CoreDB: A Custom Database Management System

Arnav Sharda (123cs0064) Rohit Chauhan (123cs0054)

Department of Computer Science IIITDM Kurnool

November 5, 2025

Outline

Project Overview

- **CoreDB** is a lightweight, fully functional relational DBMS built entirely from scratch.
- Designed to provide a **transparent understanding of DB internals** — from parsing to storage.
- Includes a full SQL engine, REST API, and web-based frontend.
- Emphasizes modularity, debuggability, and extensibility.

Core Objectives

- Implement a complete SQL pipeline: `Lexer` → `Parser` → `Executor` → `Storage`.
- Ensure transparency in query execution.
- Deliver an intuitive, browser-based learning interface.

Motivation

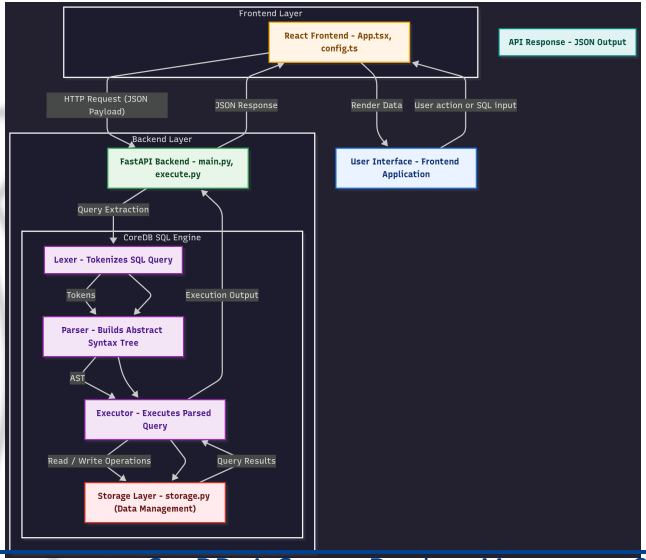
Challenges

- Most DBMSs are opaque "black boxes".
- Students rarely explore internal SQL execution.
- Limited educational DBMS tools for learning internals.

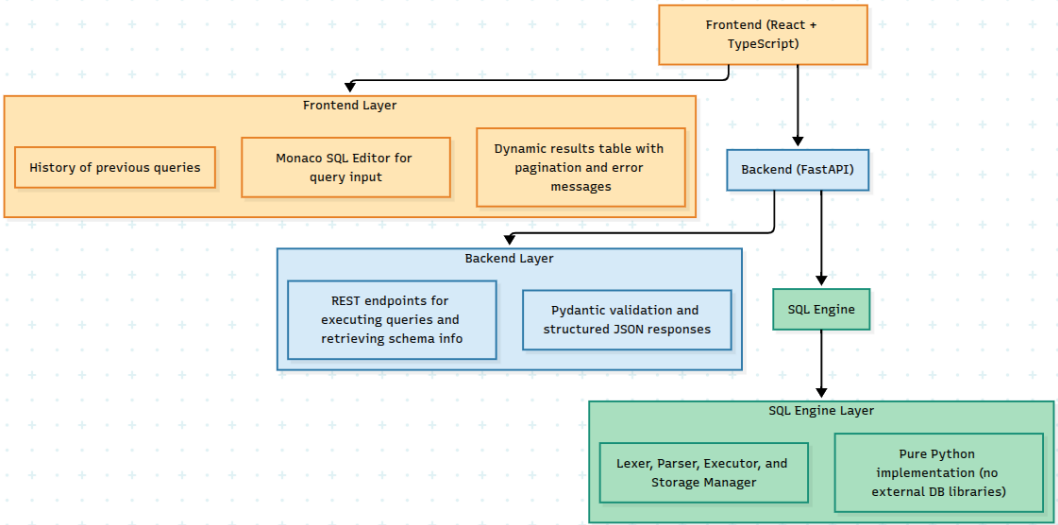
Our Approach

- Build a DBMS from first principles.
- Design modular components for isolated testing.
- Expose internal flow from query parsing to storage.

High-Level Architecture



System Architecture Layers



Query Processing Pipeline

1. **Lexer:** Tokenizes SQL input into keywords and identifiers.
2. **Parser:** Builds Abstract Syntax Tree (AST) from tokens.
3. **Executor:** Traverses AST to perform logical operations.
4. **Storage:** Persists structured data as JSON.
5. **API Layer:** Returns JSON responses to the frontend.

Key Idea

Each SQL query flows through a modular pipeline — enabling transparency, debugging, and independent testing.

Core Engine Modules

- **Lexer:**
  - Converts SQL text into tokens and detects syntax errors early.
- **Parser:**
  - Uses recursive descent parsing.
  - Generates structured AST for nested queries.
- **Executor:**
  - Executes DDL/DML statements and applies constraints.
- **Storage Manager:**
  - Handles data persistence via JSON files.
  - Maintains schema and metadata consistency.

Supported SQL Features

Data Types:

- INT, TEXT, FLOAT, BOOLEAN

DDL Statements:

- CREATE TABLE, DROP TABLE

Constraints:

- PRIMARY KEY, FOREIGN KEY, NOT NULL

DML Operations:

- INSERT, SELECT, UPDATE, DELETE

Advanced SQL:

- JOINs (INNER, LEFT, RIGHT)
- GROUP BY, HAVING, ORDER BY
- Aggregates: COUNT, SUM, AVG, MAX, MIN
- Nested subqueries and conditionals

Frontend and User Experience

Highlights

- Modern SQL editor (Monaco) with syntax highlighting.
- Formatted result table and execution logs.
- Query history sidebar with auto-scroll.
- Light/Dark mode toggle.

Tech Stack

React, TypeScript, Axios, Monaco Editor, Lucide Icons

API Layer (FastAPI)

Key Endpoints

- `POST /api/v1/execute` – Execute SQL queries.
- `GET /api/v1/tables` – Retrieve schema metadata.
- `GET /api/v1/history` – Access query history.
- `POST /api/v1/reset` – Reset the database.

Features

- CORS-enabled API for seamless React integration.
- Pydantic validation for structured input/output.
- Meaningful error handling and response codes.

Storage Manager

Design Overview

- **schema.json:** Stores table definitions and constraints.
- **table**$_{n}$*ame.json* : *Holdsactualrowdata*.

Advantages

- Human-readable and easy to debug.
- Schema validation ensures consistency.
- Lightweight, atomic updates for reliability.
- **Automatic sparse indexing on primary keys** for faster lookups and joins.

Technology Stack

Backend:

- Python 3.11+, FastAPI, Pydantic
- JSON-based file storage with sparse indexing

Testing:

- pytest for unit and integration testing

Frontend:

- React 19, TypeScript, Axios
- Monaco Editor, Lucide Icons

DevOps:

- Git, Docker, modular structure

Results and Testing

Performance

- Simple queries: $<$10 ms
- Complex joins: 50–200 ms
- Sparse primary key indexing improves lookup performance by up to 3$\times$.
- Data persistence across sessions

Testing Summary

- Unit tests for lexer, parser, executor, API.
- Integration tests for end-to-end query flow.
- Verified SQL syntax and constraints.

Unique Highlights

Educational Perspective

- Transparent SQL execution visualization.
- Practical understanding of database internals.
- Modular components for learning and debugging.

Technical Perspective

- 100% custom implementation — no DB libraries.
- RESTful modular architecture.
- Sparse primary key indexing for performance.
- Strong separation of logic and data layers.

Future Enhancements

- Query optimization and multi-column indexing.
- Support for transactions, views, and authentication.
- Visualization of query plans and execution stats.
- Distributed data storage and replication.

Conclusion

Achievements

- Developed an end-to-end relational DBMS.
- Introduced **sparse primary key indexing** for efficient query execution.
- Designed a transparent, educational SQL engine.
- Delivered a responsive, full-stack implementation.

Key Takeaways

- Hands-on understanding of database internals.
- Experience in modular software design.
- Foundation for advanced DB optimization research.