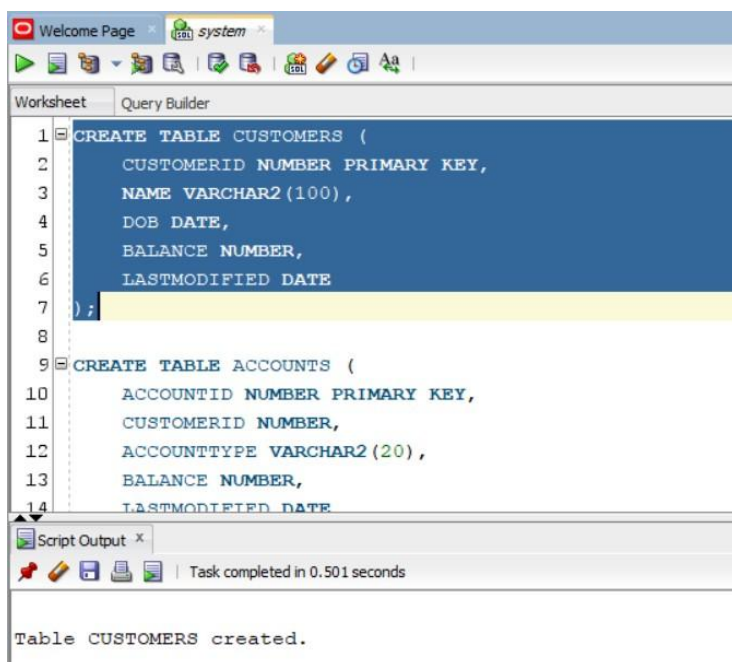


PL/SQL

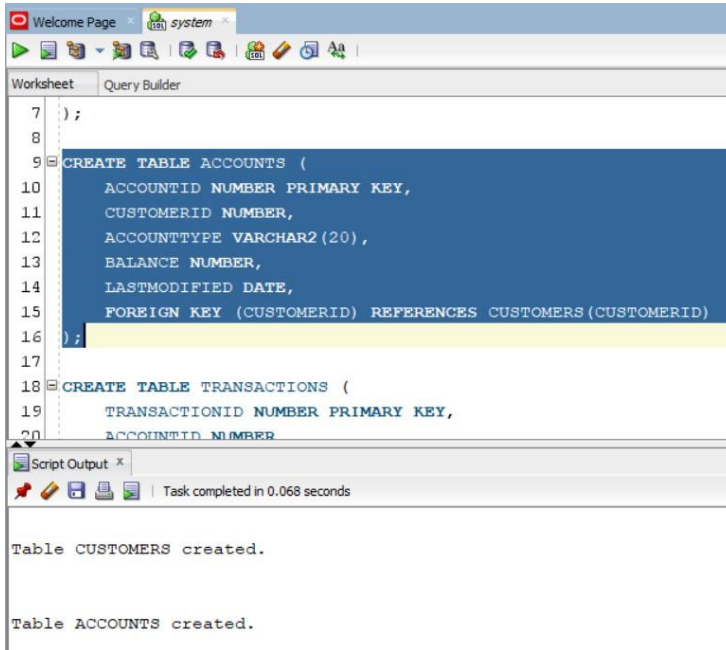
Schema Creation

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

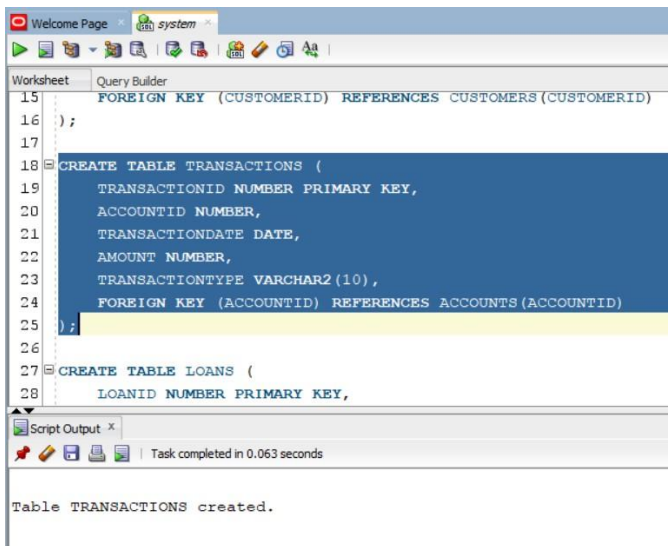


```
CREATE TABLE ACCOUNTS (  
    ACCOUNTID NUMBER PRIMARY KEY,  
    CUSTOMERID NUMBER,  
    ACCOUNTTYPE VARCHAR2(20),
```

```
BALANCE    NUMBER,  
LASTMODIFIED DATE,  
FOREIGN KEY ( CUSTOMERID )  
REFERENCES CUSTOMERS ( CUSTOMERID )  
);
```



