**Procedural Language /SQL**

**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.

DECLARE

v\_age NUMBER;

v\_original\_rate NUMBER;

v\_new\_rate NUMBER;

BEGIN

DBMS\_OUTPUT.PUT\_LINE(

RPAD('Status', 18) ||

RPAD('Customer Name', 20) ||

RPAD('Loan ID', 10) ||

RPAD('Age', 6) ||

RPAD('Old Rate', 12) ||

RPAD('New Rate', 10)

);

FOR cust\_rec IN (

SELECT

c.CustomerID,

c.Name AS CustomerName,

c.DOB,

l.LoanID,

l.InterestRate

FROM Customers c

JOIN Loans l ON c.CustomerID = l.CustomerID

) LOOP

v\_age := TRUNC(MONTHS\_BETWEEN(SYSDATE, cust\_rec.DOB) / 12);

v\_original\_rate := cust\_rec.InterestRate;

IF v\_age > 60 THEN

v\_new\_rate := v\_original\_rate - (v\_original\_rate \* 0.01);

UPDATE Loans

SET InterestRate = v\_new\_rate

WHERE LoanID = cust\_rec.LoanID;

DBMS\_OUTPUT.PUT\_LINE(

RPAD('Discount Applied', 18) ||

RPAD(cust\_rec.CustomerName, 20) ||

RPAD(cust\_rec.LoanID, 10) ||

RPAD(v\_age, 6) ||

RPAD(TO\_CHAR(v\_original\_rate, '90.00') || '%', 10) ||

TO\_CHAR(v\_new\_rate, '90.00') || '%'

);

ELSE

DBMS\_OUTPUT.PUT\_LINE(

RPAD('No Discount', 18) ||

RPAD(cust\_rec.CustomerName, 20) ||

RPAD(cust\_rec.LoanID, 10) ||

RPAD(v\_age, 6) ||

RPAD(TO\_CHAR(v\_original\_rate, '90.00') || '%', 10)

);

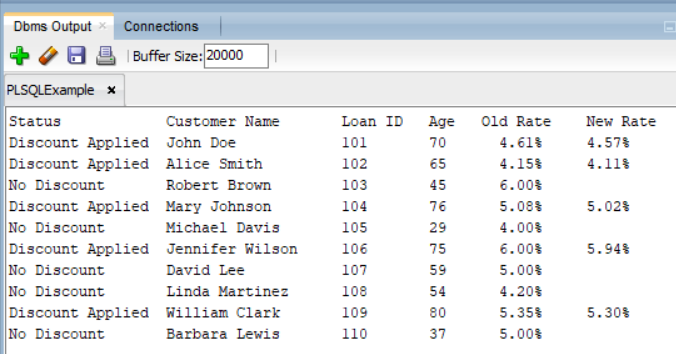
END IF;

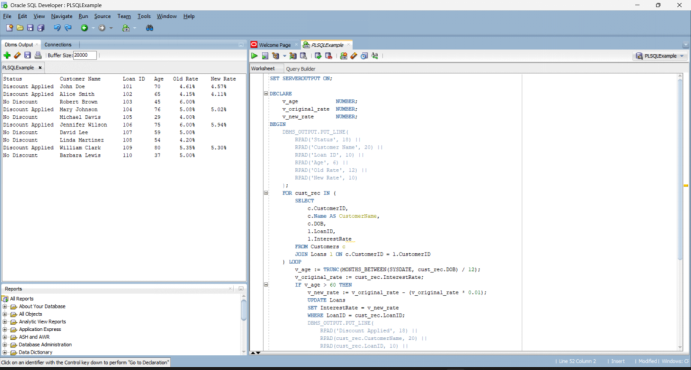
END LOOP;

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.

