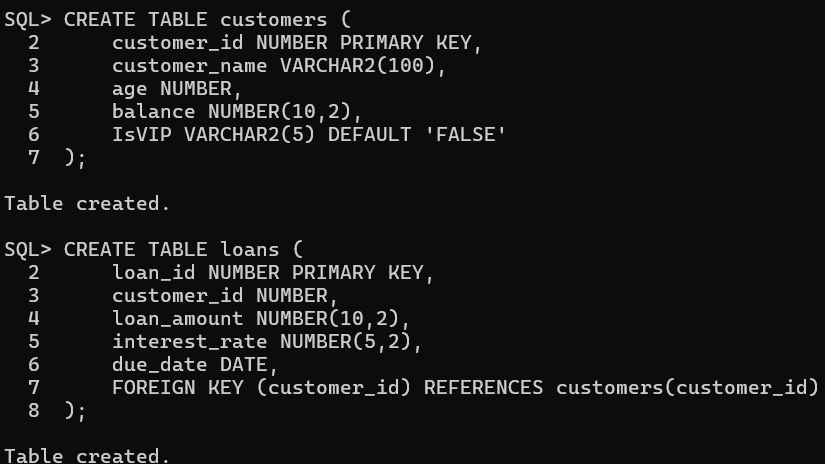
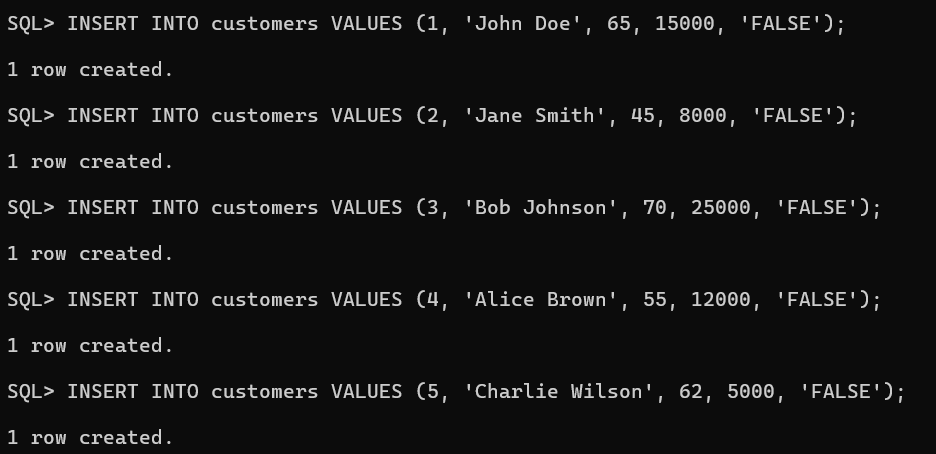
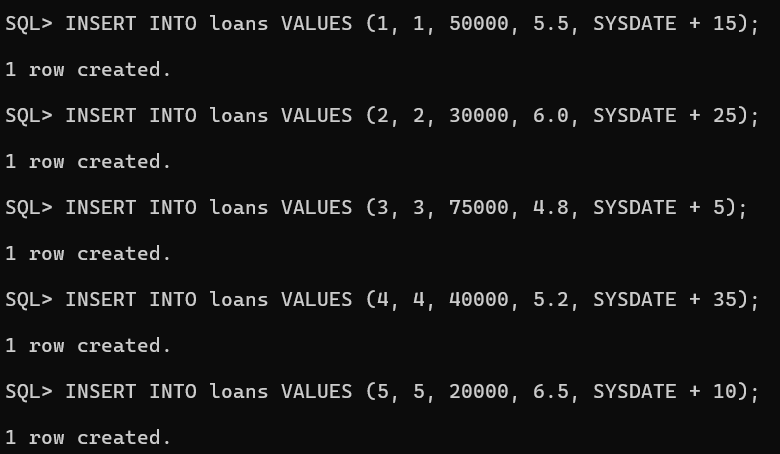
**PL/SQL EXCERSICES :**

