**“PL-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.**

***SOLUTION:***

BEGIN

FOR cust\_rec IN (

SELECT l.LoanID, l.InterestRate, c.DOB

FROM Loans l

JOIN Customers c ON l.CustomerID = c.CustomerID

) LOOP

-- Print initial status

DBMS\_OUTPUT.PUT\_LINE('Loan ID: ' || cust\_rec.LoanID ||

' | Original Interest Rate: ' || cust\_rec.InterestRate);

-- Check age

IF TRUNC(MONTHS\_BETWEEN(SYSDATE, cust\_rec.DOB) / 12) > 60 THEN

-- Apply 1% discount

UPDATE Loans

SET InterestRate = cust\_rec.InterestRate - (cust\_rec.InterestRate\*0.01)

WHERE LoanID = cust\_rec.LoanID;

-- Print updated status

DBMS\_OUTPUT.PUT\_LINE('→ Updated Interest Rate for Loan ID ' || cust\_rec.LoanID ||

' to ' || TO\_CHAR(cust\_rec.InterestRate - 1));

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

ELSE

DBMS\_OUTPUT.PUT\_LINE('→ No change (Customer under 60)');

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

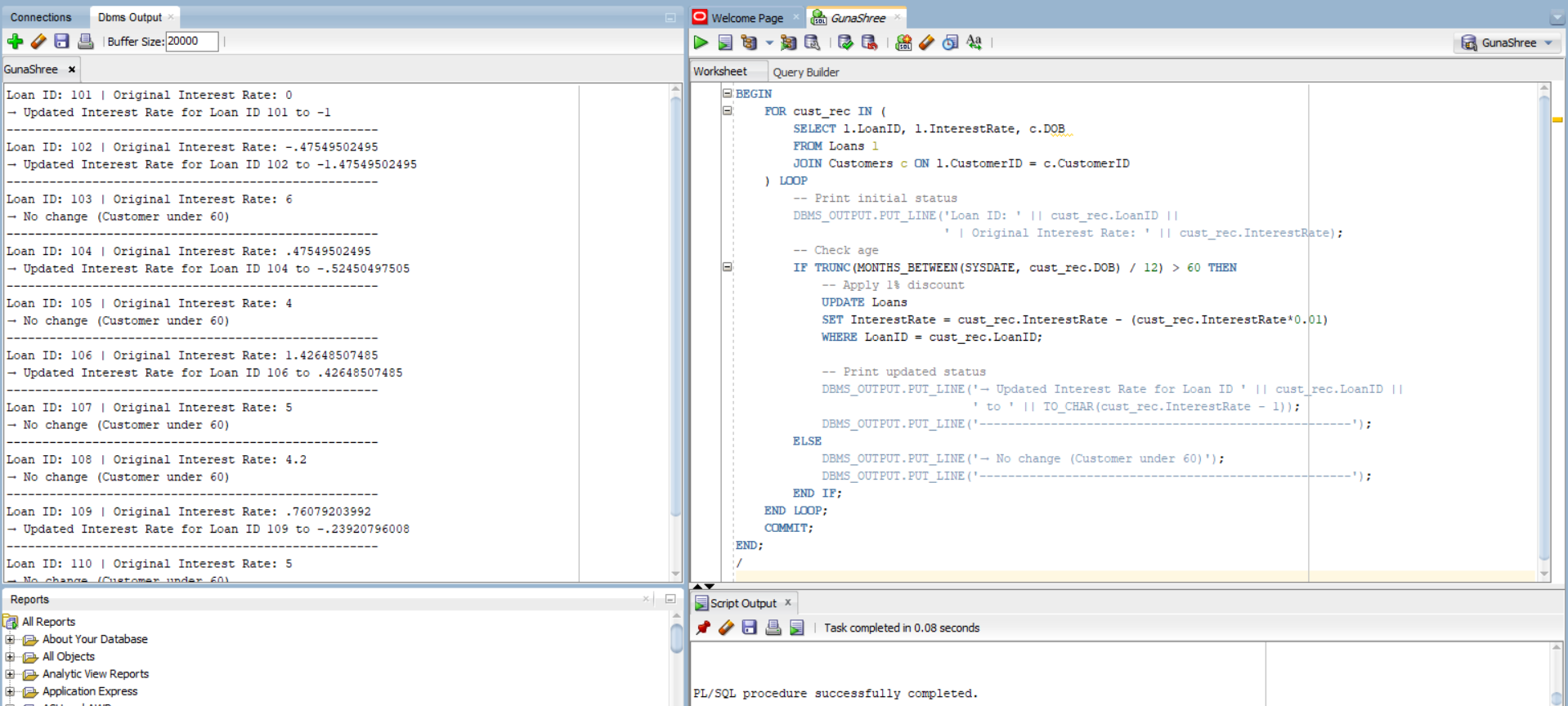
END IF;

