**Exercise 1: Control Structures**

**Scenario 1: Apply 1% interest discount for customers above 60**

**CODE:**

BEGIN

FOR rec IN (SELECT c.CustomerID, l.LoanID, l.InterestRate, c.DOB

FROM Customers c

JOIN Loans l ON c.CustomerID = l.CustomerID) LOOP

IF 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: Promote to VIP if balance > 10,000**

**CODE:**

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;

END IF;

END LOOP;

COMMIT;

END;

/

Scenario 3: Print loan reminders for due loans in next 30 days

**CODE:**

BEGIN

FOR rec IN (

SELECT c.Name, l.LoanID, 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, 'DD-MON-YYYY'));

END LOOP;

END;

/

**Exercise 2: Error Handling**

**Scenario 1: SafeTransferFunds Procedure (with rollback on error)**

**CODE:**

CREATE OR REPLACE PROCEDURE SafeTransferFunds(

p\_FromAccountID IN NUMBER,

p\_ToAccountID IN NUMBER,

p\_Amount IN NUMBER

) IS

v\_FromBalance NUMBER;

BEGIN

SELECT Balance INTO v\_FromBalance FROM Accounts WHERE AccountID = p\_FromAccountID FOR UPDATE;

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;

COMMIT;

EXCEPTION

WHEN OTHERS THEN

ROLLBACK;

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

END;

/

**Scenario 2: UpdateSalary Procedure (with exception handling for invalid employee ID)**

**CODE:**

CREATE OR REPLACE PROCEDURE UpdateSalary(

p\_EmployeeID IN NUMBER,

p\_Percentage IN NUMBER

) IS

BEGIN

UPDATE Employees

SET Salary = Salary + (Salary \* p\_Percentage / 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: AddNewCustomer Procedure (prevent duplicate customer ID)**

**CODE:**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 with ID ' || p\_CustomerID || ' already exists.');

WHEN OTHERS THEN

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

END;

/

**Exercise 3: Stored Procedures**

**Scenario 1: ProcessMonthlyInterest – apply 1% interest to all savings accounts**

**CODE:**CREATE OR REPLACE PROCEDURE ProcessMonthlyInterest IS

BEGIN

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

UPDATE Accounts

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

LastModified = SYSDATE

WHERE AccountID = acc.AccountID;

END LOOP;

COMMIT;

END;

/

**Scenario 2: UpdateEmployeeBonus – update salaries in a department**

**CODE:**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;

COMMIT;

END;

/

**Scenario 3: TransferFunds – transfer money between two accounts**

**CODE:**CREATE OR REPLACE PROCEDURE TransferFunds(

p\_SourceAccountID IN NUMBER,

p\_DestinationAccountID IN NUMBER,

p\_Amount IN NUMBER

) IS

v\_SourceBalance NUMBER;

BEGIN

SELECT Balance INTO v\_SourceBalance

FROM Accounts

WHERE AccountID = p\_SourceAccountID FOR UPDATE;

IF v\_SourceBalance < p\_Amount THEN

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

END IF;

-- Deduct from source

UPDATE Accounts

SET Balance = Balance - p\_Amount, LastModified = SYSDATE

WHERE AccountID = p\_SourceAccountID;

-- Add to destination

UPDATE Accounts

SET Balance = Balance + p\_Amount, LastModified = SYSDATE

WHERE AccountID = p\_DestinationAccountID;

COMMIT;

END;

/

**Exercise 4: Functions**

**Scenario 1: CalculateAge – return age in years from DOB**

**CODE:**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;

/

SELECT Name, CalculateAge(DOB) AS Age FROM Customers;

**Scenario 2: CalculateMonthlyInstallment**

**CODE:**CREATE OR REPLACE FUNCTION CalculateMonthlyInstallment(

p\_LoanAmount IN NUMBER,

p\_AnnualInterestRate IN NUMBER,

p\_DurationYears IN NUMBER

) RETURN NUMBER IS

r NUMBER := p\_AnnualInterestRate / 12 / 100;

n NUMBER := p\_DurationYears \* 12;

emi NUMBER;

