FleetDB
A Schema-Free Database in Clojure

Mark McGranaghan

January 8, 2010

Motivation

Walkthrough

Implementation

QA

Motivation

Motivation

optimize for agile development

Walkthrough

- Client Quickstart
- Basic Queries
- Concurrency Tools

Client Quickstart

```
(use 'fleetdb.client)
(def client (connect))

(client ["ping"])
=> "pong"
```

Insert

```
(client
  ["insert" "accounts"
    {"id" 1 "owner" "Eve" "credits" 100}])
```

Insert

```
(client
  ["insert" "accounts"
    {"id" 1 "owner" "Eve" "credits" 100}])

=> 1
```

Select

```
(client
  ["select" "accounts" {"where" ["=" "id" 1]}])
```

Select

```
(client
  ["select" "accounts" {"where" ["=" "id" 1]}])

=> [{"id" 1 "owner" "Eve" "credits" 100}]
```

Select with Conditions

```
(client
  ["select" "accounts"
    {"where" [">=" "credits" 150]}])
```

Select with Order and Limit

```
(client
  ["select" "accounts"
    {"order" ["credits" "asc"] "limit" 2}])
```

Update

```
(client
  ["update" "accounts" {"credits" 105}
    {"where" ["=" "owner" "Eve"]}])
```

Explain (I)

```
(client
  ["explain" ["select" "accounts"
              {"where" ["=" "owner" "Eve"]}]])
```

Explain (I)

```
(client
  ["explain" ["select" "accounts"
              {"where" ["=" "owner" "Eve"]}]])

=> ["filter" ["=" "owner" "Eve"]
     ["collection-scan" "accounts"]]
```

Create Index

```
(client
  ["create-index" "accounts" "owner"])
```

Explain (II)

```
(client
  ["explain" ["select" "accounts"
              {"where" ["=" "owner" "Eve"]}]])

=> ["index-lookup" ["accounts" "owner" "Eve"]]]
```

Multi-Write

Multi-Write

```
(client
  ["multi-write"
   [write-query-1 write-query-2 ...]])
```

Multi-Write

```
(client
  ["multi-write"
   [write-query-1 write-query-2 ...]])

=> [result-1 result-2 ...]
```

Checked-Write

Checked-Write

```
(client
  ["checked-write"
   read-query
   expected-read-result
   write-query])
```

Checked-Write

```
(client
  ["checked-write"
   read-query
   expected-read-result
   write-query])

=> [true write-result]
```

Checked-Write

```
(client
  ["checked-write"
   read-query
   expected-read-result
   write-query])

=> [true write-result]

=> [false actual-read-result]
```

Implementation

- Background
- Organization
- Key ideas

Background

- http://clojure.org/data_structures
- http://clojure.org/sequences
- http://clojure.org/state

Organization

Organization

- `fleetdb.core`: pure functions

Organization

- `fleetdb.core`: pure functions
- `fleetdb.embedded`: identity and durability

Organization

- `fleetdb.core`: pure functions
- `fleetdb.embedded`: identity and durability
- `fleetdb.server`: network interface

`fleetdb.core`

- Pure functions

`fleetdb.core`

- Pure functions
- Databases as Clojure data structures

`fleetdb.core`

- Pure functions
- Databases as Clojure data structures
- Read: db + query $\longrightarrow$ result

`fleetdb.core`

- Pure functions
- Databases as Clojure data structures
- Read: db + query $\longrightarrow$ result
- 'Write': db + query $\longrightarrow$ result + new db

`fleetdb.core` Query Planner

`fleetdb.core` Query Planner

- Implements declarative queries

`fleetdb.core` Query Planner

- Implements declarative queries
- Database + query $\longrightarrow$ plan

`fleetdb.core` Query Planner

- Implements declarative queries
- Database + query $\longrightarrow$ plan

```
["select" "accounts"
  {"where" ["=" "owner" "Eve"]}]
```

`fleetdb.core` Query Planner

- Implements declarative queries
- Database + query $\longrightarrow$ plan

```
["select" "accounts"
  {"where" ["=" "owner" "Eve"]}]

["filter" ["=" "owner" "Eve"]
  ["record-scan" "accounts"]]
```

`fleetdb.core` Query Planner

- Implements declarative queries
- Database + query $\longrightarrow$ plan

```
["select" "accounts"
  {"where" ["=" "owner" "Eve"]}]

["filter" ["=" "owner" "Eve"]
  ["record-scan" "accounts"]]

["index-lookup" ["accounts" "owner" "Eve"]]
```

`fleetdb.core` Query Executor

`fleetdb.core` Query Executor

- Database + plan $\longrightarrow$ result

`fleetdb.core` Query Executor

- Database + plan $\longrightarrow$ result

```
["filter" ["=" "owner" "Eve"]
  ["record-scan" "accounts"]]
```

`fleetdb.core` Query Executor

- Database + plan $\longrightarrow$ result

```
["filter" ["=" "owner" "Eve"]
  ["record-scan" "accounts"]]
```

```
(filter (fn [r] (= (r "owner") "Eve"))
  (vals (:rmap (db "accounts"))))
```

fleetdb.embedded

- Wraps `fleetdb.core`

`fleetdb.embedded`

- Wraps `fleetdb.core`
- Adds identity and durability

`fleetdb.embedded`

- Wraps `fleetdb.core`
- Adds identity and durability
- Databases in atoms

`fleetdb.embedded`

- Wraps `fleetdb.core`
- Adds identity and durability
- Databases in atoms
- Append-only log

fleetdb.embedded Read Path

`fleetdb.embedded` Read Path

- Dereference database atom

`fleetdb.embedded` Read Path

- Dereference database atom
- Pass to `fleetdb.core`

`fleetdb.embedded` Read Path

- Dereference database atom
- Pass to `fleetdb.core`
- Return result

`fleetdb.embedded` Write Path

`fleetdb.embedded` Write Path

- Enter fair lock

`fleetdb.embedded` Write Path

- Enter fair lock
- Dereference database atom

`fleetdb.embedded` Write Path

- Enter fair lock
- Dereference database atom
- Pass to `fleetdb.core`

`fleetdb.embedded` Write Path

- Enter fair lock
- Dereference database atom
- Pass to `fleetdb.core`
- Append query to log

`fleetdb.embedded` Write Path

- Enter fair lock
- Dereference database atom
- Pass to `fleetdb.core`
- Append query to log
- Swap in new database value

`fleetdb.embedded` Write Path

- Enter fair lock
- Dereference database atom
- Pass to `fleetdb.core`
- Append query to log
- Swap in new database value
- Return result

`fleetdb.server`

- Wraps `fleetdb.embedded`

`fleetdb.server`

- Wraps `fleetdb.embedded`
- Adds JSON client API

Aside: Source Size

Aside: Source Size

Lines of Code	
core	630
embedded	130
server	120
lint	280
utilities	140
total	1300

Takeaways

Takeaways

- Clojure's data structures are awesome

Takeaways

- Clojure's data structures are awesome
- Clojure is viable for infrastructure software

Takeaways

- Clojure's data structures are awesome
- Clojure is viable for infrastructure software
- Try FleetDB!

Thanks for listening!

Questions?

http://FleetDB.org

http://github.com/mmcgrana