**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.

**PL SQL code:**

DECLARE

v\_age NUMBER;

BEGIN

FOR c IN (SELECT CustomerID, DOB FROM Customers) LOOP

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

IF v\_age > 60 THEN

UPDATE Loans

SET InterestRate = InterestRate - 1

WHERE CustomerID = c.CustomerID;

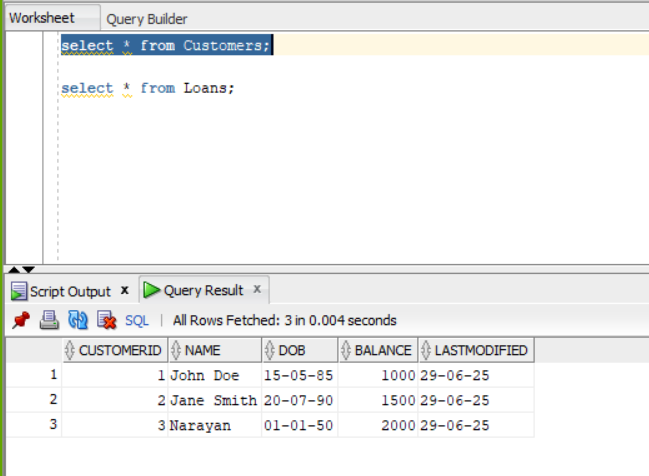
END IF;

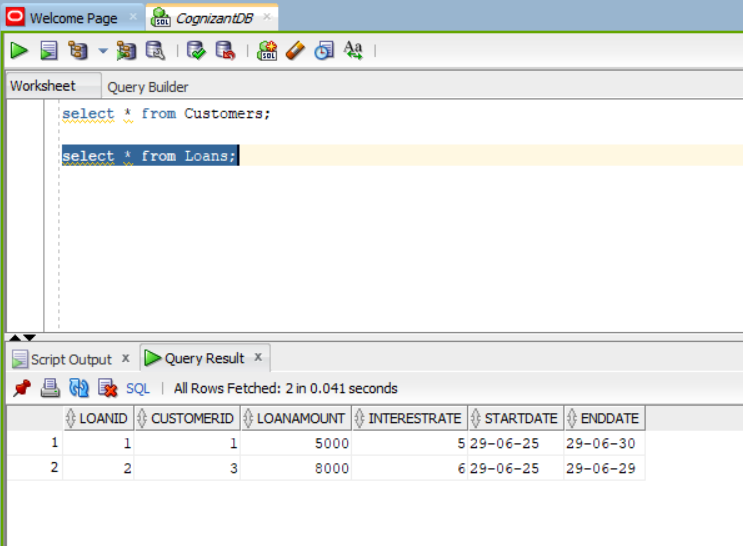
END LOOP;

COMMIT;

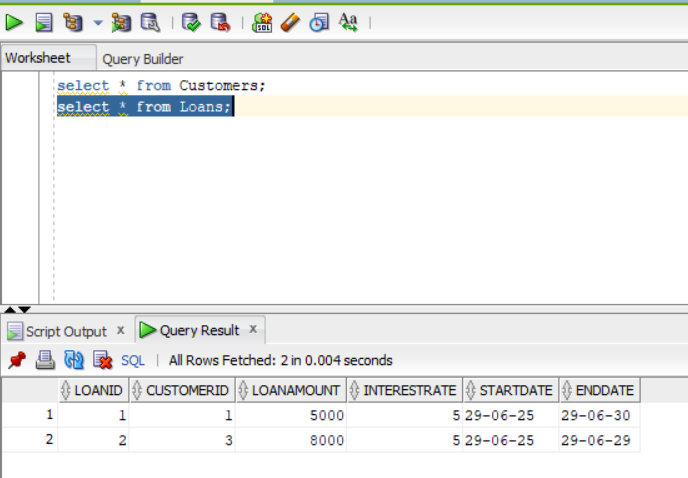
END;

**OUTPUT: (Before code execution)**

****

****

**OUTPUT: After Code Execution:**

****

**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.

**Code:**

ALTER TABLE Customers ADD IsVIP VARCHAR2(5);

BEGIN

FOR c IN (SELECT CustomerID, Balance FROM Customers) LOOP

IF c.Balance > 10000 THEN

UPDATE Customers

SET IsVIP = 'TRUE'

WHERE CustomerID = c.CustomerID;

ELSE

UPDATE Customers

SET IsVIP = 'FALSE'

WHERE CustomerID = c.CustomerID;

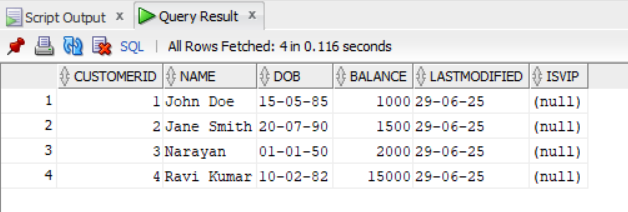
END IF;

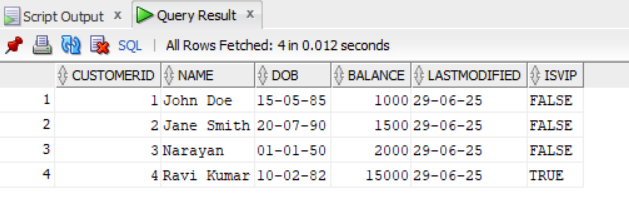
END LOOP;

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.

