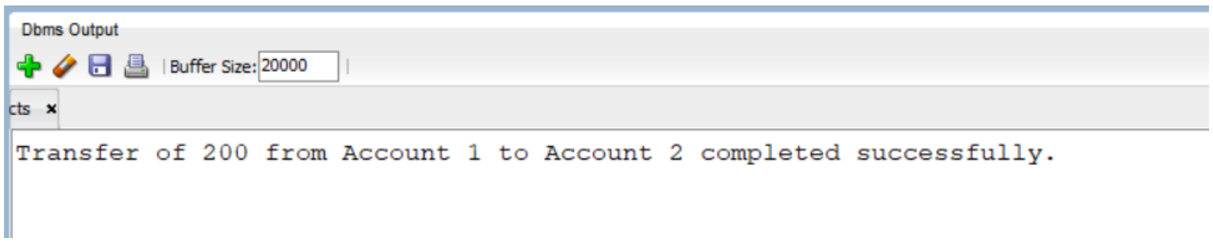
Exercise 2: Error Handling

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.

Solution:

```
CREATE OR REPLACE PROCEDURE SafeTransferFunds (
   from_acc_id IN NUMBER,
   to_acc_id IN NUMBER,
   amount IN NUMBER
) AS
   from_balance NUMBER;
BEGIN
   -- Lock and get balance from source account
   SELECT Balance INTO from_balance
   FROM Accounts
   WHERE AccountID = from_acc_id
   FOR UPDATE;

   IF from_balance < amount THEN
      RAISE_APPLICATION_ERROR(-20001, 'Insufficient balance in Account ' ||
from_acc_id);
   END IF;

   -- Deduct from sender
   UPDATE Accounts
   SET Balance = Balance - amount,
      LastModified = SYSDATE
   WHERE AccountID = from_acc_id;

   -- Add to receiver
   UPDATE Accounts
   SET Balance = Balance + amount,
      LastModified = SYSDATE
   WHERE AccountID = to_acc_id;

   COMMIT;

   DBMS_OUTPUT.PUT_LINE('Transfer of ' || amount || ' from Account ' || from_acc_id || '
to Account ' || to_acc_id || ' completed successfully.');
```

```
EXCEPTION
   WHEN OTHERS THEN
      ROLLBACK;
      DBMS_OUTPUT.PUT_LINE('Error during fund transfer: ' || SQLERRM);
END;
/

EXEC SafeTransferFunds(1, 2, 200);
```



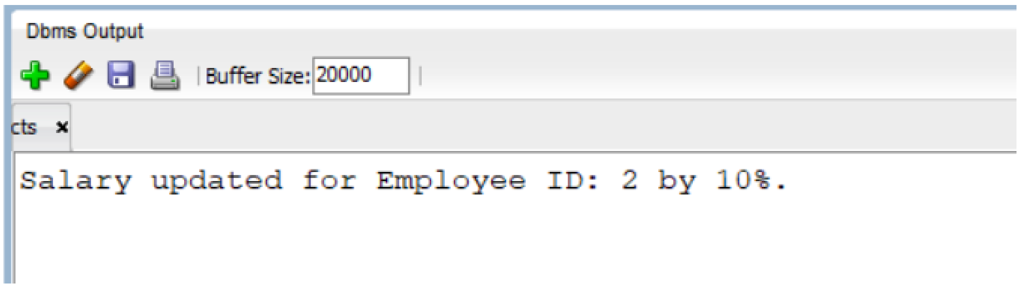
```
Transfer of 200 from Account 1 to Account 2 completed successfully.
```

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.

Solution:

```
CREATE OR REPLACE PROCEDURE UpdateSalary (
   emp_id IN NUMBER,
   percent IN NUMBER
) AS
   current_salary NUMBER;
BEGIN
   SELECT Salary INTO current_salary
   FROM Employees
   WHERE EmployeeID = emp_id
   FOR UPDATE;

   UPDATE Employees
   SET Salary = current_salary + (current_salary * percent / 100)
   WHERE EmployeeID = emp_id;

   COMMIT;

   DBMS_OUTPUT.PUT_LINE('Salary updated for Employee ID: ' || emp_id || ' by ' ||
percent || '%.');
```

```
EXCEPTION
   WHEN NO_DATA_FOUND THEN
      DBMS_OUTPUT.PUT_LINE('Error: Employee ID ' || emp_id || ' not found.');
   WHEN OTHERS THEN
      ROLLBACK;
      DBMS_OUTPUT.PUT_LINE('Unexpected error: ' || SQLERRM);
END;
/

EXEC UpdateSalary(2, 10);
```



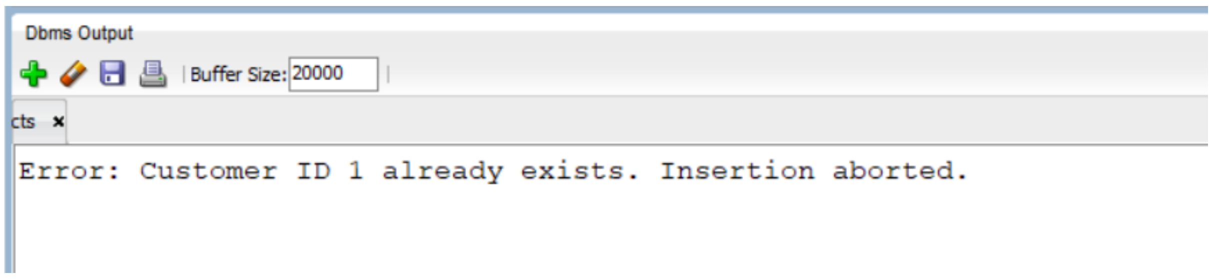
```
Salary updated for Employee ID: 2 by 10%.
```

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.

Solution:

```
CREATE OR REPLACE PROCEDURE AddNewCustomer (
   cust_id IN NUMBER,
   name IN VARCHAR2,
   dob IN DATE,
   balance IN NUMBER
) AS
BEGIN
   INSERT INTO Customers (CustomerID, Name, DOB, Balance, LastModified)
   VALUES (cust_id, name, dob, balance, SYSDATE);

   COMMIT;

   DBMS_OUTPUT.PUT_LINE('Customer added successfully: ' || name);
EXCEPTION
   WHEN DUP_VAL_ON_INDEX THEN
```

```
        DBMS_OUTPUT.PUT_LINE('Error: Customer ID ' || cust_id || ' already exists. Insertion
aborted.');
    WHEN OTHERS THEN
        ROLLBACK;
        DBMS_OUTPUT.PUT_LINE('Unexpected error while adding customer: ' ||
SQLERRM);
END;
/

EXEC AddNewCustomer(1, 'Test Duplicate', TO_DATE('1999-01-01','YYYY-MM-DD'),
1000);
```



```
Error: Customer ID 1 already exists. Insertion aborted.
```