**EXERCISE 1:**

Scenario 1:

DECLARE

v\_customer\_age NUMBER;

v\_current\_rate NUMBER;

CURSOR c\_customers IS

SELECT CustomerID, (EXTRACT(YEAR FROM SYSDATE) - EXTRACT(YEAR FROM DOB)) AS Age

FROM Customers;

BEGIN

FOR customer\_rec IN c\_customers LOOP

v\_customer\_age := customer\_rec.Age;

IF v\_customer\_age > 60 THEN

FOR loan\_rec IN (

SELECT LoanID, InterestRate

FROM Loans

WHERE CustomerID = customer\_rec.CustomerID

) LOOP

v\_current\_rate := loan\_rec.InterestRate;

UPDATE Loans

SET InterestRate = v\_current\_rate - 1

WHERE LoanID = loan\_rec.LoanID;

END LOOP;

END IF;

END LOOP;

COMMIT;

END;

/

Scenario 2:

ALTER TABLE Customers ADD (IsVIP VARCHAR2(3) DEFAULT 'NO');

Scenario 3:

BEGIN

FOR customer\_rec IN (

SELECT CustomerID, Balance

FROM Customers

) LOOP

IF customer\_rec.Balance > 10000 THEN

UPDATE Customers

SET IsVIP = 'YES'

WHERE CustomerID = customer\_rec.CustomerID;

ELSE

UPDATE Customers

SET IsVIP = 'NO'

WHERE CustomerID = customer\_rec.CustomerID;

END IF;

END LOOP;

COMMIT;

END;

/

**EXERCISE 2:**

Scenario 1:

CREATE SEQUENCE TRANSACTION\_SEQ

START WITH 1

INCREMENT BY 1;

CREATE OR REPLACE PROCEDURE SafeTransferFunds (

p\_from\_account\_id IN NUMBER,

p\_to\_account\_id IN NUMBER,

p\_amount IN NUMBER

) AS

v\_from\_balance NUMBER;

v\_to\_balance NUMBER;

BEGIN

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

IF v\_from\_balance < p\_amount THEN

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

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;

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

VALUES (TRANSACTION\_SEQ.NEXTVAL, p\_from\_account\_id, SYSDATE, p\_amount, 'Transfer Out');

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

VALUES (TRANSACTION\_SEQ.NEXTVAL, p\_to\_account\_id, SYSDATE, p\_amount, 'Transfer In');

COMMIT;

EXCEPTION

WHEN OTHERS THEN

ROLLBACK;

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

END SafeTransferFunds;

/

Scenario 2:

CREATE OR REPLACE PROCEDURE UpdateSalary (

p\_employee\_id IN NUMBER,

p\_percentage IN NUMBER

) AS

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

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

WHEN OTHERS THEN

ROLLBACK;

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

END UpdateSalary;

/

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

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 ID ' || p\_customer\_id || ' already exists.');

WHEN OTHERS THEN

ROLLBACK;

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

END AddNewCustomer;

/

**EXERCISE 3:**

Scenario 1:

CREATE OR REPLACE PROCEDURE ProcessMonthlyInterest AS

BEGIN

UPDATE Accounts

SET Balance = Balance \* 1.01

WHERE AccountType = 'Savings';

COMMIT;

END ProcessMonthlyInterest;

/

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 UpdateEmployeeBonus;

/

Scenario 3:

CREATE OR REPLACE PROCEDURE TransferFunds (

p\_from\_account\_id IN NUMBER,

p\_to\_account\_id IN NUMBER,

p\_amount IN NUMBER

) AS

v\_from\_balance NUMBER;

BEGIN

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

IF v\_from\_balance < p\_amount THEN

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

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;

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

VALUES (TRANSACTION\_SEQ.NEXTVAL, p\_from\_account\_id, SYSDATE, p\_amount, 'Transfer Out');

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

VALUES (TRANSACTION\_SEQ.NEXTVAL, p\_to\_account\_id, SYSDATE, p\_amount, 'Transfer In');

COMMIT;

EXCEPTION

WHEN OTHERS THEN

ROLLBACK;

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

END TransferFunds;

/

**EXERCISE 4:**

Scenario 1:

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 CalculateAge;

/

Scenario 2:

CREATE OR REPLACE FUNCTION CalculateMonthlyInstallment (

p\_loan\_amount NUMBER,

p\_interest\_rate NUMBER,

p\_loan\_duration\_years NUMBER

) RETURN NUMBER IS

v\_monthly\_rate NUMBER;

v\_num\_payments NUMBER;

v\_monthly\_installment NUMBER;

BEGIN

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

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

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

RETURN v\_monthly\_installment;

END CalculateMonthlyInstallment;

/

Scenario 3:

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;

IF v\_balance >= p\_amount THEN

RETURN TRUE;

ELSE

RETURN FALSE;

END IF;

END HasSufficientBalance;

/

**EXERCISE 5:**

Scenario 1:

CREATE OR REPLACE TRIGGER UpdateCustomerLastModified

BEFORE UPDATE ON Customers

FOR EACH ROW

BEGIN

:NEW.LastModified := SYSDATE;

END UpdateCustomerLastModified;

/

Scenario 2:

CREATE TABLE AuditLog (

LogID NUMBER PRIMARY KEY,

TransactionID NUMBER,

LogDate DATE,

Action VARCHAR2(50),

FOREIGN KEY (TransactionID) REFERENCES Transactions(TransactionID)

);

CREATE SEQUENCE AUDITLOG\_SEQ

START WITH 1

INCREMENT BY 1;

CREATE OR REPLACE TRIGGER LogTransaction

AFTER INSERT ON Transactions

FOR EACH ROW

BEGIN

INSERT INTO AuditLog (LogID, TransactionID, LogDate, Action)

VALUES (AUDITLOG\_SEQ.NEXTVAL, :NEW.TransactionID, SYSDATE, 'INSERT');

END LogTransaction;

/

Scenario 3:

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 v\_balance < :NEW.Amount THEN

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

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 CheckTransactionRules;

/

**EXERCISE 6:**

Scenario 1:

DECLARE

CURSOR c\_monthly\_transactions IS

SELECT c.CustomerID, c.Name, t.TransactionID, 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 EXTRACT(MONTH FROM t.TransactionDate) = EXTRACT(MONTH FROM SYSDATE)

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

v\_customer\_id Customers.CustomerID%TYPE;

v\_customer\_name Customers.Name%TYPE;

v\_transaction\_id Transactions.TransactionID%TYPE;

v\_transaction\_date Transactions.TransactionDate%TYPE;

v\_amount Transactions.Amount%TYPE;

v\_transaction\_type Transactions.TransactionType%TYPE;

BEGIN

OPEN c\_monthly\_transactions;

LOOP

FETCH c\_monthly\_transactions INTO v\_customer\_id, v\_customer\_name, v\_transaction\_id, v\_transaction\_date, v\_amount, v\_transaction\_type;

EXIT WHEN c\_monthly\_transactions%NOTFOUND;

DBMS\_OUTPUT.PUT\_LINE('Customer ID: ' || v\_customer\_id);

DBMS\_OUTPUT.PUT\_LINE('Customer Name: ' || v\_customer\_name);

DBMS\_OUTPUT.PUT\_LINE('Transaction ID: ' || v\_transaction\_id);

DBMS\_OUTPUT.PUT\_LINE('Date: ' || TO\_CHAR(v\_transaction\_date, 'YYYY-MM-DD'));

DBMS\_OUTPUT.PUT\_LINE('Amount: ' || v\_amount);

DBMS\_OUTPUT.PUT\_LINE('Type: ' || v\_transaction\_type);

DBMS\_OUTPUT.PUT\_LINE('-----------------------------------');

END LOOP;

CLOSE c\_monthly\_transactions;

END;

/

Scenario 2:

DECLARE

CURSOR c\_accounts IS

SELECT AccountID, Balance

FROM Accounts;

v\_account\_id Accounts.AccountID%TYPE;

v\_balance Accounts.Balance%TYPE;

v\_annual\_fee NUMBER := 50;

BEGIN

OPEN c\_accounts;

LOOP

FETCH c\_accounts INTO v\_account\_id, v\_balance;

EXIT WHEN c\_accounts%NOTFOUND;

UPDATE Accounts

SET Balance = v\_balance - v\_annual\_fee

WHERE AccountID = v\_account\_id;

END LOOP;

CLOSE c\_accounts;

COMMIT;

END;

/

Scenario 3:

DECLARE

CURSOR c\_loans IS

SELECT LoanID, InterestRate

FROM Loans;

v\_loan\_id Loans.LoanID%TYPE;

v\_current\_interest\_rate Loans.InterestRate%TYPE;

v\_new\_interest\_rate NUMBER := 4.5;

BEGIN

OPEN c\_loans;

LOOP

FETCH c\_loans INTO v\_loan\_id, v\_current\_interest\_rate;

EXIT WHEN c\_loans%NOTFOUND;

UPDATE Loans

SET InterestRate = v\_new\_interest\_rate

WHERE LoanID = v\_loan\_id;

END LOOP;

CLOSE c\_loans;

COMMIT;

END;

/

**EXERCISE 7:**

Scenario 1:

CREATE OR REPLACE PACKAGE CustomerManagement AS

PROCEDURE AddCustomer(

p\_customer\_id IN NUMBER,

p\_name IN VARCHAR2,

p\_dob IN DATE,

p\_balance IN NUMBER

);

PROCEDURE UpdateCustomer(

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 AddCustomer(

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 AddCustomer;

PROCEDURE UpdateCustomer(

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 UpdateCustomer;

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;

EXCEPTION

WHEN NO\_DATA\_FOUND THEN

RETURN 0;

END GetCustomerBalance;

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

);

PROCEDURE UpdateEmployee(

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;

/

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 UpdateEmployee(

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 UpdateEmployee;

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;

EXCEPTION

WHEN NO\_DATA\_FOUND THEN

RETURN 0;

END CalculateAnnualSalary;

END EmployeeManagement;

/

Scenario 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;

/

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 NVL(v\_total\_balance, 0);

END GetTotalBalance;

END AccountOperations;

/