**Exercise 1: Control Structures**

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

* + **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.

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

* + **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.

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

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

-- Exercise 1

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

);

-- Scripts for Sample Data Insertion

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

VALUES (1, 'John Doe', TO\_DATE('1963-05-15', 'YYYY-MM-DD'), 1000, SYSDATE);

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

VALUES (2, 'Jane Smith', TO\_DATE('1990-07-20', 'YYYY-MM-DD'), 1500, SYSDATE);

INSERT INTO Accounts (AccountID, CustomerID, AccountType, Balance, LastModified)

VALUES (1, 1, 'Savings', 1000, SYSDATE);

INSERT INTO Accounts (AccountID, CustomerID, AccountType, Balance, LastModified)

VALUES (2, 2, 'Checking', 1500, SYSDATE);

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

VALUES (1, 1, SYSDATE, 200, 'Deposit');

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

VALUES (2, 2, SYSDATE, 300, 'Withdrawal');

INSERT INTO Loans (LoanID, CustomerID, LoanAmount, InterestRate, StartDate, EndDate)

VALUES (1, 1, 5000, 5, SYSDATE, ADD\_MONTHS(SYSDATE, 60));

INSERT INTO Loans (LoanID, CustomerID, LoanAmount, InterestRate, StartDate, EndDate)

VALUES (2, 2, 10000, 5, SYSDATE, SYSDATE+25);

INSERT INTO Employees (EmployeeID, Name, Position, Salary, Department, HireDate)

VALUES (1, 'Alice Johnson', 'Manager', 70000, 'HR', TO\_DATE('2015-06-15', 'YYYY-MM-DD'));

INSERT INTO Employees (EmployeeID, Name, Position, Salary, Department, HireDate)

VALUES (2, 'Bob Brown', 'Developer', 60000, 'IT', TO\_DATE('2017-03-20', 'YYYY-MM-DD'));

-- Enable server output

SET SERVEROUTPUT ON;

-- Adding IsVIP column to Customers table

ALTER TABLE Customers ADD IsVIP CHAR(1);

-- SCENARIO - 1

-- Procedure to Apply Interest Discount to Customers Above 60 Years Old

CREATE OR REPLACE PROCEDURE ApplyInterestDiscount IS

CURSOR customer\_cursor IS

SELECT CustomerID, DOB FROM Customers;

customer\_id Customers.CustomerID%TYPE;

customer\_dob Customers.DOB%TYPE;

today\_date DATE := SYSDATE;

BEGIN

FOR customer\_rec IN customer\_cursor LOOP

IF MONTHS\_BETWEEN(today\_date, customer\_rec.DOB) / 12 > 60 THEN

UPDATE Loans

SET InterestRate = InterestRate \* 0.99

WHERE CustomerID = customer\_rec.CustomerID;

DBMS\_OUTPUT.PUT\_LINE('CustomerID: ' || customer\_rec.CustomerID || ' interest rate has been decreased by 1%.');

END IF;

END LOOP;

END ApplyInterestDiscount;

/

-- SCENARIO - 2

-- Procedure to Promote Customers to VIP Status Based on Balance

CREATE OR REPLACE PROCEDURE PromoteToVIP IS

CURSOR account\_cursor IS

SELECT CustomerID, Balance FROM Accounts;

acc\_customer\_id Accounts.CustomerID%TYPE;

acc\_balance Accounts.Balance%TYPE;

BEGIN

FOR account\_rec IN account\_cursor LOOP

IF account\_rec.Balance > 5000 THEN

UPDATE Customers

SET IsVIP = 'Y'

WHERE CustomerID = account\_rec.CustomerID;

DBMS\_OUTPUT.PUT\_LINE('CustomerID: ' || account\_rec.CustomerID || ' promoted to VIP!');

ELSE

UPDATE Customers

SET IsVIP = 'N'

WHERE CustomerID = account\_rec.CustomerID;

DBMS\_OUTPUT.PUT\_LINE('CustomerID: ' || account\_rec.CustomerID || ' demoted from VIP!');

END IF;

END LOOP;

END PromoteToVIP;

/

-- SCENARIO - 3

-- Procedure to Send Loan Reminders for Loans Due Within the Next 30 Days

CREATE OR REPLACE PROCEDURE SendLoanReminders IS

CURSOR loan\_cursor IS

SELECT CustomerID, EndDate

FROM Loans

WHERE EndDate BETWEEN SYSDATE AND SYSDATE + 30;

loan\_customer\_id Loans.CustomerID%TYPE;

loan\_end\_date Loans.EndDate%TYPE;

BEGIN

FOR loan\_rec IN loan\_cursor LOOP

DBMS\_OUTPUT.PUT\_LINE('Reminder: Loan due on ' || TO\_CHAR(loan\_rec.EndDate, 'YYYY-MM-DD') || ' for CustomerID: ' || loan\_rec.CustomerID);

END LOOP;

END SendLoanReminders;

/

-- Call the procedures

BEGIN

DBMS\_OUTPUT.PUT\_LINE('SCENARIO - 1: Apply discount to interest rate for customers above age 60');

ApplyInterestDiscount; -- Apply discount to interest rate for the customers above age 60

END;

/

BEGIN

DBMS\_OUTPUT.PUT\_LINE('SCENARIO - 2: Promote customers to VIP status based on balance');

PromoteToVIP; -- Promote customers to VIP status based on balance

END;

/

BEGIN

DBMS\_OUTPUT.PUT\_LINE('SCENARIO - 3: Reminders for loans due within the next 30 days');

SendLoanReminders; -- Send Loan Reminders for loans due within the next 30 days

END;

/

**Exercise 2: Error Handling**

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

* + **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.

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

* + **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.

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

* + **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.

