**PL/SQL**

**Schema Creation**

CREATE TABLE CUSTOMERS (

CUSTOMERID NUMBER PRIMARY KEY,

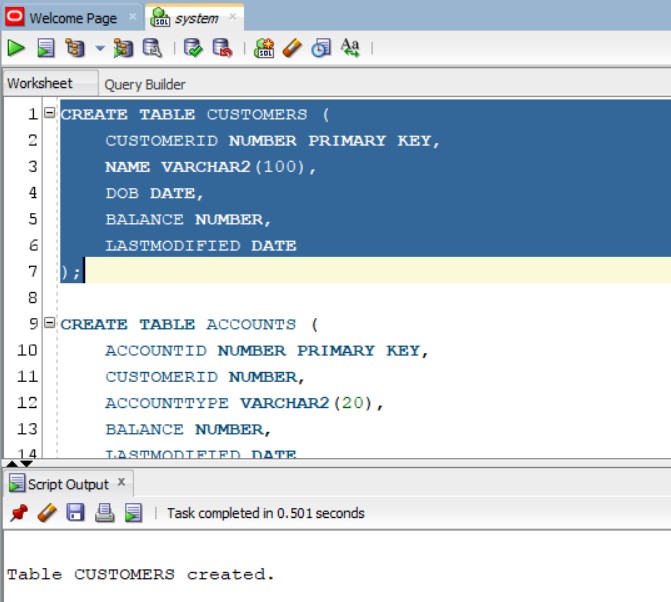
NAME VARCHAR2(100),

DOB DATE,

BALANCE NUMBER,

LASTMODIFIED DATE

);



CREATE TABLE ACCOUNTS (

ACCOUNTID NUMBER PRIMARY KEY,

CUSTOMERID NUMBER,

ACCOUNTTYPE VARCHAR2(20),

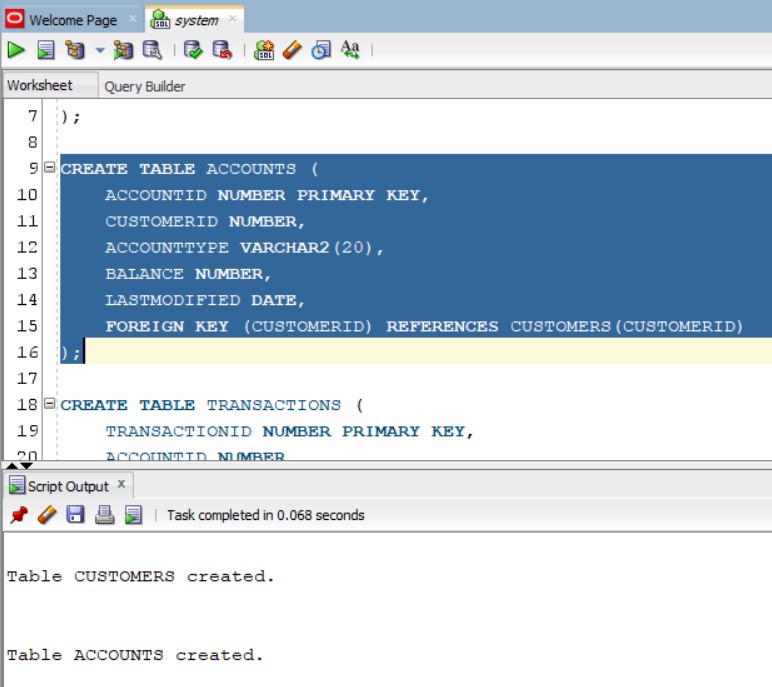
BALANCE NUMBER,

LASTMODIFIED DATE,

FOREIGN KEY ( CUSTOMERID )

REFERENCES CUSTOMERS ( CUSTOMERID )

);



