

MongoDB-1

WPI, Mohamed Eltabakh

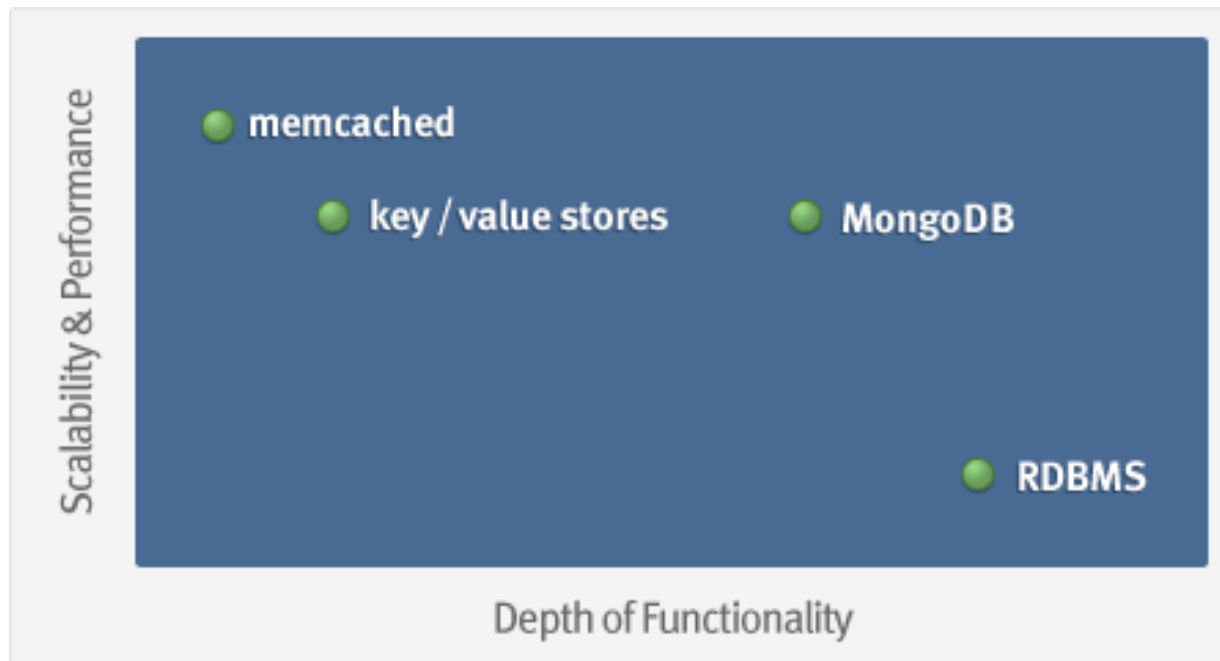
High-Level Overview



- Name comes from “Hum**mongo**us” & huge data
- Written in C++, developed in 2009
- Creator: 10gen, former doubleclick

MongoDB: Goal

- **Goal:** bridge the gap between key-value stores (which are fast and scalable) and relational databases (which have rich functionality).




What is MongoDB?

- **Defination:** MongoDB is an **open source, document-oriented** database designed with both scalability and developer agility in mind.
- Instead of storing your data in **tables and rows** as you would with a relational database, in MongoDB you store **JSON-like documents** with **dynamic schemas (schema-free, schemaless)**.

What is MongoDB? (Cont'd)

- **Document-Oriented DB**

- Unit object is a document
relational DBs



Three JSON documents representing users:

```
{name: "will", eyes: "blue", birthplace: "NY", aliases: ["bill", "la ciacco"], loc: [32.7, 63.4], boss: "ben"}
```

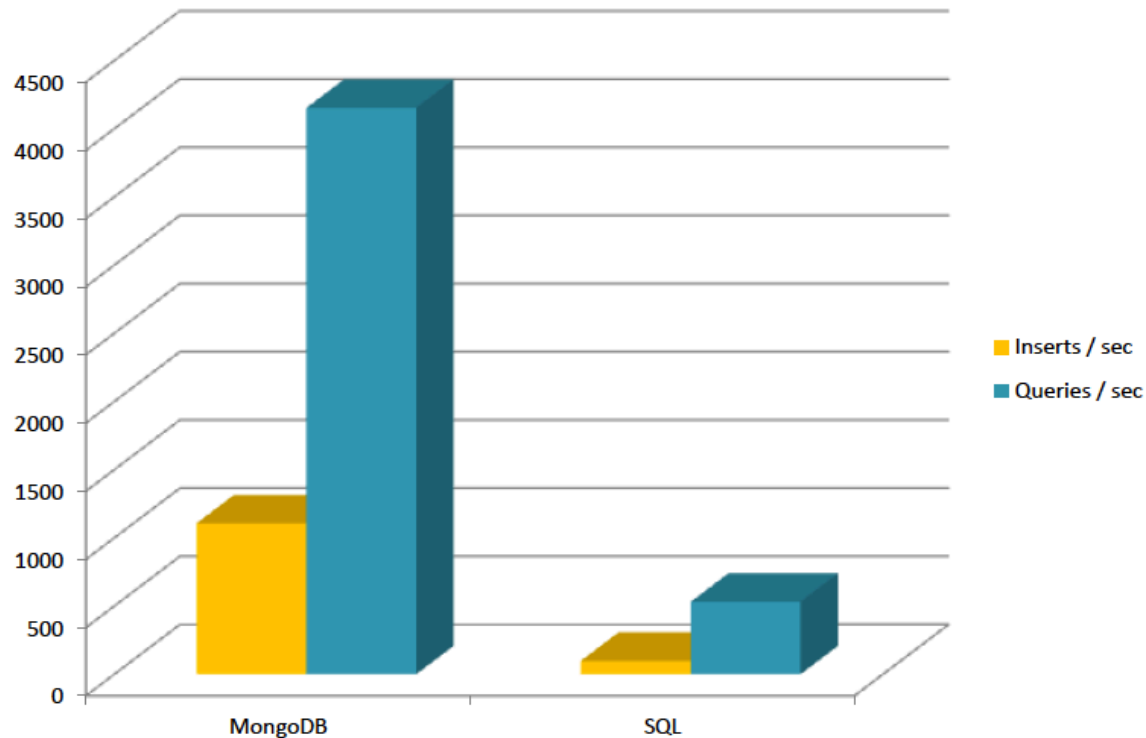
```
{name: "jeff", eyes: "blue", loc: [40.7, 73.4], boss: "ben"}
```

```
{name: "ben", hat: "yes"}
```

```
> db.user.findOne({age:39})
{
  "_id" : ObjectId("5114e0bd42..."),
  "first" : "John",
  "last" : "Doe",
  "age" : 39,
  "interests" : [
    "Reading",
    "Mountain Biking ]
  "favorites": {
    "color": "Blue",
    "sport": "Soccer"}
}
```

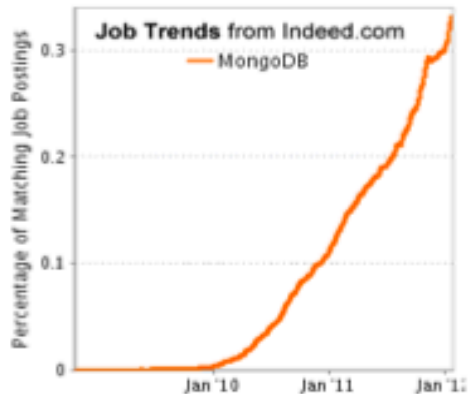
Is It Fast?

- For semi-structured & complex relationships: Yes



It is Growing Fast

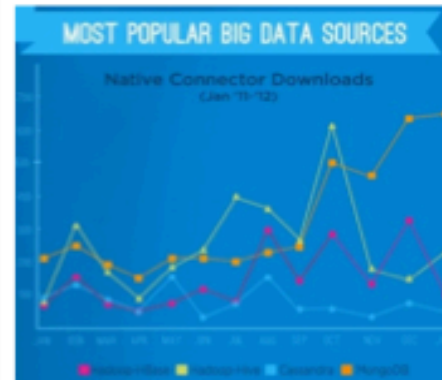
#2 ON INDEED'S FASTEST GROWING JOBS



Top Job Trends

1. [HTML5](#)
2. **MongoDB**
3. [iOS](#)
4. [Android](#)
5. [Mobile_app](#)
6. [Puppet](#)
7. [Hadoop](#)
8. [jQuery](#)
9. [PaaS](#)
10. [Social Media](#)

JASPER SOFTWARE BIGDATA INDEX



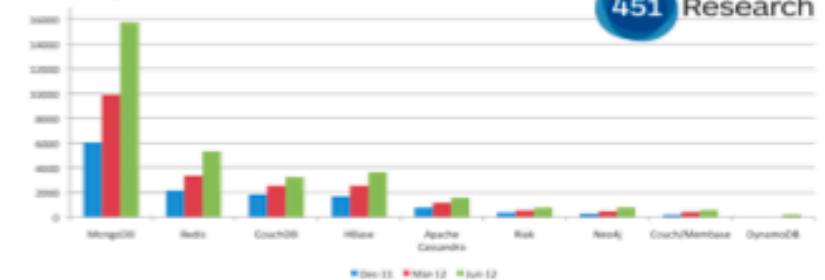
Demand for MongoDB, the document-oriented NoSQL database, saw the biggest spike with over 200% growth in 2011.

GOOGLE SEARCHES



451 GROUP "MONGODB INCREASING ITS DOMINANCE"

Relative adoption of NoSQL - LinkedIn member skills



Integration with Others

- C
- C++
- Erlang
- Haskell
- Java
- Javascript
- .NET (C# F#, PowerShell)
- Node.js
- Perl
- PHP
- Python
- Ruby
- Scala



