**Exercise 1: Control Structures**

1. DECLARE

v\_age NUMBER;

BEGIN

FOR loan\_record IN (SELECT LoanID, CustomerID, InterestRate FROM Loans) LOOP

SELECT FLOOR(MONTHS\_BETWEEN(SYSDATE, DOB) / 12) INTO v\_age FROM Customers WHERE CustomerID = loan\_record.CustomerID;

IF v\_age > 60 THEN

UPDATE Loans

SET InterestRate = InterestRate - 0.01

WHERE LoanID = loan\_record.LoanID;

END IF;

END LOOP;

COMMIT;

END;

1. DECLARE

BEGIN

FOR customer\_record IN (SELECT CustomerID, Balance FROM Customers) LOOP

IF customer\_record.Balance > 10000 THEN

-- Assuming IsVIP is a column in Customers table

UPDATE Customers

SET IsVIP = TRUE

WHERE CustomerID = customer\_record.CustomerID;

END IF;

END LOOP;

COMMIT;

END;

1. DECLARE

BEGIN

FOR loan\_record IN (SELECT LoanID, CustomerID, EndDate FROM Loans WHERE EndDate <= SYSDATE + 30) LOOP

DBMS\_OUTPUT.PUT\_LINE('Reminder: Customer ' || loan\_record.CustomerID ||

', your loan ID ' || loan\_record.LoanID ||

' is due on ' || TO\_CHAR(loan\_record.EndDate, 'DD-MON-YYYY'));

END LOOP;

END;

**Exercise 2: Error Handling**

1. CREATE OR REPLACE PROCEDURE SafeTransferFunds (

p\_from\_account\_id IN NUMBER,

p\_to\_account\_id IN NUMBER,

p\_amount IN NUMBER

) IS

insufficient\_funds EXCEPTION;

v\_balance\_from NUMBER;

BEGIN

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

IF v\_balance\_from < p\_amount THEN

RAISE insufficient\_funds;

END IF;

UPDATE Accounts SET Balance = Balance - p\_amount WHERE AccountID = p\_from\_account\_id;

UPDATE Accounts SET Balance = Balance + p\_amount WHERE AccountID = p\_to\_account\_id;

COMMIT;

EXCEPTION

WHEN insufficient\_funds THEN

ROLLBACK;

DBMS\_OUTPUT.PUT\_LINE('Error: Insufficient funds in account ' || p\_from\_account\_id);

WHEN OTHERS THEN

ROLLBACK;

DBMS\_OUTPUT.PUT\_LINE('Error: An unexpected error occurred during the fund transfer.');

END SafeTransferFunds;

1. CREATE OR REPLACE PROCEDURE UpdateSalary (

p\_employee\_id IN NUMBER,

p\_percentage IN NUMBER

) IS

v\_current\_salary NUMBER;

BEGIN

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

UPDATE Employees

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

WHERE EmployeeID = p\_employee\_id;

COMMIT;

EXCEPTION

WHEN NO\_DATA\_FOUND THEN

ROLLBACK;

DBMS\_OUTPUT.PUT\_LINE('Error: Employee with ID ' || p\_employee\_id || ' does not exist.');

WHEN OTHERS THEN

ROLLBACK;

DBMS\_OUTPUT.PUT\_LINE('Error: An unexpected error occurred while updating the salary.');

END UpdateSalary;

1. CREATE OR REPLACE 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;

EXCEPTION

WHEN DUP\_VAL\_ON\_INDEX THEN

ROLLBACK;

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

WHEN OTHERS THEN

ROLLBACK;

DBMS\_OUTPUT.PUT\_LINE('Error: An unexpected error occurred while adding a new customer.');

END AddNewCustomer;

**Exercise 3: Stored Procedures**

1. CREATE OR REPLACE PROCEDURE ProcessMonthlyInterest IS

BEGIN

FOR account\_record IN (SELECT AccountID, Balance FROM Accounts WHERE AccountType = 'Savings') LOOP

UPDATE Accounts

SET Balance = Balance + (Balance \* 0.01),

LastModified = SYSDATE

WHERE AccountID = account\_record.AccountID;

END LOOP;

COMMIT;

