Traffic Data Inquiry System Report

YIN, Yin; KONG, Mingwei; LIU, Liangjie

December 13, 2024

1 Introduction

The project aims to design and implement an inquiry system interface for traffic data. With the increasing amount of data generated by traffic management systems, having an efficient and user-friendly interface to query and manipulate this data is crucial for transportation authorities and researchers. The goal of this system is to enable users to perform tasks such as searching traffic data, sorting it by various criteria, combining data from multiple sheets, and importing or exporting data in a variety of formats.

This report outlines the design process, the key functionalities implemented, the system architecture (UML), and examples of how to use the system. The project utilizes a combination of user interface (UI) and backend code to ensure smooth interaction with the traffic data.

2 Pre-design

2.1 Requirement Analysis

The system must meet the following requirements:

- **User Interface**: The system should offer either a command-line interface (CLI) or a graphical user interface (GUI), or both.
- **Core Functionalities**: The system should provide at least five key functionalities:
  - **Search**: Search traffic data based on certain criteria.
  - **Sort**: Sort traffic data by various fields (e.g., date, location).
  - **Join**: Combine multiple traffic data sheets into one for unified analysis.
  - **Import**: Import traffic data from external files.
  - **Export**: Export traffic data into different formats (e.g., CSV, Excel).

The system will also require a UML diagram to describe the relationship between the user interface, algorithms, and file handling modules.

2.2 Feature List

Feature	Description
Search	Allows users to search for specific traffic data based on filters (e.g., location, date).
Sort	Enables users to sort traffic data by specified fields.

Join	Combines multiple traffic data sheets into one unified dataset.
Import	Allows the system to load traffic data from various file formats.
Export	Provides functionality to save traffic data in different formats.

2.3 UI Design

The design of the user interface will be a simple and intuitive layout, featuring the following elements:

- **Input Fields**: For entering search criteria.
- **Buttons**: For sorting, importing, and exporting data.
- **Data Tables**: To display traffic data, with sorting options on columns.
- **Status Indicators**: To show whether the import or export actions were successful.

