario 2 - Answer:

```
BEGIN
   FOR cust IN (SELECT * FROM Customers) LOOP
      IF cust.Balance > 10000 THEN
         UPDATE Customers SET IsVIP = TRUE WHERE CustomerID =
cust.CustomerID;
      END IF;
   END LOOP;
   COMMIT;
END;
```

Scenario 3 - Answer:

```
BEGIN
   FOR loan IN (SELECT * FROM Loans WHERE EndDate BETWEEN
SYSDATE AND SYSDATE + 30) LOOP
      DBMS_OUTPUT.PUT_LINE('Reminder: Loan ' || loan.LoanID || ' for
Customer ' || loan.CustomerID || ' is due soon.');
  END LOOP;
END;
```

Exercise 2

Scenario 1 - Answer:

```
CREATE OR REPLACE PROCEDURE SafeTransferFunds(p_fromAcc
NUMBER, p_toAcc NUMBER, p_amount NUMBER) IS
  v_balance NUMBER;
BEGIN
  SELECT Balance INTO v_balance FROM Accounts WHERE AccountID =
p_fromAcc;
  IF v_balance < p_amount THEN
    RAISE_APPLICATION_ERROR(-20001, 'Insufficient funds');
  END IF;

  UPDATE Accounts SET Balance = Balance - p_amount WHERE
AccountID = p_fromAcc;
  UPDATE Accounts SET Balance = Balance + p_amount WHERE
AccountID = p_toAcc;
  COMMIT;
EXCEPTION
  WHEN OTHERS THEN
    ROLLBACK;
    DBMS_OUTPUT.PUT_LINE('Error: ' || SQLERRM);
END;
```

Scenario 2 - Answer:

```
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(-20002, 'Employee not found');
  END IF;
```

```
   COMMIT;
EXCEPTION
   WHEN OTHERS THEN
      DBMS_OUTPUT.PUT_LINE('Error: ' || SQLERRM);
END;
```

Scenario 3 - Answer:

```
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('Error: Customer ID already exists.');
      ROLLBACK;
END;
```

Exercise 3: Stored Procedures

Scenario 1: Process Monthly Interest

CREATE OR REPLACE PROCEDURE ProcessMonthlyInterest IS

BEGIN

UPDATE Accounts

SET Balance = Balance + (Balance * 0.01)

WHERE AccountType = 'Savings';

COMMIT;

END;

Scenario 2: Update Employee Bonus

CREATE OR REPLACE PROCEDURE UpdateEmployeeBonus(p_dept VARCHAR2,
p_bonus_percent NUMBER) IS

```
BEGIN
   UPDATE Employees
   SET Salary = Salary + (Salary * p_bonus_percent / 100)
   WHERE Department = p_dept;
   COMMIT;
END;
```

Scenario 3: Transfer Funds Between Accounts

```
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(-20003, 'Insufficient Balance');
   END IF;
END;
```

Exercise 4: Functions

Scenario 1: Calculate Age

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

Scenario 2: Calculate Monthly Installment

```
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 / (12 * 100);

BEGIN

  RETURN (p_amount * v_monthly_rate) / (1 - POWER(1 + v_monthly_rate, -v_months));

END;
```

Scenario 3: Check Sufficient Balance

```
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;

END;
```

Exercise 5: Triggers

Scenario 1: Update Last Modified Date

```
CREATE OR REPLACE TRIGGER UpdateCustomerLastModified

BEFORE UPDATE ON Customers

FOR EACH ROW

BEGIN

  :NEW.LastModified := SYSDATE;

END;
```

Scenario 2: Log Transaction

```
CREATE OR REPLACE TRIGGER LogTransaction

AFTER INSERT ON Transactions
```

```
FOR EACH ROW
BEGIN
  INSERT INTO AuditLog (TransactionID, Action, ActionDate)
  VALUES (:NEW.TransactionID, 'INSERT', SYSDATE);
END;
```

Scenario 3: Check Transaction Rules

```
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(-20004, 'Withdrawal exceeds balance');
  ELSIF :NEW.TransactionType = 'Deposit' AND :NEW.Amount <= 0 THEN
    RAISE_APPLICATION_ERROR(-20005, 'Deposit must be positive');
  END IF;
END;
```

Exercise 6: Cursors

Scenario 1: Generate Monthly Statements

```
DECLARE
  CURSOR cur_trans IS
    SELECT * FROM Transactions
```

```
        WHERE EXTRACT(MONTH FROM TransactionDate) = EXTRACT(MONTH FROM
SYSDATE);

BEGIN

    FOR rec IN cur_trans LOOP

        DBMS_OUTPUT.PUT_LINE('Customer transaction: ' || rec.AccountID || ', Amount: ' ||
rec.Amount);

    END LOOP;

END;
```

Scenario 2: Apply Annual Fee

```
DECLARE

    CURSOR cur_accounts IS SELECT AccountID, Balance FROM Accounts;

BEGIN

    FOR acc IN cur_accounts LOOP

        UPDATE Accounts SET Balance = acc.Balance - 100 WHERE AccountID = acc.AccountID;

    END LOOP;

    COMMIT;

END;
```

Scenario 3: Update Loan Interest Rates

```
DECLARE

    CURSOR cur_loans IS SELECT LoanID, InterestRate FROM Loans;

BEGIN

    FOR loan IN cur_loans LOOP

        UPDATE Loans SET InterestRate = loan.InterestRate * 1.05 WHERE LoanID =
loan.LoanID;

    END LOOP;

    COMMIT;

END;
```

Exercise 7: Packages

Scenario 1: CustomerManagement Package

```
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 GetBalance(p_id NUMBER) RETURN NUMBER;

END CustomerManagement;

/


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 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: EmployeeManagement Package

```
CREATE OR REPLACE PACKAGE EmployeeManagement IS

  PROCEDURE HireEmployee(p_id NUMBER, p_name VARCHAR2, p_position VARCHAR2,
p_salary NUMBER, p_dept VARCHAR2);

  PROCEDURE UpdateEmployee(p_id NUMBER, p_salary NUMBER);

  FUNCTION AnnualSalary(p_id NUMBER) RETURN NUMBER;

END EmployeeManagement;

/


CREATE OR REPLACE PACKAGE BODY EmployeeManagement IS

  PROCEDURE HireEmployee(...) IS ... END;

  PROCEDURE UpdateEmployee(...) IS ... END;

  FUNCTION AnnualSalary(...) RETURN NUMBER IS ... END;

END EmployeeManagement;
```

Scenario 3: AccountOperations Package

```
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 TotalBalance(p_custID NUMBER) RETURN NUMBER;

END AccountOperations;

/


CREATE OR REPLACE PACKAGE BODY AccountOperations IS

  PROCEDURE OpenAccount(...) IS ... END;
  PROCEDURE CloseAccount(...) IS ... END;

  FUNCTION TotalBalance(...) RETURN NUMBER IS ... END;

END AccountOperations;
```