**CS5300 PROGRAMMING PROJECT01**

**DATABASE NORMALIZATION**

**Team members**

Kummathi kalpana(12610771), Nithish Yadav Thotla(12612279)

**Sample Input:**

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

**Functional Dependencies:**

OrderID -> CustomerID

Done

**Multi-valued dependencies:**

OrderID ->> DrinkID

Done

**JOIN Dependecies:**

OrderID,CustomerID

OrderID,DrinkID

OrderID,FoodID

**Code Description and Execution**

This Python script streamlines the process of normalizing a database table to a specified normal form level, ranging from 1NF to 5NF. It reads a CSV file containing the dataset and leverages user-specified functional dependencies (FDs), multi-valued dependencies (MVDs), and join dependencies (JDs) to perform the normalization. The final output, including SQL CREATE TABLE statements and schema information, is saved to a text file on the desktop for easy access.Detailed Code Sections

**1. Import Libraries**

import pandas as pd

from itertools import combinations

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

data\_file\_path = 'C:/Users/saiki/OneDrive/Desktop/Sampledata.csv'

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

output\_file\_path = 'C:/Users/saiki/OneDrive/Desktop/output1.txt'

* Loads the CSV file containing the dataset and prepares an output text file to record results, including SQL statements and schema details.

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

* **Primary Keys**:
  + Prompts for primary key attributes as a comma-separated list, then parses them into a list.
* **Functional Dependencies (FDs)**:
  + Collects FDs in the format X,Y -> Z,W and stores them in a dictionary where keys represent the left-hand side (LHS) of the dependency.
* **Multi-Valued Dependencies (MVDs)**:
  + If the target form is 4NF or higher, prompts for MVDs in the format A ->> B and stores them in a dictionary.
* **Join Dependencies (JDs)** (for 5NF only):
  + For 5NF, collects JDs as a comma-separated list of attributes for each dependency.

**4. Normalization Functions**

Each function checks and/or decomposes the data to satisfy a specific normal form level:

* **1NF Conversion**:

def enforce\_1nf(df):

atomic\_cols = []

for col in df.columns:

if df[col].apply(lambda x: isinstance(x, list)).any():

atomic\_cols.append(col)

df = df.explode(col)

return df, atomic\_cols

* + **Purpose**: Ensures all attributes are atomic by expanding non-atomic columns.
  + **Output**: Returns a DataFrame with atomic values and a list of columns that were adjusted for 1NF.
* **2NF Check**:

def check\_2nf(df, pk\_columns, deps):

non\_pk\_columns = [col for col in df.columns if col not in pk\_columns]

for lhs, rhs in deps.items():

if set(lhs).issubset(pk\_columns) and any(attr in non\_pk\_columns for attr in rhs):

return False

return True

* + **Purpose**: Verifies 2NF by ensuring no partial dependencies exist on any part of a composite primary key.
  + **Output**: Returns True if 2NF is met, otherwise False.
* **3NF Check**:

def check\_3nf(df, pk\_columns, deps):

for lhs, rhs in deps.items():

if set(lhs) != set(pk\_columns) and not set(rhs).issubset(pk\_columns):

return False

return True

* + **Purpose**: Ensures that non-prime attributes are fully dependent on superkeys.
  + **Output**: Returns True if 3NF is satisfied, otherwise False.
* **BCNF Check**:

def check\_bcnf(df, pk\_columns, deps):

for lhs, rhs in deps.items():

if not set(lhs).issubset(pk\_columns):

return False

return True

* + **Purpose**: Checks that each determinant is a superkey.
  + **Output**: Returns True if BCNF is met, otherwise False.
* **4NF Decomposition (for MVDs)**:

def apply\_4nf(df, mv\_deps):

decomposed\_tables = []

remaining\_columns = list(df.columns)

for lhs, rhs\_list in mv\_deps.items():

for rhs in rhs\_list:

decomposed\_table = df[[lhs, rhs]].drop\_duplicates()

decomposed\_tables.append((decomposed\_table, [lhs, rhs]))

if rhs in remaining\_columns:

remaining\_columns.remove(rhs)

output\_file.write(f"4NF Decomposition: Created table with {lhs} ->> {rhs}\n")

main\_table = df[remaining\_columns].drop\_duplicates()

return [(main\_table, primary\_key\_columns)] + decomposed\_tables

* + **Purpose**: Decomposes the table to remove multi-valued dependencies, creating separate tables as needed.
  + **Output**: Returns a list of decomposed tables with their primary keys.
* **5NF Decomposition (for JDs)**:

def apply\_5nf(df, jd\_deps):

decomposed\_tables = []

for jd in jd\_deps:

if set(jd).issubset(df.columns):

decomposed\_table = df[jd].drop\_duplicates()

decomposed\_tables.append((decomposed\_table, jd))

output\_file.write(f"5NF Decomposition: Created table for JD ({', '.join(jd)})\n")

if not decomposed\_tables:

output\_file.write("No further 5NF decomposition required.\n")

return [(df, primary\_key\_columns)] + decomposed\_tables

* + **Purpose**: Decomposes the table based on join dependencies, as required for 5NF normalization.
  + **Output**: Returns a list of decomposed tables based on specified join dependencies.

**5. SQL Query Generator**

def generate\_sql\_schema(table\_name, df, pk\_columns):

sql\_query = f"CREATE TABLE {table\_name} (\n"

for col, dtype in zip(df.columns, df.dtypes):

col\_type = 'INT' if 'int' in str(dtype) else 'VARCHAR(255)'

sql\_query += f" {col} {col\_type},\n"

pk\_str = ", ".join(pk\_columns)

sql\_query += f" PRIMARY KEY ({pk\_str})\n);"

output\_file.write(sql\_query + "\n\n")

* **Purpose**: Generates CREATE TABLE SQL statements for each decomposed table, specifying column data types and primary keys.
* **Output**: Writes SQL statements to the output file, allowing for database creation based on the normalized schema.

**6. Main Normalization Process**

def normalize\_data(df, pk\_columns, f\_deps, mv\_deps, target\_form, jd\_deps=[]):

...

* **Purpose**: Coordinates the normalization process by applying each normal form check and decomposition function in sequence up to the specified target form.
* **Steps**:
  + **1NF**: Converts columns to atomic values.
  + **2NF**: Eliminates partial dependencies.
  + **3NF**: Removes transitive dependencies.
  + **BCNF**: Ensures every determinant is a candidate key.
  + **4NF**: Decomposes based on multi-valued dependencies.
  + **5NF**: Decomposes based on join dependencies.
* **Output**: Logs each normalization step and writes the final SQL schema to the output file.

**Execution Steps**

1. **Prepare the Dataset**: Ensure Sampledata.csv is at the specified location.
2. **Run the Script**: Execute in a Python environment (e.g., Jupyter Notebook or directly in Python).
3. **User Prompts**:
   * Enter **Primary Keys** as a comma-separated list.
   * Enter **FDs, MVDs, and JDs** as prompted.
   * Select **Target Normal Form** level.

**Sample OUTPUT**

A screenshot of a computer

Description automatically generatedA screenshot of a computer

Description automatically generated