SRM

INSTITUTE OF SCIENCE & TECHNOLOGY

(Deemed to be **University** u/s 3 of UGC Act, 1956)

DELHI NCR CAMPUS, GHAZIABAD, (U.P.)

S R M INSTITUTE OF SCIENCE & TECHNOLOGY

(FACULTY OF SCIENCE & HUMANITIES)

DEPARTMENT OF COMPUTER APPLICATIONS

PRACTICAL FILE

INTRODUCTION TO COMPUTER VISION

[PGI25D01J]

MCA(Core) 1ST YEAR, 1ST SEMESTER

Session: 2025-26

SUBMITTED TO:

Anand Kumar
Assistant Professor
Department of Computer Applications

SUBMITTED BY :

RA2532241030005
Vansh Kumar
Department of Computer Applications

SRM
INSTITUTE OF SCIENCE & TECHNOLOGY
(Deemed to be University u/s 3 of UGC Act, 1956)
DELHI NCR CAMPUS, GHAZIABAD, (U.P.)

S R M INSTITUTE OF SCIENCE & TECHNOLOGY

DELHI NCR CAMPUS, MODINAGAR

(FACULTY OF SCIENCE & HUMANITIES)

DEPARTMENT OF COMPUTER APPLICATIONS

Register No. : RA2532241030005

BONAFIDE CERTIFICATE

Certified to be the Bonafide record of the work done by **Vansh Kumar** of MCA **1ST Year/ 1ST** Semester for the Award of **Master's Degree** in the **DEPARTMENT OF COMPUTER APPLICATIONS** for Database Technology **[PCA25C03J]** laboratory during the Academic Year **2025-26.**

SUBJECT IN – CHARGE **HEAD – COMPUTER APPLICATIONS**

Submitted for the University Examination held on __________________________.

INTERNAL EXAMINER - 1 **INTERNAL EXAMINER – 2**

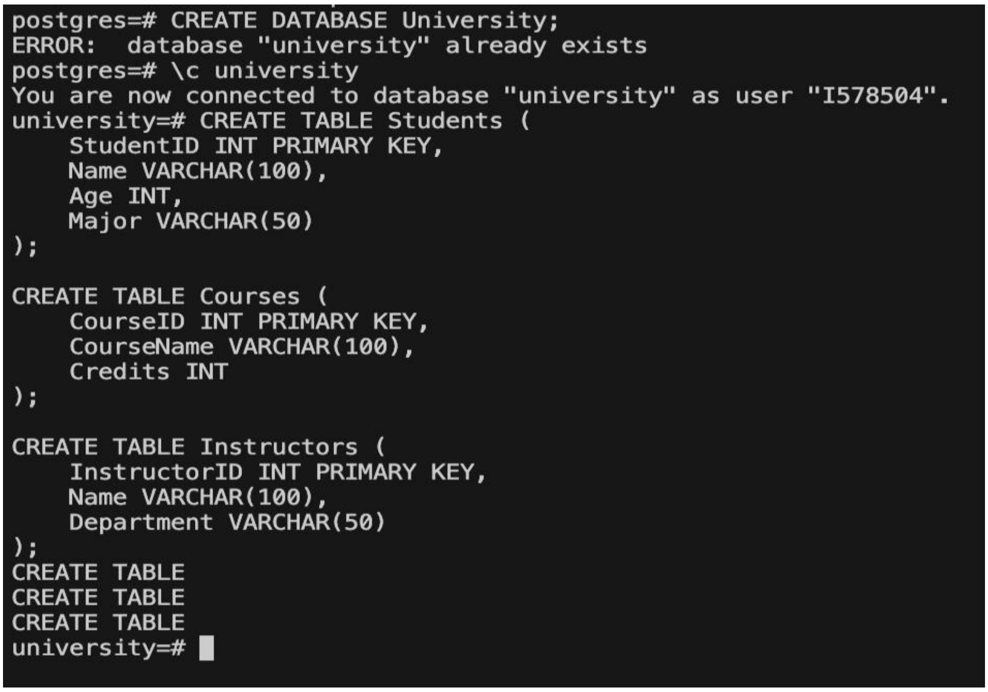
INDEX

Practice 1: Create a Database Schema for University Database

- **Objective:** Design a database schema for a university database to store information about students, courses, and instructors.
- **Source Code:**

```
CREATE DATABASE University;
USE University;

CREATE TABLE Students (
   StudentID INT PRIMARY KEY,
   Name VARCHAR(100),
   Age INT,
   Major VARCHAR(50)
);

CREATE TABLE Courses (
   CourseID INT PRIMARY KEY,CourseName VARCHAR(100), Credits INT
);

CREATE TABLE Instructors (
   InstructorID INT PRIMARY KEY,
   Name VARCHAR(100),
   Department VARCHAR(50)
);
```

- **Output:**



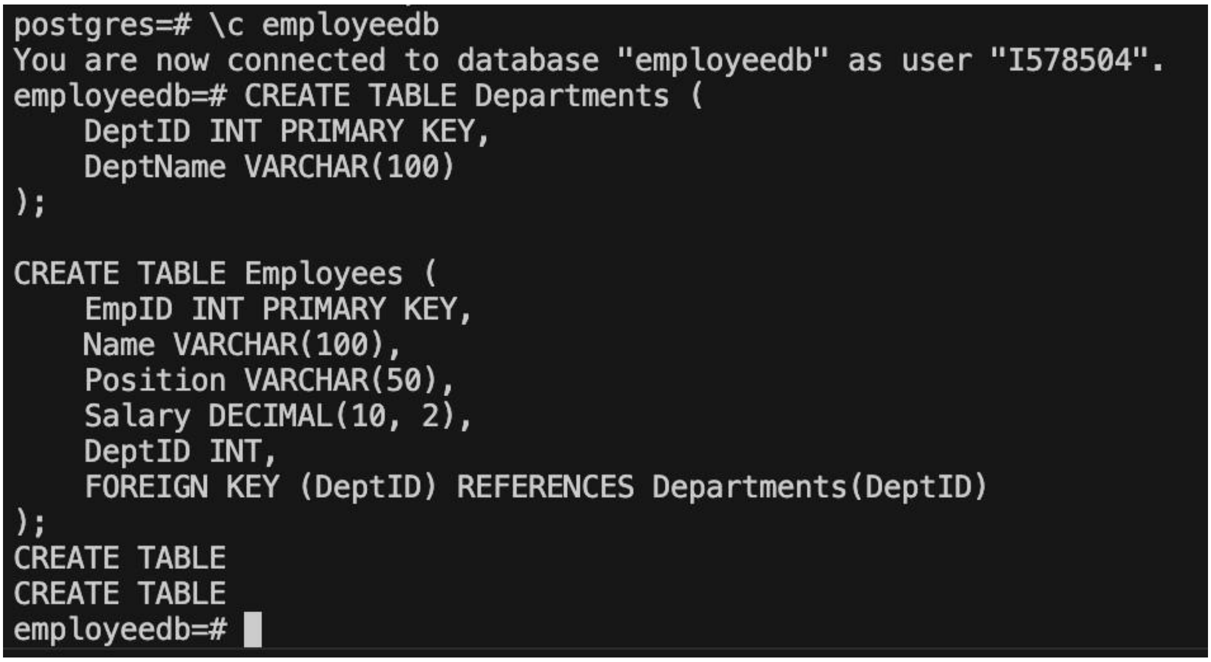
```
postgres=# CREATE DATABASE University;
ERROR:  database "university" already exists
postgres=# \c university
You are now connected to database "university" as user "I578504".
university=# CREATE TABLE Students (
    StudentID INT PRIMARY KEY,
    Name VARCHAR(100),
    Age INT,
    Major VARCHAR(50)
);

CREATE TABLE Courses (
    CourseID INT PRIMARY KEY,
    CourseName VARCHAR(100),
    Credits INT
);

CREATE TABLE Instructors (
    InstructorID INT PRIMARY KEY,
    Name VARCHAR(100),
    Department VARCHAR(50)
);
CREATE TABLE
CREATE TABLE
CREATE TABLE
university=#
```