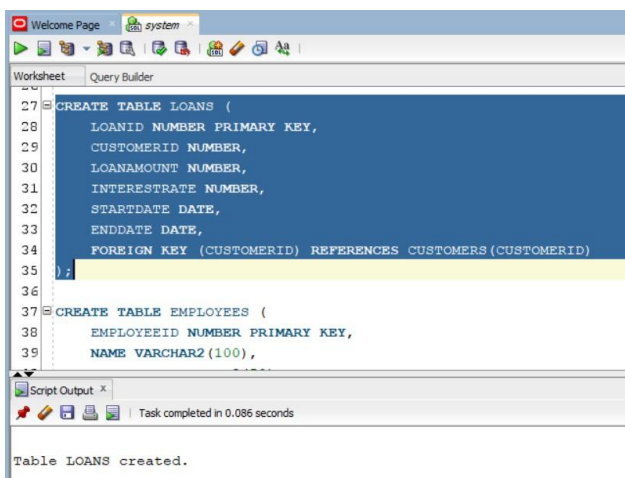
```
CREATE TABLE TRANSACTIONS (  
TRANSACTIONID  NUMBER PRIMARY KEY,  
ACCOUNTID      NUMBER,  
TRANSACTIONDATE DATE,  
AMOUNT         NUMBER,  
TRANSACTIONTYPE VARCHAR2(10),  
FOREIGN KEY ( ACCOUNTID )  
REFERENCES ACCOUNTS ( ACCOUNTID )  
);
```



```

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),

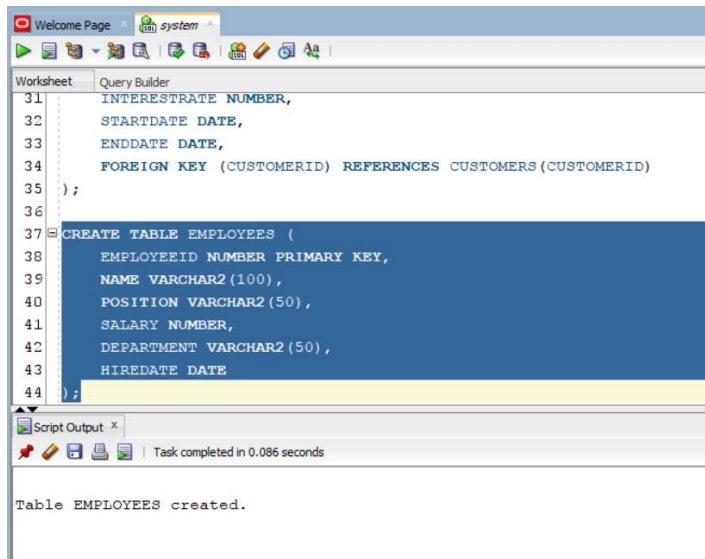
POSITION VARCHAR2(50),

SALARY NUMBER,

DEPARTMENT VARCHAR2(50),

HIREDATE DATE

);



The screenshot shows the SQL Developer interface with a 'Query Builder' window. The SQL script in the editor is as follows:

```
31 INTERESTRATE NUMBER,  
32 STARTDATE DATE,  
33 ENDDATE DATE,  
34 FOREIGN KEY (CUSTOMERID) REFERENCES CUSTOMERS (CUSTOMERID)  
35 );  
36  
37 CREATE TABLE EMPLOYEES (  
38     EMPLOYEEID NUMBER PRIMARY KEY,  
39     NAME VARCHAR2(100),  
40     POSITION VARCHAR2(50),  
41     SALARY NUMBER,  
42     DEPARTMENT VARCHAR2(50),  
43     HIREDATE DATE  
44 );
```

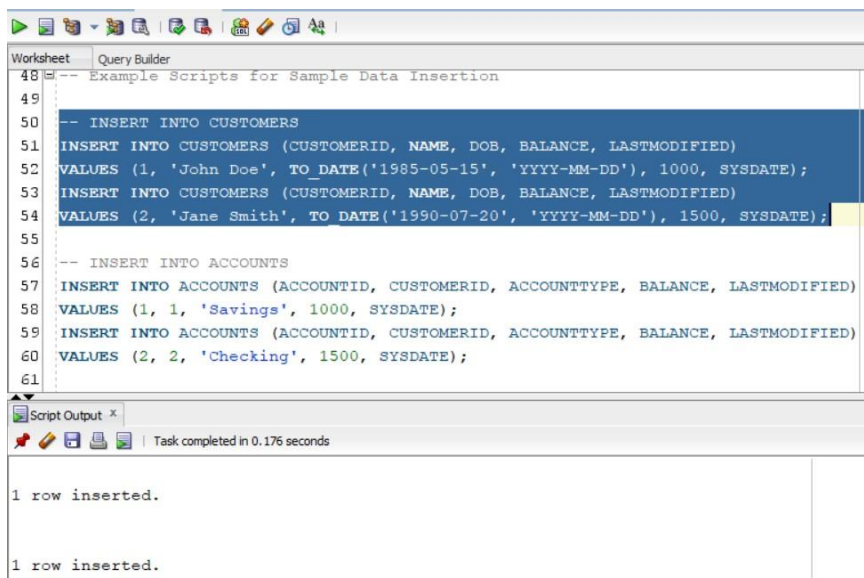
The 'Script Output' window at the bottom shows the message: 'Table EMPLOYEES created.'

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);



The screenshot shows the SQL Developer interface with a 'Query Builder' window. The SQL script in the editor is as follows:

```
48 -- Example Scripts for Sample Data Insertion  
49  
50 -- INSERT INTO CUSTOMERS  
51 INSERT INTO CUSTOMERS (CUSTOMERID, NAME, DOB, BALANCE, LASTMODIFIED)  
52 VALUES (1, 'John Doe', TO_DATE('1985-05-15', 'YYYY-MM-DD'), 1000, SYSDATE);  
53 INSERT INTO CUSTOMERS (CUSTOMERID, NAME, DOB, BALANCE, LASTMODIFIED)  
54 VALUES (2, 'Jane Smith', TO_DATE('1990-07-20', 'YYYY-MM-DD'), 1500, SYSDATE);  
55  
56 -- INSERT INTO ACCOUNTS  
57 INSERT INTO ACCOUNTS (ACCOUNTID, CUSTOMERID, ACCOUNTTYPE, BALANCE, LASTMODIFIED)  
58 VALUES (1, 1, 'Savings', 1000, SYSDATE);  
59 INSERT INTO ACCOUNTS (ACCOUNTID, CUSTOMERID, ACCOUNTTYPE, BALANCE, LASTMODIFIED)  
60 VALUES (2, 2, 'Checking', 1500, SYSDATE);  
61
```

The 'Script Output' window at the bottom shows the messages: '1 row inserted.' and '1 row inserted.'

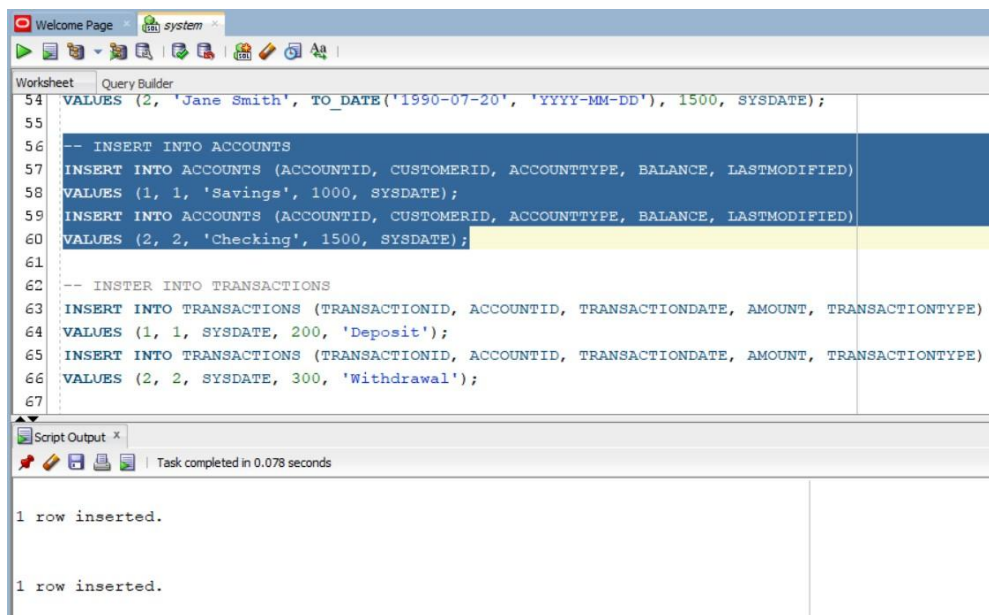
-- INSERT INTO ACCOUNTS

```
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);
```



The screenshot shows a SQL script execution window with a 'Query Builder' tab. The script contains the following SQL commands:

```
54 VALUES (2, 'Jane Smith', TO_DATE('1990-07-20', 'YYYY-MM-DD'), 1500, SYSDATE);
55
56 -- INSERT INTO ACCOUNTS
57 INSERT INTO ACCOUNTS (ACCOUNTID, CUSTOMERID, ACCOUNTTYPE, BALANCE, LASTMODIFIED)
58 VALUES (1, 1, 'Savings', 1000, SYSDATE);
59 INSERT INTO ACCOUNTS (ACCOUNTID, CUSTOMERID, ACCOUNTTYPE, BALANCE, LASTMODIFIED)
60 VALUES (2, 2, 'Checking', 1500, SYSDATE);
61
62 -- INSTER INTO TRANSACTIONS
63 INSERT INTO TRANSACTIONS (TRANSACTIONID, ACCOUNTID, TRANSACTIONDATE, AMOUNT, TRANSACTIONTYPE)
64 VALUES (1, 1, SYSDATE, 200, 'Deposit');
65 INSERT INTO TRANSACTIONS (TRANSACTIONID, ACCOUNTID, TRANSACTIONDATE, AMOUNT, TRANSACTIONTYPE)
66 VALUES (2, 2, SYSDATE, 300, 'Withdrawal');
67
```

The 'Script Output' tab shows the results of the execution:

```
1 row inserted.

1 row inserted.
```

The status bar indicates 'Task completed in 0.078 seconds'.

-- 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');
```

```
Worksheet | Query Builder
59 INSERT INTO ACCOUNTS (ACCOUNTID, CUSTOMERID, ACCOUNTTYPE, BALANCE, LASTMODIFIED)
60 VALUES (2, 2, 'Checking', 1500, SYSDATE);
61
62 -- INSERT INTO TRANSACTIONS
63 INSERT INTO TRANSACTIONS (TRANSACTIONID, ACCOUNTID, TRANSACTIONDATE, AMOUNT, TRANSACTIONTYPE)
64 VALUES (1, 1, SYSDATE, 200, 'Deposit');
65 INSERT INTO TRANSACTIONS (TRANSACTIONID, ACCOUNTID, TRANSACTIONDATE, AMOUNT, TRANSACTIONTYPE)
66 VALUES (2, 2, SYSDATE, 300, 'Withdrawal');
67
68 -- INSERT INTO LOANS
69 INSERT INTO LOANS (LOANID, CUSTOMERID, LOANAMOUNT, INTERESTRATE, STARTDATE, ENDDATE)
70 VALUES (1, 1, 5000, 5, SYSDATE, ADD_MONTHS(SYSDATE, 60));
71
```

Script Output x

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));

```
Worksheet | Query Builder
64 VALUES (1, 1, SYSDATE, 200, 'Deposit');
65 INSERT INTO TRANSACTIONS (TRANSACTIONID, ACCOUNTID, TRANSACTIONDATE, AMOUNT, TRANSACTIONTYPE)
66 VALUES (2, 2, SYSDATE, 300, 'Withdrawal');
67
68 -- INSERT INTO LOANS
69 INSERT INTO LOANS (LOANID, CUSTOMERID, LOANAMOUNT, INTERESTRATE, STARTDATE, ENDDATE)
70 VALUES (1, 1, 5000, 5, SYSDATE, ADD_MONTHS(SYSDATE, 60));
71
72 -- INSERT INTO EMPLOYEES
73 INSERT INTO EMPLOYEES (EMPLOYEEID, NAME, POSITION, SALARY, DEPARTMENT, HIREDATE)
74 VALUES (1, 'Alice Johnson', 'Manager', 70000, 'HR', TO_DATE('2015-06-15', 'YYYY-MM-DD'));
75 INSERT INTO EMPLOYEES (EMPLOYEEID, NAME, POSITION, SALARY, DEPARTMENT, HIREDATE)
76 VALUES (2, 'Bob Brown', 'Developer', 60000, 'IT', TO_DATE('2017-03-20', 'YYYY-MM-DD'));
77
```

Script Output x

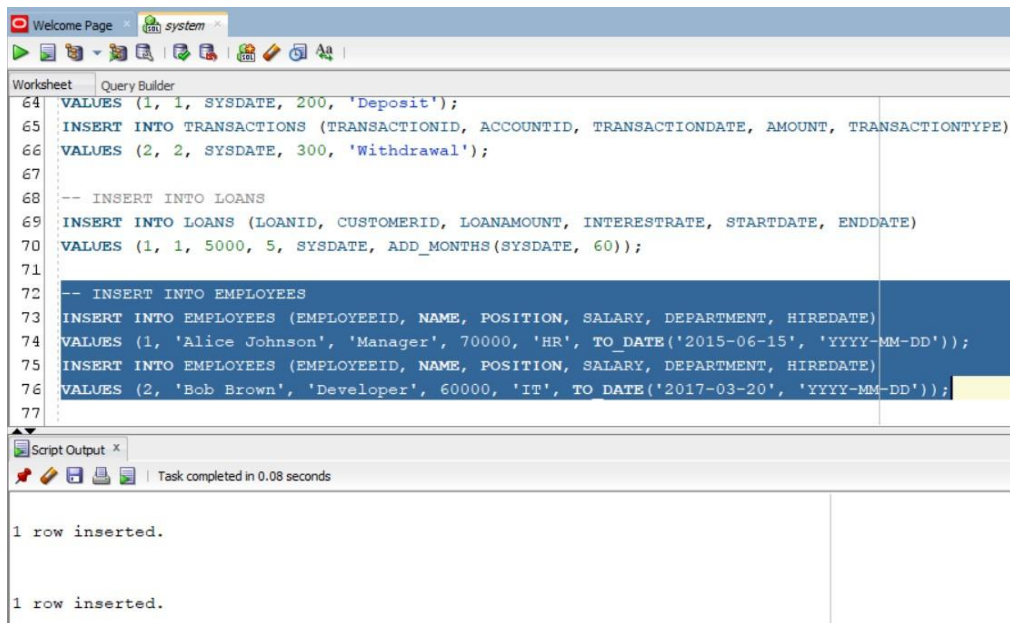
Task completed in 0.039 seconds

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'));
```



The screenshot shows the Oracle SQL Developer interface. The top pane is the 'Query Builder' with a SQL script. The bottom pane is the 'Script Output' window showing the results of the script execution.

```
64 VALUES (1, 1, SYSDATE, 200, 'Deposit');
65 INSERT INTO TRANSACTIONS (TRANSACTIONID, ACCOUNTID, TRANSACTIONDATE, AMOUNT, TRANSACTIONTYPE)
66 VALUES (2, 2, SYSDATE, 300, 'Withdrawal');
67
68 -- INSERT INTO LOANS
69 INSERT INTO LOANS (LOANID, CUSTOMERID, LOANAMOUNT, INTERESTRATE, STARTDATE, ENDDATE)
70 VALUES (1, 1, 5000, 5, SYSDATE, ADD_MONTHS(SYSDATE, 60));
71
72 -- INSERT INTO EMPLOYEES
73 INSERT INTO EMPLOYEES (EMPLOYEEID, NAME, POSITION, SALARY, DEPARTMENT, HIREDATE)
74 VALUES (1, 'Alice Johnson', 'Manager', 70000, 'HR', TO_DATE('2015-06-15', 'YYYY-MM-DD'));
75 INSERT INTO EMPLOYEES (EMPLOYEEID, NAME, POSITION, SALARY, DEPARTMENT, HIREDATE)
76 VALUES (2, 'Bob Brown', 'Developer', 60000, 'IT', TO_DATE('2017-03-20', 'YYYY-MM-DD'));
77
```

Script Output X

Task completed in 0.08 seconds

```
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;

```


The screenshot shows a SQL IDE window with a 'Query Builder' tab. The script is a PL/SQL block that declares a cursor, loops through customers, and updates loan interest rates for those aged 60 or older. The results pane shows the output of the script, indicating that interest rates were updated for two customers.

```
107 DECLARE
108     CURSOR CUSTOMER_CURSOR IS
109         SELECT CUSTOMERID, EXTRACT(YEAR FROM SYSDATE) - EXTRACT(YEAR FROM DOB) AS AGE
110         FROM CUSTOMERS;
111     VAR_CUSTOMER_ID CUSTOMERS.CUSTOMERID%TYPE;
112     VAR_AGE NUMBER;
113 BEGIN
114     FOR CUSTOMER_RECORD IN CUSTOMER_CURSOR LOOP
115         VAR_CUSTOMER_ID := CUSTOMER_RECORD.CUSTOMERID;
116         VAR_AGE := CUSTOMER_RECORD.AGE;
117         IF VAR_AGE > 60 THEN
118             UPDATE LOANS
119             SET INTERESTRATE = INTERESTRATE - 1
120             WHERE CUSTOMERID = VAR_CUSTOMER_ID;
121         ELSE
122             DBMS_OUTPUT.PUT_LINE('CUSTOMER WITH CUSTOMER ID : ' || VAR_CUSTOMER_ID || ' IS OF AGE : ' || VAR_AGE);
123             DBMS_OUTPUT.PUT_LINE('NO CHANGE IN LOAN');
124         END IF;
125     END LOOP;
126     COMMIT;
127 END;
```

Query Result 1 x | Script Output x | Query Result 2 x
Task completed in 0.409 seconds

```
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;

The screenshot shows the Oracle SQL Developer interface. The top pane, titled 'Query Builder', contains a PL/SQL script. The script starts with a SELECT statement, followed by SET SERVEROUTPUT ON, and a DECLARE section. It then uses a cursor to loop through customer records, checking if their balance is greater than 10,000. If so, it updates the CUSTOMERS table to set ISVIP to 'TRUE'. Otherwise, it sets ISVIP to 'FALSE'. The bottom pane, titled 'Script Output', shows the results of the script execution, which completed in 0.419 seconds. The output displays two lines: 'CUSTOMER ID : 1 HAS BALANCE LESSER THAN 10000' and 'CUSTOMER ID : 2 HAS BALANCE LESSER THAN 10000'. At the bottom, a message states 'PL/SQL procedure successfully completed.'

```
137 SELECT * FROM CUSTOMERS;
138 SET SERVEROUTPUT ON;
139 DECLARE
140     CURSOR CUSTOMER_CURSOR IS
141         SELECT CUSTOMERID, BALANCE
142         FROM CUSTOMERS;
143     VAR_CUSTOMER_ID CUSTOMERS.CUSTOMERID%TYPE;
144     VAR_BALANCE CUSTOMERS.BALANCE%TYPE;
145 BEGIN
146     FOR CUSTOMER_RECORD IN CUSTOMER_CURSOR LOOP
147         VAR_CUSTOMER_ID := CUSTOMER_RECORD.CUSTOMERID;
148         VAR_BALANCE := CUSTOMER_RECORD.BALANCE;
149         IF VAR_BALANCE > 10000 THEN
150             DBMS_OUTPUT.PUT_LINE('CUSTOMER ID : ' || VAR_CUSTOMER_ID || ' HAS BALANCE GREATER THAN 10000');
151             UPDATE CUSTOMERS
152             SET ISVIP = 'TRUE'
153             WHERE CUSTOMERID = VAR_CUSTOMER_ID;
154         ELSE
155             DBMS_OUTPUT.PUT_LINE('CUSTOMER ID : ' || VAR_CUSTOMER_ID || ' HAS BALANCE LESSER THAN 10000');
156             UPDATE CUSTOMERS
157             SET ISVIP = 'FALSE'
```

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;