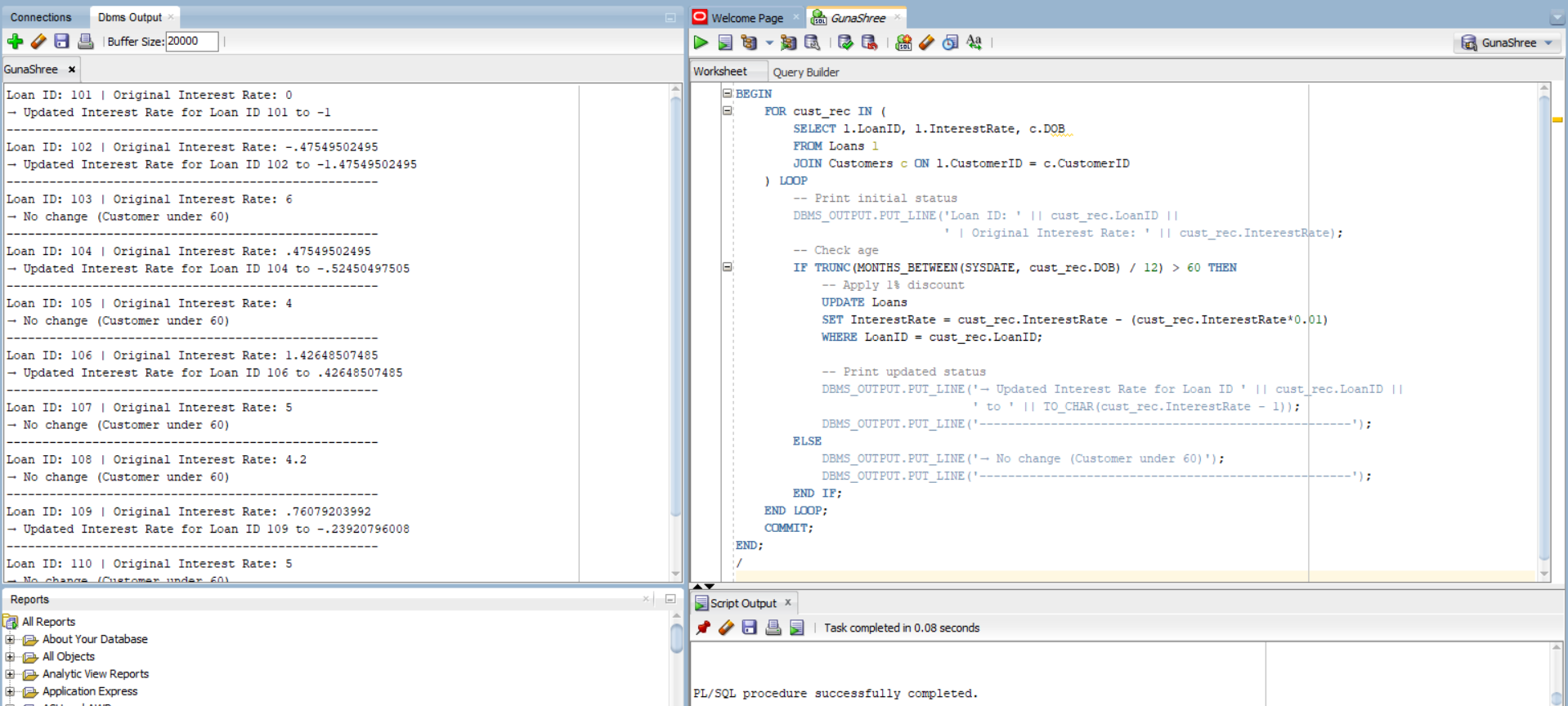
END LOOP;

COMMIT;

END;

/





**--------------------------------------------------------------------------------------------------------------------------------------**

**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.**

***SOLUTION:***

SET SERVEROUTPUT ON;

ALTER TABLE Customers ADD IsVIP VARCHAR2(5);

BEGIN

FOR cust\_rec IN (

SELECT CUSTOMERID, BALANCE

FROM CUSTOMERS

) LOOP

-- Print balance and VIP status before update

DBMS\_OUTPUT.PUT\_LINE('Customer ID: ' || cust\_rec.CustomerID ||

' | Balance: ' || cust\_rec.Balance);

IF cust\_rec.BALANCE > 10000 THEN

UPDATE CUSTOMERS

SET IsVIP = 'TRUE'

WHERE CUSTOMERID = cust\_rec.CUSTOMERID;

DBMS\_OUTPUT.PUT\_LINE('→ IsVIP status set to TRUE');

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

ELSE

UPDATE CUSTOMERS

SET IsVIP = 'FALSE'

WHERE CustomerID = cust\_rec.CUSTOMERID;

DBMS\_OUTPUT.PUT\_LINE('→ IsVIP status set to FALSE');

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

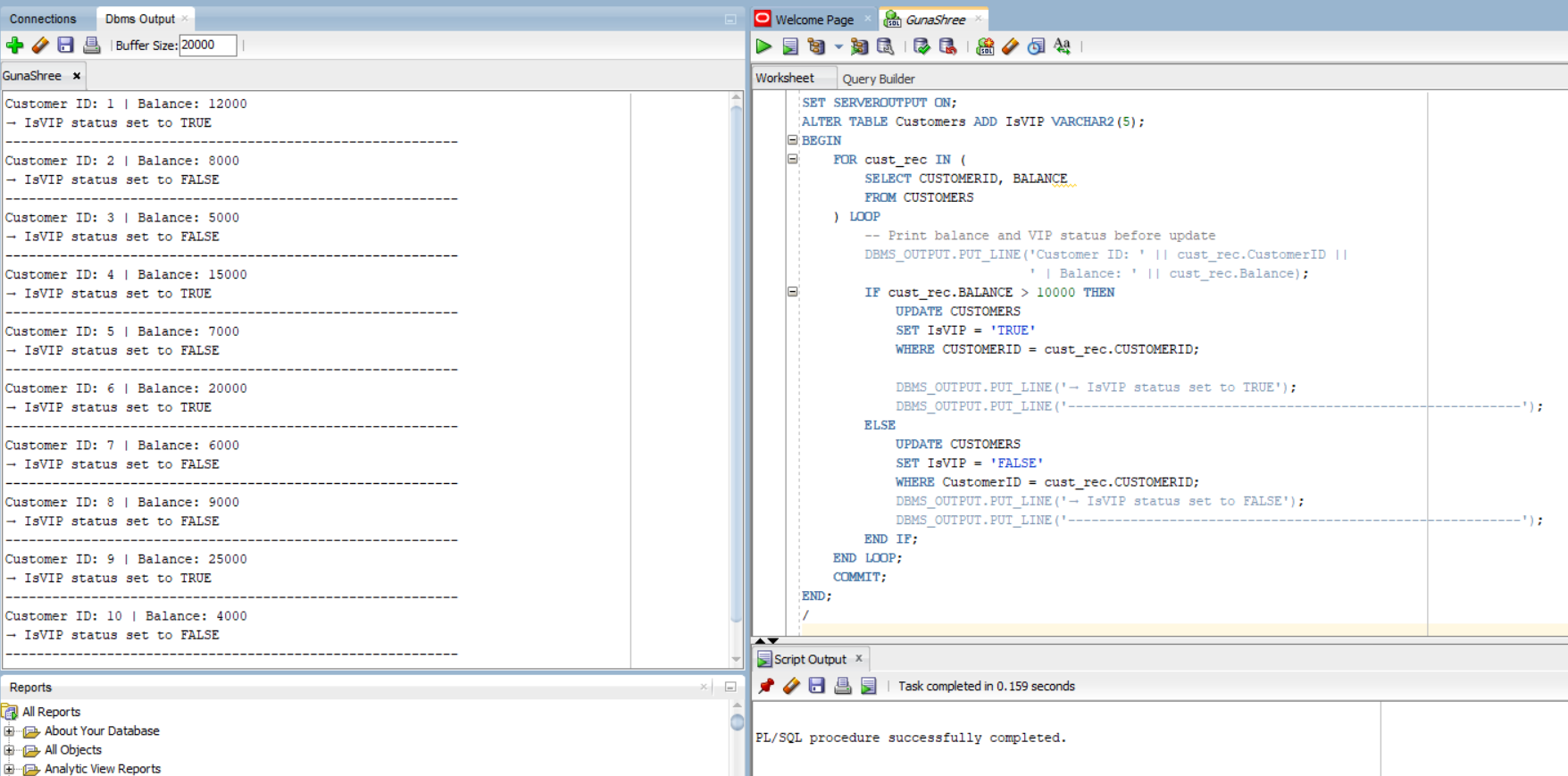
END IF;