-- Enable server output

SET SERVEROUTPUT ON;

-- Create tables

CREATE TABLE Customers (

CustomerID INT PRIMARY KEY,

Name VARCHAR2(100),

DOB DATE,

Balance INT,

LastModified DATE,

IsVIP CHAR(1)

);

CREATE TABLE Accounts (

AccountID INT PRIMARY KEY,

CustomerID INT,

AccountType VARCHAR2(20),

Balance INT,

LastModified DATE,

FOREIGN KEY (CustomerID) REFERENCES Customers(CustomerID)

);

CREATE TABLE Transactions (

TransactionID INT PRIMARY KEY,

AccountID INT,

TransactionDate DATE,

Amount INT,

TransactionType VARCHAR2(10),

FOREIGN KEY (AccountID) REFERENCES Accounts(AccountID)

);

CREATE TABLE Loans (

LoanID INT PRIMARY KEY,

CustomerID INT,

LoanAmount INT,

InterestRate INT,

StartDate DATE,

EndDate DATE,

FOREIGN KEY (CustomerID) REFERENCES Customers(CustomerID)

);

CREATE TABLE Employees (

EmployeeID INT PRIMARY KEY,

Name VARCHAR2(100),

Position VARCHAR2(50),

Salary INT,

Department VARCHAR2(50),

HireDate DATE

);

-- Create ErrorLogs table for logging errors

CREATE TABLE ErrorLogs (

ErrorID INT PRIMARY KEY,

ErrorMessage VARCHAR2(255),

ErrorDate DATE

);

-- Insert data

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

VALUES (1, 'John Doe', TO\_DATE('1985-05-15', 'YYYY-MM-DD'), 1000, SYSDATE);

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

VALUES (2, 'Jane Smith', TO\_DATE('1990-07-20', 'YYYY-MM-DD'), 1500, SYSDATE);

INSERT INTO Accounts (AccountID, CustomerID, AccountType, Balance, LastModified)

VALUES (1, 1, 'Savings', 1000, SYSDATE);

INSERT INTO Accounts (AccountID, CustomerID, AccountType, Balance, LastModified)

VALUES (2, 2, 'Checking', 1500, SYSDATE);

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

VALUES (1, 1, SYSDATE, 200, 'Deposit');

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

VALUES (2, 2, SYSDATE, 300, 'Withdrawal');

INSERT INTO Loans (LoanID, CustomerID, LoanAmount, InterestRate, StartDate, EndDate)

VALUES (1, 1, 5000, 5, SYSDATE, ADD\_MONTHS(SYSDATE, 60));

INSERT INTO Employees (EmployeeID, Name, Position, Salary, Department, HireDate)

VALUES (1, 'Alice Johnson', 'Manager', 70000, 'HR', TO\_DATE('2015-06-15', 'YYYY-MM-DD'));

INSERT INTO Employees (EmployeeID, Name, Position, Salary, Department, HireDate)

VALUES (2, 'Bob Brown', 'Developer', 60000, 'IT', TO\_DATE('2017-03-20', 'YYYY-MM-DD'));

-- Procedure to Apply Interest Discount to Customers Above 60 Years Old

CREATE OR REPLACE PROCEDURE ApplyInterestDiscount IS

CURSOR customer\_cursor IS

SELECT CustomerID, DOB FROM Customers;

customer\_id Customers.CustomerID%TYPE;

customer\_dob Customers.DOB%TYPE;

today\_date DATE := SYSDATE;

BEGIN

FOR customer\_rec IN customer\_cursor LOOP

IF MONTHS\_BETWEEN(today\_date, customer\_rec.DOB) / 12 > 60 THEN

UPDATE Loans

SET InterestRate = InterestRate \* 0.99

WHERE CustomerID = customer\_rec.CustomerID;

END IF;

END LOOP;

END ApplyInterestDiscount;

/

-- Procedure to Promote Customers to VIP Status Based on Balance

CREATE OR REPLACE PROCEDURE PromoteToVIP IS

CURSOR account\_cursor IS

SELECT CustomerID, Balance FROM Accounts;

acc\_customer\_id Accounts.CustomerID%TYPE;

acc\_balance Accounts.Balance%TYPE;

BEGIN

FOR account\_rec IN account\_cursor LOOP

IF account\_rec.Balance > 10000 THEN

UPDATE Customers

SET IsVIP = 'Y'

WHERE CustomerID = account\_rec.CustomerID;

ELSE

UPDATE Customers

SET IsVIP = 'N'

WHERE CustomerID = account\_rec.CustomerID;

END IF;

END LOOP;

END PromoteToVIP;

/

-- Procedure to Send Loan Reminders for Loans Due Within the Next 30 Days

CREATE OR REPLACE PROCEDURE SendLoanReminders IS

CURSOR loan\_cursor IS

SELECT CustomerID, EndDate

FROM Loans

WHERE EndDate BETWEEN SYSDATE AND SYSDATE + 30;

loan\_customer\_id Loans.CustomerID%TYPE;

loan\_end\_date Loans.EndDate%TYPE;

BEGIN

FOR loan\_rec IN loan\_cursor LOOP

DBMS\_OUTPUT.PUT\_LINE('Reminder: Loan due on ' || TO\_CHAR(loan\_rec.EndDate, 'YYYY-MM-DD') || ' for CustomerID: ' || loan\_rec.CustomerID);

END LOOP;

END SendLoanReminders;

/

-- Procedure to Handle Exceptions During Fund Transfers Between Accounts

CREATE OR REPLACE PROCEDURE SafeTransferFunds(

srcAccountID IN INT,

destAccountID IN INT,

transferAmount IN DECIMAL

) IS

BEGIN

DECLARE

insufficientFunds EXCEPTION;

PRAGMA EXCEPTION\_INIT(insufficientFunds, -20001);

BEGIN

-- Start transaction

SAVEPOINT start\_trans;

-- Check if the source account has enough balance

DECLARE

src\_balance INT;

BEGIN

SELECT Balance INTO src\_balance

FROM Accounts

WHERE AccountID = srcAccountID;

IF src\_balance < transferAmount THEN

DBMS\_OUTPUT.PUT\_LINE('Insufficient Funds');

RAISE insufficientFunds;

END IF;

END;

-- Deduct amount from source account

UPDATE Accounts

SET Balance = Balance - transferAmount

WHERE AccountID = srcAccountID;

-- Add amount to destination account

UPDATE Accounts

SET Balance = Balance + transferAmount

WHERE AccountID = destAccountID;

DBMS\_OUTPUT.PUT\_LINE('Funds transferred safely...');

-- Commit transaction

COMMIT;

EXCEPTION

WHEN insufficientFunds THEN

INSERT INTO ErrorLogs (ErrorMessage, ErrorDate)

VALUES ('Insufficient funds for transfer from AccountID: ' || srcAccountID, SYSDATE);

ROLLBACK TO start\_trans;

WHEN OTHERS THEN

INSERT INTO ErrorLogs (ErrorMessage, ErrorDate)

VALUES ('SQL Error during transfer from AccountID: ' || srcAccountID || ' to AccountID: ' || destAccountID, SYSDATE);

ROLLBACK TO start\_trans;

END;

END SafeTransferFunds;

