Database Project Documentation:

Group Members: ROHINI KUKKA - 12618604
PAVAN SAI TIRUMALASETTI - 12616379

Here the project is designed for database normalization, which mainly aims to transform a relational database to a normalized form.

Input:-

1. Database Table (.csv) file path
2. Primary Key
3. Candidate Key
4. Functional Dependencies - (Type ‘Done’ if everything is entered to exit the condition)
5. Multi Valued Dependencies - (Type ‘Done’ if everything is entered to exit the condition)
6. Choice of the highest normal form to reach (1: 1NF, 2: 2NF, 3: 3NF, B: BCNF, 4: 4NF, 5: 5NF).

Output:-

1. If the given input table is not in 1NF then the normalized 1NF would be written to Output.txt file
2. Rest all Normalized forms (2NF,3NF,BCNF,4NF) would be printed in the terminal itself.
3. The output will be the sql queries for the tables decomposed to highest normal form taken as input
4. Output.txt file containing the details of Highest normal form of the given table if the user asks for it.
5. And also whenever we run the program,first 5NF will be printed then its asks for other inputs.

Here is_integer, is_alphanumeric, is_data functions are used to identify the datatype and return. Next check datatype functions checks the datatype and assigns whether it is INT,VARCHAR or DATE.

1NF Function:

`convert_to_1NF(csv_filePath)`:

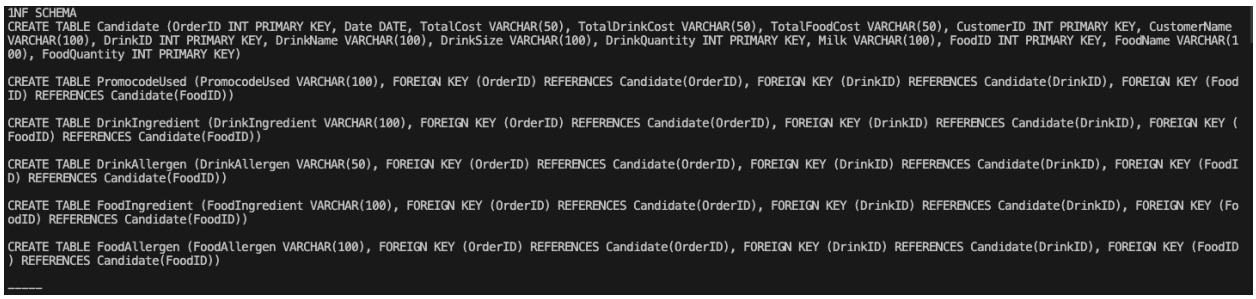
Input: `csv_filePath` - the path to a CSV file.
Output: A dictionary of relations with atomic columns. An output file is also printed with the following data in it after normalizing

Steps:

1. Reads a CSV file.
2. Identifies non-atomic columns with multiple values (e.g., lists or sets).
3. Splits non-atomic columns to ensure all values are atomic.

Usage: Ensures 1NF compliance by "exploding" non-atomic columns into individual rows. Non-atomic attributes are split into multiple records, and results are stored in newRelations.

Sample Output:



2NF Function:

`convert_to_2NF(tables, FD, primaryKey)`:

- **Input**:
  - `tables`: Initial set of tables.
  - `FD`: Functional dependencies as a list of strings in "LHS --> RHS" format.
  - `primaryKey`: Primary key of the table.
- **Output**: A dictionary of tables modified for 2NF compliance.

  Steps:

  1. Iterates over functional dependencies.
  2. Checks if the left-hand side of a dependency is a subset of the primary key but not equal to it, indicating a partial dependency.
  3. Creates new tables to store attributes affected by partial dependencies.

- **Usage**: Detects and resolves partial dependencies by creating new tables for each partial dependency. Uses the LHS of each FD to confirm if it's a subset of the primary key, creating new tables if necessary.