END ProcessMonthlyInterest;

1. CREATE OR REPLACE PROCEDURE UpdateEmployeeBonus (

p\_department IN VARCHAR2,

p\_bonus\_percentage IN NUMBER

) IS

BEGIN

UPDATE Employees

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

WHERE Department = p\_department;

COMMIT;

END UpdateEmployeeBonus;

1. CREATE OR REPLACE PROCEDURE TransferFunds (

p\_from\_account\_id IN NUMBER,

p\_to\_account\_id IN NUMBER,

p\_amount IN NUMBER

) IS

insufficient\_funds EXCEPTION;

v\_balance\_from NUMBER;

BEGIN

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

IF v\_balance\_from < p\_amount THEN

RAISE insufficient\_funds;

END IF;

UPDATE Accounts SET Balance = Balance - p\_amount WHERE AccountID = p\_from\_account\_id;

UPDATE Accounts SET Balance = Balance + p\_amount WHERE AccountID = p\_to\_account\_id;

COMMIT;

EXCEPTION

WHEN insufficient\_funds THEN

ROLLBACK;

DBMS\_OUTPUT.PUT\_LINE('Error: Insufficient funds in account ' || p\_from\_account\_id);

WHEN OTHERS THEN

ROLLBACK;

DBMS\_OUTPUT.PUT\_LINE('Error: An unexpected error occurred during the fund transfer.');

END TransferFunds;

**Exercise 4: Functions**

1. CREATE OR REPLACE FUNCTION CalculateAge (

p\_dob IN DATE

) RETURN NUMBER IS

v\_age NUMBER;

BEGIN

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

INTO v\_age

FROM DUAL;

RETURN v\_age;

END CalculateAge;

1. CREATE OR REPLACE FUNCTION CalculateMonthlyInstallment (

p\_loan\_amount IN NUMBER,

p\_interest\_rate IN NUMBER,

p\_loan\_duration\_years IN NUMBER

) RETURN NUMBER IS

v\_monthly\_installment NUMBER;

v\_monthly\_rate NUMBER;

v\_total\_payments NUMBER;

BEGIN

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

v\_total\_payments := p\_loan\_duration\_years \* 12;

v\_monthly\_installment := p\_loan\_amount \* v\_monthly\_rate /

(1 - POWER(1 + v\_monthly\_rate, -v\_total\_payments));

RETURN v\_monthly\_installment;

END CalculateMonthlyInstallment;

3) CREATE OR REPLACE FUNCTION HasSufficientBalance (

p\_account\_id IN NUMBER,

p\_amount IN NUMBER

) RETURN BOOLEAN IS

v\_balance NUMBER;

BEGIN

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

IF v\_balance >= p\_amount THEN

RETURN TRUE;

ELSE

RETURN FALSE;

END IF;

END HasSufficientBalance;

**Exercise 5: Triggers**

1. CREATE OR REPLACE TRIGGER UpdateCustomerLastModified

BEFORE UPDATE ON Customers

FOR EACH ROW

BEGIN

:NEW.LastModified := SYSDATE;

END UpdateCustomerLastModified;

1. CREATE OR REPLACE TRIGGER LogTransaction

AFTER INSERT ON Transactions

FOR EACH ROW

BEGIN

INSERT INTO AuditLog (transaction\_id, account\_id, transaction\_type, amount, transaction\_date)

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

END LogTransaction;

1. CREATE OR REPLACE TRIGGER CheckTransactionRules

BEFORE INSERT ON Transactions

FOR EACH ROW

DECLARE

v\_balance NUMBER;

BEGIN

IF :NEW.TransactionType = 'withdrawal' THEN

SELECT Balance INTO v\_balance FROM Accounts WHERE AccountID = :NEW.AccountID;

IF :NEW.Amount > v\_balance THEN

RAISE\_APPLICATION\_ERROR(-20001, 'Insufficient balance for withdrawal.');

END IF;

ELSIF :NEW.TransactionType = 'deposit' THEN

IF :NEW.Amount <= 0 THEN

RAISE\_APPLICATION\_ERROR(-20002, 'Deposit amount must be positive.');

END IF;

END IF;

END CheckTransactionRules;