**Exercise 1: Control Structures  
  
INITIAL TABLES AND VALUES INSERTED INTO THEM TO PERFORM THE REQUIRED OPERATIONS**







**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:**

DECLARE

CURSOR customer\_cursor IS

SELECT customer\_id, customer\_name, age

FROM customers

WHERE age > 60;

v\_customer\_id customers.customer\_id%TYPE;

v\_customer\_name customers.customer\_name%TYPE;

v\_age customers.age%TYPE;

v\_discount\_count NUMBER := 0;

BEGIN

DBMS\_OUTPUT.PUT\_LINE('Applying 1% discount to customers above 60 ');

FOR customer\_rec IN customer\_cursor LOOP

UPDATE loans

SET interest\_rate = interest\_rate - 1

WHERE customer\_id = customer\_rec.customer\_id

AND interest\_rate > 1;

IF SQL%ROWCOUNT > 0 THEN

v\_discount\_count := v\_discount\_count + 1;

DBMS\_OUTPUT.PUT\_LINE('Discount applied to customer: ' || customer\_rec.customer\_name ||

' (Age: ' || customer\_rec.age || ')');

END IF;

END LOOP;

DBMS\_OUTPUT.PUT\_LINE('Total customers who received discount: ' || v\_discount\_count);

COMMIT;

EXCEPTION

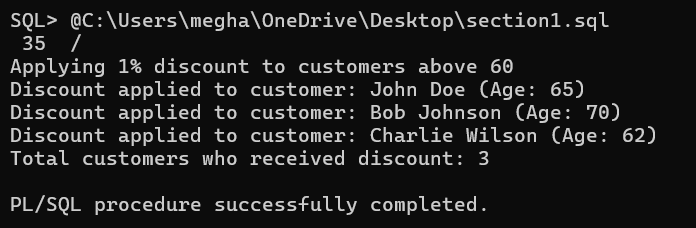
WHEN OTHERS THEN

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

ROLLBACK;

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:**

DECLARE

CURSOR customer\_cursor IS

SELECT customer\_id, customer\_name, balance

FROM customers;

v\_vip\_count NUMBER := 0;

BEGIN

DBMS\_OUTPUT.PUT\_LINE(' Setting VIP status for high-balance customers ');

FOR customer\_rec IN customer\_cursor LOOP

IF customer\_rec.balance > 10000 THEN

UPDATE customers

SET IsVIP = 'TRUE'

WHERE customer\_id = customer\_rec.customer\_id;

v\_vip\_count := v\_vip\_count + 1;

DBMS\_OUTPUT.PUT\_LINE('VIP status granted to: ' || customer\_rec.customer\_name ||

' (Balance: $' || customer\_rec.balance || ')');

ELSE

UPDATE customers

SET IsVIP = 'FALSE'

WHERE customer\_id = customer\_rec.customer\_id;

END IF;

END LOOP;

DBMS\_OUTPUT.PUT\_LINE('Total VIP customers: ' || v\_vip\_count);

COMMIT;

EXCEPTION

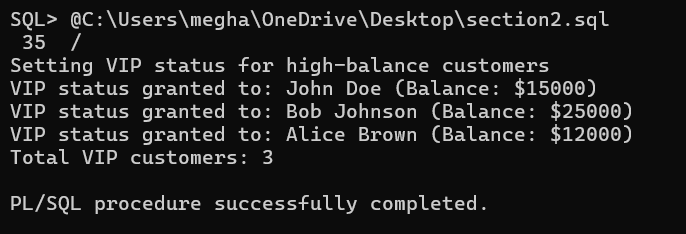
WHEN OTHERS THEN

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

ROLLBACK;

