**Database Normalization Project**

**Introduction**

This project focuses on automating the normalization of database tables up to various Normal Forms (1NF, 2NF, 3NF, BCNF, 4NF, 5NF). The goal is to parse input tables and their functional dependencies from an Excel file and generate normalized SQL queries. The system also supports user input to specify the desired highest level of normalization.

**Problem Statement**

Redundant data and improper dependencies in databases lead to inconsistencies. This project solves these issues through **sequential normalization up to 5NF**. The objectives include:

* Eliminating partial, transitive, and multivalued dependencies.
* Ensuring that **join dependencies** are correctly managed in 5NF.
* Outputting **SQL queries** for normalized tables to make the solution usable in real-world database systems.

**How the Code Works**

1. **Input Structure**

The input is an **Excel file** with two sheets:

* **Tables**: Contains details about each table, its attributes, primary keys, and multi-valued attributes.
* **Functional Dependencies (FDs)**: Includes both **functional dependencies** and **multivalued dependencies** relevant for higher normal forms.

Tables Sheet Example:

|  |  |  |  |
| --- | --- | --- | --- |
| **Table Name** | **Attributes** | **Primary Key** | **Multi-Valued Attributes** |
| StudentInfo | StudentID, StudentName, Age | StudentID | Hobbies |

Functional Dependencies (FDs) Sheet Example:

|  |  |  |  |
| --- | --- | --- | --- |
| **Table Name** | **LHS (Determinants)** | **RHS (Dependents)** | **FD Type** |
| StudentInfo | StudentID | StudentName, Age | Functional Dependency |
| StudentInfo | StudentID | CourseID, Hobbies | Multivalued Dependency |

1. **Terminal Inputs**

**Dynamic File Input**:  
Users can provide either:

* + The **full path** to the Excel file if it is outside the current working directory.
  + The **file name** if the Excel file is in the same folder as the script.

**Specify the Highest Normal Form**:  
Users are prompted to enter the highest normal form (e.g., 1NF, 2NF, 3NF, BCNF, 4NF, or 5NF) up to which the normalization should be performed.

The code will **normalize** the input data only up to the specified form.

1. **Key Functions and Their Roles**

**a. parse\_tables\_and\_fds()**

* Reads the Excel file and extracts table details and dependencies.
* Returns dictionaries of tables and FDs for further processing.

**b. normalize\_to\_1nf()**

* First Normal Form (1NF):
* Handles multi-valued attributes by creating new relations.

**c. normalize\_to\_2nf()**

* Second Normal Form (2NF):
* Removes partial dependencies.

**d. normalize\_to\_3nf()**

* Third Normal Form (3NF):
* Eliminates transitive dependencies.

**e. normalize\_to\_bcnf()**

* BCNF (Boyce-Codd Normal Form):
* Ensures every FD has a superkey on the LHS.

**f. normalize\_to\_4nf()**

* Fourth Normal Form (4NF):
* Handles multivalued dependencies by decomposing tables if necessary.

**g. normalize\_to\_5nf()**

* Fifth Normal Form (5NF):
* Decomposes tables to eliminate join dependencies (if a relation can only be decomposed by satisfying the join property).

**h. generate\_sql\_queries()**

* Creates SQL queries for the final normalized tables.

**i. save\_queries\_to\_file()**

* Saves SQL queries to a .sql file for further use.

1. **Full Workflow**

**User Input**:

* + User provides the **Excel file path or name**.
  + User specifies the **highest normal form** to which normalization should be performed.

**Parsing Input**: Code extracts tables and dependencies from the Excel sheets.

**Normalization Sequence**:  
Based on the user’s input, the code applies normalization up to the selected highest normal form:

* + 1NF → 2NF → 3NF → BCNF → 4NF → 5NF

**SQL Generation**: SQL CREATE TABLE queries are generated.

**Saving Queries**: Queries are printed on the console and saved to a normalized\_tables.sql file.

1. **How to Run the Code**

**Prepare the Input File**: Ensure the input Excel file is formatted with two sheets: Tables and Functional Dependencies.

**Execute the Script**:

python PPDbms-5NF-Final.py

**Provide Inputs**:

* + Enter the **file path or name** when prompted.
  + Enter the **highest normal form** to be applied.

1. **Error Handling**

* **File Not Found**: If the file path is incorrect, the program will ask the user to re-enter the path or name.
* **Invalid Normal Form Input**: If the user enters an incorrect form (e.g., "7NF"), the program will display an error message and prompt the user to enter a valid option.
* **Missing Data**: If any required fields are missing in the input sheets, the code will raise a parsing error with appropriate feedback.

**Conclusion**

This project provides a **comprehensive solution** for automating normalization up to **5NF**. The **Python-based approach** ensures that:

* Redundancy and unnecessary dependencies are eliminated.
* The tables are in the highest possible normal form.
* SQL queries are readily available for use in a real database.

This tool is especially useful for **database designers and students** looking to understand normalization techniques or apply them in practical settings.