**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.

**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.

**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.

**STEP 1: CREATING TABLES AND INSERTING VALUES**

Create ACCOUNTS table with AccountID, CustomerID, AccountType, Balance.

Query:

|  |
| --- |
| CREATE TABLE ACCOUNTS (  AccountID NUMBER PRIMARY KEY,  CustomerID NUMBER,  AccountType VARCHAR2(20),  Balance NUMBER  ); |

Output:

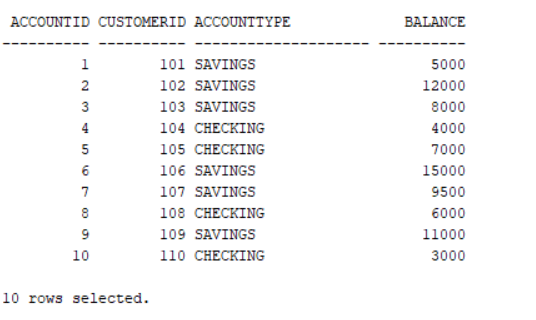


Inserting values into ACCOUNTS Table.

Query:

|  |
| --- |
| INSERT INTO ACCOUNTS VALUES (1, 101, 'SAVINGS', 5000);  INSERT INTO ACCOUNTS VALUES (2, 102, 'SAVINGS', 12000);  INSERT INTO ACCOUNTS VALUES (3, 103, 'SAVINGS', 8000);  INSERT INTO ACCOUNTS VALUES (4, 104, 'CHECKING', 4000);  INSERT INTO ACCOUNTS VALUES (5, 105, 'CHECKING', 7000);  INSERT INTO ACCOUNTS VALUES (6, 106, 'SAVINGS', 15000);  INSERT INTO ACCOUNTS VALUES (7, 107, 'SAVINGS', 9500);  INSERT INTO ACCOUNTS VALUES (8, 108, 'CHECKING', 6000);  INSERT INTO ACCOUNTS VALUES (9, 109, 'SAVINGS', 11000);  INSERT INTO ACCOUNTS VALUES (10, 110, 'CHECKING', 3000);  COMMIT;  SELECT \* FROM ACCOUNTS; |

Output:



Create Employee table with EmpID, Name, DepartmentID, Salary.

Query:

|  |
| --- |
| CREATE TABLE EMPLOYEE (  EmpID NUMBER PRIMARY KEY,  Name VARCHAR2(100),  DepartmentID NUMBER,  Salary NUMBER  ); |

Output:

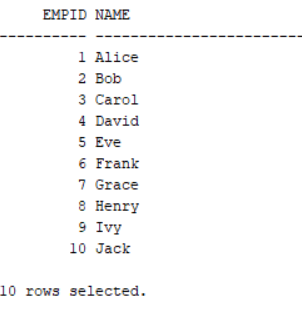


Inserting Values into Employee table

Query:

|  |
| --- |
| INSERT INTO EMPLOYEE VALUES (1, 'Alice', 10, 50000);  INSERT INTO EMPLOYEE VALUES (2, 'Bob', 20, 55000);  INSERT INTO EMPLOYEE VALUES (3, 'Carol', 10, 60000);  INSERT INTO EMPLOYEE VALUES (4, 'David', 20, 58000);  INSERT INTO EMPLOYEE VALUES (5, 'Eve', 10, 62000);  INSERT INTO EMPLOYEE VALUES (6, 'Frank', 30, 45000);  INSERT INTO EMPLOYEE VALUES (7, 'Grace', 30, 48000);  INSERT INTO EMPLOYEE VALUES (8, 'Henry', 20, 52000);  INSERT INTO EMPLOYEE VALUES (9, 'Ivy', 10, 61000);  INSERT INTO EMPLOYEE VALUES (10, 'Jack', 30, 47000);  COMMIT;  SELECT \* FROM EMPLOYEE; |

Output:



**STEP 2 – IMPLEMENTING SCENARIOS**

**SCENARIO 1**

The bank needs to process monthly interest for all savings accounts.

