DATABASE NORMALIZATION PROJECT DOCUMENTATION OF APPROACH EXPLANATION

This document provides an in-depth overview and explanation of the database normalization project, including approach methodology, code structure, and functionality for each component. This project normalizes input data through various normal forms (1NF, 2NF, 3NF, BCNF, 4NF,5NF) and generates SQL queries for creating normalized tables.

1. Project Overview

The primary goal of this project is to automate the normalization of a database schema based on user-provided functional dependencies (FDs) and multi-valued dependencies (MVDs). The project identifies dependencies, applies normalization techniques, and generates SQL queries that create tables in each normal form. The output SQL queries include primary keys and dynamic foreign key assignments, based on the dependencies, making the resulting schema adaptable to various datasets.

2. Approach Explanation

The project implements a systematic approach to normalize data based on input table data, user-defined functional dependencies (FDs) and multi-valued dependencies (MVDs). The primary steps in the approach are as follows:

Normalization Process

1. 1NF (First Normal Form): Ensures that all attribute values are atomic, meaning there are no repeating groups or multi-valued attributes.
   2. 2NF (Second Normal Form): Eliminates partial dependencies for tables with composite keys, ensuring that each non-key attribute is fully dependent on the primary key.
   3. 3NF (Third Normal Form): Removes transitive dependencies by ensuring all non-key attributes are only dependent on primary key attributes.
   4. BCNF (Boyce-Codd Normal Form): Ensures that every determinant is a superkey, eliminating any remaining anomalies after 3NF.
   5. 4NF (Fourth Normal Form): Removes multi-valued dependencies to achieve further decomposition.
   6.5NF(Fifth Normal Form): Is concerned with eliminating redundancy in a relation by ensuring that all join dependencies are a consequence of the candidate keys. A table is in 5NF if it is in 4NF and every non-trivial join dependency in the relation is implied by the candidate keys.

For each normal form, the program checks if the data satisfies the conditions. If a condition is not met, the program applies decomposition to produce relations that satisfy the required normal form.

SQL Query Generation

At each stage of normalization, SQL queries are generated for the normalized relations. These queries include constraints like datatype checking, primary key, foreign key relationships dynamically derived from the FDs and MVDs, ensuring that referential integrity is maintained between tables. The program outputs these queries as part of the normalization process.

3. Code Analysis and Documentation

Main Program (main.py)

The main.py file is the entry point for the project. It manages input processing, dependency handling, and the orchestration of the normalization steps. Key responsibilities include:

Reading Input:Loads the user-provided input table data from referenceInputTable.csv and functional dependencies from fds.txt. Then after user entered multi-valued dependencies .
Handling Dependencies: Passes dependencies to the normalization functions in normalizer.py.
Normalization Control: Coordinates the normalization steps by calling respective functions.
Output SQL Queries: Displays generated SQL queries for normalized relations. Creates SQL statements for each normalized relation, dynamically assigning primary and foreign keys based on detected dependencies.
Each function in main.py is documented with comments that describe its role, arguments, and functionality.

Normalizer Module (normalizer.py)

The normalizer.py file contains core functions to implement each stage of normalization. For each normal form, functions check if the relation satisfies the required conditions and, if not, decomposes the relation accordingly. Key functions include:
Normalization Functions (1NF to 5NF): Each function checks dependencies and decomposes relations if necessary to meet the specific normal form.
Decomposition Logic: Implements techniques for splitting relations based on partial and transitive dependencies, and for handling multi-valued dependencies.
Each function is thoroughly commented with descriptions of its purpose, arguments, and logic for implementing normalization.

Here's an outline of code with the breakdown of each component and function:
Below is the main.py file that where programs execution starts with dependencies: The main file serves as a tool to automate the process of database normalization, which is crucial for eliminating redundancy and ensuring data integrity in relational databases. This script reads data from a CSV file, processes functional dependencies (FDs) and multi-valued dependencies (MVDs), and normalizes the data through various normal forms, generating SQL statements for the resulting normalized tables.

1. Importing Libraries:
pandas: Library used for data manipulation and analysis, referred to as pd.
itertools.combinations: Function from itertools used for generating combinations of elements.
re: Module for working with regular expressions, helpful for string manipulation..
csv: A standard library for reading and writing CSV files, which is utilized for input handling.
normalizer: A custom module that contains functions responsible for the normalization processes.
re: A library for regular expressions, which used for parsing strings

File Structure of Input Files

1. referenceInputTable.csv: Contains the initial data set. Each column represents an attribute, and each row represents a record. The file should be formatted as a standard CSV, where values are separated by commas.
2. fds.txt: This text file lists functional dependencies in the format X -> Y, where X is a set of attributes determining Y. Each dependency should be on a new line. This input is critical for determining how the data should be decomposed into normalized forms.

