**Schema:**

*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 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)*

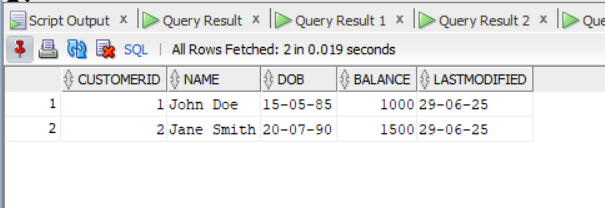
*VALUES (1, 'Alice Johnson', 'Manager', 70000, 'HR', TO\_DATE('2015-06-15', 'YYYY-MM-DD'));*

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

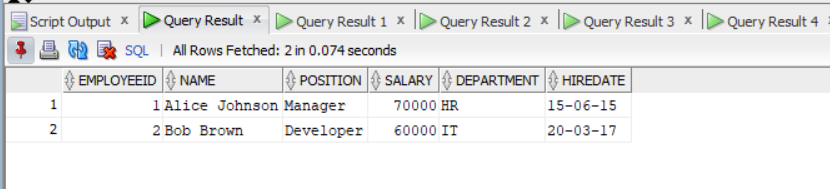
*VALUES (2, 'Bob Brown', 'Developer', 60000, 'IT', TO\_DATE('2017-03-20', 'YYYY-MM-DD'));*

**Tables:**

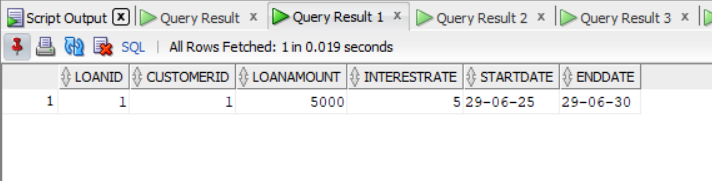
Customers:



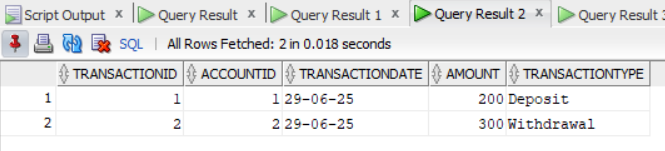
Employees:

****

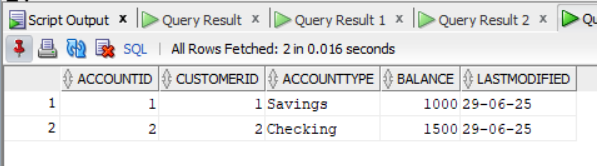
Loans:

****

Transactions:

****

Accounts:

****

**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.

**Code :**

BEGIN

DBMS\_OUTPUT.PUT\_LINE('--- Applying 1% Discount to Senior Citizens ---');

FOR cust IN (

SELECT CustomerID, Name, DOB

FROM Customers

) LOOP

IF TRUNC(MONTHS\_BETWEEN(SYSDATE, cust.DOB) / 12) > 60 THEN

UPDATE Loans

SET InterestRate = InterestRate - 1

WHERE CustomerID = cust.CustomerID;

DBMS\_OUTPUT.PUT\_LINE('Interest rate for customer ' || cust.Name || ' (ID: ' || cust.CustomerID || ') discounted by 1%.');

END IF;

END LOOP;

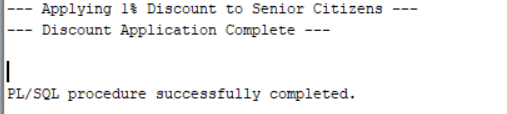
COMMIT;

DBMS\_OUTPUT.PUT\_LINE('--- Discount Application Complete ---');

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.

Code:

ALTER TABLE Customers ADD (IsVIP CHAR(1));

BEGIN

DBMS\_OUTPUT.PUT\_LINE('--- Promoting Customers to VIP ---');

FOR cust IN (

SELECT CustomerID, Name, Balance FROM Customers

) LOOP

IF cust.Balance > 10000 THEN

UPDATE Customers

SET IsVIP = 'TRUE'

WHERE CustomerID = cust.CustomerID;

else

UPDATE Customers

SET IsVIP = 'FALSE'

WHERE CustomerID = cust.CustomerID;

DBMS\_OUTPUT.PUT\_LINE('Customer ' || cust.Name || ' (ID: ' || cust.CustomerID || ') marked as VIP.');

END IF;

END LOOP;

COMMIT;

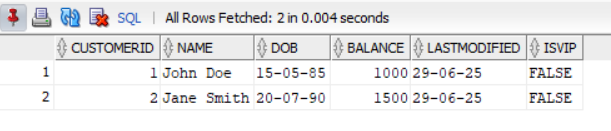
DBMS\_OUTPUT.PUT\_LINE('--- VIP Promotion Complete ---');

END;

/

select \* from customers;

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.

**Code:**

BEGIN

DBMS\_OUTPUT.PUT\_LINE('--- Loan Due Reminders ---');

FOR loan IN (

SELECT l.LoanID, l.CustomerID, l.EndDate, c.Name

FROM Loans l

JOIN Customers c ON l.CustomerID = c.CustomerID

WHERE l.EndDate <= SYSDATE + 30

) LOOP

DBMS\_OUTPUT.PUT\_LINE('Reminder: Loan ' || loan.LoanID ||

' for Customer ' || loan.Name ||

' (ID: ' || loan.CustomerID || ') is due on ' ||

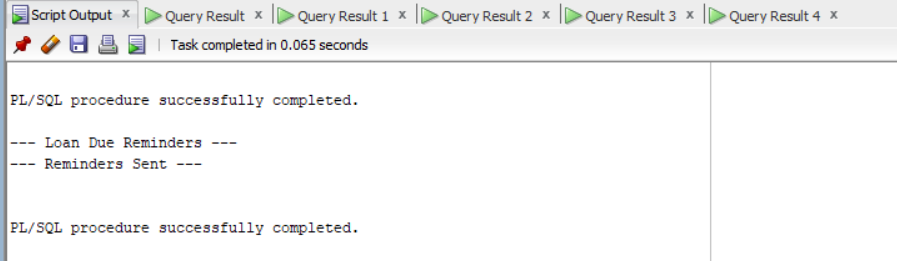
TO\_CHAR(loan.EndDate, 'DD-MON-YYYY'));

END LOOP;

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

END;

/

Output: (Note: There is no loan due in 30 for customer so my output is this )  


**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.

**Code:**

CREATE OR REPLACE PROCEDURE ProcessMonthlyInterest IS

BEGIN

DBMS\_OUTPUT.PUT\_LINE('--- Processing Monthly Interest for Savings Accounts ---');

FOR acc IN (

SELECT AccountID, Balance FROM Accounts WHERE AccountType = 'Savings'

) LOOP

UPDATE Accounts

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

LastModified = SYSDATE

WHERE AccountID = acc.AccountID;

DBMS\_OUTPUT.PUT\_LINE('Interest added to Account ID: ' || acc.AccountID);

END LOOP;

COMMIT;

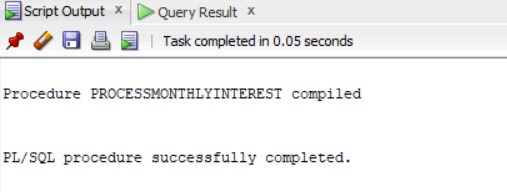
DBMS\_OUTPUT.PUT\_LINE('--- Monthly Interest Processing Complete ---');

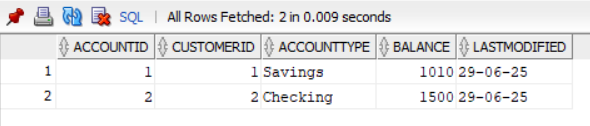
END;

/

SELECT \* FROM accounts;

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.

**Code:**

CREATE OR REPLACE PROCEDURE UpdateEmployeeBonus (

dept\_name IN VARCHAR2,

bonus\_percent IN NUMBER

) IS

BEGIN

DBMS\_OUTPUT.PUT\_LINE('--- Applying Bonus to Department: ' || dept\_name || ' ---');

FOR emp IN (

SELECT EmployeeID, Salary FROM Employees WHERE Department = dept\_name

) LOOP

UPDATE Employees

SET Salary = Salary + (emp.Salary \* bonus\_percent / 100)

WHERE EmployeeID = emp.EmployeeID;

DBMS\_OUTPUT.PUT\_LINE('Bonus applied to Employee ID: ' || emp.EmployeeID);

END LOOP;

COMMIT;

DBMS\_OUTPUT.PUT\_LINE('--- Bonus Application Complete ---');

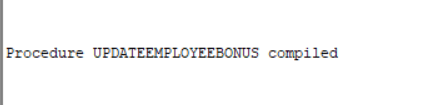
END;

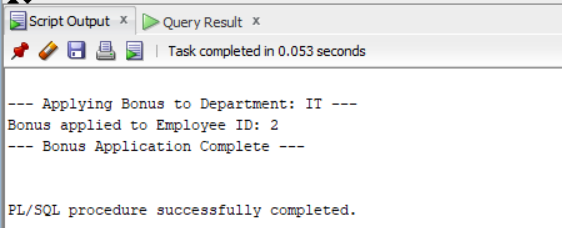
/

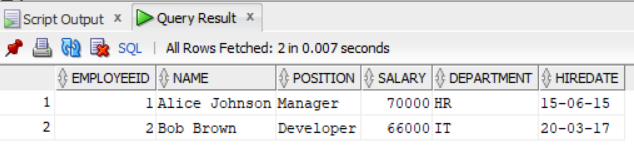
EXEC UpdateEmployeeBonus('IT', 10); -- Adds 10% bonus to all employees in IT

select \* from employees;

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.

**Code:**

CREATE SEQUENCE Transactions\_seq1 START WITH 3 INCREMENT BY 1;

ALTER TABLE Transactions MODIFY TransactionType VARCHAR2(20);

CREATE OR REPLACE PROCEDURE TransferFunds (

p\_from\_account IN NUMBER,

p\_to\_account IN NUMBER,

p\_amount IN NUMBER

) IS

v\_balance NUMBER;

BEGIN

-- Get balance of source account

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

-- Check if sufficient funds

IF v\_balance >= p\_amount THEN

-- Deduct from source

UPDATE Accounts

SET Balance = Balance - p\_amount,

LastModified = SYSDATE

WHERE AccountID = p\_from\_account;

-- Add to destination

UPDATE Accounts

SET Balance = Balance + p\_amount,

LastModified = SYSDATE

WHERE AccountID = p\_to\_account;

-- Insert transaction logs (optional)

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

VALUES (Transactions\_seq.NEXTVAL, p\_from\_account, SYSDATE, p\_amount, 'Transfer-OUT');

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

VALUES (Transactions\_seq.NEXTVAL, p\_to\_account, SYSDATE, p\_amount, 'Transfer-IN');

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

COMMIT;

ELSE

DBMS\_OUTPUT.PUT\_LINE('Insufficient balance in source account.');

END IF;

EXCEPTION

WHEN NO\_DATA\_FOUND THEN

DBMS\_OUTPUT.PUT\_LINE('One of the accounts does not exist.');

WHEN OTHERS THEN

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

ROLLBACK;

END;

/

EXEC TransferFunds(1, 2, 500);

select \* from transactions;Output:

