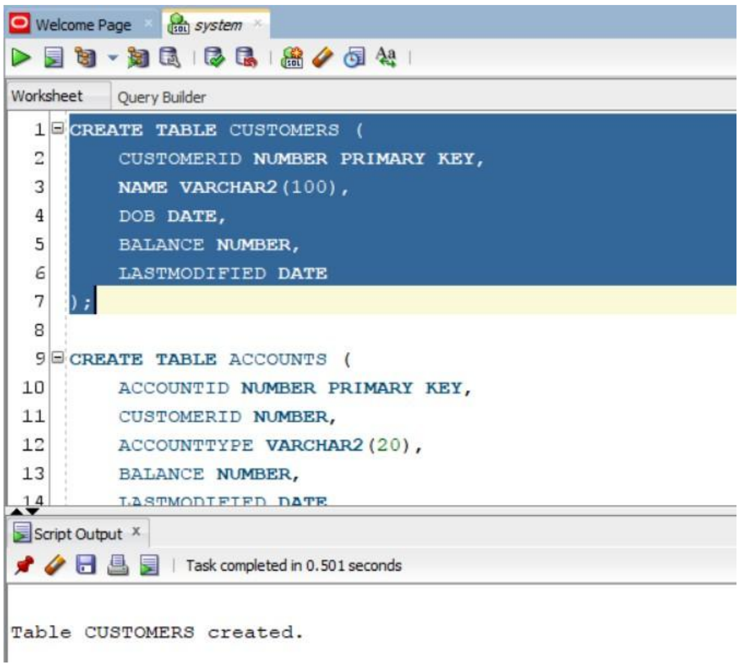
PL/SQL

Schema Creation

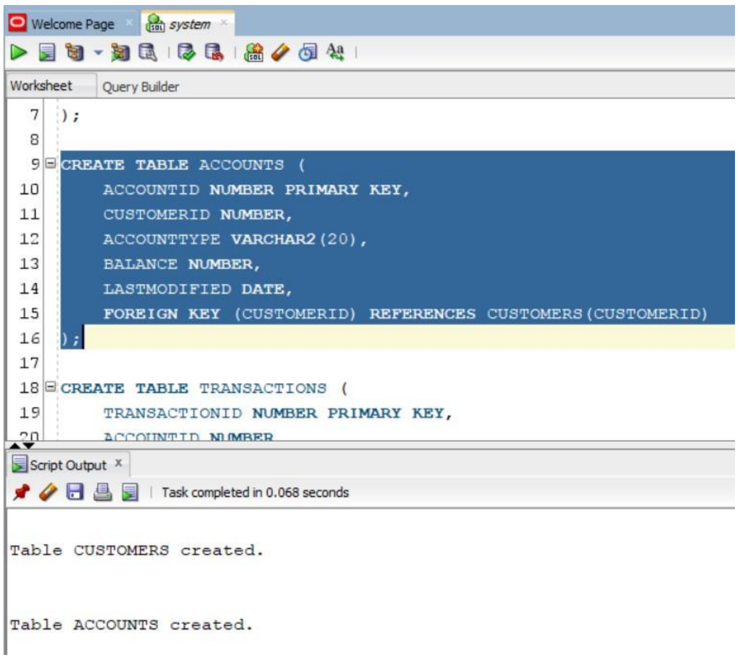
```
CREATE TABLE CUSTOMERS (
    CUSTOMERID   NUMBER PRIMARY KEY,
    NAME         VARCHAR2(100),
    DOB          DATE,
    BALANCE      NUMBER,
    LASTMODIFIED DATE
);
```



```
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
```

```
Table CUSTOMERS created.
```

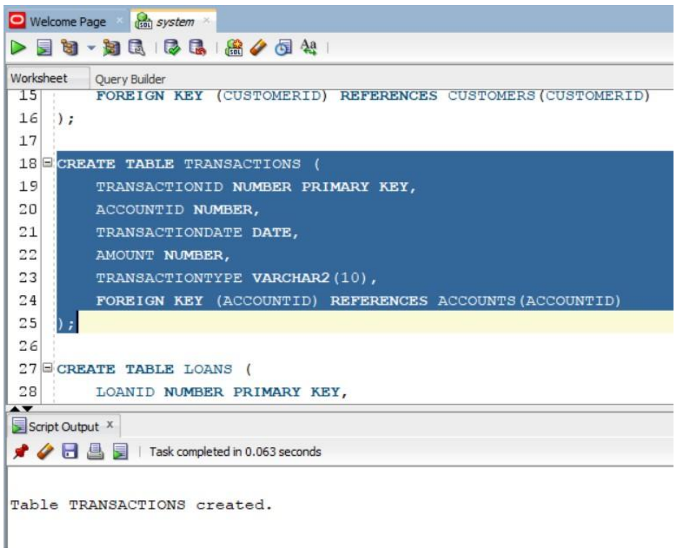
```
CREATE TABLE ACCOUNTS (
    ACCOUNTID    NUMBER PRIMARY KEY,
    CUSTOMERID   NUMBER,
    ACCOUNTTYPE  VARCHAR2(20),
    BALANCE      NUMBER,
    LASTMODIFIED DATE,
    FOREIGN KEY ( CUSTOMERID )
        REFERENCES CUSTOMERS ( CUSTOMERID )
);
```



```
);

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
```

```
Table CUSTOMERS created.


Table ACCOUNTS created.
```

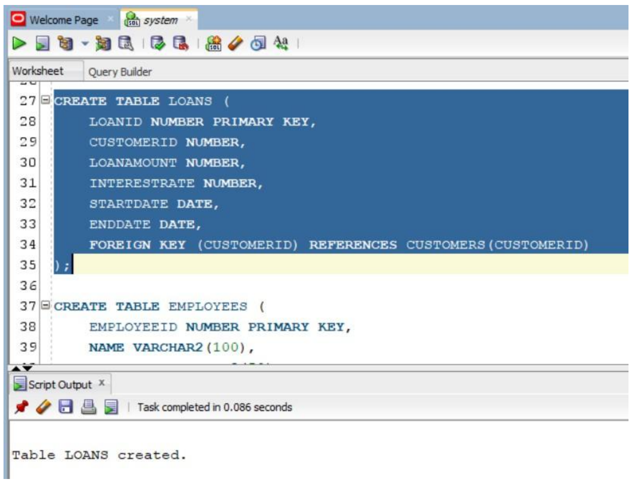
```
CREATE TABLE TRANSACTIONS (
    TRANSACTIONID   NUMBER PRIMARY KEY,
    ACCOUNTID       NUMBER,
    TRANSACTIONDATE DATE,
    AMOUNT          NUMBER,
    TRANSACTIONTYPE VARCHAR2(10),
    FOREIGN KEY ( ACCOUNTID )
        REFERENCES ACCOUNTS ( ACCOUNTID )
);
```



```
    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,
```

```
Table TRANSACTIONS created.
```

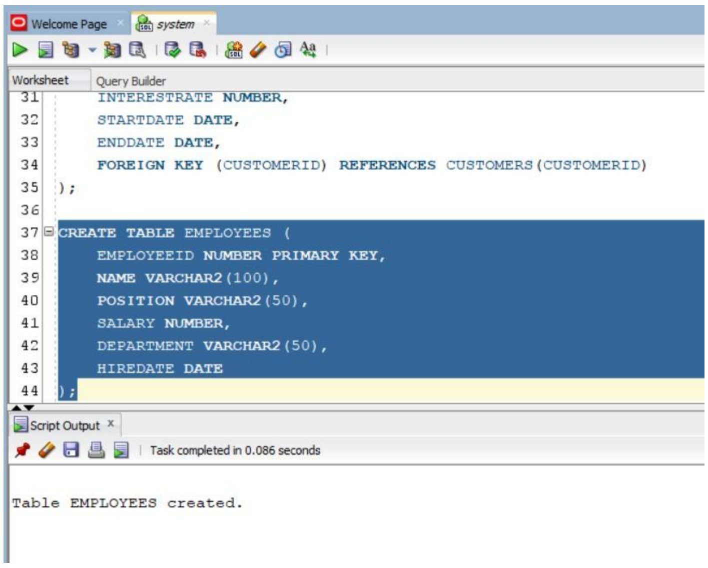
```
CREATE TABLE LOANS (
   LOANID       NUMBER PRIMARY KEY,
   CUSTOMERID   NUMBER,
   LOANAMOUNT   NUMBER,
   INTERESTRATE NUMBER,
   STARTDATE    DATE,
   ENDDATE      DATE,
   FOREIGN KEY ( CUSTOMERID )
      REFERENCES CUSTOMERS ( CUSTOMERID )
);
```



```
CREATE TABLE 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),
```

```
Table LOANS created.
```

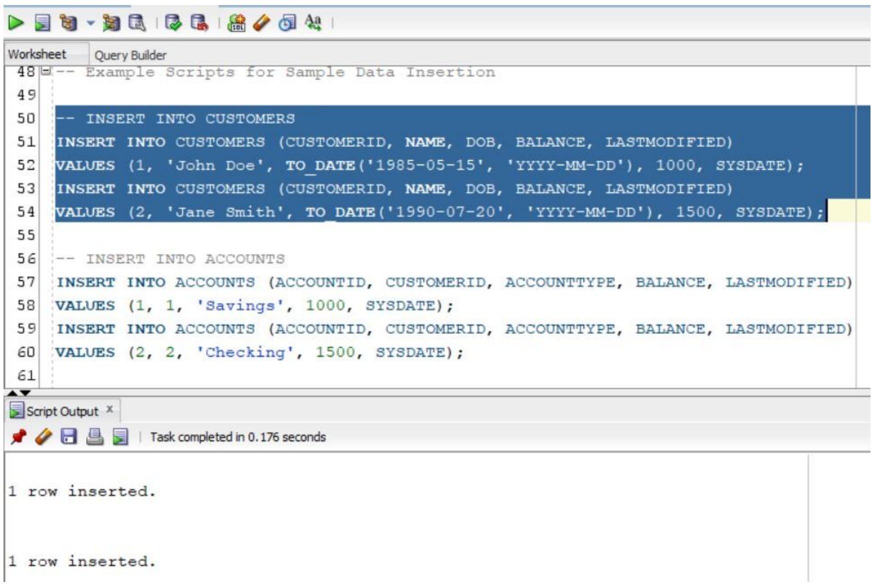
```
CREATE TABLE EMPLOYEES (
    EMPLOYEEID NUMBER PRIMARY KEY,
    NAME       VARCHAR2(100),
    POSITION   VARCHAR2(50),
    SALARY     NUMBER,
    DEPARTMENT VARCHAR2(50),
    HIREDATE   DATE
);
```



```
    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
);
```

```
Table EMPLOYEES created.
```

```
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);
```



```
-- Example Scripts for Sample Data Insertion

-- INSERT INTO CUSTOMERS
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
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);
```

```
1 row inserted.


1 row inserted.
```

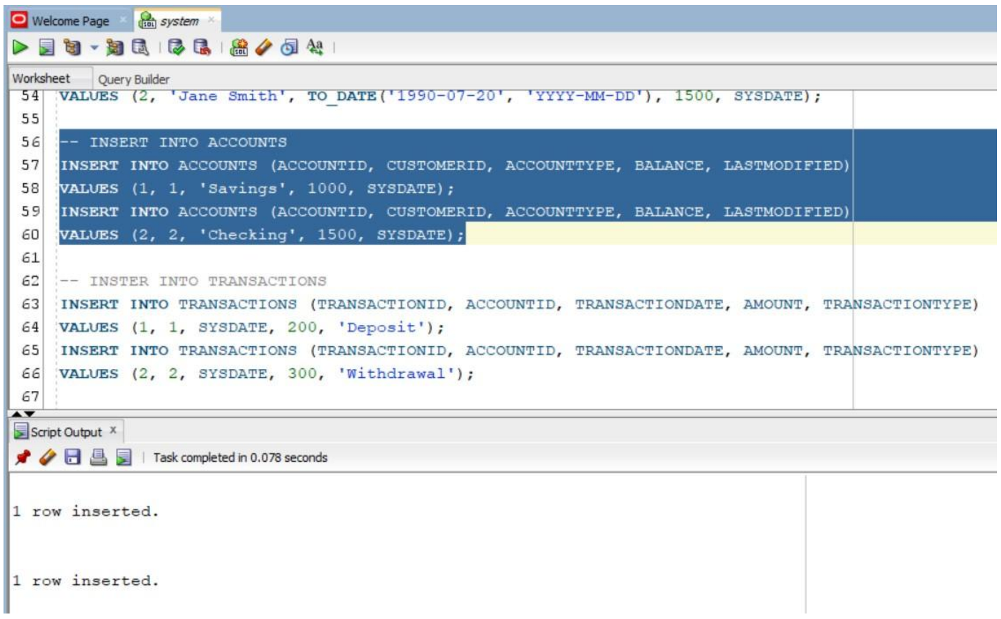
-- INSERT INTO ACCOUNTS

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);



```
VALUES (2, 'Jane Smith', TO_DATE('1990-07-20', 'YYYY-MM-DD'), 1500, SYSDATE);

-- INSERT INTO ACCOUNTS
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);

-- INSTER INTO TRANSACTIONS
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');
```

```
1 row inserted.


1 row inserted.
```

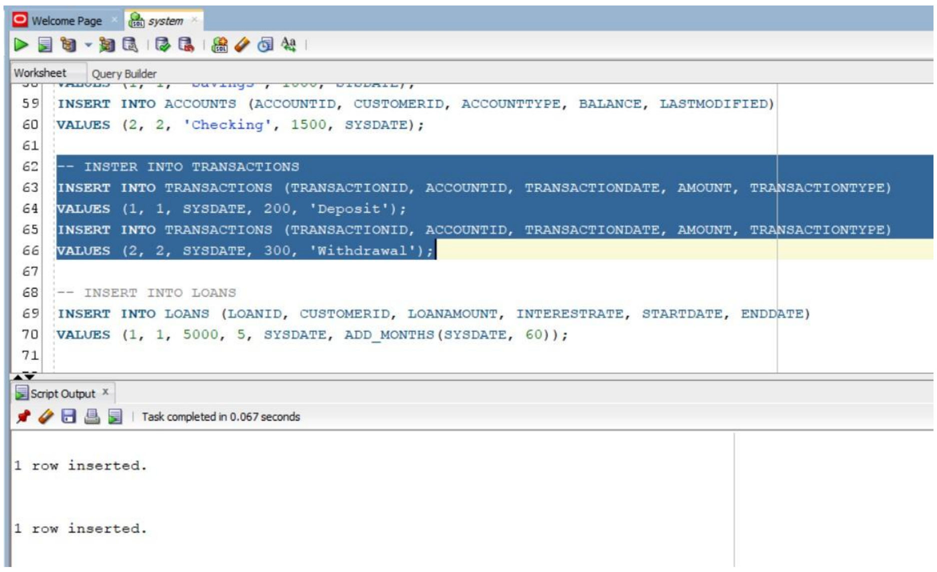
-- INSTER INTO TRANSACTIONS

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');



```
VALUES (1, 1, 'Savings', 1000, SYSDATE);
INSERT INTO ACCOUNTS (ACCOUNTID, CUSTOMERID, ACCOUNTTYPE, BALANCE, LASTMODIFIED)
VALUES (2, 2, 'Checking', 1500, SYSDATE);

-- INSTER INTO TRANSACTIONS
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
INSERT INTO LOANS (LOANID, CUSTOMERID, LOANAMOUNT, INTERESTRATE, STARTDATE, ENDDATE)
VALUES (1, 1, 5000, 5, SYSDATE, ADD_MONTHS(SYSDATE, 60));
```

