**Exercise 1: Control Structures**

**Scenario 1**

BEGIN

FOR rec IN (SELECT CustomerID, InterestRate FROM Loans WHERE CustomerID IN

(SELECT CustomerID FROM Customers WHERE EXTRACT(YEAR FROM SYSDATE) -

EXTRACT(YEAR FROM DOB) > 60)) LOOP

UPDATE Loans

SET InterestRate = InterestRate - 1

WHERE CustomerID = rec.CustomerID;

END LOOP;

END;

/

**Scenario 2**

BEGIN

FOR rec IN (SELECT CustomerID, Balance FROM Customers WHERE

Balance > 10000) LOOP

UPDATE Customers

SET IsVIP = 'TRUE'

WHERE CustomerID = rec.CustomerID;

END LOOP;

END;

/

**Scenario 3**

BEGIN

FOR rec IN (SELECT CustomerID, LoanID, EndDate FROM Loans WHERE EndDate

BETWEEN SYSDATE AND SYSDATE + 30) LOOP

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

rec.CustomerID || ' is due on ' || rec.EndDate);

END LOOP;

END;

/

**Exercise 2: Error Handling**

**Scenario 1**

CREATE OR REPLACE PROCEDURE SafeTransferFunds(p\_from\_account

IN NUMBER, p\_to\_account IN NUMBER, p\_amount IN NUMBER) AS

BEGIN

BEGIN

UPDATE Accounts

SET Balance = Balance - p\_amount

WHERE AccountID = p\_from\_account;

IF SQL%ROWCOUNT = 0 THEN

RAISE\_APPLICATION\_ERROR(-20001, 'Source account not found.');

END IF;

IF (SELECT Balance FROM Accounts WHERE AccountID =

p\_from\_account) < p\_amount THEN

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

END IF;

UPDATE Accounts

SET Balance = Balance + p\_amount

WHERE AccountID = p\_to\_account;

IF SQL%ROWCOUNT = 0 THEN

RAISE\_APPLICATION\_ERROR(-20003, 'Destination account not

found.');

END IF;

COMMIT;

EXCEPTION

WHEN OTHERS THEN

ROLLBACK;

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

END;

END;

/

**Scenario 2**

CREATE OR REPLACE PROCEDURE UpdateSalary(p\_employee\_id IN NUMBER,

p\_percentage IN NUMBER) AS

BEGIN

UPDATE Employees

SET Salary = Salary \* (1 + p\_percentage / 100)

WHERE EmployeeID = p\_employee\_id;

IF SQL%ROWCOUNT = 0 THEN

RAISE\_APPLICATION\_ERROR(-20004, 'Employee ID not found.');

END IF;

COMMIT;

EXCEPTION

WHEN OTHERS THEN

ROLLBACK;

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

END**;**

/

**Scenario 3**

CREATE OR REPLACE PROCEDURE AddNewCustomer(p\_customer\_id IN NUMBER,

p\_name IN VARCHAR2, p\_dob IN DATE, p\_balance IN NUMBER) AS

BEGIN

BEGIN

INSERT INTO Customers (CustomerID, Name, DOB, Balance, LastModified)

VALUES (p\_customer\_id, p\_name, p\_dob, p\_balance, SYSDATE);

COMMIT;

EXCEPTION

WHEN DUP\_VAL\_ON\_INDEX THEN

DBMS\_OUTPUT.PUT\_LINE('Error: Customer with ID ' || p\_customer\_id || ' already

exists.');

END;

END;

/

**Exercise 3: Stored Procedures**

**Scenario 1**

CREATE OR REPLACE PROCEDURE ProcessMonthlyInterest AS

BEGIN

FOR rec IN (SELECT AccountID, Balance FROM Accounts) LOOP

UPDATE Accounts

SET Balance = Balance \* 1.01

WHERE AccountID = rec.AccountID;

END LOOP;

COMMIT;

END;

/

**Scenario 2**

CREATE OR REPLACE PROCEDURE UpdateEmployeeBonus(p\_department IN

VARCHAR2, p\_bonus\_percentage IN NUMBER) AS

BEGIN

UPDATE Employees

SET Salary = Salary \* (1 + p\_bonus\_percentage / 100)

WHERE Department = p\_department;

COMMIT;

END;

/

**Scenario 3**

CREATE OR REPLACE PROCEDURE TransferFunds(p\_from\_account IN NUMBER,

p\_to\_account IN NUMBER, p\_amount IN NUMBER) AS

v\_balance NUMBER;

