A. Im, G. Cai, H. Tunc, J. Stevens, Y. Barve, S. Hei
Vanderbilt University

Content

- Part 1: Introduction & Basics
- 2: CRUD
- 3: Schema Design
- 4: Indexes
- 5: Aggregation
- 6: Replication & Sharding

History

- mongoDB = "Hu**mongo**us DB"
  - Open-source
  - Document-based
  - "High performance, high availability"
  - Automatic scaling

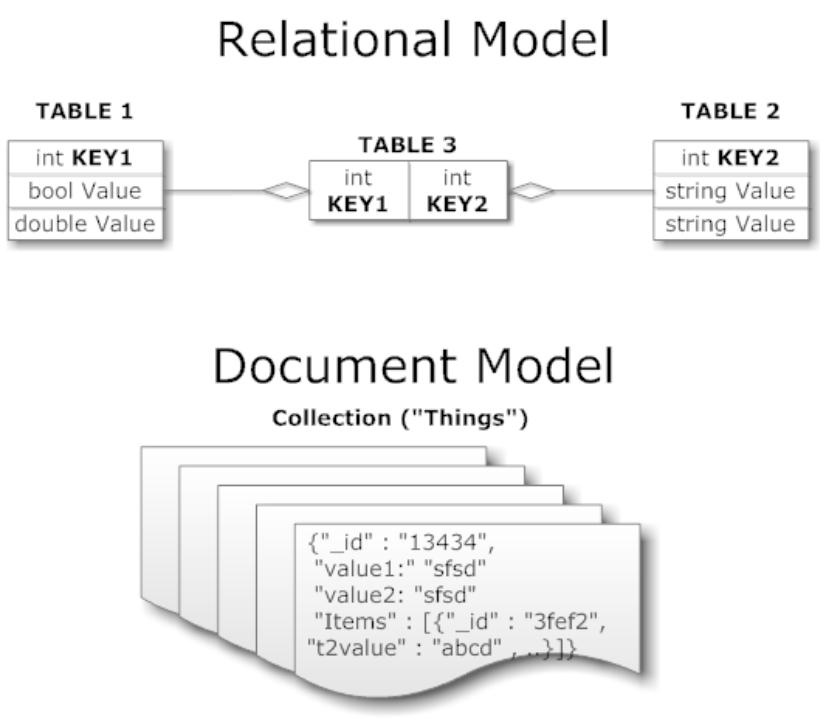
-blog.mongodb.org/post/475279604/on-distributed-consistency-part-1
-mongodb.org/manual

Other NoSQL Types

Key/value (Dynamo)

Columnar/tabular (HBase)

Document (mongoDB)



http://www.aaronstannard.com/post/2011/06/30/MongoDB-vs-SQL-Server.aspx

Motivations

- Problems with SQL
  - Rigid schema
  - Not easily scalable (designed for 90's technology or worse)
  - Requires unintuitive joins

- Perks of mongoDB
  - Easy interface with common languages (Java, Javascript, PHP, etc.)
  - DB tech should run anywhere (VM's, cloud, etc.)
  - Keeps essential features of RDBMS's while learning from key-value noSQL systems

http://www.slideshare.net/spf13/mongodb-9794741?v=qf1&b=&from_search=13

In Good Company

mongoDB

-Steve Francia, http://www.slideshare.net/spf13/mongodb-9794741?v=qf1&b=&from_search=13

JSON

- "JavaScript Object Notation"
- Easy for humans to write/read, easy for computers to parse/generate
- Objects can be nested
- Built on
  - name/value pairs
  - Ordered list of values

http://json.org/

JSON Example

```
{
"_id" :     "37010"
"city" :    "ADAMS",
"pop" :     2660,
"state" :   "TN",
“councilman” : {
            name: “John Smith”
            address: “13 Scenic Way”
          }
}
```