CREATE TABLE TRANSACTIONS (

TRANSACTIONID NUMBER PRIMARY KEY,

ACCOUNTID NUMBER,

TRANSACTIONDATE DATE,

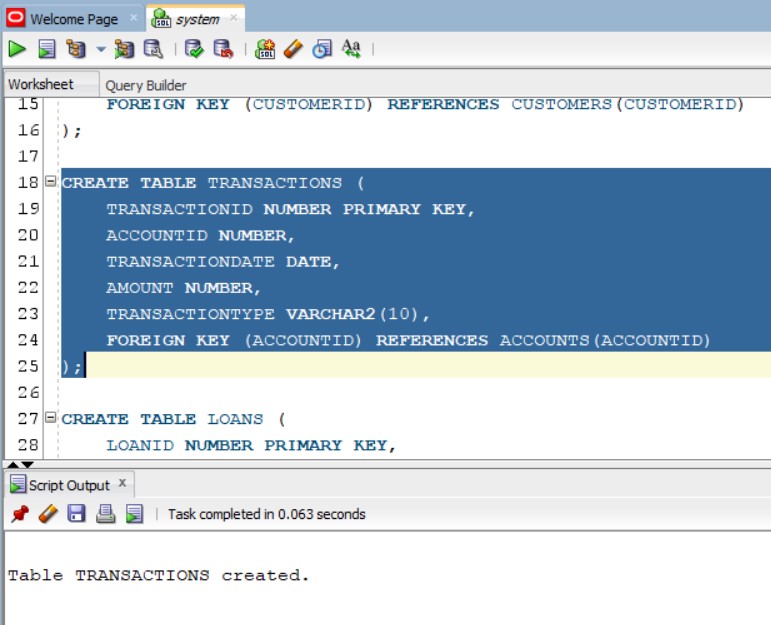
AMOUNT NUMBER,

TRANSACTIONTYPE VARCHAR2(10),

FOREIGN KEY ( ACCOUNTID )

REFERENCES ACCOUNTS ( ACCOUNTID )

);



CREATE TABLE LOANS (

LOANID NUMBER PRIMARY KEY,

CUSTOMERID NUMBER,

LOANAMOUNT NUMBER,

INTERESTRATE NUMBER,

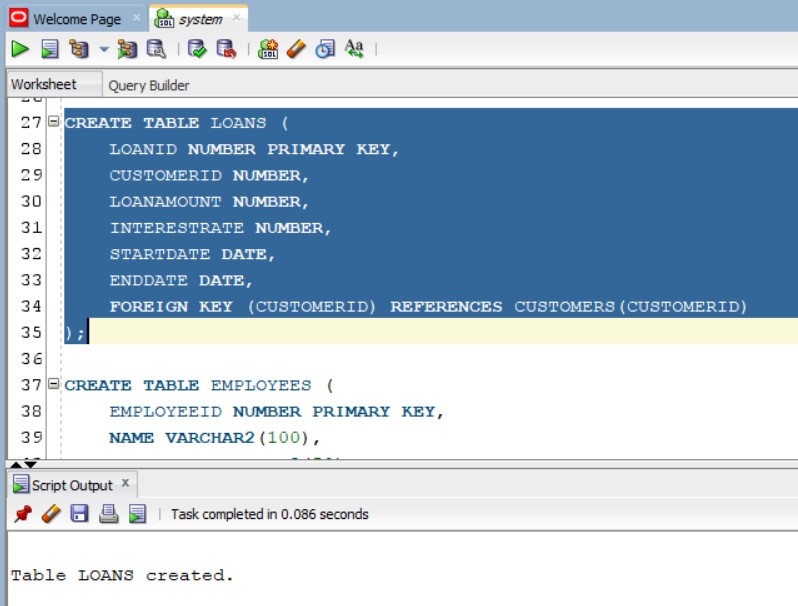
STARTDATE DATE,

ENDDATE DATE,

FOREIGN KEY ( CUSTOMERID )

REFERENCES CUSTOMERS ( CUSTOMERID )

);



CREATE TABLE EMPLOYEES (

EMPLOYEEID NUMBER PRIMARY KEY,

NAME VARCHAR2(100),

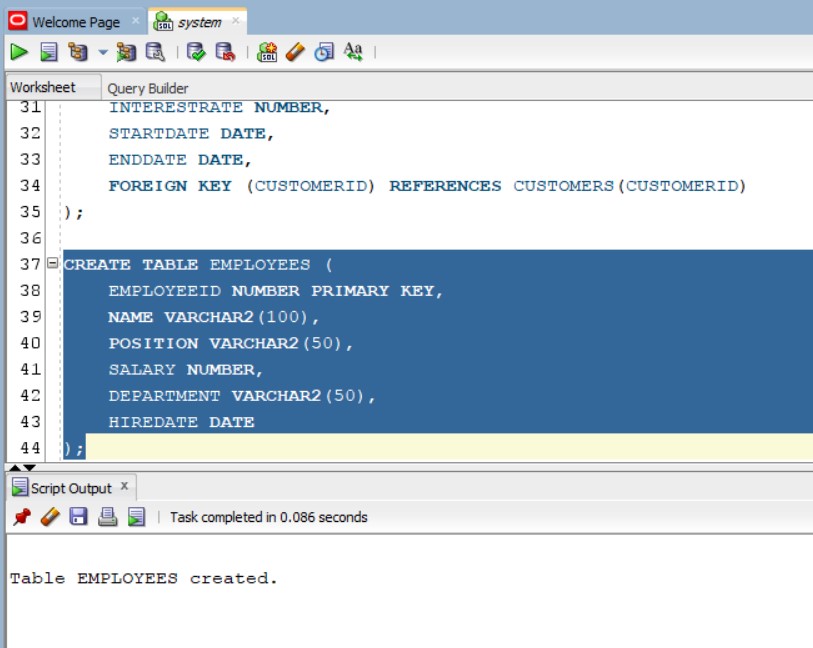
POSITION VARCHAR2(50),

SALARY NUMBER,

DEPARTMENT VARCHAR2(50),

HIREDATE DATE

);

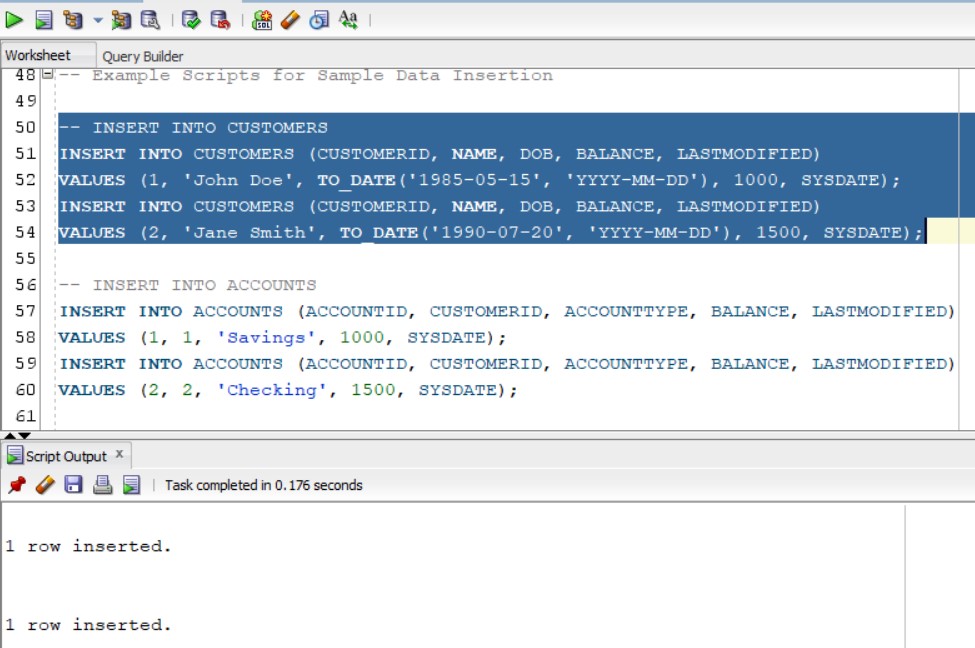


INSERT INTO CUSTOMERS (CUSTOMERID, NAME, DOB, BALANCE, LASTMODIFIED)

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);



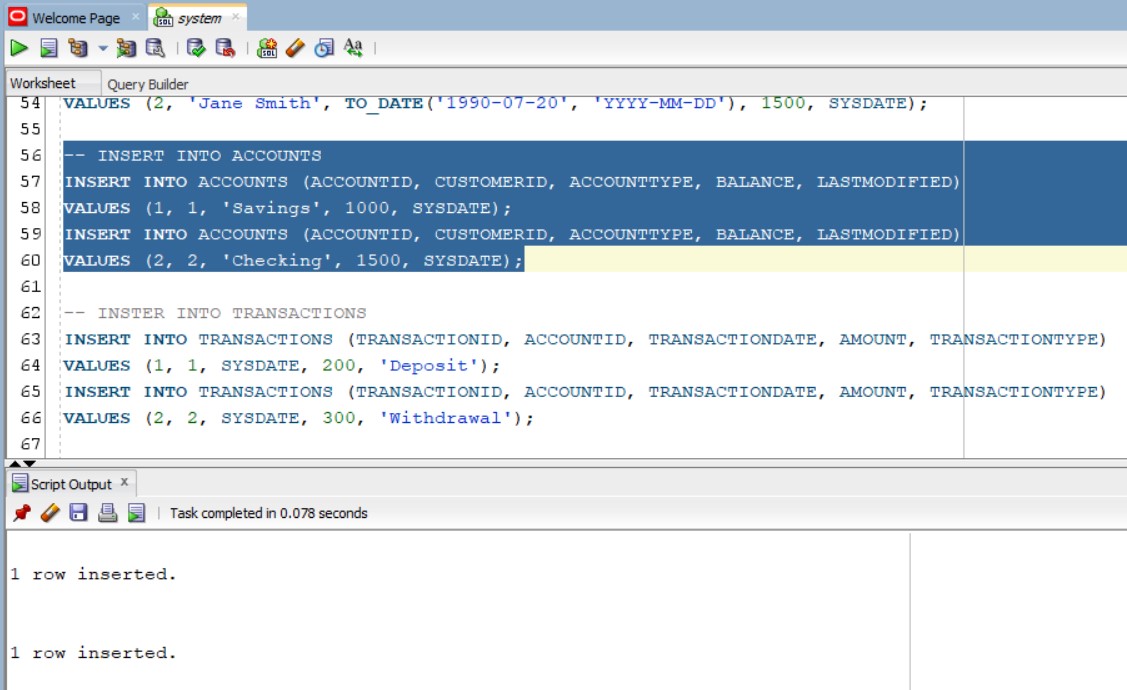
**-- INSERT INTO ACCOUNTS**

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);



