Exercise 1:

Scenario 1:

DECLARE

v\_customer\_age NUMBER;

CURSOR cur\_customers IS

SELECT CustomerID, DOB FROM Customers;

BEGIN

FOR cust\_rec IN cur\_customers LOOP

v\_customer\_age := TRUNC(MONTHS\_BETWEEN(SYSDATE, cust\_rec.DOB) / 12);

IF v\_customer\_age > 60 THEN

UPDATE Loans

SET InterestRate = InterestRate - 1

WHERE CustomerID = cust\_rec.CustomerID;

END IF;

END LOOP;

COMMIT;

END;

/

Scenario 2:

BEGIN

UPDATE Customers

SET IsVIP = 'YES'

WHERE Balance > 10000

COMMIT;

END;

/

Scenario 3:

DECLARE

CURSOR cur\_loans IS

SELECT LoanID, CustomerID, EndDate

FROM Loans

WHERE EndDate <= SYSDATE + 30;

v\_customer\_name VARCHAR2(100);

BEGIN

FOR loan\_rec IN cur\_loans LOOP

SELECT Name INTO v\_customer\_name

FROM Customers

WHERE CustomerID = loan\_rec.CustomerID;

DBMS\_OUTPUT.PUT\_LINE('Reminder: Customer ' || v\_customer\_name ||

' has a loan due on ' || TO\_CHAR(loan\_rec.EndDate, 'YYYY-MM-DD'));

END LOOP;

END;

/

Exercise 2:

Scenario 1:

CREATE OR REPLACE PROCEDURE SafeTransferFunds(

p\_from\_account\_id IN NUMBER,

p\_to\_account\_id IN NUMBER,

p\_amount IN NUMBER

) IS

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 account.');

END IF;

UPDATE Accounts

SET Balance = Balance - p\_amount,

LastModified = SYSDATE

WHERE AccountID = p\_from\_account\_id;

UPDATE Accounts

SET Balance = Balance + p\_amount,

LastModified = SYSDATE

WHERE AccountID = p\_to\_account\_id;

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

) IS

v\_current\_salary NUMBER;

BEGIN

SELECT Salary INTO v\_current\_salary

FROM Employees

WHERE EmployeeID = p\_employee\_id;

UPDATE Employees

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

HireDate = SYSDATE

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

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

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

WHEN OTHERS THEN

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

END AddNewCustomer;

/

Exercise 3:

Scenario 1:

CREATE OR REPLACE PROCEDURE ProcessMonthlyInterest IS

BEGIN

UPDATE Accounts

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

LastModified = SYSDATE

WHERE AccountType = 'Savings';

COMMIT;

END ProcessMonthlyInterest;

/

Scenario 2:

CREATE OR REPLACE PROCEDURE UpdateEmployeeBonus(

p\_department IN VARCHAR2,

p\_bonus\_percentage IN NUMBER

) IS

BEGIN

UPDATE Employees

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

HireDate = SYSDATE

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

) IS

v\_from\_balance NUMBER;

BEGIN

SELECT Balance INTO v\_from\_balance

FROM Accounts

WHERE AccountID = p\_from\_account\_id

FOR UPDATE;

IF v\_from\_balance < p\_amount THEN

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

END IF;

UPDATE Accounts

SET Balance = Balance - p\_amount,

LastModified = SYSDATE

WHERE AccountID = p\_from\_account\_id;

UPDATE Accounts

SET Balance = Balance + p\_amount,

LastModified = SYSDATE

WHERE AccountID = p\_to\_account\_id;

COMMIT;

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\_duration\_years NUMBER

) RETURN NUMBER IS

v\_monthly\_rate NUMBER;

v\_installment NUMBER;

BEGIN

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

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

(1 - POWER(1 + v\_monthly\_rate, -p\_duration\_years \* 12));

RETURN v\_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;

RETURN v\_balance >= p\_amount;

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 (

AuditID NUMBER PRIMARY KEY,

TransactionID NUMBER,

AccountID NUMBER,

TransactionDate DATE,

Amount NUMBER,

TransactionType VARCHAR2(10),

LogDate DATE

);