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:**

DECLARE

CURSOR loan\_cursor IS

SELECT l.loan\_id, l.customer\_id, c.customer\_name, l.loan\_amount, l.due\_date

FROM loans l

JOIN customers c ON l.customer\_id = c.customer\_id

WHERE l.due\_date BETWEEN SYSDATE AND SYSDATE + 30

ORDER BY l.due\_date;

v\_reminder\_count NUMBER := 0;

v\_days\_remaining NUMBER;

BEGIN

DBMS\_OUTPUT.PUT\_LINE('Loan Due Reminders (Next 30 Days) ');

DBMS\_OUTPUT.PUT\_LINE('Current Date: ' || TO\_CHAR(SYSDATE, 'DD-MON-YYYY'));

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

FOR loan\_rec IN loan\_cursor LOOP

v\_days\_remaining := TRUNC(loan\_rec.due\_date - SYSDATE);

v\_reminder\_count := v\_reminder\_count + 1;

DBMS\_OUTPUT.PUT\_LINE('REMINDER #' || v\_reminder\_count);

DBMS\_OUTPUT.PUT\_LINE('Customer: ' || loan\_rec.customer\_name);

DBMS\_OUTPUT.PUT\_LINE('Loan ID: ' || loan\_rec.loan\_id);

DBMS\_OUTPUT.PUT\_LINE('Loan Amount: $' || loan\_rec.loan\_amount);

DBMS\_OUTPUT.PUT\_LINE('Due Date: ' || TO\_CHAR(loan\_rec.due\_date, 'DD-MON-YYYY'));

DBMS\_OUTPUT.PUT\_LINE('Days Remaining: ' || v\_days\_remaining);

IF v\_days\_remaining <= 7 THEN

DBMS\_OUTPUT.PUT\_LINE('\*\*\* URGENT: Payment due within a week! \*\*\*');

ELSIF v\_days\_remaining <= 15 THEN

DBMS\_OUTPUT.PUT\_LINE('\*\* IMPORTANT: Payment due soon \*\*');

END IF;

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

END LOOP;

IF v\_reminder\_count = 0 THEN

DBMS\_OUTPUT.PUT\_LINE('No loans due within the next 30 days.');

ELSE

DBMS\_OUTPUT.PUT\_LINE('Total reminders sent: ' || v\_reminder\_count);

END IF;

