**Exercise 1: Control Structures**

**Scenario 1:** The bank wants to apply a discount to loan interest rates for customers above 60 years old.

**Answer**:

DECLARE

CURSOR senior\_customers IS

SELECT c.CustomerID, l.LoanID, l.InterestRate

FROM Customers c

JOIN Loans l ON c.CustomerID = l.CustomerID

WHERE MONTHS\_BETWEEN(SYSDATE, c.DOB) / 12 > 60;

BEGIN

FOR rec IN senior\_customers LOOP

UPDATE Loans

SET InterestRate = InterestRate - 1

WHERE LoanID = rec.LoanID;

DBMS\_OUTPUT.PUT\_LINE('Discount applied to Customer ID: ' || rec.CustomerID);

END LOOP;

END;

/

**Scenario 2:** A customer can be promoted to VIP status based on their balance.

**Answer**:

ALTER TABLE Customers ADD IsVIP VARCHAR2(5);

BEGIN

FOR rec IN (SELECT CustomerID, Balance FROM Customers) LOOP

IF rec.Balance > 10000 THEN

UPDATE Customers

SET IsVIP = 'TRUE'

WHERE CustomerID = rec.CustomerID;

DBMS\_OUTPUT.PUT\_LINE('Customer ' || rec.CustomerID || ' promoted to VIP');

END IF;

END LOOP;

END;

/

**Scenario 3:** The bank wants to send reminders to customers whose loans are due within the next 30 days.

**Answer:**

BEGIN

FOR rec IN (

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

FROM Loans l

JOIN Customers c ON l.CustomerID = c.CustomerID

WHERE l.EndDate <= SYSDATE + 30

) LOOP

DBMS\_OUTPUT.PUT\_LINE('Reminder: Loan ID ' || rec.LoanID ||

' for ' || rec.Name || ' is due on ' || TO\_CHAR(rec.EndDate, 'YYYY-MM-DD'));

END LOOP;

END;

/

**Exercise 2: Error Handling**

**Scenario 1:** Handle exceptions during fund transfers between accounts.

**Answer:**

CREATE OR REPLACE PROCEDURE SafeTransferFunds(

p\_FromAccountID IN NUMBER,

p\_ToAccountID IN NUMBER,

p\_Amount IN NUMBER

) IS

v\_FromBalance NUMBER;

BEGIN

-- Get balance from source account

SELECT Balance INTO v\_FromBalance

FROM Accounts

WHERE AccountID = p\_FromAccountID;

IF v\_FromBalance < p\_Amount THEN

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

END IF;

UPDATE Accounts

SET Balance = Balance - p\_Amount

WHERE AccountID = p\_FromAccountID;

UPDATE Accounts

SET Balance = Balance + p\_Amount

WHERE AccountID = p\_ToAccountID;

INSERT INTO Transactions (TransactionID, AccountID, TransactionDate, Amount, TransactionType)

VALUES (Transactions\_seq.NEXTVAL, p\_FromAccountID, SYSDATE, -p\_Amount, 'Transfer');

INSERT INTO Transactions (TransactionID, AccountID, TransactionDate, Amount, TransactionType)

VALUES (Transactions\_seq.NEXTVAL, p\_ToAccountID, SYSDATE, p\_Amount, 'Transfer');

COMMIT;

EXCEPTION

WHEN OTHERS THEN

ROLLBACK;

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

END;

/

**Scenario 2:** Manage errors when updating employee salaries.

**Answer:**

CREATE OR REPLACE PROCEDURE UpdateSalary(

p\_EmployeeID IN NUMBER,

p\_Percent IN NUMBER

) IS

BEGIN

UPDATE Employees

SET Salary = Salary + (Salary \* p\_Percent / 100)

WHERE EmployeeID = p\_EmployeeID;

IF SQL%ROWCOUNT = 0 THEN

RAISE\_APPLICATION\_ERROR(-20002, 'Employee ID not found.');

END IF;

COMMIT;

EXCEPTION

WHEN OTHERS THEN

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

END;

/

**Scenario 3:** Ensure data integrity when adding a new customer.

**Answer:**

CREATE OR REPLACE 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);

COMMIT;

EXCEPTION

WHEN DUP\_VAL\_ON\_INDEX THEN

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

WHEN OTHERS THEN

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

END;

/

**Exercise 3: Stored Procedures**

**Scenario 1:** The bank needs to process monthly interest for all savings accounts.

**Answer:**

CREATE OR REPLACE PROCEDURE ProcessMonthlyInterest IS

BEGIN

FOR rec IN (

SELECT AccountID, Balance

FROM Accounts

WHERE AccountType = 'Savings'

) LOOP

UPDATE Accounts

SET Balance = Balance + (rec.Balance \* 0.01)

WHERE AccountID = rec.AccountID;

DBMS\_OUTPUT.PUT\_LINE('Interest applied to Account ID: ' || rec.AccountID);

END LOOP;

COMMIT;

END;

/

**Scenario 2:** The bank wants to implement a bonus scheme for employees based on their performance.

**Answer:**

CREATE OR REPLACE PROCEDURE UpdateEmployeeBonus(

p\_Department IN VARCHAR2,

p\_BonusPercent IN NUMBER

) IS

BEGIN

UPDATE Employees

SET Salary = Salary + (Salary \* p\_BonusPercent / 100)

WHERE Department = p\_Department;

DBMS\_OUTPUT.PUT\_LINE('Bonus updated for department: ' || p\_Department);

COMMIT;

END;

/

**Scenario 3:** Customers should be able to transfer funds between their accounts.

**Answer:**

CREATE OR REPLACE PROCEDURE TransferFunds(

p\_SourceAccountID IN NUMBER,

p\_TargetAccountID IN NUMBER,

p\_Amount IN NUMBER

) IS

v\_SourceBalance NUMBER;

BEGIN

SELECT Balance INTO v\_SourceBalance

FROM Accounts

WHERE AccountID = p\_SourceAccountID;

IF v\_SourceBalance < p\_Amount THEN

RAISE\_APPLICATION\_ERROR(-20003, 'Insufficient balance in source account.');

END IF;

UPDATE Accounts

SET Balance = Balance - p\_Amount

WHERE AccountID = p\_SourceAccountID;

UPDATE Accounts

SET Balance = Balance + p\_Amount

WHERE AccountID = p\_TargetAccountID;

COMMIT;

DBMS\_OUTPUT.PUT\_LINE('Funds transferred successfully.');

EXCEPTION

WHEN OTHERS THEN

ROLLBACK;

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

END;

/

**Exercise 4: Functions**

**Scenario 1:** Calculate the age of customers for eligibility checks.

**Answer:**

CREATE OR REPLACE FUNCTION CalculateAge(p\_DOB IN DATE)

RETURN NUMBER IS

v\_Age NUMBER;

BEGIN

v\_Age := FLOOR(MONTHS\_BETWEEN(SYSDATE, p\_DOB) / 12);

RETURN v\_Age;

END;

/

**Scenario 2:** The bank needs to compute the monthly installment for a loan.

**Answer:**

CREATE OR REPLACE FUNCTION CalculateMonthlyInstallment(

p\_LoanAmount IN NUMBER,

p\_InterestRate IN NUMBER,

p\_DurationYears IN NUMBER

) RETURN NUMBER IS

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

v\_Months NUMBER := p\_DurationYears \* 12;

v\_Installment NUMBER;

BEGIN

v\_Installment := (p\_LoanAmount \* v\_MonthlyRate) / (1 - POWER(1 + v\_MonthlyRate, -v\_Months));

RETURN ROUND(v\_Installment, 2);

END;

/

**Scenario 3:** Check if a customer has sufficient balance before making a transaction.

**Answer:**

CREATE OR REPLACE FUNCTION HasSufficientBalance(

p\_AccountID IN NUMBER,

p\_Amount IN 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:** Automatically update the last modified date when a customer's record is updated.

**Answer:**

CREATE OR REPLACE TRIGGER UpdateCustomerLastModified

BEFORE UPDATE ON Customers

FOR EACH ROW

BEGIN

:NEW.LastModified := SYSDATE;

END;

/

**Scenario 2:** Maintain an audit log for all transactions.

**Answer:**

CREATE TABLE AuditLog (

LogID NUMBER GENERATED ALWAYS AS IDENTITY PRIMARY KEY,

TransactionID NUMBER,

LogDate DATE DEFAULT SYSDATE,

Action VARCHAR2(50)

);

CREATE OR REPLACE TRIGGER LogTransaction

AFTER INSERT ON Transactions

FOR EACH ROW

BEGIN

INSERT INTO AuditLog (TransactionID, Action)

VALUES (:NEW.TransactionID, 'Transaction Inserted');

END;

/

**Scenario 3:** Enforce business rules on deposits and withdrawals.

**Answer:**

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 available 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 for all customers.

**Answer:**

DECLARE

CURSOR monthly\_cursor IS

SELECT t.TransactionID, c.Name, t.Amount, t.TransactionType

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');

BEGIN

FOR rec IN monthly\_cursor LOOP

DBMS\_OUTPUT.PUT\_LINE('Customer: ' || rec.Name || ' | Txn ID: ' || rec.TransactionID ||

' | Amount: ' || rec.Amount || ' | Type: ' || rec.TransactionType);

END LOOP;

END;

/

**Scenario 2:** Apply annual fee to all accounts.

**Answer:**

DECLARE

CURSOR fee\_cursor IS SELECT AccountID FROM Accounts;

v\_Fee NUMBER := 100;

BEGIN

FOR rec IN fee\_cursor LOOP

UPDATE Accounts

SET Balance = Balance - v\_Fee

WHERE AccountID = rec.AccountID;

DBMS\_OUTPUT.PUT\_LINE('Annual fee deducted from Account ID: ' || rec.AccountID);

END LOOP;

COMMIT;

END;

/

**Scenario 3:** Update the interest rate for all loans based on a new policy.

**Answer:**

DECLARE

CURSOR loan\_cursor IS SELECT LoanID, InterestRate FROM Loans;

v\_NewRate NUMBER := 4.5;

BEGIN

FOR rec IN loan\_cursor LOOP

UPDATE Loans

SET InterestRate = v\_NewRate

WHERE LoanID = rec.LoanID;

DBMS\_OUTPUT.PUT\_LINE('Loan ID ' || rec.LoanID || ' interest updated to ' || v\_NewRate || '%');

END LOOP;

COMMIT;

END;

/

**Exercise 7: Packages**

**Scenario 1:** Group all customer-related procedures and functions into a package.

**Answer:**

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\_Balance NUMBER);

FUNCTION GetCustomerBalance(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);

END;

PROCEDURE UpdateCustomer(p\_ID NUMBER, p\_Balance NUMBER) IS

BEGIN

UPDATE Customers SET Balance = p\_Balance, LastModified = SYSDATE 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;

END;

END CustomerManagement;

/

**Scenario 2:** Create a package to manage employee data.

**Answer:**

CREATE OR REPLACE PACKAGE EmployeeManagement AS

PROCEDURE HireEmployee(p\_ID NUMBER, p\_Name VARCHAR2, p\_Pos VARCHAR2, p\_Salary NUMBER, p\_Dept VARCHAR2);

PROCEDURE UpdateDetails(p\_ID NUMBER, p\_Salary NUMBER);

FUNCTION GetAnnualSalary(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\_Salary NUMBER, p\_Dept VARCHAR2) IS

BEGIN

INSERT INTO Employees VALUES (p\_ID, p\_Name, p\_Pos, p\_Salary, p\_Dept, SYSDATE);

END;

PROCEDURE UpdateDetails(p\_ID NUMBER, p\_Salary NUMBER) IS

BEGIN

UPDATE Employees SET Salary = p\_Salary WHERE EmployeeID = p\_ID;

END;

FUNCTION GetAnnualSalary(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;

END;

END EmployeeManagement;

/

**Scenario 3:** Group all account-related operations into a package.

**Answer:**

CREATE OR REPLACE PACKAGE AccountOperations AS

PROCEDURE OpenAccount(p\_ID NUMBER, p\_CustID NUMBER, p\_Type VARCHAR2, p\_Balance NUMBER);

PROCEDURE CloseAccount(p\_ID NUMBER);

FUNCTION GetTotalBalance(p\_CustID NUMBER) RETURN NUMBER;

END AccountOperations;

/

CREATE OR REPLACE PACKAGE BODY AccountOperations AS

PROCEDURE OpenAccount(p\_ID NUMBER, p\_CustID NUMBER, p\_Type VARCHAR2, p\_Balance NUMBER) IS

BEGIN

INSERT INTO Accounts VALUES (p\_ID, p\_CustID, p\_Type, p\_Balance, SYSDATE);

END;

PROCEDURE CloseAccount(p\_ID NUMBER) IS

BEGIN

DELETE FROM Accounts WHERE AccountID = p\_ID;

END;

FUNCTION GetTotalBalance(p\_CustID NUMBER) RETURN NUMBER IS

v\_Total NUMBER;

BEGIN

SELECT SUM(Balance) INTO v\_Total FROM Accounts WHERE CustomerID = p\_CustID;

RETURN v\_Total;

END;

END AccountOperations;

/