NoSQL DBs

NoSQL: Categories

- Key-value



- Graph database



- Document-oriented



- Column family



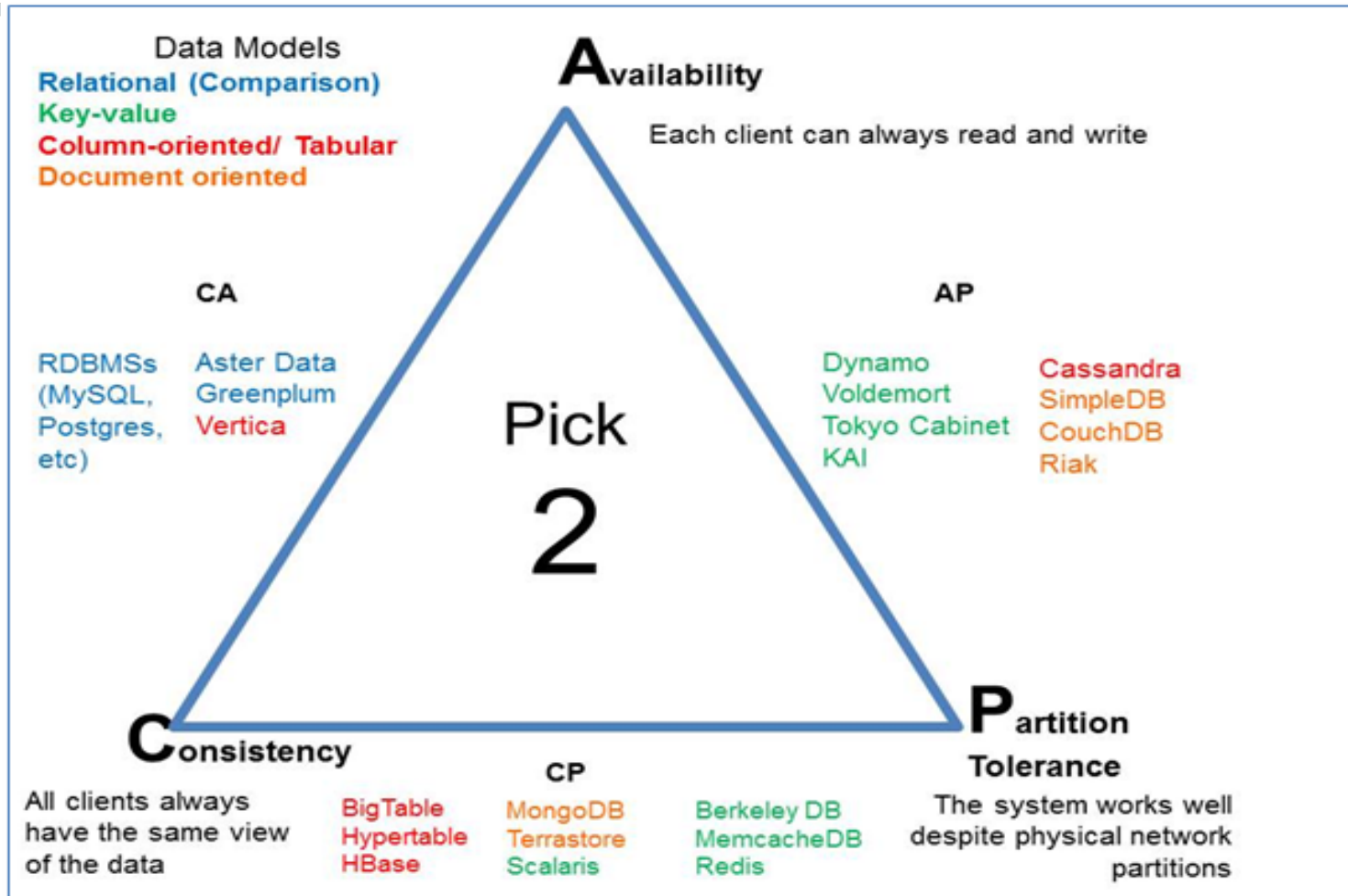
What is NoSQL

- Stands for **N**ot **O**nly **SQL**??
- Class of non-relational data storage systems
- Usually do not require a fixed table schema nor do they use the concept of joins
 - Distributed data storage systems
- All NoSQL offerings relax one or more of the ACID properties
 - will talk about the CAP theorem

CAP Theorem

- Three properties of a system
 - **Consistency** (all copies have same value)
 - **Availability** (system can run even if parts have failed)
 - Via replication
 - **Partition Tolerance** (Even if part is down, others can take over)
- CAP “Theorem”:
 - You can have at most two of these three properties for any system
 - Pick two !!!

CAP Theorem



Availability

- Traditionally, thought of as the server/process available five 9's (99.999 %).
 - Failures are rare
- In modern commodity distributed systems:
 - Want a system that is resilient in the face of network disruption
 - **Use Replication**

Eventual Consistency

- When no updates occur for a long period of time:
 - Eventually all updates will propagate through the system and all the nodes will be consistent
- For a given accepted update and a given node:
 - Eventually either the update reaches the node or the node is removed from service

Eventual Consistency

- ***BASE*** Concept
 - Basically **A**vailable, **S**oft state, **E**ventual consistency
 - As opposed to ACID in RDBMS
 - Soft state: copies of a data item may be inconsistent
 - Eventually Consistent – copies becomes consistent at some later time if there are no more updates to that data item

What does NoSQL Not Provide



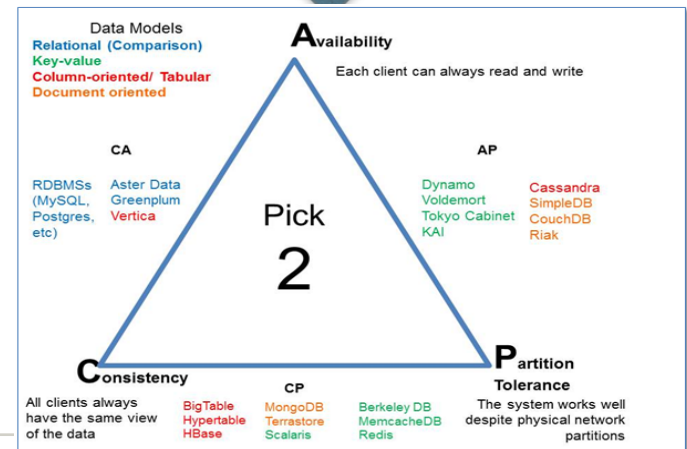
ISA

NO★SQL



Follows

- No built-in join
- No ACID transactions
- No SQL



Data Model

Data Model

- BSON format (binary JSON)
- Developers can easily map to modern object-oriented languages without a complicated ORM layer.
- lightweight, traversable, efficient

Terms Mapping (DB vs. MongoDB)

RDBMS	MongoDB
Database	Database
Table	Collection
Tuple/Row	Document
column	Field
Table Join	Embedded Documents
Primary Key	Primary Key (Default key <code>_id</code> provided by mongodb itself)
Database Server and Client	
Mysqld/Oracle	mongod
mysql/sqlplus	mongo

JSON

Field Name

Field Value

One document

```
{
  "firstName": "John",
  "lastName": "Smith",
  "isAlive": true,
  "age": 25,
  "height_cm": 167.6,
  "address": {
    "streetAddress": "21 2nd Street",
    "city": "New York",
    "state": "NY",
    "postalCode": "10021-3100"
  },
  "phoneNumbers": [
    {
      "type": "home",
      "number": "212 555-1234"
    },
    {
      "type": "office",
      "number": "646 555-4567"
    }
  ],
  "children": [],
  "spouse": null
}
```

- ***Field Value***
 - Scalar (Int, Boolean, String, Date, ...)
 - Document (Embedding or Nesting)
 - Array of JSON objects

Another Example

```
{ author: 'joe',  
  created : new Date('03/28/2009'),  
  title : 'Yet another blog post',  
  text : 'Here is the text...',  
  tags : [ 'example', 'joe' ],  
  comments : [  
    { author: 'jim',  
      comment: 'I disagree'  
    },  
    { author: 'nancy',  
      comment: 'Good post'  
    }  
  ]  
}
```



Remember it is stored in binary formats (BSON)

```
"\x16\x00\x00\x00\x02hello\x00  
\x06\x00\x00\x00world\x00\x00"  
"1\x00\x00\x00\x04BSON\x00&\x00  
\x00\x00\x020\x00\x08\x00\x00  
\x00awesome\x00\x011\x00333333  
\x14@\x102\x00\xc2\x07\x00\x00  
\x00\x00"
```

MongoDB Model

One *document* (e.g., one *tuple* in RDBMS)

```
{
  name: "sue",
  age: 26,
  status: "A",
  groups: [ "news", "sports" ]
}
```

← field: value
← field: value
← field: value
← field: value

- **Collection** is a group of similar documents
- Within a collection, each document must have a unique Id

One *Collection* (e.g., one *Table* in RDBMS)

```
{
  name: "al",
  age: 18,
  status: "D",
  groups: [ "politics", "news" ]
}
```

