MongoDB:
Model and Operations

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 : Relational DB vs. MongoDB

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

JSON

Field Name

Field Value

```
{
  "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

Example: JSON to BSON

```
{ 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'
              }
  ]
}
```

For efficiency, 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",                  ← field: value
  age: 26,                      ← field: value
  status: "A",                  ← field: value
  groups: [ "news", "sports" ]  ← field: value
}
```

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

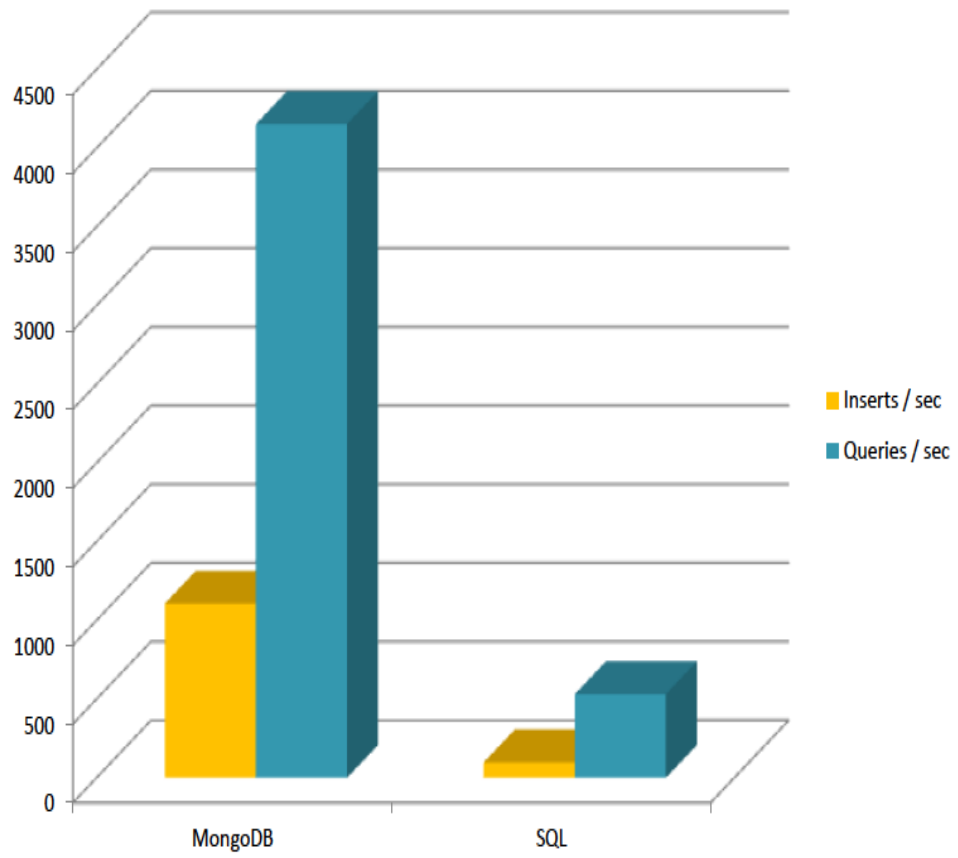
```
{
  name: "al",
  age: 18,
  status: "D",
  groups: [ "politics", "news" ]
}
```

Collection

- **Document** is equivalent to a tuple
- **Collection** groups similar documents
- Within a collection, each d

Unlike RDBMS: No Integrity Constraints in MongoDB

Performance



Unlike RDBMS: No Integrity Constraints in MongoDB

MongoDB Model (restrictions)

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

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

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

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

Collection

Field names:

- **cannot** start with the **$** character (command)
- **cannot** contain the **“.”** character (position)
- Max size of single document 16MB

```
{
  _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
    }
  ]
}
```