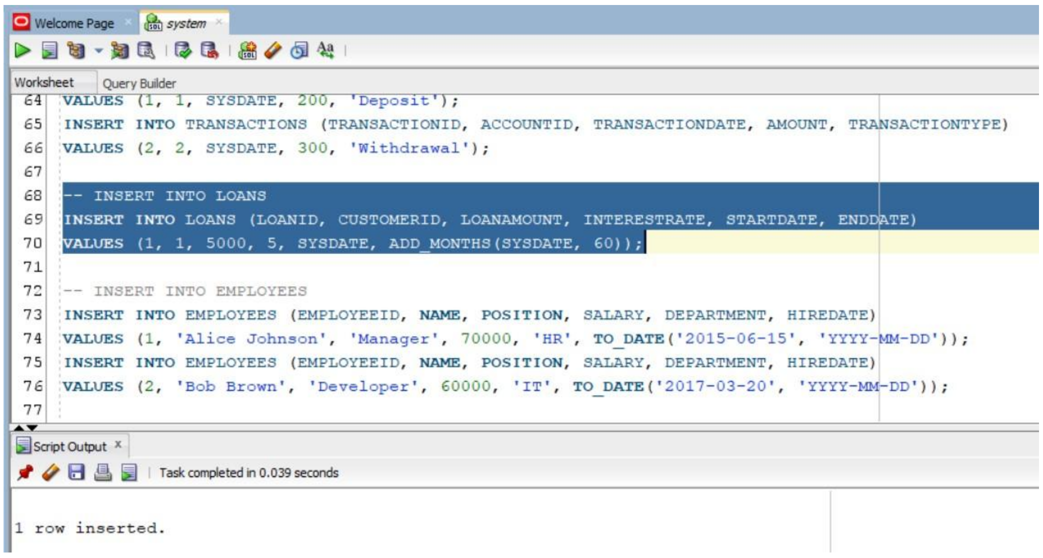
Task completed in 0.067 seconds

```
1 row inserted.


1 row inserted.
```

-- INSERT INTO LOANS

INSERT INTO LOANS (LOANID, CUSTOMERID, LOANAMOUNT, INTERESTRATE, STARTDATE, ENDDATE)

VALUES (1, 1, 5000, 5, SYSDATE, ADD_MONTHS(SYSDATE, 60));



```
VALUES (1, 1, SYSDATE, 200, 'Deposit');
INSERT INTO TRANSACTIONS (TRANSACTIONID, ACCOUNTID, TRANSACTIONDATE, AMOUNT, TRANSACTIONTYPE)
VALUES (2, 2, SYSDATE, 300, 'Withdrawal');

-- INSERT INTO LOANS
INSERT INTO LOANS (LOANID, CUSTOMERID, LOANAMOUNT, INTERESTRATE, STARTDATE, ENDDATE)
VALUES (1, 1, 5000, 5, SYSDATE, ADD_MONTHS(SYSDATE, 60));

-- INSERT INTO EMPLOYEES
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'));
```

Task completed in 0.039 seconds

```
1 row inserted.
```

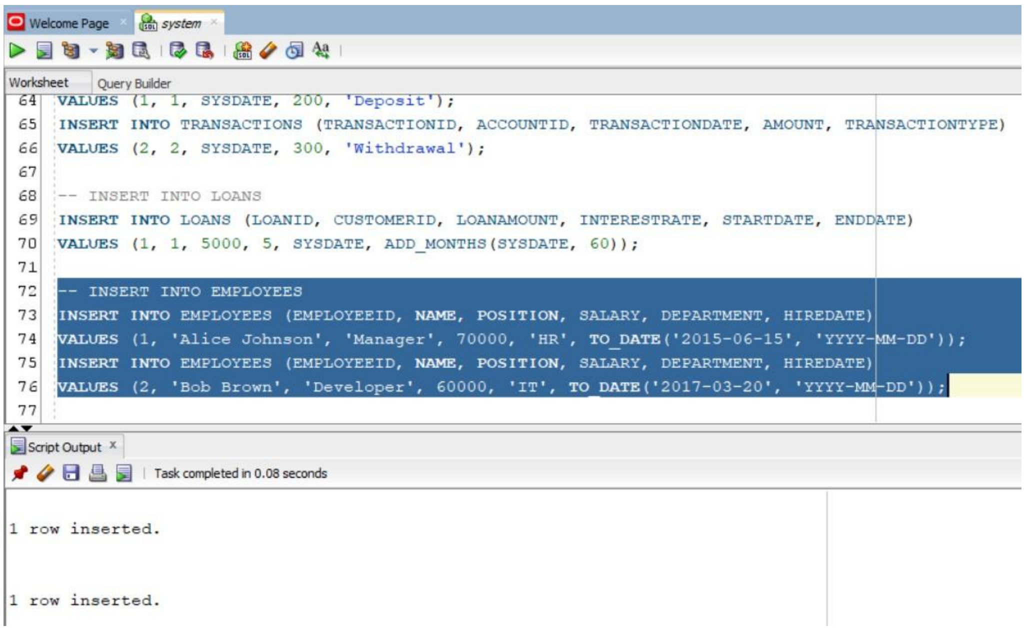
-- INSERT INTO EMPLOYEES

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'));



```
VALUES (1, 1, SYSDATE, 200, 'Deposit');
INSERT INTO TRANSACTIONS (TRANSACTIONID, ACCOUNTID, TRANSACTIONDATE, AMOUNT, TRANSACTIONTYPE)
VALUES (2, 2, SYSDATE, 300, 'Withdrawal');

-- INSERT INTO LOANS
INSERT INTO LOANS (LOANID, CUSTOMERID, LOANAMOUNT, INTERESTRATE, STARTDATE, ENDDATE)
VALUES (1, 1, 5000, 5, SYSDATE, ADD_MONTHS(SYSDATE, 60));

-- INSERT INTO EMPLOYEES
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'));
```

```
1 row inserted.


1 row inserted.
```

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.

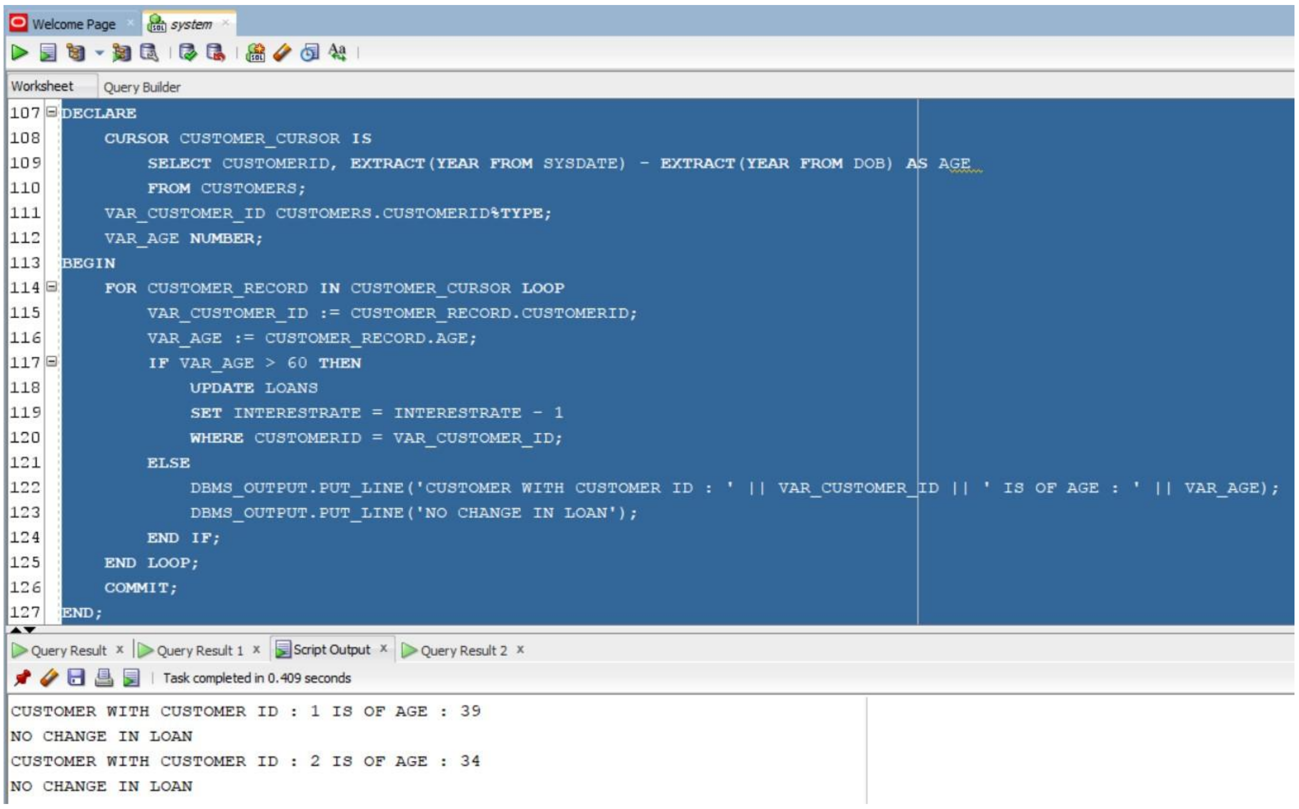
- **Question:** Write a PL/SQL block that fetches all loans due in the next 30 days and prints a reminder message for each customer.

