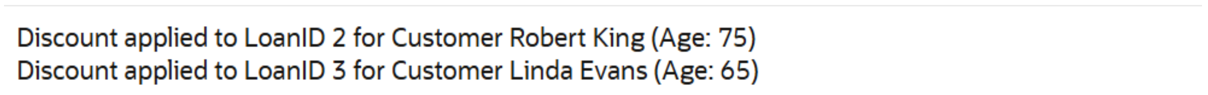
WEEK 2
MODULE 3 ASSIGNMENT

Exercise 1: Control Structures

Scenario 1:

```
SET SERVEROUTPUT ON;
BEGIN
  FOR cust IN (
    SELECT c.CustomerID, c.Name, c.DOB, l.LoanID, l.InterestRate
    FROM Customers c
    JOIN Loans l ON c.CustomerID = l.CustomerID
  ) LOOP
    IF FLOOR(MONTHS_BETWEEN(SYSDATE, cust.DOB) / 12) > 60 THEN
      UPDATE Loans
      SET InterestRate = InterestRate - 1
      WHERE LoanID = cust.LoanID;

      DBMS_OUTPUT.PUT_LINE('Discount applied to LoanID ' || cust.LoanID ||
                     ' for Customer ' || cust.Name ||
                     ' (Age: ' || FLOOR(MONTHS_BETWEEN(SYSDATE, cust.DOB) / 12) || ')');
    END IF;
  END LOOP;

  COMMIT;
END;
/.
```

OUTPUT:

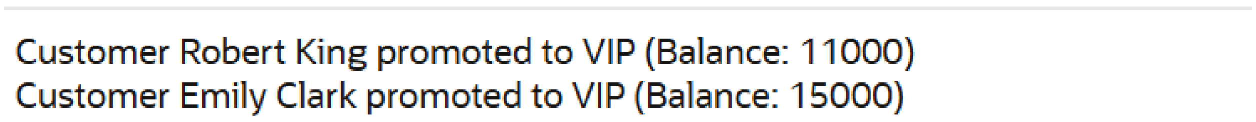


```
Discount applied to LoanID 2 for Customer Robert King (Age: 75)
Discount applied to LoanID 3 for Customer Linda Evans (Age: 65)
```

Scenario 2:

```
SET SERVEROUTPUT ON;
BEGIN
  FOR cust IN (
    SELECT CustomerID, Name, Balance
    FROM Customers
  ) LOOP
    IF cust.Balance > 10000 THEN
      UPDATE Customers
      SET IsVIP = 'TRUE'
      WHERE CustomerID = cust.CustomerID;

      DBMS_OUTPUT.PUT_LINE('Customer ' || cust.Name || ' promoted to VIP (Balance: ' ||
cust.Balance || ')');
    END IF;
  END LOOP;
  COMMIT;
END;
/
```

OUTPUT:

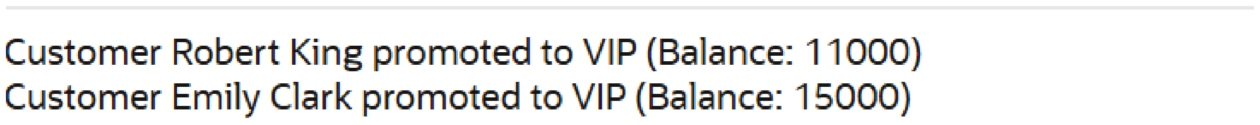


```
Customer Robert King promoted to VIP (Balance: 11000)
Customer Emily Clark promoted to VIP (Balance: 15000)
```

Scenario 3:

```
SET SERVEROUTPUT ON;

BEGIN
  FOR loan_rec IN (
    SELECT l.LoanID, c.Name, l.EndDate
    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: Loan ID ' || loan_rec.LoanID ||
                         ' for ' || loan_rec.Name ||
                         ' is due on ' || TO_CHAR(loan_rec.EndDate, 'DD-MON-YYYY'));
  END LOOP;
END;
/
```

OUTPUT:



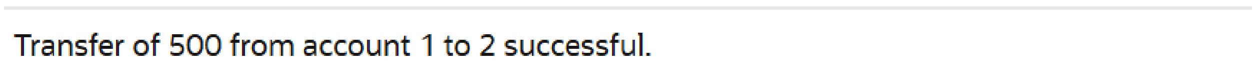
```
Customer Robert King promoted to VIP (Balance: 11000)
Customer Emily Clark promoted to VIP (Balance: 15000)
```

Exercise 2: Error Handling

Scenario 1:

```
CREATE OR REPLACE PROCEDURE SafeTransferFunds(
  p_fromAccountID IN NUMBER,
  p_toAccountID IN NUMBER,
  p_amount IN NUMBER
) AS v_balance NUMBER;
BEGIN
  SELECT Balance INTO v_balance FROM Accounts WHERE AccountID = p_fromAccountID;
  IF v_balance < p_amount THEN
    RAISE_APPLICATION_ERROR(-20001, 'Insufficient funds in source account');
  END IF;
  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('Transfer of ' || p_amount || ' from account ' || p_fromAccountID ||
' to ' || p_toAccountID || ' successful.');
EXCEPTION
  WHEN OTHERS THEN
    ROLLBACK;
    DBMS_OUTPUT.PUT_LINE('Transfer failed: ' || SQLERRM);
END;
/
EXEC SafeTransferFunds(1, 2, 500);
```

OUTPUT:

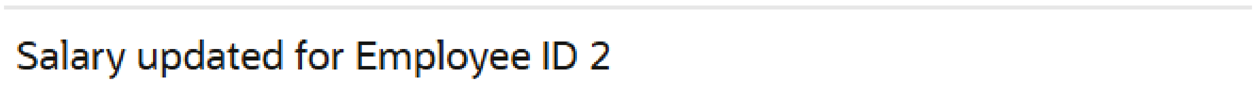


```
Transfer of 500 from account 1 to 2 successful.
```

Scenario 2:

```
CREATE OR REPLACE PROCEDURE UpdateSalary(
  p_empID IN NUMBER,
  p_percent IN NUMBER
) AS
BEGIN
  UPDATE Employees
  SET Salary = Salary + (Salary * p_percent / 100)
  WHERE EmployeeID = p_empID;

  IF SQL%ROWCOUNT = 0 THEN
    RAISE_APPLICATION_ERROR(-20002, 'Employee ID not found');
  END IF;

  COMMIT;
  DBMS_OUTPUT.PUT_LINE('Salary updated for Employee ID ' || p_empID);
EXCEPTION
  WHEN OTHERS THEN
    DBMS_OUTPUT.PUT_LINE('Error updating salary: ' || SQLERRM);
END;
/
EXEC UpdateSalary(2, 10);
```

OUTPUT:

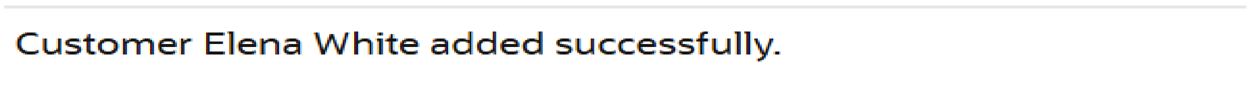


```
Salary updated for Employee ID 2
```

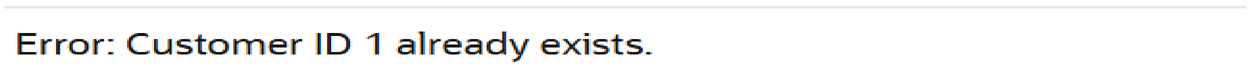
Scenario 3:

```
CREATE OR REPLACE PROCEDURE AddNewCustomer(
  p_CustomerID IN NUMBER,
  p_Name IN VARCHAR2,
  p_DOB IN DATE,
  p_Balance IN NUMBER
) AS
BEGIN
  INSERT INTO Customers (CustomerID, Name, DOB, Balance, LastModified)
  VALUES (p_CustomerID, p_Name, p_DOB, p_Balance, SYSDATE);
  COMMIT;
  DBMS_OUTPUT.PUT_LINE('Customer ' || p_Name || ' added successfully.');
EXCEPTION
  WHEN DUP_VAL_ON_INDEX THEN
    DBMS_OUTPUT.PUT_LINE('Error: Customer ID ' || p_CustomerID || ' already exists.');
  WHEN OTHERS THEN
    DBMS_OUTPUT.PUT_LINE('Unexpected error: ' || SQLERRM);
END;
/
EXEC AddNewCustomer(10, 'Elena White', TO_DATE('1992-03-15', 'YYYY-MM-DD'),
12000);
EXEC AddNewCustomer(1, 'Duplicate John', TO_DATE('1980-01-01', 'YYYY-MM-DD'),
5000);
```

OUTPUT:



```
Customer Elena White added successfully.
```



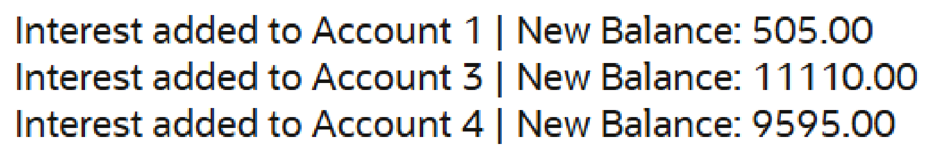
```
Error: Customer ID 1 already exists.
```

Exercise 3: Stored Procedures

Scenario 1:

```
CREATE OR REPLACE PROCEDURE ProcessMonthlyInterest AS
BEGIN
  FOR acc IN (
    SELECT AccountID, Balance
    FROM Accounts
    WHERE AccountType = 'Savings'
  ) LOOP
    UPDATE Accounts
    SET Balance = Balance + (acc.Balance * 0.01),
        LastModified = SYSDATE
    WHERE AccountID = acc.AccountID;

    DBMS_OUTPUT.PUT_LINE('Interest added to Account ' || acc.AccountID ||
                         ' | New Balance: ' || TO_CHAR(acc.Balance * 1.01, '999999.99'));
  END LOOP;
  COMMIT;
END;
/
EXEC ProcessMonthlyInterest;
```

OUTPUT:

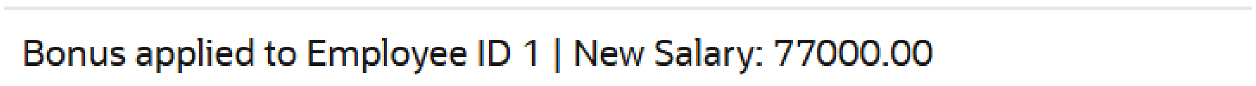


```
Interest added to Account 1 | New Balance: 505.00
Interest added to Account 3 | New Balance: 11110.00
Interest added to Account 4 | New Balance: 9595.00
```

Scenario 2:

```
CREATE OR REPLACE PROCEDURE UpdateEmployeeBonus(
  p_department IN VARCHAR2,
  p_bonusPercent IN NUMBER
) AS
BEGIN
  FOR emp IN (
    SELECT EmployeeID, Salary FROM Employees WHERE Department = p_department
  ) LOOP
    UPDATE Employees
    SET Salary = Salary + (emp.Salary * p_bonusPercent / 100)
    WHERE EmployeeID = emp.EmployeeID;

    DBMS_OUTPUT.PUT_LINE('Bonus applied to Employee ID ' || emp.EmployeeID ||
                         ' | New Salary: ' || TO_CHAR(emp.Salary * (1 + p_bonusPercent / 100),
'999999.99'));
  END LOOP;

  COMMIT;
END;
/
EXEC UpdateEmployeeBonus('HR', 10);
```

OUTPUT:

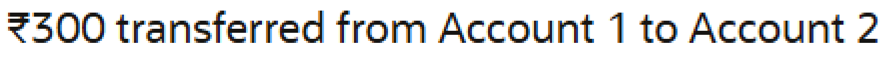


```
Bonus applied to Employee ID 1 | New Salary: 77000.00
```

Scenario 3:

```
CREATE OR REPLACE PROCEDURE TransferFunds(
  p_fromAccountID IN NUMBER,
  p_toAccountID IN NUMBER,
  p_amount IN NUMBER
) AS
  v_balance NUMBER;
BEGIN
  SELECT Balance INTO v_balance FROM Accounts WHERE AccountID = p_fromAccountID;
  IF v_balance < p_amount THEN
    RAISE_APPLICATION_ERROR(-20001, 'Insufficient balance in source account.');
  END IF;
  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('₹' || p_amount || ' transferred from Account ' || p_fromAccountID
|| ' to Account ' || p_toAccountID);
EXCEPTION
  WHEN OTHERS THEN
    ROLLBACK;
    DBMS_OUTPUT.PUT_LINE('Transfer failed: ' || SQLERRM);
END;
/


EXEC TransferFunds(1, 2, 300);
```

OUTPUT:



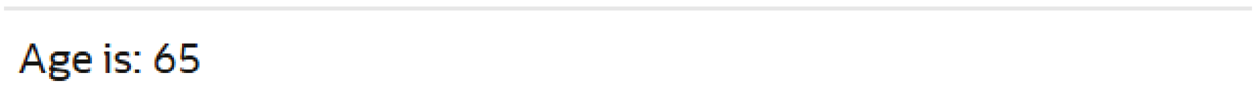
```
₹300 transferred from Account 1 to Account 2
```

Exercise 4: Functions

Scenario 1:

```
CREATE OR REPLACE FUNCTION CalculateAge(
  p_dob IN DATE
) RETURN NUMBER IS
  v_age NUMBER;
BEGIN
  v_age := FLOOR(MONTHS_BETWEEN(SYSDATE, p_dob) / 12);
  RETURN v_age;
END;
/
SET SERVEROUTPUT ON;

DECLARE
  v_age NUMBER;
BEGIN
  v_age := CalculateAge(TO_DATE('1960-01-01', 'YYYY-MM-DD'));
  DBMS_OUTPUT.PUT_LINE('Age is: ' || v_age);
END;
/
```

OUTPUT:

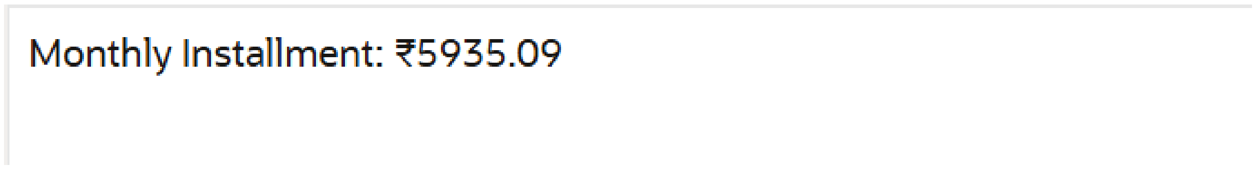


```
Age is: 65
```

Scenario 2:

```
CREATE OR REPLACE FUNCTION CalculateMonthlyInstallment(
  p_loanAmount IN NUMBER,
  p_annualInterestRate IN NUMBER,
  p_durationYears IN NUMBER
) RETURN NUMBER IS
  v_monthlyRate NUMBER;
  v_months NUMBER;
  v_installment NUMBER;
BEGIN
  v_monthlyRate := p_annualInterestRate / 12 / 100;
  v_months := p_durationYears * 12;

  -- EMI formula: P * r * (1 + r)^n / ((1 + r)^n - 1)
  v_installment := p_loanAmount * v_monthlyRate * POWER(1 + v_monthlyRate, v_months) /
                   (POWER(1 + v_monthlyRate, v_months) - 1);

  RETURN ROUND(v_installment, 2);
END;
/
SET SERVEROUTPUT ON;

DECLARE
  v_emi NUMBER;
BEGIN
  v_emi := CalculateMonthlyInstallment(500000, 7.5, 10); -- ₹5 lakhs, 7.5% interest, 10 years
  DBMS_OUTPUT.PUT_LINE('Monthly Installment: ₹' || v_emi);
END;
/
```

OUTPUT:



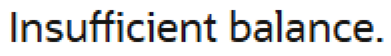
```
Monthly Installment: ₹5935.09
```

Scenario 3:

```
CREATE OR REPLACE FUNCTION HasSufficientBalance(
  p_accountID IN NUMBER,
  p_amount IN NUMBER
) RETURN BOOLEAN IS
  v_balance NUMBER;
BEGIN
  SELECT Balance INTO v_balance FROM Accounts WHERE AccountID = p_accountID;

  IF v_balance >= p_amount THEN
    RETURN TRUE;
  ELSE
    RETURN FALSE;
  END IF;
EXCEPTION
  WHEN NO_DATA_FOUND THEN
    RETURN FALSE;
END;
/
```

```
SET SERVEROUTPUT ON;

DECLARE
  result BOOLEAN;
BEGIN
  result := HasSufficientBalance(1, 500);

  IF result THEN
    DBMS_OUTPUT.PUT_LINE('Sufficient balance available.');
  ELSE
    DBMS_OUTPUT.PUT_LINE('Insufficient balance.');
  END IF;
END;
/
```

OUTPUT:



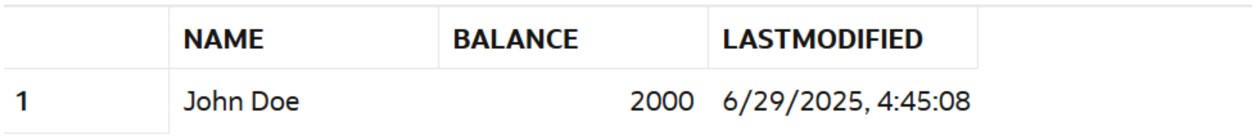
```
Insufficient balance.
```

Exercise 5: Triggers

Scenario 1:

```
CREATE OR REPLACE TRIGGER UpdateCustomerLastModified
BEFORE UPDATE ON Customers
FOR EACH ROW
BEGIN
  :NEW.LastModified := SYSDATE;
END;
/

UPDATE Customers
SET Balance = Balance + 1000
WHERE CustomerID = 1;

SELECT Name, Balance, LastModified
FROM Customers
WHERE CustomerID = 1;
```

OUTPUT:

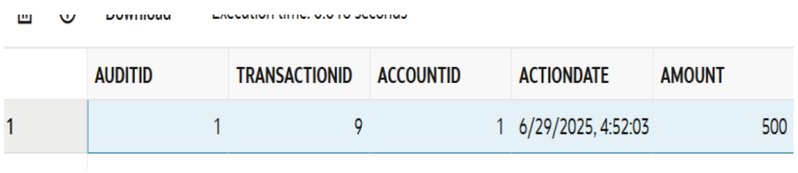


	NAME	BALANCE	LASTMODIFIED
1	John Doe	2000	6/29/2025, 4:45:08

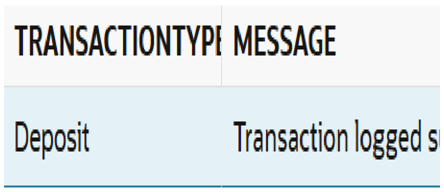
Scenario 2:

```
DROP TABLE AuditLog;

CREATE OR REPLACE TRIGGER LogTransaction

AFTER INSERT ON Transactions

FOR EACH ROW

BEGIN

  INSERT INTO AuditLog (

    TransactionID, AccountID, ActionDate,

    Amount, TransactionType, Message

  )

  VALUES (

    :NEW.TransactionID, :NEW.AccountID, SYSDATE,

    :NEW.Amount, :NEW.TransactionType,

    'Transaction logged successfully'

  );

END;

/

INSERT INTO Transactions (TransactionID, AccountID, TransactionDate, Amount,
TransactionType)

VALUES (9, 1, SYSDATE, 500, 'Deposit');

SELECT * FROM AuditLog WHERE TransactionID = 9;
```

OUTPUT:



	AUDITID	TRANSACTIONID	ACCOUNTID	ACTIONDATE	AMOUNT
1	1	9	1	6/29/2025, 4:52:03	500

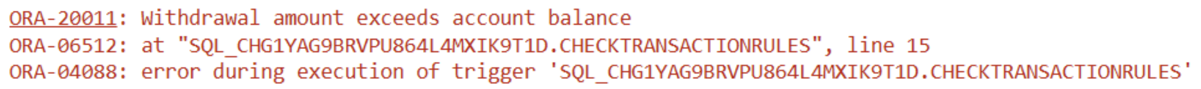


TRANSACTIONTYPI	MESSAGE
Deposit	Transaction logged s

Scenario 3:

```
CREATE OR REPLACE TRIGGER CheckTransactionRules
BEFORE INSERT ON Transactions
FOR EACH ROW
DECLARE
  v_balance NUMBER;
BEGIN
  -- Check deposits are positive
  IF :NEW.TransactionType = 'Deposit' THEN
    IF :NEW.Amount <= 0 THEN
      RAISE_APPLICATION_ERROR(-20010, 'Deposit amount must be positive');
    END IF;

  ELSIF :NEW.TransactionType = 'Withdrawal' THEN
    -- Check sufficient balance for withdrawal
    SELECT Balance INTO v_balance FROM Accounts WHERE AccountID =
:NEW.AccountID;
    IF :NEW.Amount > v_balance THEN
      RAISE_APPLICATION_ERROR(-20011, 'Withdrawal amount exceeds account balance');
    END IF;
  ELSE
    RAISE_APPLICATION_ERROR(-20012, 'Invalid transaction type');
  END IF;
END;
/

INSERT INTO Transactions (TransactionID, AccountID, TransactionDate, Amount,
TransactionType)
VALUES (13, 1, SYSDATE, 1000000, 'Withdrawal');
```

OUTPUT:



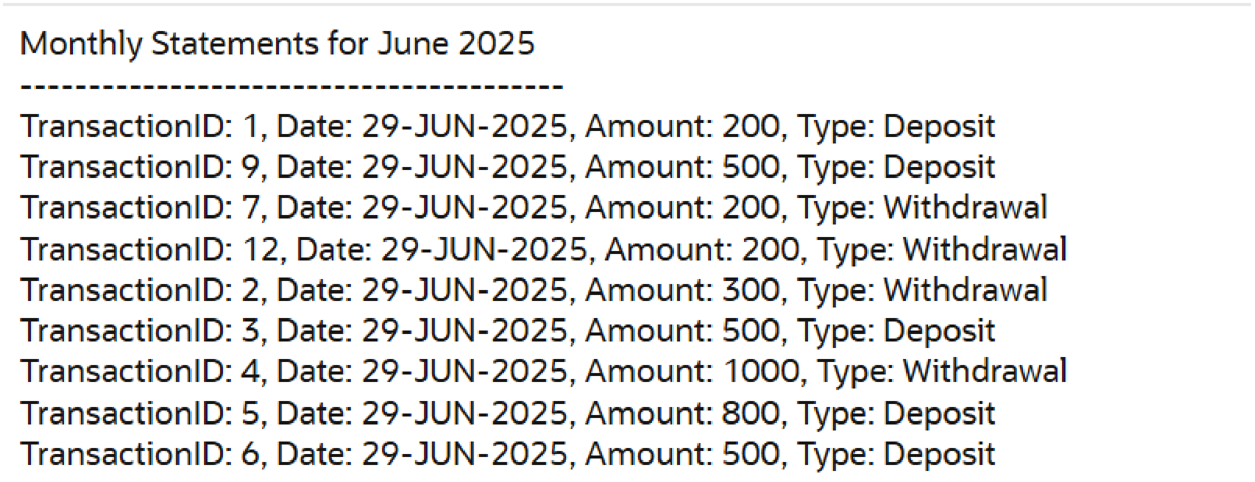
```
ORA-20011: Withdrawal amount exceeds account balance
ORA-06512: at "SQL_CHG1YAG9BRVPU864L4MXIK9T1D.CHECKTRANSACTIONRULES", line 15
ORA-04088: error during execution of trigger 'SQL_CHG1YAG9BRVPU864L4MXIK9T1D.CHECKTRANSACTIONRULES'
```

Exercise 6: Cursors

Scenario 1:

```
SET SERVEROUTPUT ON;

DECLARE
  CURSOR GenerateMonthlyStatements IS
    SELECT c.CustomerID, c.Name, t.TransactionID, t.TransactionDate, t.Amount,
t.TransactionType
    FROM Customers c
    JOIN Accounts a ON c.CustomerID = a.CustomerID
    JOIN Transactions t ON a.AccountID = t.AccountID
    WHERE TRUNC(t.TransactionDate, 'MM') = TRUNC(SYSDATE, 'MM')
    ORDER BY c.CustomerID, t.TransactionDate;

  v_currentCustomerID Customers.CustomerID%TYPE := NULL;
BEGIN
  DBMS_OUTPUT.PUT_LINE('Monthly Statements for ' || TO_CHAR(SYSDATE, 'Month
YYYY'));
  DBMS_OUTPUT.PUT_LINE('----------------------------------------');

  FOR rec IN GenerateMonthlyStatements LOOP
    IF v_currentCustomerID != rec.CustomerID THEN
      -- New customer header
      DBMS_OUTPUT.PUT_LINE('Customer ID: ' || rec.CustomerID || ' | Name: ' || rec.Name);
      v_currentCustomerID := rec.CustomerID;
    END IF;

    DBMS_OUTPUT.PUT_LINE('  TransactionID: ' || rec.TransactionID ||
                     ', Date: ' || TO_CHAR(rec.TransactionDate, 'DD-MON-YYYY') ||
                     ', Amount: ' || rec.Amount ||
                        ', Type: ' || rec.TransactionType);
  END LOOP;
END;
/
```

OUTPUT:

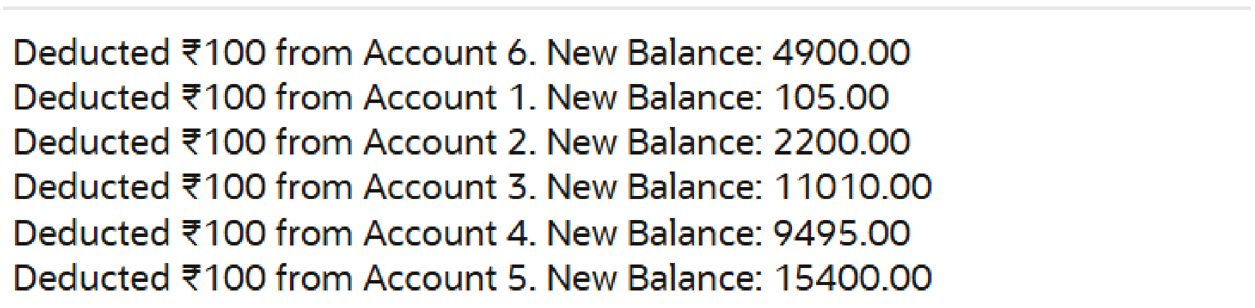


```
Monthly Statements for June 2025
----------------------------------------
TransactionID: 1, Date: 29-JUN-2025, Amount: 200, Type: Deposit
TransactionID: 9, Date: 29-JUN-2025, Amount: 500, Type: Deposit
TransactionID: 7, Date: 29-JUN-2025, Amount: 200, Type: Withdrawal
TransactionID: 12, Date: 29-JUN-2025, Amount: 200, Type: Withdrawal
TransactionID: 2, Date: 29-JUN-2025, Amount: 300, Type: Withdrawal
TransactionID: 3, Date: 29-JUN-2025, Amount: 500, Type: Deposit
TransactionID: 4, Date: 29-JUN-2025, Amount: 1000, Type: Withdrawal
TransactionID: 5, Date: 29-JUN-2025, Amount: 800, Type: Deposit
TransactionID: 6, Date: 29-JUN-2025, Amount: 500, Type: Deposit
```

Scenario 2:

```
SET SERVEROUTPUT ON;

DECLARE
  CURSOR ApplyAnnualFee IS
    SELECT AccountID, Balance FROM Accounts;

  v_fee CONSTANT NUMBER := 100;  -- Annual maintenance fee amount

BEGIN
  FOR acc IN ApplyAnnualFee LOOP
    IF acc.Balance >= v_fee THEN
      UPDATE Accounts
      SET Balance = Balance - v_fee,
          LastModified = SYSDATE
      WHERE AccountID = acc.AccountID;

      DBMS_OUTPUT.PUT_LINE('Deducted ₹' || v_fee || ' from Account ' || acc.AccountID ||
                         '. New Balance: ' || TO_CHAR(acc.Balance - v_fee, '999999.99'));
    ELSE
      DBMS_OUTPUT.PUT_LINE('Account ' || acc.AccountID || ' has insufficient balance for fee
deduction.');
    END IF;
  END LOOP;

  COMMIT;
END;
/
```

OUTPUT:

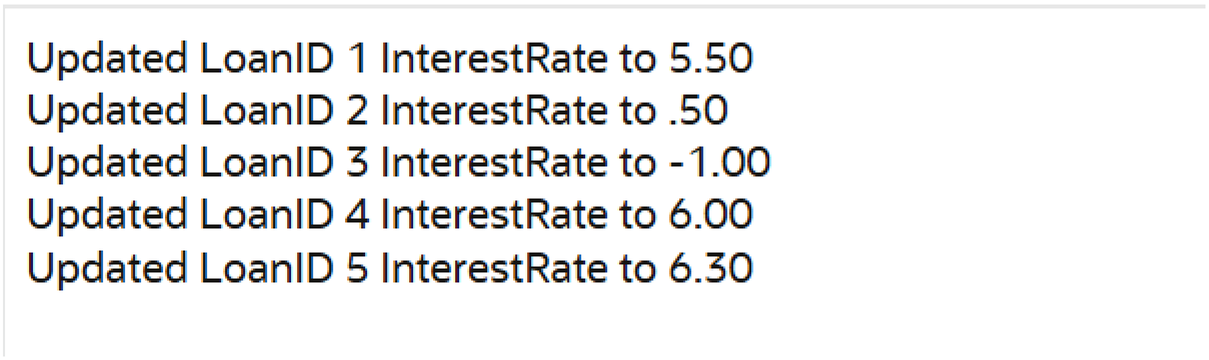


```
Deducted ₹100 from Account 6. New Balance: 4900.00
Deducted ₹100 from Account 1. New Balance: 105.00
Deducted ₹100 from Account 2. New Balance: 2200.00
Deducted ₹100 from Account 3. New Balance: 11010.00
Deducted ₹100 from Account 4. New Balance: 9495.00
Deducted ₹100 from Account 5. New Balance: 15400.00
```

Scenario 3:

```
SET SERVEROUTPUT ON;

DECLARE
  CURSOR UpdateLoanInterestRates IS
    SELECT LoanID, InterestRate FROM Loans;

  v_newInterestRate NUMBER;
BEGIN
  FOR loan_rec IN UpdateLoanInterestRates LOOP
    -- Example policy: increase interest rate by 0.5% if current rate < 7%
    IF loan_rec.InterestRate < 7 THEN
      v_newInterestRate := loan_rec.InterestRate + 0.5;
    ELSE
      v_newInterestRate := loan_rec.InterestRate;
    END IF;

    UPDATE Loans
    SET InterestRate = v_newInterestRate
    WHERE LoanID = loan_rec.LoanID;

    DBMS_OUTPUT.PUT_LINE('Updated LoanID ' || loan_rec.LoanID ||
                         ' InterestRate to ' || TO_CHAR(v_newInterestRate, '9.99'));
  END LOOP;

  COMMIT;
END;
/
```

OUTPUT:



```
Updated LoanID 1 InterestRate to 5.50
Updated LoanID 2 InterestRate to .50
Updated LoanID 3 InterestRate to -1.00
Updated LoanID 4 InterestRate to 6.00
Updated LoanID 5 InterestRate to 6.30
```

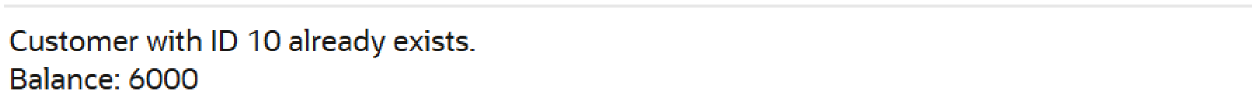
Exercise 7: Packages

Scenario 1:

```
CREATE OR REPLACE PACKAGE CustomerManagement AS
  PROCEDURE AddCustomer(
    p_CustomerID IN NUMBER,
    p_Name IN VARCHAR2,
    p_DOB IN DATE,
    p_Balance IN NUMBER
  );

  PROCEDURE UpdateCustomerBalance(
    p_CustomerID IN NUMBER,
    p_NewBalance IN NUMBER
  );

  FUNCTION GetCustomerBalance(
    p_CustomerID IN NUMBER
  ) RETURN NUMBER;
END CustomerManagement;
/
CREATE OR REPLACE PACKAGE BODY CustomerManagement AS

  PROCEDURE AddCustomer(
    p_CustomerID 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_CustomerID, p_Name, p_DOB, p_Balance, SYSDATE);
  EXCEPTION
    WHEN DUP_VAL_ON_INDEX THEN
      DBMS_OUTPUT.PUT_LINE('Customer with ID ' || p_CustomerID || ' already exists.');
  END AddCustomer;

  PROCEDURE UpdateCustomerBalance(
    p_CustomerID IN NUMBER,
    p_NewBalance IN NUMBER
  ) IS
  BEGIN
    UPDATE Customers
    SET Balance = p_NewBalance,
        LastModified = SYSDATE
    WHERE CustomerID = p_CustomerID;

    IF SQL%ROWCOUNT = 0 THEN
      DBMS_OUTPUT.PUT_LINE('Customer ID ' || p_CustomerID || ' not found.');
    END IF;
  END UpdateCustomerBalance;

  FUNCTION GetCustomerBalance(
    p_CustomerID IN NUMBER
  ) RETURN NUMBER IS
    v_Balance NUMBER;
  BEGIN
    SELECT Balance INTO v_Balance FROM Customers WHERE CustomerID =
p_CustomerID;
    RETURN v_Balance;
  EXCEPTION
```

```
    WHEN NO_DATA_FOUND THEN
      RETURN NULL;
  END GetCustomerBalance;

END CustomerManagement;
/
SET SERVEROUTPUT ON;

BEGIN
  CustomerManagement.AddCustomer(10, 'Alice Wonderland', TO_DATE('1995-08-25', 'YYYY-MM-DD'), 5000);
  CustomerManagement.UpdateCustomerBalance(10, 6000);

  DBMS_OUTPUT.PUT_LINE('Balance: ' || CustomerManagement.GetCustomerBalance(10));
END;
/
```

OUTPUT:



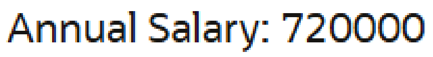
```
Customer with ID 10 already exists.
Balance: 6000
```

Scenario 2:

```
CREATE OR REPLACE PACKAGE EmployeeManagement AS

  PROCEDURE HireEmployee(
    p_EmployeeID IN NUMBER,
    p_Name IN VARCHAR2,
    p_Position IN VARCHAR2,
    p_Salary IN NUMBER,
    p_Department IN VARCHAR2,
    p_HireDate IN DATE
  );

  PROCEDURE UpdateEmployeeDetails(
    p_EmployeeID IN NUMBER,
    p_Position IN VARCHAR2,
    p_Salary IN NUMBER,
    p_Department IN VARCHAR2
  );

  FUNCTION CalculateAnnualSalary(
    p_EmployeeID IN NUMBER
  ) RETURN NUMBER;

END EmployeeManagement;
/
CREATE OR REPLACE PACKAGE BODY EmployeeManagement AS

  PROCEDURE HireEmployee(
    p_EmployeeID IN NUMBER,
    p_Name IN VARCHAR2,
    p_Position IN VARCHAR2,
    p_Salary IN NUMBER,
    p_Department IN VARCHAR2,
    p_HireDate IN DATE
  ) IS
  BEGIN
    INSERT INTO Employees (EmployeeID, Name, Position, Salary, Department, HireDate)
    VALUES (p_EmployeeID, p_Name, p_Position, p_Salary, p_Department, p_HireDate);
  EXCEPTION
    WHEN DUP_VAL_ON_INDEX THEN
      DBMS_OUTPUT.PUT_LINE('Employee with ID ' || p_EmployeeID || ' already exists.');
  END HireEmployee;

  PROCEDURE UpdateEmployeeDetails(
    p_EmployeeID IN NUMBER,
    p_Position IN VARCHAR2,
    p_Salary IN NUMBER,
    p_Department IN VARCHAR2
  ) IS
  BEGIN
    UPDATE Employees
    SET Position = p_Position,
        Salary = p_Salary,
        Department = p_Department
    WHERE EmployeeID = p_EmployeeID;

    IF SQL%ROWCOUNT = 0 THEN
      DBMS_OUTPUT.PUT_LINE('Employee ID ' || p_EmployeeID || ' not found.');
    END IF;
  END UpdateEmployeeDetails;
```

```
  FUNCTION CalculateAnnualSalary(
    p_EmployeeID IN NUMBER
  ) RETURN NUMBER IS
    v_Salary NUMBER;
  BEGIN
    SELECT Salary INTO v_Salary FROM Employees WHERE EmployeeID = p_EmployeeID;
    RETURN v_Salary * 12;
  EXCEPTION
    WHEN NO_DATA_FOUND THEN
      RETURN NULL;
  END CalculateAnnualSalary;

END EmployeeManagement;
/
SET SERVEROUTPUT ON;

BEGIN
  EmployeeManagement.HireEmployee(100, 'David Green', 'Analyst', 50000, 'Finance',
SYSDATE);
  EmployeeManagement.UpdateEmployeeDetails(100, 'Senior Analyst', 60000, 'Finance');
  DBMS_OUTPUT.PUT_LINE('Annual Salary: ' ||
EmployeeManagement.CalculateAnnualSalary(100));
END;
/
```

OUTPUT:



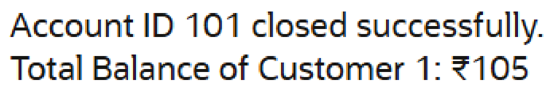
```
Annual Salary: 720000
```

Scenario 1:

```
CREATE OR REPLACE PACKAGE AccountOperations AS

  PROCEDURE OpenAccount(
    p_AccountID IN NUMBER,
    p_CustomerID IN NUMBER,
    p_AccountType IN VARCHAR2,
    p_InitialBalance IN NUMBER
  );

  PROCEDURE CloseAccount(
    p_AccountID IN NUMBER
  );

  FUNCTION GetTotalBalance(
    p_CustomerID IN NUMBER
  ) RETURN NUMBER;

END AccountOperations;
/
CREATE OR REPLACE PACKAGE BODY AccountOperations AS

  PROCEDURE OpenAccount(
    p_AccountID IN NUMBER,
    p_CustomerID IN NUMBER,
    p_AccountType IN VARCHAR2,
    p_InitialBalance IN NUMBER
  ) IS
  BEGIN
    INSERT INTO Accounts (AccountID, CustomerID, AccountType, Balance, LastModified)
    VALUES (p_AccountID, p_CustomerID, p_AccountType, p_InitialBalance, SYSDATE);
  EXCEPTION
    WHEN DUP_VAL_ON_INDEX THEN
      DBMS_OUTPUT.PUT_LINE('Account with ID ' || p_AccountID || ' already exists.');
    WHEN OTHERS THEN
      DBMS_OUTPUT.PUT_LINE('Error: ' || SQLERRM);
  END OpenAccount;

  PROCEDURE CloseAccount(
    p_AccountID IN NUMBER
  ) IS
  BEGIN
    DELETE FROM Accounts WHERE AccountID = p_AccountID;
    IF SQL%ROWCOUNT = 0 THEN
      DBMS_OUTPUT.PUT_LINE('No account found with ID ' || p_AccountID);
    ELSE
      DBMS_OUTPUT.PUT_LINE('Account ID ' || p_AccountID || ' closed successfully.');
    END IF;
  END CloseAccount;

  FUNCTION GetTotalBalance(
    p_CustomerID IN NUMBER
  ) RETURN NUMBER IS
    v_total NUMBER := 0;
  BEGIN
    SELECT NVL(SUM(Balance), 0)
    INTO v_total
    FROM Accounts
    WHERE CustomerID = p_CustomerID;
```

```
    RETURN v_total;
  EXCEPTION
    WHEN NO_DATA_FOUND THEN
      RETURN 0;
  END GetTotalBalance;

END AccountOperations;
/
SET SERVEROUTPUT ON;

BEGIN
  AccountOperations.OpenAccount(101, 1, 'Savings', 2500);
  AccountOperations.CloseAccount(101);

  DBMS_OUTPUT.PUT_LINE('Total Balance of Customer 1: ₹' ||
AccountOperations.GetTotalBalance(1));
END;
/
```

OUTPUT:



```
Account ID 101 closed successfully.
Total Balance of Customer 1: ₹105
```