**Consider the following table:**

**TASK (EmployeeID, EmpLastName, EmpFirstName, Phone, OfficeNumber, ProjectName, Sponsor, WorkDate, HoursWorked)**

**With the following possible functional dependencies:**

**EmployeeID = (EmpLastName, EmpFirstName, Phone, OfficeNumber)**

**Project Name = Sponsor**

1. **Write SQL statements to display the values of any rows that violate these functional dependencies.**

**EmployeeID = (EmpLastName, EmpFirstName, Phone, OfficeNumber)**

SELECT \*

FROM TASK t1

JOIN TASK t2 ON t1.EmployeeID = t2.EmployeeID

WHERE t1.EmpLastName <> t2.EmpLastName

OR t1.EmpFirstName <> t2.EmpFirstName

OR t1.Phone <> t2.Phone

OR t1.OfficeNumber <> t2.OfficeNumber;

**Project Name = Sponsor**

SELECT \*

FROM TASK

WHERE ProjectName <> Sponsor;

1. **If no data violate these functional dependencies, can we assume these data are valid? Why or why not?**

If no data violates the given functional dependencies, it suggests that the data in the table is consistent with the defined dependencies. However, it does not guarantee that the data is entirely valid or error-free.

Functional dependencies define relationships between attributes in a table and specify how values in one set of attributes determine the values in another set. When there are no violations, it means that the specified dependencies hold for the existing data. However, it doesn't guarantee the absence of other data quality issues such as missing or incorrect values, duplicates, or semantic inconsistencies.

To ensure the overall validity and quality of the data, it is essential to consider other aspects such as domain constraints, referential integrity, data completeness, and accuracy checks. Thorough data validation processes should be applied to verify the integrity and correctness of the data beyond just checking functional dependencies.

1. **Assume that these functional dependencies are accurate and that operators have updated the data, as necessary, to reflect them. Write all SQL statements necessary to re-design this table into a set of tables in BCNF and 4NF. Assume that the table has data values that need transformation as appropriate to the new design.**

To redesign the Task table into a set of tables in Boyce-Codd Normal Form (BCNF) and Fourth Normal Form (4NF), we split the original table into smaller tables ensuring that each table satisfies the normalization criteria. The new tables are below:

1. Employees (EmployeeID, EmpLastName, EmpFirstName, Phone, OfficeNumber)
2. Projects (ProjectName, Sponsor)
3. Tasks (EmployeeID, ProjectName, WorkDate, HoursWorked)

To ensure referential integrity and maintain the relationships between these tables, we can define the following foreign keys in the Tasks table:

EmployeeID references Employees(EmployeeID)

ProjectName references Projects(ProjectName)

The SQL statements to create the above tables are:

CREATE TABLE Employees (

EmployeeID INT PRIMARY KEY,

EmpLastName VARCHAR(255),

EmpFirstName VARCHAR(255),

Phone VARCHAR(255),

OfficeNumber VARCHAR(255)

);

CREATE TABLE Projects (

ProjectName VARCHAR(255) PRIMARY KEY,

Sponsor VARCHAR(255)

);

CREATE TABLE Tasks (

EmployeeID INT,

ProjectName VARCHAR(255),

WorkDate DATE,

HoursWorked INT,

PRIMARY KEY (EmployeeID, ProjectName, WorkDate),

FOREIGN KEY (EmployeeID) REFERENCES Employees(EmployeeID),

FOREIGN KEY (ProjectName) REFERENCES Projects(ProjectName)

);