CS5300 Database Systems

Programming Project: RDBMS Normalizer

Team members: SaiRaj Aggani, Hrishitha Reddy Mandadi

Main Program
Main program performs normalization of given table from 1NF to 5NF
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:

OrderID -> Date, TotalCost, TotalDrinkCost, TotalFoodCost, CustomerID, CustomerName

OrderID, DrinkID -> DrinkName, DrinkSize, DrinkQuantity, Milk

OrderID, FoodID -> FoodQuantity

CustomerID -> CustomerName

DrinkID -> DrinkName

FoodID -> FoodName

Multi Valued Dependencies:

OrderID, DrinkID -->> FoodID , DrinkIngredient

OrderID, DrinkID -->> FoodID , DrinkAllergen

OrderID, FoodID -->> DrinkID , FoodIngredient

OrderID, FoodID -->> DrinkID , FoodAllergen

OrderID -->> DrinkID , FoodID

Main Features:

Normalization to Different Forms:

The program 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.

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.

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

Execution Process:
- Run python Project1.py.
- Enter primary keys : OrderID, DrinkID, FoodID
- Enter the desired normal form (e.g., 4 for BCNF).
- The script reads functional dependencies and dataset attributes, applies normalization, and outputs SQL table creation commands for the target normal form in both the console and Output.sql.

MVDAnalyzer:

The program to autonomously identify multi-valued dependencies without relying on user-provided MVD data. (mvd.py) and perform database normalization up to the Fifth Normal Form (5NF)

Key Components:
Files:
Mvd.py (Program file), test.csv (Input csv file)
Functional Dependency Detection:
- get_functional_dependencies(self):
- Identifies functional dependencies in the data by checking if for every value of one column, there is only one corresponding value in another column.

Multi-Valued Dependency (MVD) Checking:
- check_mvd_pattern(self, determinant_cols, dependent_col)
- Checks if there exists an MVD for the specified determinant columns and dependent column by grouping data and analyzing combinations.

4NF Decomposition:
- perform_4nf_decomposition(self):
- Decomposes the original table into 4NF based on the discovered MVDs, creating new tables with appropriate primary keys.

Join Dependency Identification:
- identify_join_dependencies(self):

- Looks for join dependencies among the tables generated from the 4NF decomposition**.**

5NF decomposition :

- perform_5nf_decomposition(self):
- Further decomposes the tables into 5NF based on the identified join dependencies.

SQL Query Generation

- generate_create_table_queries(self, tables):
- Generates SQL CREATE TABLE statements for the decomposed tables, inferring data types based on the original DataFrame.

Main Functionality

- analyze_and_print_normalization(csv_file):
- This is the main function that orchestrates the analysis. It creates an instance of MVDAnalyzer, prints identified MVDs, performs 4NF decomposition, and prints the corresponding SQL queries. It also performs 5NF decomposition and prints the relevant queries.

Example Usage

- In terminal run python script by
- python mvd.py

Output:

- Prints the identified mvds into terminal
- Prints the queries for table after performing 4NF and 5NF.

Domain-key normal form:

The DKNF Normalizer class implements an algorithm for normalizing a database schema to Domain-Key Normal Form (DKNF).

Key Components:

Data Loading:

load_data_from_csv(self, csv_path: str)

- Loads data from a specified CSV file(orders.csv).
- Extracts attributes from the CSV file's header.

load_constraints_from_file(self, constraints_path: str):

- Loads functional dependencies, key constraints, and domain constraints from a specified text file(constraints.txt).
- The text file should have the following format:
- Functional Dependencies (FD): FD: A,B -> C

- Key Constraints: KEY: A,B
- Domain Constraints: DOMAIN: A: value1,value2

Functional Dependency and Key Management:

_compute_closure(self, attributes: Set[str],
fds: List[Tuple[Set[str], Set[str]]]) -> Set[str]:

- Computes the closure of a set of attributes based on the provided functional dependencies. This is used to determine if a set of attributes is a superkey.

_is_superkey(self, attributes: Set[str], relation_attrs: Set[str]) -> bool:

- Checks if a given set of attributes forms a superkey for the relation by comparing the closure with the relation's attributes.

DKNF Decomposition

_decompose_to_dknf(self) -> List[Dict]:

- Performs the decomposition of relations to DKNF based on the key constraints.
- Creates separate relations for each key constraint and handles any remaining attributes.
- Prints the relations created during the decomposition process.

SQL Query Generation

generate_sql_queries(self, relations: List[Dict]) -> List[str]:

- Generates SQL CREATE TABLE statements for the normalized relations.
- Each attribute is assigned a data type of VARCHAR(255), and the primary key is defined for each relation.

Normalization Process

normalize(self):

- Executes the complete normalization process by decomposing the data to DKNF and generating the corresponding SQL queries.

Main Functionality

main():

- This is the entry point of the program. It creates an instance of DKNFNormalizer, loads data and constraints, and performs the normalization process.

Example Usage

To use this script in terminal Run by using command : **dknf.py**

Output:

- Prints the resultant queries after performing DKNF on the given CSV table.