**Programming Project 01 -Documentation**

**5300 – Database Systems**

**Group -   
Prasanna Kancharla  
D Shilpa Reddy  
Rajlakshmi Maurya  
Monica Gungi  
Shreyas Shekar Koushik**

1.**Objective**

The project's primary goal is to develop a robust program that accepts a dataset and its associated functional dependencies. The core function of the program is to methodically normalize the input relations in accordance with the given functional dependencies, enhancing data integrity and minimizing redundancy. As a part of the normalization process, the program is engineered to automatically generate SQL queries that facilitate the creation of structured, optimized database tables.

2.**Introduction**

The Database Normalization Code is designed to assist users in organizing and optimizing their database structures. It takes a dataset (relation) and a set of functional dependencies as input, offering database normalization and SQL query generation based on the provided dependencies.

3. **System Overview**

The Database Normalization Code consists of four core components: the Input Parser, Normalizer, SQL Query Generator, and Normal Form Finder. These components collaborate to transform unnormalized data into a structured and efficient database.

**4. Developer Guide**

**4.1. Code Structure**

The code focuses on parsing the input dataset (relation) and functional dependencies responsible for the normalization of the dataset up to the desired normal form and generating the corresponding SQL queries. Below is the decoded clear explanation of the code where it is explained in 2 halves.

**Part 1: Input Parsing and Functional Dependencies Identification**

* **Import Statements and Class Definitions**

The code begins with Python import statements that import necessary libraries such as csv for reading CSV files and sys for accessing system-specific parameters and functions.

Classes are defined to represent Attributes and Functional Dependencies (FDs). The Attribute class encapsulates the properties of a database attribute, while the FunctionalDependency class represents an FD, holding both the determinant and the dependent as sets of Attribute objects.

* **Reading the Input**

The csv module is utilized to read input CSV files, which contain the dataset to be normalized. The data is read row-by-row and stored for further processing.

Functional dependencies are read from a text file, where each line represents a single FD. The FDs are parsed and stored as instances of the FunctionalDependency class.

* **Preprocessing and Validation**

After the initial parsing, the code performs preprocessing steps to ensure data consistency and correctness. This includes validating the format of the input and handling potential errors in the dataset or functional dependencies.

**Part 2: Normalization and SQL Query Generation**

* **Normalization Functions**

For each normal form, specific functions are implemented to perform the necessary transformations. The code abstracts the normalization rules into these functions, which progressively decompose the relation into tables that satisfy the conditions of 2NF, 3NF, and BCNF. Each function accounts for the removal of partial, transitive, and other types of dependencies that violate the rules of the respective normal form.

* **SQL Query Generation**

After normalization, the code generates SQL **CREATE TABLE** statements corresponding to each table in the normalized form. This includes the definition of primary keys and data types inferred from the attributes. The generation of SQL is dynamic, and it ensures that the output can be directly executed in a relational database management system (RDBMS).

**4.2. Functionality**

**This project relies on the following Python libraries:**

ordered\_set: Utilized for maintaining ordered sets.

collections: Employed for working with ordered dictionaries.

CSV: Used for parsing CSV files.

CSV Parsing: The script parses the input dataset stored in a CSV file and stores it in a Python dictionary.

Functional Dependency Parsing: It also parses the functional dependencies from a text file. Functional dependencies are provided in the form of strings (e.g., "A, B -> C"), where A and B together determine C.

MVD Parsing : For handling 4NF and 5NF, the script can parse Multivalued Dependencies (MVDs) from a text file. MVDs are in the format "A, B ->> C," indicating that A and B together determine multiple values of C.

Normalization: The script offers functionality to normalize the dataset up to the Boyce-Codd Normal Form (BCNF). It decomposes tables based on functional dependencies and provides the option to work with composite keys.

SQL Query Generation: The script generates SQL queries for the decomposed tables based on the chosen normal form and stores them in a query.txt file. These queries are designed to recreate the decomposed tables with appropriate primary keys and data types.

**4.3 Usage**

**Flow and Logic of the Code**

The logic of the code is structured to follow the logical flow of database normalization:

* Input is read and validated: CSV and text files are parsed to create a list of attributes and functional dependencies.

Input Files: Prepare the following input files for your database:

input\_dataset.csv: Contains the dataset in CSV format.

functional\_dependencies.txt: Specifies the functional dependencies for the database.

mvd.txt: Contains Multivalued Dependencies (MVDs).

* Command Line Execution: Run the script from the command line, providing the input file paths as command-line arguments. **For example: python your\_script.py input\_dataset.csv functional\_dependencies.txt**
* Normalization is performed: Based on the user's choice, the data is normalized to the specified normal form using the dedicated functions for each level of normalization.
* SQL queries are generated: For each of the resulting tables after normalization, SQL statements are constructed, reflecting the new structure of the database.
* Output is provided: The final SQL statements are derived in an output.txt file. User Interaction where the script will prompt you to choose the highest normal form you wish to achieve (1NF, 2NF, 3NF, BCNF, 4NF, or 5NF). If you select 4NF or 5NF, you will need to provide the mvd.txt file.

**5. Output and Results**

**5.1. Normalization**

A computer screen shot of a black screen

Description automatically generated

A computer screen shot of a program

Description automatically generated

A computer screen shot of a computer program

Description automatically generated

**5.2. Generated Sql Queries**

A screenshot of a computer

Description automatically generated

**6. Conclusion**

The script will generate SQL queries for the decomposed tables and save them in a file named query.txt. The parsed data will be saved in output.txt.