BEGIN

SELECT Balance INTO v\_balance FROM Accounts WHERE AccountID = p\_from\_account;

IF v\_balance < p\_amount THEN

RAISE\_APPLICATION\_ERROR(-20005, 'Insufficient balance.');

END IF;

UPDATE Accounts

SET Balance = Balance - p\_amount

WHERE AccountID = p\_from\_account;

UPDATE Accounts

SET Balance = Balance + p\_amount

WHERE AccountID = p\_to\_account;

COMMIT;

END;

/

**Exercise 4: Functions**

**Scenario 1**

CREATE OR REPLACE FUNCTION CalculateAge(p\_dob DATE) RETURN NUMBER AS

BEGIN

RETURN FLOOR(MONTHS\_BETWEEN(SYSDATE, p\_dob) / 12);

END;

/

**Scenario 2**

CREATE OR REPLACE FUNCTION CalculateMonthlyInstallment(p\_loan\_amount NUMBER,

p\_interest\_rate NUMBER, p\_duration\_years NUMBER) RETURN NUMBER AS

v\_monthly\_rate NUMBER;

v\_installment NUMBER;

BEGIN

v\_monthly\_rate := (p\_interest\_rate / 100) / 12;

v\_installment := (p\_loan\_amount \* v\_monthly\_rate) / (1 - POWER(1 + v\_monthly\_rate, -

p\_duration\_years \* 12));

RETURN v\_installment;

END;

/

**Scenario 3**

CREATE OR REPLACE FUNCTION HasSufficientBalance(p\_account\_id NUMBER,

p\_amount NUMBER) RETURN BOOLEAN AS

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;

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 OR REPLACE TRIGGER LogTransaction

AFTER INSERT ON Transactions

FOR EACH ROW

BEGIN

INSERT INTO AuditLog (TransactionID, LogDate, Action)

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

END;

/

**Scenario 3**

CREATE OR REPLACE TRIGGER CheckTransactionRules

BEFORE INSERT ON Transactions

FOR EACH ROW

BEGIN

IF :NEW.TransactionType = 'Withdrawal' AND

(SELECT Balance FROM Accounts WHERE AccountID = :NEW.AccountID) <

:NEW.Amount THEN

RAISE\_APPLICATION\_ERROR(-20006, 'Withdrawal exceeds balance.');

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

RAISE\_APPLICATION\_ERROR(-20007, 'Deposit amount must be positive.');

END IF;

END;

/

**Exercise 6: Cursors**

**Scenario 1**

DECLARE

CURSOR c\_transactions IS

SELECT \* FROM Transactions WHERE EXTRACT(MONTH FROM TransactionDate) =

EXTRACT(MONTH FROM SYSDATE)

AND EXTRACT(YEAR FROM TransactionDate) = EXTRACT(YEAR FROM SYSDATE);

v\_statement VARCHAR2(4000);

BEGIN

FOR rec IN c\_transactions LOOP

v\_statement := 'Statement for Account ID ' || rec.AccountID || ': ' ||

'Transaction Date: ' || rec.TransactionDate || ', ' ||

'Amount: ' || rec.Amount || ', ' ||

'Type: ' || rec.TransactionType;

DBMS\_OUTPUT.PUT\_LINE(v\_statement);

END LOOP;

END;

/

**Scenario 2**

DECLARE

CURSOR c\_accounts IS

SELECT AccountID, Balance FROM Accounts;

BEGIN

FOR rec IN c\_accounts LOOP

UPDATE Accounts

SET Balance = Balance - 50 -- assuming annual fee is 50

WHERE AccountID = rec.AccountID;

END LOOP;

COMMIT;

END;

/

**Scenario 3**

DECLARE

CURSOR c\_loans IS

SELECT LoanID, InterestRate FROM Loans;

BEGIN

FOR rec IN c\_loans LOOP

UPDATE Loans

SET InterestRate = InterestRate + 0.5

WHERE LoanID = rec.LoanID;

END LOOP;

COMMIT;

END;

**Exercise 7: Packages**

**Scenario 1**

CREATE OR REPLACE PACKAGE CustomerManagement AS

PROCEDURE AddNewCustomer(p\_customer\_id IN NUMBER, p\_name IN VARCHAR2,

p\_dob IN DATE, p\_balance IN NUMBER);

PROCEDURE UpdateCustomerDetails(p\_customer\_id IN NUMBER, p\_name IN

VARCHAR2, p\_balance IN NUMBER);

FUNCTION GetCustomerBalance(p\_customer\_id IN NUMBER) RETURN NUMBER;

END CustomerManagement;

