4) Create a Table with Constraints a) Create an EmployeeDetails table with EmployeeID as the PRIMARY KEY and DepartmentID as a FOREIGN KEY referencing a Department table. b) Insert a valid employee record with an existing DepartmentID, then attempt to insert an employee with a non-existent DepartmentID and observe the constraint violation. c) Insert an employee with a duplicate EmployeeID and check how the primary key constraint prevents duplicate entries. d) Modify the Salary column to have a UNIQUE constraint and attempt to insert two employees with the same salary to test the constraint. e) Write a query to delete an employee from EmployeeDetails and ensure that the deletion does not violate any referential integrity constraints.

Which recent tool or technology have you studied for database management, and can you briefly explain its key features and why it is used in the industry?

CREATE TABLE Department (

DepartmentID INT PRIMARY KEY,

DepartmentName VARCHAR(50)

);

CREATE TABLE EmployeeDetails (

EmployeeID INT PRIMARY KEY,

EmployeeName VARCHAR(50),

Salary DECIMAL(10, 2),

DepartmentID INT,

FOREIGN KEY (DepartmentID) REFERENCES Department(DepartmentID)

);

INSERT INTO Department (DepartmentID, DepartmentName)

VALUES (1, 'Human Resources');

INSERT INTO EmployeeDetails (EmployeeID, EmployeeName, Salary, DepartmentID)

VALUES (1001, 'Amit Kumar', 50000.00, 1);

INSERT INTO EmployeeDetails (EmployeeID, EmployeeName, Salary, DepartmentID)

VALUES (1002, 'Rohit Singh', 55000.00, 99);

-- EmployeeID 1001 already exists

INSERT INTO EmployeeDetails (EmployeeID, EmployeeName, Salary, DepartmentID)

VALUES (1001, 'Sneha Sharma', 60000.00, 1);

ALTER TABLE EmployeeDetails

ADD CONSTRAINT unique\_salary UNIQUE (Salary);

INSERT INTO EmployeeDetails (EmployeeID, EmployeeName, Salary, DepartmentID)

VALUES (1003, 'Vikas Jain', 60000.00, 1);

INSERT INTO EmployeeDetails (EmployeeID, EmployeeName, Salary, DepartmentID)

VALUES (1004, 'Anita Mehra', 60000.00, 1);

DELETE FROM EmployeeDetails

WHERE EmployeeID = 1001;

**📚 Recently Studied Database Management Tool: CockroachDB**

✅ **Key Features:**

* **Distributed SQL Database:** Looks and behaves like a traditional SQL database but is massively scalable and resilient.
* **Automatic Replication and Failover:** Data is automatically replicated across multiple nodes for fault tolerance.
* **Strong Consistency:** Guarantees ACID transactions even across data centers.
* **Self-healing Infrastructure:** Automatically rebalances workloads and repairs failures without downtime.
* **PostgreSQL Compatibility:** Works with existing SQL tools and libraries.
* **Cloud Native:** Optimized for Kubernetes, hybrid, and multi-cloud deployments.

✅ **Why it’s used in Industry:**

* Perfect for **global applications** requiring **zero downtime** and **high consistency** (e.g., financial apps, real-time multiplayer games).
* Companies can easily scale horizontally without major redesigns.
* Offers a blend of **traditional RDBMS reliability** with **NoSQL-level scalability**.