WEEK-2

PL/SQL PROGRAMMING

Exercise 1: Control Structures

Scenario 1: 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.

File name:senior\_discount.sql

DECLARE

CURSOR senior\_customers IS

SELECT CustomerID

FROM Customers

WHERE MONTHS\_BETWEEN(SYSDATE, DOB) / 12 > 60;

BEGIN

FOR cust\_rec IN senior\_customers LOOP

UPDATE Loans

SET InterestRate = InterestRate - 1

WHERE CustomerID = cust\_rec.CustomerID;

UPDATE Customers

SET LastModified = SYSDATE

WHERE CustomerID = cust\_rec.CustomerID;

END LOOP;

COMMIT;

END;

Rows Inserted:

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

VALUES (3, 'Senior Citizen', TO\_DATE('1950-01-01', 'YYYY-MM-DD'), 2000, SYSDATE);

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

VALUES (2, 3, 10000, 6.5, SYSDATE, ADD\_MONTHS(SYSDATE, 60));

Table Name: Customers

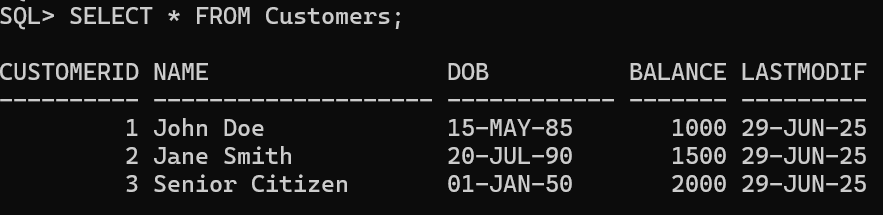
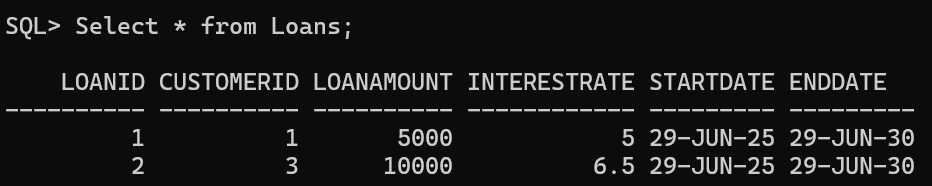
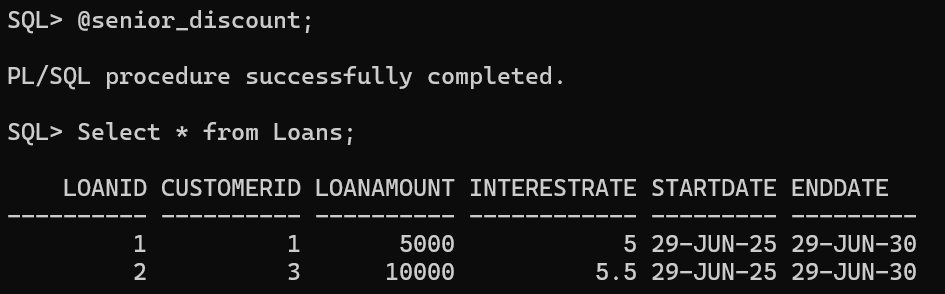


Table Name: Loans



Output:



Scenario 2: 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.

File name:update\_vip\_status.sql;

ALTER TABLE Customers ADD IsVIP CHAR(1) DEFAULT 'N';

DECLARE

customer\_count NUMBER := 0;

vip\_count NUMBER := 0;

BEGIN

FOR customer IN (SELECT CustomerID, Name, Balance FROM Customers) LOOP

customer\_count := customer\_count + 1;

IF customer.Balance > 10000 THEN

UPDATE Customers

SET IsVIP = 'Y', LastModified = SYSDATE

WHERE CustomerID = customer.CustomerID;

vip\_count := vip\_count + 1;

DBMS\_OUTPUT.PUT\_LINE(customer.Name || ' promoted to VIP! Balance: $' || customer.Balance);

ELSE

UPDATE Customers

SET IsVIP = 'N', LastModified = SYSDATE

WHERE CustomerID = customer.CustomerID;