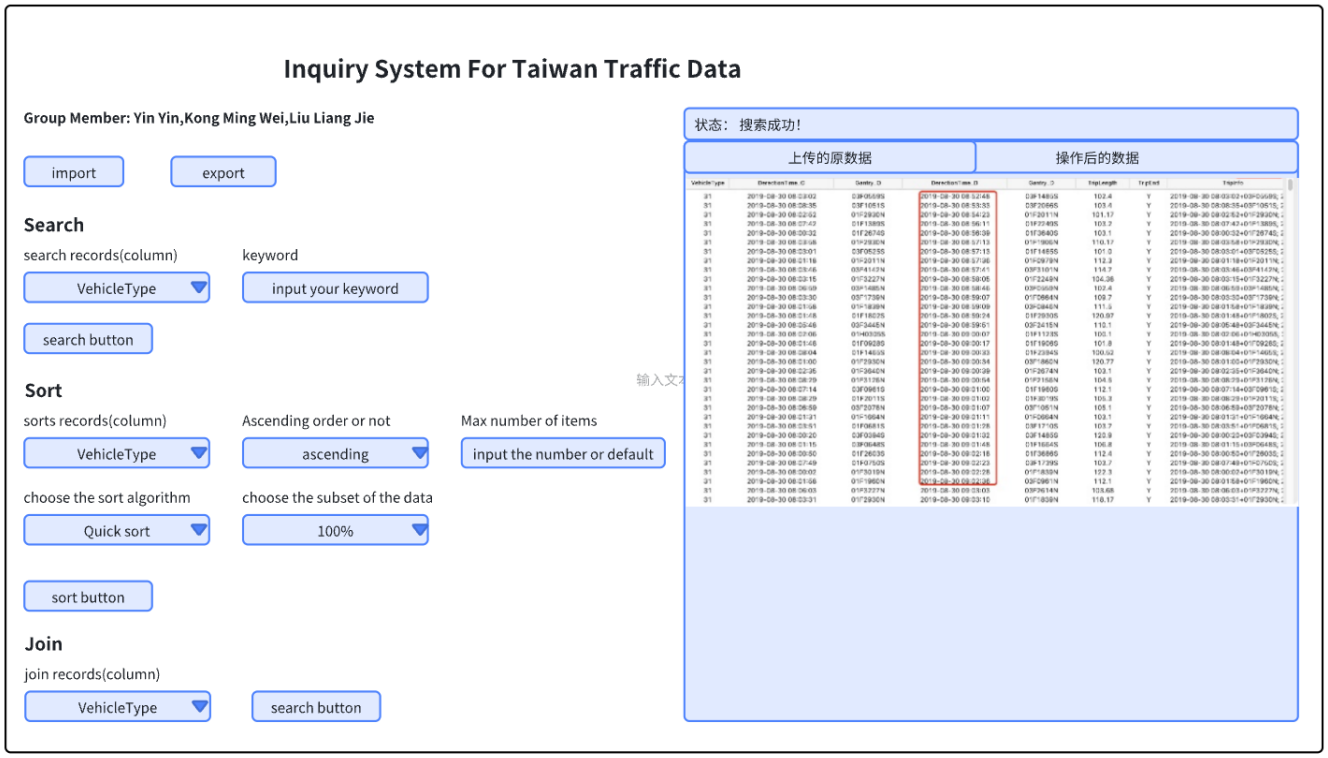

Figure 1: UI Design Sketch

3 UML Design

3.1 UML Diagram Overview

The UML diagram illustrates the system's three-layer architecture: **GUI Layer**, **Business Layer**, and **Data Layer**. Each layer has distinct responsibilities and interacts through well-defined interfaces to ensure modularity and extensibility.

