**Pl/Sql**

CREATE TABLE Customers (

CustomerID NUMBER PRIMARY KEY,

Name VARCHAR2(100),

DOB DATE,

Balance NUMBER,

LastModified DATE

);

CREATE TABLE Accounts (

AccountID NUMBER PRIMARY KEY,

CustomerID NUMBER,

AccountType VARCHAR2(20),

Balance NUMBER,

LastModified DATE,

FOREIGN KEY (CustomerID) REFERENCES Customers(CustomerID)

);

CREATE TABLE Transactions (

TransactionID NUMBER PRIMARY KEY,

AccountID NUMBER,

TransactionDate DATE,

Amount NUMBER,

TransactionType VARCHAR2(10),

FOREIGN KEY (AccountID) REFERENCES Accounts(AccountID)

);

CREATE TABLE Loans (

LoanID NUMBER PRIMARY KEY,

CustomerID NUMBER,

LoanAmount NUMBER,

InterestRate NUMBER,

StartDate DATE,

EndDate DATE,

FOREIGN KEY (CustomerID) REFERENCES Customers(CustomerID)

);

CREATE TABLE Employees (

EmployeeID NUMBER PRIMARY KEY,

Name VARCHAR2(100),

Position VARCHAR2(50),

Salary NUMBER,

Department VARCHAR2(50),

HireDate DATE

);

Insert all

INTO Customers (CustomerID, Name, DOB, Balance, LastModified) VALUES (1, 'John Doe', TO\_DATE('1980-05-15', 'YYYY-MM-DD'), 1500.00, TO\_DATE('2024-07-01', 'YYYY-MM-DD'))

INTO Customers (CustomerID, Name, DOB, Balance, LastModified) VALUES (2, 'Jane Smith', TO\_DATE('1990-11-23', 'YYYY-MM-DD'), 2500.50, TO\_DATE('2024-07-02', 'YYYY-MM-DD'))

INTO Customers (CustomerID, Name, DOB, Balance, LastModified) VALUES (3, 'Robert Brown', TO\_DATE('1975-04-10', 'YYYY-MM-DD'), 3200.00, TO\_DATE('2024-07-03', 'YYYY-MM-DD'))

INTO Customers (CustomerID, Name, DOB, Balance, LastModified) VALUES (4, 'Emily Davis', TO\_DATE('1988-08-30', 'YYYY-MM-DD'), 4100.75, TO\_DATE('2024-07-04', 'YYYY-MM-DD'))

INTO Customers (CustomerID, Name, DOB, Balance, LastModified) VALUES (5, 'Michael Johnson', TO\_DATE('1995-03-14', 'YYYY-MM-DD'), 5400.20, TO\_DATE('2024-07-05', 'YYYY-MM-DD'))

INTO Customers (CustomerID, Name, DOB, Balance, LastModified) VALUES (6, 'Jessica White', TO\_DATE('1982-01-21', 'YYYY-MM-DD'), 6300.80, TO\_DATE('2024-07-06', 'YYYY-MM-DD'))

INTO Customers (CustomerID, Name, DOB, Balance, LastModified) VALUES (7, 'Daniel Garcia', TO\_DATE('1978-12-05', 'YYYY-MM-DD'), 7200.65, TO\_DATE('2024-07-07', 'YYYY-MM-DD'))

Select \* from dual;

Insert all

INTO Accounts (AccountID, CustomerID, AccountType, Balance, LastModified) VALUES (1, 1, 'Savings', 1500.00, TO\_DATE('2024-07-01', 'YYYY-MM-DD'))

INTO Accounts (AccountID, CustomerID, AccountType, Balance, LastModified) VALUES (2, 2, 'Checking', 2500.50, TO\_DATE('2024-07-02', 'YYYY-MM-DD'))

INTO Accounts (AccountID, CustomerID, AccountType, Balance, LastModified) VALUES (3, 3, 'Savings', 3200.00, TO\_DATE('2024-07-03', 'YYYY-MM-DD'))

INTO Accounts (AccountID, CustomerID, AccountType, Balance, LastModified) VALUES (4, 4, 'Checking', 4100.75, TO\_DATE('2024-07-04', 'YYYY-MM-DD'))