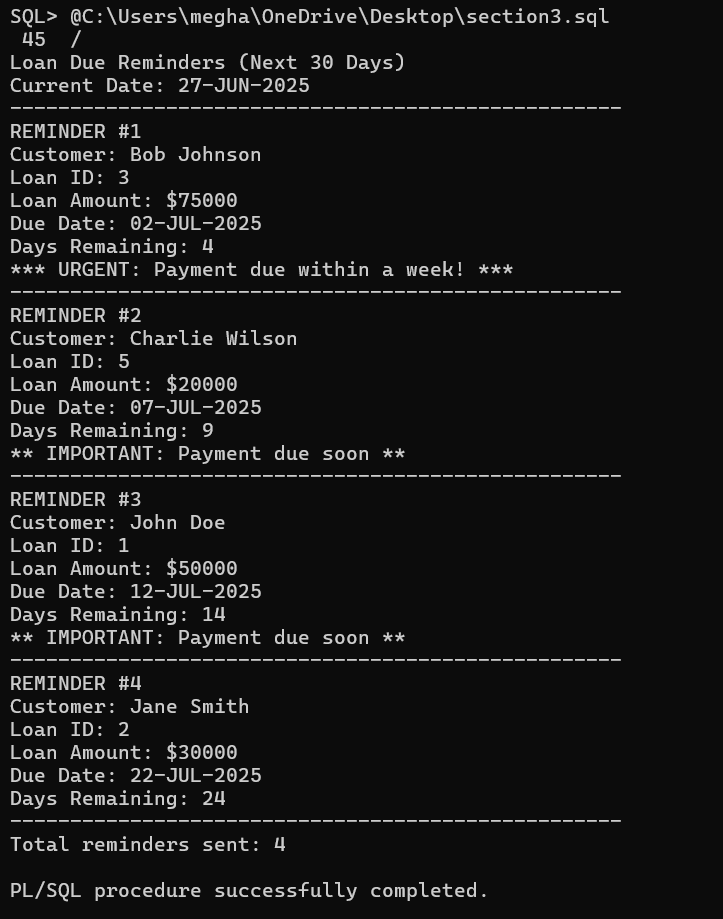
EXCEPTION

WHEN OTHERS THEN

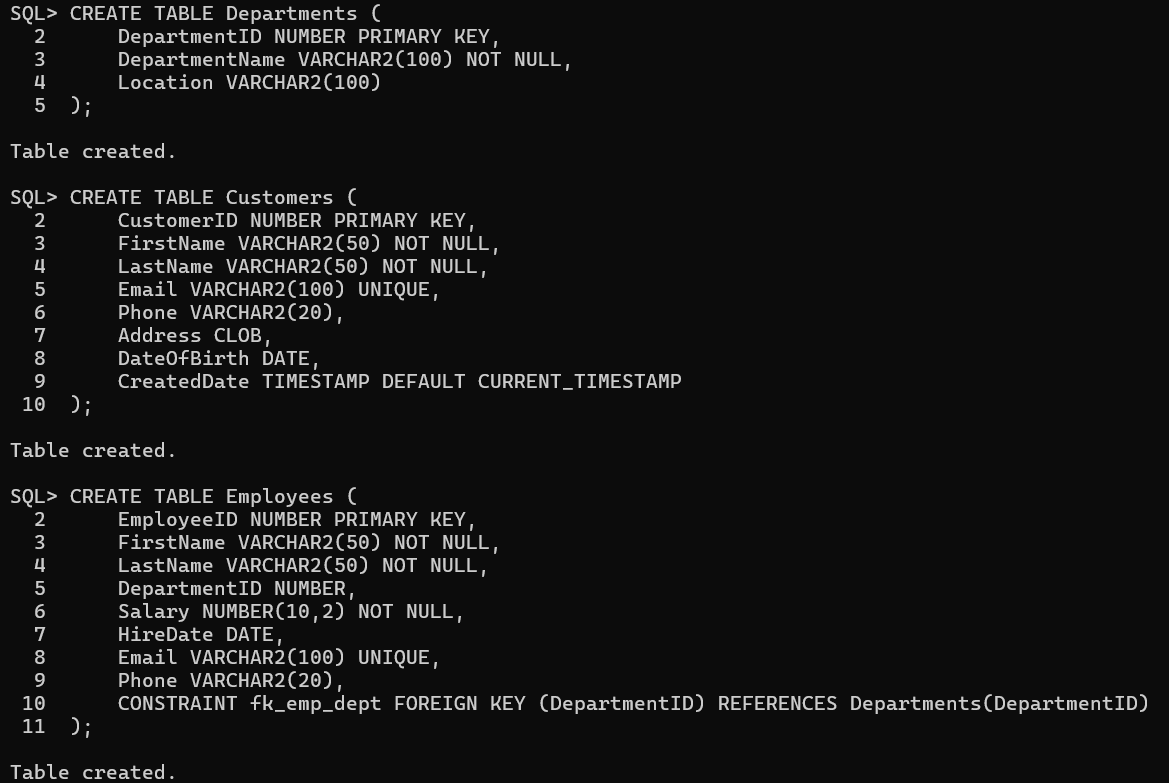
DBMS\_OUTPUT.PUT\_LINE('Error: ' || SQLERRM);