BSON

- "Binary JSON"
- Binary-encoded serialization of JSON-like docs
- Also allows "referencing"
- Embedded structure reduces need for joins
- Goals
  - Lightweight
  - Traversable
  - Efficient (decoding and encoding)

http://bsonspec.org/

BSON Example

{ "hello" : "world" } //JSON

\x16\x00\x00\x00 // total document size

\x02 // 0x02 = type String

hello\x00 // field name

\x06\x00\x00\x00world\x00 // field value (size of value, value, null terminator

\x00 // 0x00 = type EOO ('end of object')

BSON Types

Type	Number
Double	1
String	2
Object	3
Array	4
Binary data	5
Object id	7
Boolean	8
Date	9
Null	10
Regular Expression	11
JavaScript	13
Symbol	14
JavaScript (with scope)	15
32-bit integer	16
Timestamp	17
64-bit integer	18
Min key	255
Max key	127

The number can be used with the $type operator to query by type!

http://docs.mongodb.org/manual/reference/bson-types/

Data Model

- Document-Based (max 16 MB)
- Documents are in BSON format, consisting of field-value pairs
- Each document stored in a collection
- Collections
  - Have index set in common
  - Like tables of relational db's.
  - Documents do not have to have uniform structure

-docs.mongodb.org/manual/

The _id Field

- By default, each document contains an _id field. This field has a number of special characteristics:
  - Value serves as primary key for collection.
  - Value is unique, immutable, and may be any non-array type.
  - Default data type is ObjectId, which is "small, likely unique, fast to generate, and ordered." Sorting on an ObjectId value is roughly equivalent to sorting on creation time.

http://docs.mongodb.org/manual/reference/bson-types/

mongoDB vs. SQL

mongoDB	SQL
Document	Tuple
Collection	Table/View
PK: _id Field	PK: Any Attribute(s)
Uniformity not Required	Uniform Relation Schema
Index	Index
Embedded Structure	Joins
Shard	Partition

Getting Started with mongoDB

To install mongoDB, go to this link and click on the appropriate OS and architecture: https://www.mongodb.com/try/download/community

Install mongodb shell

https://www.mongodb.com/try/download/shell

Install VSCode extension: MongoDB for VS Code

Connect to `mongodb://localhost:27017`

CRUD

Create, Read, Update, Delete

CRUD: Using the Shell

To check which db you're using `db`

Show all databases `db.getMongo().getDBs();`

Switch db's/make a new one `use('<name>')`

See what collections exist `db.getCollectionNames()`

Note: db's are not actually created until you insert data!

CRUD: Inserting Data

Insert one document

```
db. getCollection(<collection>).insertOne({<field>:<value>})
```

```
INSERT INTO <table>
VALUES(<attributevalues>);
```

Inserting a document with a field name new to the collection is inherently supported by the BSON model.

To insert multiple documents, use an array.

CRUD: Querying

- Done on collections.
- Get all docs: db.getCollection(<collection>).find()
  - Returns a cursor, which is iterated over shell to display first 20 results.
  - Add .limit(<number>) to limit results
  - SELECT * FROM <table>;
- Get one doc: db. getCollection (<collection>).findOne()

CRUD: Querying

To match a specific value:

db.getCollection(<collection>).find({<field>:<value>})

"AND"

db. getCollection(<collection>).find({<field1>:<value1>,
<field2>:<value2>
})

SELECT *
FROM <table>
WHERE <field1> = <value1> AND <field2> = <value2>;

CRUD: Querying

OR
db. getCollection(<collection>).find({ $or: [
{<field>:<value1>}
{<field>:<value2>}]
})

```
SELECT *
FROM <table>
WHERE <field> = <value1> OR <field> = <value2>;
```

Checking for multiple values of same field

db. getCollection(<collection>).find({<field>: {$in : [<value>, <value>]}})

CRUD: Querying

Including/excluding document fields

db. getCollection(<collection>).find({<field1>:<value>}, {<field2>: 0})

SELECT field1
FROM <table>;

db. getCollection(<collection>).find({<field>:<value>}, {<field2>: 1})

CRUD: Updating

```
db.getCollection(<collection>).updateOne(
{<field1>:<value1>},     //all docs in which field = value
{$set: {<field2>:<value2>}} )
```

upsert: if true, creates a new doc when none matches search criteria.

```
UPDATE <table>
SET <field2> = <value2>
WHERE <field1> = <value1>;
```

CRUD: Updating

To remove a field

```
db.<collection>.updateOne({<field>:<value>},
{ $unset: { <field>: 1}})
```

Replace all field-value pairs

```
db.<collection>.updateMany({<field>:<value>},
{$set: {<field2>:<value2>}})
```

*NOTE: This overwrites ALL the contents of a document, even removing fields.

