**Exercise 1: Control Structures**

**Scenario 1: Discount on Loan Interest Rates for Customers Above 60**

DECLARE

v\_age NUMBER;

v\_discount NUMBER := 0.01;

BEGIN

FOR cust IN (SELECT CustomerID, TRUNC(MONTHS\_BETWEEN(SYSDATE, DOB) / 12) AS age

FROM Customers) LOOP

v\_age := cust.age;

IF v\_age > 60 THEN

UPDATE Loans

SET InterestRate = InterestRate - v\_discount

WHERE CustomerID = cust.CustomerID;

END IF;

END LOOP;

END;

/

**Scenario 2: Set VIP Status for Customers with High Balance**

DECLARE

v\_threshold NUMBER := 10000;

BEGIN

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

IF cust.Balance > v\_threshold THEN

UPDATE Customers

SET IsVIP = 'TRUE'

WHERE CustomerID = cust.CustomerID;

END IF;

END LOOP;

END;

/

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

BEGIN

FOR loan IN (SELECT LoanID, CustomerID, EndDate

FROM Loans

WHERE EndDate BETWEEN SYSDATE AND SYSDATE + 30) LOOP

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

', your loan (LoanID: ' || loan.LoanID ||

') is due on ' || loan.EndDate);

END LOOP;

END;

/

**Exercise 2: Error Handling**

**Scenario 1: SafeTransferFunds Stored Procedure**

CREATE OR REPLACE PROCEDURE SafeTransferFunds(

p\_from\_account NUMBER,

p\_to\_account NUMBER,

p\_amount NUMBER

) IS

v\_balance NUMBER;

BEGIN

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

IF v\_balance < p\_amount THEN

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

ELSE

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

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

END IF;

COMMIT;

EXCEPTION

WHEN OTHERS THEN

ROLLBACK;

DBMS\_OUTPUT.PUT\_LINE(SQLERRM);

END;

/

**Scenario 2: UpdateSalary Stored Procedure**

CREATE OR REPLACE PROCEDURE UpdateSalary(

p\_employee\_id NUMBER,

p\_percentage NUMBER

) IS

v\_salary NUMBER;

BEGIN

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

UPDATE Employees SET Salary = Salary + (v\_salary \* p\_percentage / 100) WHERE EmployeeID = p\_employee\_id;

COMMIT;

EXCEPTION

WHEN NO\_DATA\_FOUND THEN

DBMS\_OUTPUT.PUT\_LINE('Employee ID not found.');

WHEN OTHERS THEN

DBMS\_OUTPUT.PUT\_LINE(SQLERRM);

END;

/

**Scenario 3: AddNewCustomer Stored Procedure**

CREATE OR REPLACE PROCEDURE AddNewCustomer(

p\_customer\_id NUMBER,

p\_name VARCHAR2,

p\_dob DATE,

p\_balance 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

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

WHEN OTHERS THEN

DBMS\_OUTPUT.PUT\_LINE(SQLERRM);

END;

/

**Exercise 3: Stored Procedures**

**Scenario 1: ProcessMonthlyInterest**

CREATE OR REPLACE PROCEDURE ProcessMonthlyInterest IS

BEGIN

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

UPDATE Accounts

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

WHERE AccountID = acc.AccountID;

END LOOP;

COMMIT;

END;

/

**Scenario 2: UpdateEmployeeBonus**

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**

CREATE OR REPLACE PROCEDURE TransferFunds(

p\_from\_account NUMBER,

p\_to\_account NUMBER,

p\_amount NUMBER

) IS

v\_balance NUMBER;

BEGIN

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

IF v\_balance < p\_amount THEN

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

ELSE

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

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

END IF;

COMMIT;

EXCEPTION

WHEN OTHERS THEN

ROLLBACK;

DBMS\_OUTPUT.PUT\_LINE(SQLERRM);

END;

/

**Exercise 4: Functions**

**Scenario 1: CalculateAge**

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

v\_age NUMBER;

BEGIN

SELECT TRUNC(MONTHS\_BETWEEN(SYSDATE, p\_dob) / 12) INTO v\_age FROM DUAL;

RETURN v\_age;

END;

/

**Scenario 2: CalculateMonthlyInstallment**

CREATE OR REPLACE FUNCTION CalculateMonthlyInstallment(

p\_loan\_amount NUMBER,

p\_interest\_rate NUMBER,

p\_duration\_years NUMBER

) RETURN NUMBER IS

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

v\_months NUMBER := p\_duration\_years \* 12;

v\_monthly\_installment NUMBER;

BEGIN

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

RETURN v\_monthly\_installment;

END;

/

**Scenario 3: HasSufficientBalance**

CREATE OR REPLACE FUNCTION HasSufficientBalance(

p\_account\_id NUMBER,

p\_amount NUMBER

) RETURN BOOLEAN IS

v\_balance NUMBER;