**CODE:**

DECLARE

v\_name VARCHAR2(100);

v\_end\_date DATE;

BEGIN

FOR loan\_rec IN (

SELECT LoanID, CustomerID, EndDate

FROM Loans

WHERE EndDate BETWEEN SYSDATE AND SYSDATE + 30

) LOOP

SELECT Name INTO v\_name

FROM Customers

WHERE CustomerID = loan\_rec.CustomerID;

DBMS\_OUTPUT.PUT\_LINE(

'Reminder: Dear ' || v\_name ||

', your loan (Loan ID: ' || loan\_rec.LoanID ||

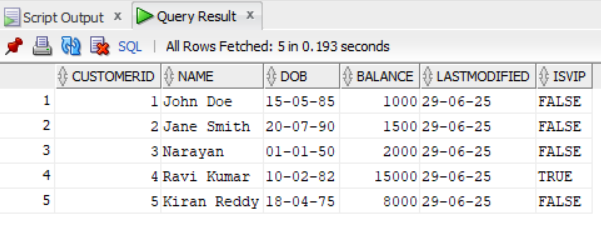
') is due on ' || TO\_CHAR(loan\_rec.EndDate, 'DD-MON-YYYY')

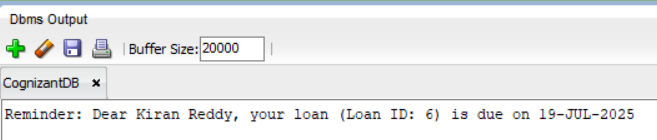
);

END LOOP;

END;

**OUTPUT:**

****

****

**Exercise 2: Error Handling**

**Scenario 1:** Handle exceptions during fund transfers between accounts.

**Question:** Write a stored procedure **SafeTransferFunds** that transfers funds between two accounts. Ensure that if any error occurs (e.g., insufficient funds), an appropriate error message is logged and the transaction is rolled back.

**CODE:**

CREATE OR REPLACE PROCEDURE SafeTransferFunds(

p\_from\_account\_id IN NUMBER,

p\_to\_account\_id IN NUMBER,

p\_amount IN NUMBER

)

IS

v\_balance NUMBER;

BEGIN

SELECT Balance INTO v\_balance

FROM Accounts

WHERE AccountID = p\_from\_account\_id;

IF v\_balance < p\_amount THEN

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

END IF;

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 successful.');

EXCEPTION

WHEN OTHERS THEN

ROLLBACK;

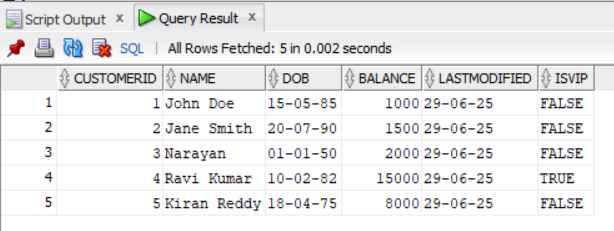
DBMS\_OUTPUT.PUT\_LINE('Transfer failed: ' || SQLERRM);

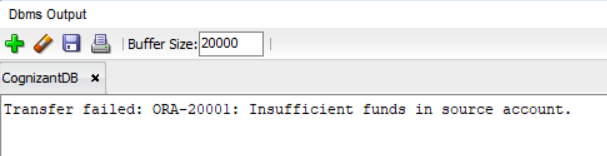
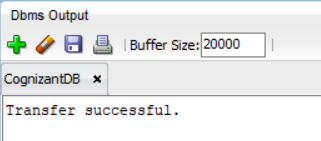
END;

EXEC SafeTransferFunds(1, 2, 500);

EXEC SafeTransferFunds(1, 2, 10000);

**OUTPUT:**

****

****

**Scenario 2:** Manage errors when updating employee salaries.

**Question:** Write a stored procedure **UpdateSalary** that increases the salary of an employee by a given percentage. If the employee ID does not exist, handle the exception and log an error message.

**CODE:**

CREATE OR REPLACE PROCEDURE UpdateSalary(

p\_employee\_id IN NUMBER,

p\_percentage IN NUMBER

)

IS

BEGIN

UPDATE Employees

SET Salary = Salary + (Salary \* p\_percentage / 100)

WHERE EmployeeID = p\_employee\_id;

IF SQL%ROWCOUNT = 0 THEN

RAISE\_APPLICATION\_ERROR(-20002, 'Employee ID not found.');

END IF;

COMMIT;

DBMS\_OUTPUT.PUT\_LINE('Salary updated successfully.');

EXCEPTION

WHEN OTHERS THEN

ROLLBACK;

DBMS\_OUTPUT.PUT\_LINE('Salary update failed: ' || SQLERRM);

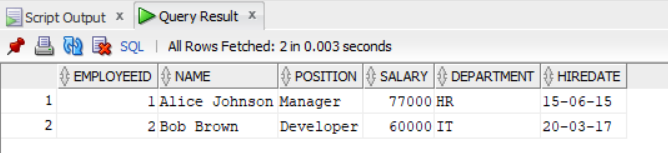
END;

EXEC UpdateSalary(2, 10);

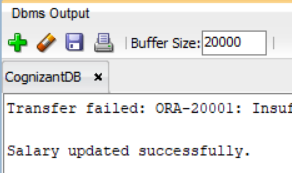
EXEC UpdateSalary(3, 10);

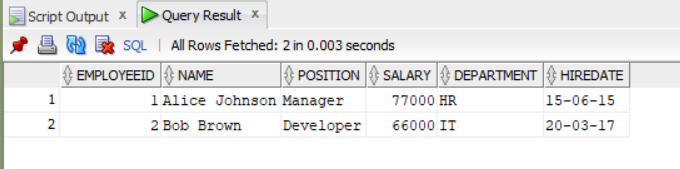
**OUTPUT:**

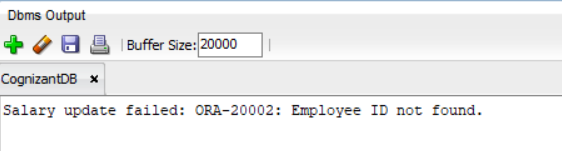
**BEFORE CODE EXECUTION**

****

**AFTER CODE EXECUTION**

****

****

****

**Scenario 3:** Ensure data integrity when adding a new customer.

**Question:** Write a stored procedure **AddNewCustomer** that inserts a new customer into the Customers table. If a customer with the same ID already exists, handle the exception by logging an error and preventing the insertion.

**CODE:**

CREATE OR REPLACE PROCEDURE AddNewCustomer(

p\_customer\_id IN NUMBER,

p\_name IN VARCHAR2,

p\_dob IN DATE,

p\_balance IN NUMBER

)

IS

BEGIN

INSERT INTO Customers (CustomerID, Name, DOB, Balance, LastModified)

VALUES (p\_customer\_id, p\_name, p\_dob, p\_balance, SYSDATE);

COMMIT;

DBMS\_OUTPUT.PUT\_LINE('Customer added successfully.');

EXCEPTION

WHEN DUP\_VAL\_ON\_INDEX THEN

DBMS\_OUTPUT.PUT\_LINE('Insert failed: Customer with ID ' || p\_customer\_id || ' already exists.');

WHEN OTHERS THEN

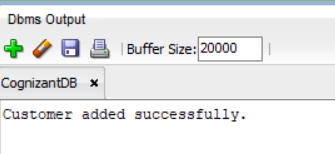
DBMS\_OUTPUT.PUT\_LINE('Insert failed: ' || SQLERRM);

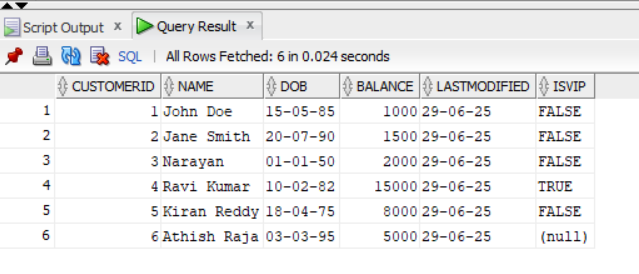
END;

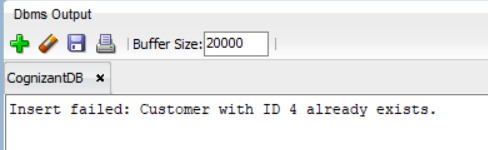
EXEC AddNewCustomer(6, 'Athish Raja', TO\_DATE('1995-03-03', 'YYYY-MM-DD'), 5000);

EXEC AddNewCustomer(4, 'Joel Gladwin', TO\_DATE('1995-03-03', 'YYYY-MM-DD'), 8000);

**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.

**CODE:**

CREATE OR REPLACE PROCEDURE ProcessMonthlyInterest

IS

BEGIN

UPDATE Accounts

SET Balance = Balance + (Balance \* 0.01),

LastModified = SYSDATE

WHERE AccountType = 'Savings';

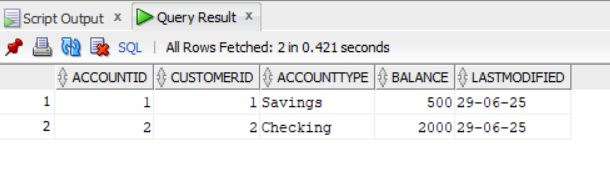
COMMIT;

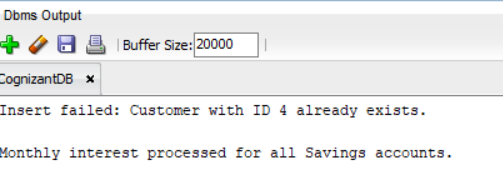
DBMS\_OUTPUT.PUT\_LINE('Monthly interest processed for all Savings accounts.');

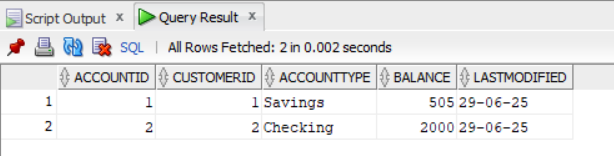
END;

EXEC ProcessMonthlyInterest;

**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.

**CODE:**

CREATE OR REPLACE PROCEDURE UpdateEmployeeBonus(

p\_department IN VARCHAR2,

p\_bonus\_percent IN NUMBER

)

IS

BEGIN

UPDATE Employees

SET Salary = Salary + (Salary \* p\_bonus\_percent / 100)

WHERE Department = p\_department;

IF SQL%ROWCOUNT > 0 THEN

DBMS\_OUTPUT.PUT\_LINE(SQL%ROWCOUNT || ' employee(s) in department ' || p\_department || ' received a bonus.');

ELSE

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

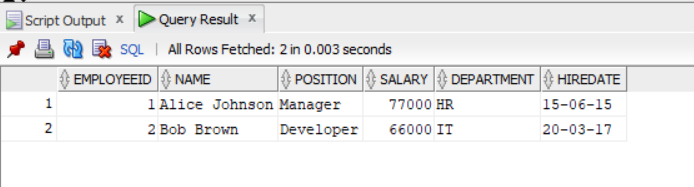
END IF;

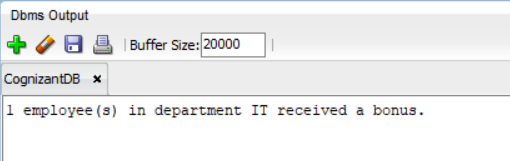
COMMIT;

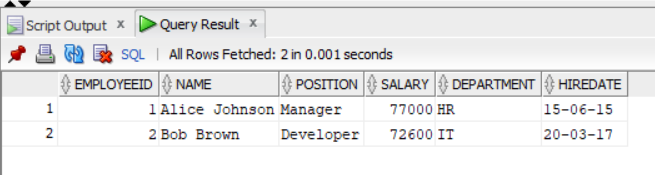
END;

EXEC UpdateEmployeeBonus('IT', 10);

**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.

**CODE:**

CREATE OR REPLACE PROCEDURE TransferFunds(

p\_from\_account IN NUMBER,

p\_to\_account IN NUMBER,

p\_amount IN NUMBER

)

IS

v\_from\_balance NUMBER;

v\_from\_cust NUMBER;

v\_to\_cust NUMBER;

BEGIN

SELECT Balance, CustomerID INTO v\_from\_balance, v\_from\_cust

FROM Accounts

WHERE AccountID = p\_from\_account;

SELECT CustomerID INTO v\_to\_cust

FROM Accounts

WHERE AccountID = p\_to\_account;

IF v\_from\_cust != v\_to\_cust THEN

RAISE\_APPLICATION\_ERROR(-20010, 'Accounts do not belong to the same customer.');

END IF;

IF v\_from\_balance < p\_amount THEN

RAISE\_APPLICATION\_ERROR(-20011, 'Insufficient balance for transfer.');

END IF;

UPDATE Accounts

SET Balance = Balance - p\_amount,

LastModified = SYSDATE

WHERE AccountID = p\_from\_account;

UPDATE Accounts

SET Balance = Balance + p\_amount,

LastModified = SYSDATE

WHERE AccountID = p\_to\_account;

COMMIT;

DBMS\_OUTPUT.PUT\_LINE('Transfer of ₹' || p\_amount || ' between accounts successful.');

EXCEPTION

WHEN OTHERS THEN

ROLLBACK;

DBMS\_OUTPUT.PUT\_LINE('Transfer failed: ' || SQLERRM);

END;

EXEC TransferFunds(1, 3, 300);

**OUTPUT:**

