**Mandatory Questions for PL/SQL Programming**

To fully test all scenarios across exercises, I have added some additional customers, accounts, loans, transactions, and employees. Below are the insert statements grouped by the purpose they serve.

**1) Insert Elder Customers (To Test Age > 60 Condition for Loan Discounts)**

These customers were added to test **Scenario 1: Applying a 1% interest discount for customers above 60 years old**:

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

VALUES (3, 'Ramesh Sharma', TO\_DATE('1950-02-15', 'YYYY-MM-DD'), 12000, SYSDATE);

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

VALUES (4, 'Savitri Devi', TO\_DATE('1955-11-10', 'YYYY-MM-DD'), 3000, SYSDATE);

**2) Insert Younger and Middle-aged Customers (To Cover Various Balances and Ages)**

These customers let you test **VIP promotion**, balances around $10,000, and more general scenarios:

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

VALUES (5, 'Arjun Singh', TO\_DATE('1978-06-22', 'YYYY-MM-DD'), 500, SYSDATE);

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

VALUES (6, 'Priya Verma', TO\_DATE('1989-12-05', 'YYYY-MM-DD'), 25000, SYSDATE);

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

VALUES (7, 'Vikram Rao', TO\_DATE('1995-04-18', 'YYYY-MM-DD'), 9500, SYSDATE);

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

VALUES (8, 'Anjali Nair', TO\_DATE('2000-09-09', 'YYYY-MM-DD'), 10500, SYSDATE);

**3) Insert Accounts for New Customers (To Test Interest, Transfers, and Account Updates)**

Each customer needs at least one account to properly test **interest application, fund transfers, and balance checks**:

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

VALUES (3, 3, 'Savings', 12000, SYSDATE);

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

VALUES (4, 4, 'Savings', 3000, SYSDATE);

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

VALUES (5, 5, 'Checking', 500, SYSDATE);

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

VALUES (6, 6, 'Savings', 25000, SYSDATE);

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

VALUES (7, 7, 'Checking', 9500, SYSDATE);

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

VALUES (8, 8, 'Savings', 10500, SYSDATE);

**4) Insert Transactions (To Support Monthly Statements and Fees Scenarios)**

Transactions are necessary to test **cursors for generating statements, applying fees, etc.**:

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

VALUES (3, 3, SYSDATE, 500, 'Deposit');

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

VALUES (4, 4, SYSDATE, 150, 'Withdrawal');

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

VALUES (5, 5, SYSDATE, 200, 'Deposit');

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

VALUES (6, 6, SYSDATE, 300, 'Deposit');

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

VALUES (7, 7, SYSDATE, 400, 'Withdrawal');

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

VALUES (8, 8, SYSDATE, 600, 'Deposit');

**5) Insert Loans (To Test Loan Due Reminders and Various End Dates)**

These loans let you check **Scenario 3 reminders (due within 30 days) and other loan-based operations**:

-- Loan due within 30 days (should trigger reminder)

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

VALUES (3, 3, 7000, 6, SYSDATE, SYSDATE + 15);

-- Loan due later (should NOT trigger reminder)

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

VALUES (4, 4, 4000, 7, SYSDATE, SYSDATE + 90);

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

VALUES (5, 5, 5000, 8, SYSDATE, ADD\_MONTHS(SYSDATE, 12));

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

VALUES (6, 6, 10000, 5, SYSDATE, ADD\_MONTHS(SYSDATE, 24));

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

VALUES (7, 7, 12000, 9, SYSDATE, SYSDATE + 10);

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

VALUES (8, 8, 15000, 6, SYSDATE, ADD\_MONTHS(SYSDATE, 36));

**6) Insert Additional Employees (To Test Salary Updates and Bonuses)**

Extra employees allow thorough testing of **UpdateSalary and UpdateEmployeeBonus procedures**:

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

VALUES (3, 'Neha Agarwal', 'Analyst', 50000, 'Finance', TO\_DATE('2019-01-10', 'YYYY-MM-DD'));

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

VALUES (4, 'Rahul Kapoor', 'Team Lead', 80000, 'IT', TO\_DATE('2016-08-22', 'YYYY-MM-DD'));

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

VALUES (5, 'Deepika Joshi', 'HR Executive', 45000, 'HR', TO\_DATE('2018-11-05', 'YYYY-MM-DD'));