Collection

**Unlike RDBMS:
No Integrity Constraints in
MongoDB**

MongoDB Model

One *document* (e.g., one *tuple* in RDBMS)

```
{
  name: "sue",
  age: 26,
  status: "A",
  groups: [ "news", "sports" ]
}
```

← field: value
← field: value
← field: value
← field: value

- The field names **cannot** start with the **\$** character
- The field names **cannot** contain the **.** character
- Max size of single document 16MB

One *Collection* (e.g., one *Table* in RDBMS)

```
{
  name: "al",
  age: 18,
  status: "D",
  groups: [ "politics", "news" ]
}
```

Collection

```
{
  _id: ObjectId(7df78ad8902c)
  title: 'MongoDB Overview',
  description: 'MongoDB is no sql database',
  by: 'tutorials point',
  url: 'http://www.tutorialspoint.com',
  tags: ['mongodb', 'database', 'NoSQL'],
  likes: 100,
  comments: [
    {
      user:'user1',
      message: 'My first comment',
      dateCreated: new Date(2011,1,20,2,15),
      like: 0
    },
    {
      user:'user2',
      message: 'My second comments',
      dateCreated: new Date(2011,1,25,7,45),
      like: 5
    }
  ]
}
```

Example Document in MongoDB

- `_id` is a special column in each document
- Unique within each collection
- `_id` \leftrightarrow Primary Key in RDBMS
- `_id` is 12 Bytes, you can set it yourself
- Or:
 - 1st 4 bytes \rightarrow timestamp
 - Next 3 bytes \rightarrow machine id
 - Next 2 bytes \rightarrow Process id
 - Last 3 bytes \rightarrow incremental values

No Defined Schema (Schema-free Or Schema-less)

- MongoDB does not need any defined data schema.
- Every document could have different data!

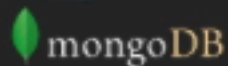
```
{name: "will",  
  eyes: "blue",  
  birthplace: "NY",  
  aliases: ["bill", "la  
ciacco"],  
  gender: "???",  
  boss: "ben"}
```

```
{name: "jeff",  
  eyes: "blue",  
  height: 72,  
  boss: "ben"}
```

```
{name: "brendan",  
  aliases: ["el diablo"]}
```

```
{name: "ben",  
  hat: "yes"}
```

```
{name: "matt",  
  pizza: "DiGiorno",  
  height: 72,  
  boss: 555.555.1212}
```



Data Model Comparison

Relational DB vs. NoSQL

This is hard...