/

-- Procedure to Manage Errors When Updating Employee Salaries

CREATE OR REPLACE PROCEDURE UpdateSalary(

empID IN INT,

salaryIncreasePercentage IN DECIMAL

) IS

BEGIN

DECLARE

empNotFound EXCEPTION;

PRAGMA EXCEPTION\_INIT(empNotFound, -20001);

BEGIN

-- Start transaction

SAVEPOINT start\_trans;

-- Update salary

UPDATE Employees

SET Salary = Salary + (Salary \* (salaryIncreasePercentage / 100))

WHERE EmployeeID = empID;

DBMS\_OUTPUT.PUT\_LINE('Employee salary increased by ' || salaryIncreasePercentage || ' percentage');

-- Check if the update affected any row

IF SQL%ROWCOUNT = 0 THEN

RAISE empNotFound;

END IF;

-- Commit transaction

COMMIT;

EXCEPTION

WHEN empNotFound THEN

INSERT INTO ErrorLogs (ErrorMessage, ErrorDate)

VALUES ('Employee ID does not exist: ' || empID, SYSDATE);

DBMS\_OUTPUT.PUT\_LINE('Employee ID does not exist!!');

ROLLBACK TO start\_trans;

WHEN OTHERS THEN

INSERT INTO ErrorLogs (ErrorMessage, ErrorDate)

VALUES ('Error updating salary for EmployeeID: ' || empID, SYSDATE);

ROLLBACK TO start\_trans;

END;

END UpdateSalary;

/

-- Procedure to Ensure Data Integrity When Adding a New Customer

CREATE OR REPLACE PROCEDURE AddNewCustomer(

newCusID IN NUMBER,

newCusName IN VARCHAR2,

newCusDOB IN DATE,

newCusBalance IN NUMBER,

newCusLastModified IN DATE

)

IS

IDAlreadyExists EXCEPTION;

existingCustomerCount NUMBER;

BEGIN

SELECT count(\*) INTO existingCustomerCount FROM Customers WHERE CustomerID = newCusID;

IF existingCustomerCount > 0 THEN

RAISE IDAlreadyExists;

END IF;

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

VALUES (newCusID, newCusName, newCusDOB, newCusBalance, newCusLastModified);

DBMS\_OUTPUT.PUT\_LINE('Customer registered successfully');

EXCEPTION

WHEN IDAlreadyExists THEN

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

END;

/

-- Call the procedures

BEGIN

ApplyInterestDiscount;

END;

/

BEGIN

PromoteToVIP;

END;

/

BEGIN

SendLoanReminders;

END;

/

-- Example calls for SafeTransferFunds, UpdateSalary, and AddNewCustomer

BEGIN

SafeTransferFunds(srcAccountID => 1, destAccountID => 2, transferAmount => 100.00);

END;

/

BEGIN

UpdateSalary(empID => 1, salaryIncreasePercentage => 10.00);

END;

/

BEGIN

AddNewCustomer(newCusID => 3, newCusName => 'Oshith', newCusDOB => SYSDATE, newCusBalance => 5000, newCusLastModified => SYSDATE);

END;

/

**Exercise 3: Stored Procedures**

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

* + **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.

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

* + **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.

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

* + **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.

-- Enable server output

SET SERVEROUTPUT ON;

-- Create tables

CREATE TABLE Customers (

CustomerID INT PRIMARY KEY,

Name VARCHAR2(100),

DOB DATE,

Balance INT,

LastModified DATE,

IsVIP CHAR(1)

);

CREATE TABLE Accounts (

AccountID INT PRIMARY KEY,

CustomerID INT,

AccountType VARCHAR2(20),

Balance INT,

LastModified DATE,

FOREIGN KEY (CustomerID) REFERENCES Customers(CustomerID)

);

CREATE TABLE Transactions (

TransactionID INT PRIMARY KEY,

AccountID INT,

TransactionDate DATE,

Amount INT,

TransactionType VARCHAR2(10),

FOREIGN KEY (AccountID) REFERENCES Accounts(AccountID)

);

CREATE TABLE Loans (

LoanID INT PRIMARY KEY,

CustomerID INT,

LoanAmount INT,

InterestRate INT,

StartDate DATE,

EndDate DATE,

FOREIGN KEY (CustomerID) REFERENCES Customers(CustomerID)

);

CREATE TABLE Employees (

EmployeeID INT PRIMARY KEY,

Name VARCHAR2(100),

Position VARCHAR2(50),

Salary INT,

Department VARCHAR2(50),

DepartmentID INT,

HireDate DATE

);

-- Create ErrorLogs table for logging errors

CREATE TABLE ErrorLogs (

ErrorID INT PRIMARY KEY,

ErrorMessage VARCHAR2(255),

ErrorDate DATE

);

-- Scripts for Sample Data Insertion

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

VALUES (1, 'John Doe', TO\_DATE('1963-05-15', 'YYYY-MM-DD'), 1000, SYSDATE);

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

VALUES (2, 'Jane Smith', TO\_DATE('1990-07-20', 'YYYY-MM-DD'), 1500, SYSDATE);

INSERT INTO Accounts (AccountID, CustomerID, AccountType, Balance, LastModified)

VALUES (1, 1, 'Savings', 1000, SYSDATE);

INSERT INTO Accounts (AccountID, CustomerID, AccountType, Balance, LastModified)

VALUES (2, 2, 'Checking', 1500, SYSDATE);

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

VALUES (1, 1, SYSDATE, 200, 'Deposit');

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

VALUES (2, 2, SYSDATE, 300, 'Withdrawal');

INSERT INTO Loans (LoanID, CustomerID, LoanAmount, InterestRate, StartDate, EndDate)

VALUES (1, 1, 5000, 5, SYSDATE, ADD\_MONTHS(SYSDATE, 60));

INSERT INTO Loans (LoanID, CustomerID, LoanAmount, InterestRate, StartDate, EndDate)

VALUES (2, 2, 10000, 5, SYSDATE, SYSDATE+25);

INSERT INTO Employees (EmployeeID, Name, Position, Salary, Department, HireDate)

VALUES (1, 'Alice Johnson', 'Manager', 70000, 'HR', TO\_DATE('2015-06-15', 'YYYY-MM-DD'));

INSERT INTO Employees (EmployeeID, Name, Position, Salary, Department, HireDate)

VALUES (2, 'Bob Brown', 'Developer', 60000, 'IT', TO\_DATE('2017-03-20', 'YYYY-MM-DD'));