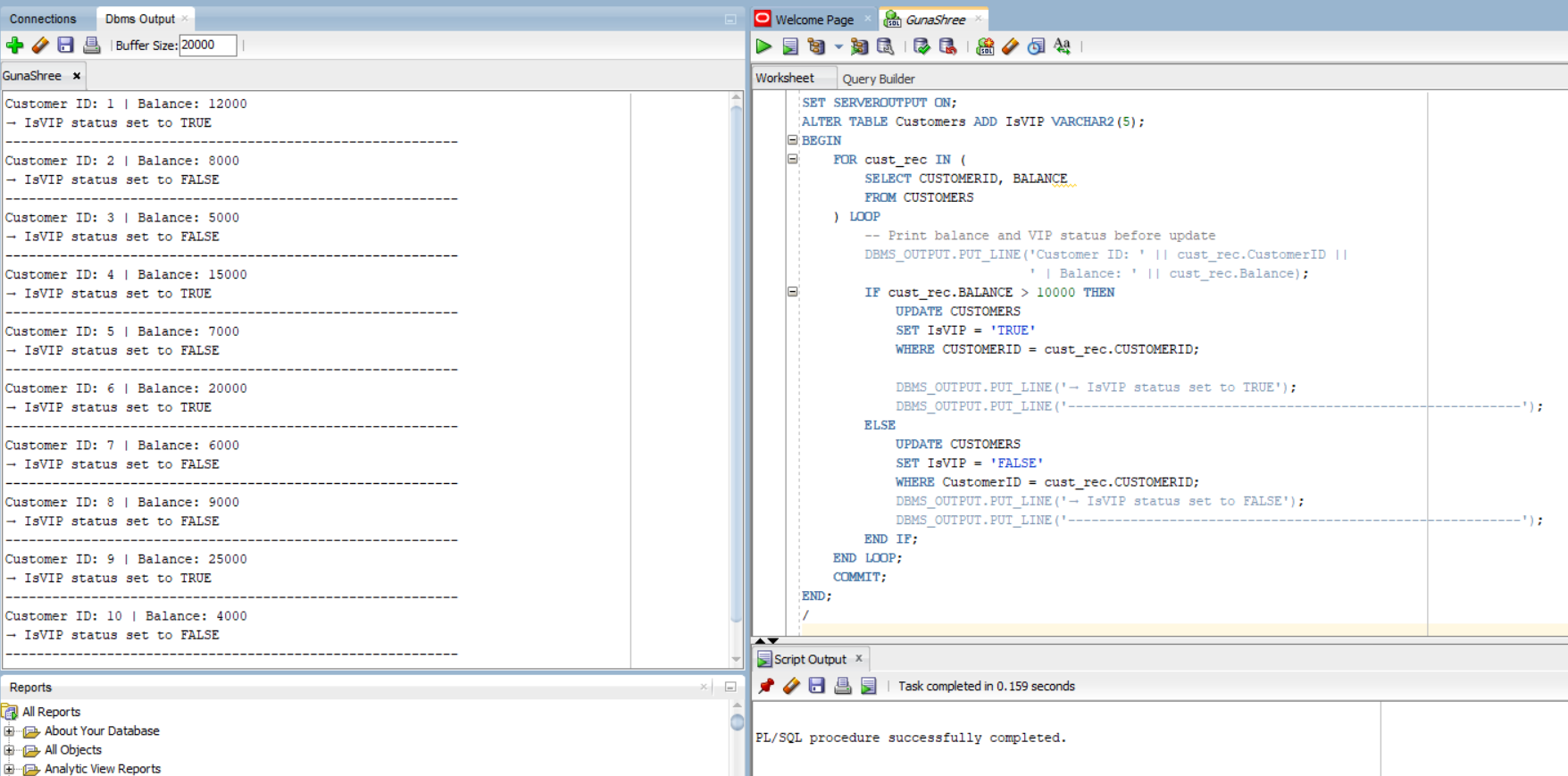
END LOOP;

COMMIT;

END;