/

CREATE OR REPLACE PACKAGE BODY CustomerManagement AS

PROCEDURE AddNewCustomer(p\_customer\_id 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\_customer\_id, p\_name, p\_dob, p\_balance, SYSDATE);

COMMIT;

END;

PROCEDURE UpdateCustomerDetails(p\_customer\_id IN NUMBER, p\_name IN

VARCHAR2, p\_balance IN NUMBER) IS

BEGIN

UPDATE Customers

SET Name = p\_name, Balance = p\_balance, LastModified = SYSDATE

WHERE CustomerID = p\_customer\_id;

COMMIT;

END;

FUNCTION GetCustomerBalance(p\_customer\_id IN NUMBER) RETURN NUMBER IS

v\_balance NUMBER;

BEGIN

SELECT Balance INTO v\_balance FROM Customers WHERE CustomerID = p\_customer\_id;

RETURN v\_balance;

END;

END CustomerManagement;

/

**Scenario 2**

CREATE OR REPLACE PACKAGE EmployeeManagement AS

PROCEDURE HireEmployee(p\_employee\_id IN NUMBER, p\_name IN VARCHAR2,

p\_position IN VARCHAR2, p\_salary IN NUMBER, p\_department IN VARCHAR2,

p\_hire\_date IN DATE);

PROCEDURE UpdateEmployeeDetails(p\_employee\_id IN NUMBER, p\_name IN

VARCHAR2, p\_salary IN NUMBER, p\_department IN VARCHAR2);

FUNCTION CalculateAnnualSalary(p\_employee\_id IN NUMBER) RETURN NUMBER;

END EmployeeManagement;

/

CREATE OR REPLACE PACKAGE BODY EmployeeManagement AS

PROCEDURE HireEmployee(p\_employee\_id IN NUMBER, p\_name IN VARCHAR2,

p\_position IN VARCHAR2, p\_salary IN NUMBER, p\_department IN VARCHAR2,

p\_hire\_date IN DATE) IS

BEGIN

INSERT INTO Employees (EmployeeID, Name, Position, Salary, Department, HireDate)

VALUES (p\_employee\_id, p\_name, p\_position, p\_salary, p\_department, p\_hire\_date);

COMMIT;

END;

PROCEDURE UpdateEmployeeDetails(p\_employee\_id IN NUMBER, p\_name IN

VARCHAR2, p\_salary IN NUMBER, p\_department IN VARCHAR2) IS

BEGIN

UPDATE Employees

SET Name = p\_name, Salary = p\_salary, Department = p\_department

WHERE EmployeeID = p\_employee\_id;

COMMIT;

END;

FUNCTION CalculateAnnualSalary(p\_employee\_id IN NUMBER) RETURN NUMBER IS

v\_salary NUMBER;

BEGIN

SELECT Salary INTO v\_salary FROM Employees WHERE EmployeeID = p\_employee\_id;

RETURN v\_salary \* 12;

END;

END EmployeeManagement;

/

**Scenario 3**

CREATE OR REPLACE PACKAGE AccountOperations AS

PROCEDURE OpenNewAccount(p\_account\_id IN NUMBER, p\_customer\_id IN NUMBER,

p\_account\_type IN VARCHAR2, p\_balance IN NUMBER);

PROCEDURE CloseAccount(p\_account\_id IN NUMBER);

FUNCTION GetTotalBalance(p\_customer\_id IN NUMBER) RETURN NUMBER;

END AccountOperations;

/

CREATE OR REPLACE PACKAGE BODY AccountOperations AS

PROCEDURE OpenNewAccount(p\_account\_id IN NUMBER, p\_customer\_id IN NUMBER,

p\_account\_type IN VARCHAR2, p\_balance IN NUMBER) IS

BEGIN

INSERT INTO Accounts (AccountID, CustomerID, AccountType, Balance, LastModified)

VALUES (p\_account\_id, p\_customer\_id, p\_account\_type, p\_balance, SYSDATE);

COMMIT;

END;

PROCEDURE CloseAccount(p\_account\_id IN NUMBER) IS

BEGIN

DELETE FROM Accounts WHERE AccountID = p\_account\_id;

COMMIT;

END;

FUNCTION GetTotalBalance(p\_customer\_id IN NUMBER) RETURN NUMBER IS

v\_total\_balance NUMBER := 0;

BEGIN

SELECT SUM(Balance) INTO v\_total\_balance FROM Accounts WHERE CustomerID =

p\_customer\_id;

RETURN v\_total\_balance;

END;

END AccountOperations;

/