**Exercise 6: Cursors**

1. DECLARE

CURSOR cur\_transactions IS

SELECT CustomerID, AccountID, TransactionDate, Amount, TransactionType

FROM Transactions

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

BEGIN

FOR trans\_record IN cur\_transactions LOOP

DBMS\_OUTPUT.PUT\_LINE('Customer ID: ' || trans\_record.CustomerID ||

', Account ID: ' || trans\_record.AccountID ||

', Date: ' || TO\_CHAR(trans\_record.TransactionDate, 'DD-MON-YYYY') ||

', Type: ' || trans\_record.TransactionType ||

', Amount: ' || trans\_record.Amount);

END LOOP;

END;

1. DECLARE

CURSOR cur\_accounts IS

SELECT AccountID, Balance FROM Accounts;

v\_annual\_fee NUMBER := 100; -- Assuming a fixed annual fee

BEGIN

FOR account\_record IN cur\_accounts LOOP

UPDATE Accounts

SET Balance = Balance - v\_annual\_fee,

LastModified = SYSDATE

WHERE AccountID = account\_record.AccountID;

END LOOP;

COMMIT;

END;

1. DECLARE

CURSOR cur\_loans IS

SELECT LoanID, InterestRate FROM Loans;

v\_new\_interest\_rate NUMBER := 0.05; -- New interest rate of 5%

BEGIN

FOR loan\_record IN cur\_loans LOOP

UPDATE Loans

SET InterestRate = v\_new\_interest\_rate

WHERE LoanID = loan\_record.LoanID;

END LOOP;

COMMIT;

END;

**Exercise 7: Packages**

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\_dob IN DATE

);

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 AddNewCustomer;

PROCEDURE UpdateCustomerDetails (

p\_customer\_id IN NUMBER,

p\_name IN VARCHAR2,

p\_dob IN DATE

) IS

BEGIN

UPDATE Customers

SET Name = p\_name, DOB = p\_dob, LastModified = SYSDATE

WHERE CustomerID = p\_customer\_id;

COMMIT;

END UpdateCustomerDetails;

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 GetCustomerBalance;

END CustomerManagement;

/

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

);

PROCEDURE UpdateEmployeeDetails (

p\_employee\_id IN NUMBER,

p\_name IN VARCHAR2,

p\_position IN VARCHAR2,

p\_salary IN NUMBER,

p\_department IN VARCHAR2

);

FUNCTION CalculateAnnualSalary (

p\_employee\_id IN NUMBER

) RETURN NUMBER;

END EmployeeManagement;

/

1. 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

) IS

BEGIN

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

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

COMMIT;

END HireEmployee;

PROCEDURE UpdateEmployeeDetails (

p\_employee\_id IN NUMBER,

p\_name IN VARCHAR2,

p\_position IN VARCHAR2,

p\_salary IN NUMBER,

p\_department IN VARCHAR2

) IS

BEGIN

UPDATE Employees

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

WHERE EmployeeID = p\_employee\_id;

COMMIT;

END UpdateEmployeeDetails;

FUNCTION CalculateAnnualSalary (

p\_employee\_id IN NUMBER

) RETURN NUMBER IS

v\_salary NUMBER;

v\_annual\_salary NUMBER;

BEGIN

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

v\_annual\_salary := v\_salary \* 12;

RETURN v\_annual\_salary;

END CalculateAnnualSalary;

END EmployeeManagement;

/

3) CREATE OR REPLACE PACKAGE AccountOperations AS

PROCEDURE OpenAccount (

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;

/

1. CREATE OR REPLACE PACKAGE BODY AccountOperations AS

PROCEDURE OpenAccount (

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 OpenAccount;

PROCEDURE CloseAccount (

p\_account\_id IN NUMBER

) IS

BEGIN

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

COMMIT;

END CloseAccount;

FUNCTION GetTotalBalance (

p\_customer\_id IN NUMBER

) RETURN NUMBER IS

v\_total\_balance NUMBER;

BEGIN

SELECT SUM(Balance) INTO v\_total\_balance FROM Accounts WHERE CustomerID = p\_customer\_id;

RETURN v\_total\_balance;

END GetTotalBalance;

END AccountOperations;

/