SET SERVEROUTPUT ON;

ALTER TABLE Customers ADD IsVIP VARCHAR2(5);

BEGIN

DBMS\_OUTPUT.PUT\_LINE(

RPAD('Customer ID', 15) ||

RPAD('Balance', 15) ||

RPAD('VIP Status', 15)

);

DBMS\_OUTPUT.PUT\_LINE(RPAD('-', 45, '-'));

FOR cust\_rec IN (

SELECT CustomerID, Balance

FROM Customers

) LOOP

IF cust\_rec.Balance > 10000 THEN

UPDATE Customers

SET IsVIP = 'TRUE'

WHERE CustomerID = cust\_rec.CustomerID;

DBMS\_OUTPUT.PUT\_LINE(

RPAD(cust\_rec.CustomerID, 15) ||

RPAD(TO\_CHAR(cust\_rec.Balance, '9999990.00'), 15) ||

'TRUE'

);

ELSE

UPDATE Customers

SET IsVIP = 'FALSE'

WHERE CustomerID = cust\_rec.CustomerID;

DBMS\_OUTPUT.PUT\_LINE(

RPAD(cust\_rec.CustomerID, 15) ||

RPAD(TO\_CHAR(cust\_rec.Balance, '9999990.00'), 15) ||

'FALSE'

);

END IF;

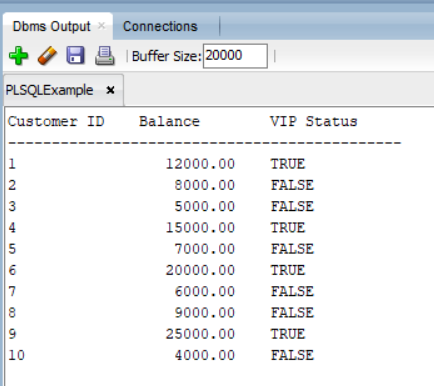
END LOOP;

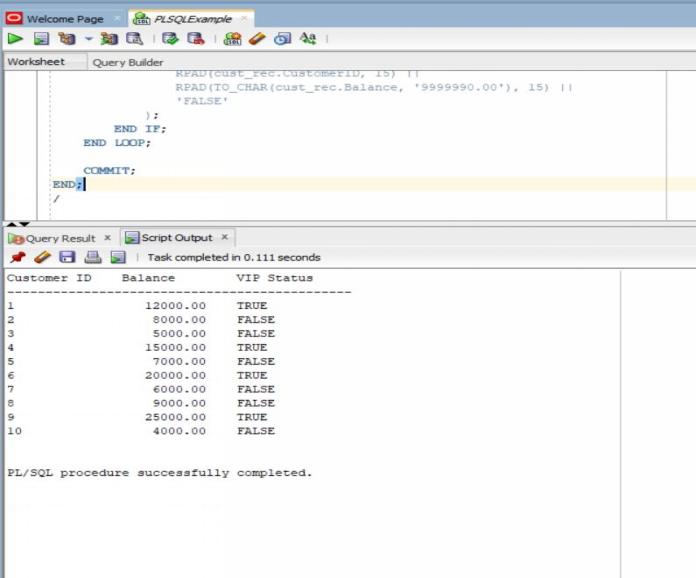
COMMIT;

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.

SET SERVEROUTPUT ON;

DECLARE

v\_count NUMBER := 0;

BEGIN

FOR loan\_rec IN (

SELECT l.LoanID, c.Name AS CustomerName, c.CustomerID, l.EndDate

FROM Loans l

JOIN Customers c ON c.CustomerID = l.CustomerID

WHERE l.EndDate BETWEEN SYSDATE AND SYSDATE + 30

ORDER BY l.EndDate

) LOOP

v\_count := v\_count + 1;

DBMS\_OUTPUT.PUT\_LINE('Reminder: Dear ' || loan\_rec.CustomerName || ',');

DBMS\_OUTPUT.PUT\_LINE('Your loan (Loan ID: ' || loan\_rec.LoanID || ') is due for repayment on '||TO\_CHAR(loan\_rec.EndDate, 'FMMonth DD, YYYY')||'.');

DBMS\_OUTPUT.PUT\_LINE('Please ensure timely payment to avoid penalties.');

DBMS\_OUTPUT.PUT\_LINE(''); -- Blank line for separation

END LOOP;

IF v\_count = 0 THEN

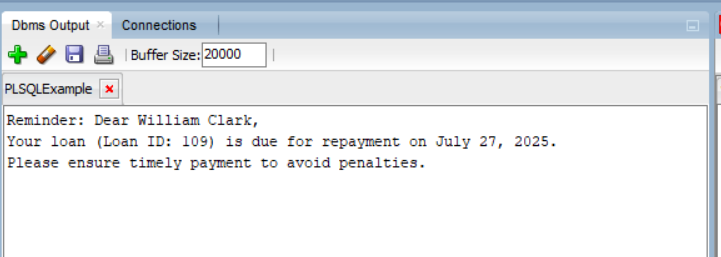
DBMS\_OUTPUT.PUT\_LINE('There are no loans due for repayment in the next 30 days.');

END IF;

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.