END IF;

END LOOP

COMMIT;

DBMS\_OUTPUT.PUT\_LINE('Total customers: ' || customer\_count);

DBMS\_OUTPUT.PUT\_LINE('VIP customers: ' || vip\_count);

EXCEPTION

WHEN OTHERS THEN

ROLLBACK;

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

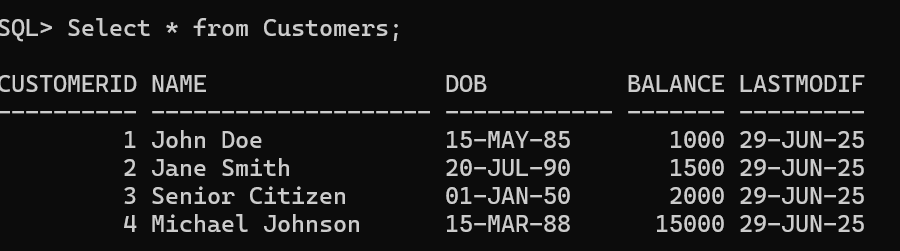
END;

Row Inserted:

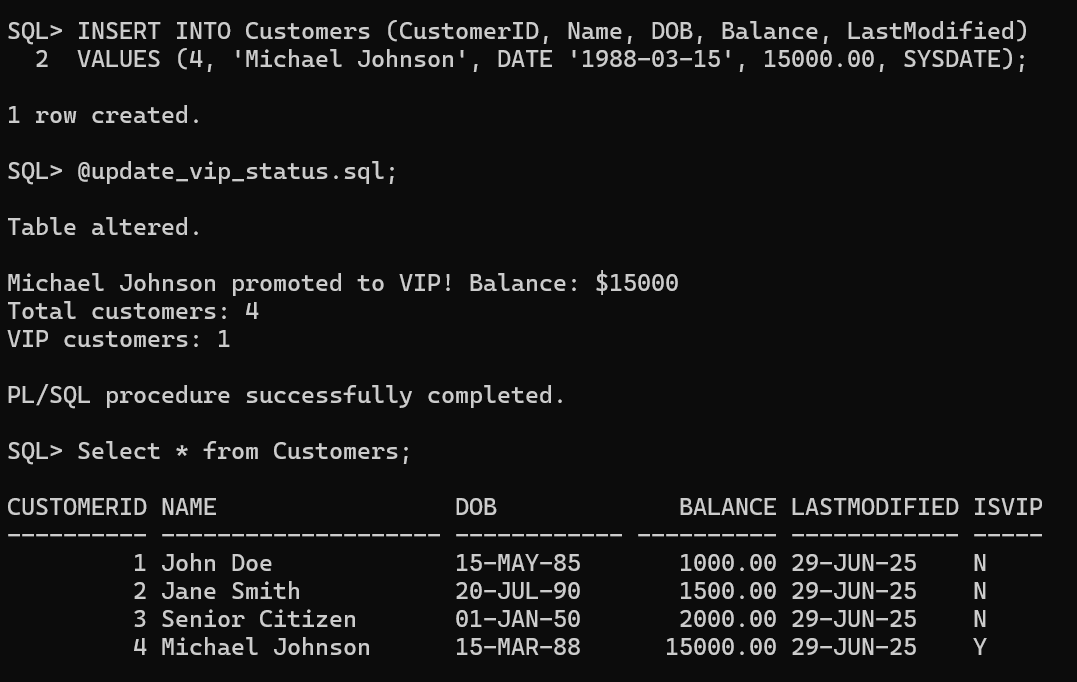
INSERT INTO Customers (CustomerID, Name, DOB, Balance, LastModified)

VALUES (4, 'Michael Johnson', DATE '1988-03-15', 15000.00, SYSDATE);

Table name: Customers;



Output:



Scenario 3: Write a PL/SQL block that fetches all loans due in the next 30 days and prints a reminder message for each customer.