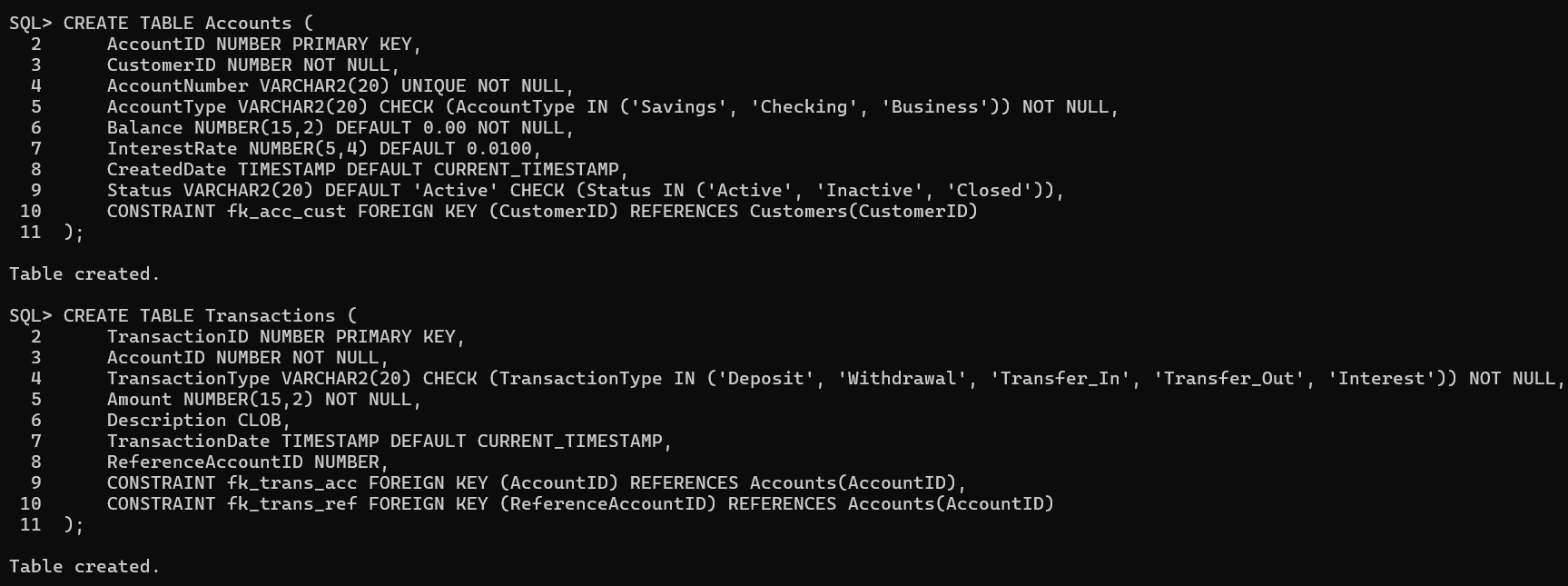
END;

**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:**

CREATE OR REPLACE PROCEDURE ProcessMonthlyInterest

IS

v\_accounts\_processed NUMBER := 0;

v\_interest\_amount NUMBER(15,2);

v\_new\_balance NUMBER(15,2);

CURSOR savings\_cursor IS

SELECT AccountID, Balance, InterestRate

FROM Accounts

WHERE AccountType = 'Savings' AND Status = 'Active';

BEGIN

FOR rec IN savings\_cursor LOOP

v\_interest\_amount := rec.Balance \* rec.InterestRate;

v\_new\_balance := rec.Balance + v\_interest\_amount;

UPDATE Accounts

SET Balance = v\_new\_balance

WHERE AccountID = rec.AccountID;

INSERT INTO Transactions (AccountID, TransactionType, Amount, Description)

VALUES (rec.AccountID, 'Interest', v\_interest\_amount,

'Monthly interest applied at ' || (rec.InterestRate \* 100) || '% rate');

v\_accounts\_processed := v\_accounts\_processed + 1;

END LOOP;

DBMS\_OUTPUT.PUT\_LINE('Monthly interest processed for ' || v\_accounts\_processed || ' savings accounts');

