Team:
1. Mubeena Mohammed (12600222)
2. Sai Tejaswar Ponugoti (12601283)

Sample Input:

CoffeeShopData:

OrderID
Date
PromocodeUsed
TotalCost
TotalDrinkCost
TotalFoodCost
CustomerID
CustomerName
DrinkID
DrinkName
DrinkSize
DrinkQuantity
Milk
DrinkIngredient
DrinkAllergen
FoodID
FoodName
FoodQuantity
FoodIngredient
FoodAllergen

Functional Dependencies: The functional dependencies used for the dataset are:

1. OrderID --> {Date, TotalCost, TotalDrinkCost, TotalFoodCost, CustomerID, CustomerName}
2. {OrderID, DrinkID} --> {DrinkSize, DrinkQuantity, Milk}
3. {OrderID, FoodID} --> {FoodQuantity}
4. CustomerID --> CustomerName
5. DrinkID --> DrinkName
6. FoodID --> FoodName

Multi-valued dependencies: The multi-valued dependencies used for the dataset are:

1. OrderID -->> DrinkID
2. OrderID -->> FoodID

The functional dependencies and multi-valued dependencies are taken as .txt file(input.txt). User Input:

- Choice of the highest normal form to read(1NF,2NF,3NF,BCNF,4NF,5NF).
  Primary key used {OrderID, DrinkID, FoodID}

Core Components:

- Input Parser: The parser function is used to read the csv file and the txt file which is used to store the functional dependencies. It breaks down the text into recognized strings of characters for further analysis.
- Normalizer: The Normalizer decomposes the input dataset into the required normal form based on the given functional dependencies. The normalization methods are used to decompose the given dataset into the user-required normal form.
- SQL Query Generator: It refers to a tool or functionality that helps to generate the SQL queries.

Deliverables:
Source Code: Here, we have used the Python programming language to normalize the given dataset. Code
Description:

- RDBMSNormalizer Class Documentation is as below,
- The Python libraries are imported, to perform the required operations
- The RDBMSNormalizer class is designed to automate the process of normalizing relational database tables from an input CSV file, based on specified functional dependencies (FDs).
- It normalizes data up to various normal forms (1NF, 2NF, 3NF, BCNF, and 4NF) and generates SQL statements to create the resulting normalized tables.
- Key Methods
- _init__(self, csv_file, fd_file)
- Initializes the normalizer with an input CSV file and a file containing functional dependencies.
- Loads the data and FDs and sets up structures for tracking dependencies and normalized tables.
- load_data(self)
- Loads table data from a CSV file and reads functional dependencies from a text file. It populates `table_data` and `fd_set` with the initial data and dependency structures.
- _determine_highest_normal_form(self)
- Evaluates the current highest normal form (1NF to BCNF) of each table based on its functional dependencies. This helps identify the normalization level required.
- Normalization Functions
- normalize_1nf(self): Ensures tables adhere to First Normal Form by removing repeating groups and ensuring atomic values.
- normalize_2nf(self): Enforces 2NF by eliminating partial dependencies based on the primary key.
- normalize_3nf(self): Ensures 3NF by removing transitive dependencies.
- normalize_bcnf(self): Enforces Boyce-Codd Normal Form (BCNF) by decomposing tables to remove any functional dependencies where the determinant is not a superkey.
- normalize_to_4nf(self): Normalizes tables to Fourth Normal Form (4NF) by identifying and eliminating multivalued dependencies (MVDs).
- Dependency Analysis Helper Functions
- is_superkey(self, determinant_attrs, table_data):
- Checks if given attributes can uniquely identify rows in a table.
- is_partial_dependency(self, determinant_attrs, primary_key): Determines if a functional dependency is partial (relevant for 2NF).
- is_transitive_dependency(self, determinant_attrs, dependent_attrs, primary_key): Identifies transitive dependencies, helping enforce 3NF.
- parse_dependencies(self, dependencies): Separates functional dependencies and multivalued dependencies based on provided syntax.
- extract_dependencies(self, table_name, table_data, primary_key, functional_dependencies): Validates and extracts dependencies that are relevant to each table.
- generate_sql(self)

- Generates SQL `CREATE TABLE` statements for the normalized tables. Automatically infers data types based on table data and saves the statements to an output file.
- user_input(self)
- Provides an interactive interface for users to determine the highest normal form of a table, select a desired normalization level, and initiate the normalization process.

Execution Flow

- Initialization and Loading: The class loads data and functional dependencies. From the inputTable.csv and fd.txt file respectively.
- User Interaction: Users can determine the highest normal form and select a target normalization level.
- Normalization Process: Based on the selected level, the normalizer progressively enforces higher normal forms (1NF through 5NF).
- SQL Generation: Generates SQL statements to create the normalized tables in a database.
- This overview provides an organized and modular explanation of each component, making it easier to understand and use `RDBMSNormalizer` for database normalization tasks.