**Maybe add some sort of IRC? - I want more networking here to make the website easier to implement**

**LATER: Convert to a simple project using a simple CSV database. Then convert to using sqlite amalgamation and then start the website project from here.**

**Part 1: MVP**

* Pt1: Saving to files, marshall and unmarshall
* Tabular data
* relational database
* Basic interface, CLI and then Network. Make it both via link time polymorphism
* Share via github
* Benchmarking

**Part 2: Performance Optimizations**

Few options they can choose: Provide a paper so they can implement

* Better data structures for lookup - Btrees
* Atomicity - Error recovery
* Independence: Concurrency

**Part 3:**

* Deploy as daemon
* Configuration
* Add way to manage daemon through systemd and signals

Next week: Deployment, network interface and querying by relationships

Relational tabular data with protocol to interact over network or something, can manage the service using signals. Add to startup through systemd.

Implement error recovery mechanism from reading paper

Make multithreaded

**Additional Features:**

* User management
* Better data structures
* Network interface
* Frontend to make interaction easier

Optional Parts:

Cli interface

On git/github

Make it a network daemon manageable by systemd - manage with signals

Custom protocol to interact with

Pattern for separating IO with logic

Pattern for separating specific implementation from actual database (1 layer indirection)

###################

====================================================================

This is a high flexibility assignment, think of how YOU would like to store data. Some popular formats are…

* Tabular with Relationships between

The user interface can be whatever you want. You will need to design some sort of Standard/Protocol for communication between the user and Database.

Note: Decouple the IO code from the logic code, that way it's easy to change the method of IO without having to go into logic code.

Eg.

int get\_user\_action\_stdin(\*UserAction); // Reads stdin and fills in empty user action during execution, returns 0 on failure

int read\_database(\*UserAction); // Reads database, returns 0 on failure

Improve your MVP

Make an improvement to your database. Create a way of benchmarking your database. Then implement a performance boost.

**Week 2**

Here are some papers that may help:

Ideas:

* Atomicity
* Error recovery

**Deployment**

* Add network user interface
* Deploy as service (upload your service configuration file
* Config through environment variables
* Manpages documentation

Create a protocol!

Separate the IO logic from the Database logic. That way 1. Its easier to test and 2. We can easily switch the IO device (CLI to Network sockets)

## Programming Language Creation

You don’t need to know everything about making programming languages, just the basics!

At the end of the day, it's just fancy string processing in 3 steps, these steps may be somewhat familiar if you are currently taking COSI 131, since their Shell PA did Scanning and

1. Scanner: Convert string to a list of data types. This is where syntax errors are caught

<https://craftinginterpreters.com/scanning.html#the-interpreter-framework>

eg.

INSERT BOOK 1984 “George Orwell” 328

[ActionType {Insert}, Datatype {Book}, DataField{1984}, DataField{George Orwell}, DataField{328}]

1. Parsing: Give the List of data types grammatical structure.
2. Virtual Machine: Execute

#### Room for Improvement

Implement your own Query Language (or just use SQL). For this, I would look into [Crafting Interpreters](https://craftinginterpreters.com/contents.html) or [this page](https://cstack.github.io/db_tutorial/parts/part2.html)