CS 5300 DATABASE SYSTEMS

Programming Project

Project Members:

Srikala Jalla - 12616020

Kishore Kumar Jami – 12614380

Objective: To develop a program that takes a database (relations) and functional dependencies as input, normalizes the relations based on the provided functional dependencies, produces SQL queries to generate the normalized database tables, and optionally determines the highest normal form of the input table.

Input file:

- Sampledata.csv

Output file:

- normalization_output1.txt

Sample Input Dataset:

Here are the attributes (columns) used in the **Sampledata.csv** file:

1. **OrderID** - Identifier for each order
2. **Date** - Date when the order was placed
3. **PromocodeUsed** - Indicates if a promotional code was used
4. **TotalCost** - Total cost of the order
5. **TotalDrinkCost** - Total cost for drinks in the order
6. **TotalFoodCost** - Total cost for food in the order
7. **CustomerID** - Unique identifier for the customer
8. **CustomerName** - Name of the customer

9. **DrinkID** - Identifier for each drink item
10. **DrinkName** - Name of the drink
11. **DrinkSize** - Size of the drink ordered
12. **DrinkQuantity** - Quantity of the drink ordered
13. **Milk** - Type of milk (if any) used in the drink
14. **DrinkIngredient** - Ingredients used in the drink
15. **DrinkAllergen** - Allergens present in the drink
16. **FoodID** - Identifier for each food item
17. **FoodName** - Name of the food item
18. **FoodQuantity** - Quantity of the food item ordered
19. **FoodIngredient** - Ingredients used in the food item
20. **FoodAllergen** - Allergens present in the food item

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

OrderID → CustomerID

- Each OrderID uniquely determines a CustomerID.

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

- OrderID has a multi-valued dependency on DrinkID, indicating that an order can have multiple drinks associated with it.

Core Components:

The **core components** used in your database normalization project likely include the following elements:

1. **Functional Dependency Analysis**:
    - Identifying dependencies like OrderID -> CustomerID and multi-valued dependencies such as OrderID ->> DrinkID to guide the normalization process.
2. **Normalization Process**:
    - Applying each normal form (1NF, 2NF, 3NF, BCNF, and 4NF) systematically to decompose tables and ensure minimal redundancy.
    - Breaking down the schema based on the functional and multi-valued dependencies to achieve the highest possible normal form.
3. **Data Structures**:
    - Tables representing different stages of normalization, each defined with a unique set of attributes, primary keys, and foreign keys.
    - Example: Tables created during 4NF decomposition to handle multi-valued dependencies separately.
4. **SQL Query Generation**:
    - SQL CREATE TABLE statements to define the structure of normalized tables for each normal form.
    - Defining primary keys and foreign keys in SQL to enforce referential integrity and dependency constraints.
5. **Python and Pandas**:
    - Using Python (with libraries like Pandas) for data processing, parsing input data, and verifying dependencies.

- Python functions and methods for loading, analyzing, and outputting data in normalized form.

6. **User-Defined File Paths and Encodings**:
    - File paths for both input data (e.g., Sampledata.csv) and output files where results are saved.
    - Encoding specifications to handle data file compatibility.

Deliverables:

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

Code Description:

Library Imports and File Handling:

- The code imports pandas for data handling and combinations from itertools to manage dependencies.
- **Input File**: Loads Sampledata.csv using pd.read_csv to read customer and order data.
- **Output File**: Specifies an output file, normalization_output1.txt, to save results from the normalization process.

User Prompts:

- **Primary Keys**: Asks the user to enter primary keys for the table.
- **Functional Dependencies**: Allows the user to input functional dependencies in the format A,B -> C,D, which are stored in a dictionary.
- **Highest Normal Form Selection**: Prompts the user to select the highest normal form they want to achieve (from 1NF to 5NF).
- **Multi-Valued Dependencies (4NF)**: Prompts the user for multi-valued dependencies in the format A ->> B if targeting 4NF or higher.

- **Join Dependencies (5NF)**: Prompts for join dependencies for 5NF if needed.

Normalization Functions:

- **apply_1nf**: This function converts non-atomic columns (e.g., lists in cells) to atomic form by exploding these lists into separate rows, ensuring 1NF compliance.
- **validate_2nf**: Checks if the table meets 2NF by confirming that no non-prime attribute is partially dependent on a candidate key.
- **validate_3nf**: Ensures 3NF compliance by checking that every non-key attribute depends only on a superkey.
- **validate_bcnf**: Validates BCNF by verifying that all functional dependencies are fully dependent on candidate keys.
- **apply_4nf**: Decomposes the table into separate tables to remove multi-valued dependencies and achieve 4NF.
- **apply_5nf**: Applies join dependency rules to decompose tables further if needed to reach 5NF.

SQL Schema Generation:

- **generate_sql_schema**: Creates CREATE TABLE SQL statements for each table and writes schema details (columns, data types, primary keys) to the output file.

Main Normalization Process (normalize_data):

- This function coordinates the normalization process by:
    - Applying 1NF and creating an initial SQL schema.
    - Validating and decomposing tables as needed to achieve each target normal form up to the specified level.
    - Writing achieved normal forms and details about each decomposition to the output file.

Execution:

- The normalize_data function is called with the user-defined primary keys, functional dependencies, multi-valued dependencies, and join dependencies, performing normalization and saving results in the output file.

Code Execution:

To execute the code in the notebook, follow these steps:

1. **Setup the Environment**:
    - Ensure you have Python installed with the necessary libraries, such as pandas.
    - Install pandas if not already installed:

    pip install pandas
2. **Prepare the Input File**:
    - Place the Sampledata.csv file in the specified directory or adjust the file path in the code to match the actual location of the input file.
3. **Run the Code**:
    - Open the notebook in Jupyter Notebook
    - Execute each cell in sequence by pressing **Shift + Enter**.
4. **Input Prompts**:
    - When prompted, provide the necessary inputs:
        - **Primary Keys**: Enter the primary key(s) separated by commas.
        - **Functional Dependencies**: Type dependencies in the format A,B -> C,D, one per line, and type done when finished.
        - **Highest Normal Form**: Enter the target normal form level (1 for 1NF, 2 for 2NF, etc.).

- **Multi-Valued Dependencies** (if 4NF or higher): Enter in A ->> B format, and type done when finished.
- **Join Dependencies** (if 5NF): Enter dependencies in the format A,B, and type done when finished.

5. **Output**:
   - The output will be saved in normalization_output1.txt in the specified directory. It will contain:
     - Functional dependencies, multi-valued dependencies, and join dependencies.
     - SQL CREATE TABLE statements for each normalized table.
     - The highest achieved normal form and details of each decomposition.
6. **Review the Output**:
   - Open normalization_output1.txt to view the results, which will include detailed normalization steps and SQL schema for each normalized table.