Project Overview

The primary goal of this project is to clean and normalize michigan hospital readmission data, transforming it into a format that adheres to **Normal Form (NF)**. I have used MySQL and Python as a tool to create this database. The normalization process ensures that the data structure minimizes redundancy and dependency, making the database more efficient for querying and updates. The transformation focuses on separating data into logical entities and breaking down multi-valued dependencies that could lead to anomalies.

In this project, I utilized the **PACE** framework—**Prepare, Analyze, Communicate, and Execute**—to normalize hospital data and ensure its integrity, scalability, and efficiency for future analysis. Below is an explanation of how I applied the PACE framework to this project.

- **Prepare**: I set up the database and filtered the data to focus on relevant hospitals and readmission rates, while cleaning the data and addressing null values.
- **Analyze**: I identified the need for normalization due to data redundancy and designed three normalized tables to store the hospital, condition, and readmission rate information.
- **Communicate**: I want to communicate with the viewers about the normalization process and the benefits of creating separate tables with foreign key relationships to improve data integrity and scalability.
- **Execute**: I created the normalized tables, inserted the cleaned data, handled special cases, and enforced foreign key constraints to ensure accurate relationships between the tables.

Steps and Explanation

1). Database Creation and Initial Data Handling:

I began by creating a new database AnalyticsProject and using it for the entire process. Initially, the raw data hospital_visits table was imported into MySQL AnalyticsProject database through python using sqlalchemy, mysql connector and all other processes were completed in MySQL itself. Once hospital_visits was imported it was copied into a new table Hospital_Visits_CLean to preserve the original dataset. This step ensures that the original data remains intact and untouched during the normalization process.

2).Filtering the Data for Specific Hospitals:

The focus was hospitals in Michigan, filtering for a specific list of Facility IDs (extracted from information through google) and focusing only on the '30-Day Readmission' measure. This step is essential as we need to work with a smaller, relevant subset of data for the subsequent normalization process.

3).Creating Tables for 4th Normal Form (4NF):
At this point, I proceed to normalize the data into **4th Normal Form**. I splitted the data into three distinct entities:

- **Hospitals Table**: This table stores the hospital-specific information, such as the Facility ID, name, address, city, state, zip code, county, and phone number. By separating this data, I eliminated redundant information and ensured that hospital details are stored only once.

- **Conditions Table**: This table stores information about the conditions (e.g., AMI, HF, PN), identified by Measure Id. We store the condition's Measure Name for reference. This separates the condition-related data, removing it from the Readmission Rates table and reducing redundancy.

- **Readmission Rates Table**: This table links hospitals to specific conditions and includes the readmission status, rates, and estimates. It also tracks the start and end dates for the readmission data. It contains foreign keys that reference the Michigan_Hospitals and Conditions tables. This table holds the actual data about readmission rates, and by linking it to the other two tables, we achieve normalization.

4).Populating the Tables with Data:
Now that the tables are created, we insert the filtered data into the respective tables. This step involves inserting:

- **Hospital Information**: Unique hospital data (name, address, city, etc.) is inserted into the Michigan_Hospitals table.

- **Condition Information**: Unique condition data is inserted into the Conditions table.
-
- **Readmission Data**: For each hospital and condition, the readmission data (rates, estimates, etc.) is inserted into the Readmission_Rates table.

5).Achieving Normal Form (NF):
The data has been broken down into three distinct tables, each focusing on a single entity (hospitals, conditions, and readmission rates).

Multi-valued dependencies have been eliminated. Previously, hospital information was repeated for each condition, but now hospital information is stored only once in the Michigan_Hospitals table. Each condition is stored only once in the Conditions table, and the Readmission_Rates table captures the relationship between hospitals and conditions, without introducing unnecessary redundancy.

6). Conclusion:

The database design ensures that updates to any entity (e.g., updating a hospital's address or adding a new condition) do not require multiple entries in multiple places, preventing anomalies like insertion, deletion, and update anomalies.

By following this structure, I have achieved a highly efficient and well-organized database design that maintains data integrity and minimizes redundancy. This design ensures that future queries, updates, and reports can be executed more effectively while reducing the risk of data anomalies.

There are approaches through which the Readmission_Rates table can be further normalized but looking at the need of the analysis, I believe it would satisfy the condition for now.