Identifier in MongoDB

- _id is special column in each document
- Unique within each collection
- _id ←→ Primary Key in RDBMS
- _id is 12 Bytes, you can set it yourself
- Or, system generated:
  - 1st 4 bytes → timestamp
  - Next 3 bytes → machine id
  - Next 2 bytes → process id
  - Last 3 bytes → 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"}
```

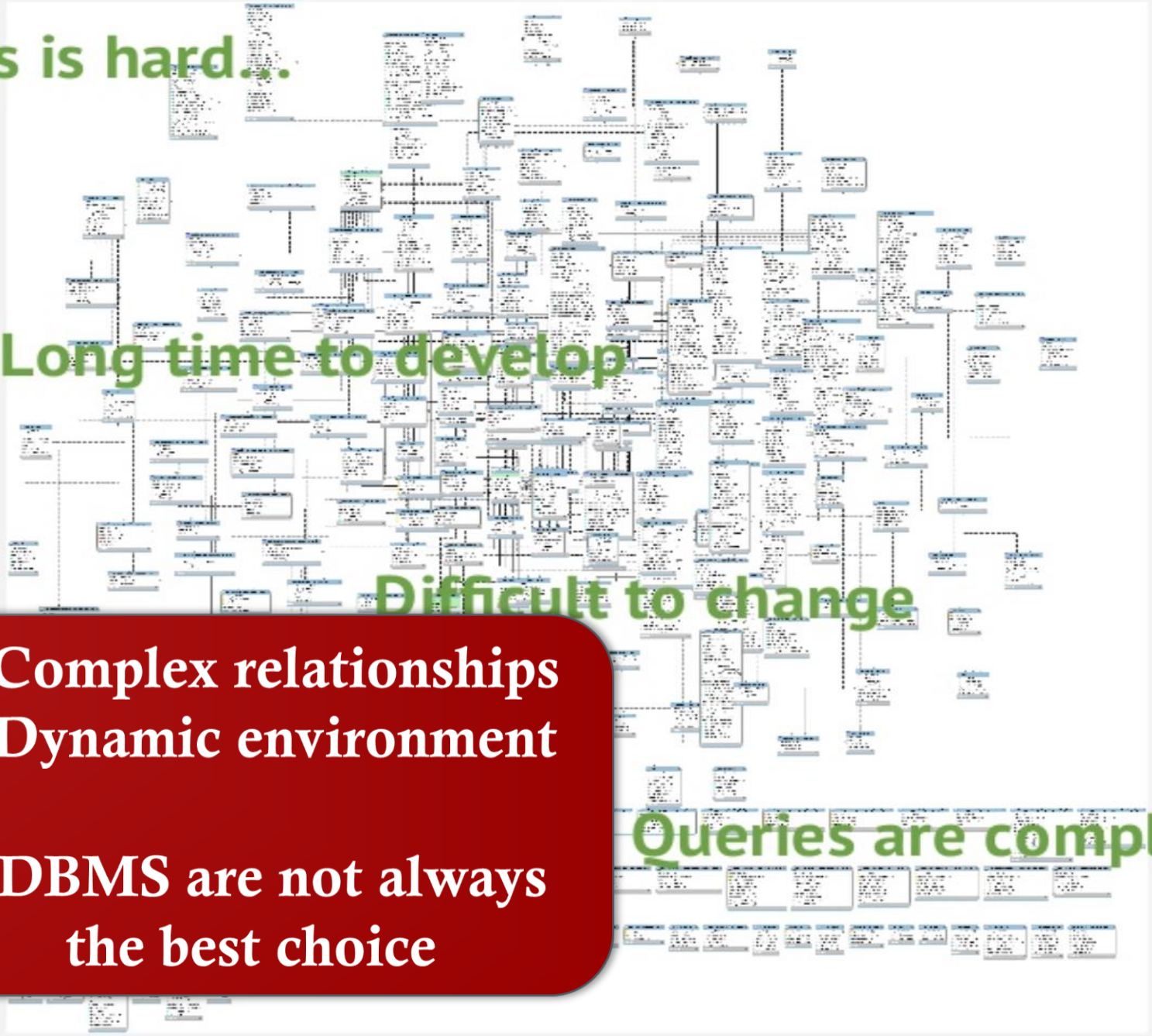
```
{name: "matt",
 pizza: "DiGiorno",