Practice 2: SQL Queries for Employee Database with Key Constraints

- **Objective:** Write SQL queries to create an employee database with primary and foreign key constraints.
- **Source Code:**

```
CREATE DATABASE EmployeeDB;
USE EmployeeDB;

CREATE TABLE Departments (
    DeptID INT PRIMARY KEY,
    DeptName VARCHAR(100)
);

CREATE TABLE Employees (
    EmpID INT PRIMARY KEY,
    Name VARCHAR(100),
    Position VARCHAR(50),
    Salary DECIMAL(10, 2),
    DeptID INT,
    FOREIGN KEY (DeptID) REFERENCES Departments(DeptID)
);
```

- **Output:**



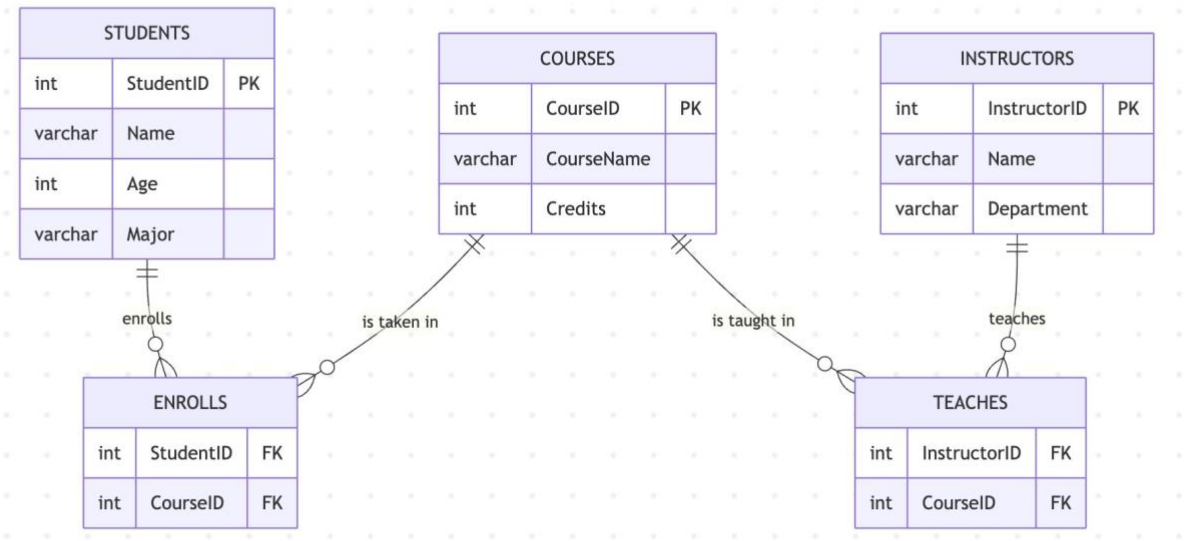
```
postgres=# \c employeedb
You are now connected to database "employeedb" as user "I578504".
employeedb=# CREATE TABLE Departments (
    DeptID INT PRIMARY KEY,
    DeptName VARCHAR(100)
);

CREATE TABLE Employees (
    EmpID INT PRIMARY KEY,
    Name VARCHAR(100),
    Position VARCHAR(50),
    Salary DECIMAL(10, 2),
    DeptID INT,
    FOREIGN KEY (DeptID) REFERENCES Departments(DeptID)
);
CREATE TABLE
CREATE TABLE
employeedb=#
```

Practice 3: Create ER Model for University Database

- **Objective:** Design an Entity-Relationship (ER) model for the university database.
- **Source Code:**

```
CREATE TABLE Students (
   StudentID INT PRIMARY KEY, Name VARCHAR(100), Age INT, Major VARCHAR(50)
);

CREATE TABLE Courses (
   CourseID INT PRIMARY KEY, CourseName VARCHAR(100), Credits INT
);

CREATE TABLE Instructors (
   InstructorID INT PRIMARY KEY, Name VARCHAR(100), Department VARCHAR(50)
);

CREATE TABLE Enrolls (
   StudentID INT, CourseID INT, PRIMARY KEY (StudentID, CourseID),
   FOREIGN KEY (StudentID) REFERENCES Students(StudentID),
   FOREIGN KEY (CourseID) REFERENCES Courses(CourseID)
);

CREATE TABLE Teaches (
   InstructorID INT, CourseID INT,
   PRIMARY KEY (InstructorID, CourseID), FOREIGN KEY (InstructorID) REFERENCES
Instructors(InstructorID), FOREIGN KEY (CourseID) REFERENCES Courses(CourseID)
);
```

- **Output:**



STUDENTS		
int	StudentID	PK
varchar	Name	
int	Age	
varchar	Major	

COURSES		
int	CourseID	PK
varchar	CourseName	
int	Credits	

INSTRUCTORS		
int	InstructorID	PK
varchar	Name	
varchar	Department	

ENROLLS		
int	StudentID	FK
int	CourseID	FK

TEACHES		
int	InstructorID	FK
int	CourseID	FK

Practice 4: Implement DDL, DML Commands

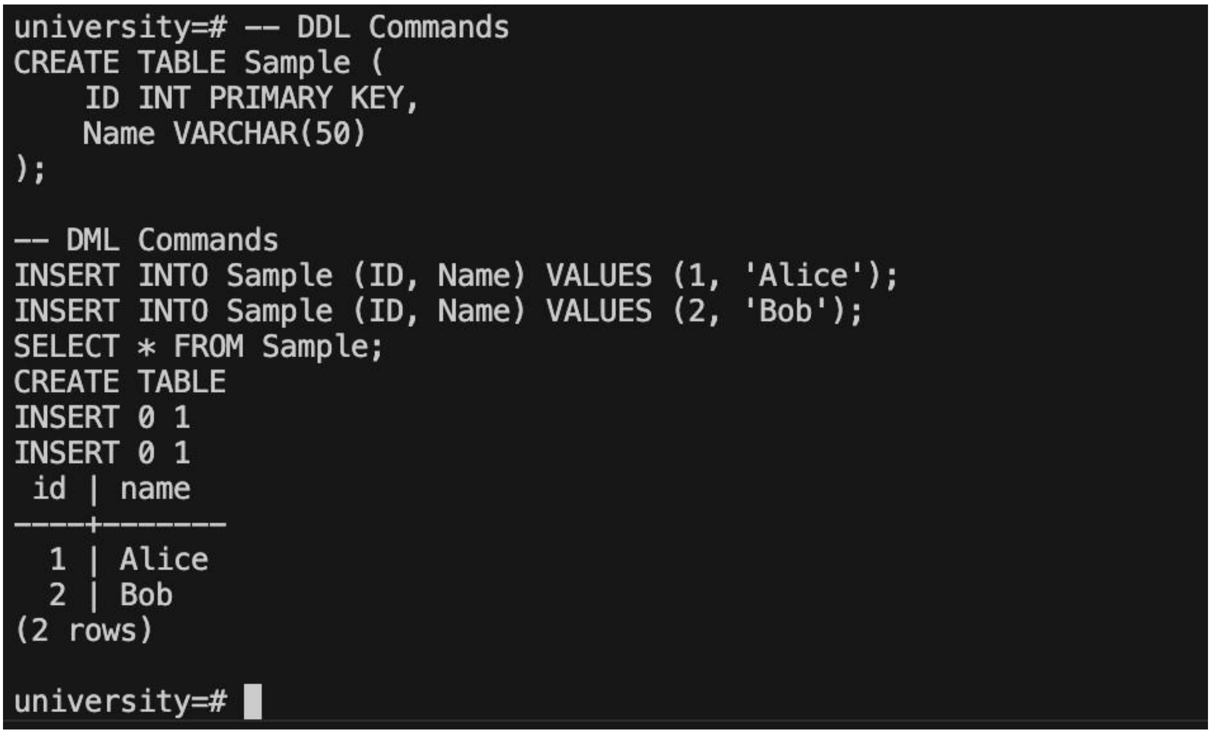
- **Objective: Demonstrate the use of DDL and DML commands**

- **Source Code:**

```
-- DDL Commands
CREATE TABLE Sample (
   ID INT PRIMARY KEY,
   Name VARCHAR(50)
);

-- DML Commands
INSERT INTO Sample (ID, Name) VALUES (1, 'Alice');
 INSERT INTO Sample (ID, Name) VALUES (2, 'Bob');
```