Set SERVEROUTPUT ON;

CREATE OR REPLACE PROCEDURE ProcessMonthlyInterest IS

v\_old\_balance NUMBER;

v\_new\_balance NUMBER;

v\_savings\_count NUMBER := 0;

v\_skipped\_count NUMBER := 0;

BEGIN

FOR acc\_rec IN (

SELECT AccountID, CustomerID, AccountType, Balance

FROM Accounts

) LOOP

IF acc\_rec.AccountType = 'Savings' THEN

v\_old\_balance := acc\_rec.Balance;

v\_new\_balance := v\_old\_balance + (v\_old\_balance \* 0.01);

UPDATE Accounts

SET Balance = v\_new\_balance,

LastModified = SYSDATE

WHERE AccountID = acc\_rec.AccountID;

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

DBMS\_OUTPUT.PUT\_LINE('Account ID: ' || acc\_rec.AccountID ||

' | Customer ID: ' || acc\_rec.CustomerID ||

' | Type: ' || acc\_rec.AccountType);

DBMS\_OUTPUT.PUT\_LINE('Old Balance: ' || TO\_CHAR(v\_old\_balance, '9999990.00') ||

' | New Balance: ' || TO\_CHAR(v\_new\_balance, '9999990.00'));

DBMS\_OUTPUT.PUT\_LINE('Status: Interest successfully applied');

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

ELSE

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

DBMS\_OUTPUT.PUT\_LINE('Account ID: ' || acc\_rec.AccountID ||

' | Customer ID: ' || acc\_rec.CustomerID ||

' | Type: ' || acc\_rec.AccountType);

DBMS\_OUTPUT.PUT\_LINE('Old Balance: ' || TO\_CHAR(acc\_rec.Balance, '9999990.00') ||

' | New Balance: Not Applicable');

DBMS\_OUTPUT.PUT\_LINE('Status: Skipped - Not a savings account');

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

END IF;

END LOOP;

DBMS\_OUTPUT.PUT\_LINE(v\_savings\_count || ' savings account(s) updated.');

DBMS\_OUTPUT.PUT\_LINE(v\_skipped\_count || ' account(s) skipped.');

END;

/

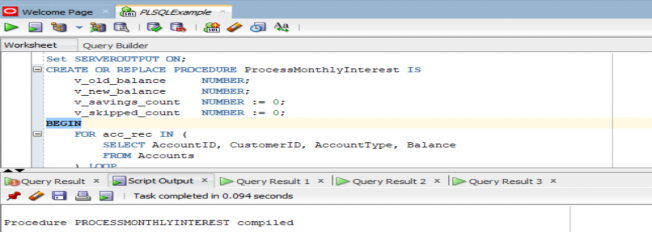
BEGIN

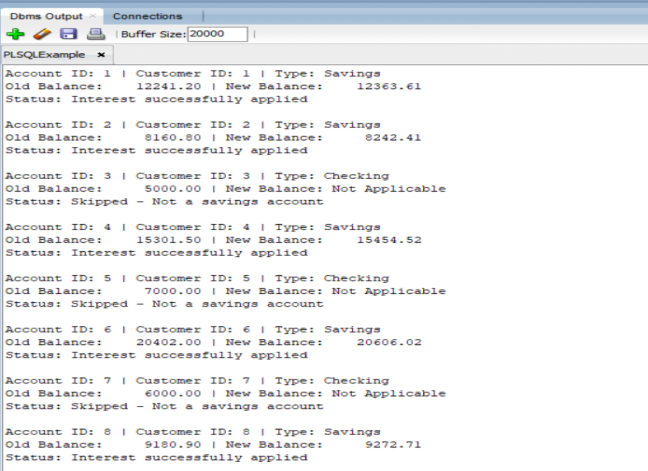
PROCESSMONTHLYINTEREST;

