

MordorDB

# Implementation

## Description

MordorDB is a console application written in C# .  Data is stored on disk using JSON.  The most recently accessed data will be stored in memory. In memory, objects are defined recursively. Each collection can contain any number of key-value pairs. The keys will be strings, and the values can either be primitive values (such as strings or integers) or sub-collections.

## Logging

The log is a JSON object stored in a separate file. It is held in memory, and flushed to the disk every n operation, where n is configurable (these are the automatic checkpoints). The user can also force the log to be written to disk using a checkpoint keyword. The user can rollback to the last checkpoint using a rollback command.

## Indexes

Indexes are implemented using B+ trees. The user can create an index on any collection or sub-collection. Using an index can help speed up a range-based or aggregate query.

# User Manual

## Getting Started

If running on Windows, MordorDB can be started by running MordorDB.exe in the Release folder. Alternatively, the source code can be downloaded from github – a Visual Studio project is included. On Mac or Linux, the project should be compatible with Mono (not tested yet, sorry). Unfortunately, you will have to compile the project yourself on these platforms. All files related to the database will be stored in the directory where the executable is located so creating a new folder for organization is recommended but not required.

## Commands

The following commands are available from the MordorDB command-line interface:

**print** –

Displays a formatted view of the current data stored in memory.

**checkpoint** –

Creates a checkpoint and forces the log to write to disk

**rollback** –

Rolls the data back to its state at the last checkpoint

**help** –

Displays a list of commands that are available

**clear** –

Clears the terminal window

**exit** –

Ends the program

**query help** –

Displays a list of common query commands with examples included.

# Query Language

MordorDB includes the capabilities to create, retrieve, update and delete via method called from command line. Results can be filtered with a where command.  In addition it includes aggregate functions for sum, max, min, average and count. All queries must end with a semicolon.

### Select

In order to retrieve data from the database the user specifies the **select** command followed by a space. Then the collection name or names and then the field name are concatenated with periods.

To retrieve the field “first” of a document in the collection “students” with the id of “jschmoe” you would enter :

**select students.jschmoe.first;**

Select queries also support wildcards, denoted by **\***.To retrieve the “first” field of all documents in the students collection you would enter**:**

**select students.\*.first;**

### Insert

To create new data in the database you use the **insert** command. You must format the new data as a Json object. You must also specify the collection into which it will be inserted as well as the key for the new data.

To add the first and last names of a student to the students collection you would enter:

**insert { “first” : “Joe”, “last” : “Schmoe” } into students.jschmoe;**

### Update

To change existing data, you use the **update** command.

To update the value of the first name field of the jschmoe item in the students collection to “Joel”, you would enter:

**update students.jschmoe.first value “Joel”;**

To update the entire item you must enter a properly formatted json string.

**update students.jschmoe value {"first" : "Joel", "last" : "schmoe", "email" : "jschmoe@gmail.com"};**

### Delete

To remove an item from the collection you use the **delete** command.

To delete any items in the students collection where the first name is less than George you would enter:

**delete students.\* where students.\*.first < “George”;**

### Where Clause

You may limit your results by using a where clause. The where clause supports the following operators: >, < , <=, >=, !=, as well as the keywords “exists” or “nexists”. The relational operators work on strings and numbers, while the existence operators work on any data.

To only retrieve the first name of students who have the last name “schmoe” you would enter:

**select students.\*.first where students.\*.last == “schmoe”;**

Note that where clauses are only supported in “select” and “delete” queries. Additionally, any wildcards in the where clause must match a corresponding wildcard in the select clause. Finally, when the argument is a string, it will be compared in lexicographic order.

### Aggregates

MordorDB supports several aggregate functions, namely **min**, **max**, **count**, **sum**, and **average**. An aggregate function can be invoked by placing its keyword after the select. The count function will work on any type of document or primitive. The other aggregate functions expect a numeric value. For example, to count the number of documents in the “students” collection:

**select count students;**

### Indexes

Indexes can be used to speed up range queries and aggregate functions. To create and index, use the **index** command with the desired field:

**index students.\*.id;**

Note that if a document does not contain the indexed field, it will not be added to the index (a warning will also be issued).

Indexes are used automatically if an appropriate query is supplied. For example, after the previous index has been created, it will be automatically used in the query:

**select students.\* where students.\*.id > 1000;**

Indexes are invalided when there are any updates, inserts, or deletes to the indexed value. They are also not updated when the user inserts into the parent collection. In this version of MordorDB, it is up to the user to keep the index up to date; otherwise, results may be incorrect.

Indexes can be removed using the **deleteindex** command:

**deleteindex students.\*.id;**