-- SCENARIO - 1

-- Procedure to Process Monthly Interest for All Savings Accounts

CREATE OR REPLACE PROCEDURE ProcessMonthlyInterest IS

-- Variable to store the number of rows updated

v\_rows\_updated NUMBER;

BEGIN

-- Update balance for all savings accounts by applying an interest rate of 1%

UPDATE Accounts

SET Balance = Balance \* 1.01

WHERE AccountType = 'Savings';

-- Get the number of rows updated

v\_rows\_updated := SQL%ROWCOUNT;

-- Display the number of rows updated

DBMS\_OUTPUT.PUT\_LINE('Number of accounts updated: ' || v\_rows\_updated);

END ProcessMonthlyInterest;

/

-- SCENARIO - 2

-- Procedure to Update Employee Bonus Based on Performance

CREATE OR REPLACE PROCEDURE UpdateEmployeeBonus(

deptID IN INT,

bonusPct IN DECIMAL

) IS

BEGIN

-- Update the salary of employees in the specified department by adding the bonus percentage

UPDATE Employees

SET Salary = Salary + (Salary \* (bonusPct / 100))

WHERE DepartmentID = deptID;

END UpdateEmployeeBonus;

/

-- SCENARIO - 3

-- Procedure to Transfer Funds Between Accounts

CREATE OR REPLACE PROCEDURE TransferFunds(

srcAccountID IN INT,

destAccountID IN INT,

transferAmount IN DECIMAL

) IS

BEGIN

DECLARE

insufficientFunds EXCEPTION;

PRAGMA EXCEPTION\_INIT(insufficientFunds, -20001);

BEGIN

-- Start transaction

SAVEPOINT start\_trans;

-- Check if the source account has enough balance

DECLARE

src\_balance INT;

BEGIN

SELECT Balance INTO src\_balance

FROM Accounts

WHERE AccountID = srcAccountID;

IF src\_balance < transferAmount THEN

RAISE insufficientFunds;

END IF;

END;

-- Deduct amount from the source account

UPDATE Accounts

SET Balance = Balance - transferAmount

WHERE AccountID = srcAccountID;

-- Add amount to the destination account

UPDATE Accounts

SET Balance = Balance + transferAmount

WHERE AccountID = destAccountID;

-- Commit transaction

COMMIT;

EXCEPTION

WHEN insufficientFunds THEN

INSERT INTO ErrorLogs (ErrorMessage, ErrorDate)

VALUES ('Insufficient funds for transfer from AccountID: ' || srcAccountID, SYSDATE);

ROLLBACK TO start\_trans;

WHEN OTHERS THEN

INSERT INTO ErrorLogs (ErrorMessage, ErrorDate)

VALUES ('SQL Error during transfer from AccountID: ' || srcAccountID || ' to AccountID: ' || destAccountID, SYSDATE);

ROLLBACK TO start\_trans;

END;

END TransferFunds;

/

-- Example calls for the procedures

BEGIN

ProcessMonthlyInterest;

END;

/

BEGIN

UpdateEmployeeBonus(deptID => 1, bonusPct => 10.00);

END;

/

BEGIN

TransferFunds(srcAccountID => 1, destAccountID => 2, transferAmount => 100.00);

END;

/

**Exercise 4: Functions**

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

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

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

* + **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.

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

* + **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.

-- Enable server output

SET SERVEROUTPUT ON;

-- Create tables

CREATE TABLE Customers (

CustomerID INT PRIMARY KEY,

Name VARCHAR2(100),

DOB DATE,

Balance INT,

LastModified DATE,

IsVIP CHAR(1)

);

CREATE TABLE Accounts (

AccountID INT PRIMARY KEY,

CustomerID INT,

AccountType VARCHAR2(20),

Balance INT,

LastModified DATE,

FOREIGN KEY (CustomerID) REFERENCES Customers(CustomerID)

);

CREATE TABLE Transactions (

TransactionID INT PRIMARY KEY,

AccountID INT,

TransactionDate DATE,

Amount INT,

TransactionType VARCHAR2(10),

FOREIGN KEY (AccountID) REFERENCES Accounts(AccountID)

);

CREATE TABLE Loans (

LoanID INT PRIMARY KEY,

CustomerID INT,

LoanAmount INT,

InterestRate INT,

StartDate DATE,

EndDate DATE,

FOREIGN KEY (CustomerID) REFERENCES Customers(CustomerID)

);

CREATE TABLE Employees (

EmployeeID INT PRIMARY KEY,

Name VARCHAR2(100),

Position VARCHAR2(50),

Salary INT,

Department VARCHAR2(50),

DepartmentID INT,

HireDate DATE

);

-- Create ErrorLogs table for logging errors

CREATE TABLE ErrorLogs (

ErrorID INT PRIMARY KEY,

ErrorMessage VARCHAR2(255),

ErrorDate DATE

);

-- Scripts for Sample Data Insertion

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

VALUES (1, 'John Doe', TO\_DATE('1963-05-15', 'YYYY-MM-DD'), 1000, SYSDATE);

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

VALUES (2, 'Jane Smith', TO\_DATE('1990-07-20', 'YYYY-MM-DD'), 1500, SYSDATE);

INSERT INTO Accounts (AccountID, CustomerID, AccountType, Balance, LastModified)

VALUES (1, 1, 'Savings', 1000, SYSDATE);

INSERT INTO Accounts (AccountID, CustomerID, AccountType, Balance, LastModified)

VALUES (2, 2, 'Checking', 1500, SYSDATE);

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

VALUES (1, 1, SYSDATE, 200, 'Deposit');

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

VALUES (2, 2, SYSDATE, 300, 'Withdrawal');

INSERT INTO Loans (LoanID, CustomerID, LoanAmount, InterestRate, StartDate, EndDate)

VALUES (1, 1, 5000, 5, SYSDATE, ADD\_MONTHS(SYSDATE, 60));

INSERT INTO Loans (LoanID, CustomerID, LoanAmount, InterestRate, StartDate, EndDate)

VALUES (2, 2, 10000, 5, SYSDATE, SYSDATE+25);

INSERT INTO Employees (EmployeeID, Name, Position, Salary, Department, HireDate)

VALUES (1, 'Alice Johnson', 'Manager', 70000, 'HR', TO\_DATE('2015-06-15', 'YYYY-MM-DD'));

INSERT INTO Employees (EmployeeID, Name, Position, Salary, Department, HireDate)

