Databases Class. 2025-10

Final Project: FullDB - Custom SQL Engine with Front- and Backend. Alternative # 3

In this project, you will have the opportunity to design and develop a Minimum Viable Product (MVP) that demonstrates the use of an SQL-based engine with both frontend and backend components. The project emphasizes your ability to apply database knowledge, implement data persistence, and develop full-stack systems. This project will be deployed in Openlabs@uninorte for future continuous work by other database classes. What you need to deliver at the end of your deadline is 1) a working MVP, 2) a written report about the project, and 3) a short presentation of your project.

1. General Requirements

Your MVP must meet the following specifications:

- Your project must be a web application with both frontend and backend components.
- The frontend should be built using React or another modern framework.
- The backend must support SQL parsing and execution, with persistent data storage.
- You may choose any programming language and libraries for the backend (e.g., Node.js, Python, Go), but you must develop your own integration and logic.
- You must use existing SQL parser libraries or engines (e.g., SQLite or PostgreSQL parsers, sql.js, alasql, node-sql-parser). A SQL parser does things such as: a) validate the syntax b) translate the SQL to another SQL dialect c) create an Abstract Syntax Tree (AST) or a JSON object of the query
- The system should allow users to:
    - Load structured data (e.g., from CSV or user-defined tables). You must offer loading a database from a CSV
    - Execute SQL commands via a web interface.
    - Persist the data between sessions (e.g., via a file system on the backend, you might get support from some libraries but NOT use a full system like an existing database).

2. MVP Requirements

a) **Frontend (Client Side)**
  - Built with React (or similar framework).
  - Includes:
    - Input for SQL queries (Supported queries: CRUD of tables and database).
    - Table or chart display of results.
    - File upload or URL input for structured data (optional).
    - Error handling for syntax issues or execution failures.

b) **Backend (Server Side)**
  - Must be deployed using Docker for easy portability to OpenLab servers.
  - Implements a REST API or similar interface.
  - Receives SQL instructions from the frontend.

    - Uses a library or engine to parse and execute the SQL.
    - Saves data persistently (e.g., using a specialized file system)
    - May expose endpoints for:
        - Database and Table creation and deletion.
        - Data insertion, selection, updates, and deletion.
        - Query execution and result retrieval.
    - (Optional): implements cache for query results
    - (Optional): implements a query optimizer and/or planner
    - (Optional): Implements the whole lifecycle of a query, or stages of it. See here[1]

c) **Persistence Layer**
    - Data should be saved regardless if the backend process is killed.
    - This can NOT be implemented via a database engine, it must be a custom file storage system or a combination of helping libraries (please check with professor).

3. Written Report of the Project

The report must contain the following sections:

- Explanation of the Problem: Define the problem this project addresses.
- Technology Stack: Describe the tools and libraries used (React, parser, CSV handler, etc.).
- Architecture Description: Component structure and flow of data.
- Screenshots and Examples: Illustrate use cases.
- Challenges and Solutions: Explain problems faced during development and how they were solved.

4. Short Presentation

You will have 7 minutes on **June 5th (Week 18)** to present your application. The presentation should cover: a) The problem and your solution, b) Live demo or video of the tool in use, c) Summary of technologies and main features.

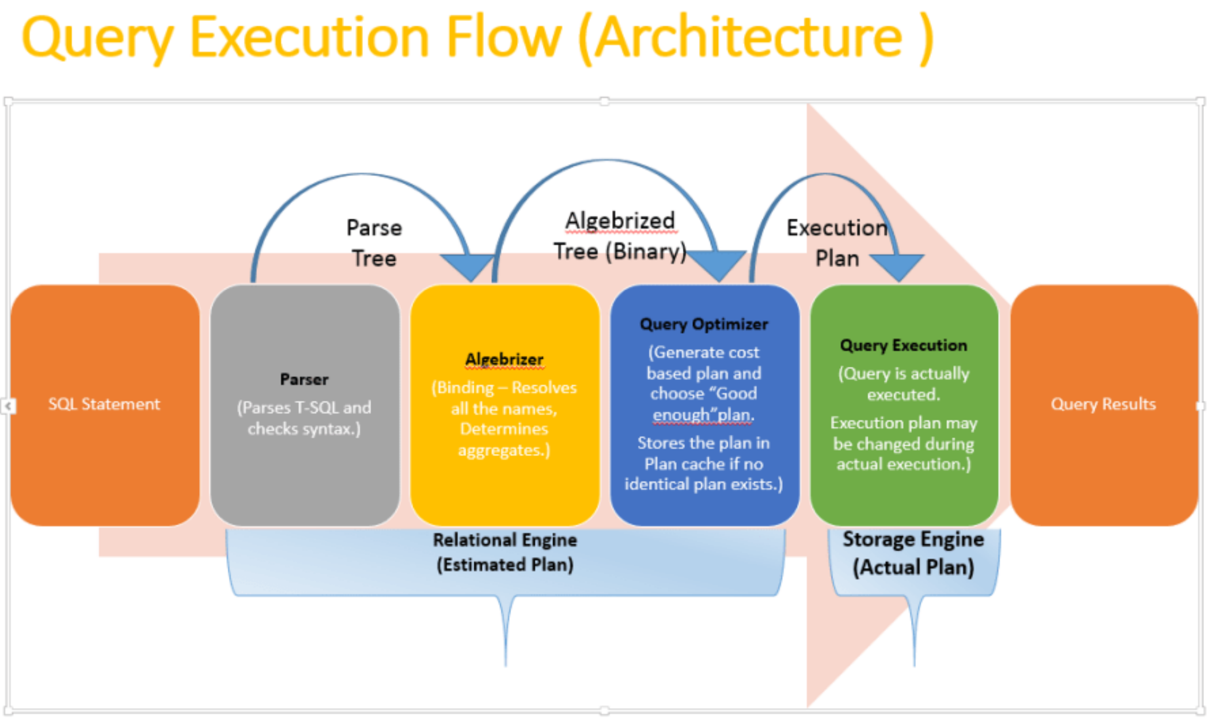
Grading Breakdown

- The project deadline is divided into two:
    - Deliver the report: **The day of the presentation, 11.59 PM**
    - Presentation of your MVP: **June 5th**.
- Grading is divided as follows:
    - MVP: 70% (45% back, 25% front)
    - Report: 20%
    - Presentation: 10%. For this, a judge will be invited (Me, and Myself), who will grade your presentation according to the topics needed for the report.
- Important note: Only two projects will be allowed to present this Alternative # 3. By the beginning of week 15 (on Monday), the groups might present the number of functionalities they

[1] https://sqlrelease.com/query-execution-flow-architecture-sql-server?amp

WILL implement. The two with MOST functionalities will do this project and it will count as Final Exam and Final Project (30% of the class). The remaining groups will do one of the other two alternatives.

Anexos


Query Execution Flow (Architecture)
Parse Tree
Algebrized Tree (Binary)
Execution Plan
SQL Statement
Parser
(Parses T-SQL and checks syntax.)
Algebrizer
(Binding – Resolves all the names, Determines aggregates.)
Query Optimizer
(Generate cost based plan and choose "Good enough"plan.
Stores the plan in Plan cache if no identical plan exists.)
Query Execution
(Query is actually executed.
Execution plan may be changed during actual execution.)
Query Results
Relational Engine (Estimated Plan)
Storage Engine (Actual Plan)

Fuente: https://sqlrelease.com/query-execution-flow-architecture-sql-server?amp

Referencias

Parsers para Python: https://medium.com/@elizapan/open-source-python-sql-parsers-8dcfaf0c896a

SQL Parser: https://github.com/tobymao/sqlglot

Query Execution flow: https://sqlrelease.com/query-execution-flow-architecture-sql-server?amp

Query processing architecture: https://learn.microsoft.com/en-us/sql/relational-databases/query-processing-architecture-guide?view=sql-server-ver16

Online query optimizer: https://www.explo.co/sql-tools/ai-sql-optimizer