SELECT * FROM Sample;

- **Output:**



```
university=# -- DDL Commands
CREATE TABLE Sample (
    ID INT PRIMARY KEY,
    Name VARCHAR(50)
);

-- DML Commands
INSERT INTO Sample (ID, Name) VALUES (1, 'Alice');
INSERT INTO Sample (ID, Name) VALUES (2, 'Bob');
SELECT * FROM Sample;
CREATE TABLE
INSERT 0 1
INSERT 0 1
 id | name
----+-------
  1 | Alice
  2 | Bob
(2 rows)

university=# 
```

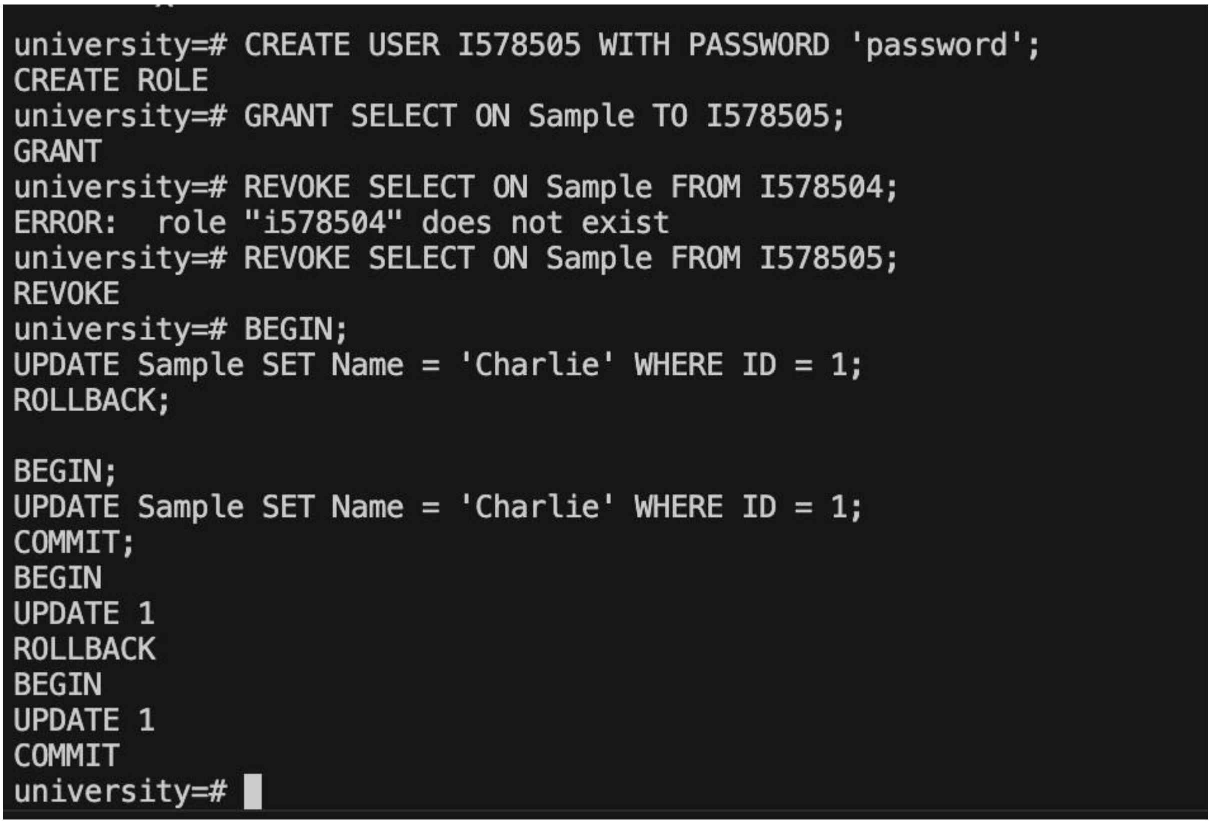
Practice 5: Implement DCL, TCL Commands

- **Objective:** Demonstrate the use of DCL and TCL commands.

- **Source Code:**

```
-- DCL Commands
GRANT SELECT ON Sample TO 'user';
REVOKE SELECT ON Sample FROM 'user';
BEGIN;
UPDATE Sample SET Name = 'Charlie' WHERE ID = 1;
ROLLBACK;

BEGIN;
UPDATE Sample SET Name = 'Charlie' WHERE ID = 1; COMMIT;
```

- **Output:**



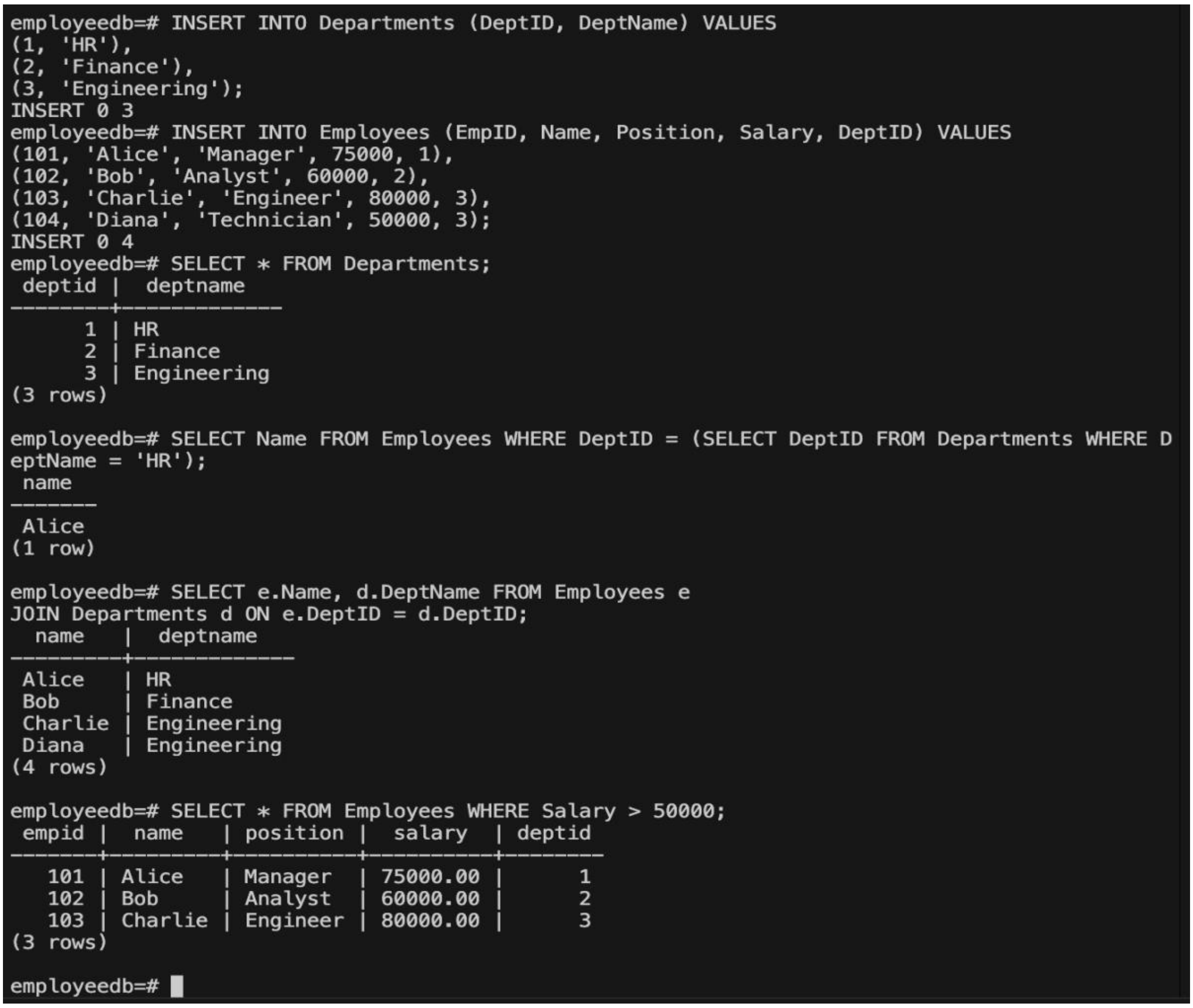
```
university=# CREATE USER I578505 WITH PASSWORD 'password';
CREATE ROLE
university=# GRANT SELECT ON Sample TO I578505;
GRANT
university=# REVOKE SELECT ON Sample FROM I578504;
ERROR:  role "i578504" does not exist
university=# REVOKE SELECT ON Sample FROM I578505;
REVOKE
university=# BEGIN;
UPDATE Sample SET Name = 'Charlie' WHERE ID = 1;
ROLLBACK;

BEGIN;
UPDATE Sample SET Name = 'Charlie' WHERE ID = 1;
COMMIT;
BEGIN
UPDATE 1
ROLLBACK
BEGIN
UPDATE 1
COMMIT
university=# █
```

Practice 6: Implement SQL Subqueries, Joins, and Clauses

- **Objective:** Write SQL queries using subqueries, joins, and clauses.
- **Source Code:**

```
-- Subquery
SELECT Name FROM Employees WHERE DeptID = (SELECT DeptID FROM Departments
WHERE DeptName = 'HR');

-- Join
SELECT e.Name, d.DeptName FROM Employees e
JOIN Departments d ON e.DeptID = d.DeptID;

-- Clause
SELECT * FROM Employees WHERE Salary > 50000;
```

- **Output:**



```
employeedb=# INSERT INTO Departments (DeptID, DeptName) VALUES
(1, 'HR'),
(2, 'Finance'),
(3, 'Engineering');
INSERT 0 3
employeedb=# INSERT INTO Employees (EmpID, Name, Position, Salary, DeptID) VALUES
(101, 'Alice', 'Manager', 75000, 1),
(102, 'Bob', 'Analyst', 60000, 2),
(103, 'Charlie', 'Engineer', 80000, 3),
(104, 'Diana', 'Technician', 50000, 3);
INSERT 0 4
employeedb=# SELECT * FROM Departments;
 deptid |  deptname
--------+-------------
      1 | HR
      2 | Finance
      3 | Engineering
(3 rows)

employeedb=# SELECT Name FROM Employees WHERE DeptID = (SELECT DeptID FROM Departments WHERE D
eptName = 'HR');
 name
-------
 Alice
(1 row)

employeedb=# SELECT e.Name, d.DeptName FROM Employees e
JOIN Departments d ON e.DeptID = d.DeptID;
  name   |  deptname
---------+-------------
 Alice   | HR
 Bob     | Finance
 Charlie | Engineering
 Diana   | Engineering
(4 rows)

employeedb=# SELECT * FROM Employees WHERE Salary > 50000;
 empid |  name   | position |  salary  | deptid
-------+---------+----------+----------+--------
   101 | Alice   | Manager  | 75000.00 |      1
   102 | Bob     | Analyst  | 60000.00 |      2
   103 | Charlie | Engineer | 80000.00 |      3
(3 rows)

employeedb=# █
```

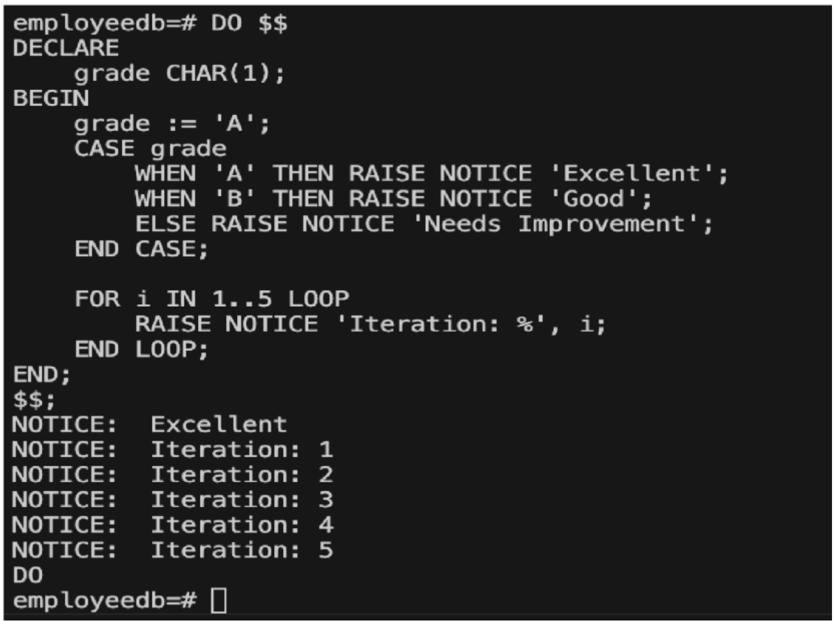
Practice 7: PL/SQL: Case, Loop

- **Objective:** demonstrate the use of CASE and LOOP in PL/SQL.

- **Source Code:**

```
DECLARE
   grade CHAR(1);
BEGIN
   grade := 'A';
   CASE grade
      WHEN 'A' THEN DBMS_OUTPUT.PUT_LINE('Excellent');
      WHEN 'B' THEN DBMS_OUTPUT.PUT_LINE('Good');
      ELSE DBMS_OUTPUT.PUT_LINE('Needs Improvement');
   END CASE;

   FOR i IN 1..5 LOOP
      DBMS_OUTPUT.PUT_LINE('Iteration: ' || i);
   END LOOP;
END;
```

- **Output:**



```
employeedb=# DO $$
DECLARE
    grade CHAR(1);
BEGIN
    grade := 'A';
    CASE grade
        WHEN 'A' THEN RAISE NOTICE 'Excellent';
        WHEN 'B' THEN RAISE NOTICE 'Good';
        ELSE RAISE NOTICE 'Needs Improvement';
    END CASE;

    FOR i IN 1..5 LOOP
        RAISE NOTICE 'Iteration: %', i;
    END LOOP;
END;
$$;
NOTICE:  Excellent
NOTICE:  Iteration: 1
NOTICE:  Iteration: 2
NOTICE:  Iteration: 3
NOTICE:  Iteration: 4
NOTICE:  Iteration: 5
DO
employeedb=# 
```

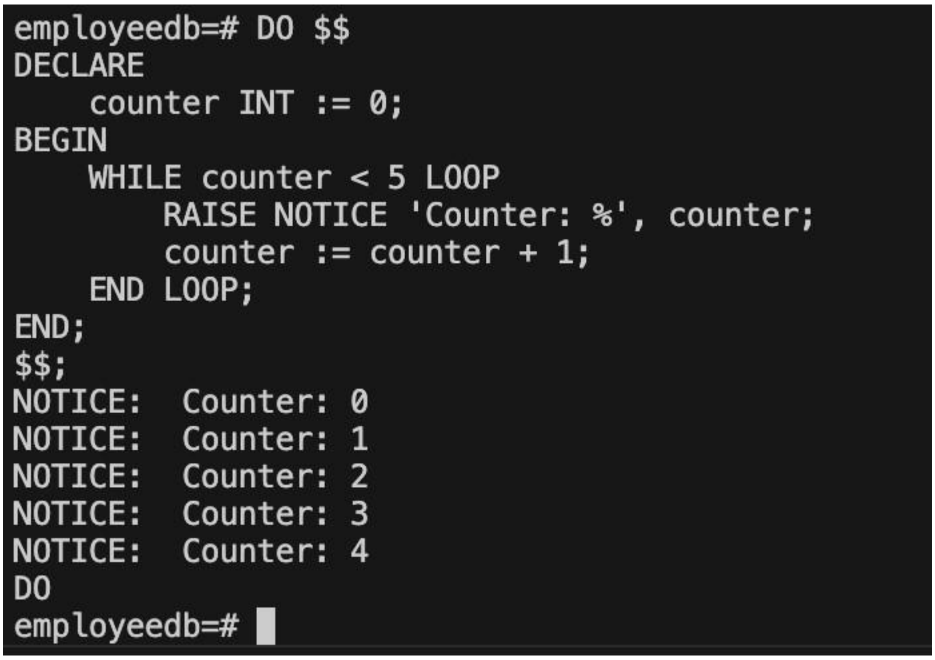
Practice 8: Implementing PL/SQL Conditional Statements, Looping Statements

- **Objective:** Write PL/SQL code for conditional and looping statements.

- **Source Code:**

```
DECLARE
   counter INT := 0;
BEGIN
   WHILE counter < 5 LOOP
      DBMS_OUTPUT.PUT_LINE('Counter: ' || counter);
counter := counter + 1;
   END LOOP; END;
```

- **Output:**



```
employeedb=# DO $$
DECLARE
    counter INT := 0;
BEGIN
    WHILE counter < 5 LOOP
        RAISE NOTICE 'Counter: %', counter;
        counter := counter + 1;
    END LOOP;
END;
$$;
NOTICE:  Counter: 0
NOTICE:  Counter: 1
NOTICE:  Counter: 2
NOTICE:  Counter: 3
NOTICE:  Counter: 4
DO
employeedb=# 
```

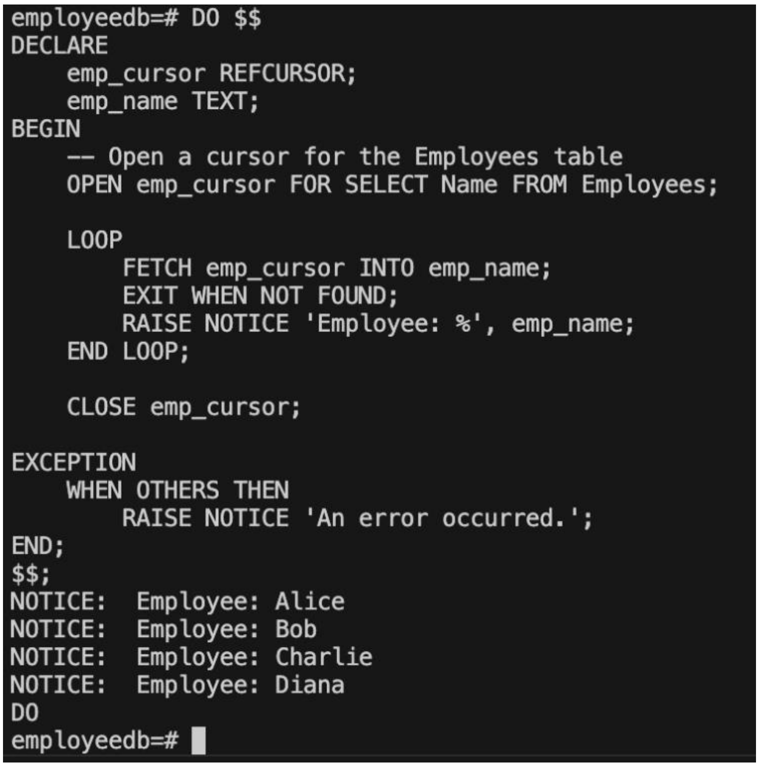
Practice 9: Sample Programs for Cursors and Exceptions

- **Objective:** Demonstrate the use of cursors and exception handling in PL/SQL.

- **Source Code:**

```
DO $$
  DECLARE
     emp_cursor REFCURSOR;
     emp_name TEXT;
  BEGIN
     -- Open a cursor for the Employees table
     OPEN emp_cursor FOR SELECT Name FROM Employees;

     LOOP
        FETCH emp_cursor INTO emp_name;
        EXIT WHEN NOT FOUND;
        RAISE NOTICE 'Employee: %', emp_name;
     END LOOP;

     CLOSE emp_cursor;

  EXCEPTION
     WHEN OTHERS THEN
        RAISE NOTICE 'An error occurred.';
  END;
  $$;
```

- **Output:**



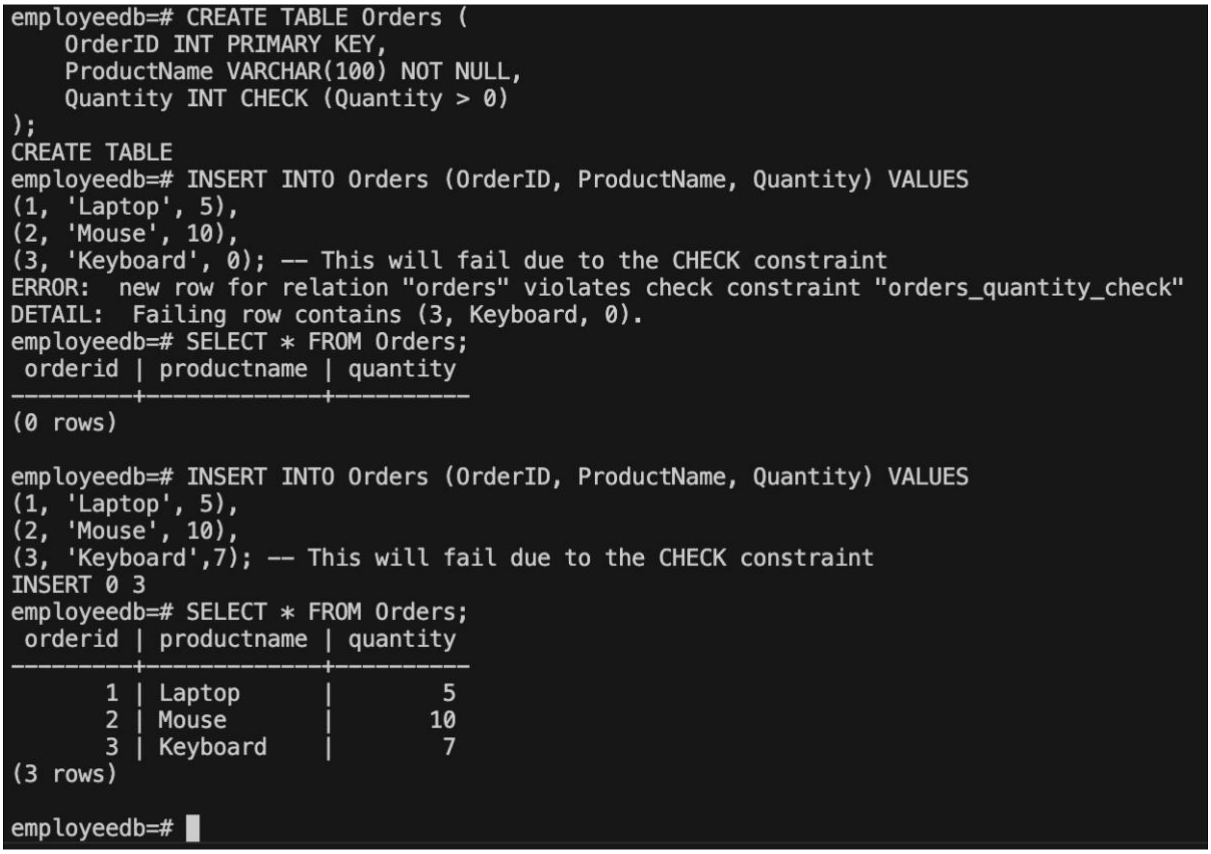
```
employeedb=# DO $$
DECLARE
    emp_cursor REFCURSOR;
    emp_name TEXT;
BEGIN
    -- Open a cursor for the Employees table
    OPEN emp_cursor FOR SELECT Name FROM Employees;

    LOOP
        FETCH emp_cursor INTO emp_name;
        EXIT WHEN NOT FOUND;
        RAISE NOTICE 'Employee: %', emp_name;
    END LOOP;

    CLOSE emp_cursor;

EXCEPTION
    WHEN OTHERS THEN
        RAISE NOTICE 'An error occurred.';
END;
$$;
NOTICE:  Employee: Alice
NOTICE:  Employee: Bob
NOTICE:  Employee: Charlie
NOTICE:  Employee: Diana
DO
employeedb=# █
```

Practice 10: Implement Integrity Constraints

- **Objective:** Demonstrate the use of integrity constraints in SQL.
- **Source Code:**

```
 CREATE TABLE Orders (
   OrderID INT PRIMARY KEY,
   ProductName VARCHAR(100) NOT NULL,
   Quantity INT CHECK (Quantity > 0)
);

INSERT INTO Orders (OrderID, ProductName, Quantity) VALUES
(1, 'Laptop', 5),
(2, 'Mouse', 10),
(3, 'Keyboard', 0); -- This will fail due to the CHECK constraint

SELECT * FROM Orders;
```

- **Output:**



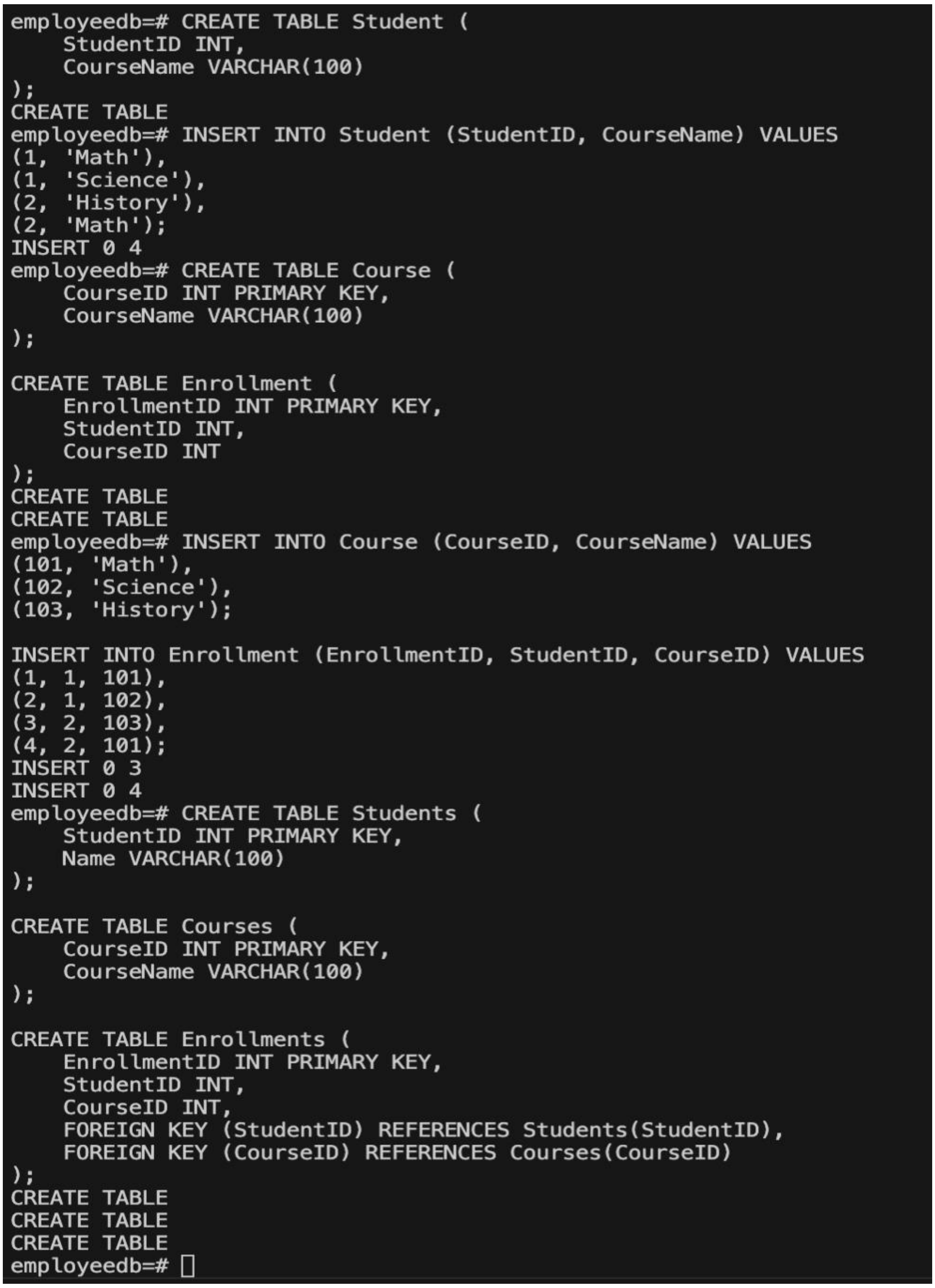
```
employeedb=# CREATE TABLE Orders (
    OrderID INT PRIMARY KEY,
    ProductName VARCHAR(100) NOT NULL,
    Quantity INT CHECK (Quantity > 0)
);
CREATE TABLE
employeedb=# INSERT INTO Orders (OrderID, ProductName, Quantity) VALUES
(1, 'Laptop', 5),
(2, 'Mouse', 10),
(3, 'Keyboard', 0); -- This will fail due to the CHECK constraint
ERROR:  new row for relation "orders" violates check constraint "orders_quantity_check"
DETAIL:  Failing row contains (3, Keyboard, 0).
employeedb=# SELECT * FROM Orders;
 orderid | productname | quantity
---------+-------------+----------
(0 rows)

employeedb=# INSERT INTO Orders (OrderID, ProductName, Quantity) VALUES
(1, 'Laptop', 5),
(2, 'Mouse', 10),
(3, 'Keyboard',7); -- This will fail due to the CHECK constraint
INSERT 0 3
employeedb=# SELECT * FROM Orders;
 orderid | productname | quantity
---------+-------------+----------
       1 | Laptop      |        5
       2 | Mouse       |       10
       3 | Keyboard    |        7
(3 rows)

employeedb=# █
```

Practice 11: Implement First, Second, and Third Normalization Techniques

- **Objective:** Normalize a database to 1NF, 2NF, and 3NF.
- **Source Code:**

```
 -- 1NF: Remove duplicate columns
CREATE TABLE Student ( StudentID INT, CourseName VARCHAR(100)
);

INSERT INTO Student (StudentID, CourseName) VALUES
(1, 'Math'), (1, 'Science'), (2, 'History'), (2, 'Math');

-- 2NF: Remove partial dependencies
CREATE TABLE Course (
   CourseID INT PRIMARY KEY,
   CourseName VARCHAR(100)
);

-- 3NF: Remove transitive dependencies
CREATE TABLE Enrollment (
   EnrollmentID INT PRIMARY KEY,
   StudentID INT,
   CourseID INT
);

INSERT INTO Course (CourseID, CourseName) VALUES
(101, 'Math'),
(102, 'Science'),
(103, 'History');

INSERT INTO Enrollment (EnrollmentID, StudentID, CourseID) VALUES
(1, 1, 101),
(2, 1, 102),
(3, 2, 103),
(4, 2, 101);

CREATE TABLE Students (
   StudentID INT PRIMARY KEY,
   Name VARCHAR(100)
);

CREATE TABLE Courses (
   CourseID INT PRIMARY KEY,
   CourseName VARCHAR(100)
);

CREATE TABLE Enrollments (
   EnrollmentID INT PRIMARY KEY,
   StudentID INT,
   CourseID INT,
   FOREIGN KEY (StudentID) REFERENCES Students(StudentID),
   FOREIGN KEY (CourseID) REFERENCES Courses(CourseID)
);
```

- **Output:**



```
employeedb=# CREATE TABLE Student (
    StudentID INT,
    CourseName VARCHAR(100)
);
CREATE TABLE
employeedb=# INSERT INTO Student (StudentID, CourseName) VALUES
(1, 'Math'),
(1, 'Science'),
(2, 'History'),
(2, 'Math');
INSERT 0 4
employeedb=# CREATE TABLE Course (
    CourseID INT PRIMARY KEY,
    CourseName VARCHAR(100)
);

CREATE TABLE Enrollment (
    EnrollmentID INT PRIMARY KEY,
    StudentID INT,
    CourseID INT
);
CREATE TABLE
CREATE TABLE
employeedb=# INSERT INTO Course (CourseID, CourseName) VALUES
(101, 'Math'),
(102, 'Science'),
(103, 'History');

INSERT INTO Enrollment (EnrollmentID, StudentID, CourseID) VALUES
(1, 1, 101),
(2, 1, 102),
(3, 2, 103),
(4, 2, 101);
INSERT 0 3
INSERT 0 4
employeedb=# CREATE TABLE Students (
    StudentID INT PRIMARY KEY,
    Name VARCHAR(100)
);

CREATE TABLE Courses (
    CourseID INT PRIMARY KEY,
    CourseName VARCHAR(100)
);

CREATE TABLE Enrollments (
    EnrollmentID INT PRIMARY KEY,
    StudentID INT,
    CourseID INT,
    FOREIGN KEY (StudentID) REFERENCES Students(StudentID),
    FOREIGN KEY (CourseID) REFERENCES Courses(CourseID)
);
CREATE TABLE
CREATE TABLE
CREATE TABLE
employeedb=# ▯
```

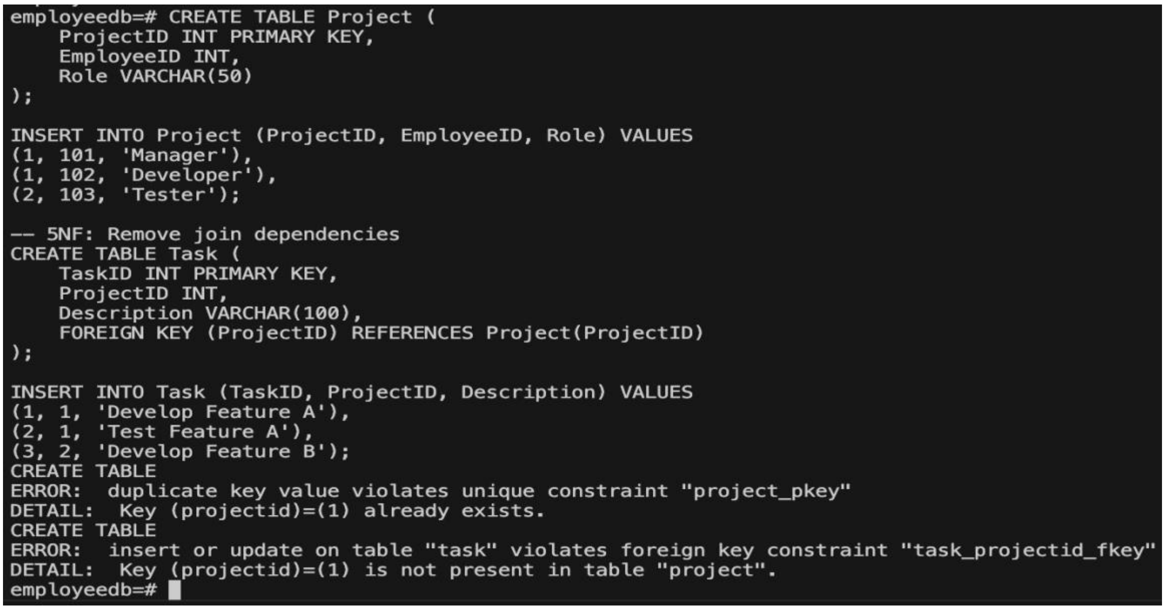
Practice 12: Implement Fourth and Fifth Form of Normalization Techniques

- **Objective:** Normalize a database to 4NF and 5NF.

- **Source Code:**

```
-- 4NF: Remove multi-valued dependencies
CREATE TABLE Project (
    ProjectID INT PRIMARY KEY,
    EmployeeID INT,
    Role VARCHAR(50)
);

INSERT INTO Project (ProjectID, EmployeeID, Role) VALUES
(1, 101, 'Manager'), (1, 102, 'Developer'), (2, 103, 'Tester');

-- 5NF: Remove join dependencies
CREATE TABLE Task (
    TaskID INT PRIMARY KEY,
    ProjectID INT,
    Description VARCHAR(100),
    FOREIGN KEY (ProjectID) REFERENCES Project(ProjectID)
);

INSERT INTO Task (TaskID, ProjectID, Description) VALUES
(1, 1, 'Develop Feature A'),
(2, 1, 'Test Feature A'), (3,
2, 'Develop Feature B');
```

- **Output:**



```
employeedb=# CREATE TABLE Project (
    ProjectID INT PRIMARY KEY,
    EmployeeID INT,
    Role VARCHAR(50)
);

INSERT INTO Project (ProjectID, EmployeeID, Role) VALUES
(1, 101, 'Manager'),
(1, 102, 'Developer'),
(2, 103, 'Tester');

-- 5NF: Remove join dependencies
CREATE TABLE Task (
    TaskID INT PRIMARY KEY,
    ProjectID INT,
    Description VARCHAR(100),
    FOREIGN KEY (ProjectID) REFERENCES Project(ProjectID)
);

INSERT INTO Task (TaskID, ProjectID, Description) VALUES
(1, 1, 'Develop Feature A'),
(2, 1, 'Test Feature A'),
(3, 2, 'Develop Feature B');
CREATE TABLE
ERROR:  duplicate key value violates unique constraint "project_pkey"
DETAIL:  Key (projectid)=(1) already exists.
CREATE TABLE
ERROR:  insert or update on table "task" violates foreign key constraint "task_projectid_fkey"
DETAIL:  Key (projectid)=(1) is not present in table "project".
employeedb=# 
```

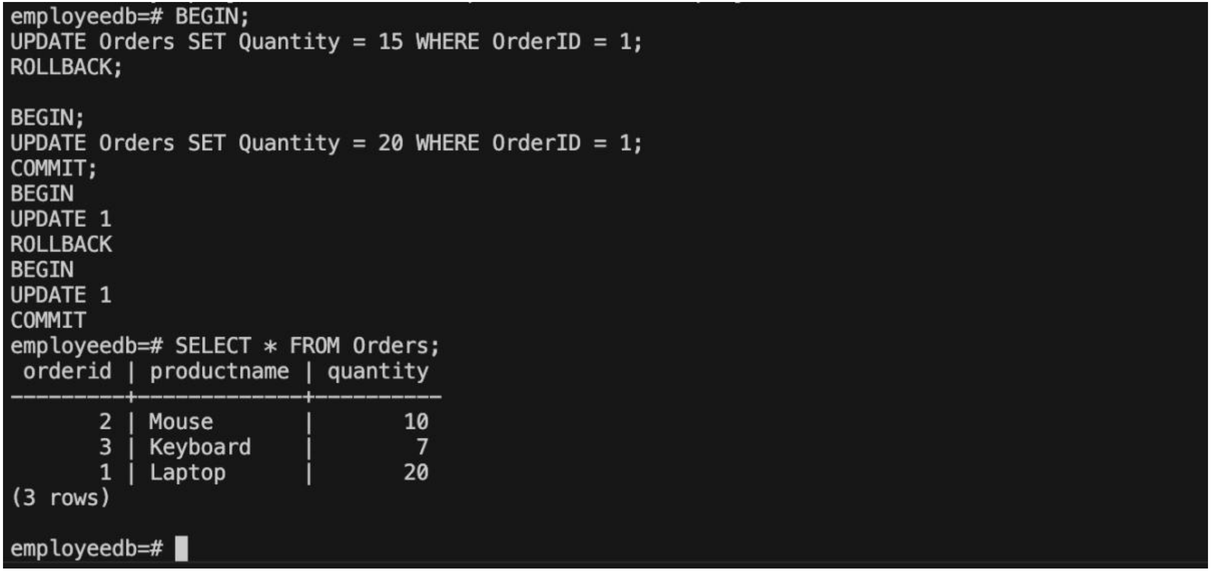
Practice 13: Implement Functions/Procedures to Begin, Commit, and Rollback Transactions

- **Objective:** Write functions/procedures for transaction management.

- **Source Code:**

```
 BEGIN;
UPDATE Orders SET Quantity = 15 WHERE OrderID = 1;
ROLLBACK;

BEGIN;
UPDATE Orders SET Quantity = 20 WHERE OrderID = 1;
COMMIT;
```

- **Output:**



```
employeedb=# BEGIN;
UPDATE Orders SET Quantity = 15 WHERE OrderID = 1;
ROLLBACK;

BEGIN;
UPDATE Orders SET Quantity = 20 WHERE OrderID = 1;
COMMIT;
BEGIN
UPDATE 1
ROLLBACK
BEGIN
UPDATE 1
COMMIT
employeedb=# SELECT * FROM Orders;
 orderid | productname | quantity
---------+-------------+----------
       2 | Mouse       |       10
       3 | Keyboard    |        7
       1 | Laptop      |       20
(3 rows)

employeedb=# █
```

Practice 14: Analyze the Structure and Properties of B-tree Index and Its Variants

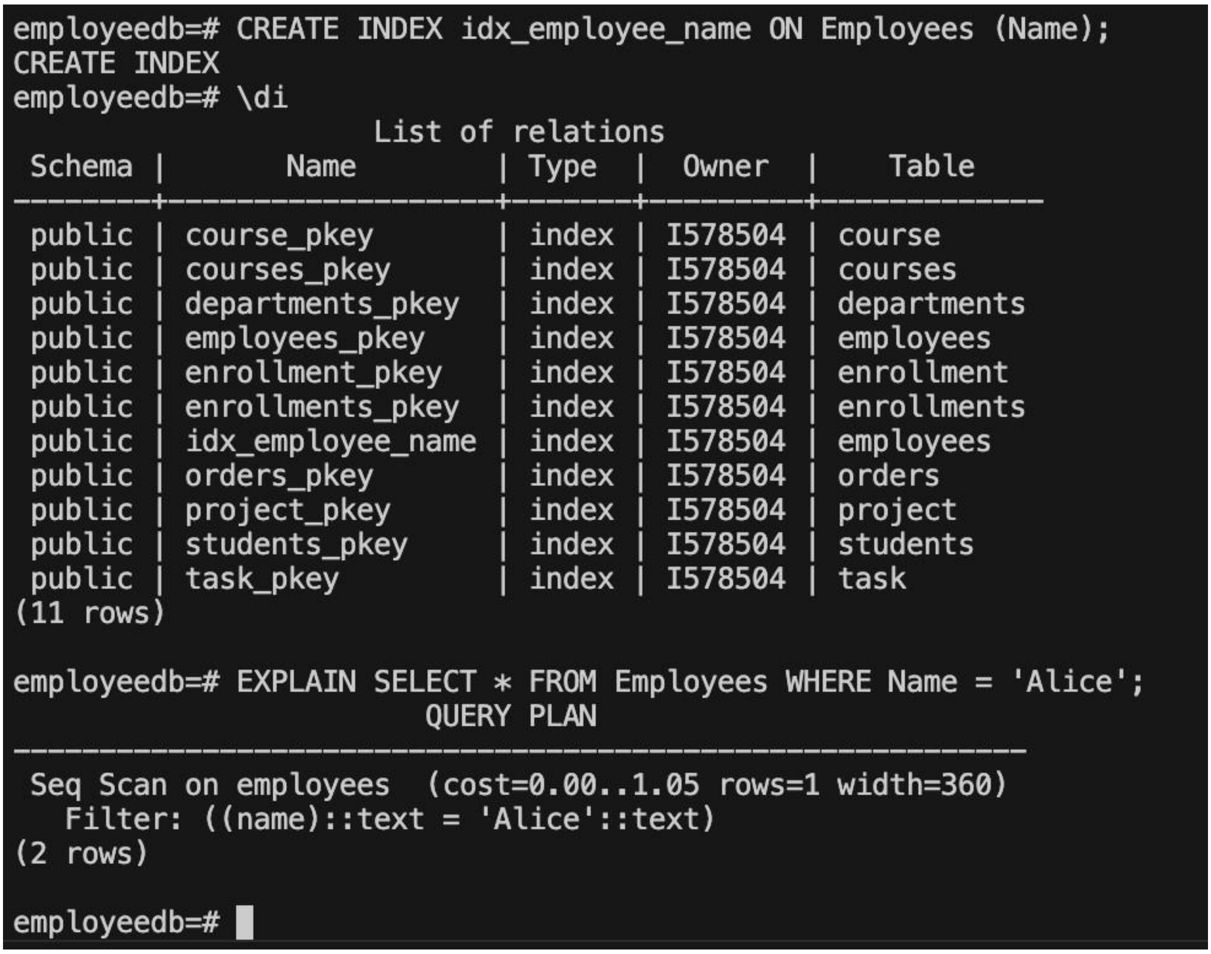
- **Objective:** Write functions/procedures for transaction management.
- **Source Code:**

  CREATE INDEX idx_name ON Employees (Name);

  \di

  EXPLAIN SELECT * FROM Employees WHERE Name = 'Alice';

- **Output:**



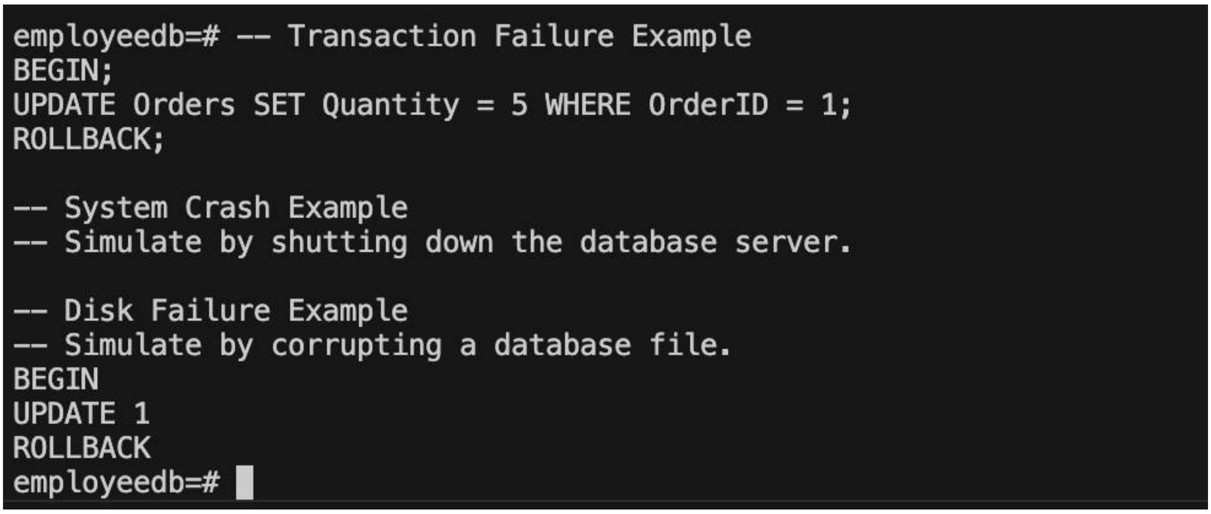
```
employeedb=# CREATE INDEX idx_employee_name ON Employees (Name);
CREATE INDEX
employeedb=# \di
                      List of relations
 Schema |       Name        | Type  |  Owner   |    Table
--------+-------------------+-------+----------+-------------
 public | course_pkey       | index | I578504 | course
 public | courses_pkey      | index | I578504 | courses
 public | departments_pkey  | index | I578504 | departments
 public | employees_pkey    | index | I578504 | employees
 public | enrollment_pkey   | index | I578504 | enrollment
 public | enrollments_pkey  | index | I578504 | enrollments
 public | idx_employee_name | index | I578504 | employees
 public | orders_pkey       | index | I578504 | orders
 public | project_pkey      | index | I578504 | project
 public | students_pkey     | index | I578504 | students
 public | task_pkey         | index | I578504 | task
(11 rows)

employeedb=# EXPLAIN SELECT * FROM Employees WHERE Name = 'Alice';
                         QUERY PLAN
-----------------------------------------------------------
 Seq Scan on employees  (cost=0.00..1.05 rows=1 width=360)
   Filter: ((name)::text = 'Alice'::text)
(2 rows)

employeedb=# 
```

Practice 15: Case Study: Analyze Different Types of Failures

- **Objective:** Analyze transaction failures, system crashes, and disk failures.
- **Source Code:**

```
-- Transaction Failure Example
BEGIN;
UPDATE Orders SET Quantity = 5 WHERE OrderID = 1;
ROLLBACK;

-- System Crash Example
-- Simulate by shutting down the database server.

-- Disk Failure Example
-- Simulate by corrupting a database file.
```

- **Output:**



```
employeedb=# -- Transaction Failure Example
BEGIN;
UPDATE Orders SET Quantity = 5 WHERE OrderID = 1;
ROLLBACK;

-- System Crash Example
-- Simulate by shutting down the database server.

-- Disk Failure Example
-- Simulate by corrupting a database file.
BEGIN
UPDATE 1
ROLLBACK
employeedb=# █
```