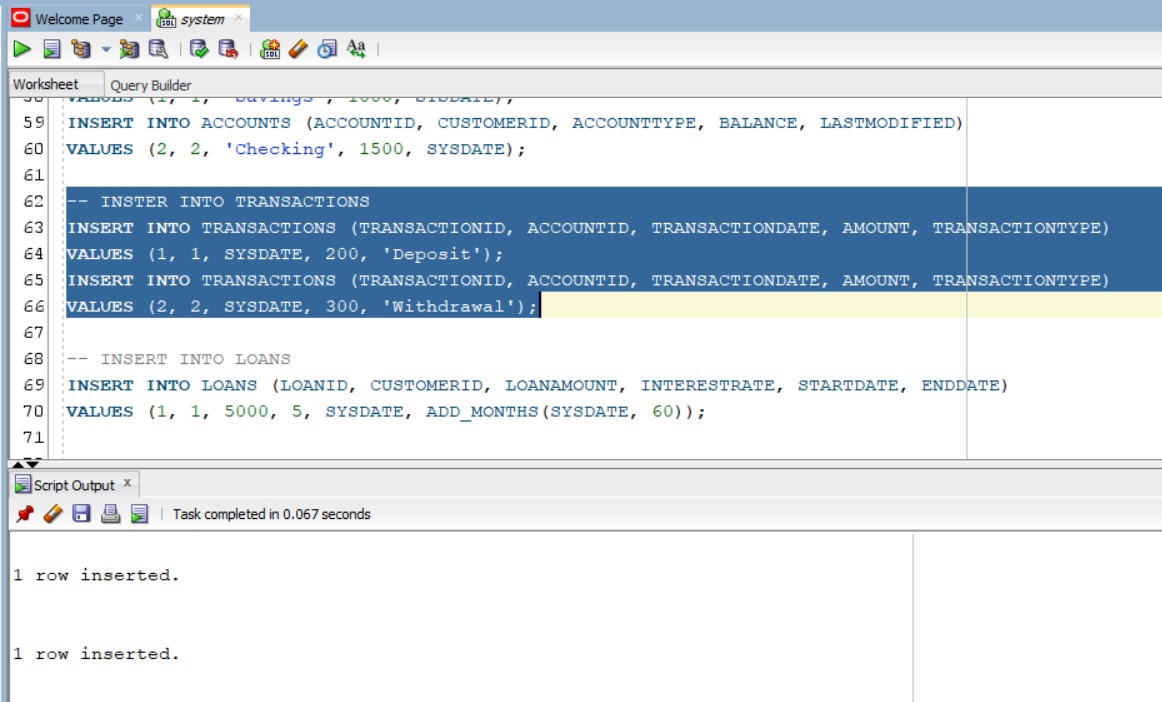
**-- INSTER INTO TRANSACTIONS**

INSERT INTO TRANSACTIONS (TRANSACTIONID, ACCOUNTID, TRANSACTIONDATE, AMOUNT, TRANSACTIONTYPE)

VALUES (1, 1, SYSDATE, 200, 'Deposit');

INSERT INTO TRANSACTIONS (TRANSACTIONID, ACCOUNTID, TRANSACTIONDATE, AMOUNT, TRANSACTIONTYPE)

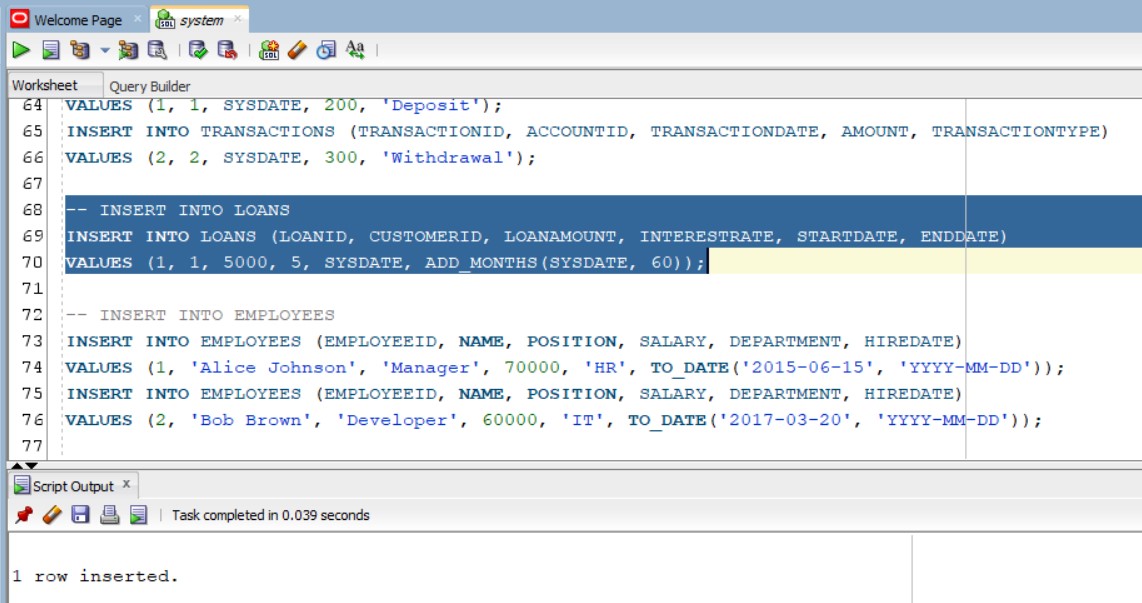
VALUES (2, 2, SYSDATE, 300, 'Withdrawal');



**-- INSERT INTO LOANS**

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

VALUES (1, 1, 5000, 5, SYSDATE, ADD\_MONTHS(SYSDATE, 60));



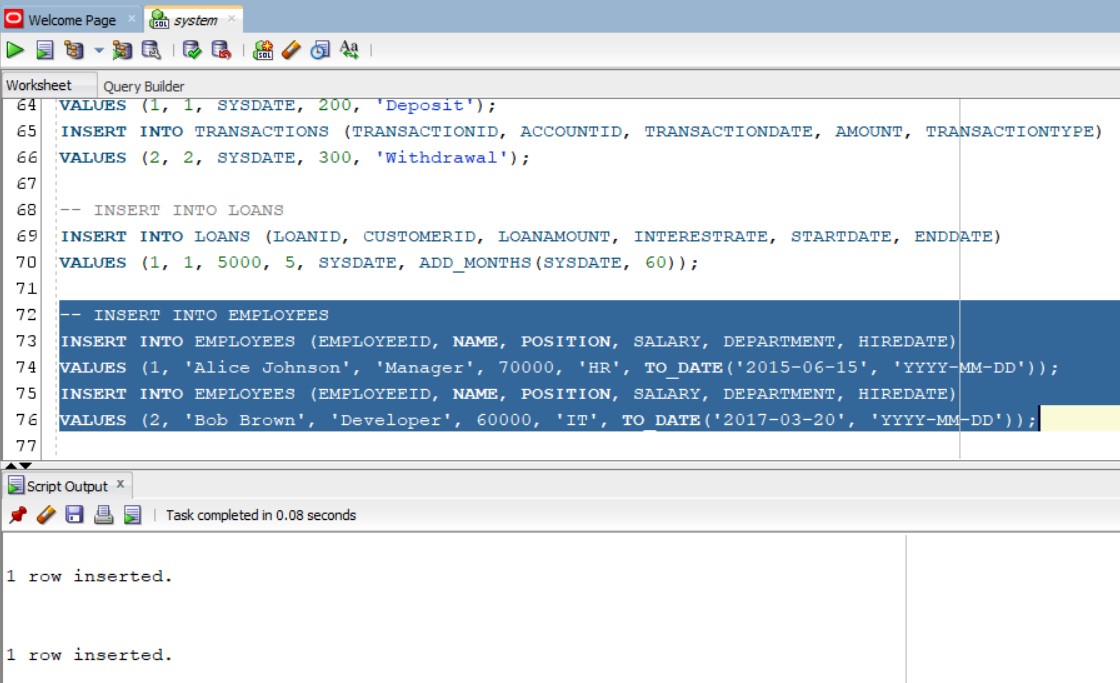
**-- INSERT INTO EMPLOYEES**

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'));



**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;

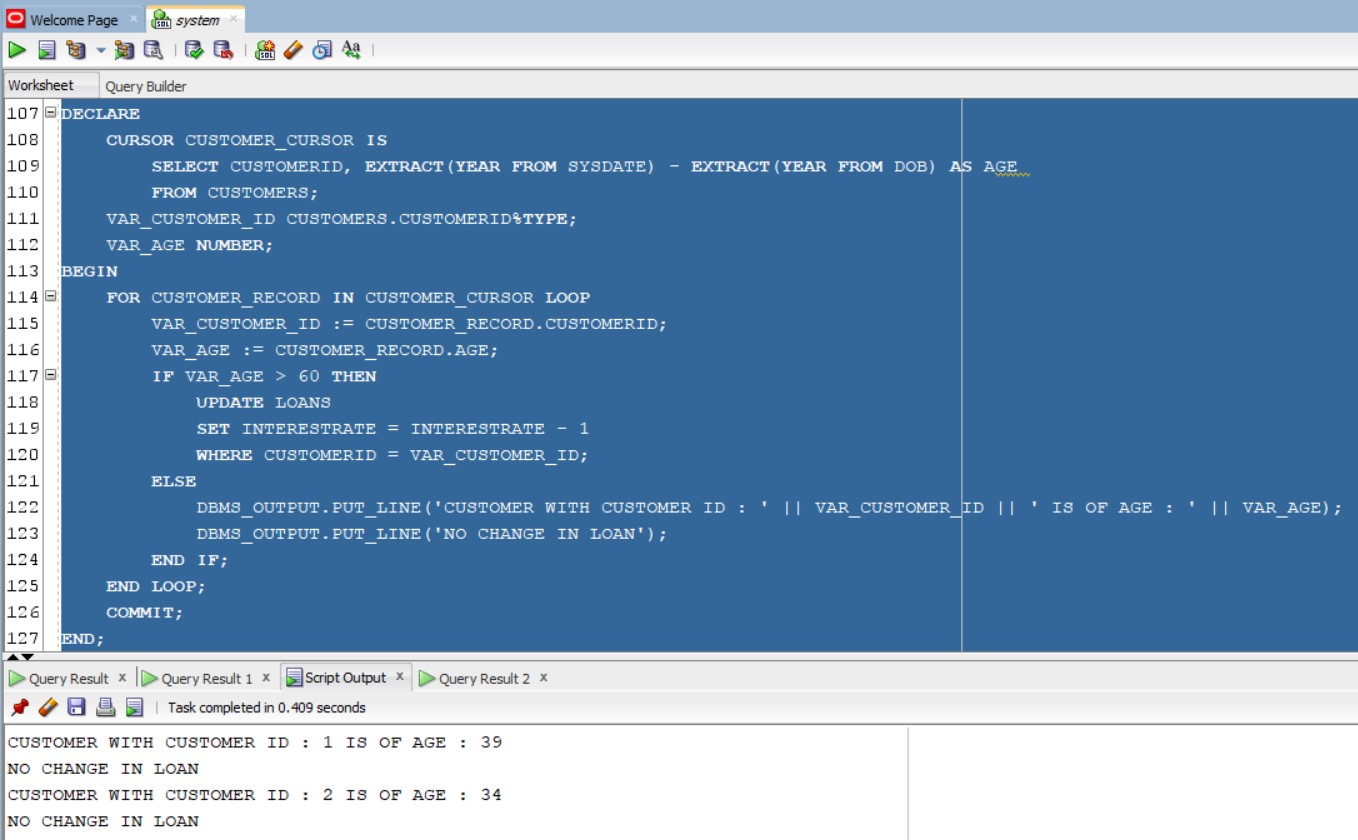
END LOOP;

COMMIT;

END;

/

SELECT \* FROM LOANS;



**-- 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;

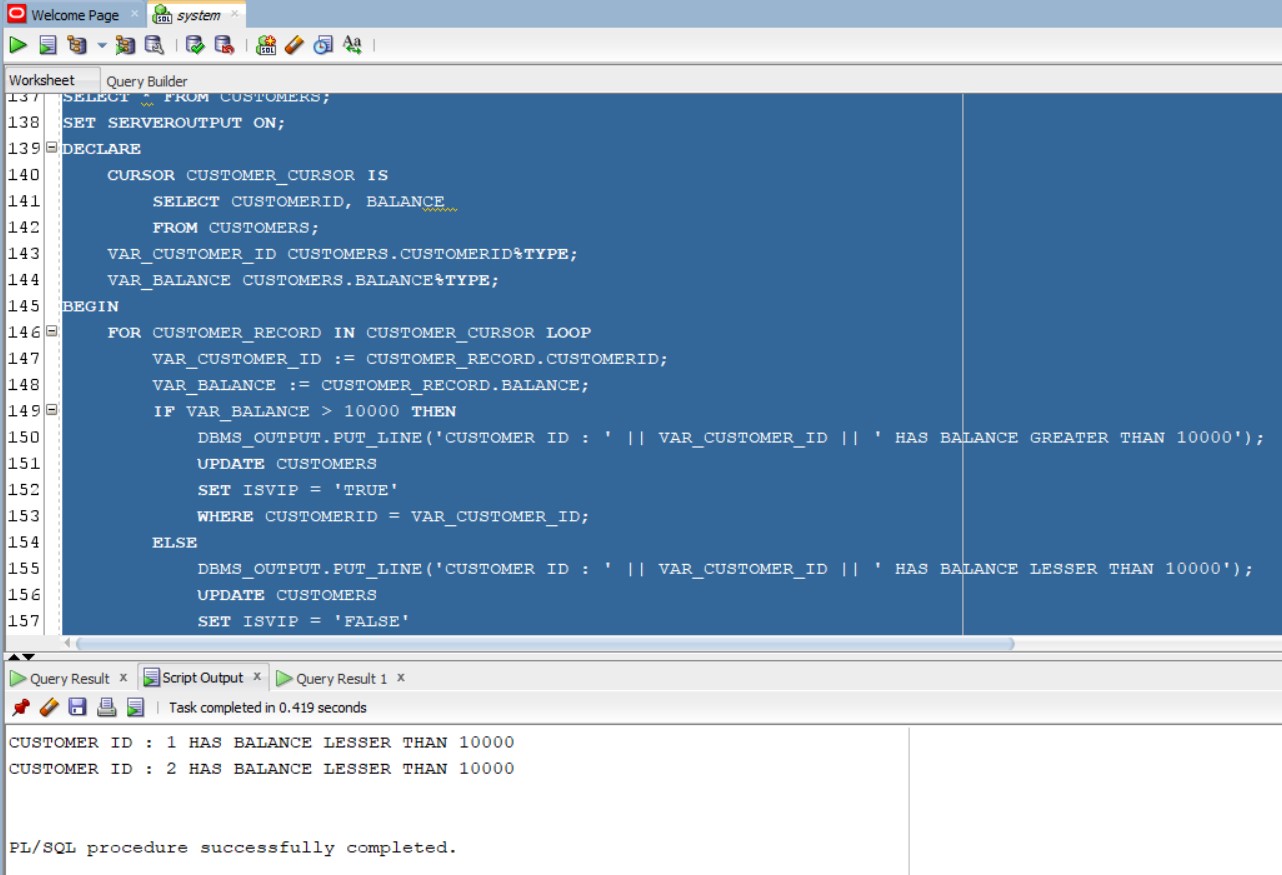
END LOOP;

COMMIT;

END;

/

SELECT \* FROM CUSTOMERS;



**-- 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;

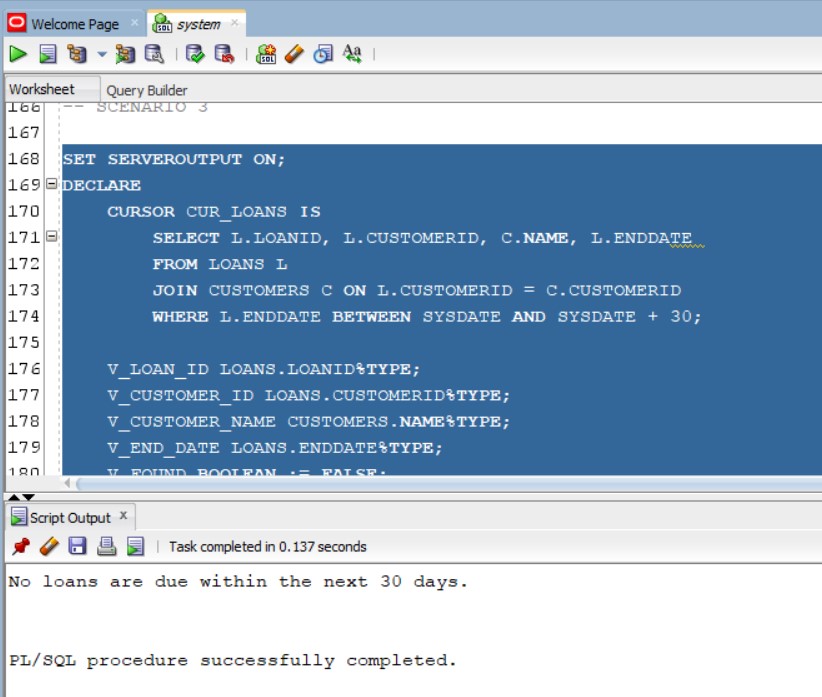
IF NOT V\_FOUND THEN

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

END IF;

END;

/



**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;

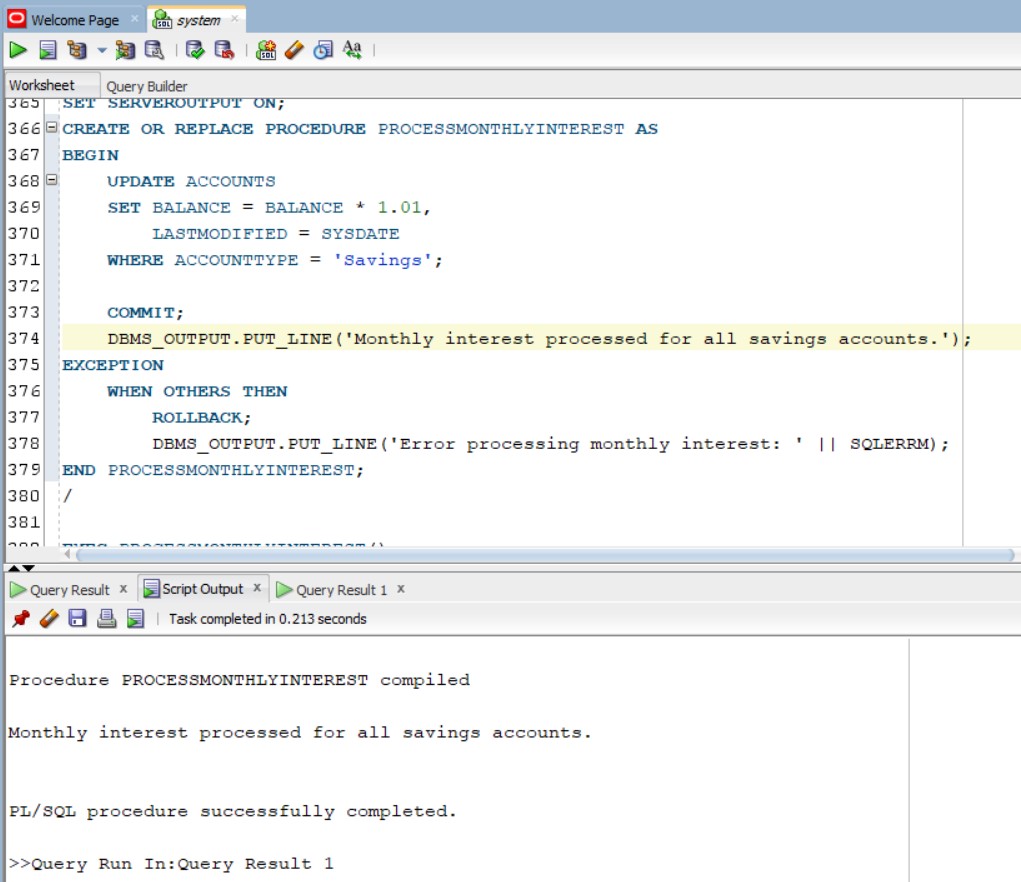
DBMS\_OUTPUT.PUT\_LINE('Error processing monthly interest: ' || SQLERRM);

END PROCESSMONTHLYINTEREST;

/

EXEC PROCESSMONTHLYINTEREST();

SELECT \* FROM ACCOUNTS;



**-- 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);

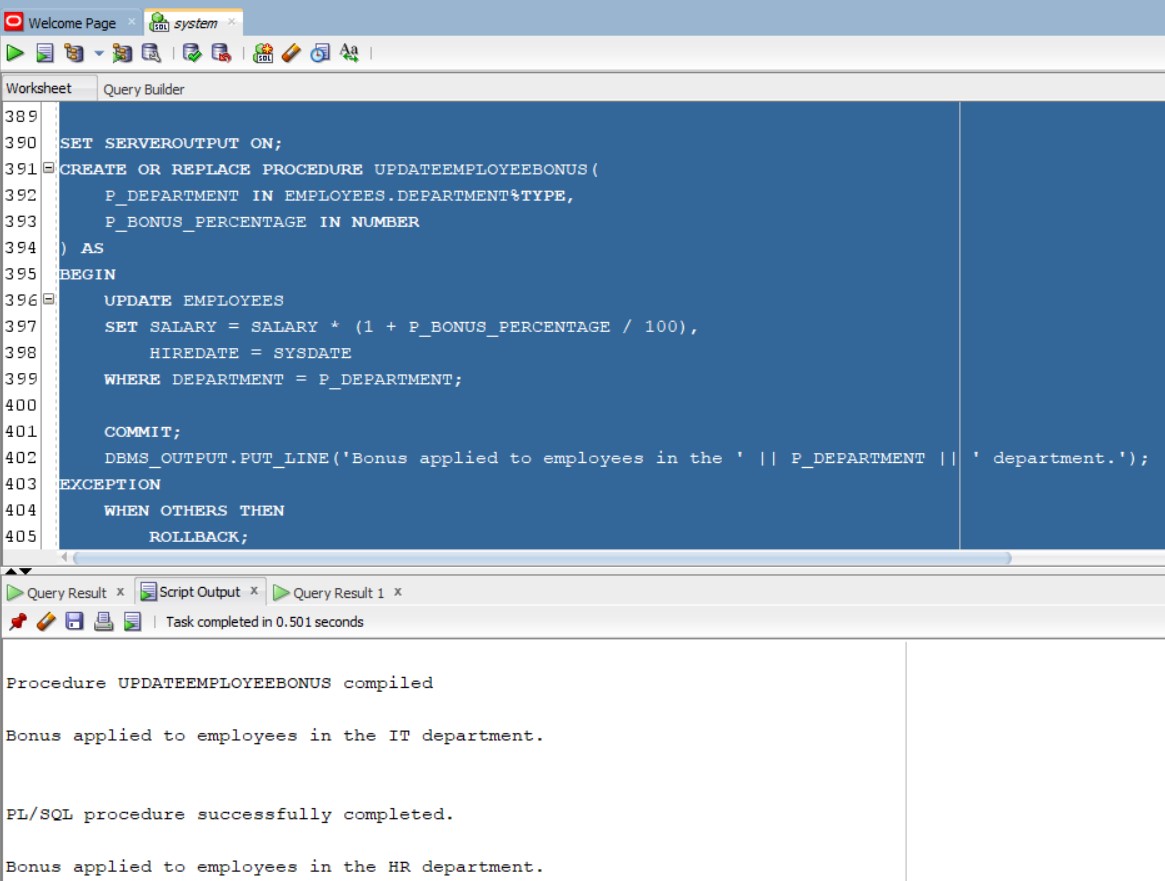
END UPDATEEMPLOYEEBONUS;

/

EXEC UPDATEEMPLOYEEBONUS('IT',5);

EXEC UPDATEEMPLOYEEBONUS('HR',3);

SELECT \* FROM EMPLOYEES;



**-- 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;