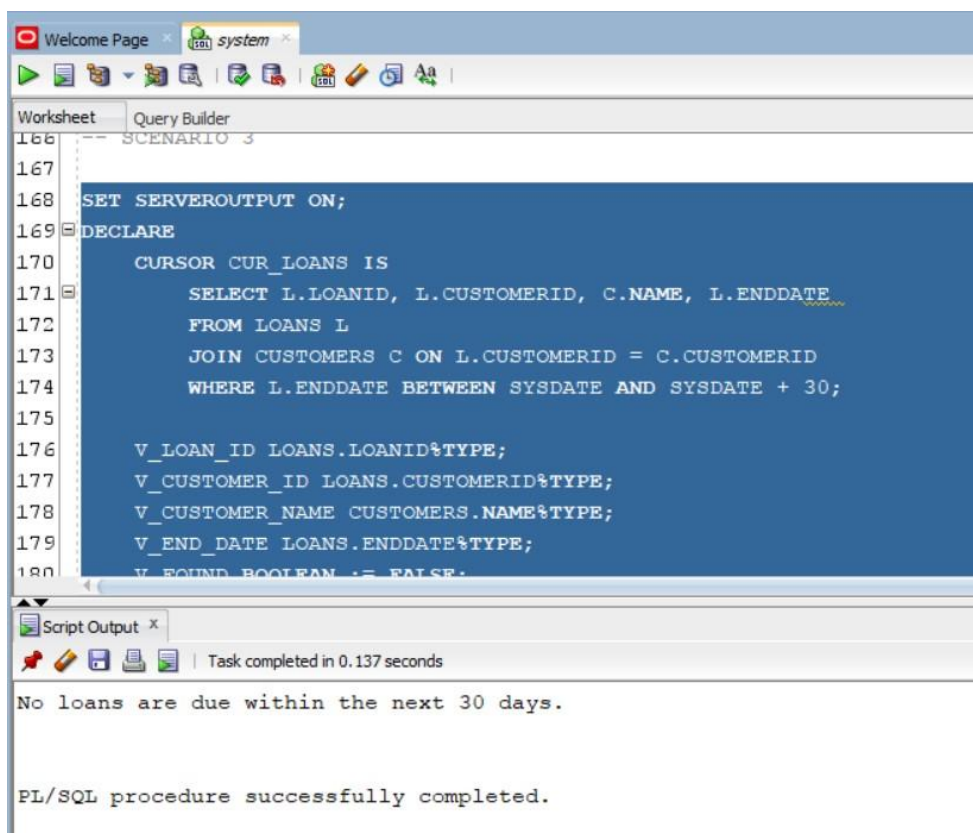
IF NOT V_FOUND THEN

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

END IF;

END;

/



The screenshot shows the Oracle SQL Developer interface. The top toolbar includes icons for running, saving, and other database operations. The main window is titled 'Query Builder' and contains a PL/SQL script. The script starts with 'SET SERVEROUTPUT ON;' and 'DECLARE'. It defines a cursor 'CUR_LOANS' with a SELECT statement that joins the 'LOANS' and 'CUSTOMERS' tables, filtering for loans due within the next 30 days. The script then declares variables 'V_LOAN_ID', 'V_CUSTOMER_ID', 'V_CUSTOMER_NAME', 'V_END_DATE', and 'V_FOUND' (a BOOLEAN). The output window at the bottom, titled 'Script Output', shows the message 'No loans are due within the next 30 days.' and 'PL/SQL procedure successfully completed.'.

```

166 -- SCENARIO 3
167
168 SET SERVEROUTPUT ON;
169 DECLARE
170     CURSOR CUR_LOANS IS
171         SELECT L.LOANID, L.CUSTOMERID, C.NAME, L.ENDDATE
172         FROM LOANS L
173         JOIN CUSTOMERS C ON L.CUSTOMERID = C.CUSTOMERID
174         WHERE L.ENDDATE BETWEEN SYSDATE AND SYSDATE + 30;
175
176     V_LOAN_ID LOANS.LOANID%TYPE;
177     V_CUSTOMER_ID LOANS.CUSTOMERID%TYPE;
178     V_CUSTOMER_NAME CUSTOMERS.NAME%TYPE;
179     V_END_DATE LOANS.ENDDATE%TYPE;
180     V_FOUND BOOLEAN := FALSE;
```

Script Output x

Task completed in 0.137 seconds

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;