INTO Accounts (AccountID, CustomerID, AccountType, Balance, LastModified) VALUES (5, 5, 'Savings', 5400.20, TO\_DATE('2024-07-05', 'YYYY-MM-DD'))

INTO Accounts (AccountID, CustomerID, AccountType, Balance, LastModified) VALUES (6, 6, 'Checking', 6300.80, TO\_DATE('2024-07-06', 'YYYY-MM-DD'))

INTO Accounts (AccountID, CustomerID, AccountType, Balance, LastModified) VALUES (7, 7, 'Savings', 7200.65, TO\_DATE('2024-07-07', 'YYYY-MM-DD'))

Select \* from dual;

Insert all

INTO Transactions (TransactionID, AccountID, TransactionDate, Amount, TransactionType) VALUES (1, 1, TO\_DATE('2024-07-01', 'YYYY-MM-DD'), 200.00, 'Credit')

INTO Transactions (TransactionID, AccountID, TransactionDate, Amount, TransactionType) VALUES (2, 2, TO\_DATE('2024-07-02', 'YYYY-MM-DD'), 300.50, 'Debit')

INTO Transactions (TransactionID, AccountID, TransactionDate, Amount, TransactionType) VALUES (3, 3, TO\_DATE('2024-07-03', 'YYYY-MM-DD'), 150.00, 'Credit')

INTO Transactions (TransactionID, AccountID, TransactionDate, Amount, TransactionType) VALUES (4, 4, TO\_DATE('2024-07-04', 'YYYY-MM-DD'), 250.75, 'Debit')

INTO Transactions (TransactionID, AccountID, TransactionDate, Amount, TransactionType) VALUES (5, 5, TO\_DATE('2024-07-05', 'YYYY-MM-DD'), 300.20, 'Credit')

INTO Transactions (TransactionID, AccountID, TransactionDate, Amount, TransactionType) VALUES (6, 6, TO\_DATE('2024-07-06', 'YYYY-MM-DD'), 200.80, 'Debit')

INTO Transactions (TransactionID, AccountID, TransactionDate, Amount, TransactionType) VALUES (7, 7, TO\_DATE('2024-07-07', 'YYYY-MM-DD'), 250.65, 'Credit')

Select \* from dual;

INSERT ALL

INTO Loans (LoanID, CustomerID, LoanAmount, InterestRate, StartDate, EndDate) VALUES (1, 1, 5000.00, 5.5, TO\_DATE('2024-01-01', 'YYYY-MM-DD'), TO\_DATE('2029-01-01', 'YYYY-MM-DD'))

INTO Loans (LoanID, CustomerID, LoanAmount, InterestRate, StartDate, EndDate) VALUES (2, 2, 10000.00, 4.5, TO\_DATE('2024-02-01', 'YYYY-MM-DD'), TO\_DATE('2029-02-01', 'YYYY-MM-DD'))

INTO Loans (LoanID, CustomerID, LoanAmount, InterestRate, StartDate, EndDate) VALUES (3, 3, 15000.00, 3.5, TO\_DATE('2024-03-01', 'YYYY-MM-DD'), TO\_DATE('2029-03-01', 'YYYY-MM-DD'))

INTO Loans (LoanID, CustomerID, LoanAmount, InterestRate, StartDate, EndDate) VALUES (4, 4, 20000.00, 6.0, TO\_DATE('2024-04-01', 'YYYY-MM-DD'), TO\_DATE('2029-04-01', 'YYYY-MM-DD'))

INTO Loans (LoanID, CustomerID, LoanAmount, InterestRate, StartDate, EndDate) VALUES (5, 5, 25000.00, 4.0, TO\_DATE('2024-05-01', 'YYYY-MM-DD'), TO\_DATE('2029-05-01', 'YYYY-MM-DD'))

INTO Loans (LoanID, CustomerID, LoanAmount, InterestRate, StartDate, EndDate) VALUES (6, 6, 30000.00, 5.0, TO\_DATE('2024-06-01', 'YYYY-MM-DD'), TO\_DATE('2029-06-01', 'YYYY-MM-DD'))

INTO Loans (LoanID, CustomerID, LoanAmount, InterestRate, StartDate, EndDate) VALUES (7, 7, 35000.00, 3.0, TO\_DATE('2024-07-01', 'YYYY-MM-DD'), TO\_DATE('2029-07-01', 'YYYY-MM-DD'))

SELECT \* FROM DUAL;

INSERT ALL

INTO Employees (EmployeeID, Name, Position, Salary, Department, HireDate) VALUES (1, 'Alice Johnson', 'Manager', 80000.00, 'Finance', TO\_DATE('2015-06-01', 'YYYY-MM-DD'))

INTO Employees (EmployeeID, Name, Position, Salary, Department, HireDate) VALUES (2, 'Bob Smith', 'Analyst', 60000.00, 'Marketing', TO\_DATE('2016-07-01', 'YYYY-MM-DD'))

INTO Employees (EmployeeID, Name, Position, Salary, Department, HireDate) VALUES (3, 'Charlie Brown', 'Developer', 70000.00, 'IT', TO\_DATE('2017-08-01', 'YYYY-MM-DD'))

INTO Employees (EmployeeID, Name, Position, Salary, Department, HireDate) VALUES (4, 'Diane Clark', 'HR Specialist', 50000.00, 'HR', TO\_DATE('2018-09-01', 'YYYY-MM-DD'))

INTO Employees (EmployeeID, Name, Position, Salary, Department, HireDate) VALUES (5, 'Eve Lewis', 'Director', 90000.00, 'Operations', TO\_DATE('2019-10-01', 'YYYY-MM-DD'))

INTO Employees (EmployeeID, Name, Position, Salary, Department, HireDate) VALUES (6, 'Frank Moore', 'Consultant', 55000.00, 'Consulting', TO\_DATE('2020-11-01', 'YYYY-MM-DD'))

INTO Employees (EmployeeID, Name, Position, Salary, Department, HireDate) VALUES (7, 'Grace Hall', 'Coordinator', 45000.00, 'Admin', TO\_DATE('2021-12-01', 'YYYY-MM-DD'))

SELECT \* FROM DUAL;

/\*EXERCISE 1\*:CONTROL STRUCTURES\*/

/\*Scenario 1: The bank wants to apply a discount to loan interest rates for customers above 60 years old.

o Question: Write a PL/SQL block that loops through all customers, checks their age, and if they are above 60, apply a 1% discount to their current loan interest rates.

\*/

DECLARE

v\_customer\_age NUMBER;

v\_loan\_interest\_rate NUMBER;

BEGIN

FOR customer\_rec IN (SELECT CustomerID, DOB FROM Customers) LOOP

v\_customer\_age := TRUNC((SYSDATE - customer\_rec.DOB) / 365.25);

IF v\_customer\_age > 60 THEN

FOR loan\_rec IN (SELECT LoanID, InterestRate FROM Loans WHERE CustomerID = customer\_rec.CustomerID) LOOP

v\_loan\_interest\_rate := loan\_rec.InterestRate - 1;

UPDATE Loans

SET InterestRate = v\_loan\_interest\_rate

WHERE LoanID = loan\_rec.LoanID;

END LOOP;

END IF;

END LOOP;

COMMIT;

END;

/

/\*Scenario 2: A customer can be promoted to VIP status based on their balance.

o Question: Write a PL/SQL block that iterates through all customers and sets a flag IsVIP to TRUE for those with a balance over $10,000.

\*/

ALTER TABLE Customers ADD (IsVIP VARCHAR2(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;

/

/\*Scenario 3: The bank wants to send reminders to customers whose loans are due within the next 30 days.

o Question: Write a PL/SQL block that fetches all loans due in the next 30 days and prints a reminder message for each customer.

\*/

DECLARE

v\_due\_date DATE;

v\_customer\_name VARCHAR2(100);

BEGIN

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

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

DBMS\_OUTPUT.PUT\_LINE('Reminder: Dear ' || v\_customer\_name || ', your loan with LoanID ' || loan\_rec.LoanID || ' is due on ' || TO\_CHAR(loan\_rec.EndDate, 'YYYY-MM-DD') || '.');

END LOOP;

END;

/

/\*EXERCISE 2:ERROR HANDLING \*/

CREATE TABLE ErrorLog (

ErrorMessage VARCHAR2(4000),

ErrorDate DATE

);

/\*Scenario 1: Handle exceptions during fund transfers between accounts.

o Question: Write a stored procedure SafeTransferFunds that transfers funds between two accounts. Ensure that if any error occurs (e.g., insufficient funds), an appropriate error message is logged and the transaction is rolled back.