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

VALUES (6, 'Sandeep Mehra', 'Developer', 60000, 'IT', TO\_DATE('2020-03-18', 'YYYY-MM-DD'));

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

VALUES (7, 'Sunita Rao', 'Branch Manager', 90000, 'Operations', TO\_DATE('2014-07-12', 'YYYY-MM-DD'))

**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.

***Solution:***

SET SERVEROUTPUT ON;

DECLARE

-- Cursor to get each customer with their calculated age

CURSOR cust\_cursor IS

SELECT CustomerID, TRUNC(MONTHS\_BETWEEN(SYSDATE, DOB) / 12) AS age

FROM Customers;

BEGIN

-- Loop through each customer from the cursor

FOR cust\_record IN cust\_cursor LOOP

-- Check if customer is older than 60

IF cust\_record.age > 60 THEN

-- Apply a 1% discount on their loan interest rate

UPDATE Loans

SET InterestRate = InterestRate - 1

WHERE CustomerID = cust\_record.CustomerID;

-- Output a confirmation message

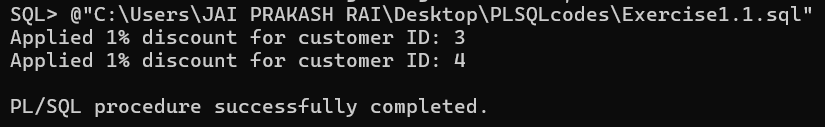
DBMS\_OUTPUT.PUT\_LINE('Applied 1% discount for customer ID: ' || cust\_record.CustomerID);

END IF;

END LOOP;

END;

/

***Output:*** ****

**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.

***Solution:***SET SERVEROUTPUT ON;

-- IMPORTANT: Make sure you've added the IsVIP column:

-- ALTER TABLE Customers ADD (IsVIP VARCHAR2(5));

DECLARE

-- Cursor to get each customer's ID and balance

CURSOR vip\_cursor IS

SELECT CustomerID, Balance FROM Customers;

BEGIN

-- Loop through each customer

FOR vip\_record IN vip\_cursor LOOP

-- Check if balance is greater than 10,000

IF vip\_record.Balance > 10000 THEN

-- Update the IsVIP flag for this customer

UPDATE Customers

SET IsVIP = 'TRUE'

WHERE CustomerID = vip\_record.CustomerID;

-- Output a confirmation message

DBMS\_OUTPUT.PUT\_LINE('Promoted customer ID ' || vip\_record.CustomerID || ' to VIP.');

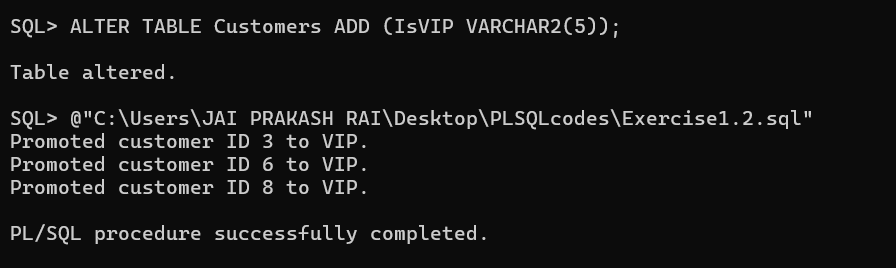
END IF;

END LOOP;

END;

/

***Output:***

****

**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.

***Solution:***  
  
SET SERVEROUTPUT ON;

DECLARE

-- Cursor to fetch loans due in the next 30 days, along with customer name

CURSOR due\_cursor IS

SELECT Loans.LoanID, Loans.CustomerID, Loans.EndDate, Customers.Name

FROM Loans

JOIN Customers ON Loans.CustomerID = Customers.CustomerID

WHERE Loans.EndDate BETWEEN SYSDATE AND (SYSDATE + 30);

BEGIN

-- Loop through each loan found by the cursor

FOR due\_record IN due\_cursor LOOP

-- Output a reminder message for each loan due soon

DBMS\_OUTPUT.PUT\_LINE(

'Reminder: Loan ID ' || due\_record.LoanID ||

' for customer ' || due\_record.Name ||

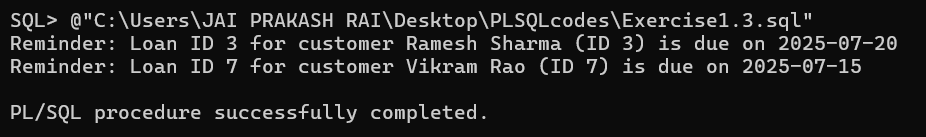
' (ID ' || due\_record.CustomerID || ') is due on ' || TO\_CHAR(due\_record.EndDate, 'YYYY-MM-DD')

);

END LOOP;

END;

/

***Output:***  
  


**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.

***Solution:***

SET SERVEROUTPUT ON;

CREATE OR REPLACE PROCEDURE ProcessMonthlyInterest AS

BEGIN

-- Update all savings accounts by adding 1% of their current balance

UPDATE Accounts

SET Balance = Balance + (Balance \* 0.01)

WHERE AccountType = 'Savings';

DBMS\_OUTPUT.PUT\_LINE('Monthly interest applied to all savings accounts.');

EXCEPTION

WHEN OTHERS THEN

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

END;

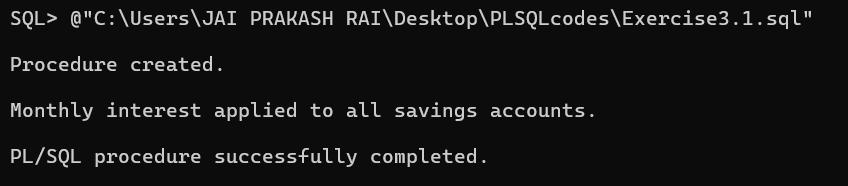
/

BEGIN

ProcessMonthlyInterest;

END;

/

***Output:***  


**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.

***Solution:***  
  
SET SERVEROUTPUT ON;

CREATE OR REPLACE PROCEDURE UpdateEmployeeBonus (

p\_department IN VARCHAR2,

p\_bonus\_pct IN NUMBER

) AS

BEGIN

-- Increase salary for employees in the specified department by bonus percentage

UPDATE Employees

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

WHERE Department = p\_department;

DBMS\_OUTPUT.PUT\_LINE('Updated salaries in department ' || p\_department || ' by ' || p\_bonus\_pct || '%.');

EXCEPTION

WHEN OTHERS THEN

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

END;

/

BEGIN

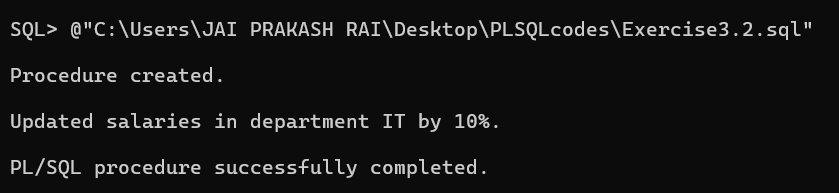
-- Example: give 10% bonus to IT department

UpdateEmployeeBonus('IT', 10);

END;

/

***Output:***



**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.

***Solution:***

SET SERVEROUTPUT ON;

CREATE OR REPLACE PROCEDURE TransferFunds (

p\_source\_account IN NUMBER,

p\_target\_account IN NUMBER,

p\_amount IN NUMBER

) AS

v\_source\_balance NUMBER;

BEGIN

-- Get current balance of source account

SELECT Balance INTO v\_source\_balance FROM Accounts

WHERE AccountID = p\_source\_account

FOR UPDATE;

-- Check if source account has enough funds

IF v\_source\_balance < p\_amount THEN

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

END IF;

-- Deduct from source account

UPDATE Accounts

SET Balance = Balance - p\_amount

WHERE AccountID = p\_source\_account;

-- Add to target account

UPDATE Accounts

SET Balance = Balance + p\_amount

WHERE AccountID = p\_target\_account;

DBMS\_OUTPUT.PUT\_LINE('Transferred ' || p\_amount || ' from Account ' || p\_source\_account || ' to Account ' || p\_target\_account || '.');

EXCEPTION

WHEN NO\_DATA\_FOUND THEN

DBMS\_OUTPUT.PUT\_LINE('Source or target account not found.');

WHEN OTHERS THEN

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

ROLLBACK;

END;

/

-- Immediately call the procedure in an anonymous block

BEGIN

-- Example: transfer 500 from account 6 to account 7

TransferFunds(6, 7, 500);

END;

/  
  
***Output:***