EXCEPTION

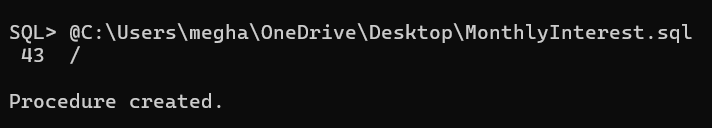
WHEN OTHERS THEN

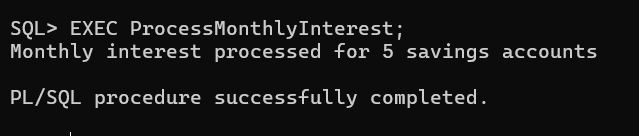
ROLLBACK;

RAISE\_APPLICATION\_ERROR(-20001, 'Error processing monthly interest: ' || SQLERRM);

END ProcessMonthlyInterest;

**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:**

CREATE OR REPLACE PROCEDURE UpdateEmployeeBonus(

p\_department\_id IN NUMBER,

p\_bonus\_percentage IN NUMBER

)

IS

v\_employees\_updated NUMBER := 0;

v\_department\_name VARCHAR2(100);

BEGIN

IF p\_department\_id IS NULL OR p\_bonus\_percentage IS NULL THEN

RAISE\_APPLICATION\_ERROR(-20002, 'Department ID and bonus percentage are required');

END IF;

IF p\_bonus\_percentage < 0 OR p\_bonus\_percentage > 100 THEN

RAISE\_APPLICATION\_ERROR(-20003, 'Bonus percentage must be between 0 and 100');

END IF;

-- Check if department exists

BEGIN

SELECT DepartmentName INTO v\_department\_name

FROM Departments

WHERE DepartmentID = p\_department\_id;

EXCEPTION

WHEN NO\_DATA\_FOUND THEN

RAISE\_APPLICATION\_ERROR(-20004, 'Department does not exist');

END;

-- Update employee salaries with bonus

UPDATE Employees

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

WHERE DepartmentID = p\_department\_id;

v\_employees\_updated := SQL%ROWCOUNT;

DBMS\_OUTPUT.PUT\_LINE('Department: ' || v\_department\_name);

DBMS\_OUTPUT.PUT\_LINE('Employees Updated: ' || v\_employees\_updated);

DBMS\_OUTPUT.PUT\_LINE('Bonus Percentage: ' || p\_bonus\_percentage || '%');

DBMS\_OUTPUT.PUT\_LINE('Bonus of ' || p\_bonus\_percentage || '% applied to ' ||

v\_employees\_updated || ' employees in ' || v\_department\_name);

EXCEPTION

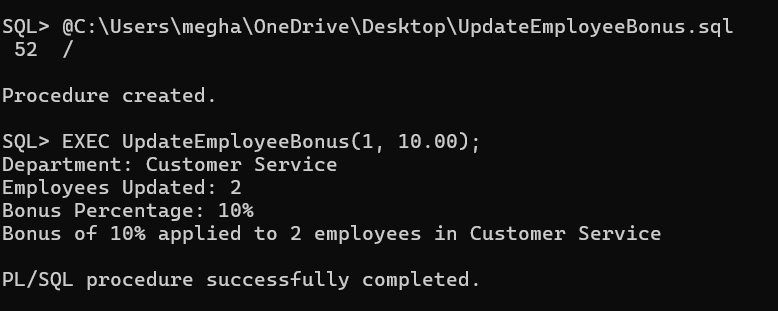
WHEN OTHERS THEN

ROLLBACK;

RAISE\_APPLICATION\_ERROR(-20005, 'Error updating employee bonus: ' || SQLERRM);

END UpdateEmployeeBonus;

**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:**

CREATE OR REPLACE PROCEDURE TransferFunds(

p\_from\_account\_id IN NUMBER,

p\_to\_account\_id IN NUMBER,

p\_amount IN NUMBER,

p\_description IN VARCHAR2 DEFAULT NULL

)

IS

v\_from\_balance NUMBER(15,2);

v\_to\_balance NUMBER(15,2);

v\_from\_account\_number VARCHAR2(20);

v\_to\_account\_number VARCHAR2(20);

v\_transfer\_description VARCHAR2(4000);

BEGIN

IF p\_from\_account\_id IS NULL OR p\_to\_account\_id IS NULL OR p\_amount IS NULL THEN

RAISE\_APPLICATION\_ERROR(-20006, 'From Account ID, To Account ID, and Amount are required');

END IF;

IF p\_amount <= 0 THEN

RAISE\_APPLICATION\_ERROR(-20007, 'Transfer amount must be greater than zero');

END IF;

IF p\_from\_account\_id = p\_to\_account\_id THEN

RAISE\_APPLICATION\_ERROR(-20008, 'Cannot transfer to the same account');

END IF;

BEGIN

SELECT Balance, AccountNumber INTO v\_from\_balance, v\_from\_account\_number

FROM Accounts

WHERE AccountID = p\_from\_account\_id AND Status = 'Active';

EXCEPTION

WHEN NO\_DATA\_FOUND THEN

RAISE\_APPLICATION\_ERROR(-20009, 'Source account does not exist or is not active');

END;

BEGIN

SELECT Balance, AccountNumber INTO v\_to\_balance, v\_to\_account\_number

FROM Accounts

WHERE AccountID = p\_to\_account\_id AND Status = 'Active';

EXCEPTION

WHEN NO\_DATA\_FOUND THEN

RAISE\_APPLICATION\_ERROR(-20010, 'Destination account does not exist or is not active');

END;

-- Check sufficient balance

IF v\_from\_balance < p\_amount THEN

RAISE\_APPLICATION\_ERROR(-20011, 'Insufficient balance in source account');

END IF;

IF p\_description IS NOT NULL THEN

v\_transfer\_description := p\_description;

ELSE

v\_transfer\_description := 'Transfer from ' || v\_from\_account\_number || ' to ' || v\_to\_account\_number;

END IF;

-- Debit from source account

UPDATE Accounts

SET Balance = Balance - p\_amount

WHERE AccountID = p\_from\_account\_id;

-- Credit to destination account

UPDATE Accounts

SET Balance = Balance + p\_amount

WHERE AccountID = p\_to\_account\_id;

-- Insert transaction records

INSERT INTO Transactions (AccountID, TransactionType, Amount, Description, ReferenceAccountID)

VALUES (p\_from\_account\_id, 'Transfer\_Out', p\_amount, v\_transfer\_description, p\_to\_account\_id);

INSERT INTO Transactions (AccountID, TransactionType, Amount, Description, ReferenceAccountID)

VALUES (p\_to\_account\_id, 'Transfer\_In', p\_amount, v\_transfer\_description, p\_from\_account\_id);

-- Commit the transaction

COMMIT;

DBMS\_OUTPUT.PUT\_LINE('Transfer Successful');

DBMS\_OUTPUT.PUT\_LINE('From Account: ' || v\_from\_account\_number);

DBMS\_OUTPUT.PUT\_LINE('To Account: ' || v\_to\_account\_number);

DBMS\_OUTPUT.PUT\_LINE('Amount Transferred: ' || p\_amount);

DBMS\_OUTPUT.PUT\_LINE('Description: ' || v\_transfer\_description);

DBMS\_OUTPUT.PUT\_LINE('Transfer Date: ' || TO\_CHAR(SYSDATE, 'DD-MON-YYYY HH24:MI:SS'));

EXCEPTION

WHEN OTHERS THEN

ROLLBACK;

RAISE\_APPLICATION\_ERROR(-20012, 'Error during funds transfer: ' || SQLERRM);

END TransferFunds;

**Output:**