CREATE OR REPLACE TRIGGER LogTransaction

AFTER INSERT ON Transactions

FOR EACH ROW

BEGIN

INSERT INTO AuditLog (AuditID, TransactionID, AccountID, TransactionDate, Amount, TransactionType, LogDate)

VALUES (

AuditLog\_SEQ.NEXTVAL,

:NEW.TransactionID,

:NEW.AccountID,

:NEW.TransactionDate,

:NEW.Amount,

:NEW.TransactionType,

SYSDATE

);

END LogTransaction;

/

Scenario 3:

CREATE OR REPLACE TRIGGER CheckTransactionRules

BEFORE INSERT ON Transactions

FOR EACH ROW

DECLARE

v\_balance NUMBER;

BEGIN

SELECT Balance INTO v\_balance

FROM Accounts

WHERE AccountID = :NEW.AccountID

FOR UPDATE;

IF :NEW.TransactionType = 'Withdrawal' AND :NEW.Amount > v\_balance THEN

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

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

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

END IF;

END CheckTransactionRules;

Exercise 6:

Scenario 1:

DECLARE

CURSOR cur\_statements IS

SELECT c.CustomerID, c.Name, 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 TRUNC(t.TransactionDate, 'MM') = TRUNC(SYSDATE, 'MM');

v\_customer\_name VARCHAR2(100);

v\_transaction\_date DATE;

v\_amount NUMBER;

v\_transaction\_type VARCHAR2(10);

BEGIN

OPEN cur\_statements;

LOOP

FETCH cur\_statements INTO v\_customer\_name, v\_transaction\_date, v\_amount, v\_transaction\_type;

EXIT WHEN cur\_statements%NOTFOUND;

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

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

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

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

END LOOP;

CLOSE cur\_statements;

END;

/

Scenario 2:

DECLARE

CURSOR cur\_accounts IS

SELECT AccountID, Balance FROM Accounts;

v\_account\_id NUMBER;

v\_balance NUMBER;

v\_fee CONSTANT NUMBER := 50; -- Annual fee amount

BEGIN

OPEN cur\_accounts;

LOOP

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

EXIT WHEN cur\_accounts%NOTFOUND;

-- Deduct fee from the account balance

UPDATE Accounts

SET Balance = Balance - v\_fee

WHERE AccountID = v\_account\_id;

END LOOP;

CLOSE cur\_accounts;

COMMIT;

END;

/

Scenario 3:

DECLARE

CURSOR cur\_loans IS

SELECT LoanID, InterestRate FROM Loans;

v\_loan\_id NUMBER;

v\_interest\_rate NUMBER;

BEGIN

OPEN cur\_loans;

LOOP

FETCH cur\_loans INTO v\_loan\_id, v\_interest\_rate;

EXIT WHEN cur\_loans%NOTFOUND;

-- Update interest rate based on the current rate

IF v\_interest\_rate < 5 THEN

UPDATE Loans

SET InterestRate = v\_interest\_rate + 0.5

WHERE LoanID = v\_loan\_id;

ELSE

UPDATE Loans

SET InterestRate = v\_interest\_rate + 0.25

WHERE LoanID = v\_loan\_id;

END IF;

END LOOP;

CLOSE cur\_loans;

COMMIT;

END;

/

Exercise 7:

Scenario 1:

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 UpdateCustomer(

p\_customer\_id IN NUMBER,

p\_name IN VARCHAR2,

p\_dob IN DATE,

p\_balance IN NUMBER

) IS

BEGIN

UPDATE Customers

SET Name = p\_name, DOB = p\_dob, Balance = p\_balance, 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;

END GetCustomerBalance;

END CustomerManagement;

/

Scenario 2:

CREATE OR REPLACE PACKAGE BODY EmployeeManagement AS

PROCEDURE HireNewEmployee(

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 HireNewEmployee;

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;

END CalculateAnnualSalary;

END EmployeeManagement;

/

Scenario 3:

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 OpenNewAccount;

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;

/