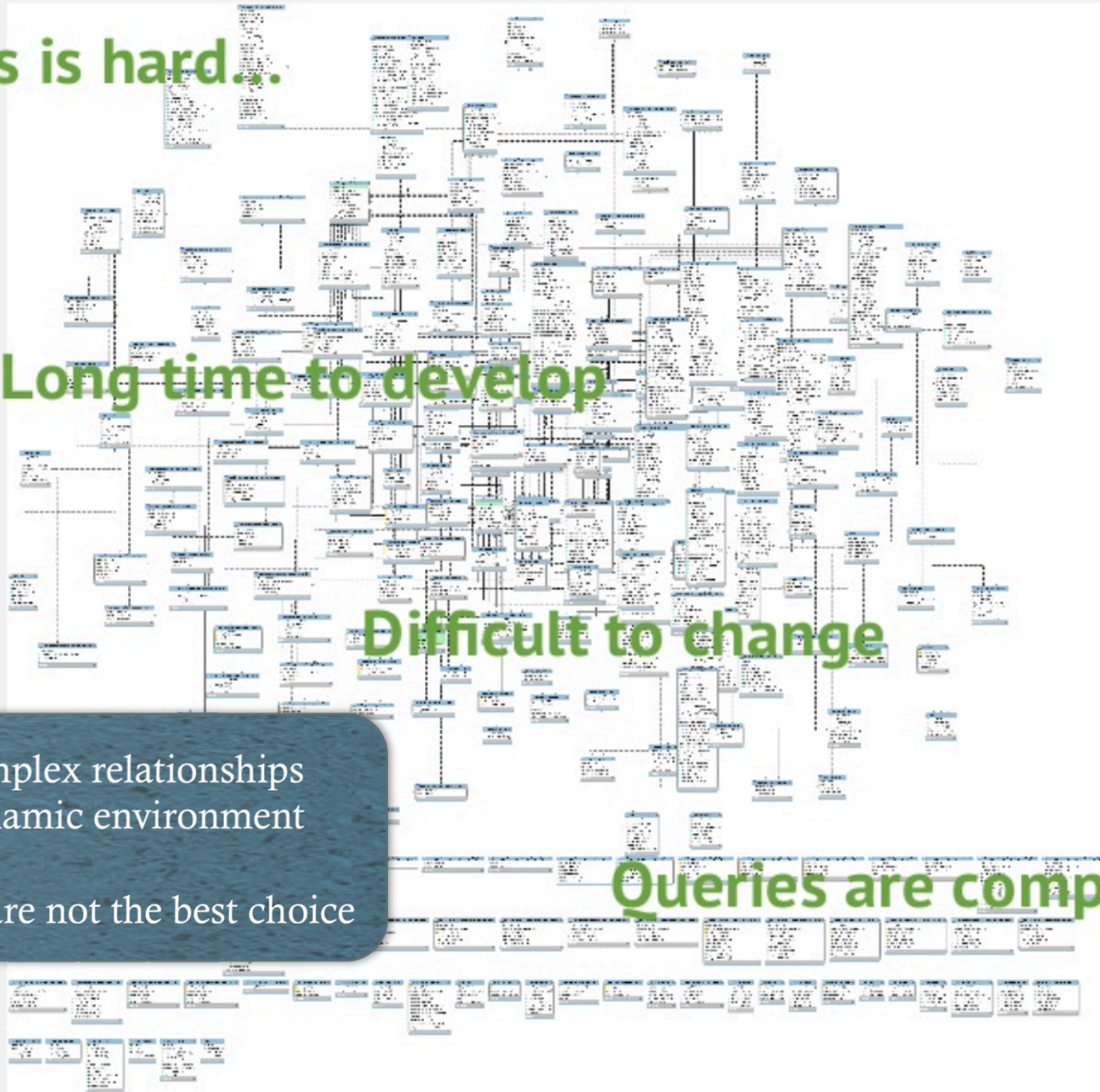
Long time to develop

Difficult to change

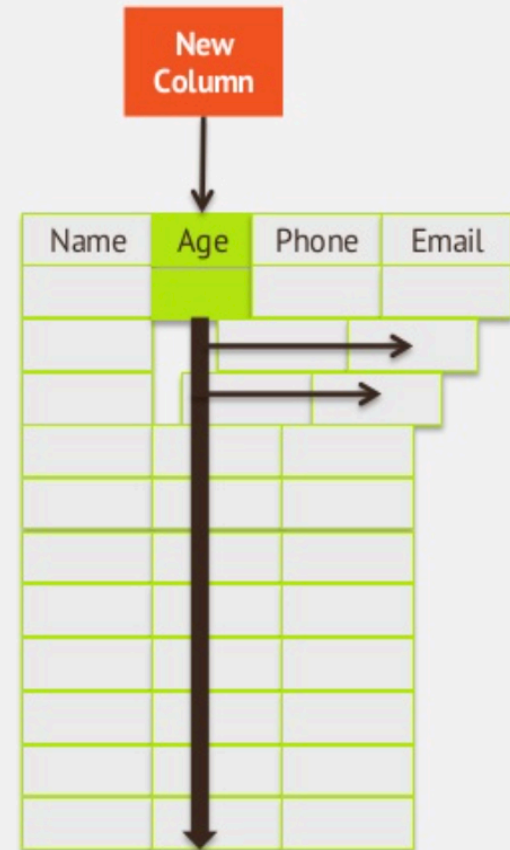
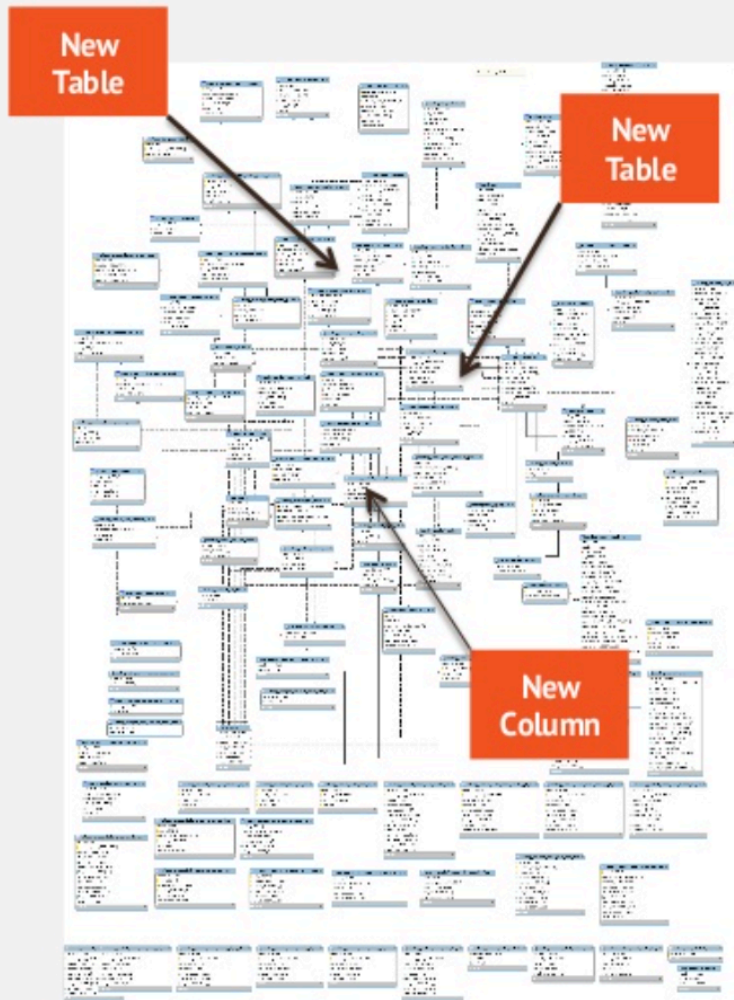
- Complex relationships
- Dynamic environment

RDBMS are not the best choice

Queries are complex



Hard to make changes



Key-Value Data Model

Key → Value store

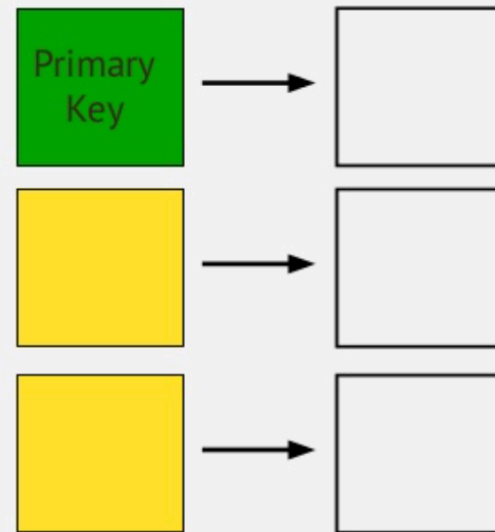
- One-dimensional storage
- Single value is a blob
- Query on key only
- No schema
- Value can be replaced but not updated



Relational Data Model

Relational Record

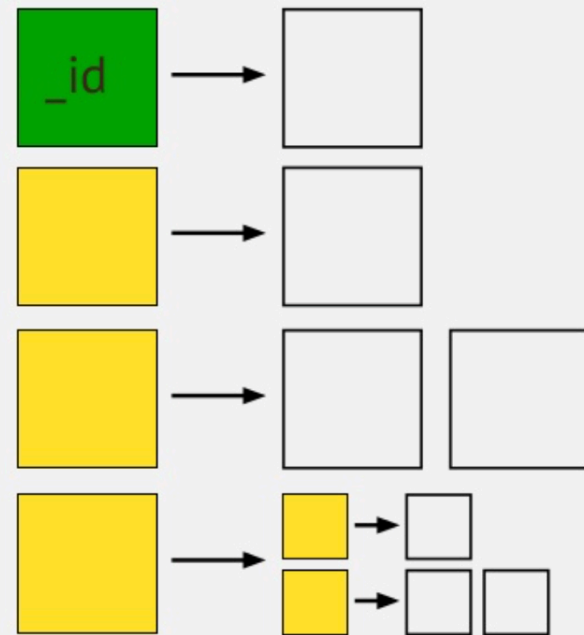
- Two-dimensional storage
- Field contains a single value
- Query on any field
- Very structured schema
- Poor data locality requires many tables, joins, and indexes.