-- SCENARIO 1

SELECT * FROM CUSTOMERS;

SELECT * FROM LOANS;

```
SET SERVEROUTPUT ON;
DECLARE
   CURSOR CUSTOMER_CURSOR IS
      SELECT CUSTOMERID, EXTRACT(YEAR FROM SYSDATE) - EXTRACT(YEAR FROM DOB) AS AGE
      FROM CUSTOMERS;
   VAR_CUSTOMER_ID CUSTOMERS.CUSTOMERID%TYPE;
   VAR_AGE NUMBER;
BEGIN
   FOR CUSTOMER_RECORD IN CUSTOMER_CURSOR LOOP
      VAR_CUSTOMER_ID := CUSTOMER_RECORD.CUSTOMERID;
      VAR_AGE := CUSTOMER_RECORD.AGE;
      IF VAR_AGE > 60 THEN
         UPDATE LOANS
         SET INTERESTRATE = INTERESTRATE - 1
         WHERE CUSTOMERID = VAR_CUSTOMER_ID;
      ELSE
         DBMS_OUTPUT.PUT_LINE('CUSTOMER WITH CUSTOMER ID : ' || VAR_CUSTOMER_ID || ' IS OF AGE :
' || VAR_AGE);
         DBMS_OUTPUT.PUT_LINE('NO CHANGE IN LOAN');
      END IF;
   END LOOP;
   COMMIT;
END;
/

SELECT * FROM LOANS;
```



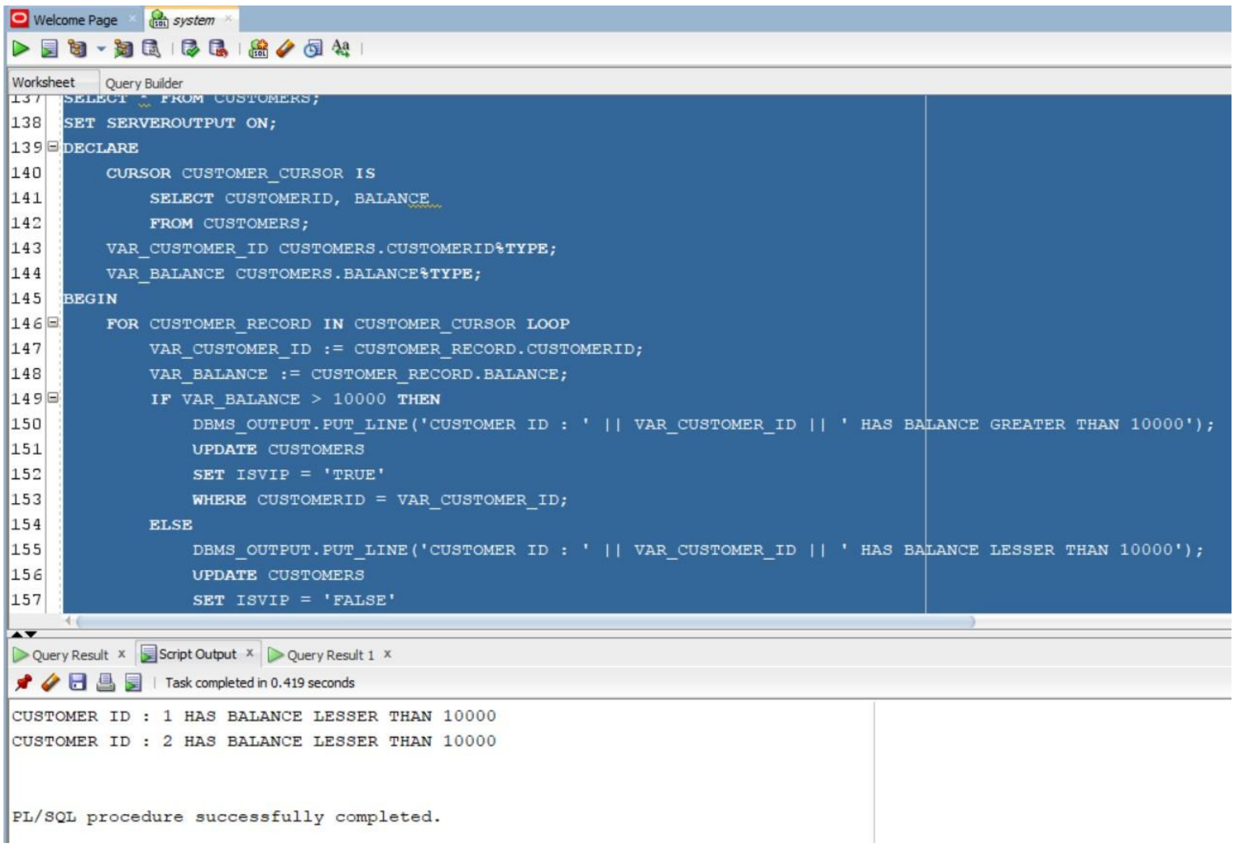
```
DECLARE
    CURSOR CUSTOMER_CURSOR IS
        SELECT CUSTOMERID, EXTRACT(YEAR FROM SYSDATE) - EXTRACT(YEAR FROM DOB) AS AGE
        FROM CUSTOMERS;
    VAR_CUSTOMER_ID CUSTOMERS.CUSTOMERID%TYPE;
    VAR_AGE NUMBER;
BEGIN
    FOR CUSTOMER_RECORD IN CUSTOMER_CURSOR LOOP
        VAR_CUSTOMER_ID := CUSTOMER_RECORD.CUSTOMERID;
        VAR_AGE := CUSTOMER_RECORD.AGE;
        IF VAR_AGE > 60 THEN
            UPDATE LOANS
            SET INTERESTRATE = INTERESTRATE - 1
            WHERE CUSTOMERID = VAR_CUSTOMER_ID;
        ELSE
            DBMS_OUTPUT.PUT_LINE('CUSTOMER WITH CUSTOMER ID : ' || VAR_CUSTOMER_ID || ' IS OF AGE : ' || VAR_AGE);
            DBMS_OUTPUT.PUT_LINE('NO CHANGE IN LOAN');
        END IF;
    END LOOP;
    COMMIT;
END;
```

```
CUSTOMER WITH CUSTOMER ID : 1 IS OF AGE : 39
NO CHANGE IN LOAN
CUSTOMER WITH CUSTOMER ID : 2 IS OF AGE : 34
NO CHANGE IN LOAN
```

-- SCENARIO 2

```
DESC CUSTOMERS;

ALTER TABLE CUSTOMERS ADD ISVIP CHAR(10) CONSTRAINT CHK1 CHECK(ISVIP IN ('TRUE','FALSE')) ;


SELECT * FROM CUSTOMERS;

SET SERVEROUTPUT ON;

DECLARE

  CURSOR CUSTOMER_CURSOR IS

    SELECT CUSTOMERID, BALANCE

    FROM CUSTOMERS;

  VAR_CUSTOMER_ID CUSTOMERS.CUSTOMERID%TYPE;

  VAR_BALANCE CUSTOMERS.BALANCE%TYPE;
```

```
BEGIN

   FOR CUSTOMER_RECORD IN CUSTOMER_CURSOR LOOP

      VAR_CUSTOMER_ID := CUSTOMER_RECORD.CUSTOMERID;

      VAR_BALANCE := CUSTOMER_RECORD.BALANCE;

      IF VAR_BALANCE > 10000 THEN

         DBMS_OUTPUT.PUT_LINE('CUSTOMER ID : ' || VAR_CUSTOMER_ID || ' HAS BALANCE GREATER
THAN 10000');

         UPDATE CUSTOMERS

         SET ISVIP = 'TRUE'

         WHERE CUSTOMERID = VAR_CUSTOMER_ID;

      ELSE

         DBMS_OUTPUT.PUT_LINE('CUSTOMER ID : ' || VAR_CUSTOMER_ID || ' HAS BALANCE LESSER THAN
10000');

         UPDATE CUSTOMERS

         SET ISVIP = 'FALSE'

         WHERE CUSTOMERID = VAR_CUSTOMER_ID;

      END IF;

   END LOOP;

   COMMIT;

END;

/

SELECT * FROM CUSTOMERS;
```



```
SELECT * FROM CUSTOMERS;
SET SERVEROUTPUT ON;
DECLARE
    CURSOR CUSTOMER_CURSOR IS
        SELECT CUSTOMERID, BALANCE
        FROM CUSTOMERS;
    VAR_CUSTOMER_ID CUSTOMERS.CUSTOMERID%TYPE;
    VAR_BALANCE CUSTOMERS.BALANCE%TYPE;
BEGIN
    FOR CUSTOMER_RECORD IN CUSTOMER_CURSOR LOOP
        VAR_CUSTOMER_ID := CUSTOMER_RECORD.CUSTOMERID;
        VAR_BALANCE := CUSTOMER_RECORD.BALANCE;
        IF VAR_BALANCE > 10000 THEN
            DBMS_OUTPUT.PUT_LINE('CUSTOMER ID : ' || VAR_CUSTOMER_ID || ' HAS BALANCE GREATER THAN 10000');
            UPDATE CUSTOMERS
            SET ISVIP = 'TRUE'
            WHERE CUSTOMERID = VAR_CUSTOMER_ID;
        ELSE
            DBMS_OUTPUT.PUT_LINE('CUSTOMER ID : ' || VAR_CUSTOMER_ID || ' HAS BALANCE LESSER THAN 10000');
            UPDATE CUSTOMERS
            SET ISVIP = 'FALSE'
```

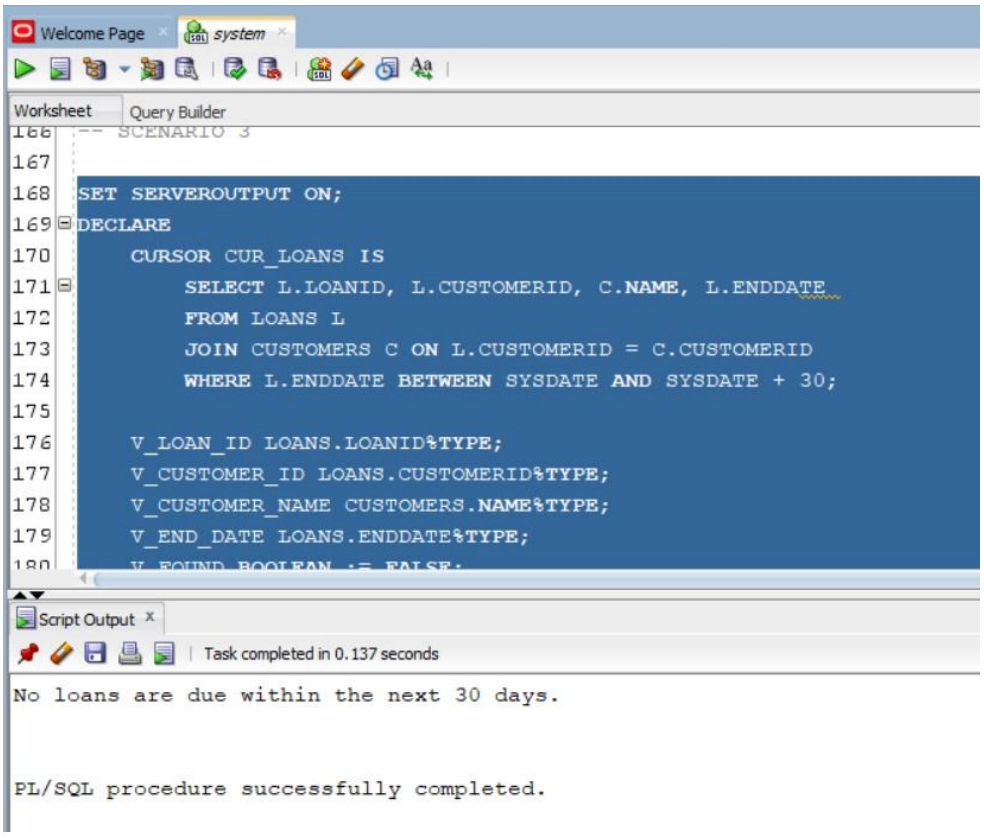
Task completed in 0.419 seconds

```
CUSTOMER ID : 1 HAS BALANCE LESSER THAN 10000
CUSTOMER ID : 2 HAS BALANCE LESSER THAN 10000


PL/SQL procedure successfully completed.
```

-- SCENARIO 3

```
SET SERVEROUTPUT ON;
DECLARE
  CURSOR CUR_LOANS IS
    SELECT L.LOANID, L.CUSTOMERID, C.NAME, L.ENDDATE
    FROM LOANS L
    JOIN CUSTOMERS C ON L.CUSTOMERID = C.CUSTOMERID
    WHERE L.ENDDATE BETWEEN SYSDATE AND SYSDATE + 30;

  V_LOAN_ID LOANS.LOANID%TYPE;
  V_CUSTOMER_ID LOANS.CUSTOMERID%TYPE;
  V_CUSTOMER_NAME CUSTOMERS.NAME%TYPE;
  V_END_DATE LOANS.ENDDATE%TYPE;
  V_FOUND BOOLEAN := FALSE;
BEGIN
```

```
  OPEN CUR_LOANS;

  LOOP

    FETCH CUR_LOANS INTO V_LOAN_ID, V_CUSTOMER_ID, V_CUSTOMER_NAME, V_END_DATE;

    EXIT WHEN CUR_LOANS%NOTFOUND;


    V_FOUND := TRUE;

    DBMS_OUTPUT.PUT_LINE('Reminder: Loan ' || V_LOAN_ID || ' for customer ' || V_CUSTOMER_NAME
|| ' (ID: ' || V_CUSTOMER_ID || ') is due on ' || TO_CHAR(V_END_DATE, 'YYYY-MM-DD'));

  END LOOP;

  CLOSE CUR_LOANS;


  IF NOT V_FOUND THEN

    DBMS_OUTPUT.PUT_LINE('No loans are due within the next 30 days.');

  END IF;

END;

/
```



```
-- SCENARIO 3

SET SERVEROUTPUT ON;
DECLARE
    CURSOR CUR_LOANS IS
        SELECT L.LOANID, L.CUSTOMERID, C.NAME, L.ENDDATE
        FROM LOANS L
        JOIN CUSTOMERS C ON L.CUSTOMERID = C.CUSTOMERID
        WHERE L.ENDDATE BETWEEN SYSDATE AND SYSDATE + 30;

    V_LOAN_ID LOANS.LOANID%TYPE;
    V_CUSTOMER_ID LOANS.CUSTOMERID%TYPE;
    V_CUSTOMER_NAME CUSTOMERS.NAME%TYPE;
    V_END_DATE LOANS.ENDDATE%TYPE;
    V_FOUND BOOLEAN := FALSE;
```

```
No loans are due within the next 30 days.


PL/SQL procedure successfully completed.
```

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.

