CS5300 Database Systems

Programming Project: RDBMS Normalizer

Team members: SaiRaj Aggani , Hrishitha Reddy Mandadi

Key Files:

- MainData.csv: The input dataset containing initial data.
- FunctionalDependencies.txt: The text file with user-defined functional dependencies.
- Project1.py: The Python script that performs normalization, generates SQL queries, and saves them to an output file.
- Output.sql: File where generated SQL queries are saved.

Dataset:

The example data includes the following attributes: CSV file with columns: OrderID, Date, PromocodeUsed, TotalCost, TotalDrinkCost, TotalFoodCost, CustomerID, CustomerName, DrinkID, DrinkName, DrinkSize, DrinkQuantity, Milk, DrinkIngredient, DrinkAllergen, FoodID, FoodName, FoodQuantity, FoodIngredient, FoodAllergen.

Functional Dependencies:

1. **Order-related FDs:**
    - OrderID → Date, PromocodeUsed, TotalCost, TotalDrinkCost, TotalFoodCost, CustomerID
        - An OrderID uniquely determines the date, promo codes used, total costs, and associated CustomerID.
    - CustomerID → CustomerName
        - A CustomerID uniquely determines the CustomerName.
2. **Drink-related FDs:**
    - DrinkID → DrinkName, DrinkSize, Milk, DrinkIngredient, DrinkAllergen
        - A DrinkID uniquely determines the name, size, milk type, ingredients, and allergens.
    - OrderID, DrinkID → DrinkQuantity
        - The combination of OrderID and DrinkID determines the DrinkQuantity for that order.
3. **Food-related FDs:**
    - FoodID → FoodName, FoodIngredient, FoodAllergen
        - A FoodID uniquely determines the name, ingredients, and allergens.
    - OrderID, FoodID → FoodQuantity
        - The combination of OrderID and FoodID determines the FoodQuantity for that order.

Multi Valued Dependencies:

The multi-valued dependencies used for the dataset are:

1) OrderID ↠ DrinkID

   For each OrderID, there can be multiple DrinkID values associated with it, representing the different drinks included in a single order. The choice of drinks in an order is independent of any food items or promo codes applied.

2) OrderID ↠ FoodID

   For each OrderID, there can be multiple FoodID values associated with it, representing different food items included in the same order. This means that the food items in an order are independent of the drinks chosen.

3) DrinkID ↠ DrinkIngredient

   For each DrinkID, there can be multiple DrinkIngredient values, which specify the ingredients used in that particular drink. For example, a specific coffee drink may contain multiple ingredients like espresso, milk, and syrup, which are independent of other drink attributes.

4) FoodID ↠ FoodIngredient

   For each FoodID, there can be multiple FoodIngredient values. For example, a muffin might contain flour, sugar, eggs, and blueberries, independent of other characteristics of the food item.

Main Features:

Normalization to Different Forms:

This supports normalization to:

- 1NF (First Normal Form)
- 2NF (Second Normal Form)
- 3NF (Third Normal Form)
- BCNF (Boyce-Codd Normal Form)
- 4NF (Fourth Normal Form)
- 5NF (Fifth Normal Form)

Functional Dependency Parsing:

Reads functional dependencies (FDs) and multivalued dependencies (MVDs) from a text file and organizes them for processing. The format is either X -> Y for standard FDs or X --> Y for MVDs.

SQL Query Generation:
Automatically generates SQL queries for creating tables that match the normalized forms, saving them in an output file (Output.sql).

Workflow:

Input Data:
A CSV file containing the initial dataset (e.g., MainData.csv).
A text file (FunctionalDependencies.txt) listing the functional dependencies in the format {determinants} -> {dependents}.
Normalization Process:
Read the dataset and functional dependencies.
Normalize the schema progressively to the specified normal form, generating intermediate tables and SQL creation queries.
Handle partial, transitive, and multivalued dependencies to ensure higher normal form compliance.
Output:
The generated SQL queries for the normalized tables are saved in Output.sql.
Users can view the SQL commands for creating the tables in their chosen normal form.
Deliverables:
Source Code:
we used Python programming language to normalize the given dataset Code
Description:
The Project1.py script normalizes a relational database schema based on user-defined functional dependencies, guiding it to a chosen normal form (1NF to 5NF). It generates SQL queries for the creation of normalized tables, ensuring the data is efficiently structured without redundancy or dependency issues.

1. Imports and Dependencies
Libraries:
pandas: For reading and manipulating CSV data.
os, re: Used for file handling and regular expressions.
defaultdict, combinations (from collections and itertools): For storing and processing functional dependencies and attribute combinations.
Custom Classes:
FunctionalDependency: Represents a functional or multivalued dependency between attributes.
2. Class: FunctionalDependency
Attributes:
determinants: Set of attributes on the left-hand side (LHS) of the dependency.
dependents: Set of attributes on the right-hand side (RHS) of the dependency.

is_multivalued: Boolean indicating if the dependency is multivalued (MVD).

Methods:
__str__(): Returns a string representation of the dependency, formatted as {determinants} -> {dependents} or {determinants} --> {dependents} for MVDs.

3. Functions for Parsing and Input Handling

parse_fd_file(file_path):
Reads functional dependencies from a specified file.
Supports both standard FDs (X -> Y) and MVDs (X --> Y).
Returns a list of FunctionalDependency objects.

read_csv(file_path):
Reads a CSV file and returns a pandas DataFrame.
Handles file-not-found errors.

4. Normalization Functions

normalize_to_1nf(df, primary_keys):
Converts a DataFrame to 1NF by splitting multivalued attributes into separate tables.
Generates SQL queries for the 1NF schema.
Returns a list of SQL queries and information about the created tables.

generate_2nf_queries(tables_info, fds), generate_3nf_queries(), generate_bcnf_queries(), generate_4nf_queries(), generate_5nf_queries():
Progressively normalize tables to higher normal forms.
Each function identifies specific dependency violations and decomposes tables to address them.
Generates SQL queries and stores information about each decomposed table.

5. Dependency Checking Functions

check_partial_dependencies():
Identifies partial dependencies for normalization to 2NF.
Returns tables that require decomposition.

find_transitive_dependencies():
Finds transitive dependencies for normalization to 3NF.

find_4nf_violations():
Detects non-trivial MVDs to ensure compliance with 4NF.

find_join_dependencies():
Identifies join dependencies for ensuring 5NF, checking for lossless decompositions.

6. Helper Functions

compute_closure():
Computes the attribute closure for a given set of attributes under the defined FDs.
Used to determine if a set of attributes is a superkey.

is_superkey():
Checks if a given set of attributes qualifies as a superkey by comparing the closure with all attributes.

validate_mvd():
Validates if a given MVD holds in the dataset.
is_lossless_join():
Checks if a decomposition satisfies the lossless join property.
7. Main Function: main()
User Input:
Asks the user to input primary keys.
For given Table : OrderID,FoodID,DrinkID

Reads the desired highest normal form (1NF to 5NF).

- ➔ Enter desired normal form number and enter
- ➔ Normalized queries are generated

Workflow:
Reads the dataset and functional dependencies.
Calls normalization functions based on the user's choice.
Saves generated SQL queries to Output.sql using save_queries_to_file().
Output:
Displays the generated SQL queries for the normalized database schema.
8. save_queries_to_file()
Saves the generated SQL queries into a specified file (Output.sql).
Formats each query with comments indicating the target table.