Document Data Model

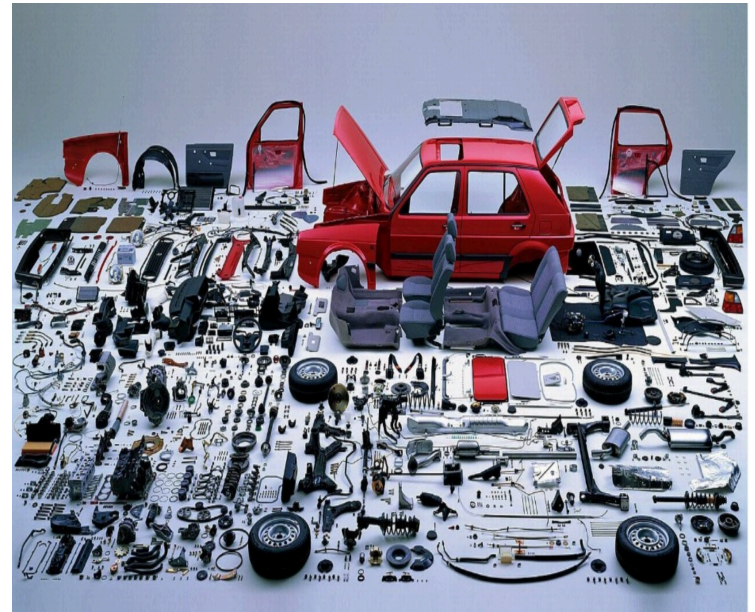
MongoDB Document

- **N-dimensional** storage
- Field can contain **many** values and **embedded** values
- Query on **any field & level**
- **Flexible** schema
- Optimal data locality requires fewer **indexes** and provides better **performance**

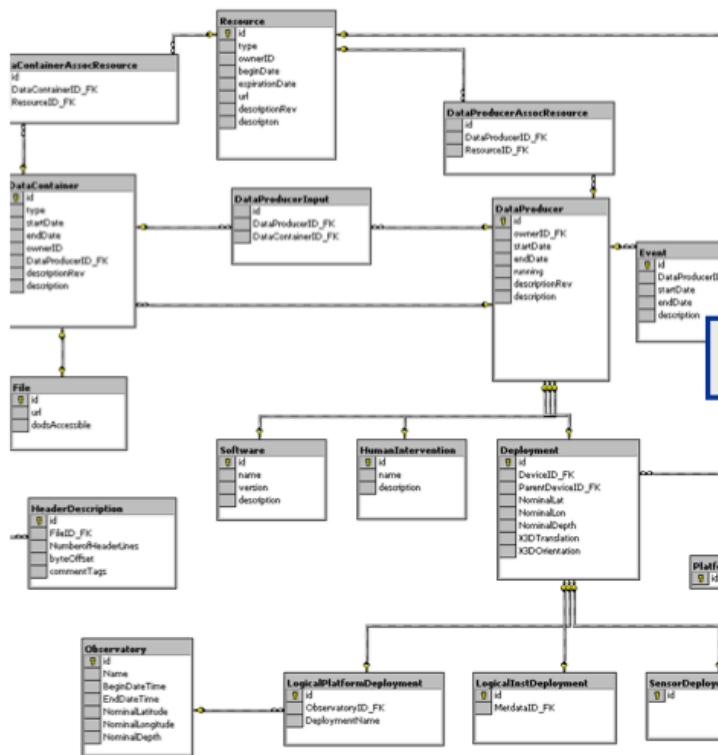


Document vs. Relational Models

- **Relational**
 - Focus on data storage
 - At query time → build your business objects
- **Document**
 - Focus on data usage
 - Always maintain your business object



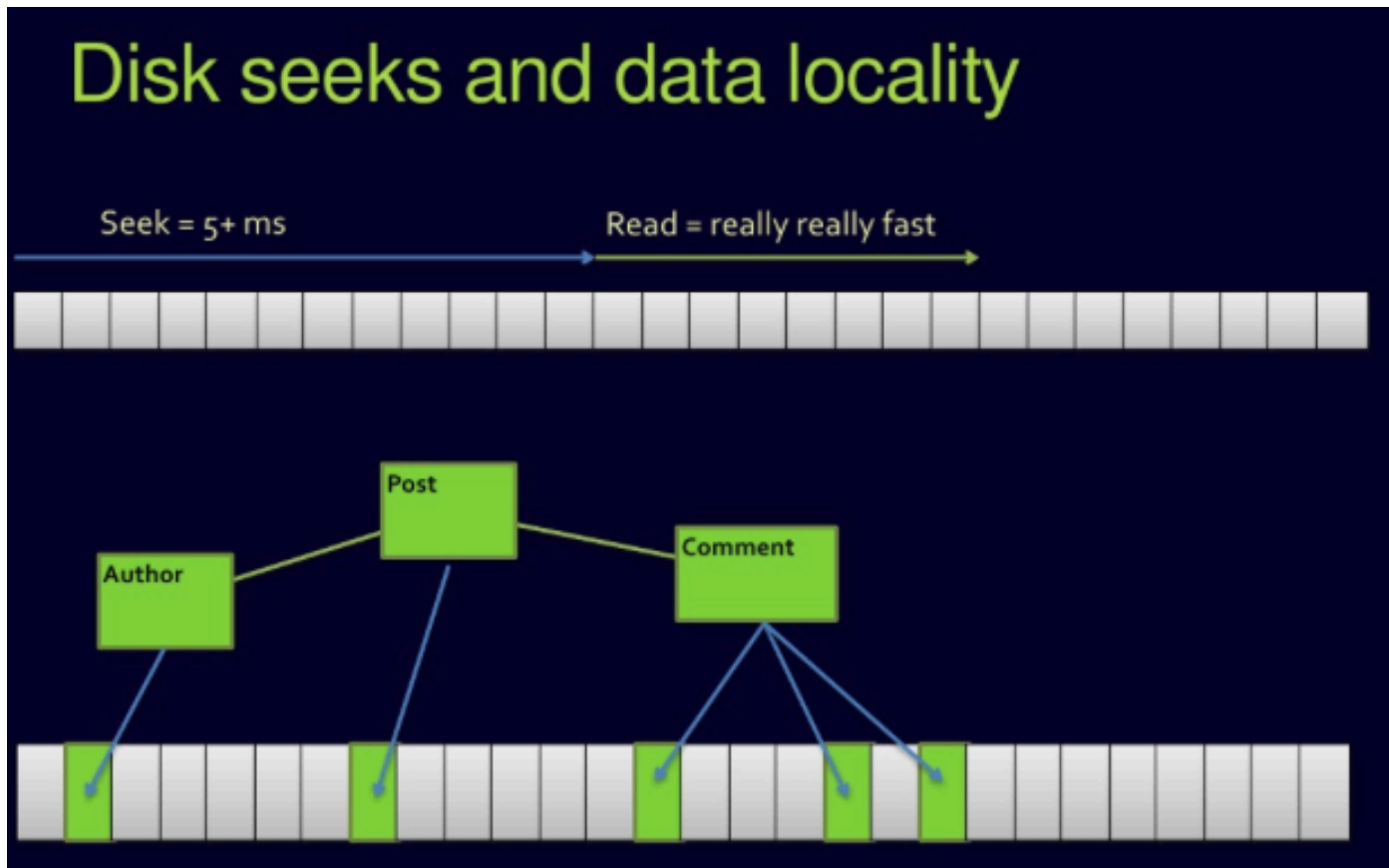
Tradeoff: Normalization vs. Easy Usage



```
{
  title: 'MongoDB',
  contributors: [
    { name: 'Eliot Horowitz',
      email: 'eliot@10gen.com' },
    { name: 'Dwight Merriman',
      email: 'dwight@10gen.com' }
  ],
  model: {
    relational: false,
    awesome: true
  }
}
```

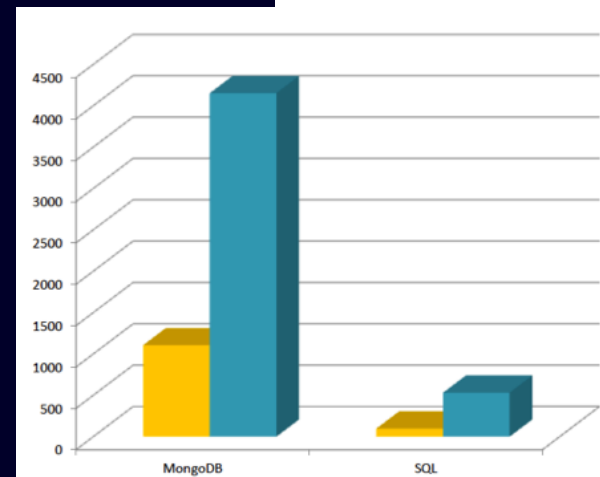
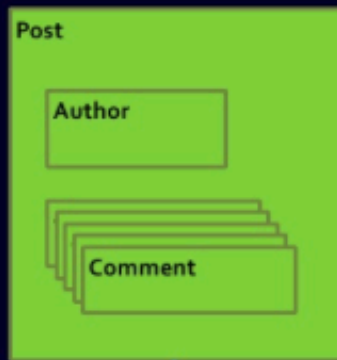
Complex Join Queries

Relational DBs



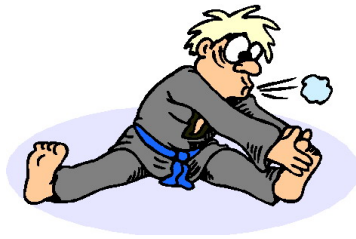
No Joins in MongoDB

Disk seeks and data locality



Updates & Querying

Must Practice It



Install it



Practice simple stuff



Move to complex stuff

Install it from here: <http://www.mongodb.org>

Manual: <http://docs.mongodb.org/master/MongoDB-manual.pdf>
(Focus on Ch. 3, 4 for now)

Dataset: <http://docs.mongodb.org/manual/reference/bios-example-collection/>

CRUD

- **C**reate
 - `db.collection.insert(<document>)`
 - `db.collection.save(<document>)`
 - `db.collection.update(<query>, <update>, { upsert: true })`
- **R**ead
 - `db.collection.find(<query>, <projection>)`
 - `db.collection.findOne(<query>, <projection>)`
- **U**ppdate
 - `db.collection.update(<query>, <update>, <options>)`
- **D**elete
 - `db.collection.remove(<query>, <justOne>)`

CRUD Examples

```
> db.user.insert({
  first: "John",
  last : "Doe",
  age: 39
})
```

```
> db.user.find ()
{
  "_id" : ObjectId("51..."),
  "first" : "John",
  "last" : "Doe",
  "age" : 39
}
```

```
> db.user.update(
  {"_id" : ObjectId("51...")},
  {
    $set: {
      age: 40,
      salary: 7000}
  }
)
```

```
> db.user.remove({
  "first": /^J/
})
```

Examples

In RDBMS

```
CREATE TABLE users (  
  id MEDIUMINT NOT NULL  
    AUTO_INCREMENT,  
  user_id Varchar(30),  
  age Number,  
  status char(1),  
  PRIMARY KEY (id)  
)
```

```
DROP TABLE users
```

In MongoDB

Either insert the 1st document

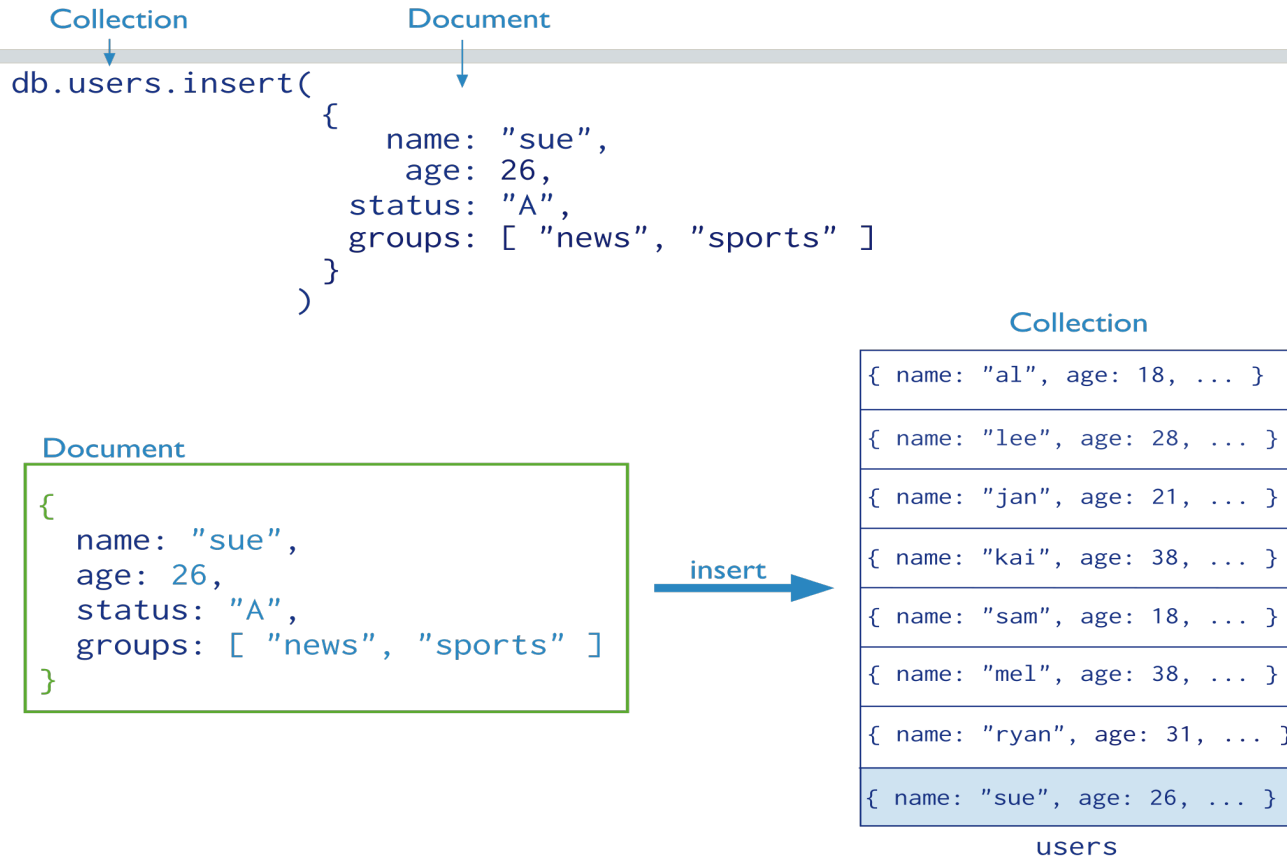
```
db.users.insert( {  
  user_id: "abc123",  
  age: 55,  
  status: "A"  
} )
```

Or create “Users” collection explicitly

```
db.createCollection("users")
```

```
db.users.drop()
```

Insertion

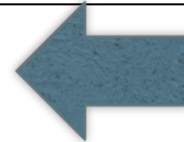


- The collection “users” is created automatically if it does not exist

Multi-Document Insertion (Use of Arrays)

```
var mydocuments =  
  [  
    {  
      item: "ABC2",  
      details: { model: "14Q3", manufacturer: "M1 Corporation" },  
      stock: [ { size: "M", qty: 50 } ],  
      category: "clothing"  
    },  
    {  
      item: "MNO2",  
      details: { model: "14Q3", manufacturer: "ABC Company" },  
      stock: [ { size: "S", qty: 5 }, { size: "M", qty: 5 }, { size: "L", qty: 1 } ],  
      category: "clothing"  
    },  
    {  
      item: "IJK2",  
      details: { model: "14Q2", manufacturer: "M5 Corporation" },  
      stock: [ { size: "S", qty: 5 }, { size: "L", qty: 1 } ],  
      category: "houseware"  
    }  
  ]  
};
```

```
db.inventory.insert( mydocuments );
```



All the documents are
inserted at once

Multi-Document Insertion (Bulk Operation)

- A temporary object in memory
- Holds your insertions and uploads them at once

There is also *Bulk Ordered* object



```
var bulk = db.inventory.initializeUnorderedBulkOp();
bulk.insert(
  {
    item: "BE10",
    details: { model: "14Q2", manufacturer: "XYZ Company" },
    stock: [ { size: "L", qty: 5 } ],
    category: "clothing"
  }
);
bulk.insert(
  {
    item: "ZYT1",
    details: { model: "14Q1", manufacturer: "ABC Company" },
    stock: [ { size: "S", qty: 5 }, { size: "M", qty: 5 } ],
    category: "houseware"
  }
);
bulk.execute();
```

_id column is added
automatically

Deletion (Remove Operation)

- You can put condition on any field in the document (even ***_id***)

```
db.users.remove(  
  { status: "D" }  
)
```

← collection
← remove criteria

The following diagram shows the same query in SQL:

```
DELETE FROM users  
WHERE status = 'D'
```

← table
← delete criteria

db.users.remove ()



Removes all documents from *users* collection

Update

```
db.users.update(  
  { age: { $gt: 18 } },  
  { $set: { status: "A" } },  
  { multi: true }  
)
```

← collection
← update criteria
← update action
← update option

Otherwise, it will update only the 1st matching document

Equivalent to in SQL:

```
UPDATE users  
SET status = 'A'  
WHERE age > 18
```

← table
← update action
← update criteria

Update (Cont'd)

Two operators

```
db.inventory.update(  
  { item: "MNO2" },  
  {  
    $set: {  
      category: "apparel",  
      details: { model: "14Q3", manufacturer: "XYZ Company" }  
    },  
    $currentDate: { lastModified: true }  
  }  
)
```

For the document with `item` equal to "MNO2", use the `$set` operator to update the `category` field and the `details` field to the specified values and the `$currentDate` operator to update the field `lastModified` with the current date.

Replace a document

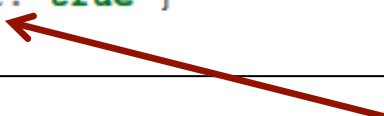
```
db.inventory.update(  
  { item: "BE10" }, ← Query Condition  
  {  
    item: "BE05",  
    stock: [ { size: "S", qty: 20 }, { size: "M", qty: 5 } ],  
    category: "apparel"  
  }  
)
```

New doc

For the document having item = "BE10", replace it with the given document

Insert or Replace

```
db.inventory.update(  
  { item: "TBD1" },  
  {  
    item: "TBD1",  
    details: { "model" : "14Q4", "manufacturer" : "ABC Company" },  
    stock: [ { "size" : "S", "qty" : 25 } ],  
    category: "houseware"  
  },  
  { upsert: true }  
)
```



The *upsert* option

If the document having item = “TBD1” is in the DB, it will be replaced
Otherwise, it will be inserted.