BEGIN

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

RETURN v\_balance >= p\_amount;

END;

/

**Exercise 5: Triggers**

**Scenario 1: UpdateCustomerLastModified**

CREATE OR REPLACE TRIGGER UpdateCustomerLastModified

BEFORE UPDATE ON Customers

FOR EACH ROW

BEGIN

:NEW.LastModified := SYSDATE;

END;

/

**Scenario 2: LogTransaction**

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**

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(-20002, 'Withdrawal amount exceeds account balance.');

END IF;

ELSIF :NEW.TransactionType = 'Deposit' THEN

IF :NEW.Amount <= 0 THEN

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

END IF;

END IF;

END;

/

**Exercise 6: Cursors**

**Scenario 1: GenerateMonthlyStatements**

DECLARE

CURSOR c\_transactions IS

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

FROM Transactions t

JOIN Accounts a ON t.AccountID = a.AccountID

JOIN Customers c ON a.CustomerID = c.CustomerID

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

BEGIN

FOR r IN c\_transactions LOOP

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

', Transaction ID: ' || r.TransactionID ||

', Date: ' || r.TransactionDate ||

', Amount: ' || r.Amount ||

', Type: ' || r.TransactionType);

END LOOP;

END;

/

**Scenario 2: ApplyAnnualFee**

DECLARE

CURSOR c\_accounts IS

SELECT AccountID, Balance FROM Accounts;

v\_fee NUMBER := 50; -- Annual maintenance fee

BEGIN

FOR r IN c\_accounts LOOP

UPDATE Accounts SET Balance = Balance - v\_fee WHERE AccountID = r.AccountID;

END LOOP;

COMMIT;

END;

/

**Scenario 3: UpdateLoanInterestRates**

DECLARE

CURSOR c\_loans IS

SELECT LoanID, InterestRate FROM Loans;

v\_new\_rate NUMBER := 0.06; -- New interest rate policy

BEGIN

FOR r IN c\_loans LOOP

UPDATE Loans SET InterestRate = v\_new\_rate WHERE LoanID = r.LoanID;

END LOOP;

COMMIT;

END;

/

**Exercise 7: Packages**

**Scenario 1: CustomerManagement Package**

CREATE OR REPLACE PACKAGE CustomerManagement AS

PROCEDURE AddNewCustomer(p\_customer\_id NUMBER, p\_name VARCHAR2, p\_dob DATE, p\_balance NUMBER);

PROCEDURE UpdateCustomerDetails(p\_customer\_id NUMBER, p\_name VARCHAR2, p\_dob DATE, p\_balance NUMBER);

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

END CustomerManagement;

/

CREATE OR REPLACE PACKAGE BODY CustomerManagement AS

PROCEDURE AddNewCustomer(p\_customer\_id NUMBER, p\_name VARCHAR2, p\_dob DATE, p\_balance 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 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\_customer\_id;

COMMIT;

END UpdateCustomerDetails;

FUNCTION GetCustomerBalance(p\_customer\_id 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;

/

**Scenario 2: EmployeeManagement Package**

CREATE OR REPLACE PACKAGE EmployeeManagement AS

PROCEDURE HireEmployee(p\_employee\_id NUMBER, p\_name VARCHAR2, p\_position VARCHAR2, p\_salary NUMBER, p\_department VARCHAR2, p\_hiredate DATE);

PROCEDURE UpdateEmployeeDetails(p\_employee\_id NUMBER, p\_name VARCHAR2, p\_position VARCHAR2, p\_salary NUMBER, p\_department VARCHAR2);

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

END EmployeeManagement;

/

CREATE OR REPLACE PACKAGE BODY EmployeeManagement AS

PROCEDURE HireEmployee(p\_employee\_id 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\_employee\_id, p\_name, p\_position, p\_salary, p\_department, p\_hiredate);

COMMIT;

END HireEmployee;

PROCEDURE UpdateEmployeeDetails(p\_employee\_id NUMBER, p\_name VARCHAR2, p\_position VARCHAR2, p\_salary NUMBER, p\_department 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 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 CalculateAnnualSalary;

END EmployeeManagement;

/

**Scenario 3: AccountOperations Package**

CREATE OR REPLACE PACKAGE AccountOperations AS

PROCEDURE OpenAccount(p\_account\_id NUMBER, p\_customer\_id NUMBER, p\_account\_type VARCHAR2, p\_balance NUMBER);

PROCEDURE CloseAccount(p\_account\_id NUMBER);

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

END AccountOperations;

/

CREATE OR REPLACE PACKAGE BODY AccountOperations AS

PROCEDURE OpenAccount(p\_account\_id NUMBER, p\_customer\_id NUMBER, p\_account\_type VARCHAR2, p\_balance 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 NUMBER) IS

BEGIN

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

COMMIT;

END CloseAccount;

FUNCTION GetTotalBalance(p\_customer\_id 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;

/