**CS5300 PROGRAMMING PROJECT DATABASE NORMALIZATION**

**TEAM:** Bhavya Shree Kakunuri, Sai Teja Putta

**Sample Input:**

**Dataset:** TestingData (1NF-5NF).CSV, he file "TestingData (1NF-5NF).CSV" is designed to test the normalization process of a database schema through different normal forms.

**Functional Dependencies:** The dataset's functional dependencies are as follows:

OrderID --> Date, TotalCost, TotalDrinkCost, TotalFoodCost, CustomerID, CustomerName

OrderID, DrinkID --> DrinkSize, DrinkQuantity, Milk

OrderID, FoodID --> FoodQuantity

CustomerID --> CustomerName

DrinkID --> DrinkName

FoodID --> FoodName

**Multi-valued dependencies:**

OrderID ->> DrinkID

OrderID ->> FoodID

Functional dependencies and multi-valued dependencies are taken as .txt file (input.txt).

User Input:

1. Select the highest normal form to normalize from the below list 1 for 1NF ,2NF,3NF, BCNF ,4NF ,5NF.
2. Primary key used (OrderID,DrinkID,FoodID)

* Primary keys are taken as user input, allowing flexibility for composite keys. This input is crucial, as primary keys define the relationships and dependencies necessary for each normalization step.

**JSON Structure Breakdown**

* relations:
* table\_name: The name of the relation/table.
* attributes: List of column names (attributes) in the table.
* rows: Nested list where each sublist represents a row in the table.
* primary\_key: List specifying the primary key(s) for the table.
* candidate\_keys: Optionally includes any identified candidate keys.
* prime\_attributes: Attributes that are part of any candidate key.
* non\_atomic: Attributes that contain non-atomic (e.g., multivalued) entries, which need to be decomposed in 1NF.
* functional\_dependencies:
* List of objects defining functional dependencies.
* lhs: Left-hand side attributes of the dependency (determinants).
* rhs: Right-hand side attributes (dependents).
* mvds:
* List of objects defining multivalued dependencies.
* lhs: Left-hand side attributes of the dependency.
* rhs: Right-hand side attributes that are multivalued in relation to the left-hand side.
* target\_normal\_form:
* Specifies the highest normal form to achieve during normalization.Purpose of the JSON Format

The JSON input format consolidates all required data for normalization in a single file, making it convenient for the program to parse and process. It avoids multiple file dependencies by embedding all information in a structured way, making it easier to adapt and modify inputs for testing various normal forms.

**Core Components:**

1. **Input Parser:** The parse\_input() function reads the JSON file and extracts each component for processing. The data is structured into relations, functional dependencies, multivalued dependencies, and the target normal form.
2. **Normalizer:** The program breaks down the input dataset into the required normal form using functional dependencies, and the provided dataset into the user-required normal form using normalizing techniques. It processes each relation using extracted dependencies and target normal form from JSON, applying normalization steps (1NF through 5NF) depending on the target.
3. **SQL Query Generator:** It refers to a program or feature that helps in creating SQL queries.

**Deliverables:**

**Source Code:** we used Python programming language to normalize the given dataset.

**Code Description:**

**Main Function**  
The main () function is the driver for the program. It takes as input from the user: the input and output file, the normalization level-a choice between 1NF to 5NF-and the target format, a choice between SQL output and textual schema output.

It then processes the command line arguments to:

1. Check Normal Form: The normal form specified is checked to be within the range of those supported, ranging from 1NF to 5NF.

2. Input Processing: This calls load\_data to read in data from the nominated file and then calls load\_fds\_from\_file and load\_mvds\_from\_file, providing parsing of functional and multivalued dependencies, respectively.

3. Primary Key Input: It captures and processes primary key input for later steps in normalization.

4. Normalization Execution: It consecutively calls each normalization function from to\_1nf through to\_5nf depending on the highest normal form specified.

Each successive step in normalization decomposes relations according to its rules and returns a progressively normalized schema.

5. Generation of Output: Produces either SQL statements or a schema representation, based on the target format selected.

**Normalization Functions**

1. to\_1nf: This removes multivalued attributes by splitting them into other relations containing the primary key attributes of the old relation. Then, each tuple from the old relation is expanded for every single multivalued entry, which ensures that there are no more multivalued attributes left.

2. to\_2nf: It recognizes partial dependencies and decomposes them, moving dependent attributes into new relations. This will ensure that dependency will be only on a full primary key, not on subsets of it, which creates 2NF.

3. to\_3nf: It recognizes transitive dependencies and then decomposes them. This is performed by the creation of new relations such that in every functional dependency in a relation, there is a superkey or each dependent attribute is prime.

4. to\_bcnf: Further refines the schema by eliminating any remaining nontrivial functional dependencies by decomposing relations such that all dependencies have a superkey on the left side as dictated by BCNF.

5. to\_4nf: Eliminates multi-valued dependencies by decomposing relations containing multi-valued dependencies into separate relations.

6. to\_5nf: Resolves join dependencies by decomposing relations into optimally structured relations that can be joined through natural joins.

This will be achieved through the checking of join dependency and trying the decomposition until an optimal structure is reached.

Supporting Functions

* load\_data, load\_fds\_from\_file, and load\_mvds\_from\_file: Load the main dataset, functional dependencies, and multivalued dependencies from specified files.
* detect\_multivalued\_columns: Scans for and identifies multivalued attributes within the dataset to prepare them for normalization to 1NF.
* is\_superkey (inside to\_bcnf): Checks whether a given set of attributes forms a super key in regard to the primary keys.
* check\_join\_dependency (inside to\_5nf): Checks if a relation has join dependency; if so, then decompose into multiple tables for 5NF.

**Schema Generation**  
The final schema output can be generated in SQL or textual format, detailing each normalized relation and its attributes, primary keys, and dependencies as specified by the target normal form.

* This project provides an automated solution for database normalization up to 5NF, using a flexible JSON input format. By taking a single JSON file containing the database schema, functional dependencies, multivalued dependencies, and target normal form, the program simplifies and optimizes the normalization process, producing an organized and redundancy-free database schema.