Database Normalizer

Susmitha Jejigari [12613141], Venkata Sai Vorsu [12620770]

OVERVIEW

This project provides a Python-based tool to normalize database schemas through various normal forms. The normalization process is designed to streamline database design, reduce redundancy, and ensure data consistency by enforcing relational dependencies. The tool uses functional dependencies (FDs) and multivalued dependencies (MVDs) to break down relations iteratively up to the 4th and 5th normal forms.

CORE COMPONENTS

1. **Dependency.py**: Manages dependency handling, including the identification of functional dependencies and multivalued dependencies.
2. **Input.py**: Reads and parses input data (e.g., tables, primary keys, FDs, and MVDs) from Excel files, forming the base data for normalization.
3. **Main.py**: The central script that drives the normalization process. It invokes methods from other modules and oversees each step in the normalization.
4. **Normalization.py**: Contains the core logic for decomposing relations to various normal forms, addressing partial dependencies and multivalued attributes based on the specified normal form requirements.
5. **Relation.py**: Defines the Relation class, which represents each table, including its attributes, primary key, and dependencies.
6. **relation.xlsx**, **relationinput.xlsx**, **relationinput2.xlsx**: Excel files with the initial table data used as input for the normalization process.
7. **FD.txt** and **FD2.txt**: Text files listing the functional dependencies and multivalued dependencies for different tables.

APPROACH EXPLANATION & METHODOLOGY

1. **Input Handling**:
   - The input is parsed from Excel files and dependency files (FD.txt and FD2.txt). These files contain information on relations, primary keys, attributes, and dependency rules.

    - Input.py processes the Excel data and generates initial tables and relations with associated primary keys and dependencies.

2. **Dependency Analysis**:
    - Dependency.py parses and classifies dependencies, specifically FDs and MVDs. These dependencies serve as the basis for decomposing relations into higher normal forms.
    - Functional dependencies are used to break relations based on partial and transitive dependencies, while multivalued dependencies guide the decomposition into 4NF.

3. **Normalization Process**:
    - Normalization.py performs step-by-step decomposition. The tool uses the following normalization strategy:
    - Normalization.py checks the highest normal form and decomposes.
        - **1NF**: Eliminates repeating groups, ensuring each field contains atomic values.
        - **2NF**: Removes partial dependencies by decomposing relations where non-key attributes depend only on a part of the primary key.
        - **3NF**: Addresses transitive dependencies, ensuring that non-key attributes depend solely on the primary key.
        - **BCNF**: Extends 3NF by further decomposing if any dependencies exist that violate the uniqueness of candidate keys.
        - **4NF**: Eliminates multivalued dependencies to avoid unnecessary data duplication.
        - **5NF** (if required): Decomposes based on complex join dependencies to remove any remaining redundancy.

4. **Output**:
    - The tool outputs the decomposed relations, each in its respective normal form, based on the provided functional and multivalued dependencies. Each relation retains only its primary key and essential attributes, adhering to the user's preference to exclude foreign keys.

ASSUMPTIONS

1. **Atomicity**: Assumes that all attribute values are atomic by default and that any multivalued attributes are handled by the tool.
2. **Dependency Completeness**: All functional and multivalued dependencies are correctly defined in the input files.
3. **Primary Key**: The primary keys are defined explicitly and considered correct for each relation.

INPUT/OUTPUT

1. **Input**:
    a. Excel files with relational data, functional dependencies, and multivalued dependencies.
    b. Text files (FD.txt, FD2.txt) listing dependencies per relation.
2. **Output**:
    a. Normalized relations in their decomposed forms, displayed or saved as required.
    b. Each relation contains only primary keys and essential attributes.

TESTING AND VALIDATION

1. **Dependency Validation**:

    The tool validates that the dependencies provided in the input files conform to the schema structure before starting normalization.

2. **Normalization Tests**:

    Tests are run at each normalization level to ensure that relations comply with the required normal form:

    - **1NF Validation**: Ensures each relation has atomic attributes.
    - **2NF and 3NF Validation**: Ensures that each non-key attribute fully depends on the primary key or its subsets.
    - **BCNF Validation**: Confirms that no remaining attributes depend on non-primary key columns.
    - **4NF Validation**: Ensures no multivalued dependencies remain.
    - **5NF Validation**: Checks for and decomposes complex join dependencies.

3. **Output Verification**:

The tool cross-checks decomposed relations with input FDs and MVDs to verify that all dependencies are properly addressed in the output.

CODE DOCUMENTATION

1. **Dependency.py**
   Handles parsing, classifying, and verifying functional and multivalued dependencies for each relation.
2. **Input.py**
   Reads input from Excel files and prepares the initial table and relation data for normalization.
3. **Main.py**
   The main driver script orchestrates the entire normalization process, utilizing methods from other modules to achieve each normalization form iteratively.
4. **Normalization.py**
   The core module that decomposes relations into various normal forms by addressing dependencies, using methods to remove partial, transitive, and multivalued dependencies.
5. **Relation.py**
   Defines the Relation class, encapsulating attributes like primary keys, dependencies, and decomposition methods. It includes methods like add_name to dynamically add names to each relation based on decomposed outputs.
6. **Dependencies**
   The tool's dependencies are listed in the requirements file, which installs automatically when running the main script.

PROGRAM FLOW

1. **Input Data Collection:**
   a. main.py: Request user input for table Data and parses the data from the file given
2. **Table Object Construction:**
   a. relation.py: parsed table data is passed to the constructor in to create a relation Object
3. **Additional Data Input:**
   a. main.py, dependency.py: Requests additional user data such as functional dependencies, primary key, and multivalued functional dependencies.
   b. The relation object is updated with these user-provided values.
4. **Normalization Request:**
   a. main.py: Asks the user to specify the desired normal form for normalization.
5. **Normalization Process:**
   a. Normalizer.py Sequentially calls corresponding functions in normalizer.py to normalize the table. Returns normalized tables.

6. **Output:**
    a. Displays the normalized tables as output

HOW TO RUN:

1. Open terminal
2. Ensure that latest version of python is installed
3. Clone project into terminal
4. run 'python3 main.py'
5. enter csv file
6. run the program