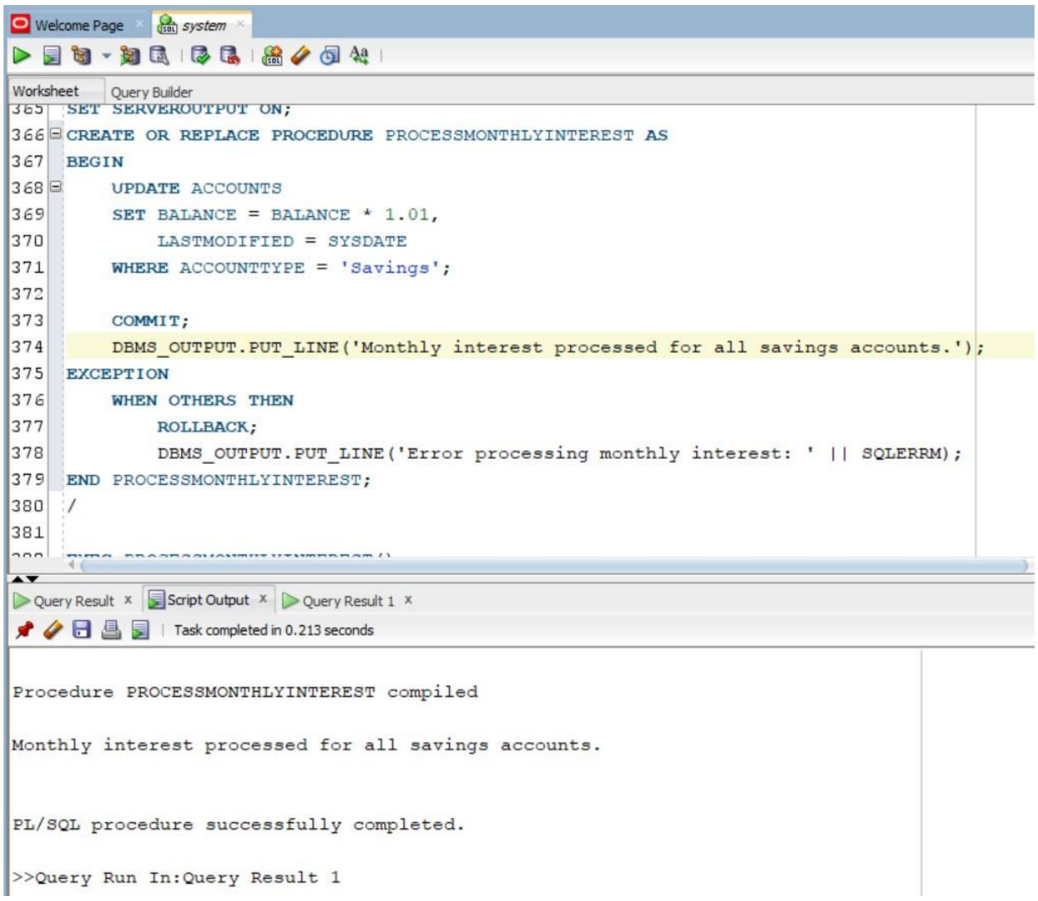
Scenario 3: Customers should be able to transfer funds between their accounts.

- **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.

-- SCENARIO 1

```
SELECT * FROM ACCOUNTS;
SET SERVEROUTPUT ON;
CREATE OR REPLACE PROCEDURE PROCESSMONTHLYINTEREST AS
BEGIN
   UPDATE ACCOUNTS
   SET BALANCE = BALANCE * 1.01,
      LASTMODIFIED = SYSDATE
   WHERE ACCOUNTTYPE = 'Savings';

   COMMIT;
   DBMS_OUTPUT.PUT_LINE('Monthly interest processed for all savings accounts.');
EXCEPTION
   WHEN OTHERS THEN
      ROLLBACK;
      DBMS_OUTPUT.PUT_LINE('Error processing monthly interest: ' || SQLERRM);
END PROCESSMONTHLYINTEREST;
/

EXEC PROCESSMONTHLYINTEREST();
```

SELECT * FROM ACCOUNTS;



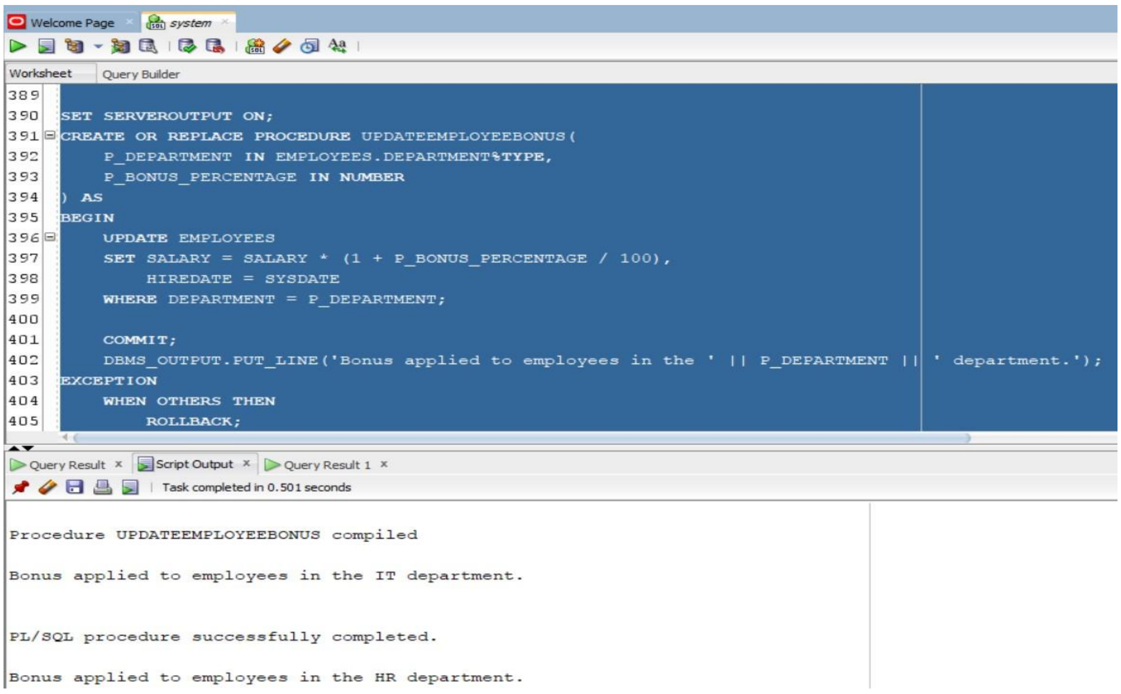
```
SET SERVEROUTPUT ON;
CREATE OR REPLACE PROCEDURE PROCESSMONTHLYINTEREST AS
BEGIN
    UPDATE ACCOUNTS
    SET BALANCE = BALANCE * 1.01,
        LASTMODIFIED = SYSDATE
    WHERE ACCOUNTTYPE = 'Savings';

    COMMIT;
    DBMS_OUTPUT.PUT_LINE('Monthly interest processed for all savings accounts.');
EXCEPTION
    WHEN OTHERS THEN
        ROLLBACK;
        DBMS_OUTPUT.PUT_LINE('Error processing monthly interest: ' || SQLERRM);
END PROCESSMONTHLYINTEREST;
/
```

```
Procedure PROCESSMONTHLYINTEREST compiled

Monthly interest processed for all savings accounts.


PL/SQL procedure successfully completed.

>>Query Run In:Query Result 1
```