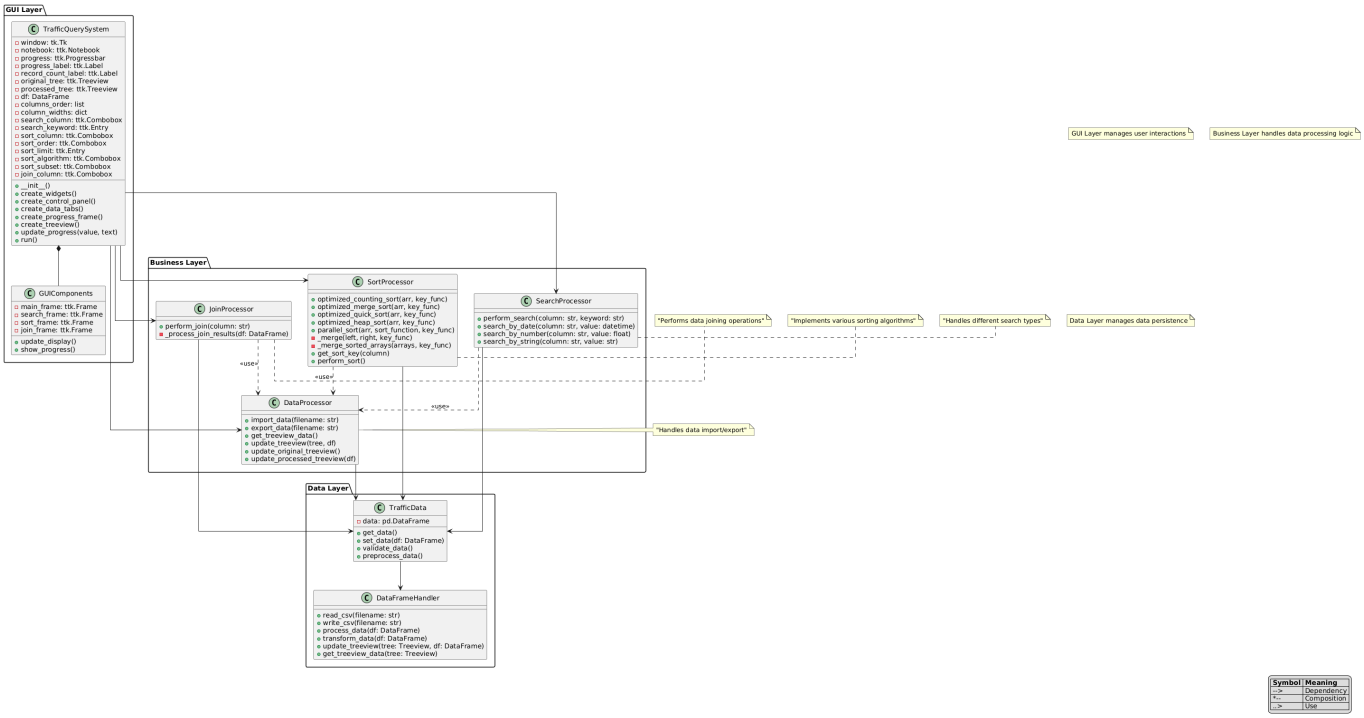

Figure 2: UML Diagram of System Architecture

3.2 Layer Breakdown

3.2.1 GUI Layer

- **TrafficQuerySystem**:
  - **Attributes**:
    - `window`: The main application window.
    - `notebook`: A tabbed interface for organizing functionalities.
    - `progress`: Progress bar for task status.
    - `progress_label`: Label to display progress text.
    - `record_count_label`: Label to show the total number of records.
    - `original_tree`: Treeview to display the original dataset.
    - `processed_tree`: Treeview to display the processed dataset.
    - `df`: The dataset currently loaded into the application.
    - `columns_order`: A list of column names in the preferred order.
    - `column_widths`: Dictionary mapping columns to their display widths.
    - `search_column`: Column used for search operations, selected by the user.
    - `search_keyword`: Input box for entering search keywords.
    - `sort_column`: Dropdown for selecting the sorting column.
    - `sort_order`: Dropdown for selecting ascending or descending sort order.
    - `sort_limit`: Input box for specifying the maximum number of records to sort.
    - `sort_algorithm`: Dropdown for selecting the sorting algorithm.
    - `sort_subset`: Dropdown for specifying the subset of data to sort.
    - `join_column`: Dropdown for selecting the column to join datasets.
  - **Methods**:
    - `create_widgets()`: Initializes all window widgets and their layouts.

    * `create_control_panel()`: Sets up the control panel for user actions (e.g., buttons, dropdowns).
    * `create_data_tabs()`: Configures the tabbed interface for data display.
    * `create_progress_frame()`: Sets up the progress bar and label for task status.
    * `create_treeview()`: Initializes and configures the table views for data.
    * `update_progress(value, text)`: Updates the progress bar with a specific value and message.
    * `run()`: Launches the main application event loop.

- **GUIComponents**:
  - **Attributes**:
    * `main_frame`: The main frame of the GUI.
    * `search_frame`: Frame containing the search controls.
    * `sort_frame`: Frame containing the sort controls.
    * `join_frame`: Frame containing the join controls.
  - **Methods**:
    * `update_display()`: Refreshes the displayed data in the GUI.
    * `show_progress()`: Updates the progress bar and displays task status.

3.2.2 Business Layer

- **JoinProcessor**:
  - **Methods**:
    * `perform_join(column: str)`: Executes the join operation on datasets based on the specified column.
    * `_process_join_results(df: DataFrame)`: Cleans and validates the results of the join operation.

- **SortProcessor**:
  - **Methods**:
    * `optimized_counting_sort(arr, key_func)`: Performs counting sort on the dataset.
    * `optimized_merge_sort(arr, key_func)`: Performs merge sort on the dataset.
    * `optimized_quick_sort(arr, key_func)`: Performs quicksort on the dataset.
    * `optimized_heap_sort(arr, key_func)`: Performs heapsort on the dataset.
    * `parallel_sort(arr, sort_function, key_func)`: Executes a parallelized sorting algorithm.
    * `_merge(left, right, key_func)`: Merges two sorted arrays.
    * `_merge_sorted_arrays(arrays, key_func)`: Merges multiple sorted arrays into a single sorted array.
    * `get_sort_key(column)`: Retrieves the sorting key for the specified column.
    * `perform_sort()`: Executes the sorting operation using the chosen algorithm.

- **SearchProcessor**:
  - **Methods**:
    * `perform_search(column: str, keyword: str)`: Performs a general search operation based on the specified column and keyword.

    * `search_by_date(column: str, value: datetime)`: Filters data by a specific date.
    * `search_by_number(column: str, value: float)`: Filters data by a numeric value.
    * `search_by_string(column: str, value: str)`: Filters data by a string value.

- **DataProcessor**:
  - **Methods**:
    * `import_data(filename: str)`: Loads data from the specified file into the application.
    * `export_data(filename: str)`: Saves the processed data to a specified file.
    * `get_treeview_data()`: Retrieves data in a format suitable for the treeview component.
    * `update_treeview(tree, df)`: Updates the GUI treeview component with the specified DataFrame.
    * `update_original_treeview()` Updates the treeview with the original dataset.
    * `update_processed_treeview(df)`: Updates the treeview with processed data.

3.2.3 Data Layer

- **TrafficData**:
  - **Attributes**:
    * `data: pd.DataFrame`: Core data stored as a Pandas DataFrame.
  - **Methods**:
    * `get_data()`: Retrieves the current dataset.
    * `set_data(data: pd.DataFrame)`: Updates the stored dataset with the given DataFrame.
    * `validate_data()`: Validates the data's integrity and structure.
    * `preprocess_data()`: Cleans and prepares the data for use in processing or analysis.

- **DataFrameHandler**:
  - **Methods**:
    * `read_csv(filename: str)`: Reads data from a CSV file into a DataFrame.
    * `write_csv(filename: str)`: Writes a DataFrame to a CSV file.
    * `process_data(df: DataFrame)`: Performs processing operations on a DataFrame.
    * `transform_data(df: DataFrame)`: Transforms a DataFrame for specific use cases or visualization.
    * `update_treeview(tree: Treeview, df: DataFrame)`: Updates the specified treeview with data from a DataFrame.
    * `get_treeview_data(tree: Treeview)`: Retrieves data from a treeview in a format suitable for further processing.

3.3 Interactions Between Layers

1. **GUI Layer and Business Layer Interaction**:
   - User interactions in the GUI Layer, such as button clicks or dropdown selections, invoke specific methods in the Business Layer.
   - The Business Layer processes the user request (e.g., search, sort, or join operations) and returns the results to the GUI Layer for display.

- For example, clicking the "Search" button triggers `SearchProcessor.perform_search`, which filters data and updates the GUI treeview.

2. **Business Layer and Data Layer Interaction**:
   - The Business Layer accesses the Data Layer to retrieve, validate, and preprocess data for operations such as search, sort, or join.
   - The Data Layer handles file-based persistence (e.g., reading from or writing to CSV files) and transforms data for use in the Business Layer.
   - For instance, importing data via `DataProcessor.import_data` invokes `DataFrameHandler.read_c` in the Data Layer.
3. **Data Flow Examples**:
   - **Search Functionality**:
     (a) The user specifies search criteria in the GUI.
     (b) `SearchProcessor.perform_search` filters the data.
     (c) The filtered results are returned from the Data Layer and displayed in the GUI.
   - **Sorting Functionality**:
     (a) The user selects a column and sorting order in the GUI.
     (b) `SortProcessor.perform_sort` applies the selected sorting algorithm.
     (c) The sorted data is sent back to the GUI and displayed.
   - **Join Functionality**:
     (a) The user selects a column for joining datasets in the GUI.
     (b) `JoinProcessor.perform_join` merges the datasets.
     (c) The joined data is validated and displayed in the GUI.
   - **Import/Export Functionality**:
     (a) Importing a file invokes `DataFrameHandler.read_csv`, which loads the data for processing.
     (b) Exporting a file invokes `DataFrameHandler.write_csv`, saving the processed data to the specified location.

4 Run Example

4.1 Import Functionality

Code Explanation: The **Import** function enables the system to load traffic data from external files. The code uses the `Importer` module to read various formats (e.g., CSV or Excel) and store the data into the system for further processing.

Operation:

1. The user selects the **Import** button.
2. A file dialog opens for the user to choose the file.
3. Once the file is selected, the system processes it and loads the data into the application.

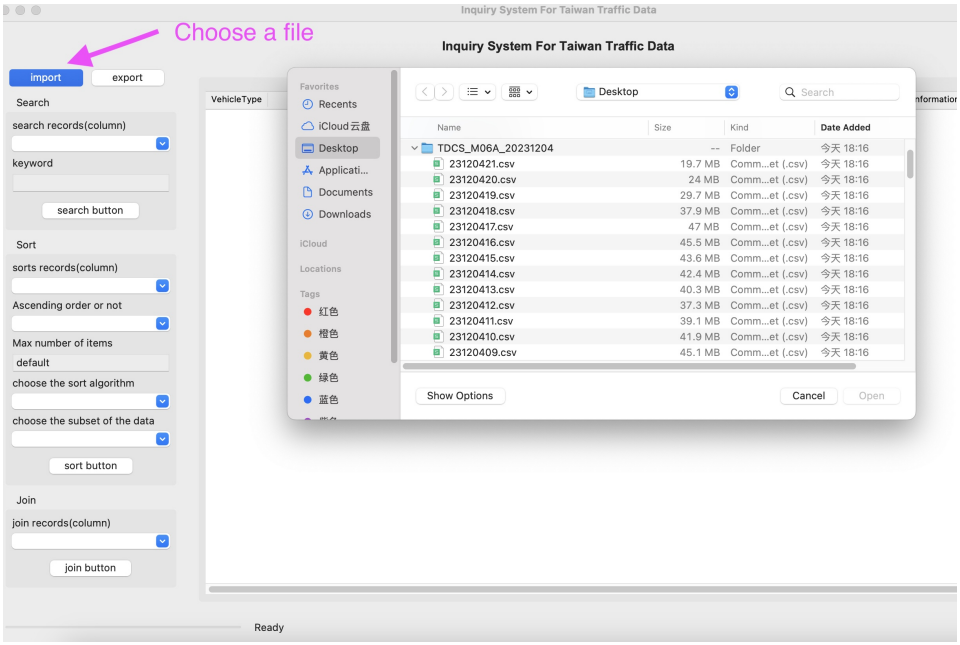

Figure 3: Import Example: File Selection

4.2 Search Functionality

Code Explanation: The **Search** functionality uses the `SearchEngine` to filter traffic data based on user-provided criteria. The search results are then displayed in a table format.

Operation:

1. The user enters search parameters in the input fields (e.g., date range, location).
2. Clicking the **Search** button triggers the search logic.
3. The system filters the dataset based on the input and displays the matching entries.

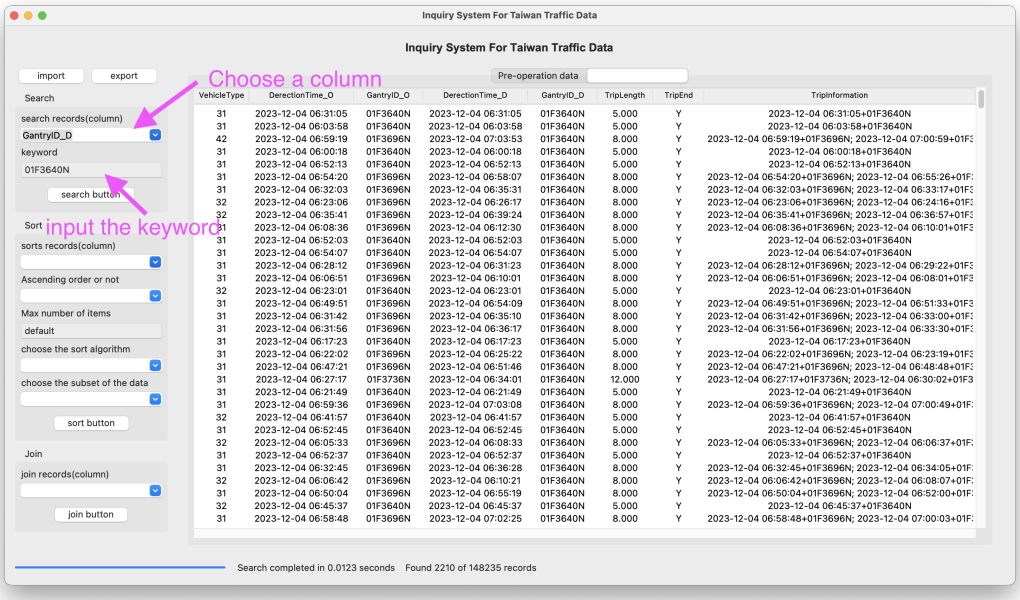

Figure 4: Search Example: Search Results

4.3 Sort Functionality

Code Explanation: The **Sort** function utilizes the `SortEngine` to arrange traffic data based on a selected column (e.g., sorting by date, location).

Operation:

1. The user selects the column to sort by.
2. Clicking the **Sort** button triggers the sorting logic.
3. The data table is updated to reflect the sorted order.

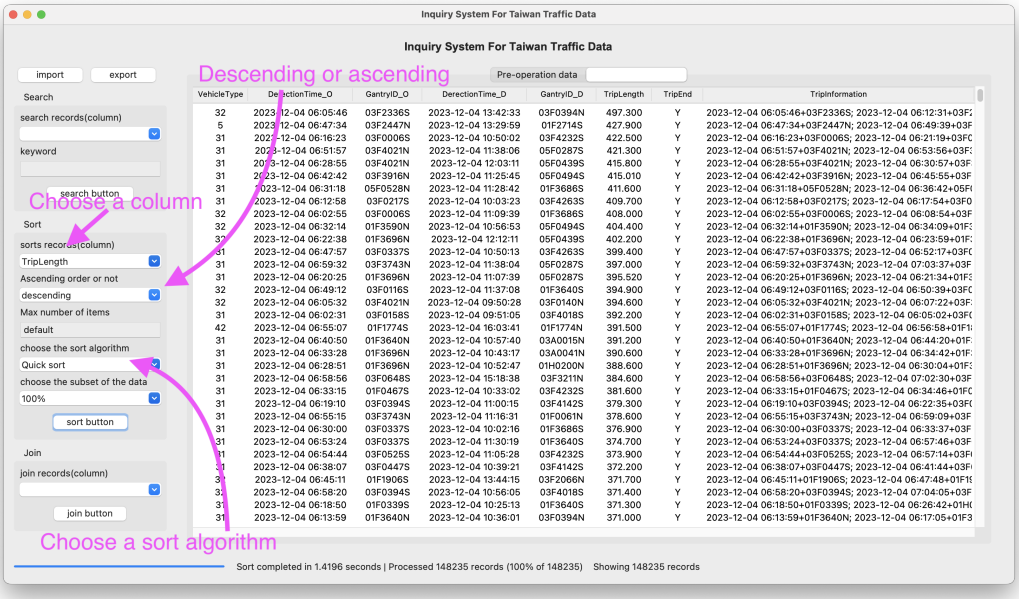

Figure 5: Sort Example: Sorted Data Table

4.4 Join Functionality

Code Explanation: The **Join** function combines data from multiple sheets into one unified dataset using the `JoinEngine`.

Operation:

1. The user selects multiple datasets to join.
2. Clicking the **Join** button merges the data and removes any duplicates.
3. The joined dataset is displayed in the table.

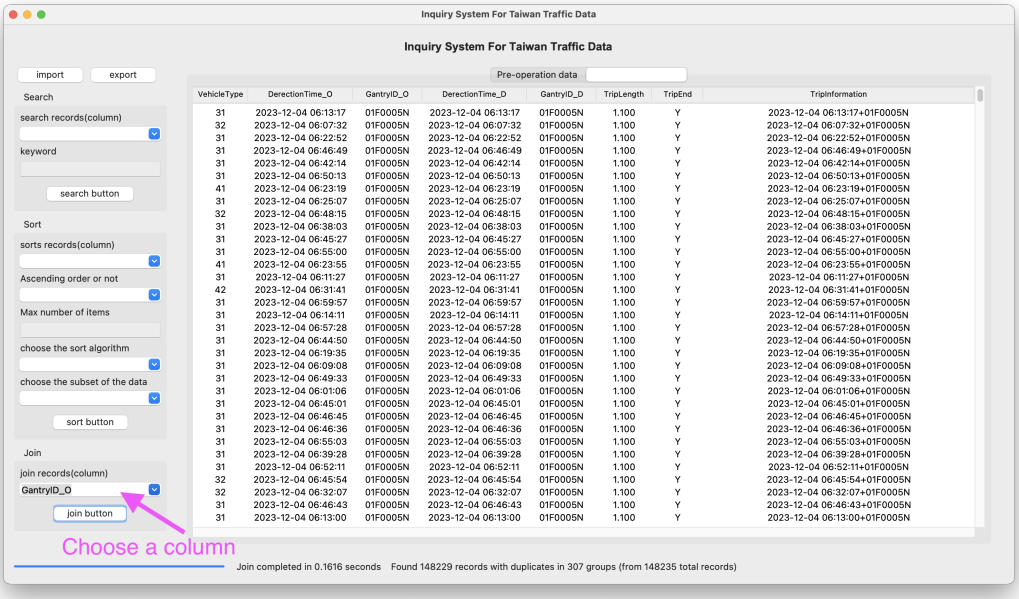

Figure 6: Join Example: Before and After Join

4.5 Export Functionality

Code Explanation: The **Export** function enables the user to save the traffic data in various formats (e.g., CSV, Excel) using the `Exporter` module.

Operation:

1. The user clicks the **Export** button.
2. A file dialog appears to choose the destination and format for saving.
3. The system exports the current data to the selected file.

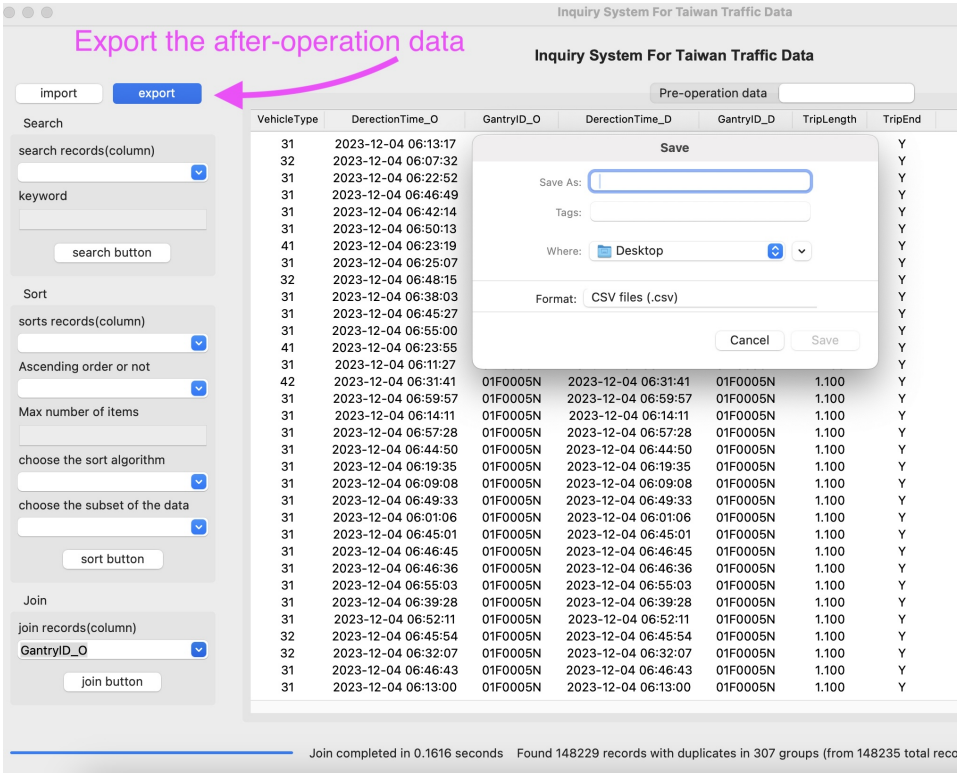

Figure 7: Export Example: File Export Dialog