BEGIN

IF r = 0 THEN

emi := p\_LoanAmount / n;

ELSE

emi := (p\_LoanAmount \* r \* POWER(1 + r, n)) /

(POWER(1 + r, n) - 1);

END IF;

RETURN ROUND(emi, 2);

END;

/

SELECT CalculateMonthlyInstallment(10000, 5, 5) AS Monthly\_Installment FROM dual;

**Scenario 3: HasSufficientBalance – returns TRUE if balance is enough**

**CODE:**

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;

/

DECLARE

result BOOLEAN;

BEGIN

result := HasSufficientBalance(1, 500);

IF result THEN

DBMS\_OUTPUT.PUT\_LINE('Sufficient balance.');

ELSE

DBMS\_OUTPUT.PUT\_LINE('Insufficient balance.');

END IF;

END;

/

**Exercise 5: Triggers**

**Scenario 1: UpdateCustomerLastModified – set LastModified to current date on update**

**CODE:**CREATE OR REPLACE TRIGGER UpdateCustomerLastModified

BEFORE UPDATE ON Customers

FOR EACH ROW

BEGIN

:NEW.LastModified := SYSDATE;

END;

/

**Scenario 2: LogTransaction – log every transaction in AuditLog table**

**CODE:**CREATE TABLE AuditLog (

LogID NUMBER GENERATED ALWAYS AS IDENTITY PRIMARY KEY,

TransactionID NUMBER,

AccountID NUMBER,

TransactionType VARCHAR2(10),

Amount NUMBER,

LoggedAt DATE

);

**Trigger:**CREATE OR REPLACE TRIGGER LogTransaction

AFTER INSERT ON Transactions

FOR EACH ROW

BEGIN

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

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

END;

/

**Scenario 3: CheckTransactionRules – enforce deposit/withdrawal rules**

**CODE:**CREATE OR REPLACE TRIGGER CheckTransactionRules

BEFORE INSERT ON Transactions

FOR EACH ROW

DECLARE

v\_Balance NUMBER;

BEGIN

IF :NEW.Amount <= 0 THEN

RAISE\_APPLICATION\_ERROR(-20004, 'Transaction amount must be positive.');

END IF;

IF :NEW.TransactionType = 'Withdrawal' THEN

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

IF v\_Balance < :NEW.Amount THEN

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

END IF;

END IF;

END;

/

**Exercise 6: Cursors**

**Scenario 1: GenerateMonthlyStatements – list all current month transactions per customer**

**CODE:**

DECLARE

CURSOR transaction\_cursor IS

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

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 txn IN transaction\_cursor LOOP

DBMS\_OUTPUT.PUT\_LINE('Customer: ' || txn.Name ||

', AccountID: ' || txn.AccountID ||

', Type: ' || txn.TransactionType ||

', Amount: ' || txn.Amount ||

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

END LOOP;

END;

/

**Scenario 2: ApplyAnnualFee – deduct annual fee from all accounts**

**CODE:**

DECLARE

CURSOR account\_cursor IS

SELECT AccountID, Balance FROM Accounts;

v\_Fee CONSTANT NUMBER := 100;

BEGIN

FOR acc IN account\_cursor LOOP

IF acc.Balance >= v\_Fee THEN

UPDATE Accounts

SET Balance = Balance - v\_Fee,

LastModified = SYSDATE

WHERE AccountID = acc.AccountID;

ELSE

DBMS\_OUTPUT.PUT\_LINE('Skipping Account ' || acc.AccountID || ' due to low balance.');

END IF;

END LOOP;

COMMIT;

END;

/

**Scenario 3: UpdateLoanInterestRates – update loans based on policy**

**CODE:**

DECLARE

CURSOR loan\_cursor IS

SELECT LoanID, LoanAmount, InterestRate FROM Loans;

BEGIN

FOR loan IN loan\_cursor LOOP

IF loan.LoanAmount > 10000 THEN

UPDATE Loans

SET InterestRate = InterestRate + 0.5

WHERE LoanID = loan.LoanID;