 height: 72,
 boss: 555.555.1212}
```

mongoDB

Data Model Comparison:

Relational DB vs. NoSQL

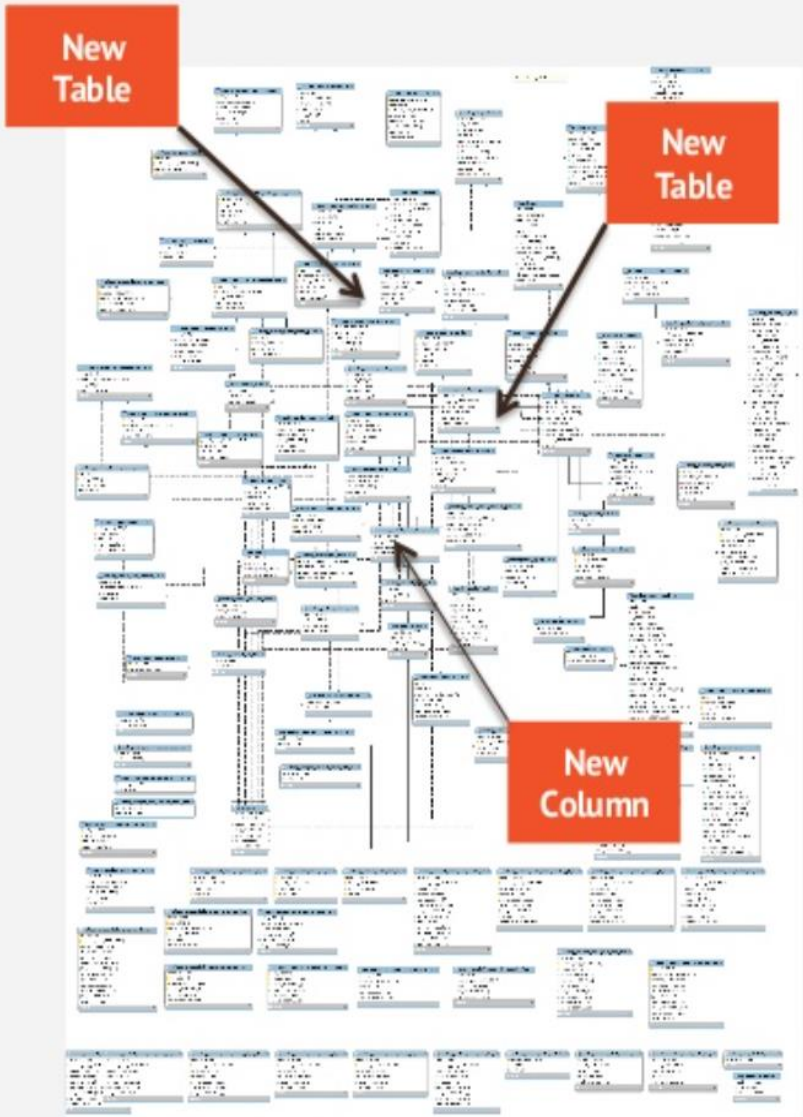
This is hard...

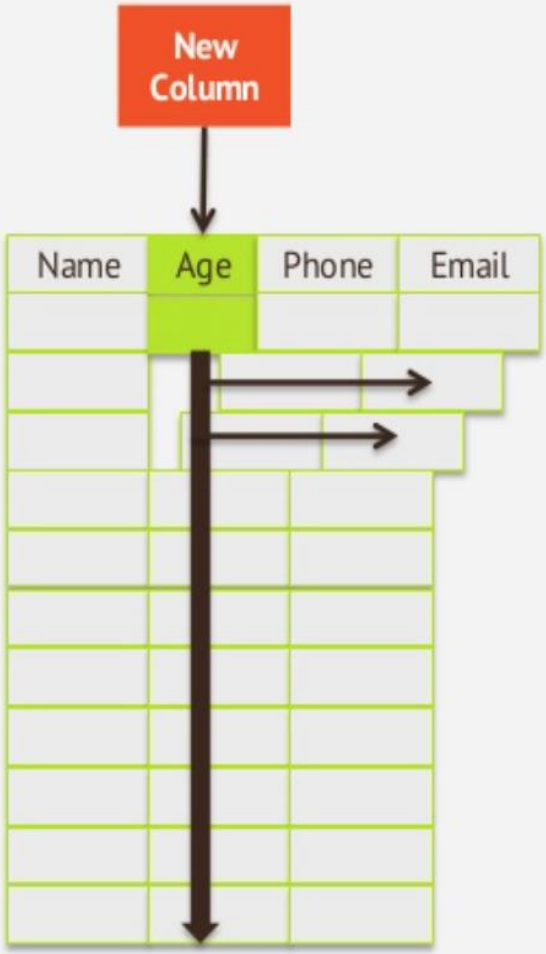


- Complex relationships
- Dynamic environment

RDBMS are not always the best choice

Hard to make changes

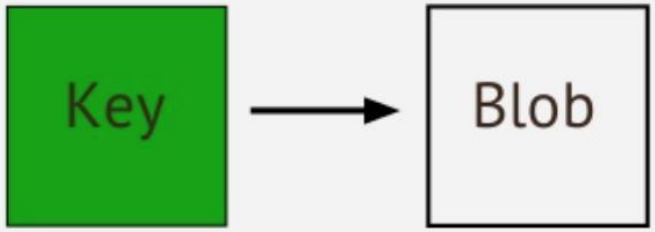




Key-Value Data Model (minimal)

Key → Value store

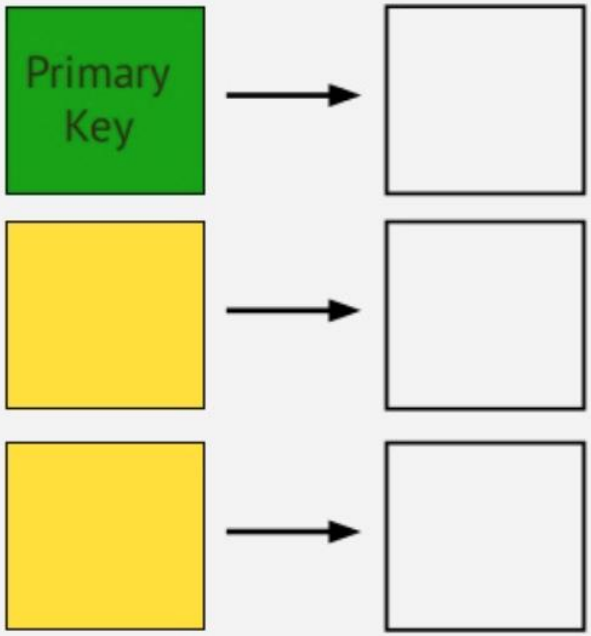