/





**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.**

***SOLUTION:***

BEGIN

FOR rec IN (

SELECT l.LoanID, l.EndDate, c.Name FROM Loans l

JOIN Customers c ON l.CustomerID = c.CustomerID

WHERE l.EndDate BETWEEN SYSDATE AND SYSDATE + 30

) LOOP

DBMS\_OUTPUT.PUT\_LINE(

'Reminder: Dear ' || rec.Name || ', your loan (Loan ID: ' || rec.LoanID ||

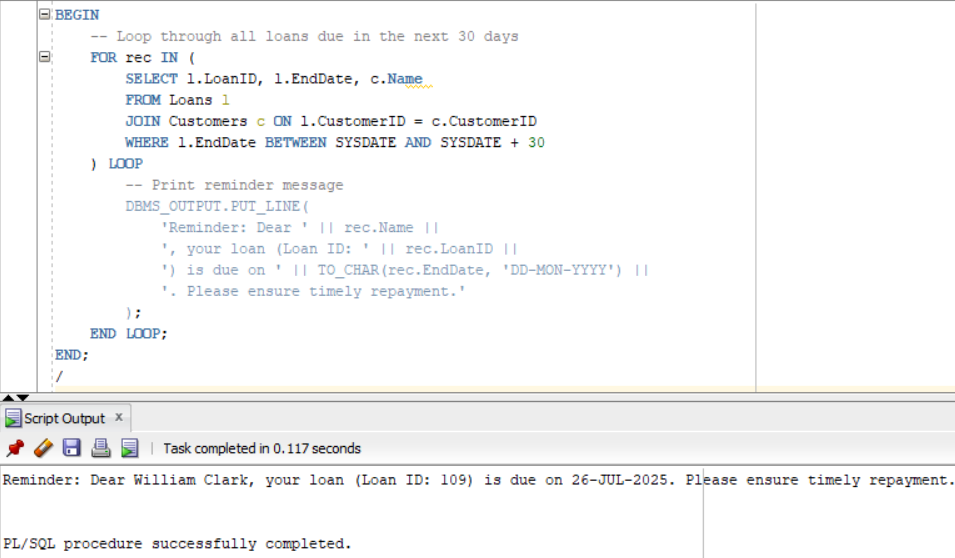
') is due on ' || TO\_CHAR(rec.EndDate, 'DD-MON-YYYY') ||

'. Please ensure timely repayment.' );

END LOOP;

END;

/



---------------------------------------------------------------------------------------------------------------------------

**“PL-SQL -- 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.**

***SOLUTION:***

SET SERVEROUTPUT ON;

CREATE OR REPLACE PROCEDURE ProcessMonthlyInterest AS

v\_new\_balance NUMBER;

BEGIN

DBMS\_OUTPUT.PUT\_LINE('--- Monthly Interest Processing Started ---');

-- Loop through all savings accounts

FOR acc\_rec IN (

SELECT AccountID, Balance

FROM ACCOUNTS

WHERE AccountType = 'Savings'

) LOOP

-- Calculate new balance

v\_new\_balance := acc\_rec.Balance + (acc\_rec.Balance \* 0.01);

-- Show before and after in output

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

DBMS\_OUTPUT.PUT\_LINE(' Before: Rs. ' || acc\_rec.Balance);

DBMS\_OUTPUT.PUT\_LINE(' After: Rs. ' || TO\_CHAR(v\_new\_balance, '9999990.00'));

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

-- Update the balance

UPDATE Accounts

SET Balance = v\_new\_balance,

LastModified = SYSDATE

WHERE AccountID = acc\_rec.AccountID;

END LOOP;

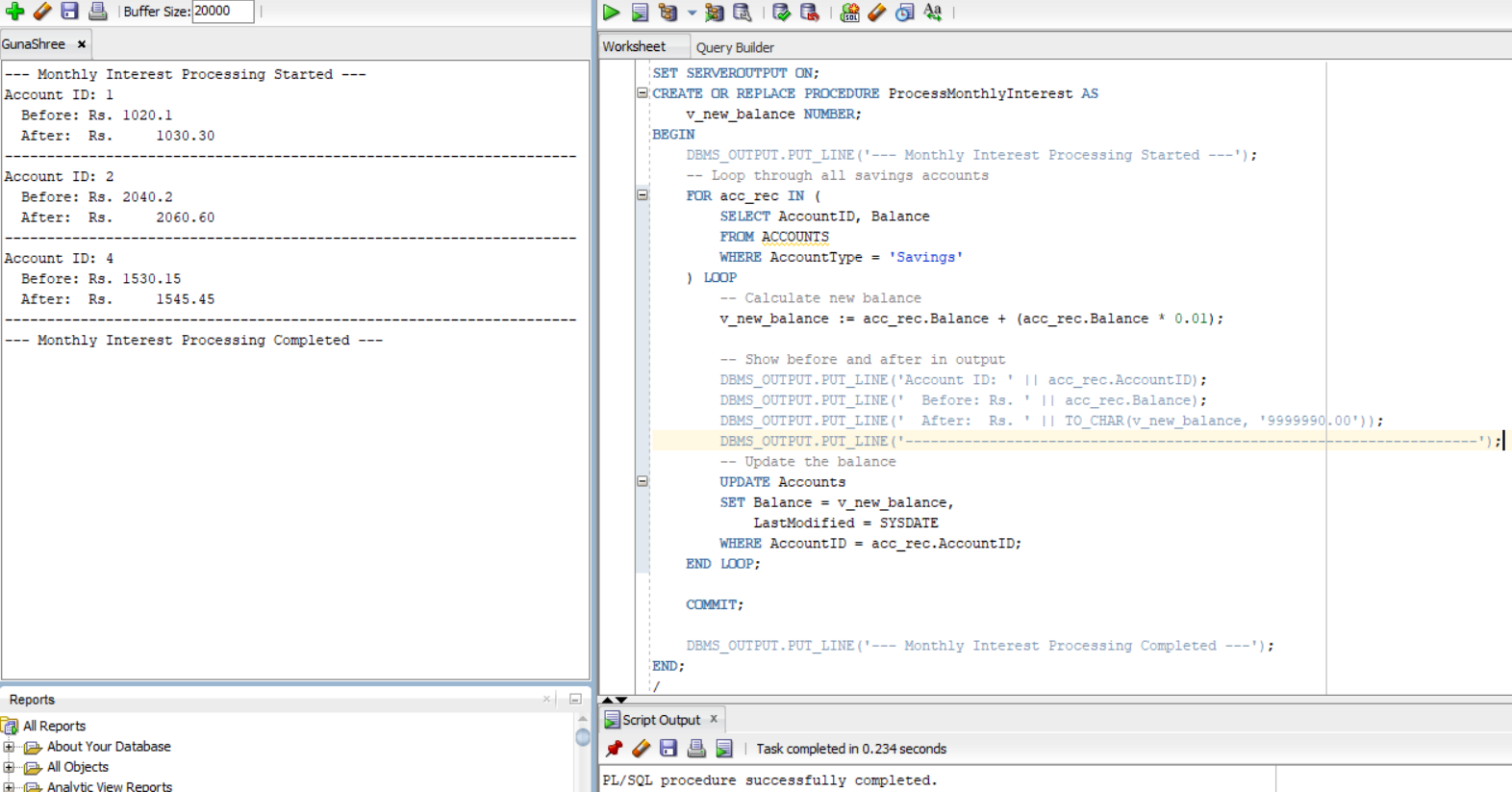
COMMIT;