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.

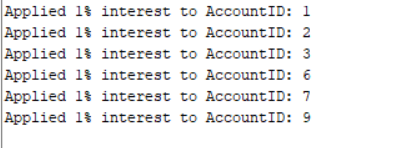
Query:

|  |
| --- |
| CREATE OR REPLACE PROCEDURE ProcessMonthlyInterest IS  BEGIN  FOR acc\_rec IN (  SELECT AccountID, Balance  FROM ACCOUNTS  WHERE AccountType = 'SAVINGS'  ) LOOP  UPDATE ACCOUNTS  SET Balance = Balance + (Balance \* 0.01)  WHERE AccountID = acc\_rec.AccountID;  DBMS\_OUTPUT.PUT\_LINE('Applied 1% interest to AccountID: ' || acc\_rec.AccountID);  END LOOP;  COMMIT;  END;  / |

Calling Procedure:

|  |
| --- |
| BEGIN  ProcessMonthlyInterest;  END;  / |

Output:



**SCENARIO 2**

The bank wants to implement a bonus scheme for employees based on their performance.

Write a stored procedure **UpdateEmployeeBonus** that updates the salary of employees in a given department by adding a bonus percentage passed as a parameter.

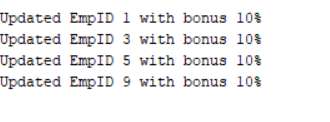
Query:

|  |
| --- |
| CREATE OR REPLACE PROCEDURE UpdateEmployeeBonus (  p\_DepartmentID IN NUMBER,  p\_BonusPercent IN NUMBER  ) IS  BEGIN  FOR emp\_rec IN (  SELECT EmpID, Salary  FROM EMPLOYEE  WHERE DepartmentID = p\_DepartmentID  ) LOOP  UPDATE EMPLOYEE  SET Salary = Salary + (Salary \* p\_BonusPercent / 100)  WHERE EmpID = emp\_rec.EmpID;  DBMS\_OUTPUT.PUT\_LINE('Updated EmpID ' || emp\_rec.EmpID || ' with bonus ' || p\_BonusPercent || '%');  END LOOP;  COMMIT;  END;  / |

Calling Procedure:

|  |
| --- |
| BEGIN  UpdateEmployeeBonus(10, 10);  END;  / |

Output:



**SCENARIO 3**

Customers should be able to transfer funds between their accounts.

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.

Query:

|  |
| --- |
| CREATE OR REPLACE PROCEDURE TransferFunds (  p\_FromAccountID IN NUMBER,  p\_ToAccountID IN NUMBER,  p\_Amount IN NUMBER  ) IS  v\_FromBalance NUMBER;  BEGIN  SELECT Balance INTO v\_FromBalance FROM ACCOUNTS WHERE AccountID = p\_FromAccountID;  IF v\_FromBalance < p\_Amount THEN  DBMS\_OUTPUT.PUT\_LINE('Insufficient funds in AccountID: ' || p\_FromAccountID);  ELSE  UPDATE ACCOUNTS  SET Balance = Balance - p\_Amount  WHERE AccountID = p\_FromAccountID;  UPDATE ACCOUNTS  SET Balance = Balance + p\_Amount  WHERE AccountID = p\_ToAccountID;  COMMIT;  DBMS\_OUTPUT.PUT\_LINE('Transferred ' || p\_Amount || ' from AccountID ' || p\_FromAccountID || ' to AccountID ' || p\_ToAccountID);  END IF;  END;  / |

Calling Procedure:

|  |
| --- |
| BEGIN  TransferFunds(2, 1, 1000);  TransferFunds(4, 5, 500);  TransferFunds(2, 5, 200);  TransferFunds(2, 3, 600);  TransferFunds(1, 7, 400);  TransferFunds(4, 6, 400);  END;  / |

Output:

