Input:  
Input dataset:  
>>> Emplyee.cvs

| EmpID | EmpName | DeptID | DeptName | Salary |
| --- | --- | --- | --- | --- |
| 001 | Adam | 070 | HR | 50000 |
| 002 | Raph | 040 | Engineering | 70000 |
| 003 | Essy | 070 | HR | 55000 |
| 004 | Michael | 020 | Finance | 60000 |
| 005 | Sarah | 010 | Science | 72000 |

Input Functional Dependencies

>>> EmpID -> EmpName, DeptID, DeptName, Salary  
>>> DeptID-> DeptName

Choice of the highest normal form to reach (1: 1NF, 2: 2NF, 3: 3NF, B: BCNF, 4: 4NF, 5: 5NF): >>> 3NF

**Output**

**SQL Queries for Normalized Schema**

CREATE TABLE Employee (

EmpID VARCHAR(255),

EmpName VARCHAR(255),

DeptID VARCHAR(255),

Salary VARCHAR(255),

PRIMARY KEY (EmpID)

);

CREATE TABLE Department (

DeptID VARCHAR(255),

DeptName VARCHAR(255),

PRIMARY KEY (DeptID)

);

**1NF (First Normal Form):** The table satisfies 1NF as all attributes contain atomic values.

**2NF (Second Normal Form):** Partial dependencies are removed by decomposing the relation into two relations: Employee and Department.

**3NF (Third Normal Form):** The relation Employee is free of transitive dependencies.

Points to note: The SQL code creates separate tables for employees and departments, ensuring data integrity and reducing redundancy.

**generateSQL Function:**

* + The function generates SQL code for each relation in the relations vector.
  + It now accurately creates separate tables for Employee and Department, ensuring that DeptName is in a separate relation based on DeptID.