Sample Output:

```
2NF SCHEMA
CREATE TABLE 1 (OrderID INT PRIMARY KEY, Date DATE, TotalCost VARCHAR(50), TotalDrinkCost VARCHAR(50), TotalFoodCost VARCHAR(50), CustomerID INT PRIMARY KEY, CustomerName VARCHAR(
100))

CREATE TABLE 2 (DrinkID INT PRIMARY KEY, DrinkSize VARCHAR(100), DrinkQuantity INT PRIMARY KEY, Milk VARCHAR(100), FOREIGN KEY (OrderID) REFERENCES 1(OrderID))

CREATE TABLE 3 (FoodID INT PRIMARY KEY, FoodQuantity INT PRIMARY KEY, FOREIGN KEY (OrderID) REFERENCES 1(OrderID))

CREATE TABLE 4 (DrinkName VARCHAR(100), FOREIGN KEY (DrinkID) REFERENCES 2(DrinkID))

CREATE TABLE 5 (FoodName VARCHAR(100), FOREIGN KEY (FoodID) REFERENCES 3(FoodID))

CREATE TABLE 6 (, FOREIGN KEY (OrderID) REFERENCES 1(OrderID), FOREIGN KEY (DrinkID) REFERENCES 2(DrinkID), FOREIGN KEY (FoodID) REFERENCES 3(FoodID))

CREATE TABLE 7 (PromocodeUsed VARCHAR(100), FOREIGN KEY (OrderID) REFERENCES 1(OrderID), FOREIGN KEY (DrinkID) REFERENCES 2(DrinkID), FOREIGN KEY (FoodID) REFERENCES 3(FoodID))

CREATE TABLE 8 (DrinkIngredient VARCHAR(100), FOREIGN KEY (OrderID) REFERENCES 1(OrderID), FOREIGN KEY (DrinkID) REFERENCES 2(DrinkID), FOREIGN KEY (FoodID) REFERENCES 3(FoodID))

CREATE TABLE 9 (DrinkAllergen VARCHAR(50), FOREIGN KEY (OrderID) REFERENCES 1(OrderID), FOREIGN KEY (DrinkID) REFERENCES 2(DrinkID), FOREIGN KEY (FoodID) REFERENCES 3(FoodID))

CREATE TABLE 10 (FoodIngredient VARCHAR(100), FOREIGN KEY (OrderID) REFERENCES 1(OrderID), FOREIGN KEY (DrinkID) REFERENCES 2(DrinkID), FOREIGN KEY (FoodID) REFERENCES 3(FoodID))

CREATE TABLE 11 (FoodAllergen VARCHAR(100), FOREIGN KEY (OrderID) REFERENCES 1(OrderID), FOREIGN KEY (DrinkID) REFERENCES 2(DrinkID), FOREIGN KEY (FoodID) REFERENCES 3(FoodID))

------
```

3NF Function:

`convert_to_3NF(tables, FD, primaryKey)`:

- **Input:**
  - `tables`: The set of tables after 2NF.
  - `FD`: Functional dependencies as a list.
  - `primaryKey`: Primary key of the relation.
- **Output:** A dictionary of tables adjusted to 3NF.

Steps:

1. Checks each functional dependency for transitive dependencies.
2. Breaks transitive dependencies by creating new tables

- **Usage:** Resolves transitive dependencies by checking each FD. If the LHS is not a superkey, the function decomposes the relation further.

Sample Output:

```
3NF SCHEMA
CREATE TABLE 1 (CustomerID INT PRIMARY KEY, CustomerName VARCHAR(100))

CREATE TABLE 2 (OrderID INT PRIMARY KEY, Date DATE, TotalCost VARCHAR(50), TotalDrinkCost VARCHAR(50), TotalFoodCost VARCHAR(50), FOREIGN KEY (CustomerID) REFERENCES 1(CustomerID)
)

CREATE TABLE 3 (DrinkID INT PRIMARY KEY, DrinkSize VARCHAR(100), DrinkQuantity INT PRIMARY KEY, Milk VARCHAR(100), FOREIGN KEY (OrderID) REFERENCES 2(OrderID))

CREATE TABLE 4 (FoodID INT PRIMARY KEY, FoodQuantity INT PRIMARY KEY, FOREIGN KEY (OrderID) REFERENCES 2(OrderID))

CREATE TABLE 5 (DrinkName VARCHAR(100), FOREIGN KEY (DrinkID) REFERENCES 3(DrinkID))

CREATE TABLE 6 (FoodName VARCHAR(100), FOREIGN KEY (FoodID) REFERENCES 4(FoodID))

CREATE TABLE 7 (, FOREIGN KEY (OrderID) REFERENCES 2(OrderID), FOREIGN KEY (DrinkID) REFERENCES 3(DrinkID), FOREIGN KEY (FoodID) REFERENCES 4(FoodID))

CREATE TABLE 8 (PromocodeUsed VARCHAR(100), FOREIGN KEY (OrderID) REFERENCES 2(OrderID), FOREIGN KEY (DrinkID) REFERENCES 3(DrinkID), FOREIGN KEY (FoodID) REFERENCES 4(FoodID))

CREATE TABLE 9 (DrinkIngredient VARCHAR(100), FOREIGN KEY (OrderID) REFERENCES 2(OrderID), FOREIGN KEY (DrinkID) REFERENCES 3(DrinkID), FOREIGN KEY (FoodID) REFERENCES 4(FoodID))

CREATE TABLE 10 (DrinkAllergen VARCHAR(50), FOREIGN KEY (OrderID) REFERENCES 2(OrderID), FOREIGN KEY (DrinkID) REFERENCES 3(DrinkID), FOREIGN KEY (FoodID) REFERENCES 4(FoodID))

CREATE TABLE 11 (FoodIngredient VARCHAR(100), FOREIGN KEY (OrderID) REFERENCES 2(OrderID), FOREIGN KEY (DrinkID) REFERENCES 3(DrinkID), FOREIGN KEY (FoodID) REFERENCES 4(FoodID))

CREATE TABLE 12 (FoodAllergen VARCHAR(100), FOREIGN KEY (OrderID) REFERENCES 2(OrderID), FOREIGN KEY (DrinkID) REFERENCES 3(DrinkID), FOREIGN KEY (FoodID) REFERENCES 4(FoodID))
```

BCNF :

For Boyce-Codd Normal Form, the program checks if the left-hand side of each functional dependency is a superkey. If not, it separates attributes to enforce BCNF, eliminating all remaining dependencies where the left-hand side is not a superkey.

Sample Output:

```
BCNF SCHEMA
CREATE TABLE 1 (CustomerID INT PRIMARY KEY, CustomerName VARCHAR(100))

CREATE TABLE 2 (OrderID INT PRIMARY KEY, Date DATE, TotalCost VARCHAR(50), TotalDrinkCost VARCHAR(50), TotalFoodCost VARCHAR(50), FOREIGN KEY (CustomerID) REFERENCES 1(CustomerID)
)

CREATE TABLE 3 (DrinkID INT PRIMARY KEY, DrinkSize VARCHAR(100), DrinkQuantity INT PRIMARY KEY, Milk VARCHAR(100), FOREIGN KEY (OrderID) REFERENCES 2(OrderID))

CREATE TABLE 4 (FoodID INT PRIMARY KEY, FoodQuantity INT PRIMARY KEY, FOREIGN KEY (OrderID) REFERENCES 2(OrderID))

CREATE TABLE 5 (DrinkName VARCHAR(100), FOREIGN KEY (DrinkID) REFERENCES 3(DrinkID))

CREATE TABLE 6 (FoodName VARCHAR(100), FOREIGN KEY (FoodID) REFERENCES 4(FoodID))

CREATE TABLE 7 (, FOREIGN KEY (OrderID) REFERENCES 2(OrderID), FOREIGN KEY (DrinkID) REFERENCES 3(DrinkID), FOREIGN KEY (FoodID) REFERENCES 4(FoodID))

CREATE TABLE 8 (PromocodeUsed VARCHAR(100), FOREIGN KEY (OrderID) REFERENCES 2(OrderID), FOREIGN KEY (DrinkID) REFERENCES 3(DrinkID), FOREIGN KEY (FoodID) REFERENCES 4(FoodID))

CREATE TABLE 9 (DrinkIngredient VARCHAR(100), FOREIGN KEY (OrderID) REFERENCES 2(OrderID), FOREIGN KEY (DrinkID) REFERENCES 3(DrinkID), FOREIGN KEY (FoodID) REFERENCES 4(FoodID))

CREATE TABLE 10 (DrinkAllergen VARCHAR(50), FOREIGN KEY (OrderID) REFERENCES 2(OrderID), FOREIGN KEY (DrinkID) REFERENCES 3(DrinkID), FOREIGN KEY (FoodID) REFERENCES 4(FoodID))

CREATE TABLE 11 (FoodIngredient VARCHAR(100), FOREIGN KEY (OrderID) REFERENCES 2(OrderID), FOREIGN KEY (DrinkID) REFERENCES 3(DrinkID), FOREIGN KEY (FoodID) REFERENCES 4(FoodID))

CREATE TABLE 12 (FoodAllergen VARCHAR(100), FOREIGN KEY (OrderID) REFERENCES 2(OrderID), FOREIGN KEY (DrinkID) REFERENCES 3(DrinkID), FOREIGN KEY (FoodID) REFERENCES 4(FoodID))

_____
```

4NF Normalization:

`convert_to_4NF(tables, FD, multi_valued_dependencies)`:

- **Purpose:** Ensures relations meet Fourth Normal Form (4NF) by eliminating multi-valued dependencies (MVDs).
- **Steps:**
    - Identify Multi-Valued Dependencies: Analyzes the dataset to detect cases where multiple independent attributes are dependent on the same key (MVDs).
    - Separate Independent Sets: For each MVD, splits the relation into new tables to isolate independent sets of attributes.
    - Ensure Atomic Values: Ensures each resulting table holds only attributes directly dependent on its keys.
- **Input:**

    - `tables`: Dictionary containing the existing table structures.
    - `FD`: List of functional dependencies.
    - `multi_valued_dependencies`: List of multi-valued dependencies.
- **Output:** Returns tables converted to 4NF, ensuring no non-trivial multi-valued dependencies remain.

Sample Output:

```
4NF
CREATE TABLE 1 (OrderID INT PRIMARY KEY, PromocodeUsed VARCHAR(100))

CREATE TABLE 2 (DrinkID INT PRIMARY KEY, DrinkIngredient VARCHAR(100), FOREIGN KEY (OrderID) REFERENCES 1(OrderID))

CREATE TABLE 3 (DrinkAllergen VARCHAR(50), FOREIGN KEY (OrderID) REFERENCES 1(OrderID), FOREIGN KEY (DrinkID) REFERENCES 2(DrinkID))

CREATE TABLE 4 (FoodID INT PRIMARY KEY, FoodIngredient VARCHAR(100), FOREIGN KEY (OrderID) REFERENCES 1(OrderID))

CREATE TABLE 5 (FoodAllergen VARCHAR(100), FOREIGN KEY (OrderID) REFERENCES 1(OrderID), FOREIGN KEY (FoodID) REFERENCES 4(FoodID))

CREATE TABLE 6 (CustomerID INT PRIMARY KEY, CustomerName VARCHAR(100))

CREATE TABLE 7 (Date DATE, TotalCost VARCHAR(50), TotalDrinkCost VARCHAR(50), TotalFoodCost VARCHAR(50), FOREIGN KEY (OrderID) REFERENCES 1(OrderID), FOREIGN KEY (CustomerID) REFE
RENCES 6(CustomerID))

CREATE TABLE 8 (DrinkSize VARCHAR(100), DrinkQuantity INT PRIMARY KEY, Milk VARCHAR(100), FOREIGN KEY (OrderID) REFERENCES 1(OrderID), FOREIGN KEY (DrinkID) REFERENCES 2(DrinkID))

CREATE TABLE 9 (FoodQuantity INT PRIMARY KEY, FOREIGN KEY (OrderID) REFERENCES 1(OrderID), FOREIGN KEY (FoodID) REFERENCES 4(FoodID))

CREATE TABLE 10 (DrinkName VARCHAR(100), FOREIGN KEY (DrinkID) REFERENCES 2(DrinkID))

CREATE TABLE 11 (FoodName VARCHAR(100), FOREIGN KEY (FoodID) REFERENCES 4(FoodID))
```

5NF Normalization:

- I have implemented the program in such a way that, when you run a program, you will get the 5NF tables automatically and also the output files. The join dependencies are all already coded in the code itself.
- Here 5NF is also implemented in another way - with a traditional approach. Choosing the highest normal form and then processing. I guess both works fine. This approach asks for join dependencies also.

Sample Output:

```
Tables after 1NF: {'Candidate': ['OrderID', 'Date', 'TotalCost', 'TotalDrinkCost', 'TotalFoodCost', 'CustomerID', 'CustomerName', 'DrinkID', 'DrinkName', 'DrinkSize', 'DrinkQuanti
ty', 'Milk', 'FoodID', 'FoodName', 'FoodQuantity'], 'PromocodeUsed': ['OrderID', 'DrinkID', 'FoodID', 'PromocodeUsed'], 'DrinkIngredient': ['OrderID', 'DrinkID', 'FoodID', 'DrinkI
ngredient'], 'DrinkAllergen': ['OrderID', 'DrinkID', 'FoodID', 'DrinkAllergen'], 'FoodIngredient': ['OrderID', 'DrinkID', 'FoodID', 'FoodIngredient'], 'FoodAllergen': ['OrderID',
'DrinkID', 'FoodID', 'FoodAllergen']}
R1 (DrinkID, Milk):
   DrinkID Milk
0        1   ND
1        1    D
2        2    D
3        3   ND
4        3    D
5        4   ND

R2 (OrderID, CustomerID, DrinkID):
   OrderID  CustomerID  DrinkID
0     1001           1        1
2     1002           1        2
3     1003           2        3
5     1003           2        4

Reconstructed Table by joining R1 and R2:
   OrderID  CustomerID  DrinkID Milk
0     1001           1        1   ND
1     1001           1        1    D
2     1002           1        2    D
3     1003           2        3   ND
4     1003           2        3    D
5     1003           2        4   ND

5NF decomposition is successful; the join is lossless.
```

```
All unique tables after normalization
{1: ['DrinkID', 'Milk'], 2: ['CustomerID', 'DrinkID', 'OrderID']}

SQL Queries
CREATE TABLE 1 (DrinkID INT, Milk VARCHAR(100), PRIMARY KEY (DrinkID, Milk))

CREATE TABLE 2 (CustomerID INT, DrinkID INT, OrderID INT, PRIMARY KEY (CustomerID, DrinkID, OrderID), FOREIGN KEY (DrinkID) REFERENCES 1(DrinkID))
```