**CS5300 PROGRAMMING PROJECT01**

**DATABASE NORMALIZATION**

**Team:** Sura Dinesh(12608152), Bhavana Dasari(12619331)

**Sample Input:**

**Dataset:** The sample data contains attributes like:

Primary key(s), separated by commas: OrderID, CustomerID, DrinkID, FoodID

**Functional Dependencies:** Functional Dependencies in format 'A,B -> C,D' (type 'done' when finished):

OrderID -> CustomerID

Done

**Multi-valued dependencies:** Multi-Valued Dependencies in format 'A ->> B' (type 'done' when finished):

OrderID ->> DrinkID

Done

**JOIN Dependecies:**

Join Dependencies in format 'A,B' (type 'done' when finished):

OrderID,CustomerID

OrderID,DrinkID

OrderID,FoodID

**Core Components:**

1. **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.
2. **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.
3. **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 and Execution**

This Python script automates the process of normalizing a database table up to a specified normal form level (1NF to 5NF). The script reads a CSV file for the dataset and uses user-defined functional dependencies (FDs), multi-valued dependencies (MVDs), and join dependencies (JDs) to normalize the data. The final output includes SQL CREATE TABLE statements and schema details, which are saved to a text file on the desktop.

**Code Sections**

**1. Import Libraries**

import pandas as pd

from itertools import combinations

* **pandas**: Used for data manipulation and handling the CSV file.
* **itertools.combinations**: Used for generating combinations of columns to check dependencies.

**2. Data Loading and Output File Setup**

input\_file = 'C:/Users/dinesh/OneDrive/Desktop/Sampledata.csv'

data = pd.read\_csv(input\_file, encoding='ISO-8859-1')

output\_file = 'C:/Users/dinesh/OneDrive/Desktop/Sampledata.output.

* Loads the CSV file containing the sample dataset and creates an output text file to save results, including SQL queries and schema details.

**3. User Input for Primary Keys, Dependencies, and Target Normal Form**

* **Primary Key**:
  + User specifies primary key attributes as a comma-separated list.
* **Functional Dependencies (FDs)**:
  + Prompted in the format A,B -> C,D and stored in a dictionary.
* **Multi-Valued Dependencies (MVDs)**:
  + Collected in the format A ->> B and stored in a dictionary.
* **Normalization Level**:
  + User selects the desired highest normal form level.
* **Join Dependencies (JDs)** (for 5NF only):
  + Collected as a comma-separated list of attributes for each JD.

**4. Normalization Functions**

Each function in this section verifies or decomposes the data to satisfy a specific normal form.

* **1NF Conversion**:

def check\_and\_convert\_1NF(df):

...

return df, non\_atomic\_columns

* + Ensures atomicity by "exploding" any non-atomic columns (e.g., lists or sets) and logging columns that were adjusted.
* **2NF Check**:

def check\_2NF(df, primary\_key, FDs):

...

return True or False

* + Checks for partial dependencies. If any non-prime attribute depends on part of a composite primary key, 2NF is not satisfied.
* **3NF Check**:

def check\_3NF(df, primary\_key, FDs):

...

return True or False

* + Ensures that non-prime attributes are fully dependent on superkeys and verifies the absence of transitive dependencies.
* **BCNF Check**:

def check\_BCNF(df, primary\_key, FDs):

...

return True or False

* + Checks that each determinant is a superkey. If not, BCNF is violated, and decomposition may be required.
* **4NF Decomposition** (based on MVDs):

def check\_and\_decompose\_4NF(df, MVDs):

...

return decomposed\_tables

* + Decomposes the table to remove multi-valued dependencies, creating separate tables as needed.
* **5NF Decomposition** (based on JDs):

def check\_and\_decompose\_5NF(df, JDs):

...

return decomposed\_tables

* + Decomposes the table based on join dependencies, applicable for 5NF normalization.

**5. SQL Query and Schema Generation**

def generate\_sql\_and\_schema(table\_name, df, primary\_keys):

...

* Generates SQL CREATE TABLE queries for each normalized table, specifying data types and primary keys.
* Writes the SQL statements and schema details to the output file.

**6. Main Normalization Process**

def normalize\_database(df, primary\_key, FDs, MVDs, max\_normal\_form, JDs=[]):

...

* This function orchestrates the normalization process by sequentially applying each normal form check and decomposition function up to the specified normal form level:
  + **1NF**: Converts columns to atomic values.
  + **2NF**: Decomposes partial dependencies.
  + **3NF**: Decomposes transitive dependencies.
  + **BCNF**: Ensures every determinant is a candidate key.
  + **4NF**: Decomposes multi-valued dependencies.
  + **5NF**: Decomposes join dependencies, if specified.
* Logs each step and writes the final SQL schema to the output file.

**7. Execution of Normalization**

normalize\_database(data, primary\_key, FDs, MVDs, max\_normal\_form, JDs)

* Calls the normalize\_database function to execute the normalization process based on the user inputs for primary keys, FDs, MVDs, and JDs.
* Writes a summary of the highest normal form achieved in the output file.

**Code Execution Steps**

1. **Prepare the Dataset**:
   * Ensure Sampledata.csv is in the specified file path.
2. **Run the Script**:
   * Execute the script in a Python environment, such as Jupyter Notebook, or by running the Python file directly.
3. **Follow User Prompts**:
   * **Enter Primary Keys**: Specify primary keys as a comma-separated list.
   * **Enter Functional Dependencies (FDs)**: Use the format A,B -> C,D.
   * **Enter Multi-Valued Dependencies (MVDs)**: Use the format A ->> B.
   * **Select Target Normal Form**: Choose the highest normal form to normalize to (1NF, 2NF, 3NF, BCNF, 4NF, 5NF).
   * **Enter Join Dependencies (JDs)**: For 5NF, enter attributes for each JD in a comma-separated format.
4. **Review Output**:
   * Open normalization\_output.txt on the desktop to view:
     + SQL CREATE TABLE statements for each decomposed table.
     + Schema details, including primary keys and any decompositions.
     + The highest normal form achieved.

**Output Summary**

* **SQL Statements**: SQL queries for each normalized table.
* **Schema Representation**: Attributes, data types, and primary keys for each table.
* **Normalization Summary**: Logs each normalization step, decomposition details, and the highest normal form achieved.

This script provides an efficient, user-driven approach to database normalization, automating the decomposition process and outputting ready-to-use SQL queries for the specified normal form.