VALUES (2, 'Bob Brown', 'Developer', 60000, 'IT', TO\_DATE('2017-03-20', 'YYYY-MM-DD'));

-- Function to Calculate Age of Customers

CREATE OR REPLACE FUNCTION CalculateAge(dob DATE)

RETURN NUMBER

IS

customerAge NUMBER;

BEGIN

-- Calculate age based on the date of birth

SELECT FLOOR((SYSDATE - dob) / 365)

INTO customerAge

FROM dual;

RETURN customerAge;

END CalculateAge;

/

SHOW ERRORS FUNCTION CalculateAge;

-- Function to Compute Monthly Installment for a Loan

CREATE OR REPLACE FUNCTION CalculateMonthlyInstallment(

principalAmount NUMBER,

annualInterestRate NUMBER,

durationYears NUMBER

)

RETURN NUMBER

IS

monthlyInterestRate NUMBER;

totalPayments NUMBER;

monthlyPayment NUMBER;

BEGIN

-- Calculate monthly interest rate

monthlyInterestRate := annualInterestRate / 100 / 12;

-- Calculate total number of payments

totalPayments := durationYears \* 12;

-- Calculate the monthly installment using the formula

monthlyPayment := principalAmount \* (monthlyInterestRate \* POWER(1 + monthlyInterestRate, totalPayments)) / (POWER(1 + monthlyInterestRate, totalPayments) - 1);

RETURN monthlyPayment;

END CalculateMonthlyInstallment;

/

SHOW ERRORS FUNCTION CalculateMonthlyInstallment;

-- Function to Check if Customer Has Sufficient Balance

CREATE OR REPLACE FUNCTION HasSufficientBalance(

accountID NUMBER,

transactionAmount NUMBER

)

RETURN BOOLEAN

IS

currentBalance NUMBER;

BEGIN

-- Get the current balance of the account

SELECT Balance INTO currentBalance

FROM Accounts

WHERE AccountID = accountID AND ROWNUM = 1;

-- Check if the balance is sufficient

IF currentBalance >= transactionAmount THEN

RETURN TRUE;

ELSE

RETURN FALSE;

END IF;

END HasSufficientBalance;

/

SHOW ERRORS FUNCTION HasSufficientBalance;

-- Anonymous block to test the functions

DECLARE

v\_customerAge NUMBER;

v\_monthlyPayment NUMBER;

v\_isSufficientBalance BOOLEAN;

BEGIN

-- Test CalculateAge function

v\_customerAge := CalculateAge(TO\_DATE('1980-01-01', 'YYYY-MM-DD'));

DBMS\_OUTPUT.PUT\_LINE('Age: ' || v\_customerAge);

-- Test CalculateMonthlyInstallment function

v\_monthlyPayment := CalculateMonthlyInstallment(10000, 5, 10);

DBMS\_OUTPUT.PUT\_LINE('Monthly Installment: ' || v\_monthlyPayment);

-- Test HasSufficientBalance function

v\_isSufficientBalance := HasSufficientBalance(1, 500);

IF v\_isSufficientBalance THEN

DBMS\_OUTPUT.PUT\_LINE('Sufficient Balance: TRUE');

ELSE

DBMS\_OUTPUT.PUT\_LINE('Sufficient Balance: FALSE');

END IF;

END;

/

**Exercise 5: Triggers**

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

* + **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.

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

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

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

* + **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.

-- Enable server output

SET SERVEROUTPUT ON;

-- Create tables

CREATE TABLE Customers (

CustomerID INT PRIMARY KEY,

Name VARCHAR2(100),

DOB DATE,

Balance INT,

LastModified DATE,

IsVIP CHAR(1)

);

CREATE TABLE Accounts (

AccountID INT PRIMARY KEY,

CustomerID INT,

AccountType VARCHAR2(20),

Balance INT,

LastModified DATE,

FOREIGN KEY (CustomerID) REFERENCES Customers(CustomerID)

);

CREATE TABLE Transactions (

TransactionID INT PRIMARY KEY,

AccountID INT,

TransactionDate DATE,

Amount INT,

TransactionType VARCHAR2(10),

FOREIGN KEY (AccountID) REFERENCES Accounts(AccountID)

);

CREATE TABLE Loans (

LoanID INT PRIMARY KEY,

CustomerID INT,

LoanAmount INT,

InterestRate INT,

StartDate DATE,

EndDate DATE,

FOREIGN KEY (CustomerID) REFERENCES Customers(CustomerID)

);

CREATE TABLE Employees (

EmployeeID INT PRIMARY KEY,

Name VARCHAR2(100),

Position VARCHAR2(50),

Salary INT,

Department VARCHAR2(50),

DepartmentID INT,

HireDate DATE

);

-- Create ErrorLogs table for logging errors

CREATE TABLE ErrorLogs (

ErrorID INT PRIMARY KEY,

ErrorMessage VARCHAR2(255),

ErrorDate DATE

);

-- Create AuditLog table for maintaining an audit log

CREATE TABLE AuditLog (

AuditID INT PRIMARY KEY,

TransactionID INT,

TransactionDate DATE,

AccountID INT,

Amount INT,

TransactionType VARCHAR2(10)

);

-- Create sequence for AuditLog primary key

CREATE SEQUENCE AuditLog\_Seq

START WITH 1

INCREMENT BY 1

NOCACHE;

-- Scripts for Sample Data Insertion

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

VALUES (1, 'John Doe', TO\_DATE('1963-05-15', 'YYYY-MM-DD'), 1000, SYSDATE);

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

VALUES (2, 'Jane Smith', TO\_DATE('1990-07-20', 'YYYY-MM-DD'), 1500, SYSDATE);

INSERT INTO Accounts (AccountID, CustomerID, AccountType, Balance, LastModified)

VALUES (1, 1, 'Savings', 1000, SYSDATE);

INSERT INTO Accounts (AccountID, CustomerID, AccountType, Balance, LastModified)

VALUES (2, 2, 'Checking', 1500, SYSDATE);

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

VALUES (1, 1, SYSDATE, 200, 'Deposit');

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

VALUES (2, 2, SYSDATE, 300, 'Withdrawal');

INSERT INTO Loans (LoanID, CustomerID, LoanAmount, InterestRate, StartDate, EndDate)

VALUES (1, 1, 5000, 5, SYSDATE, ADD\_MONTHS(SYSDATE, 60));

INSERT INTO Loans (LoanID, CustomerID, LoanAmount, InterestRate, StartDate, EndDate)