ELSE

UPDATE Loans

SET InterestRate = InterestRate - 0.5

WHERE LoanID = loan.LoanID;

END IF;

END LOOP;

COMMIT;

END;

/

**Exercise 7: Packages**

**Scenario 1: CustomerManagement – procedures and function for customers**

**PACKAGE SPECIFICATION:**

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\_Name VARCHAR2, p\_DOB DATE, p\_Balance NUMBER);

FUNCTION GetCustomerBalance(p\_ID NUMBER) RETURN NUMBER;

END CustomerManagement;

/

**PACKAGE BODY:**

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 (CustomerID, Name, DOB, Balance, LastModified)

VALUES (p\_ID, p\_Name, p\_DOB, p\_Balance, SYSDATE);

EXCEPTION

WHEN DUP\_VAL\_ON\_INDEX THEN

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

END;

PROCEDURE UpdateCustomer(p\_ID 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\_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;

EXCEPTION

WHEN NO\_DATA\_FOUND THEN

RETURN NULL;

END;

END CustomerManagement;

/

**Scenario 2: EmployeeManagement – procedures/functions for employees**

**PACKAGE SPECIFICATION:**

CREATE OR REPLACE PACKAGE EmployeeManagement AS

PROCEDURE HireEmployee(p\_ID NUMBER, p\_Name VARCHAR2, p\_Position VARCHAR2, p\_Salary NUMBER, p\_Dept VARCHAR2, p\_HireDate DATE);

PROCEDURE UpdateEmployee(p\_ID NUMBER, p\_Salary NUMBER, p\_Position VARCHAR2);

FUNCTION GetAnnualSalary(p\_ID NUMBER) RETURN NUMBER;

END EmployeeManagement;

/

**PACKAGE BODY:**

CREATE OR REPLACE PACKAGE BODY EmployeeManagement AS

PROCEDURE HireEmployee(p\_ID NUMBER, p\_Name VARCHAR2, p\_Position VARCHAR2, p\_Salary NUMBER, p\_Dept VARCHAR2, p\_HireDate DATE) IS

BEGIN

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

VALUES (p\_ID, p\_Name, p\_Position, p\_Salary, p\_Dept, p\_HireDate);

END;

PROCEDURE UpdateEmployee(p\_ID NUMBER, p\_Salary NUMBER, p\_Position VARCHAR2) IS

BEGIN

UPDATE Employees

SET Salary = p\_Salary, Position = p\_Position

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;

EXCEPTION

WHEN NO\_DATA\_FOUND THEN

RETURN NULL;

END;

END EmployeeManagement;

/

**Scenario 3: AccountOperations – procedures/functions for accounts**

**PACKAGE SPECIFICATION:**

CREATE OR REPLACE PACKAGE AccountOperations AS

PROCEDURE OpenAccount(p\_AccountID NUMBER, p\_CustomerID NUMBER, p\_Type VARCHAR2, p\_Balance NUMBER);

PROCEDURE CloseAccount(p\_AccountID NUMBER);

FUNCTION GetTotalCustomerBalance(p\_CustomerID NUMBER) RETURN NUMBER;

END AccountOperations;

/

**PACKAGE BODY:**

CREATE OR REPLACE PACKAGE BODY AccountOperations AS

PROCEDURE OpenAccount(p\_AccountID NUMBER, p\_CustomerID NUMBER, p\_Type VARCHAR2, p\_Balance NUMBER) IS

BEGIN

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

VALUES (p\_AccountID, p\_CustomerID, p\_Type, p\_Balance, SYSDATE);

END;

PROCEDURE CloseAccount(p\_AccountID NUMBER) IS

BEGIN

DELETE FROM Accounts WHERE AccountID = p\_AccountID;

END;

FUNCTION GetTotalCustomerBalance(p\_CustomerID NUMBER) RETURN NUMBER IS

v\_Total NUMBER;

BEGIN

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

RETURN v\_Total;

EXCEPTION

WHEN NO\_DATA\_FOUND THEN

RETURN 0;

END;

END AccountOperations;

/