EXERCISE 1 :

(Scenario 1) :

```
BEGIN
 FOR tbl IN (
  SELECT table_name
  FROM user_tables
  WHERE table_name IN ('CUSTOMERS', 'LOANS')
 ) LOOP
  EXECUTE IMMEDIATE 'DROP TABLE ' || tbl.table_name || ' CASCADE CONSTRAINTS';
 END LOOP;
END;
/

CREATE TABLE Customers (
  CustomerID NUMBER PRIMARY KEY,
  Name VARCHAR2(100),
  DOB DATE,
  Balance NUMBER,
  LastModified DATE
);

CREATE TABLE Loans (
  LoanID NUMBER PRIMARY KEY,
  CustomerID NUMBER,
  LoanAmount NUMBER,
  InterestRate NUMBER,
```

```
  StartDate DATE,
  EndDate DATE,
  FOREIGN KEY (CustomerID) REFERENCES Customers(CustomerID)
);

INSERT INTO Customers VALUES (1, 'Shashank', TO_DATE('19500101', 'YYYYMMDD'), 8000, SYSDATE);
INSERT INTO Customers VALUES (2, 'Gokul', TO_DATE('19950610', 'YYYYMMDD'), 12000, SYSDATE);
INSERT INTO Customers VALUES (3, 'Sivesh', TO_DATE('19981125', 'YYYYMMDD'), 3000, SYSDATE);

INSERT INTO Loans VALUES (101, 1, 10000, 5.5, SYSDATE, ADD_MONTHS(SYSDATE, 60));
INSERT INTO Loans VALUES (102, 2, 15000, 6.0, SYSDATE, ADD_MONTHS(SYSDATE, 48));
INSERT INTO Loans VALUES (103, 3, 20000, 4.5, SYSDATE, ADD_MONTHS(SYSDATE, 36));

COMMIT;

SET SERVEROUTPUT ON;

DECLARE
 v_dob DATE;
 v_age NUMBER;
 v_name VARCHAR2(100);
BEGIN
 FOR loan_rec IN (SELECT * FROM Loans) LOOP
  SELECT DOB, Name INTO v_dob, v_name FROM Customers WHERE CustomerID = loan_rec.CustomerID;
  v_age := TRUNC(MONTHS_BETWEEN(SYSDATE, v_dob) / 12);

  IF v_age > 60 THEN
   UPDATE Loans
```

```
    SET InterestRate = InterestRate  1
    WHERE LoanID = loan_rec.LoanID;

    DBMS_OUTPUT.PUT_LINE('✓ Discount applied → ID: ' || loan_rec.CustomerID || ', Name: ' ||
v_name || ', Age: ' || v_age);
   ELSE
    DBMS_OUTPUT.PUT_LINE('✗ No discount → ID: ' || loan_rec.CustomerID || ', Name: ' ||
v_name || ', Age: ' || v_age);
   END IF;
 END LOOP;

 COMMIT;
END;
/
```

OUTPUT :

✓ Discount applied → ID: 1, Name: Shashank, Age: 75

✗ No discount → ID: 2, Name: Gokul, Age: 30

✗ No discount → ID: 3, Name: Sivesh, Age: 26

(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(' Promoted to VIP → ID: ' || cust.CustomerID || ', Name: ' ||
cust.Name || ', Balance: ' || cust.Balance);
   ELSE
     DBMS_OUTPUT.PUT_LINE('  Not VIP → ID: ' || cust.CustomerID || ', Name: ' || cust.Name || ',
Balance: ' || cust.Balance);
   END IF;
  END LOOP;

  COMMIT;
END;
/

OUTPUT :

  Not VIP → ID: 1, Name: Shashank, Balance: 8000
 Promoted to VIP → ID: 2, Name: Gokul, Balance: 12000
  Not VIP → ID: 3, Name: Sivesh, Balance: 3000

      (  Scenario 3  )

SET SERVEROUTPUT ON;

BEGIN
  FOR loan_rec IN (
```

```
  SELECT LoanID, CustomerID, EndDate
  FROM Loans
  WHERE EndDate BETWEEN SYSDATE AND SYSDATE + 30
 ) LOOP
  DECLARE
   v_name Customers.Name%TYPE;
   v_days_left NUMBER;
  BEGIN
   SELECT Name INTO v_name
   FROM Customers
   WHERE CustomerID = loan_rec.CustomerID;

   v_days_left := TRUNC(loan_rec.EndDate  SYSDATE);

   DBMS_OUTPUT.PUT_LINE(' Reminder → Loan: ' || loan_rec.LoanID ||
                ', Name: ' || v_name ||
                ', ID: ' || loan_rec.CustomerID ||
                ', Due in ' || v_days_left || ' days');
  END;
 END LOOP;
END;
/
```

OUTPUT :

```
 Reminder → Loan: 101, Name: Shashank, ID: 1, Due in 10 days
 Reminder → Loan: 102, Name: Gokul, ID: 2, Due in 25 days
```

EXERCISE 2 :

SCENERIO 1

```
CREATE OR REPLACE PROCEDURE SafeTransferFunds (
 p_from_acct IN NUMBER,
 p_to_acct   IN NUMBER,
 p_amount    IN NUMBER
)
IS
 v_balance NUMBER;
BEGIN
 SELECT Balance INTO v_balance FROM Accounts WHERE AccountID = p_from_acct;

 IF v_balance < p_amount THEN
  RAISE_APPLICATION_ERROR(20001, ' Insufficient funds!');
 END IF;

 UPDATE Accounts SET Balance = Balance  p_amount WHERE AccountID = p_from_acct;
 UPDATE Accounts SET Balance = Balance + p_amount WHERE AccountID = p_to_acct;

 COMMIT;

 DBMS_OUTPUT.PUT_LINE(' Funds transferred: ' || p_amount ||
                ' from Account ' || p_from_acct ||
                ' to Account ' || p_to_acct);
EXCEPTION
 WHEN OTHERS THEN
```

```
    ROLLBACK;
    DBMS_OUTPUT.PUT_LINE(' Transfer failed: ' || SQLERRM);
END;
/



BEGIN
  SafeTransferFunds(1, 2, 500);
END;
/
```

OUTPUT :

Funds transferred: 500 from Account 1 to Account 2

SCENERIO 2

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

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

```
 END IF;

 COMMIT;

 DBMS_OUTPUT.PUT_LINE(' Salary updated for Employee ID: ' || p_emp_id);
EXCEPTION
 WHEN OTHERS THEN
  DBMS_OUTPUT.PUT_LINE(' Error updating salary: ' || SQLERRM);
  ROLLBACK;
END;
/
```

Execution

```
BEGIN
 UpdateSalary(2, 15);
END;
/
```

OUTPUT :

Salary updated for Employee ID: 2

SCENERIO 3

```
CREATE OR REPLACE PROCEDURE AddNewCustomer (
 p_id   IN NUMBER,
 p_name IN VARCHAR2,
```

```
  p_dob  IN DATE,
  p_bal  IN NUMBER
)
IS
BEGIN
  INSERT INTO Customers (CustomerID, Name, DOB, Balance, LastModified)
  VALUES (p_id, p_name, p_dob, p_bal, SYSDATE);

  COMMIT;

  DBMS_OUTPUT.PUT_LINE(' New customer added → ID: ' || p_id || ', Name: ' || p_name);
EXCEPTION
  WHEN DUP_VAL_ON_INDEX THEN
    DBMS_OUTPUT.PUT_LINE(' Customer ID already exists: ' || p_id);
    ROLLBACK;
  WHEN OTHERS THEN
    DBMS_OUTPUT.PUT_LINE(' Error adding customer: ' || SQLERRM);
    ROLLBACK;
END;
/
```

Execution :

```
BEGIN
  AddNewCustomer(4, 'Charan', TO_DATE('20020520', 'YYYYMMDD'), 7000);
END;
/
```

OUTPUT :

```
 New customer added → ID: 4, Name: Charan
```

EXERCISE 3 :

```
CREATE OR REPLACE PROCEDURE ProcessMonthlyInterest
IS
BEGIN

 UPDATE Accounts
 SET Balance = Balance + (Balance * 0.01)
 WHERE AccountType = 'Savings';

 COMMIT;

 DBMS_OUTPUT.PUT_LINE(' Monthly interest applied to all Savings accounts.');
EXCEPTION
 WHEN OTHERS THEN
  ROLLBACK;
  DBMS_OUTPUT.PUT_LINE(' Error: ' || SQLERRM);
END;
/



BEGIN
 ProcessMonthlyInterest;
END;
/
```

OUTPUT :

Monthly interest applied to all Savings accounts.

SCENERIO 2

```
CREATE OR REPLACE PROCEDURE UpdateEmployeeBonus (
  p_dept IN VARCHAR2,
  p_bonus_percent IN NUMBER
)
IS
BEGIN

  UPDATE Employees
  SET Salary = Salary + (Salary * p_bonus_percent / 100)
  WHERE Department = p_dept;

  IF SQL%ROWCOUNT = 0 THEN
    DBMS_OUTPUT.PUT_LINE(' No employees found in department: ' || p_dept);
  ELSE
    DBMS_OUTPUT.PUT_LINE(' Bonus applied to department: ' || p_dept);
  END IF;

  COMMIT;
EXCEPTION
  WHEN OTHERS THEN
    ROLLBACK;
    DBMS_OUTPUT.PUT_LINE(' Error updating bonus: ' || SQLERRM);
END;
```

/

Execution

```
BEGIN
  UpdateEmployeeBonus('IT', 10);
END;
/
```

OUTPUT :

Bonus applied to department: IT

SCENERIO 3

```
CREATE OR REPLACE PROCEDURE TransferFunds (
  p_from_acct IN NUMBER,
  p_to_acct   IN NUMBER,
  p_amount    IN NUMBER
)
IS
  v_balance NUMBER;
BEGIN
   Get source account balance
  SELECT Balance INTO v_balance FROM Accounts WHERE AccountID = p_from_acct;
```

```
  Check if balance is enough
 IF v_balance < p_amount THEN
   RAISE_APPLICATION_ERROR(20001, ' Insufficient funds');
 END IF;

  Transfer the funds
 UPDATE Accounts SET Balance = Balance  p_amount WHERE AccountID = p_from_acct;
 UPDATE Accounts SET Balance = Balance + p_amount WHERE AccountID = p_to_acct;

 COMMIT;
 DBMS_OUTPUT.PUT_LINE(' Amount ' || p_amount || ' transferred from Account ' || p_from_acct
|| ' to ' || p_to_acct);
EXCEPTION
 WHEN OTHERS THEN
   ROLLBACK;
   DBMS_OUTPUT.PUT_LINE(' Transfer failed: ' || SQLERRM);
END;
/
```

Execution :

```
BEGIN
 TransferFunds(1, 2, 1000);
END;
/
```

OUTPUT :

```
Amount 1000 transferred from Account 1 to 2
```

EXERCISE 5 :

SCENARIO 1

Trigger: Automatically updates LastModified

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

ACTION: Update customer to trigger the change

```
UPDATE Customers
SET Balance = Balance + 100
WHERE CustomerID = 1;
```

OUTPUT CHECK:

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

OUTPUT :

```
CustomerID | Name    | LastModified
||
1       | Shashank | 26JUN2025
```

SCENARIO 2

Step 1: Create AuditLog Table

```
CREATE TABLE AuditLog (
  LogID NUMBER GENERATED ALWAYS AS IDENTITY,
  Action VARCHAR2(50),
  TransactionID NUMBER,
  Timestamp DATE
);
```

Step 2: Create Trigger

```
CREATE OR REPLACE TRIGGER LogTransaction
AFTER INSERT ON Transactions
FOR EACH ROW
BEGIN
  INSERT INTO AuditLog (Action, TransactionID, Timestamp)
  VALUES ('INSERT', :NEW.TransactionID, SYSDATE);
END;
/
```

Step 3: Insert sample transaction

```
INSERT INTO Transactions (TransactionID, AccountID, Amount, TransactionType)
VALUES (201, 1, 500, 'Deposit');
```

OUTPUT CHECK:

```
SELECT * FROM AuditLog;
```

OUTPUT :

LogID | Action | TransactionID | Timestamp

|||

1 | INSERT | 201 | 26JUN2025

SCENARIO 3

Step 1: Create Trigger

```
CREATE OR REPLACE TRIGGER CheckTransactionRules
BEFORE INSERT ON Transactions
FOR EACH ROW
DECLARE
 v_balance NUMBER;
BEGIN
 IF :NEW.TransactionType = 'Withdrawal' THEN
  SELECT Balance INTO v_balance FROM Accounts WHERE AccountID = :NEW.AccountID;

  IF :NEW.Amount > v_balance THEN
   RAISE_APPLICATION_ERROR(20001, '❌ Withdrawal exceeds balance');
  END IF;
 ELSIF :NEW.TransactionType = 'Deposit' THEN
  IF :NEW.Amount <= 0 THEN
   RAISE_APPLICATION_ERROR(20002, '❌ Deposit must be positive');
  END IF;
 END IF;
END;
/
```

VALID INSERT (Deposit)

```
INSERT INTO Transactions (TransactionID, AccountID, Amount, TransactionType)
VALUES (202, 1, 1000, 'Deposit');
```

INVALID INSERT (Overdraft Withdrawal)

```
INSERT INTO Transactions (TransactionID, AccountID, Amount, TransactionType)
VALUES (203, 1, 9999999, 'Withdrawal');
```

INVALID INSERT (Negative Deposit)

```
INSERT INTO Transactions (TransactionID, AccountID, Amount, TransactionType)
VALUES (204, 1, 500, 'Deposit');
```

OUTPUT :

Success:

✓ Deposit added: TransactionID 202

Failures:

❌ ORA20001: Withdrawal exceeds balance

❌ ORA20002: Deposit must be positive