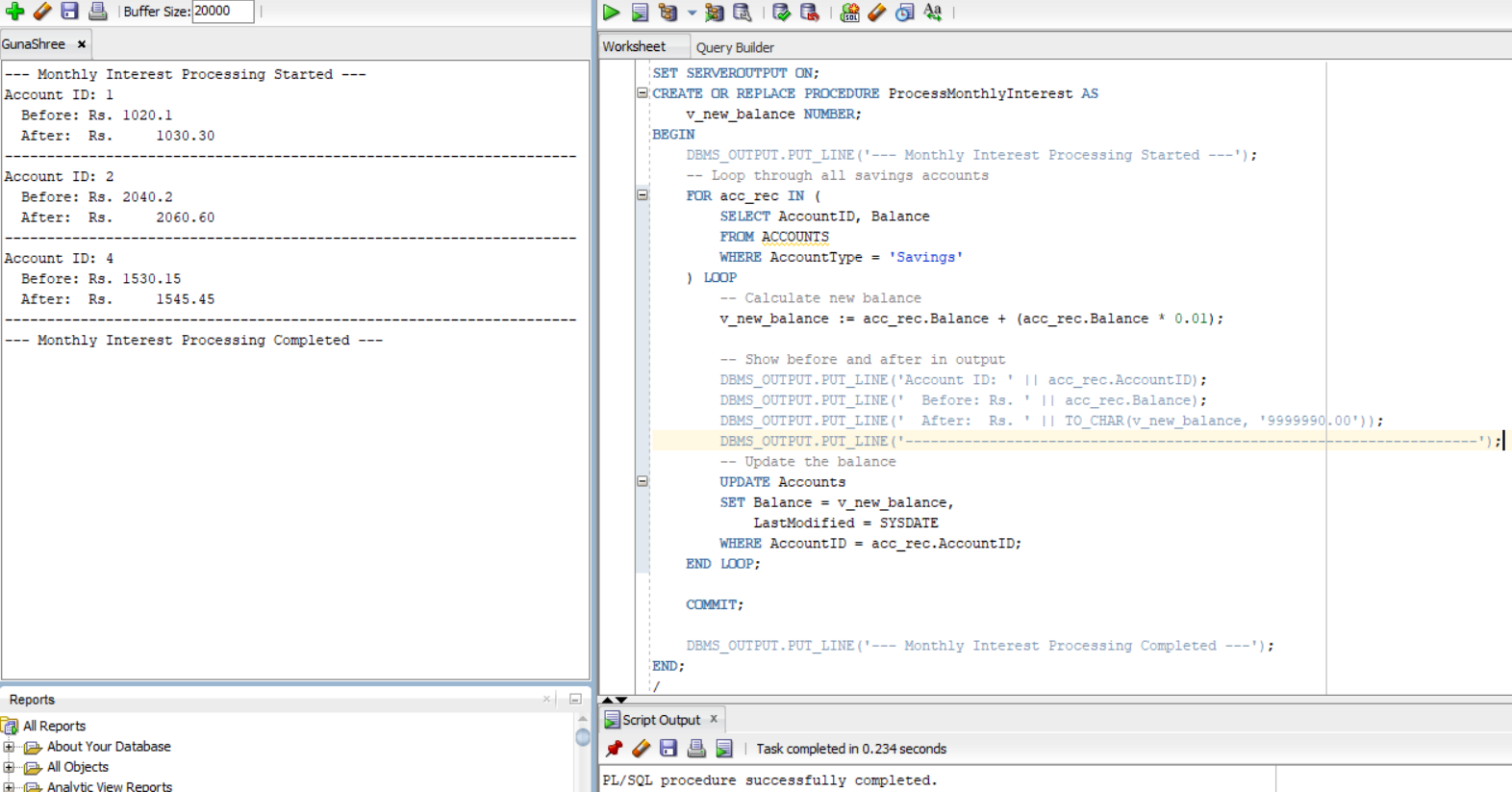
DBMS\_OUTPUT.PUT\_LINE('--- Monthly Interest Processing Completed ---');

END;

/

EXEC ProcessMonthlyInterest;





---------------------------------------------------------------------------------------------------------------------------

**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.**

***SOLUTION:***

SET SERVEROUTPUT ON;

CREATE OR REPLACE PROCEDURE UpdateEmployeeBonus (

p\_department IN VARCHAR2,

p\_bonus\_pct IN NUMBER -- e.g., pass 10 for 10%

) AS

v\_bonus\_amt NUMBER;

BEGIN

DBMS\_OUTPUT.PUT\_LINE('--- Bonus Update for Department: ' || p\_department || ' ---');

FOR emp\_rec IN (

SELECT EmployeeID, Department, Salary

FROM Employees

WHERE Department = p\_department

) LOOP

-- Calculate bonus

v\_bonus\_amt := emp\_rec.Salary \* (p\_bonus\_pct / 100);

-- Display ID, Dept, and Bonus

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

' | Dept: ' || emp\_rec.Department ||

' | Bonus: ' || TO\_CHAR(v\_bonus\_amt, '999999.00'));

-- Update salary

UPDATE Employees

SET Salary = Salary + v\_bonus\_amt

WHERE EmployeeID = emp\_rec.EmployeeID;

END LOOP;

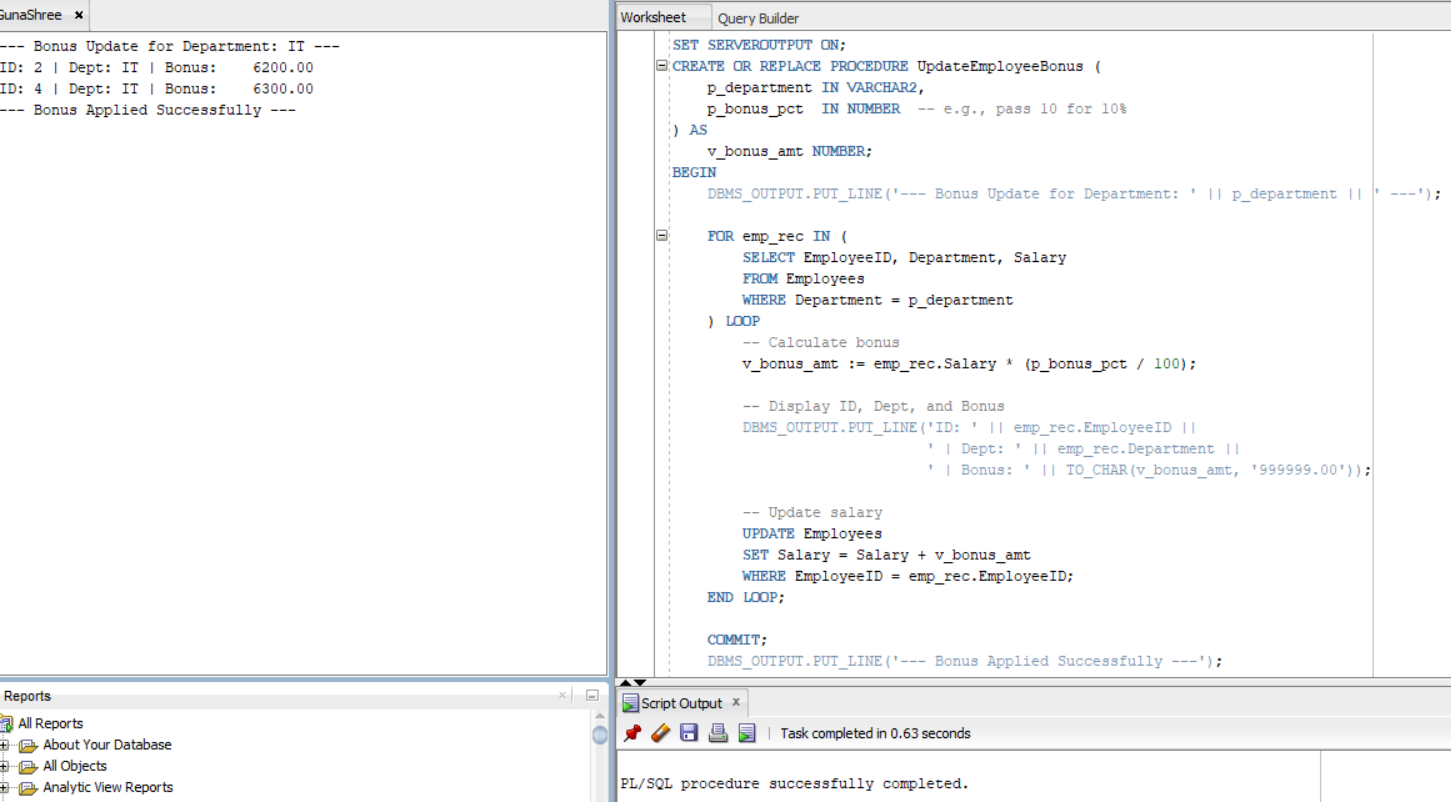
COMMIT;

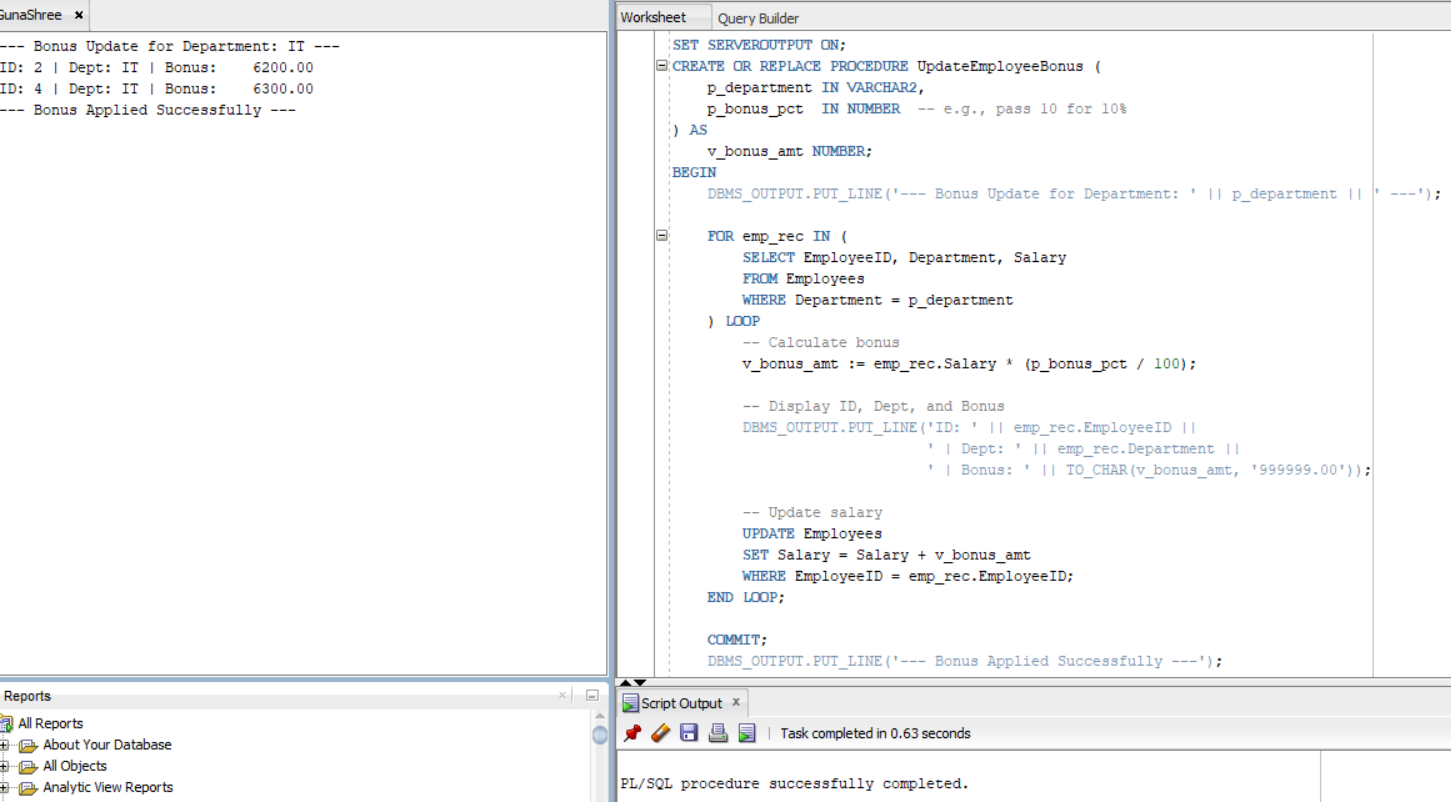
DBMS\_OUTPUT.PUT\_LINE('--- Bonus Applied Successfully ---');

END;

/

EXEC UpdateEmployeeBonus('IT',10);





---------------------------------------------------------------------------------------------------------------------------

**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.**

***SOLUTION:***

SET SERVEROUTPUT ON;

CREATE OR REPLACE PROCEDURE TransferFunds (

p\_source\_account\_id IN NUMBER,

p\_destination\_account\_id IN NUMBER,

p\_amount IN NUMBER

) AS

v\_source\_balance\_before NUMBER;

v\_destination\_balance\_before NUMBER;

v\_source\_balance\_after NUMBER;

v\_destination\_balance\_after NUMBER;

BEGIN

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

-- Step 1: Get balances before transfer

SELECT Balance INTO v\_source\_balance\_before

FROM Accounts

WHERE AccountID = p\_source\_account\_id;

SELECT Balance INTO v\_destination\_balance\_before

FROM Accounts

WHERE AccountID = p\_destination\_account\_id;

DBMS\_OUTPUT.PUT\_LINE('BEFORE TRANSFER:');

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

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

-- Step 2: Check if source account has sufficient balance

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

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

END IF;

-- Step 3: Perform the transfer

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;

-- Step 4: Get balances after transfer

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;

DBMS\_OUTPUT.PUT\_LINE('AFTER TRANSFER:');

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

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

COMMIT;

DBMS\_OUTPUT.PUT\_LINE('Transfer of Rs. ' || p\_amount || ' completed successfully.');

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

EXCEPTION

WHEN NO\_DATA\_FOUND THEN

RAISE\_APPLICATION\_ERROR(-20002, 'One of the accounts does not exist.');

WHEN OTHERS THEN

ROLLBACK;

RAISE;

END;

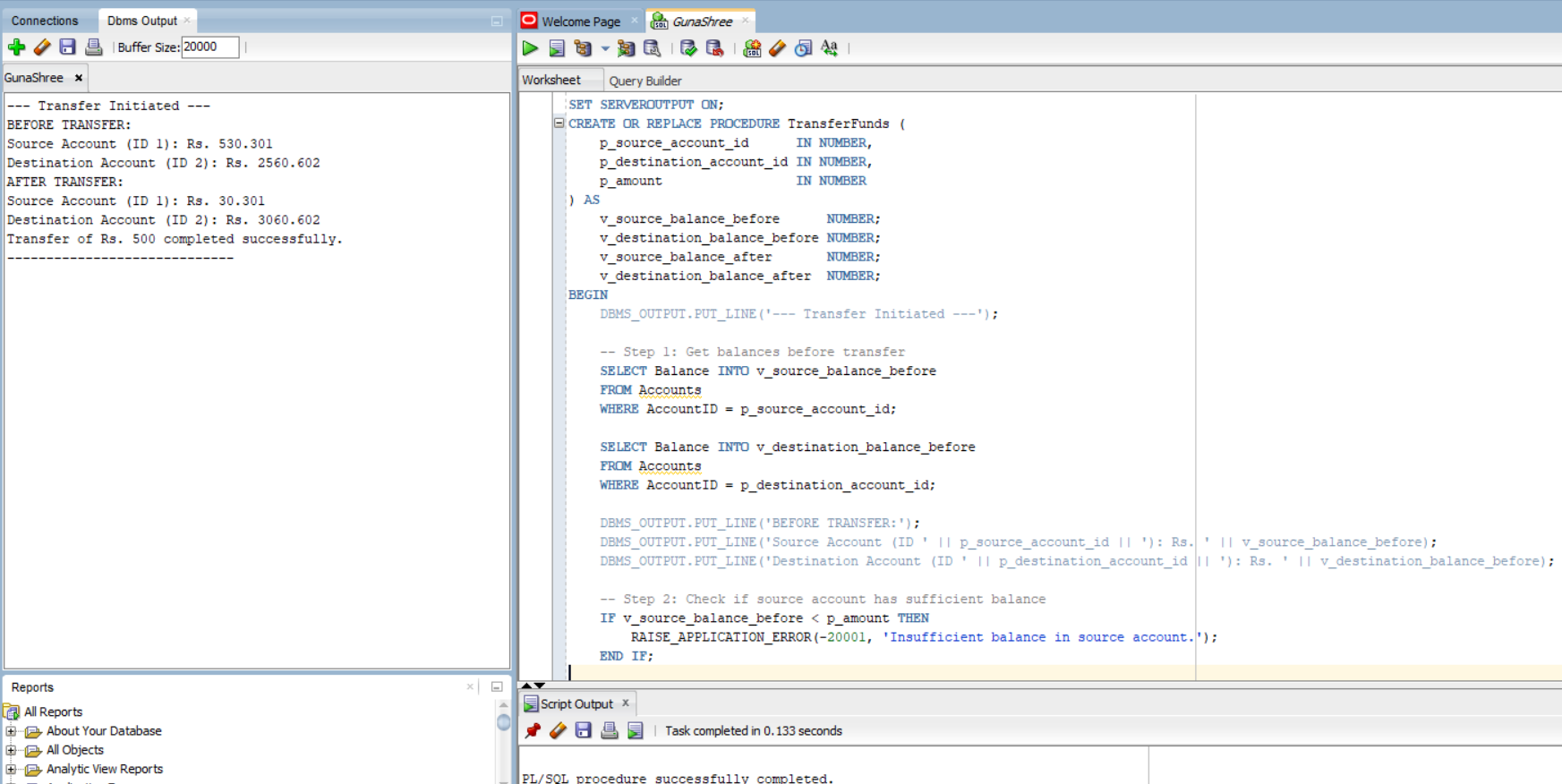
/

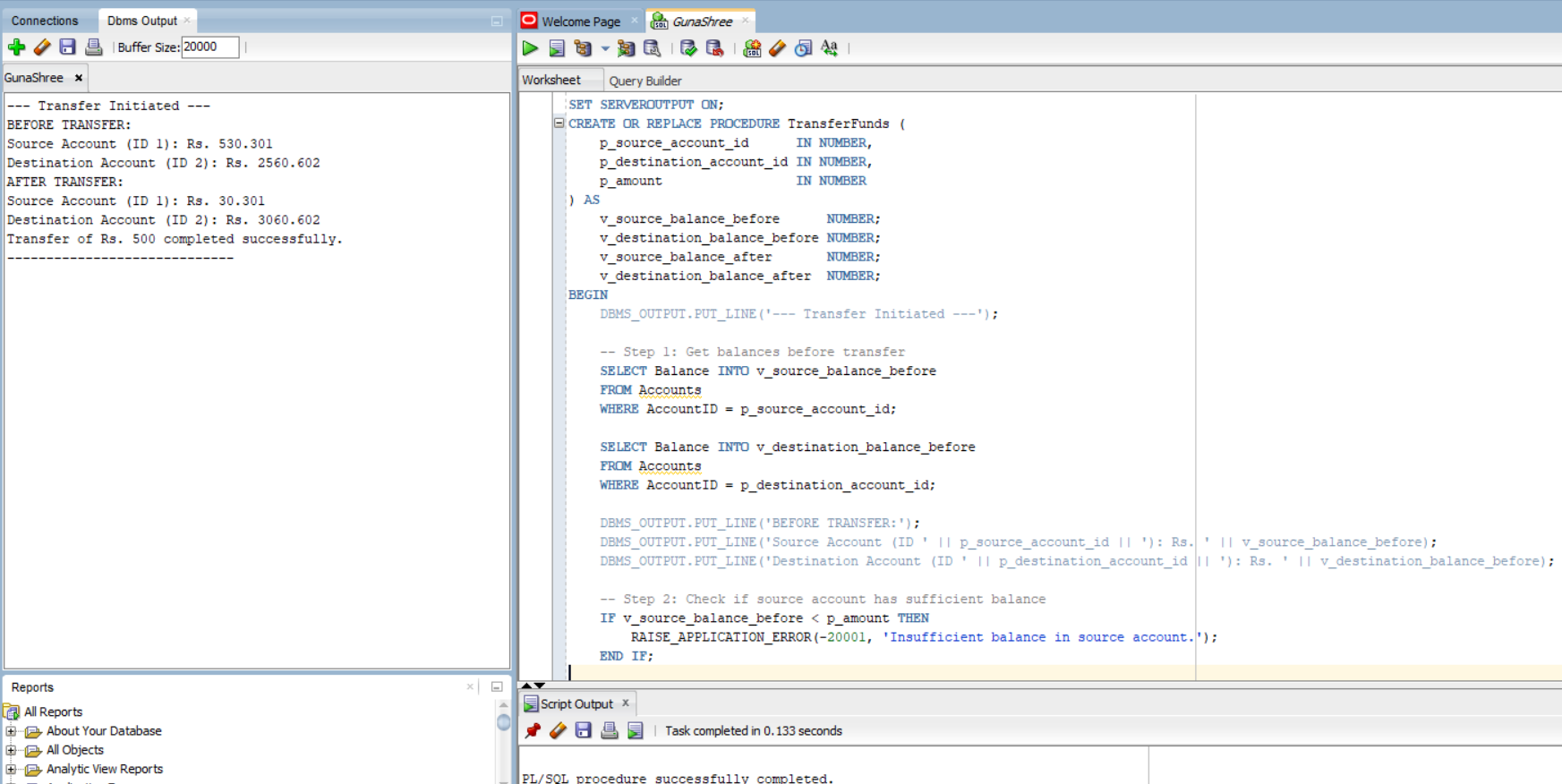
BEGIN

TransferFunds(1, 2, 500); -- Transfer ₹500 from AccountID 1 to 2

END;

/





---------------------------------------------------------------------------------------------------------------------------