-- SCENARIO 2

SELECT * FROM EMPLOYEES;

```
SET SERVEROUTPUT ON;
CREATE OR REPLACE PROCEDURE UPDATEEMPLOYEEBONUS(
   P_DEPARTMENT IN EMPLOYEES.DEPARTMENT%TYPE,
   P_BONUS_PERCENTAGE IN NUMBER
) AS
BEGIN
   UPDATE EMPLOYEES
   SET SALARY = SALARY * (1 + P_BONUS_PERCENTAGE / 100),
        HIREDATE = SYSDATE
    WHERE DEPARTMENT = P_DEPARTMENT;

    COMMIT;
    DBMS_OUTPUT.PUT_LINE('Bonus applied to employees in the ' || P_DEPARTMENT || ' department.');
EXCEPTION
    WHEN OTHERS THEN
        ROLLBACK;
        DBMS_OUTPUT.PUT_LINE('Error updating employee bonuses: ' || SQLERRM);
END UPDATEEMPLOYEEBONUS;
/

EXEC UPDATEEMPLOYEEBONUS('IT',5);
EXEC UPDATEEMPLOYEEBONUS('HR',3);

SELECT * FROM EMPLOYEES;
```



```
SET SERVEROUTPUT ON;
CREATE OR REPLACE PROCEDURE UPDATEEMPLOYEEBONUS(
    P_DEPARTMENT IN EMPLOYEES.DEPARTMENT%TYPE,
    P_BONUS_PERCENTAGE IN NUMBER
) AS
BEGIN
    UPDATE EMPLOYEES
    SET SALARY = SALARY * (1 + P_BONUS_PERCENTAGE / 100),
        HIREDATE = SYSDATE
    WHERE DEPARTMENT = P_DEPARTMENT;

    COMMIT;
    DBMS_OUTPUT.PUT_LINE('Bonus applied to employees in the ' || P_DEPARTMENT || ' department.');
EXCEPTION
    WHEN OTHERS THEN
        ROLLBACK;
```

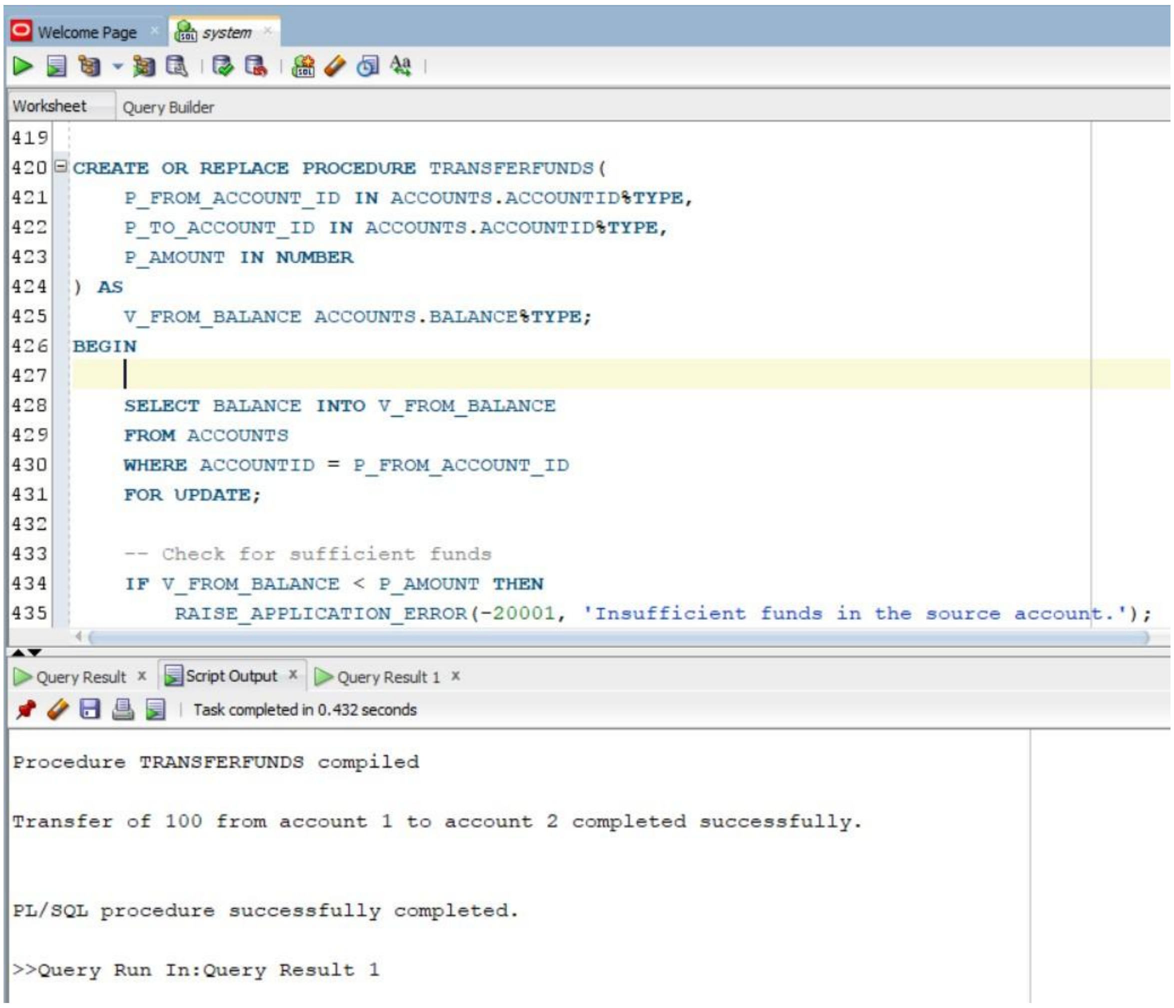
```
Procedure UPDATEEMPLOYEEBONUS compiled

Bonus applied to employees in the IT department.


PL/SQL procedure successfully completed.

Bonus applied to employees in the HR department.
```

-- SCENARIO 3

```
SELECT * FROM ACCOUNTS;
SET SERVEROUTPUT ON;
```

```
CREATE OR REPLACE PROCEDURE TRANSFERFUNDS(
    P_FROM_ACCOUNT_ID IN ACCOUNTS.ACCOUNTID%TYPE,
    P_TO_ACCOUNT_ID IN ACCOUNTS.ACCOUNTID%TYPE,
    P_AMOUNT IN NUMBER
) AS
    V_FROM_BALANCE ACCOUNTS.BALANCE%TYPE;
BEGIN

    SELECT BALANCE INTO V_FROM_BALANCE
    FROM ACCOUNTS
    WHERE ACCOUNTID = P_FROM_ACCOUNT_ID
    FOR UPDATE;

    -- Check for sufficient funds
    IF V_FROM_BALANCE < P_AMOUNT THEN
        RAISE_APPLICATION_ERROR(-20001, 'Insufficient funds in the source account.');
    END IF;

    -- Perform the transfer
    UPDATE ACCOUNTS
    SET BALANCE = BALANCE - P_AMOUNT,
        LASTMODIFIED = SYSDATE
    WHERE ACCOUNTID = P_FROM_ACCOUNT_ID;

    UPDATE ACCOUNTS
    SET BALANCE = BALANCE + P_AMOUNT,
        LASTMODIFIED = SYSDATE
    WHERE ACCOUNTID = P_TO_ACCOUNT_ID;

    COMMIT;
    DBMS_OUTPUT.PUT_LINE('Transfer of ' || P_AMOUNT || ' from account ' || P_FROM_ACCOUNT_ID || ' 
to account ' || P_TO_ACCOUNT_ID || ' completed successfully.');
```

```
EXCEPTION
   WHEN OTHERS THEN
      ROLLBACK;
      DBMS_OUTPUT.PUT_LINE('Transfer failed: ' || SQLERRM);
END TRANSFERFUNDS;
/

EXEC TRANSFERFUNDS(1,2,100);

SELECT * FROM ACCOUNTS;
```