CRUD: Removal

Remove all records where field = value

```
db.<collection>.deleteMany({<field>:<value>})
```

```
DELETE FROM <table>
WHERE <field> = <value>;
```

As above, but only remove first document

```
db.<collection>.deleteOne({<field>:<value>})
```

CRUD

- Create
  - db.collection.insertOne(<document>)
  - db.collection.insertMany([<documents>])
  - db.collection.updateOne(<query>, <update>, { upsert: true })
  - db.collection.updateMany(<query>, <update>, { upsert: true })
- Read
  - db.collection.find(<query>, <projection>)
  - db.collection.findOne(<query>, <projection>)
- Update
  - db.collection.updateOne(<query>, <update>, <options>)
  - db.collection.updateMany(<query>, <update>, <options>)
- Delete
  - db.collection.deleteOne(<query>)
  - db.collection.deleteMany(<query>)

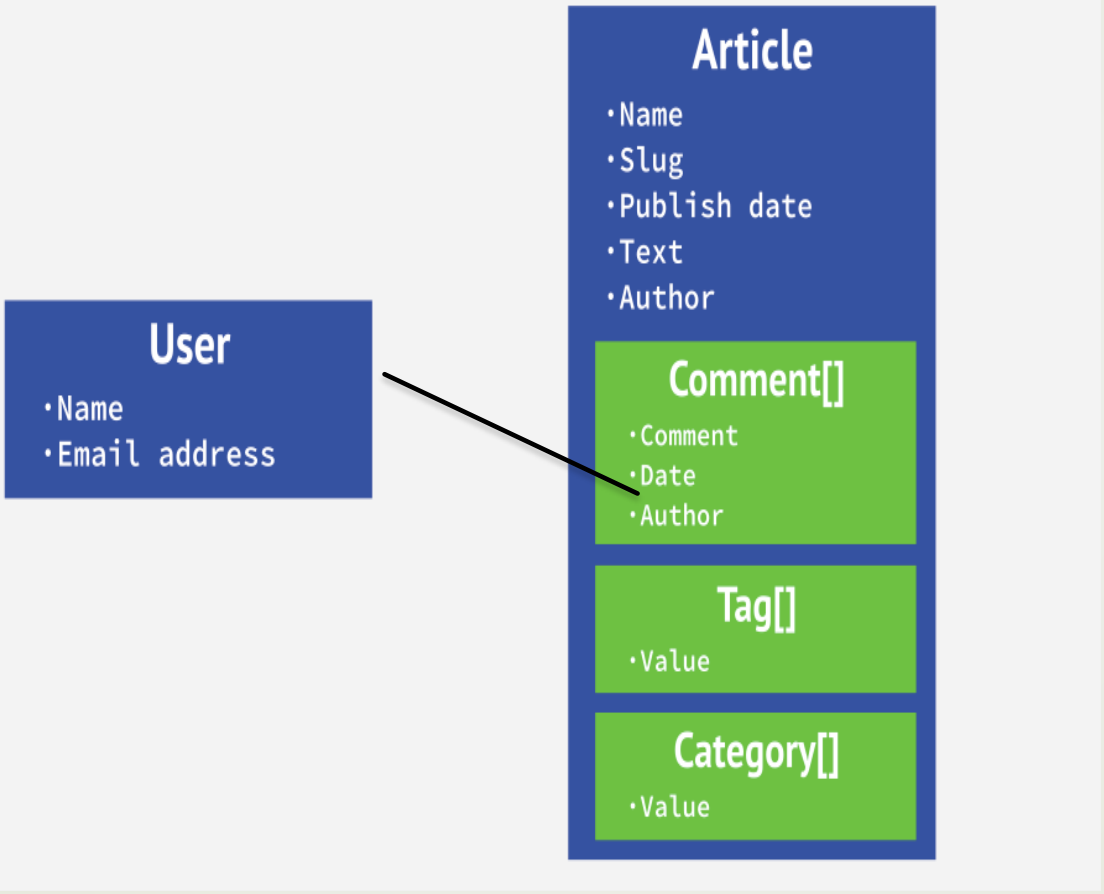
Intuition – why database exist in the first place?