Key Functionalities

1. Reading Input Files: The script uses pandas to read the CSV file into a DataFrame. This allows for easy manipulation and querying of the data. Functional dependencies are read from the `fds.txt` file, parsed, and stored in a dictionary for later use in the normalization process.
2. Input Parser:The parser processes the input table to identify any columns with comma-separated values. These are split into lists to prepare for normalization, ensuring that the data is in a suitable format for processing.
3. Multi-Valued Dependencies (MVDs):Users can interactively input MVDs, which are dependencies that indicate a relationship where one attribute can have multiple independent values. This is essential for achieving 4NF.These dependencies are stored in a structured dictionary format for processing during normalization.
4. Normal Forms Selection:Users are prompted to select the highest normal form (1NF through 5NF) that they want to achieve. This selection guides the normalization process.An additional option allows users to inquire about the highest normal form that the input table currently satisfies.
5. Primary Key Input:The script prompts the user to input primary keys for the data table. These keys can include composite keys, which are made up of multiple columns.This input is crucial for establishing uniqueness in the records and for subsequent normalization steps.
6. SQL Data Type Determination:A utility function assesses the data types of the DataFrame's columns and maps them to corresponding SQL data types (e.g., INTEGER, VARCHAR). This ensures that the SQL queries generated later are compatible with the database system.
7. SQL Query Generation:As the normalization process occurs, SQL queries are generated for each normalized relation. This includes Creating tables in SQL for each normal form achieved. Generating appropriate foreign key relationships based on user-defined FDs and MVDs.
8. Normalization Process:The code performs the normalization through the following stages: In normalizer.py file below are the functions defined

First Normal Form (1NF) Validation and Transformation:

validate_1NF(relation): Checks if a given relation (DataFrame) meets the requirements of 1NF by ensuring each column has a single, consistent data type and no nested structures.

is_nested_collection(value): Helper function to determine if a value is a collection (list, set).

transform_to_1NF(relation, pk): Transforms the relation to comply with 1NF by flattening nested structures if necessary. Returns a dictionary with the normalized relation.

Second Normal Form (2NF) Validation and Transformation:

validate_2NF(pk, fds, rel): Checks if the relation adheres to 2NF rules by ensuring non-key attributes are fully functionally dependent on the primary key.

transform_to_2NF(rel, pk, fds): Decomposes the relation into 2NF by separating out functional dependencies involving non-key attributes. Returns the transformed relation(s).

Third Normal Form (3NF) Validation and Transformation:

has_partial_dependencies(relations_dict, fds): Determines if any partial dependencies exist within the given relations and functional dependencies.

validate_3NF(relations_dict, fds): Validates whether the relation meets the criteria for 3NF.

create_new_relations(relations_dict, lhs, rhs, rel): Helper function that creates new tables by separating attributes based on partial dependencies.

transform_to_3NF(relations_dict, fds): Decomposes the relation into 3NF by resolving transitive dependencies and creating new relations as necessary.

Boyce-Codd Normal Form (BCNF) Validation and Transformation:

attribute_closure(attributes_set, functional_deps): Calculates the closure of a set of attributes based on functional dependencies to check attribute reachability within a relation.

validate_bcnf(relations_dict, pk, fds): Verifies whether each relation in the dictionary meets BCNF criteria, ensuring that non-trivial functional dependencies only occur on superkeys.

bcnf_decompose(relation, fds): Decomposes relations into BCNF by separating out any attributes that do not fully comply.

transform_to_BCNF(relations_dict, pk, fds): Uses the BCNF decomposition function to transform relations to BCNF and returns the updated dictionary of relations.

4NF Validation and Decomposition

is_in_4NF: Checks if a relation is in Fourth Normal Form (4NF) by examining multivalued dependencies (MVDs). A relation violates 4NF if it contains non-trivial MVDs that aren't also functional dependencies.

decompose_4nf: Decomposes a relation to achieve 4NF by separating attributes into separate relations based on multivalued dependencies.

transform_to_4NF: Calls decompose_4nf for relations that violate 4NF and updates the relations dictionary with the decomposed relations.

5NF Validation and Decomposition

validate_5NF: Checks for join dependencies. A relation fails 5NF if it can be decomposed into smaller relations without losing any information when joined back together.

decompose_to_5NF: Decomposes a relation to Fifth Normal Form (5NF) by identifying join dependencies that cannot be resolved in previous normal forms. It creates additional tables to ensure lossless joins while maintaining all dependencies.

9. Additional Helper Functions

- **choose_attributes**: Selects attributes for creating new tables in lossless join decomposition, used in 5NF.
- **identify_join_dependencies**: Detects join dependencies, which is essential for determining if further decomposition to 5NF is required.

For each normalization step, the script checks if the table is already in the specified normal form and prints messages indicating progress and results.

10. Highest Normal Form Reporting:

After processing, if requested by the user, the script outputs the highest normal form achieved during normalization. This is helpful for assessing the final state of the data.

11.Conclusion

This main file is a comprehensive solution for automating database normalization. By effectively transforming tables through various normal forms, it ensures that data integrity is maintained while generating SQL queries for use in relational databases. Users are empowered to customize their normalization process through interactive inputs, making this tool flexible and adaptable to different data sets.