**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

   bal accounts.balance%type;

begin

   for acc in (

      select accountid,

             balance

        from accounts

   ) loop

      bal := acc.balance;

      update accounts

         set

         balance = bal \* 1.01

       where accountid = acc.accountid;

      dbms\_output.put\_line('ACCOUNT ID: '

                           || acc.accountid

                           || ', OLD BALANCE: '

                           || bal

                           || ', NEW BALANCE: '

                           ||(bal \* 1.01));

   end loop;

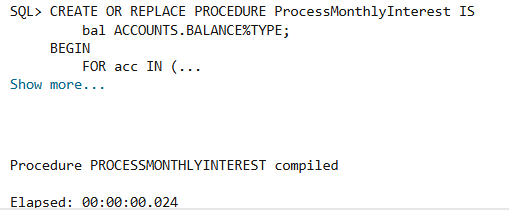
   commit;

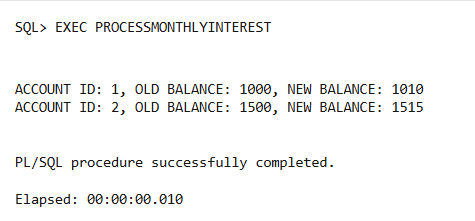
end;

/

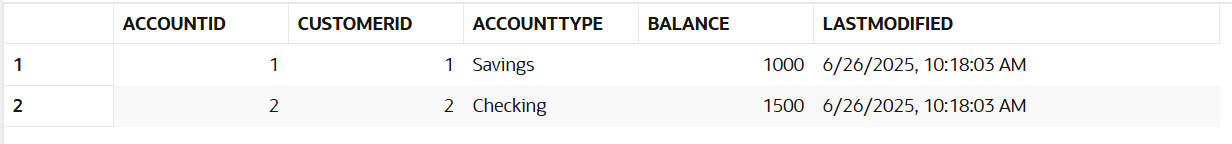
EXEC PROCESSMONTHLYINTEREST;

**OUTPUT:**

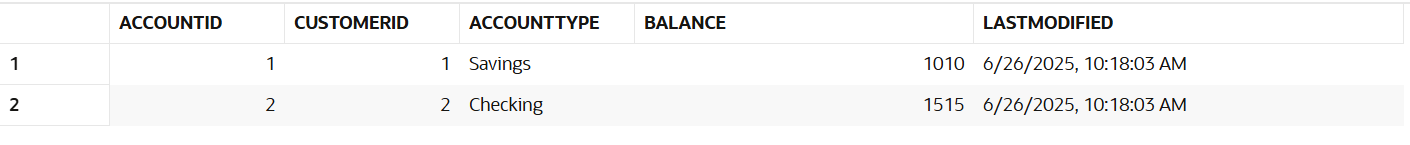
****

****

**Initial Accounts table:**

****

**Updated Accounts Table:**

****

**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 (

bonusPercentage IN NUMBER,

deptName IN VARCHAR2

) IS

bonus employees.salary%TYPE;

sal employees.salary%TYPE;

BEGIN

FOR emp IN (

SELECT employeeid, salary

FROM employees

WHERE department = deptName

) LOOP

sal := emp.salary;

bonus := sal \* (100 + bonusPercentage) / 100;

UPDATE employees

SET salary = bonus

WHERE employeeid = emp.employeeid;

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

', Old salary: ' || sal ||

', New salary: ' || bonus);

END LOOP;

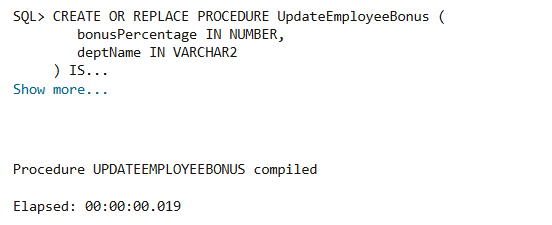
COMMIT;

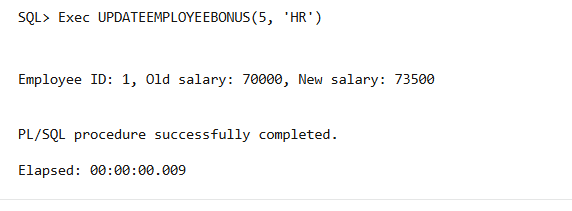
END;