File name:loan\_remiders.sql

DECLARE

reminder\_count NUMBER := 0;

BEGIN

FOR loan\_reminder IN (

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

FROM Customers c

JOIN Loans l ON c.CustomerID = l.CustomerID

WHERE l.EndDate BETWEEN SYSDATE AND SYSDATE + 30

ORDER BY l.EndDate

) LOOP

reminder\_count := reminder\_count + 1;

DBMS\_OUTPUT.PUT\_LINE('Dear ' || loan\_reminder.Name || ',');

DBMS\_OUTPUT.PUT\_LINE('Your loan of $' || loan\_reminder.LoanAmount ||

' is due on ' || TO\_CHAR(loan\_reminder.EndDate, 'DD-MON-YYYY'));

DBMS\_OUTPUT.PUT\_LINE('Please make arrangements for payment.');

DBMS\_OUTPUT.PUT\_LINE('Loan ID: ' || loan\_reminder.LoanID);

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

END LOOP;

IF reminder\_count = 0 THEN

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

ELSE

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

END IF

EXCEPTION

WHEN OTHERS THEN

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

END;

/

Rows Inserted:

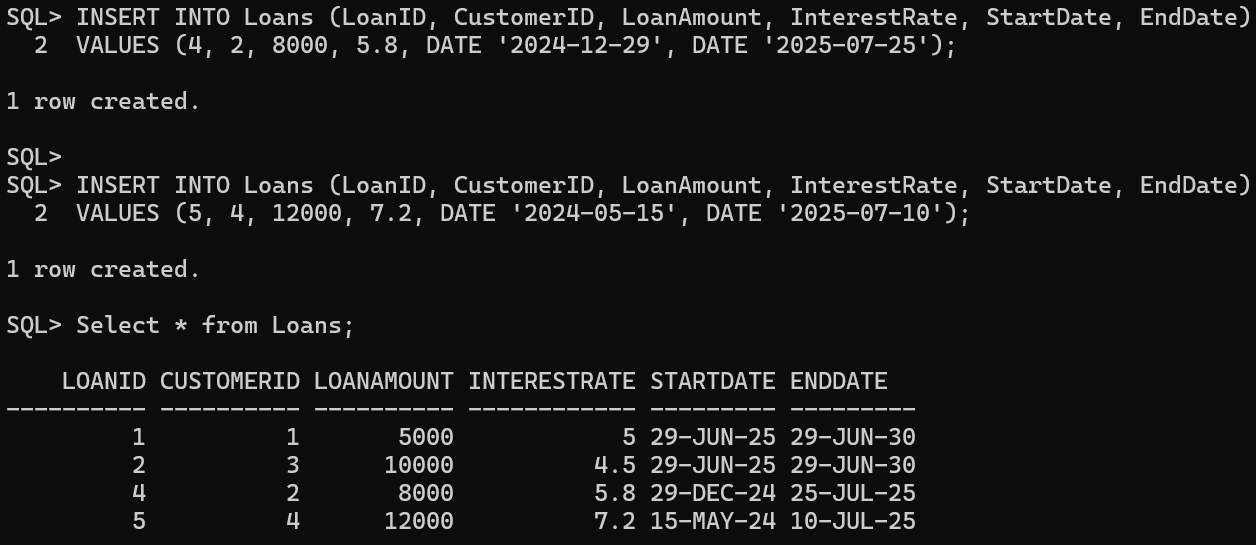
INSERT INTO Loans (LoanID, CustomerID, LoanAmount, InterestRate, StartDate, EndDate)

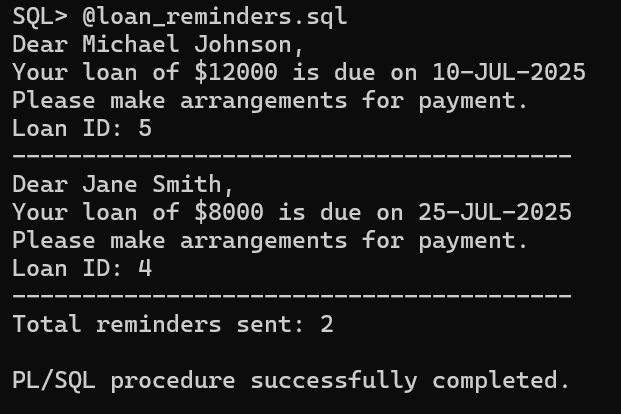
VALUES (4, 2, 8000, 5.8, DATE '2024-12-29', DATE '2025-07-25');

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

VALUES (5, 4, 12000, 7.2, DATE '2024-05-15', DATE '2025-07-10');

Output:





Exercise 3: Stored Procedures

Scenario 1: 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.

File name:ProcessMonthlyInterest.sql

CREATE OR REPLACE PROCEDURE ProcessMonthlyInterest AS

accounts\_updated NUMBER := 0;

total\_interest\_paid NUMBER := 0;

interest\_earned NUMBER;

BEGIN

FOR account IN (

SELECT a.AccountID, a.CustomerID, a.Balance, c.Name

FROM Accounts a

JOIN Customers c ON a.CustomerID = c.CustomerID

WHERE UPPER(a.AccountType) = 'SAVINGS'

AND a.Balance > 0

) LOOP

interest\_earned := account.Balance \* 0.01;

UPDATE Accounts

SET Balance = Balance + interest\_earned,

LastModified = SYSDATE

WHERE AccountID = account.AccountID;

accounts\_updated := accounts\_updated + 1;

total\_interest\_paid := total\_interest\_paid + interest\_earned;

DBMS\_OUTPUT.PUT\_LINE(account.Name || ' earned $' ||

ROUND(interest\_earned, 2) ||

' on account ' || account.AccountID);

END LOOP;

COMMIT;

DBMS\_OUTPUT.PUT\_LINE('Interest payment complete!');

DBMS\_OUTPUT.PUT\_LINE('Updated ' || accounts\_updated || ' savings accounts');

DBMS\_OUTPUT.PUT\_LINE('Total interest distributed: $' || ROUND(total\_interest\_paid, 2));

EXCEPTION

WHEN OTHERS THEN

ROLLBACK;

DBMS\_OUTPUT.PUT\_LINE('Something went wrong: ' || SQLERRM);

END ProcessMonthlyInterest;

Rows inserted:

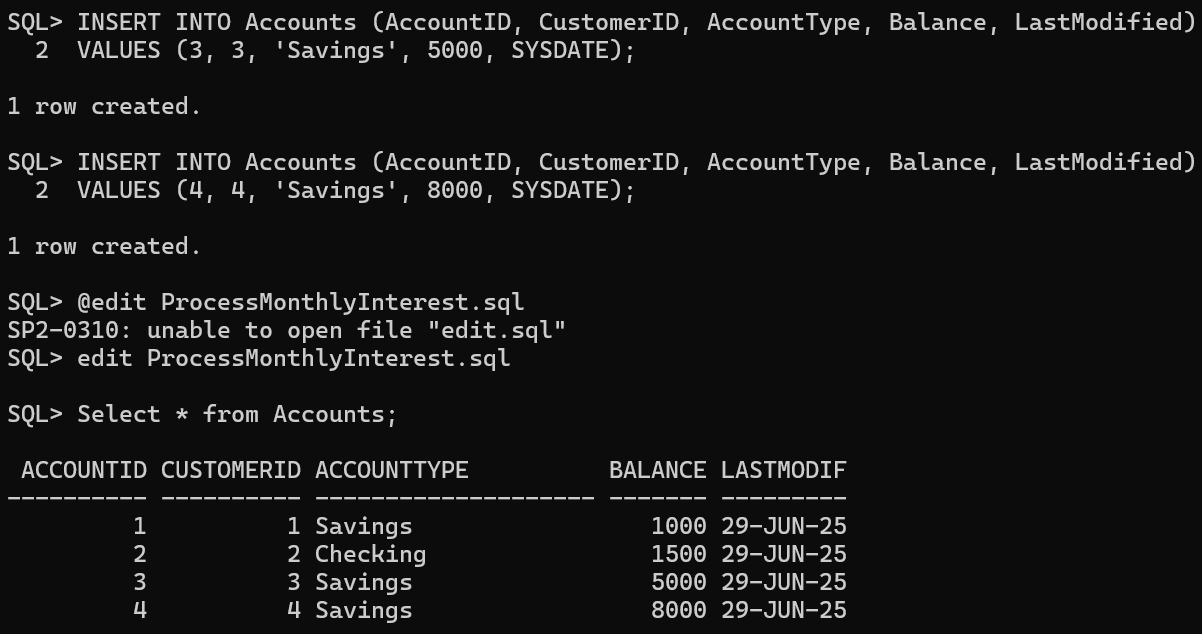
INSERT INTO Accounts (AccountID, CustomerID, AccountType, Balance, LastModified)

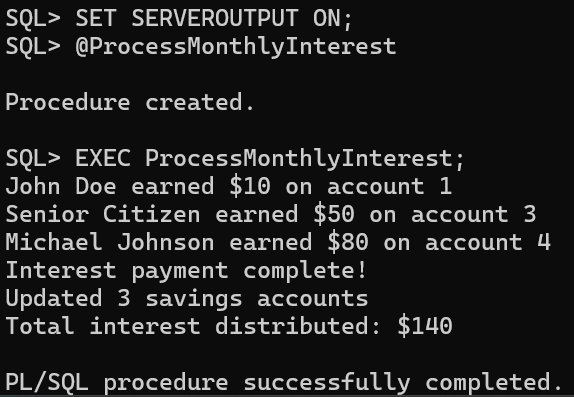
VALUES (3, 3, 'Savings', 5000, SYSDATE);

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

VALUES (4, 4, 'Savings', 8000, SYSDATE);

Output:





Scenario 2: Write a stored procedure UpdateEmployeeBonus that updates the salary of employees in a given department by adding a bonus percentage passed as a parameter.

File name: update\_employee\_bonus.sql

CREATE OR REPLACE PROCEDURE UpdateEmployeeBonus (

department\_name IN VARCHAR2,

bonus\_percentage IN NUMBER

)

IS

total\_employees\_updated NUMBER := 0;

BEGIN

UPDATE Employees

SET Salary = Salary + (Salary \* bonus\_percentage / 100)

WHERE UPPER(Department) = UPPER(department\_name);

total\_employees\_updated := SQL%ROWCOUNT;

IF total\_employees\_updated > 0 THEN

COMMIT;

DBMS\_OUTPUT.PUT\_LINE('Great news! We successfully gave bonuses to ' || total\_employees\_updated || ' amazing employees in the ' || department\_name || ' team');

DBMS\_OUTPUT.PUT\_LINE('Each person received a well-deserved ' || bonus\_percentage || '% salary boost!');

ELSE

DBMS\_OUTPUT.PUT\_LINE('Hmm, we could not find any employees in the ' || department\_name || ' department. Please double-check the department name.');

END IF;

EXCEPTION

WHEN OTHERS THEN

ROLLBACK;

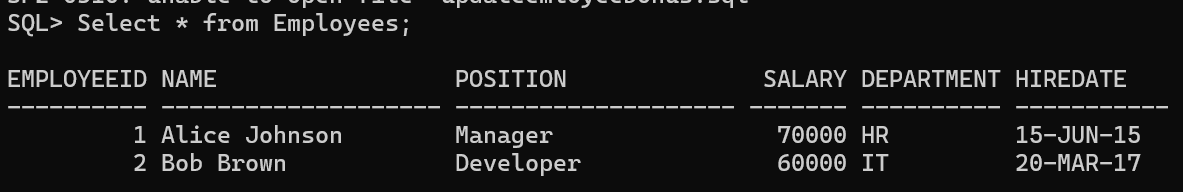
DBMS\_OUTPUT.PUT\_LINE('Oops! We ran into an issue while processing the bonuses. Error details: ' || SQLERRM);

RAISE;

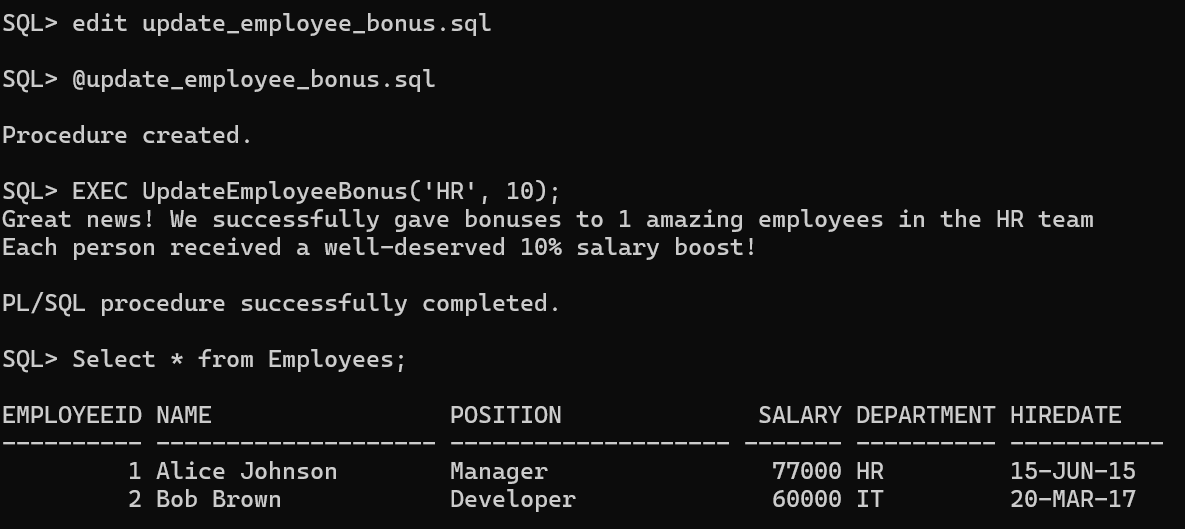
END UpdateEmployeeBonus;

/

Table name: Employees



Output:



Scenario 3: 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.

File name: Transfer\_Funds.sql

CREATE OR REPLACE PROCEDURE TransferFunds (

source\_account\_id IN NUMBER,

destination\_account\_id IN NUMBER,

transfer\_amount IN NUMBER

)

IS

source\_balance NUMBER;

dest\_account\_exists NUMBER;

next\_transaction\_id NUMBER;

BEGIN

IF transfer\_amount <= 0 THEN

DBMS\_OUTPUT.PUT\_LINE('Oops! The transfer amount must be greater than zero. Please enter a positive amount.');

RETURN;

END IF;

SELECT Balance INTO source\_balance

FROM Accounts

WHERE AccountID = source\_account\_id;

SELECT COUNT(\*) INTO dest\_account\_exists

FROM Accounts

WHERE AccountID = destination\_account\_id;

IF dest\_account\_exists = 0 THEN

DBMS\_OUTPUT.PUT\_LINE('Sorry! We could not find the destination account. Please check the account number and try again.');

RETURN;

END IF;

IF source\_balance < transfer\_amount THEN

DBMS\_OUTPUT.PUT\_LINE('Insufficient funds! Your current balance is ' || source\_balance || ' but you are trying to transfer ' || transfer\_amount || '.');

RETURN;

END IF;

UPDATE Accounts

SET Balance = Balance - transfer\_amount,

LastModified = SYSDATE

WHERE AccountID = source\_account\_id;

UPDATE Accounts

SET Balance = Balance + transfer\_amount,

LastModified = SYSDATE

WHERE AccountID = destination\_account\_id;

SELECT NVL(MAX(TransactionID), 0) + 1 INTO next\_transaction\_id FROM Transactions;

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

VALUES (next\_transaction\_id, source\_account\_id, SYSDATE, transfer\_amount, 'DEBIT');

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

VALUES (next\_transaction\_id + 1, destination\_account\_id, SYSDATE, transfer\_amount, 'CREDIT');

COMMIT;

DBMS\_OUTPUT.PUT\_LINE('Fantastic! Your transfer of ' || transfer\_amount || ' has been completed successfully!');

DBMS\_OUTPUT.PUT\_LINE('Money moved from Account ' || source\_account\_id || ' to Account ' || destination\_account\_id || '.');

EXCEPTION

WHEN NO\_DATA\_FOUND THEN

ROLLBACK;

DBMS\_OUTPUT.PUT\_LINE('Sorry! We could not find the source account. Please verify the account number.');