\*/

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;

v\_error\_message VARCHAR2(4000);

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 source account.');

END IF;

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

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;

COMMIT;

EXCEPTION

WHEN OTHERS THEN

v\_error\_message := SQLERRM;

ROLLBACK;

INSERT INTO ErrorLog (ErrorMessage, ErrorDate) VALUES (v\_error\_message, SYSDATE);

RAISE;

END SafeTransferFunds;

/

/\*Scenario 2: Manage errors when updating employee salaries.

o Question: Write a stored procedure UpdateSalary that increases the salary of an employee by a given percentage. If the employee ID does not exist, handle the exception and log an error message.

\*/

CREATE OR REPLACE PROCEDURE UpdateSalary (

p\_employee\_id IN NUMBER,

p\_percentage IN NUMBER

) AS

v\_error\_message VARCHAR2(4000);

BEGIN

UPDATE Employees

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

WHERE EmployeeID = p\_employee\_id;

IF SQL%ROWCOUNT = 0 THEN

RAISE\_APPLICATION\_ERROR(-20002, 'Employee ID does not exist.');

END IF;

COMMIT;

EXCEPTION

WHEN OTHERS THEN

ROLLBACK;

v\_error\_message := SQLERRM;

INSERT INTO ErrorLog (ErrorMessage, ErrorDate) VALUES (v\_error\_message, SYSDATE);

END UpdateSalary;

/

/\*Scenario 3: Ensure data integrity when adding a new customer.

o Question: Write a stored procedure AddNewCustomer that inserts a new customer into the Customers table. If a customer with the same ID already exists, handle the exception by logging an error and preventing the insertion.

\*/

CREATE OR REPLACE PROCEDURE AddNewCustomer (

p\_customer\_id IN NUMBER,

p\_name IN VARCHAR2,

p\_dob IN DATE,

p\_balance IN NUMBER,

p\_last\_modified IN DATE

) AS

v\_error\_message VARCHAR2(4000);

BEGIN

INSERT INTO Customers (CustomerID, Name, DOB, Balance, LastModified)

VALUES (p\_customer\_id, p\_name, p\_dob, p\_balance, p\_last\_modified);

COMMIT;

EXCEPTION

WHEN DUP\_VAL\_ON\_INDEX THEN

v\_error\_message := 'Customer ID already exists.';

INSERT INTO ErrorLog (ErrorMessage, ErrorDate) VALUES (v\_error\_message, SYSDATE);

ROLLBACK;

WHEN OTHERS THEN

v\_error\_message := SQLERRM;

INSERT INTO ErrorLog (ErrorMessage, ErrorDate) VALUES (v\_error\_message, SYSDATE);

ROLLBACK;

END AddNewCustomer;

/

/\*EXERCISE 3:Stored Procedures\*/

/\*Scenario 1: The bank needs to process monthly interest for all savings accounts.

o Question: Write a stored procedure ProcessMonthlyInterest that calculates and updates the balance of all savings accounts by applying an interest rate of 1% to the current balance.

\*/

CREATE OR REPLACE PROCEDURE ProcessMonthlyInterest AS

v\_interest\_rate CONSTANT NUMBER := 0.01;

v\_new\_balance NUMBER;

v\_account\_id Accounts.AccountID%TYPE;

v\_error\_message VARCHAR2(4000);

BEGIN

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

BEGIN

v\_new\_balance := account\_rec.Balance + (account\_rec.Balance \* v\_interest\_rate);

UPDATE Accounts

SET Balance = v\_new\_balance,

LastModified = SYSDATE

WHERE AccountID = account\_rec.AccountID;

COMMIT;

EXCEPTION

WHEN OTHERS THEN

v\_error\_message := SQLERRM;

INSERT INTO ErrorLog (ErrorMessage, ErrorDate) VALUES (v\_error\_message, SYSDATE);

ROLLBACK;

END;

END LOOP;

END ProcessMonthlyInterest;

/

/\*Scenario 2: The bank wants to implement a bonus scheme for employees based on their performance.

o Question: Write a stored procedure UpdateEmployeeBonus that updates the salary of employees in a given department by adding a bonus percentage passed as a parameter.

\*/

CREATE OR REPLACE PROCEDURE UpdateEmployeeBonus (

p\_department IN VARCHAR2,

p\_bonus\_percentage IN NUMBER

) AS

v\_new\_salary NUMBER;

v\_error\_message VARCHAR2(4000);

BEGIN

FOR emp\_rec IN (SELECT EmployeeID, Salary FROM Employees WHERE Department = p\_department) LOOP

BEGIN

v\_new\_salary := emp\_rec.Salary \* (1 + p\_bonus\_percentage / 100);

UPDATE Employees

SET Salary = v\_new\_salary

WHERE EmployeeID = emp\_rec.EmployeeID;

COMMIT;

EXCEPTION

WHEN OTHERS THEN

v\_error\_message := SQLERRM;

INSERT INTO ErrorLog (ErrorMessage, ErrorDate) VALUES (v\_error\_message, SYSDATE);

ROLLBACK;

END;

END LOOP;

END UpdateEmployeeBonus;

/

/\*Scenario 3: Customers should be able to transfer funds between their accounts.

o Question: Write a stored procedure TransferFunds that transfers a specified amount from one account to another, checking that the source account has sufficient balance before making the transfer.

\*/

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;

v\_error\_message VARCHAR2(4000);

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 source 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;

COMMIT;

EXCEPTION

WHEN OTHERS THEN

v\_error\_message := SQLERRM;

ROLLBACK;

INSERT INTO ErrorLog (ErrorMessage, ErrorDate) VALUES (v\_error\_message, SYSDATE);

END TransferFunds;

/

/\*EXERCISE 4:FUNCTIONS\*/

/\*Scenario 1: Calculate the age of customers for eligibility checks.

o Question: Write a function CalculateAge that takes a customer's date of birth as input and returns their age in years.

\*/

CREATE OR REPLACE FUNCTION CalculateAge (

p\_dob DATE

) RETURN NUMBER

IS

v\_age NUMBER;

BEGIN

v\_age := TRUNC((SYSDATE - p\_dob) / 365.25);

RETURN v\_age;

END CalculateAge;

/

/\*Scenario 2: The bank needs to compute the monthly installment for a loan.

o Question: Write a function CalculateMonthlyInstallment that takes the loan amount, interest rate, and loan duration in years as input and returns the monthly installment amount.

\*/

CREATE OR REPLACE FUNCTION CalculateMonthlyInstallment (

p\_loan\_amount NUMBER,

p\_annual\_interest\_rate NUMBER,

p\_loan\_duration\_years NUMBER

) RETURN NUMBER

IS

v\_monthly\_interest\_rate NUMBER;

v\_total\_payments NUMBER;

v\_monthly\_installment NUMBER;

BEGIN

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

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

IF v\_monthly\_interest\_rate > 0 THEN

v\_monthly\_installment := (p\_loan\_amount \* v\_monthly\_interest\_rate \* POWER(1 + v\_monthly\_interest\_rate, v\_total\_payments)) /

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

ELSE

v\_monthly\_installment := p\_loan\_amount / v\_total\_payments;

END IF;

RETURN v\_monthly\_installment;

END CalculateMonthlyInstallment;

/

/\*Scenario 3: Check if a customer has sufficient balance before making a transaction.

o Question: Write a function HasSufficientBalance that takes an account ID and an amount as input and returns a boolean indicating whether the account has at least the specified amount.

\*/

CREATE OR REPLACE FUNCTION HasSufficientBalance (

p\_account\_id IN NUMBER,

p\_amount IN 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;

WHEN OTHERS THEN

RETURN FALSE;

END HasSufficientBalance;

/

/\*EXERCISE 5:TRIGGERS\*/

/\*Scenario 1: Automatically update the last modified date when a customer's record is updated.

o Question: Write a trigger UpdateCustomerLastModified that updates the LastModified column of the Customers table to the current date whenever a customer's record is updated.

\*/

CREATE OR REPLACE TRIGGER UpdateCustomerLastModified

BEFORE UPDATE ON Customers

FOR EACH ROW

BEGIN

:NEW.LastModified := SYSDATE;

END UpdateCustomerLastModified;

/

/\*Scenario 2: Maintain an audit log for all transactions.

o Question: Write a trigger LogTransaction that inserts a record into an AuditLog table whenever a transaction is inserted into the Transactions table.

\*/

CREATE TABLE AuditLog (

AuditID NUMBER PRIMARY KEY,

TransactionID NUMBER,

ActionType VARCHAR2(50),

ActionDate DATE,

UserName VARCHAR2(100)

);

CREATE SEQUENCE AuditLogSeq

START WITH 1

INCREMENT BY 1

NOCACHE

NOCYCLE;

CREATE OR REPLACE TRIGGER AuditLogTrigger

BEFORE INSERT ON AuditLog

FOR EACH ROW

BEGIN

IF :NEW.AuditID IS NULL THEN

SELECT AuditLogSeq.NEXTVAL INTO :NEW.AuditID FROM DUAL;

END IF;

END;

/\*Scenario 3: Enforce business rules on deposits and withdrawals.

o Question: Write a trigger CheckTransactionRules that ensures withdrawals do not exceed the balance and deposits are positive before inserting a record into the Transactions table.

\*/

CREATE OR REPLACE TRIGGER CheckTransactionRules

BEFORE INSERT ON Transactions

FOR EACH ROW

DECLARE

v\_current\_balance NUMBER;

BEGIN

SELECT Balance INTO v\_current\_balance

FROM Accounts

WHERE AccountID = :NEW.AccountID;

IF :NEW.TransactionType = 'Debit' THEN

IF :NEW.Amount > v\_current\_balance THEN

RAISE\_APPLICATION\_ERROR(-20001, 'Withdrawal amount exceeds the account balance.');

END IF;

ELSIF :NEW.TransactionType = 'Credit' THEN

IF :NEW.Amount <= 0 THEN

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

END IF;

END IF;

EXCEPTION

WHEN NO\_DATA\_FOUND THEN

RAISE\_APPLICATION\_ERROR(-20003, 'Account ID does not exist.');

WHEN OTHERS THEN

RAISE\_APPLICATION\_ERROR(-20004, 'An unexpected error occurred: ' || SQLERRM);

END CheckTransactionRules;

/

/\*EXERCISE 6:CURSORS\*/

/\*Scenario 1: Generate monthly statements for all customers.

o Question: Write a PL/SQL block using an explicit cursor GenerateMonthlyStatements that retrieves all transactions for the current month and prints a statement for each customer.

\*/

SET SERVEROUTPUT ON;

DECLARE

CURSOR transaction\_cursor IS

SELECT

t.TransactionID,

t.AccountID,

t.Amount,

t.TransactionType,

t.TransactionDate,

a.CustomerID,

c.Name AS CustomerName

FROM

Transactions t

JOIN

Accounts a ON t.AccountID = a.AccountID

JOIN

Customers c ON a.CustomerID = c.CustomerID

WHERE

t.TransactionDate >= TRUNC(SYSDATE, 'MM') AND

t.TransactionDate < ADD\_MONTHS(TRUNC(SYSDATE, 'MM'), 1);

TYPE transaction\_record\_type IS RECORD (

TransactionID Transactions.TransactionID%TYPE,

AccountID Transactions.AccountID%TYPE,

Amount Transactions.Amount%TYPE,

TransactionType Transactions.TransactionType%TYPE,

TransactionDate Transactions.TransactionDate%TYPE,

CustomerID Accounts.CustomerID%TYPE,

CustomerName Customers.Name%TYPE

);

transaction\_record transaction\_record\_type;

v\_current\_customer\_id Accounts.CustomerID%TYPE := NULL;

v\_current\_customer\_name Customers.Name%TYPE;

BEGIN

OPEN transaction\_cursor;

LOOP

FETCH transaction\_cursor INTO transaction\_record;

EXIT WHEN transaction\_cursor%NOTFOUND;

IF v\_current\_customer\_id IS NULL OR v\_current\_customer\_id != transaction\_record.CustomerID THEN

IF v\_current\_customer\_id IS NOT NULL THEN

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

END IF;

v\_current\_customer\_id := transaction\_record.CustomerID;

v\_current\_customer\_name := transaction\_record.CustomerName;

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

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

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

DBMS\_OUTPUT.PUT\_LINE('Transaction ID | Account ID | Amount | Type | Date');

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

END IF;

DBMS\_OUTPUT.PUT\_LINE(transaction\_record.TransactionID || ' | ' ||

transaction\_record.AccountID || ' | ' ||

transaction\_record.Amount || ' | ' ||

transaction\_record.TransactionType || ' | ' ||

TO\_CHAR(transaction\_record.TransactionDate, 'DD-MON-YYYY'));

END LOOP;

CLOSE transaction\_cursor;

IF v\_current\_customer\_id IS NOT NULL THEN

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

END IF;

EXCEPTION

WHEN OTHERS THEN

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

END;

/

/\*Scenario 2: Apply annual fee to all accounts.

o Question: Write a PL/SQL block using an explicit cursor ApplyAnnualFee that deducts an annual maintenance fee from the balance of all accounts.

\*/

DECLARE

CURSOR account\_cursor IS

SELECT AccountID, Balance

FROM Accounts

FOR UPDATE OF Balance;

v\_annual\_fee NUMBER := 50;

v\_account\_id Accounts.AccountID%TYPE;

v\_balance Accounts.Balance%TYPE;

BEGIN

OPEN account\_cursor;

LOOP

FETCH account\_cursor INTO v\_account\_id, v\_balance;

EXIT WHEN account\_cursor%NOTFOUND;

IF v\_balance >= v\_annual\_fee THEN

UPDATE Accounts

SET Balance = Balance - v\_annual\_fee

WHERE CURRENT OF account\_cursor;

DBMS\_OUTPUT.PUT\_LINE('Account ID: ' || v\_account\_id || ' - Annual fee applied. New balance: ' || (v\_balance - v\_annual\_fee));

ELSE

DBMS\_OUTPUT.PUT\_LINE('Account ID: ' || v\_account\_id || ' - Insufficient funds to apply annual fee.');

END IF;

END LOOP;

CLOSE account\_cursor;

EXCEPTION

WHEN OTHERS THEN

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

ROLLBACK;

END;

/

/\*Scenario 3: Update the interest rate for all loans based on a new policy.

o Question: Write a PL/SQL block using an explicit cursor UpdateLoanInterestRates that fetches all loans and updates their interest rates based on the new policy.

\*/

DECLARE

CURSOR loan\_cursor IS

SELECT LoanID, InterestRate

FROM Loans

FOR UPDATE OF InterestRate;

TYPE loan\_record\_type IS RECORD (

LoanID Loans.LoanID%TYPE,

InterestRate Loans.InterestRate%TYPE

);

loan\_record loan\_record\_type;

v\_interest\_rate\_adjustment NUMBER := 0.5;

BEGIN

OPEN loan\_cursor;

LOOP

FETCH loan\_cursor INTO loan\_record;

EXIT WHEN loan\_cursor%NOTFOUND;

loan\_record.InterestRate := loan\_record.InterestRate + v\_interest\_rate\_adjustment;

UPDATE Loans

SET InterestRate = loan\_record.InterestRate

WHERE CURRENT OF loan\_cursor;

DBMS\_OUTPUT.PUT\_LINE('Updated LoanID ' || loan\_record.LoanID || ' to new InterestRate ' || loan\_record.InterestRate);

END LOOP;

CLOSE loan\_cursor;

COMMIT;

EXCEPTION

WHEN OTHERS THEN

ROLLBACK;

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

END;

/

/\*EXERCISE 7:PACKAGES\*/

/\*

Scenario 1: Group all customer-related procedures and functions into a package.

o Question: Create a package CustomerManagement with procedures for adding a new customer, updating customer details, and a function to get customer balance.

\*/

CREATE OR REPLACE PACKAGE CustomerManagement AS

PROCEDURE AddCustomer(

p\_customerid IN NUMBER,

p\_name IN VARCHAR2,

p\_dob IN DATE,

p\_balance IN NUMBER,

p\_lastmodified IN DATE,

p\_isvip IN CHAR

);

PROCEDURE UpdateCustomer(

p\_customerid IN NUMBER,

p\_name IN VARCHAR2,

p\_dob IN DATE,

p\_balance IN NUMBER,

p\_lastmodified IN DATE,

p\_isvip IN CHAR

);

FUNCTION GetCustomerBalance(p\_customerid IN NUMBER) RETURN NUMBER;

END CustomerManagement;

/

SELECT COLUMN\_NAME

FROM ALL\_TAB\_COLUMNS

WHERE TABLE\_NAME = 'CUSTOMERS';

CREATE OR REPLACE PACKAGE BODY CustomerManagement AS

PROCEDURE AddCustomer(

p\_customerid IN NUMBER,

p\_name IN VARCHAR2,

p\_dob IN DATE,

p\_balance IN NUMBER,

p\_lastmodified IN DATE,

p\_isvip IN CHAR

) IS

BEGIN

INSERT INTO Customers (

CUSTOMERID,

NAME,

DOB,

BALANCE,

LASTMODIFIED,

ISVIP

) VALUES (

p\_customerid,

p\_name,

p\_dob,

p\_balance,

p\_lastmodified,

p\_isvip

);

EXCEPTION

WHEN OTHERS THEN

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

END;

PROCEDURE UpdateCustomer(

p\_customerid IN NUMBER,

p\_name IN VARCHAR2,

p\_dob IN DATE,

p\_balance IN NUMBER,

p\_lastmodified IN DATE,

p\_isvip IN CHAR

) IS

BEGIN

UPDATE Customers

SET

NAME = p\_name,

DOB = p\_dob,

BALANCE = p\_balance,

LASTMODIFIED = p\_lastmodified,

ISVIP = p\_isvip

WHERE CUSTOMERID = p\_customerid;

EXCEPTION

WHEN OTHERS THEN

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

END;

FUNCTION GetCustomerBalance(p\_customerid IN NUMBER) RETURN NUMBER IS

v\_balance NUMBER;

BEGIN

SELECT BALANCE INTO v\_balance

FROM Customers

WHERE CUSTOMERID = p\_customerid;

RETURN v\_balance;

EXCEPTION

WHEN NO\_DATA\_FOUND THEN

RETURN NULL;

WHEN OTHERS THEN

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

RETURN NULL;

END;

END CustomerManagement;

/

/\*Scenario 2: Create a package to manage employee data.

o Question: Write a package EmployeeManagement with procedures to hire new employees, update employee details, and a function to calculate annual salary.

\*/

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, p\_hire\_date IN DATE);

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, 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);

EXCEPTION

WHEN OTHERS THEN

DBMS\_OUTPUT.PUT\_LINE('An error occurred while hiring employee: ' || SQLERRM);

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;

EXCEPTION

WHEN OTHERS THEN

DBMS\_OUTPUT.PUT\_LINE('An error occurred while updating employee: ' || SQLERRM);

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; -- Handle case where the employee is not found

WHEN OTHERS THEN

DBMS\_OUTPUT.PUT\_LINE('An error occurred while calculating annual salary: ' || SQLERRM);

RETURN 0;

END CalculateAnnualSalary;

END EmployeeManagement;

/

/\*Scenario 3: Group all account-related operations into a package.

o Question: Create a package AccountOperations with procedures for opening a new account, closing an account, and a function to get the total balance of a customer across all accounts.

\*/

-- Creating the package specification

CREATE OR REPLACE PACKAGE AccountOperations AS

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

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\_customer\_id IN NUMBER, p\_initial\_balance IN NUMBER, p\_account\_type IN VARCHAR2) IS

BEGIN

INSERT INTO Accounts (CustomerID, Balance, AccountType, DateOpened)

VALUES (p\_customer\_id, p\_initial\_balance, p\_account\_type, SYSDATE);

COMMIT;

EXCEPTION

WHEN OTHERS THEN

ROLLBACK;

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

END OpenAccount;

PROCEDURE CloseAccount(p\_account\_id IN NUMBER) IS

BEGIN

DELETE FROM Accounts

WHERE AccountID = p\_account\_id;

COMMIT;

EXCEPTION

WHEN OTHERS THEN

ROLLBACK;

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

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;

EXCEPTION

WHEN NO\_DATA\_FOUND THEN

RETURN 0;

WHEN OTHERS THEN

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

RETURN NULL;

END GetTotalBalance;

END AccountOperations;

/