/

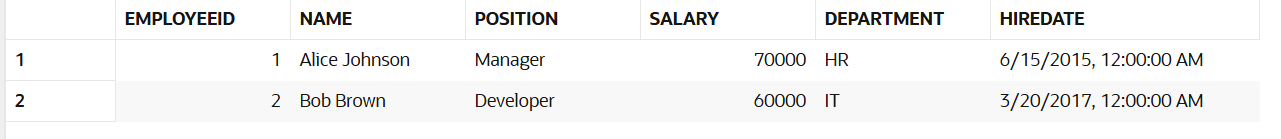
EXEC UPDATEEMPLOYEEBONUS(5, 'HR');

**OUTPUT:**

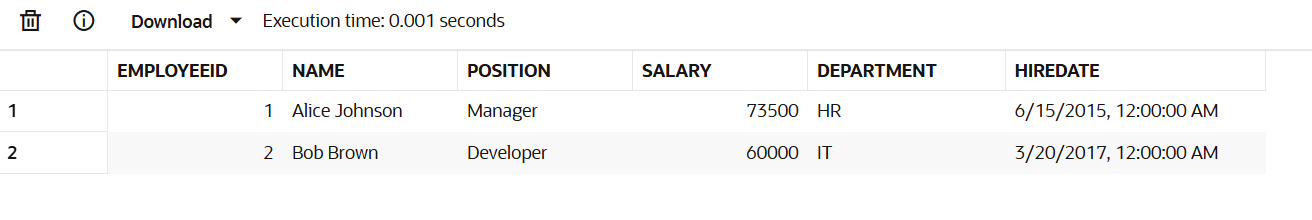
****

****

**Initial Employees Table:**

****

**Updated Employees Table:**

****

**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 (

from\_account\_id IN NUMBER,

to\_account\_id IN NUMBER,

amount\_to\_transfer IN NUMBER

)

AS

from\_balance NUMBER;

to\_balance NUMBER;

BEGIN

SELECT balance INTO from\_balance

FROM Accounts

WHERE AccountID = from\_account\_id

FOR UPDATE;

IF from\_balance < amount\_to\_transfer THEN

RAISE\_APPLICATION\_ERROR(-20001, 'Not enough balance to complete the transfer.');

END IF;

SELECT balance INTO to\_balance

FROM Accounts

WHERE AccountID = to\_account\_id

FOR UPDATE;

UPDATE Accounts

SET balance = balance - amount\_to\_transfer,

LastModified = SYSDATE

WHERE AccountID = from\_account\_id;

UPDATE Accounts

SET balance = balance + amount\_to\_transfer,

LastModified = SYSDATE

WHERE AccountID = to\_account\_id;

INSERT INTO Transactions (

TransactionID, AccountID, TransactionDate, Amount, TransactionType

) VALUES (

Transactions\_seq.NEXTVAL, from\_account\_id, SYSDATE, amount\_to\_transfer, 'Withdrawal'

);

INSERT INTO Transactions (

TransactionID, AccountID, TransactionDate, Amount, TransactionType

) VALUES (

Transactions\_seq.NEXTVAL, to\_account\_id, SYSDATE, amount\_to\_transfer, 'Deposit'

);

COMMIT;

EXCEPTION

WHEN OTHERS THEN

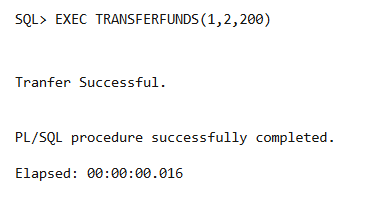
ROLLBACK;

   RAISE;

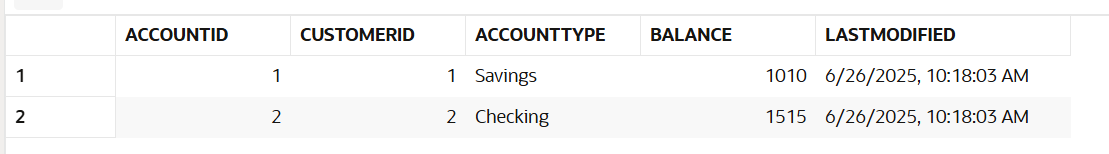
END;

/

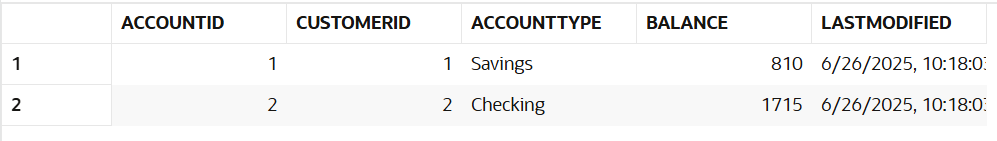
**OUTPUT:**

****

**Initial Accounts Table:**

****

**Modified Accounts Table:**

****