WHEN OTHERS THEN

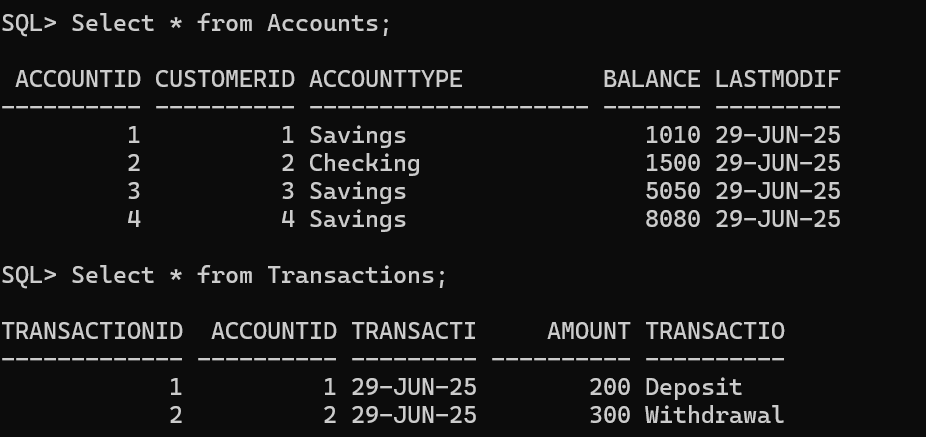
ROLLBACK;

DBMS\_OUTPUT.PUT\_LINE('Something unexpected happened during the transfer. Error: ' || SQLERRM);

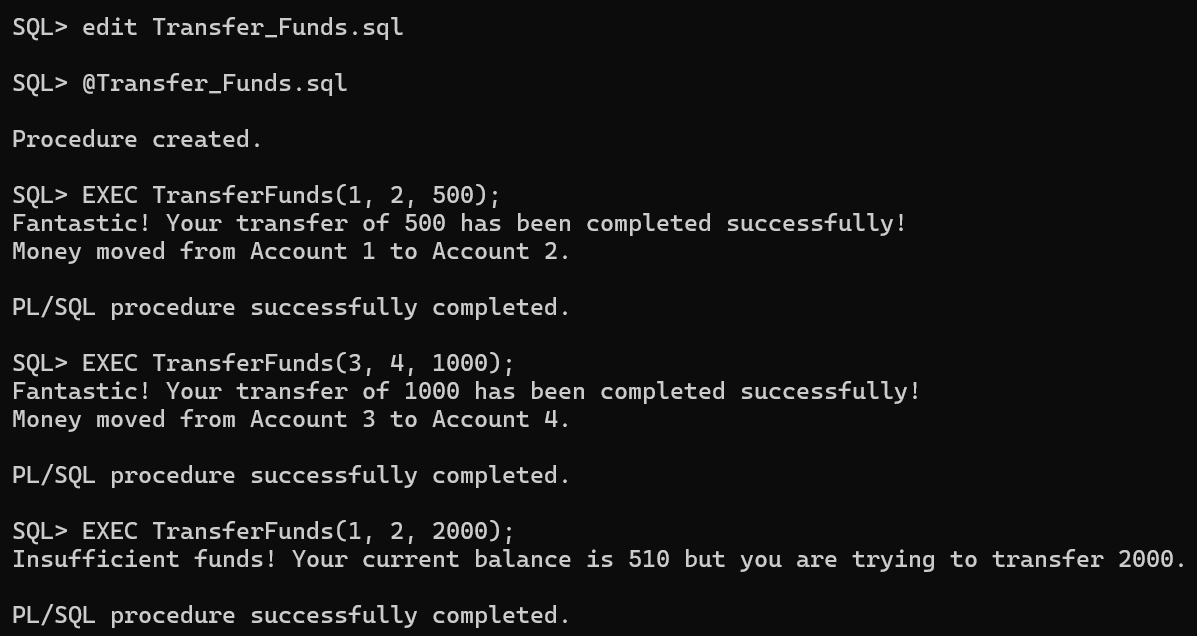
RAISE;

END TransferFunds;/

Table names: Accounts,Transactions



Output:



After transactions the tables are:

