Exercise 1:

* Scenario 1:

BEGIN

FOR customer IN (SELECT CustomerID, TRUNC((SYSDATE - DOB)/365) AS Age FROM Customers) LOOP

IF customer.Age > 60 THEN

UPDATE Loans

SET InterestRate = InterestRate - 1

WHERE CustomerID = customer.CustomerID;

END IF;

END LOOP;

END;

* Scenario 2:

ALTER TABLE Customers ADD IsVIP VARCHAR2(3);

BEGIN

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

IF customer.Balance > 10000 THEN

UPDATE Customers

SET IsVIP = 'YES'

WHERE CustomerID = customer.CustomerID;

END IF;

END LOOP;

END;

* Scenario 3:

BEGIN

FOR loan IN (SELECT CustomerID, LoanID FROM Loans WHERE EndDate BETWEEN SYSDATE AND SYSDATE + 30) LOOP

DECLARE

v\_customer\_name VARCHAR2(100);

BEGIN

SELECT Name INTO v\_customer\_name FROM Customers WHERE CustomerID = loan.CustomerID;

DBMS\_OUTPUT.PUT\_LINE('Reminder: Dear ' || v\_customer\_name || ', your loan (ID: ' || loan.LoanID || ') is due within the next 30 days.');

END;

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

insufficient\_funds EXCEPTION;

BEGIN

-- Check balance of the from account

DECLARE

v\_balance NUMBER;

BEGIN

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

IF v\_balance < p\_amount THEN

RAISE insufficient\_funds;

ELSE

-- Deduct from the source account

UPDATE Accounts

SET Balance = Balance - p\_amount

WHERE AccountID = p\_from\_account\_id;

-- Add to the destination account

UPDATE Accounts

SET Balance = Balance + p\_amount

WHERE AccountID = p\_to\_account\_id;

COMMIT;

END IF;

END;

EXCEPTION

WHEN insufficient\_funds THEN

ROLLBACK;

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

WHEN OTHERS THEN

ROLLBACK;

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

END SafeTransferFunds;

* Scenario 2:

CREATE OR REPLACE PROCEDURE UpdateSalary (

p\_employee\_id IN NUMBER,

p\_percentage IN NUMBER

) IS

employee\_not\_found EXCEPTION;

BEGIN

-- Try to update the employee's salary

UPDATE Employees

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

WHERE EmployeeID = p\_employee\_id;

-- Check if the update was successful

IF SQL%ROWCOUNT = 0 THEN

RAISE employee\_not\_found;

END IF;

COMMIT;

EXCEPTION

WHEN employee\_not\_found THEN

ROLLBACK;

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

WHEN OTHERS THEN

ROLLBACK;

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

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

customer\_exists EXCEPTION;

BEGIN

-- Attempt to insert a new customer

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 the customer.');

END AddNewCustomer;

Exercise 3:

* Scenario 1:

CREATE OR REPLACE PROCEDURE ProcessMonthlyInterest IS

BEGIN

-- Update savings accounts with a 1% interest rate

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

) IS

BEGIN

-- Update the salary by adding the bonus percentage

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

) IS

v\_balance NUMBER;

BEGIN

-- Check balance of the from account

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

IF v\_balance < p\_amount THEN

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

RETURN;

END IF;

-- Deduct from the source account

UPDATE Accounts

SET Balance = Balance - p\_amount

WHERE AccountID = p\_from\_account\_id;

-- Add to the destination account

UPDATE Accounts

SET Balance = Balance + p\_amount

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((SYSDATE - p\_dob) / 365);

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\_installment NUMBER;

v\_monthly\_rate NUMBER;

v\_total\_months NUMBER;

BEGIN

v\_monthly\_rate := p\_interest\_rate / 1200; -- Annual to monthly interest rate

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

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

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

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;

EXCEPTION

WHEN NO\_DATA\_FOUND THEN

RETURN FALSE; -- Account not found

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 GENERATED BY DEFAULT AS IDENTITY PRIMARY KEY,

TransactionID NUMBER,

ActionDate DATE,

Action VARCHAR2(50)

);

-- trigger

CREATE OR REPLACE TRIGGER LogTransaction

AFTER INSERT ON Transactions

FOR EACH ROW

BEGIN

INSERT INTO AuditLog (TransactionID, ActionDate, Action)

VALUES (:NEW.TransactionID, SYSDATE, 'Transaction Inserted');

END LogTransaction;

* Scenario 3:

CREATE OR REPLACE TRIGGER CheckTransactionRules

BEFORE INSERT ON Transactions

FOR EACH ROW

DECLARE

v\_balance NUMBER;

BEGIN

-- Ensure deposits are positive

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

RAISE\_APPLICATION\_ERROR(-20001, 'Deposit amount must be positive.');

END IF;

-- Ensure withdrawals do not exceed balance

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 balance for withdrawal.');

END IF;

END IF;

END CheckTransactionRules;

Exercise 6:

* Scenario 1:

DECLARE

CURSOR c\_monthly\_transactions IS

SELECT DISTINCT CustomerID

FROM Transactions

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

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

v\_customer\_id Customers.CustomerID%TYPE;

BEGIN

FOR r IN c\_monthly\_transactions LOOP

v\_customer\_id := r.CustomerID;

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

FOR transaction IN (

SELECT \*

FROM Transactions

WHERE CustomerID = v\_customer\_id

AND EXTRACT(MONTH FROM TransactionDate) = EXTRACT(MONTH FROM SYSDATE)

AND EXTRACT(YEAR FROM TransactionDate) = EXTRACT(YEAR FROM SYSDATE)

) LOOP

DBMS\_OUTPUT.PUT\_LINE('Transaction ID: ' || transaction.TransactionID ||

', Amount: ' || transaction.Amount ||

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

END LOOP;

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

END LOOP;

END;

* Scenario 2:

DECLARE

CURSOR c\_accounts IS

SELECT AccountID, Balance

FROM Accounts;

v\_fee NUMBER := 500; -- Example annual fee amount

BEGIN

FOR r IN c\_accounts LOOP

UPDATE Accounts

SET Balance = Balance - v\_fee

WHERE AccountID = r.AccountID;

END LOOP;

COMMIT;

END;

* Scenario 3:

DECLARE

CURSOR c\_loans IS

SELECT LoanID, InterestRate

FROM Loans;

v\_new\_interest\_rate NUMBER := 6; -- Example new interest rate

BEGIN

FOR r IN c\_loans LOOP

UPDATE Loans

SET InterestRate = v\_new\_interest\_rate

WHERE LoanID = r.LoanID;

END LOOP;

COMMIT;

END;

Exercise 7: (Implementation of the procedure and functions are already provided in previous answers)

* Scenario 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\_balance IN NUMBER

);

FUNCTION GetCustomerBalance(

p\_customer\_id IN NUMBER

) RETURN NUMBER;

END CustomerManagement;

* Scenario 2:

CREATE OR REPLACE PACKAGE 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

);

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;

* Scenario 3:

CREATE OR REPLACE PACKAGE AccountOperations AS

PROCEDURE OpenNewAccount(

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;