- Why can't we just write programs that operate on objects?
  - Memory limit
  - We cannot swap back from disk merely by OS for the page based memory management mechanism
  - Non volatile
- Why can't we have the database operating on the same data structure as in program?
  - That is where mongoDB comes in

There are some patterns

- Embedding
- Linking

Embedding & Linking



Linking vs. Embedding

- Embedding is a bit like pre-joining data
- Document level operations are easy for the server to handle
- Embed when the “many” objects always appear with (viewed in the context of) their parents.
- Linking when you need more flexibility

Replication

- What is replication?
- Purpose of replication/redundancy
  - Fault tolerance
  - Availability
    - Increase read capacity

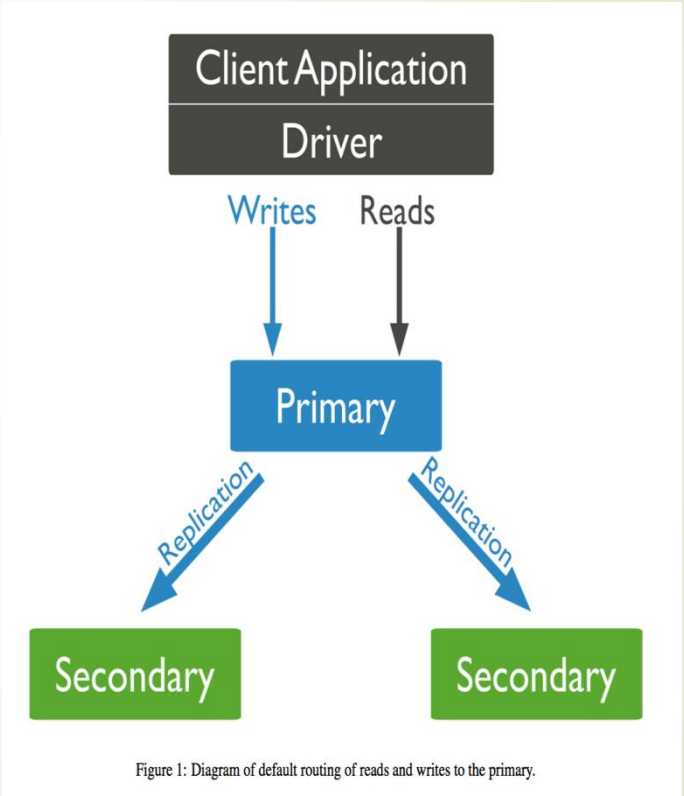


Figure 1: Diagram of default routing of reads and writes to the primary.

Replication in MongoDB

- Replica Set Members
  - Primary
    - Read, Write operations
  - Secondary
    - Asynchronous Replication
    - Can be primary
  - Arbiter
    - Voting
    - Can't be primary
  - Delayed Secondary
    - Can't be primary

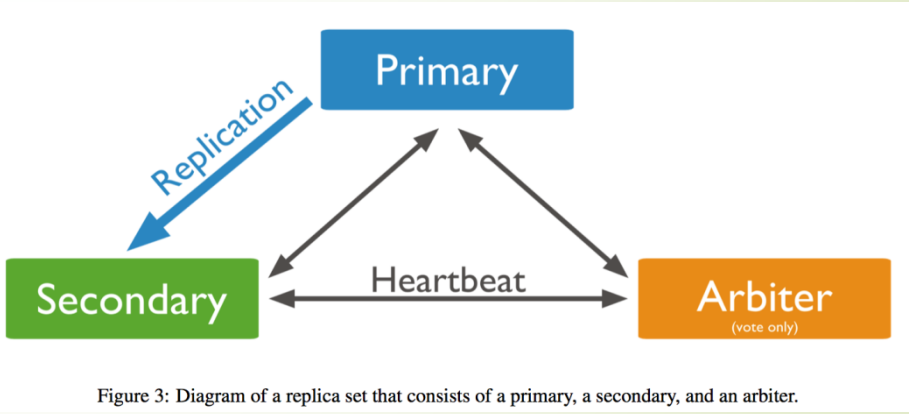
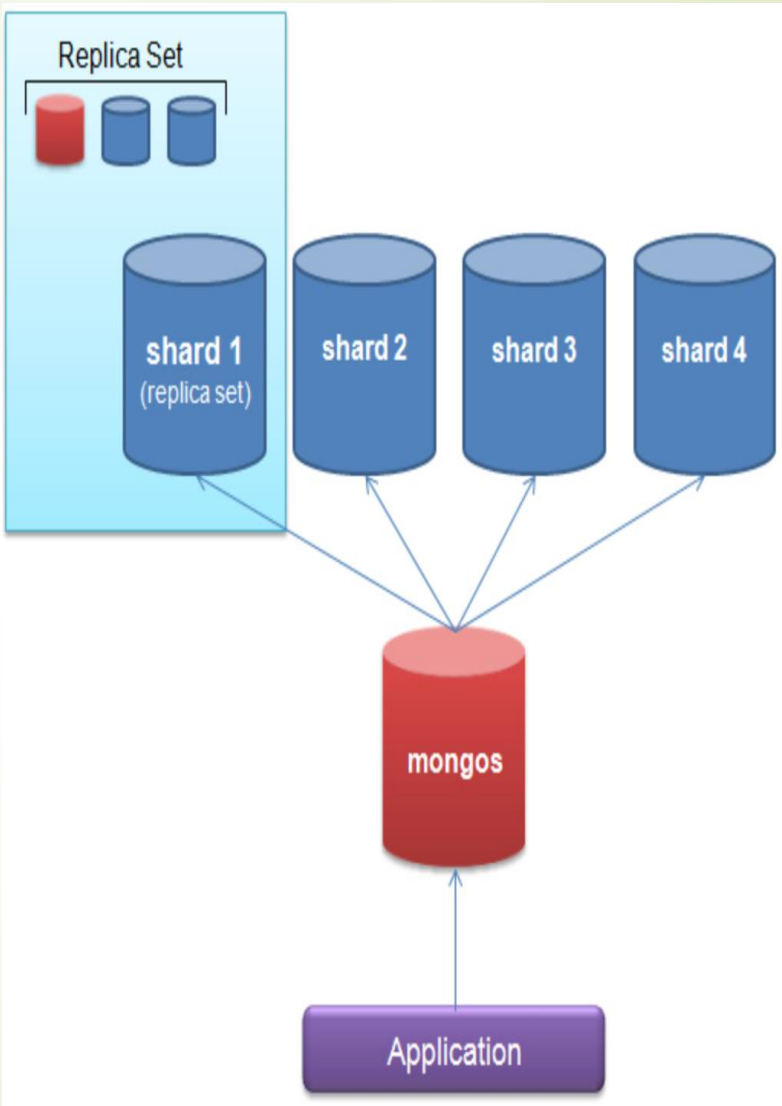


Figure 3: Diagram of a replica set that consists of a primary, a secondary, and an arbiter.

Sharding

- What is sharding?
- Purpose of sharding
  - Horizontal scaling out
- Query Routers
  - mongos
- Shard keys
  - Range based sharding
  - Cardinality
  - Avoid hotspotting



https://www.mongodb.com/docs/manual/sharding/