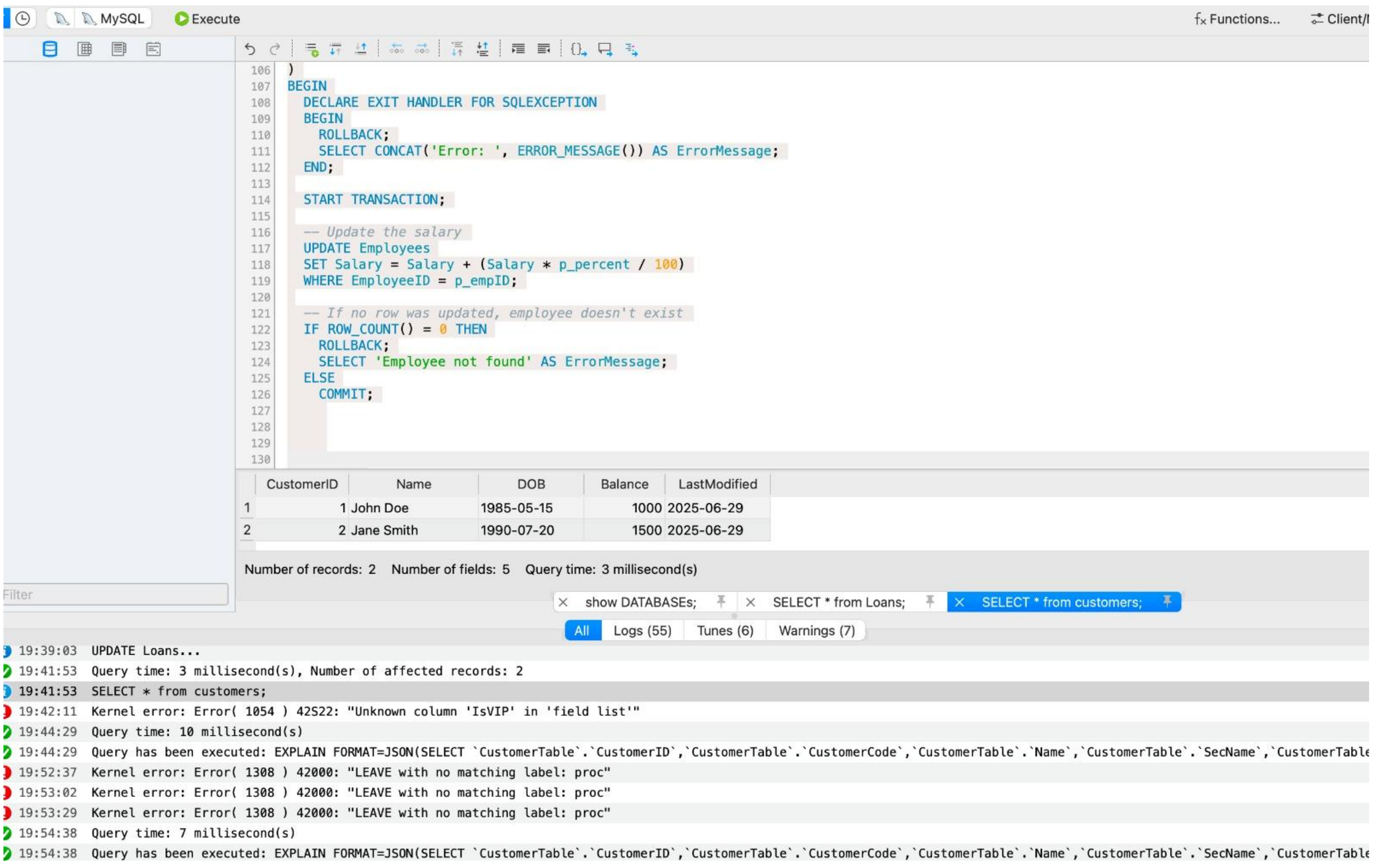
```
CREATE OR REPLACE PROCEDURE TRANSFERFUNDS(
    P_FROM_ACCOUNT_ID IN ACCOUNTS.ACCOUNTID%TYPE,
    P_TO_ACCOUNT_ID IN ACCOUNTS.ACCOUNTID%TYPE,
    P_AMOUNT IN NUMBER
) AS
    V_FROM_BALANCE ACCOUNTS.BALANCE%TYPE;
BEGIN

    SELECT BALANCE INTO V_FROM_BALANCE
    FROM ACCOUNTS
    WHERE ACCOUNTID = P_FROM_ACCOUNT_ID
    FOR UPDATE;

    -- Check for sufficient funds
    IF V_FROM_BALANCE < P_AMOUNT THEN
        RAISE_APPLICATION_ERROR(-20001, 'Insufficient funds in the source account.');
```

Task completed in 0.432 seconds

```
Procedure TRANSFERFUNDS compiled

Transfer of 100 from account 1 to account 2 completed successfully.


PL/SQL procedure successfully completed.

>>Query Run In:Query Result 1
```

Exercise 2:

SCENARIO 3



```sql
)
BEGIN
  DECLARE EXIT HANDLER FOR SQLEXCEPTION
  BEGIN
    ROLLBACK;
    SELECT CONCAT('Error: ', ERROR_MESSAGE()) AS ErrorMessage;
  END;

  START TRANSACTION;

  -- Update the salary
  UPDATE Employees
  SET Salary = Salary + (Salary * p_percent / 100)
  WHERE EmployeeID = p_empID;

  -- If no row was updated, employee doesn't exist
  IF ROW_COUNT() = 0 THEN
    ROLLBACK;
    SELECT 'Employee not found' AS ErrorMessage;
  ELSE
    COMMIT;
```

	CustomerID	Name	DOB	Balance	LastModified
1	1	John Doe	1985-05-15	1000	2025-06-29
2	2	Jane Smith	1990-07-20	1500	2025-06-29

Number of records: 2   Number of fields: 5   Query time: 3 millisecond(s)

```
19:39:03  UPDATE Loans...
19:41:53  Query time: 3 millisecond(s), Number of affected records: 2
19:41:53  SELECT * from customers;
19:42:11  Kernel error: Error( 1054 ) 42S22: "Unknown column 'IsVIP' in 'field list'"
19:44:29  Query time: 10 millisecond(s)
19:44:29  Query has been executed: EXPLAIN FORMAT=JSON(SELECT `CustomerTable`.`CustomerID`,`CustomerTable`.`CustomerCode`,`CustomerTable`.`Name`,`CustomerTable`.`SecName`,`CustomerTable
19:52:37  Kernel error: Error( 1308 ) 42000: "LEAVE with no matching label: proc"
19:53:02  Kernel error: Error( 1308 ) 42000: "LEAVE with no matching label: proc"
19:53:29  Kernel error: Error( 1308 ) 42000: "LEAVE with no matching label: proc"
19:54:38  Query time: 7 millisecond(s)
19:54:38  Query has been executed: EXPLAIN FORMAT=JSON(SELECT `CustomerTable`.`CustomerID`,`CustomerTable`.`CustomerCode`,`CustomerTable`.`Name`,`CustomerTable`.`SecName`,`CustomerTable
```