### **Exercise 1: Control Structures**

#### **Scenario 1: Apply Discount to Loan Interest Rates for Customers Above 60**

BEGIN

FOR rec IN (SELECT LoanID, CustomerID, InterestRate, DOB FROM Loans l JOIN Customers c ON l.CustomerID = c.CustomerID)

LOOP

IF TRUNC(MONTHS\_BETWEEN(SYSDATE, rec.DOB) / 12) > 60 THEN

UPDATE Loans

SET InterestRate = InterestRate - 1

WHERE LoanID = rec.LoanID;

END IF;

END LOOP;

COMMIT;

END;

#### **Scenario 2: Set IsVIP Flag for Customers with Balance Over $10,000**

BEGIN

FOR rec IN (SELECT CustomerID, Balance FROM Customers)

LOOP

IF rec.Balance > 10000 THEN

UPDATE Customers

SET IsVIP = TRUE

WHERE CustomerID = rec.CustomerID;

END IF;

END LOOP;

COMMIT;

END;

#### **Scenario 3: Send Reminders for Loans Due in the Next 30 Days**

BEGIN

FOR rec IN (SELECT LoanID, CustomerID, EndDate FROM Loans WHERE EndDate <= SYSDATE + 30)

LOOP

DBMS\_OUTPUT.PUT\_LINE('Reminder: Customer ' || rec.CustomerID || ', your loan is due on ' || TO\_CHAR(rec.EndDate, 'YYYY-MM-DD'));

END LOOP;

END;

### **Exercise 2: Error Handling**

#### **Scenario 1: SafeTransferFunds Procedure**

CREATE OR REPLACE PROCEDURE SafeTransferFunds(p\_fromAccountID NUMBER, p\_toAccountID NUMBER, p\_amount NUMBER) IS

insufficient\_funds EXCEPTION;

BEGIN

UPDATE Accounts

SET Balance = Balance - p\_amount

WHERE AccountID = p\_fromAccountID AND Balance >= p\_amount;

IF SQL%NOTFOUND THEN

RAISE insufficient\_funds;

END IF;

UPDATE Accounts

SET Balance = Balance + p\_amount

WHERE AccountID = p\_toAccountID;

COMMIT;

EXCEPTION

WHEN insufficient\_funds THEN

ROLLBACK;

INSERT INTO ErrorLog (ErrorMessage, ErrorDate) VALUES ('Insufficient funds in account ' || p\_fromAccountID, SYSDATE);

WHEN OTHERS THEN

ROLLBACK;

INSERT INTO ErrorLog (ErrorMessage, ErrorDate) VALUES (SQLERRM, SYSDATE);

END;

#### **Scenario 2: UpdateSalary Procedure**

CREATE OR REPLACE PROCEDURE UpdateSalary(p\_employeeID NUMBER, p\_percentage NUMBER) IS

employee\_not\_found EXCEPTION;

BEGIN

UPDATE Employees

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

WHERE EmployeeID = p\_employeeID;

IF SQL%NOTFOUND THEN

RAISE employee\_not\_found;

END IF;

COMMIT;

EXCEPTION

WHEN employee\_not\_found THEN

INSERT INTO ErrorLog (ErrorMessage, ErrorDate) VALUES ('Employee ID ' || p\_employeeID || ' not found.', SYSDATE);

WHEN OTHERS THEN

INSERT INTO ErrorLog (ErrorMessage, ErrorDate) VALUES (SQLERRM, SYSDATE);

END;

#### **Scenario 3: AddNewCustomer Procedure**

CREATE OR REPLACE PROCEDURE AddNewCustomer(p\_customerID NUMBER, p\_name VARCHAR2, p\_DOB DATE, p\_balance NUMBER) IS

customer\_exists EXCEPTION;

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

ROLLBACK;

RAISE customer\_exists;

WHEN customer\_exists THEN

INSERT INTO ErrorLog (ErrorMessage, ErrorDate) VALUES ('Customer ID ' || p\_customerID || ' already exists.', SYSDATE);

WHEN OTHERS THEN

ROLLBACK;

INSERT INTO ErrorLog (ErrorMessage, ErrorDate) VALUES (SQLERRM, SYSDATE);

END;

### **Exercise 3: Stored Procedures**

#### **Scenario 1: ProcessMonthlyInterest Procedure**

CREATE OR REPLACE PROCEDURE ProcessMonthlyInterest IS

BEGIN

FOR rec IN (SELECT AccountID, Balance FROM Accounts WHERE AccountType = 'Savings')

LOOP

UPDATE Accounts

SET Balance = Balance + (Balance \* 0.01)

WHERE AccountID = rec.AccountID;

END LOOP;

COMMIT;

END;

#### **Scenario 2: UpdateEmployeeBonus Procedure**

CREATE OR REPLACE PROCEDURE UpdateEmployeeBonus(p\_department VARCHAR2, p\_bonus\_percentage NUMBER) IS

BEGIN

UPDATE Employees

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

WHERE Department = p\_department;

COMMIT;

END;

#### **Scenario 3: TransferFunds Procedure**

CREATE OR REPLACE PROCEDURE TransferFunds(p\_fromAccountID NUMBER, p\_toAccountID NUMBER, p\_amount NUMBER) IS

BEGIN

IF (SELECT Balance FROM Accounts WHERE AccountID = p\_fromAccountID) >= p\_amount THEN

UPDATE Accounts

SET Balance = Balance - p\_amount

WHERE AccountID = p\_fromAccountID;

UPDATE Accounts

SET Balance = Balance + p\_amount

WHERE AccountID = p\_toAccountID;

COMMIT;

ELSE

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

END IF;

END;

### **Exercise 4: Functions**

#### **Scenario 1: CalculateAge Function**

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 Function**

CREATE OR REPLACE FUNCTION CalculateMonthlyInstallment(p\_loanAmount NUMBER, p\_interestRate NUMBER, p\_durationYears NUMBER) RETURN NUMBER IS

v\_monthlyInstallment NUMBER;

BEGIN

v\_monthlyInstallment := (p\_loanAmount \* p\_interestRate / 100 / 12) / (1 - POWER((1 + p\_interestRate / 100 / 12), (-p\_durationYears \* 12)));

RETURN v\_monthlyInstallment;

END;

#### **Scenario 3: HasSufficientBalance Function**

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: UpdateCustomerLastModified Trigger**

CREATE OR REPLACE TRIGGER UpdateCustomerLastModified

BEFORE UPDATE ON Customers

FOR EACH ROW

BEGIN

:NEW.LastModified := SYSDATE;

END;

#### **Scenario 2: LogTransaction Trigger**

CREATE OR REPLACE TRIGGER LogTransaction

AFTER INSERT ON Transactions

FOR EACH ROW

BEGIN

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

VALUES (:NEW.TransactionID, :NEW.AccountID, :NEW.TransactionDate, :NEW.Amount, :NEW.TransactionType);

END;

#### **Scenario 3: CheckTransactionRules Trigger**

CREATE OR REPLACE TRIGGER CheckTransactionRules

BEFORE INSERT ON Transactions

FOR EACH ROW

BEGIN

IF :NEW.TransactionType = 'Withdrawal' AND :NEW.Amount > (SELECT Balance FROM Accounts WHERE AccountID = :NEW.AccountID) THEN

RAISE\_APPLICATION\_ERROR(-20002, 'Withdrawal amount exceeds balance.');

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

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

END IF;

END;

### **Exercise 6: Cursors**

#### **Scenario 1: GenerateMonthlyStatements Cursor**

DECLARE

CURSOR cur\_statements IS

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

FROM Customers c

JOIN Accounts a ON c.CustomerID = a.CustomerID

JOIN Transactions t ON a.AccountID = t.AccountID

WHERE t.TransactionDate BETWEEN TRUNC(SYSDATE, 'MM') AND LAST\_DAY(SYSDATE);

BEGIN

FOR rec IN cur\_statements

LOOP

DBMS\_OUTPUT.PUT\_LINE('Customer: ' || rec.Name || ' (' || rec.CustomerID || ')');

DBMS\_OUTPUT.PUT\_LINE('Transaction Date: ' || TO\_CHAR(rec.TransactionDate, 'YYYY-MM-DD'));

DBMS\_OUTPUT.PUT\_LINE('Amount: ' || rec.Amount || ' Type: ' || rec.TransactionType);

END LOOP;

END;

#### **Scenario 2: ApplyAnnualFee Cursor**

DECLARE

CURSOR cur\_fees IS

SELECT AccountID, Balance

FROM Accounts;

BEGIN

FOR rec IN cur\_fees

LOOP

UPDATE Accounts

SET Balance = Balance - 50 -- Assuming the annual fee is $50

WHERE AccountID = rec.AccountID;

END LOOP;

COMMIT;

END;

#### **Scenario 3: UpdateLoanInterestRates Cursor**

DECLARE

CURSOR cur\_loans IS

SELECT LoanID, InterestRate

FROM Loans;

BEGIN

FOR rec IN cur\_loans

LOOP

UPDATE Loans

SET InterestRate = InterestRate + 1 -- Applying a new policy

WHERE LoanID = rec.LoanID;

END LOOP;

COMMIT;

END;

### **Exercise 7: Packages**

#### **Scenario 1: CustomerManagement Package**

CREATE OR REPLACE PACKAGE CustomerManagement IS

PROCEDURE AddCustomer(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 IS

PROCEDURE AddCustomer(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);

COMMIT;

END;

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;

COMMIT;

END;

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;

END CustomerManagement;

#### **Scenario 2: EmployeeManagement Package**

CREATE OR REPLACE PACKAGE EmployeeManagement IS

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, p\_hireDate DATE);

FUNCTION CalculateAnnualSalary(p\_employeeID NUMBER) RETURN NUMBER;

END EmployeeManagement;

/

CREATE OR REPLACE PACKAGE BODY EmployeeManagement IS

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);

COMMIT;

END;

PROCEDURE UpdateEmployeeDetails(p\_employeeID NUMBER, p\_name VARCHAR2, p\_position VARCHAR2, p\_salary NUMBER, p\_department VARCHAR2, p\_hireDate DATE) IS

BEGIN

UPDATE Employees

SET Name = p\_name, Position = p\_position, Salary = p\_salary, Department = p\_department, HireDate = p\_hireDate

WHERE EmployeeID = p\_employeeID;

COMMIT;

END;

FUNCTION CalculateAnnualSalary(p\_employeeID NUMBER) RETURN NUMBER IS

v\_annualSalary NUMBER;

BEGIN

SELECT Salary \* 12 INTO v\_annualSalary FROM Employees WHERE EmployeeID = p\_employeeID;

RETURN v\_annualSalary;

END;

END EmployeeManagement;

#### **Scenario 3: AccountOperations Package**

CREATE OR REPLACE PACKAGE AccountOperations IS

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 IS

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);

COMMIT;

END;

PROCEDURE CloseAccount(p\_accountID NUMBER) IS

BEGIN

DELETE FROM Accounts WHERE AccountID = p\_accountID;

COMMIT;

END;

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;

END AccountOperations;