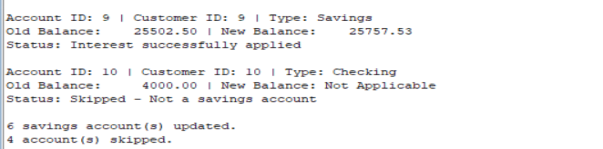
END;

/

**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.

Set SERVEROUTPUT ON;

CREATE OR REPLACE PROCEDURE UpdateEmployeeBonus(

p\_Department IN VARCHAR2,

p\_BonusPercent IN NUMBER

) IS

v\_old\_salary NUMBER;

v\_new\_salary NUMBER;

v\_count NUMBER := 0;

BEGIN

FOR emp\_rec IN (

SELECT EmployeeID, Name, Department, Salary

FROM Employees

WHERE Department = p\_Department

) LOOP

v\_old\_salary := emp\_rec.Salary;

v\_new\_salary := v\_old\_salary + (v\_old\_salary \* p\_BonusPercent / 100);

UPDATE Employees

SET Salary = v\_new\_salary

WHERE EmployeeID = emp\_rec.EmployeeID;

v\_count := v\_count + 1;

DBMS\_OUTPUT.PUT\_LINE('Employee ID : ' || emp\_rec.EmployeeID ||

' | Department: ' || emp\_rec.Department);

DBMS\_OUTPUT.PUT\_LINE('Old Salary : ' || TO\_CHAR(v\_old\_salary, '9999990.00') ||

' | New Salary: ' || TO\_CHAR(v\_new\_salary, '9999990.00'));

DBMS\_OUTPUT.PUT\_LINE('Bonus Applied : ' || p\_BonusPercent || '%');

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

END LOOP;

IF v\_count = 0 THEN

DBMS\_OUTPUT.PUT\_LINE('No employees found in department "' || p\_Department || '".');

ELSE

DBMS\_OUTPUT.PUT\_LINE(v\_count || ' employee(s) in department "' || p\_Department || '" received a ' || p\_BonusPercent || '% bonus.');

END IF;

END;

/

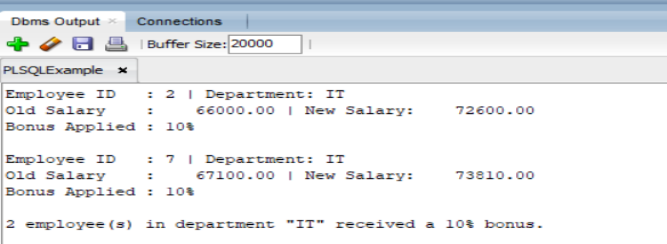
BEGIN

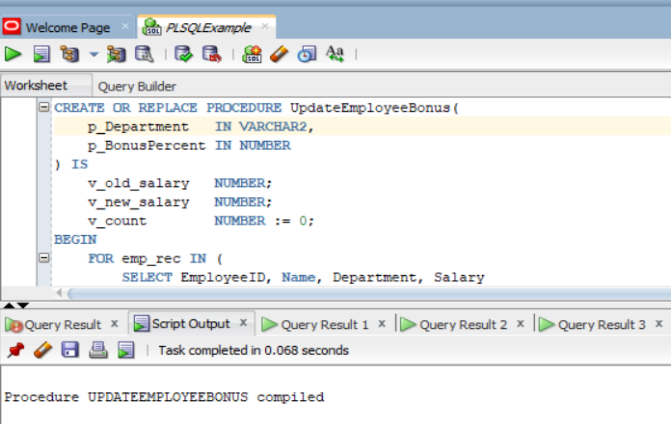
UpdateEmployeeBonus('IT', 10); -- Apply 10% bonus to IT department

END;

/

**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.

SET SERVEROUTPUT ON;

CREATE OR REPLACE PROCEDURE TransferFunds(

p\_source\_account\_id IN NUMBER,

p\_destination\_account\_id IN NUMBER,

p\_amount IN NUMBER

) IS

v\_source\_balance\_before NUMBER;

v\_destination\_balance\_before NUMBER;

v\_source\_balance\_after NUMBER;

v\_destination\_balance\_after NUMBER;

BEGIN

IF p\_source\_account\_id = p\_destination\_account\_id THEN

RAISE\_APPLICATION\_ERROR(-20003, 'Source and destination accounts must be different.');