VALUES (2, 2, 10000, 5, SYSDATE, SYSDATE + 25);

INSERT INTO Employees (EmployeeID, Name, Position, Salary, Department, HireDate)

VALUES (1, 'Alice Johnson', 'Manager', 70000, 'HR', TO\_DATE('2015-06-15', 'YYYY-MM-DD'));

INSERT INTO Employees (EmployeeID, Name, Position, Salary, Department, HireDate)

VALUES (2, 'Bob Brown', 'Developer', 60000, 'IT', TO\_DATE('2017-03-20', 'YYYY-MM-DD'));

-- Triggers

-- Trigger to Automatically Update LastModified Date When a Customer's Record is Updated

CREATE OR REPLACE TRIGGER UpdateCustomerLastModified

BEFORE UPDATE ON Customers

FOR EACH ROW

BEGIN

-- Update the LastModified column to the current date

:NEW.LastModified := SYSDATE;

END;

/

SHOW ERRORS TRIGGER UpdateCustomerLastModified;

-- Trigger to Maintain an Audit Log for All Transactions

CREATE OR REPLACE TRIGGER LogTransaction

AFTER INSERT ON Transactions

FOR EACH ROW

BEGIN

-- Insert transaction details into the AuditLog table

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

VALUES (AuditLog\_Seq.NEXTVAL, :NEW.TransactionID, :NEW.TransactionDate, :NEW.AccountID, :NEW.Amount, :NEW.TransactionType);

END;

/

SHOW ERRORS TRIGGER LogTransaction;

-- Trigger to Enforce Business Rules on Deposits and Withdrawals

CREATE OR REPLACE TRIGGER CheckTransactionRules

BEFORE INSERT ON Transactions

FOR EACH ROW

DECLARE

accountBalance DECIMAL(15,2);

BEGIN

-- Check if the transaction type is withdrawal and ensure it does not exceed the balance

IF :NEW.TransactionType = 'WITHDRAWAL' THEN

-- Get the current balance of the account

SELECT Balance INTO accountBalance

FROM Accounts

WHERE AccountID = :NEW.AccountID

FOR UPDATE;

-- Ensure the withdrawal does not exceed the balance

IF :NEW.Amount > accountBalance THEN

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

END IF;

END IF;

-- Check if the transaction type is deposit and ensure the amount is positive

IF :NEW.TransactionType = 'DEPOSIT' THEN

IF :NEW.Amount <= 0 THEN

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

END IF;

END IF;

END;

/

SHOW ERRORS TRIGGER CheckTransactionRules;

-- Anonymous block to test the triggers

BEGIN

-- Update a customer record to test UpdateCustomerLastModified trigger

UPDATE Customers SET Name = 'John Doe Updated' WHERE CustomerID = 1;

-- Insert a transaction to test LogTransaction and CheckTransactionRules triggers

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

VALUES (3, 1, SYSDATE, 500, 'DEPOSIT');

-- Insert a withdrawal transaction to test CheckTransactionRules trigger

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

VALUES (4, 1, SYSDATE, 200, 'WITHDRAWAL');

-- Test insufficient funds scenario

BEGIN

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

VALUES (5, 1, SYSDATE, 2000, 'WITHDRAWAL');

EXCEPTION

WHEN OTHERS THEN

DBMS\_OUTPUT.PUT\_LINE('before insufficient funds');

DBMS\_OUTPUT.PUT\_LINE(SQLERRM);

DBMS\_OUTPUT.PUT\_LINE('after insufficient funds');

END;

-- Test negative deposit scenario

BEGIN

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

VALUES (6, 1, SYSDATE, -500, 'DEPOSIT');

EXCEPTION

WHEN OTHERS THEN

DBMS\_OUTPUT.PUT\_LINE('before negative deposit');

DBMS\_OUTPUT.PUT\_LINE(SQLERRM);

DBMS\_OUTPUT.PUT\_LINE('after negative deposit');

END;

END;

/

**Exercise 6: Cursors**

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

* + **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.

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

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

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

* + **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.

-- Enable server output

SET SERVEROUTPUT ON;

-- Create tables

CREATE TABLE Customers (

CustomerID INT PRIMARY KEY,

Name VARCHAR2(100),

DOB DATE,

Balance INT,

LastModified DATE,

IsVIP CHAR(1)

);

CREATE TABLE Accounts (

AccountID INT PRIMARY KEY,

CustomerID INT,

AccountType VARCHAR2(20),

Balance INT,

LastModified DATE,

FOREIGN KEY (CustomerID) REFERENCES Customers(CustomerID)

);

CREATE TABLE Transactions (

TransactionID INT PRIMARY KEY,

AccountID INT,

TransactionDate DATE,

Amount INT,

TransactionType VARCHAR2(10),

FOREIGN KEY (AccountID) REFERENCES Accounts(AccountID)

);

CREATE TABLE Loans (

LoanID INT PRIMARY KEY,

CustomerID INT,

LoanAmount INT,

InterestRate INT,

StartDate DATE,

EndDate DATE,

FOREIGN KEY (CustomerID) REFERENCES Customers(CustomerID)

);

CREATE TABLE Employees (

EmployeeID INT PRIMARY KEY,

Name VARCHAR2(100),

Position VARCHAR2(50),

Salary INT,

Department VARCHAR2(50),

DepartmentID INT,

HireDate DATE

);

-- Create ErrorLogs table for logging errors

CREATE TABLE ErrorLogs (

ErrorID INT PRIMARY KEY,

ErrorMessage VARCHAR2(255),

ErrorDate DATE

);

-- Scripts for Sample Data Insertion

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

VALUES (1, 'John Doe', TO\_DATE('1963-05-15', 'YYYY-MM-DD'), 1000, SYSDATE);

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

VALUES (2, 'Jane Smith', TO\_DATE('1990-07-20', 'YYYY-MM-DD'), 1500, SYSDATE);

INSERT INTO Accounts (AccountID, CustomerID, AccountType, Balance, LastModified)

VALUES (1, 1, 'Savings', 1000, SYSDATE);

INSERT INTO Accounts (AccountID, CustomerID, AccountType, Balance, LastModified)

VALUES (2, 2, 'Checking', 1500, SYSDATE);

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

VALUES (1, 1, SYSDATE, 200, 'Deposit');

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

VALUES (2, 2, SYSDATE, 300, 'Withdrawal');

INSERT INTO Loans (LoanID, CustomerID, LoanAmount, InterestRate, StartDate, EndDate)

VALUES (1, 1, 5000, 5, SYSDATE, ADD\_MONTHS(SYSDATE, 60));

INSERT INTO Loans (LoanID, CustomerID, LoanAmount, InterestRate, StartDate, EndDate)

VALUES (2, 2, 10000, 5,SYSDATE, SYSDATE+25);

INSERT INTO Employees (EmployeeID, Name, Position, Salary, Department, HireDate)

VALUES (1, 'Alice Johnson', 'Manager', 70000, 'HR', TO\_DATE('2015-06-15', 'YYYY-MM-DD'));

INSERT INTO Employees (EmployeeID, Name, Position, Salary, Department, HireDate)

VALUES (2, 'Bob Brown', 'Developer', 60000, 'IT', TO\_DATE('2017-03-20', 'YYYY-MM-DD'));

-- Scenario 1: Generate Monthly Statements for All Customers

CREATE OR REPLACE PROCEDURE GenerateMonthlyStatements AS

-- Declare cursor and variables

CURSOR c\_monthly\_statements IS

SELECT a.CustomerID, t.TransactionDate, t.Amount

FROM Transactions t

JOIN Accounts a ON t.AccountID = a.AccountID

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

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

p\_customer\_id INT;

p\_transaction\_date DATE;

p\_amount DECIMAL(10, 2);

BEGIN

-- Loop through cursor

FOR r\_monthly\_statement IN c\_monthly\_statements LOOP

p\_customer\_id := r\_monthly\_statement.CustomerID;

p\_transaction\_date := r\_monthly\_statement.TransactionDate;

p\_amount := r\_monthly\_statement.Amount;

-- Print or process the statement

DBMS\_OUTPUT.PUT\_LINE('CustomerID: ' || p\_customer\_id ||

', Date: ' || p\_transaction\_date ||

', Amount: ' || p\_amount);

END LOOP;

END;

/

-- Scenario 2: Apply Annual Fee to All Accounts

CREATE OR REPLACE PROCEDURE ApplyAnnualFee AS

-- Declare cursor and variables

CURSOR c\_accounts IS

SELECT AccountID, Balance

FROM Accounts;

p\_account\_id INT;

p\_balance DECIMAL(10, 2);

p\_annual\_fee DECIMAL(10, 2) := 50.00; -- Example annual fee amount

BEGIN

-- Loop through cursor

FOR r\_account IN c\_accounts LOOP

p\_account\_id := r\_account.AccountID;

p\_balance := r\_account.Balance;

-- Deduct annual fee from account balance

UPDATE Accounts

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

WHERE AccountID = p\_account\_id;

DBMS\_OUTPUT.PUT\_LINE('AccountID: ' || p\_account\_id || ' - Annual fee applied.');

END LOOP;

END;

/

-- Scenario 3: Update Interest Rates for All Loans Based on a New Policy

CREATE OR REPLACE PROCEDURE UpdateLoanInterestRates AS

-- Declare cursor and variables

CURSOR c\_loans IS

SELECT LoanID, InterestRate

FROM Loans;

p\_loan\_id INT;

p\_interest\_rate DECIMAL(5, 2);

p\_new\_interest\_rate DECIMAL(5, 2);

BEGIN

-- Loop through cursor

FOR r\_loan IN c\_loans LOOP

p\_loan\_id := r\_loan.LoanID;

p\_interest\_rate := r\_loan.InterestRate;

-- Apply new interest rate based on policy

p\_new\_interest\_rate := p\_interest\_rate \* 1.05; -- Example: increase by 5%

UPDATE Loans

SET InterestRate = p\_new\_interest\_rate

WHERE LoanID = p\_loan\_id;

DBMS\_OUTPUT.PUT\_LINE('LoanID: ' || p\_loan\_id || ' - Interest rate updated to: ' || p\_new\_interest\_rate);

END LOOP;

END;

/

BEGIN

GenerateMonthlyStatements;

END;

/

BEGIN

ApplyAnnualFee;

END;

/

BEGIN

UpdateLoanInterestRates;

END;

/

**Exercise 7: Packages**

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

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

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

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

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

* + **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.

-- Enable server output

SET SERVEROUTPUT ON;

-- Create tables

CREATE TABLE Customers (

CustomerID INT PRIMARY KEY,

Name VARCHAR2(100),

DOB DATE,

Balance INT,

LastModified DATE,

IsVIP CHAR(1)

);

CREATE TABLE Accounts (

AccountID INT PRIMARY KEY,

CustomerID INT,

AccountType VARCHAR2(20),

Balance INT,

LastModified DATE,

FOREIGN KEY (CustomerID) REFERENCES Customers(CustomerID)

);

CREATE TABLE Transactions (

TransactionID INT PRIMARY KEY,

AccountID INT,

TransactionDate DATE,

Amount INT,

TransactionType VARCHAR2(10),

FOREIGN KEY (AccountID) REFERENCES Accounts(AccountID)

);

CREATE TABLE Loans (

LoanID INT PRIMARY KEY,

CustomerID INT,

LoanAmount INT,

InterestRate INT,

StartDate DATE,

EndDate DATE,

FOREIGN KEY (CustomerID) REFERENCES Customers(CustomerID)

);

CREATE TABLE Employees (

EmployeeID INT PRIMARY KEY,

Name VARCHAR2(100),

Position VARCHAR2(50),

Salary INT,

Department VARCHAR2(50),

DepartmentID INT,

HireDate DATE

);

-- Create ErrorLogs table for logging errors

CREATE TABLE ErrorLogs (

ErrorID INT PRIMARY KEY,

ErrorMessage VARCHAR2(255),

ErrorDate DATE

);

-- Scripts for Sample Data Insertion

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

VALUES (1, 'John Doe', TO\_DATE('1963-05-15', 'YYYY-MM-DD'), 1000, SYSDATE);

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

VALUES (2, 'Jane Smith', TO\_DATE('1990-07-20', 'YYYY-MM-DD'), 1500, SYSDATE);

INSERT INTO Accounts (AccountID, CustomerID, AccountType, Balance, LastModified)

VALUES (1, 1, 'Savings', 1000, SYSDATE);

INSERT INTO Accounts (AccountID, CustomerID, AccountType, Balance, LastModified)

VALUES (2, 2, 'Checking', 1500, SYSDATE);

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

VALUES (1, 1, SYSDATE, 200, 'Deposit');

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

VALUES (2, 2, SYSDATE, 300, 'Withdrawal');

INSERT INTO Loans (LoanID, CustomerID, LoanAmount, InterestRate, StartDate, EndDate)

VALUES (1, 1, 5000, 5, SYSDATE, ADD\_MONTHS(SYSDATE, 60));

INSERT INTO Loans (LoanID, CustomerID, LoanAmount, InterestRate, StartDate, EndDate)

VALUES (2, 2, 10000, 5, SYSDATE, SYSDATE+25);

INSERT INTO Employees (EmployeeID, Name, Position, Salary, Department, HireDate)

VALUES (1, 'Alice Johnson', 'Manager', 70000, 'HR', TO\_DATE('2015-06-15', 'YYYY-MM-DD'));

INSERT INTO Employees (EmployeeID, Name, Position, Salary, Department, HireDate)

VALUES (2, 'Bob Brown', 'Developer', 60000, 'IT', TO\_DATE('2017-03-20', 'YYYY-MM-DD'));

-- SCENARIO - 1

-- Package for Customer Management

CREATE OR REPLACE PROCEDURE AddCustomer(

p\_ID IN INT,

p\_FullName IN VARCHAR2,

p\_DateOfBirth IN DATE,

p\_InitialBalance IN DECIMAL

) AS

BEGIN

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

VALUES (p\_ID, p\_FullName, p\_DateOfBirth, p\_InitialBalance);

INSERT INTO Accounts (CustomerID, Balance)

VALUES (p\_ID, p\_InitialBalance);

END;

/

CREATE OR REPLACE PROCEDURE UpdateCustomerDetails(

p\_ID IN INT,

p\_FullName IN VARCHAR2,

p\_DateOfBirth IN DATE

) AS

BEGIN

UPDATE Customers

SET Name = p\_FullName, DOB = p\_DateOfBirth

WHERE CustomerID = p\_ID;

END;

/

CREATE OR REPLACE FUNCTION GetCustomerBalance(

p\_ID IN INT

)

RETURN DECIMAL IS

v\_Balance DECIMAL(10, 2);

BEGIN

SELECT Balance INTO v\_Balance

FROM Accounts

WHERE CustomerID = p\_ID;

RETURN v\_Balance;

EXCEPTION

WHEN NO\_DATA\_FOUND THEN

RETURN 0; -- Return 0 if no balance found

END;

/

-- SCENARIO - 2

-- Package for Employee Management

CREATE OR REPLACE PROCEDURE HireEmployee(

p\_ID IN INT,

p\_FullName IN VARCHAR2,

p\_JobTitle IN VARCHAR2,

p\_AnnualSalary IN DECIMAL

) AS

BEGIN

INSERT INTO Employees (EmployeeID, Name, Position, Salary)

VALUES (p\_ID, p\_FullName, p\_JobTitle, p\_AnnualSalary);

END;

/

CREATE OR REPLACE PROCEDURE UpdateEmployeeDetails(

p\_ID IN INT,

p\_FullName IN VARCHAR2,

p\_JobTitle IN VARCHAR2,

p\_AnnualSalary IN DECIMAL

) AS

BEGIN

UPDATE Employees

SET Name = p\_FullName, Position = p\_JobTitle, Salary = p\_AnnualSalary

WHERE EmployeeID = p\_ID;

END;

/

CREATE OR REPLACE FUNCTION CalculateAnnualSalary(

p\_ID IN INT

)

RETURN DECIMAL IS

v\_AnnualSalary DECIMAL(10, 2);

BEGIN

SELECT Salary \* 12 INTO v\_AnnualSalary

FROM Employees

WHERE EmployeeID = p\_ID;

RETURN v\_AnnualSalary;

EXCEPTION

WHEN NO\_DATA\_FOUND THEN

RETURN 0; -- Return 0 if employee not found

END;

/

-- SCENARIO - 3

-- Package for Account Operations

CREATE OR REPLACE PROCEDURE OpenAccount(

p\_CustomerID IN INT,

p\_InitialBalance IN DECIMAL

) AS

BEGIN

INSERT INTO Accounts (CustomerID, Balance)

VALUES (p\_CustomerID, p\_InitialBalance);

END;

/

CREATE OR REPLACE PROCEDURE CloseAccount(

p\_CustomerID IN INT

) AS

BEGIN

DELETE FROM Accounts

WHERE CustomerID = p\_CustomerID;

END;

/

CREATE OR REPLACE FUNCTION GetTotalBalance(

p\_CustomerID IN INT

)

RETURN DECIMAL IS

v\_TotalBalance DECIMAL(10, 2);

BEGIN

SELECT SUM(Balance) INTO v\_TotalBalance

FROM Accounts

WHERE CustomerID = p\_CustomerID;

RETURN v\_TotalBalance;

EXCEPTION

WHEN NO\_DATA\_FOUND THEN

RETURN 0; -- Return 0 if no accounts found

END;

/

**Schema to be Created**

*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*

*);*

**Example Scripts for Sample Data Insertion**

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

*VALUES (1, 'John Doe', TO\_DATE('1985-05-15', 'YYYY-MM-DD'), 1000, SYSDATE);*

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

*VALUES (2, 'Jane Smith', TO\_DATE('1990-07-20', 'YYYY-MM-DD'), 1500, SYSDATE);*

*INSERT INTO Accounts (AccountID, CustomerID, AccountType, Balance, LastModified)*

*VALUES (1, 1, 'Savings', 1000, SYSDATE);*

*INSERT INTO Accounts (AccountID, CustomerID, AccountType, Balance, LastModified)*

*VALUES (2, 2, 'Checking', 1500, SYSDATE);*

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

*VALUES (1, 1, SYSDATE, 200, 'Deposit');*

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

*VALUES (2, 2, SYSDATE, 300, 'Withdrawal');*

*INSERT INTO Loans (LoanID, CustomerID, LoanAmount, InterestRate, StartDate, EndDate)*

*VALUES (1, 1, 5000, 5, SYSDATE, ADD\_MONTHS(SYSDATE, 60));*

*INSERT INTO Employees (EmployeeID, Name, Position, Salary, Department, HireDate)*

*VALUES (1, 'Alice Johnson', 'Manager', 70000, 'HR', TO\_DATE('2015-06-15', 'YYYY-MM-DD'));*

*INSERT INTO Employees (EmployeeID, Name, Position, Salary, Department, HireDate)*

*VALUES (2, 'Bob Brown', 'Developer', 60000, 'IT', TO\_DATE('2017-03-20', 'YYYY-MM-DD'));*