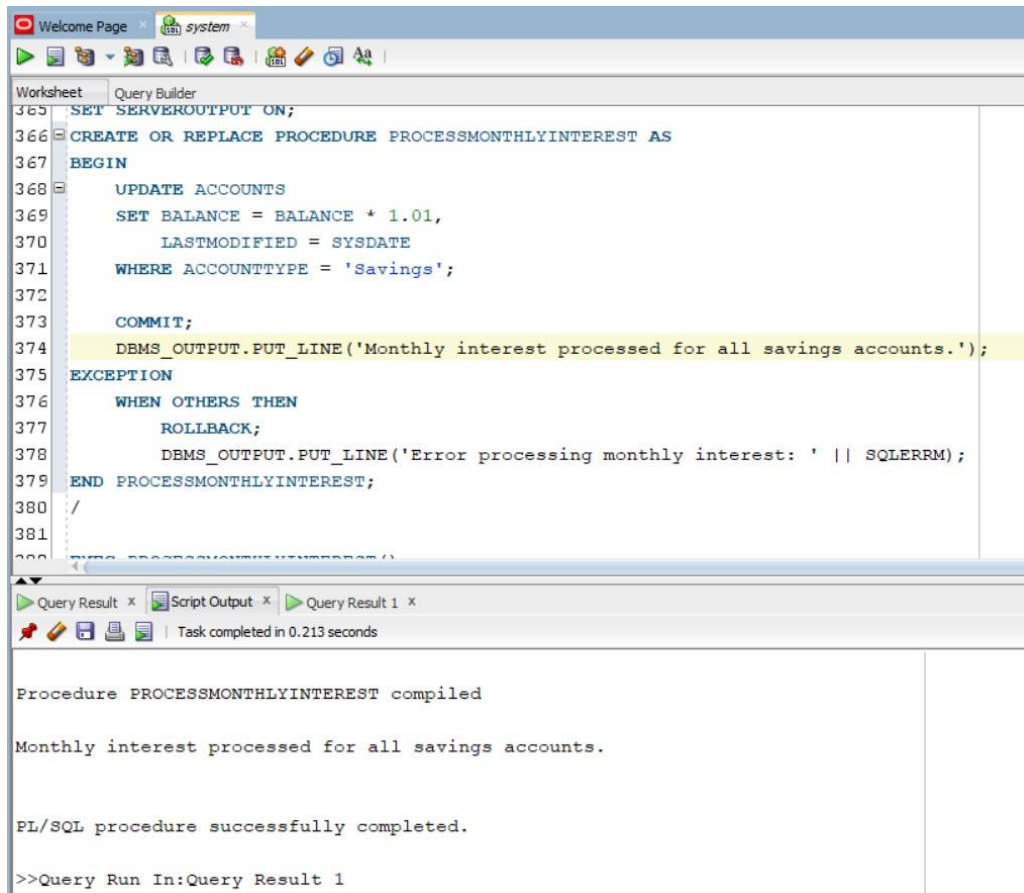
        DBMS_OUTPUT.PUT_LINE('Error processing monthly interest: ' || SQLERRM);

END PROCESSMONTHLYINTEREST;

/

EXEC PROCESSMONTHLYINTEREST();
```

```
SELECT * FROM ACCOUNTS;
```



The screenshot displays the Oracle SQL Developer environment. The top pane, titled 'Query Builder', contains a PL/SQL procedure named 'PROCESSMONTHLYINTEREST'. The procedure starts with 'SET SERVEROUTPUT ON;', followed by a 'BEGIN' block. Inside, it updates the 'ACCOUNTS' table, increasing the balance by 1.01% for 'Savings' accounts and recording the last modified date. It then commits the transaction and outputs a message: 'Monthly interest processed for all savings accounts.'. An 'EXCEPTION' block handles errors by rolling back and outputting an error message. The procedure ends with 'END PROCESSMONTHLYINTEREST;' and a forward slash. The bottom pane shows the execution results, confirming the procedure was compiled, the message was output, and the PL/SQL procedure completed successfully. The status bar indicates the task was completed in 0.213 seconds.

```
365 SET SERVEROUTPUT ON;
366 CREATE OR REPLACE PROCEDURE PROCESSMONTHLYINTEREST AS
367 BEGIN
368     UPDATE ACCOUNTS
369     SET BALANCE = BALANCE * 1.01,
370         LASTMODIFIED = SYSDATE
371     WHERE ACCOUNTTYPE = 'Savings';
372
373     COMMIT;
374     DBMS_OUTPUT.PUT_LINE('Monthly interest processed for all savings accounts.');
```

```
375 EXCEPTION
376     WHEN OTHERS THEN
377         ROLLBACK;
378         DBMS_OUTPUT.PUT_LINE('Error processing monthly interest: ' || SQLERRM);
379 END PROCESSMONTHLYINTEREST;
380 /
381
382
```

Task completed in 0.213 seconds

```
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;
```

```

389
390 SET SERVEROUTPUT ON;
391 CREATE OR REPLACE PROCEDURE UPDATEEMPLOYEEBONUS(
392     P_DEPARTMENT IN EMPLOYEES.DEPARTMENT%TYPE,
393     P_BONUS_PERCENTAGE IN NUMBER
394 ) AS
395 BEGIN
396     UPDATE EMPLOYEES
397     SET SALARY = SALARY * (1 + P_BONUS_PERCENTAGE / 100),
398         HIREDATE = SYSDATE
399     WHERE DEPARTMENT = P_DEPARTMENT;
400
401     COMMIT;
402     DBMS_OUTPUT.PUT_LINE('Bonus applied to employees in the ' || P_DEPARTMENT || ' department.');
```

Task completed in 0.501 seconds

```

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;

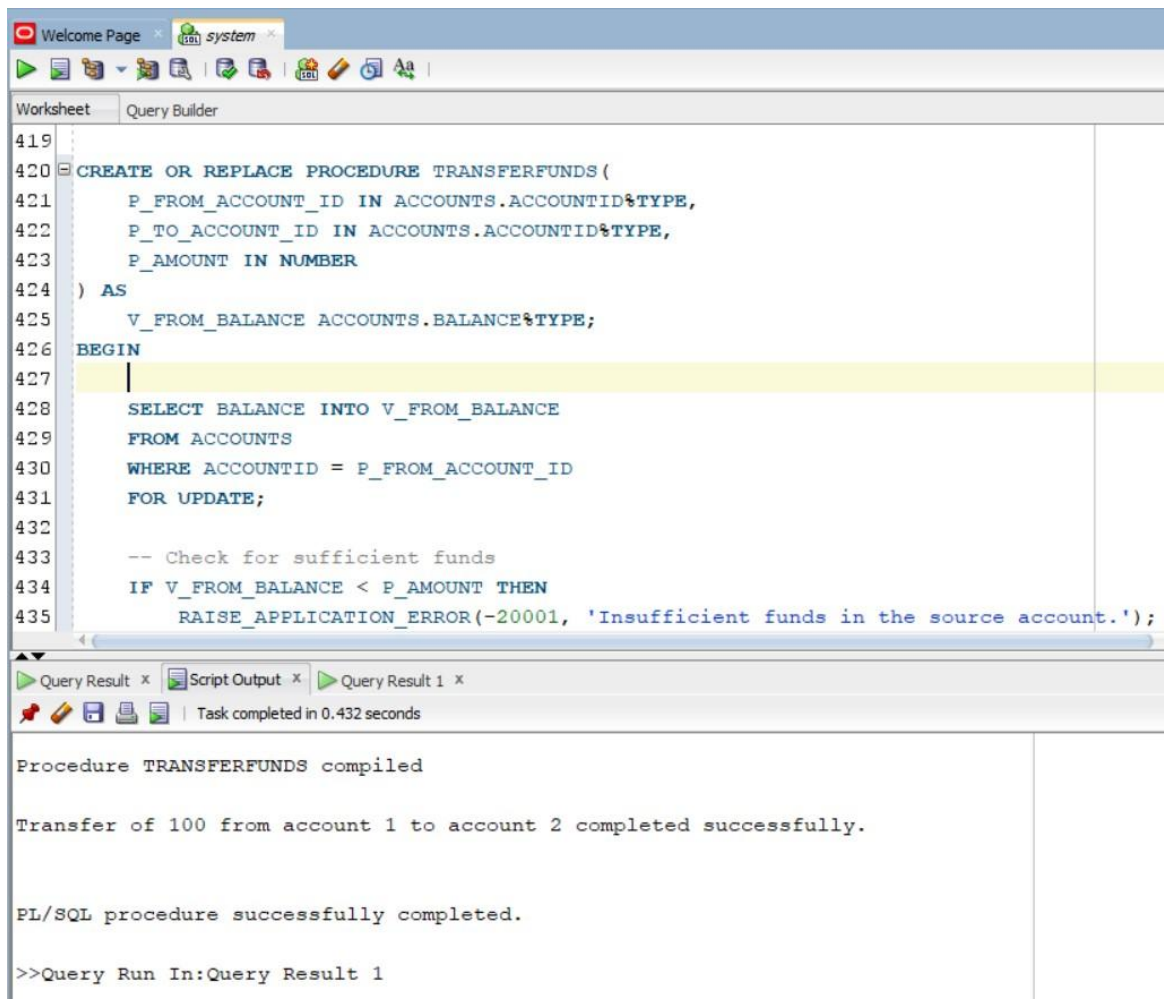
DBMS_OUTPUT.PUT_LINE('Transfer failed: ' || SQLERRM);

END TRANSFERFUNDS;

/

EXEC TRANSFERFUNDS(1,2,100);

SELECT * FROM ACCOUNTS;



The screenshot shows a SQL IDE window with a 'Query Builder' tab. The main editor displays a PL/SQL procedure named 'TRANSFERFUNDS'. The procedure has three input parameters: 'P_FROM_ACCOUNT_ID' (ACCOUNTID%TYPE), 'P_TO_ACCOUNT_ID' (ACCOUNTID%TYPE), and 'P_AMOUNT' (NUMBER). It declares a variable 'V_FROM_BALANCE' of type 'ACCOUNTS.BALANCE%TYPE'. The procedure begins with a 'SELECT' statement to fetch the balance from the 'ACCOUNTS' table where the account ID matches 'P_FROM_ACCOUNT_ID', and locks the row for update. It then checks if the balance is less than the amount. If so, it raises an application error with message number -20001 and the text 'Insufficient funds in the source account.'. The procedure is then executed, and the output window shows the following messages:

```
419
420 CREATE OR REPLACE PROCEDURE TRANSFERFUNDS (
421     P_FROM_ACCOUNT_ID IN ACCOUNTS.ACCOUNTID%TYPE,
422     P_TO_ACCOUNT_ID IN ACCOUNTS.ACCOUNTID%TYPE,
423     P_AMOUNT IN NUMBER
424 ) AS
425     V_FROM_BALANCE ACCOUNTS.BALANCE%TYPE;
426 BEGIN
427
428     SELECT BALANCE INTO V_FROM_BALANCE
429     FROM ACCOUNTS
430     WHERE ACCOUNTID = P_FROM_ACCOUNT_ID
431     FOR UPDATE;
432
433     -- Check for sufficient funds
434     IF V_FROM_BALANCE < P_AMOUNT THEN
435         RAISE_APPLICATION_ERROR(-20001, 'Insufficient funds in the source account.');
```

Query Result x | Script Output x | Query Result 1 x

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

MySQLExecute

fx Functions...Client/...

106
107
108
109
110
111
112
113
114
115
116
117
118
119
120
121
122
123
124
125
126
127
128
129
130

```
)  
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;  
  END IF;  
END
```

	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)

Filter

show DATABASEs;

SELECT * from Loans;

SELECT * from customers;

All Logs (55) Tunes (6) Warnings (7)

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) 42522: "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`