END IF;

SELECT Balance INTO v\_source\_balance\_before

FROM Accounts

WHERE AccountID = p\_source\_account\_id

FOR UPDATE;

SELECT Balance INTO v\_destination\_balance\_before

FROM Accounts

WHERE AccountID = p\_destination\_account\_id

FOR UPDATE;

IF v\_source\_balance\_before < p\_amount THEN

RAISE\_APPLICATION\_ERROR(-20001, 'Insufficient balance in the source account.');

END IF;

UPDATE Accounts

SET Balance = Balance - p\_amount,

LastModified = SYSDATE

WHERE AccountID = p\_source\_account\_id;

UPDATE Accounts

SET Balance = Balance + p\_amount,

LastModified = SYSDATE

WHERE AccountID = p\_destination\_account\_id;

SELECT Balance INTO v\_source\_balance\_after

FROM Accounts

WHERE AccountID = p\_source\_account\_id;

SELECT Balance INTO v\_destination\_balance\_after

FROM Accounts

WHERE AccountID = p\_destination\_account\_id;

COMMIT;

DBMS\_OUTPUT.PUT\_LINE('Funds Transfer Summary');

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

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

DBMS\_OUTPUT.PUT\_LINE('Source Account ID : ' || p\_source\_account\_id);

DBMS\_OUTPUT.PUT\_LINE(' - Before Transfer : Rs. ' || v\_source\_balance\_before);

DBMS\_OUTPUT.PUT\_LINE(' - After Transfer : Rs. ' || v\_source\_balance\_after);

DBMS\_OUTPUT.PUT\_LINE('Destination Account ID : ' || p\_destination\_account\_id);

DBMS\_OUTPUT.PUT\_LINE(' - Before Transfer : Rs. ' || v\_destination\_balance\_before);

DBMS\_OUTPUT.PUT\_LINE(' - After Transfer : Rs. ' || v\_destination\_balance\_after);

DBMS\_OUTPUT.PUT\_LINE('Status : Transfer completed successfully');

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

EXCEPTION

WHEN NO\_DATA\_FOUND THEN

RAISE\_APPLICATION\_ERROR(-20002, 'One or both account IDs do not exist.');

WHEN OTHERS THEN

ROLLBACK;

RAISE;

END;

/

--Transfer done

BEGIN

TransferFunds(1, 2, 500);

END;

/

--Transfer not done- id doesn’t exists

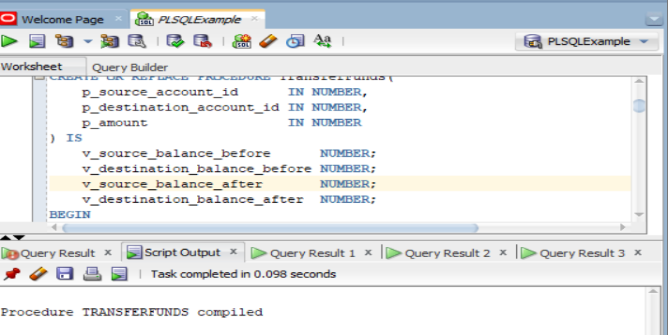
BEGIN

TransferFunds(101,102, 500);

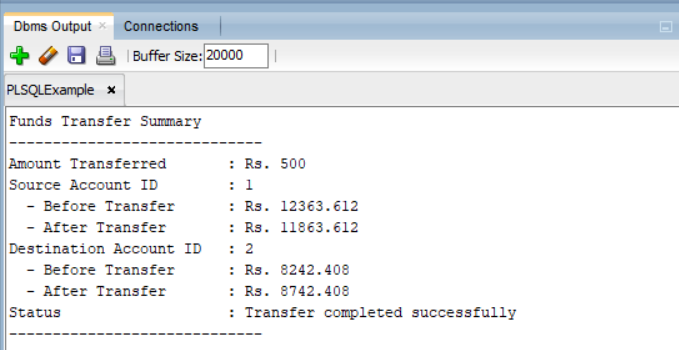
END;

/

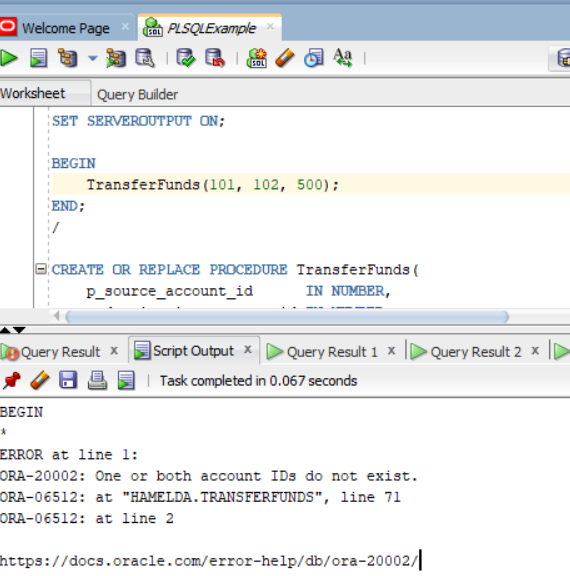
**Output:**



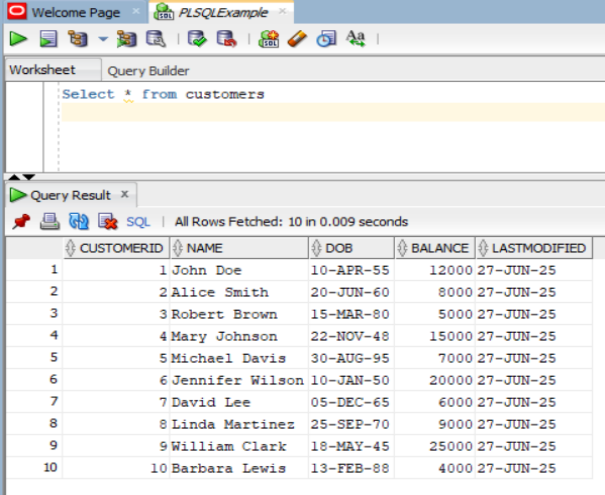
Source Account ID 1 and Destination Account ID 2 exists:



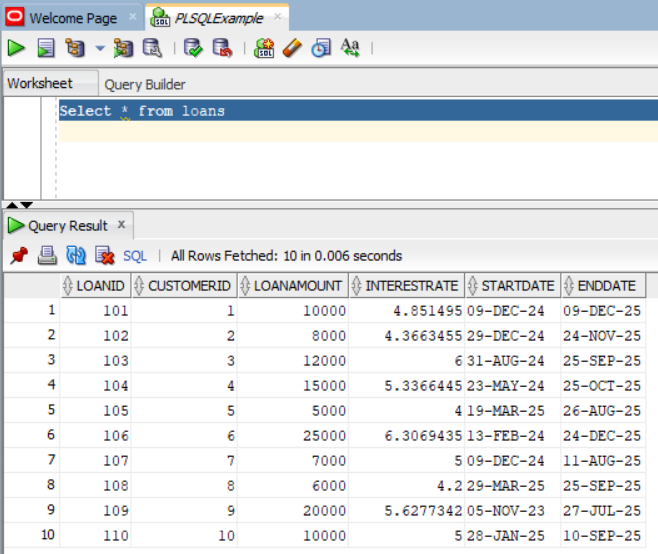
Source Account ID 101 and Destination Account ID 102 doesn’t exists:



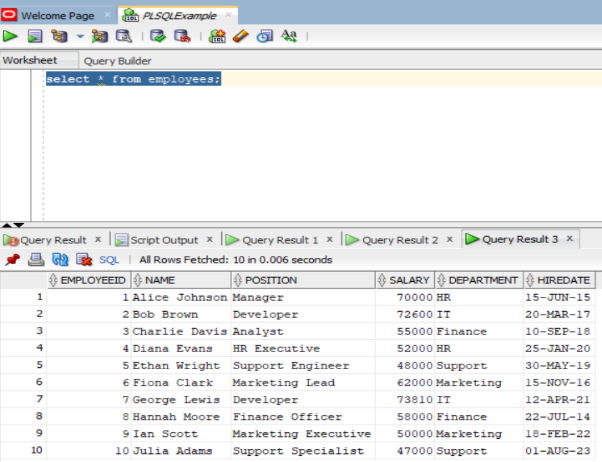
Customers data:



Loans data:



Employees data:



Accounts data:

