




PROJECT AND TEAM INFORMATION

Project Title

Query Language Compiler for CSV/JSON Data

Student / Team Information

<p><i>Team Name:</i> <i>Team #</i></p>	<p><i>The Parsers</i></p>
<p>Team member 1 (Team Lead) (Kainthola, Vaishnavi: 220122052: vkainthola206@gmail.com):</p>	<p>Kainthola, Vaishnavi: 9871539456 vkainthola206@gmail.com</p> 
<p>Team member 2 (Dhyani, Harshit: 220111613: harshitdhyani01@gmail.com):</p>	<p>Dhyani, Harshit: 7217487085 harshitdhyani01@gmail.com</p> 
<p>Team member 3 (Singh, Parikshit: 220112091: parikshitsingh683@gmail.com):</p>	<p>Singh, Parikshit: 9045491275 parikshitsingh683@gmail.com</p> 

Team member 4

(Chhetri, Pranjali: 220111506: pranjali.chhetri92@gmail.com):

Chhetri, Pranjali: 9068881771

pranjali.chhetri92@gmail.com



PROPOSAL DESCRIPTION (10 pts)

Motivation (1 pt)

In today's data-driven world, working with structured data stored in **CSV and JSON files** is a common task for analysts, researchers, and developers. However, many users struggle with writing Python scripts to filter, transform, and analyze such data efficiently. Although SQL is a popular choice for querying relational databases, there is no straightforward means for non-technical users to execute SQL-like queries on CSV/JSON files.

Currently, users must write **complex Python scripts** using **Pandas** or **JSON parsers** to extract insights from their data. This can be time-consuming, error-prone, and inefficient for those unfamiliar with programming. Additionally, large datasets require **optimized query execution**, which is difficult to achieve manually.

This project aims to develop a **Query Language Compiler** that allows users to write **SQL-like queries** to interact with CSV and JSON files **without needing to write Python code**. The compiler will parse user queries, translate them into optimized **Pandas-based Python scripts**, and execute them seamlessly.

Why is this Important?

1. **Accessibility** – Enables users with little or no programming experience to query CSV/JSON files easily.
2. **Efficiency** – Reduces manual effort in writing scripts, making data processing faster and more intuitive.
3. **Automation** – Helps automate repetitive data processing tasks with structured queries.
4. **Data Exploration** – Allows quick filtering, sorting, and aggregation of data for analysis.

State of the Art / Current solution (1 pt)

Currently, querying and processing **CSV/JSON files** is done using **programming languages** like Python, R, or JavaScript. The most common approach involves using libraries such as:

1. **Pandas (Python)** – Provides functions like `read_csv()`, `query()`, and `groupby()` to filter and analyze data.
2. **SQL-on-CSV Tools** – Some libraries like **DuckDB**, **SQLite (with CSV import)**, and **CSVQL** allow SQL-like queries on CSV files.
3. **JSON Processing Libraries** – Tools like `jq` (command-line JSON processor) and `pandas.read_json()` help manipulate JSON data.
4. **Big Data Solutions** – **Apache Spark**, **Dask**, and **Presto** enable querying large datasets efficiently but require setup and programming knowledge.

Limitations of Current Solutions

- **Requires Coding Skills** – Users must learn Python, SQL, or command-line tools.
- **Not User-Friendly** – Writing scripts manually is time-consuming and error-prone.
- **Limited for Large JSON Data** – Handling nested JSON structures is complex without advanced knowledge.

Project Goals and Milestones (2 pts)

Project Objectives

The main objective of this project is to create a Query Language Compiler that allows users to write SQL-like queries to manipulate CSV and JSON files without the need for programming skills. The compiler will:

- Offer a simple query interface for non-programmers.
- Convert SQL-like queries into Python code with Pandas and JSON processing libraries.
- Support basic SQL operations like SELECT, WHERE, ORDER BY, and LIMIT.
- Allow efficient query execution for big data.
- Offer a command-line (CLI) or web-based interface for easy access.

Milestone 1: Research & Planning

Specify project scope and requirements.

Examine the available tools such as Pandas, SQL-on-CSV tools, and JSON processors.

Milestone 2: Lexical & Syntax Analysis

Add a tokenizer to identify SQL-like keywords (SELECT, WHERE, etc.).

Create a parser to parse query structure.

Milestone 3: Semantic Analysis & Code Generation

Check column names and conditions.

Translate queries to equivalent Python code using Pandas.

Milestone 4: Query Execution & Optimization

Run Python scripts generated.

Tune performance on big data sets.

Milestone 5: User Interface Development

Implement CLI tool for executing queries.

Implement basic web interface.

Milestone 6: Testing & Documentation

Implement unit testing and performance optimization.

Implement user documentation and wrap up the project.

Project Approach (3 pts)

Solution Design

The project employs a compiler-based methodology to convert SQL-like queries into Python scripts that efficiently handle CSV/JSON files. The solution will comprise the following elements:

Lexical Analysis (Tokenizer) – Recognizes SQL-like keywords (SELECT, WHERE, etc.) and translates them into tokens.

Syntax Analysis (Parser) – Applies a parsing method (LL or LR parsing) to check query structure.

Semantic Analysis – Verifies column names are present and checks for logical errors.

Code Generation – Converts parsed queries into equivalent Python (Pandas) code for execution.

Query Execution & Optimization – Executes generated code and optimizes performance on big data.

User Interface (CLI/Web App) – Offers a simple-to-use CLI for command-line execution and a web-based interface for non-technical users.

Technologies & Platforms

Programming Language: Python (for compiler rules and execution).

Parsing & Tokenization: PLY (Python Lex-Yacc) or custom parser.

Data Processing: Pandas (for CSV), json module (for JSON).

Command-Line Interface: argparse (for CLI support).

Optional Web UI: Flask or FastAPI (to create a web-based query tool).

Database Alternative (if necessary): DuckDB (for SQL-like queries on CSV).

Implementation Plan

Prototype Development – Create a minimal SQL-to-Pandas converter.

Incremental Features – Implement JSON support, query optimizations, and a UI.

Testing & Deployment – Test for performance and usability before release.

This method guarantees a scalable, effective, and user-friendly means of querying structured data without programming.

System Architecture (High Level Diagram) (2 pts)

Main Components:

User Interface (CLI / Web UI)

- Accepts SQL-like queries from users.
- Delivers query execution results.

Query Compiler

- Lexical Analyzer: Tokenizes SQL-like queries.
- Syntax Analyzer: Checks query structure.
- Semantic Analyzer: Checks correctness (column existence, types).
- Code Generator: Translates queries to Python (Pandas/JSON code).

Query Execution Engine

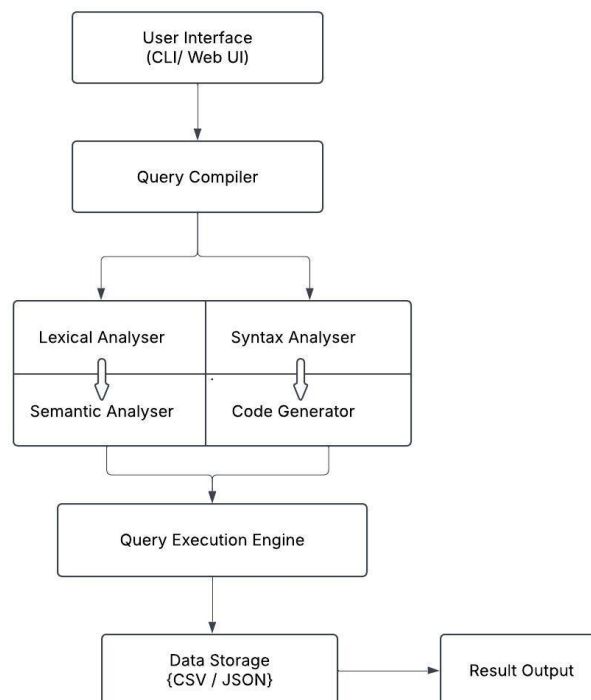
- Runs the generated Python script.
- Loads and processes data from CSV/JSON files.
- Optimizes query execution for performance.

Data Storage

- CSV / JSON files (structured data source).

Result Output

- Shows query results in the form of tables (CLI) or structured representation (Web UI).



Project Outcome / Deliverables (1 pts)

The outcome of the project will be a Query Language Compiler which enables users to run SQL-like queries against CSV and JSON data without a full-fledged database. The most important deliverables are:

A Complete Compiler

- Reads SQL-like queries and compiles them into Python code (with Pandas being used to handle CSV and JSON data).

Command-Line Interface (CLI)

- A CLI application with an easy-to-use interface to run queries and output results in a formatted way.

Web-Based Interface

- A web UI (implemented with Flask or FastAPI) for users without programming skills to run queries and view results.

Query Execution Engine

- Designed to support large CSV/JSON files efficiently.

Detailed Documentation

- Instruction on how to install, use, and extend the compiler.

GitHub Repository

- Source code, installation instructions, and test data sets.

By providing these pieces, the project will offer a lightweight, database-independent solution for querying structured data, which will be useful to data analysts, developers, and researchers.

Assumptions

Assumptions

1. **Structured Data** – The input CSV and JSON files are **well-formed** and follow a structured format without corruption.
2. **Standard SQL-Like Queries** – The query syntax follows a simplified **SQL-like** structure (e.g., SELECT column FROM file WHERE condition).
3. **Column Names Exist** – Queries reference **valid column names** in the dataset.
4. **Moderate File Size** – The system is optimized for handling **medium-sized** files (not massive, big-data scale).
5. **Python Environment** – Users have **Python installed**, along with required dependencies (e.g., Pandas).

References

- **Pandas Documentation** – <https://pandas.pydata.org/docs/>
- **SQLite & SQL Syntax** – <https://www.sqlite.org/lang.html>
- **ANTLR (Parser Generator)** – <https://pragprog.com/titles/tpantlr2/the-definitive-antlr-4-reference/>
- **FastAPI Documentation** – <https://fastapi.tiangolo.com/>
- **GeeksforGeeks** – <https://www.geeksforgeeks.org/compiler-design-tutorials/>
- **Stackoverflow** – <https://stackoverflow.com/questions/76953964/run-sql-query-on-csv-file-contents-from-command-line>