- One-dimensional storage
- Single value is a blob
- Query on key only
- No schema
- Value can be replaced but not updated



Relational Data Model (struct.)

Relational Record

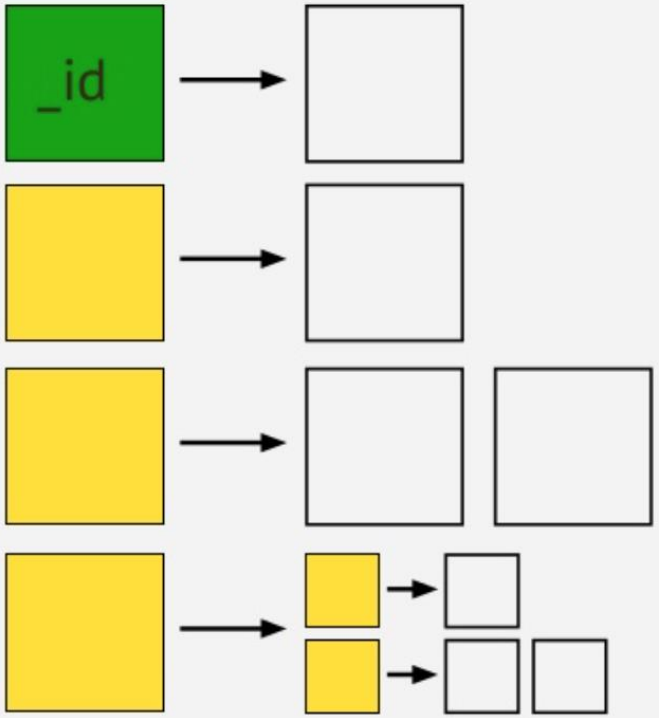
- Two-dimensional storage
- Field contains a single value
- Query on any field
- Very structured schema
- Poor data locality requires many tables, joins, and indexes.



Document Data Model (rich)

MongoDB Document

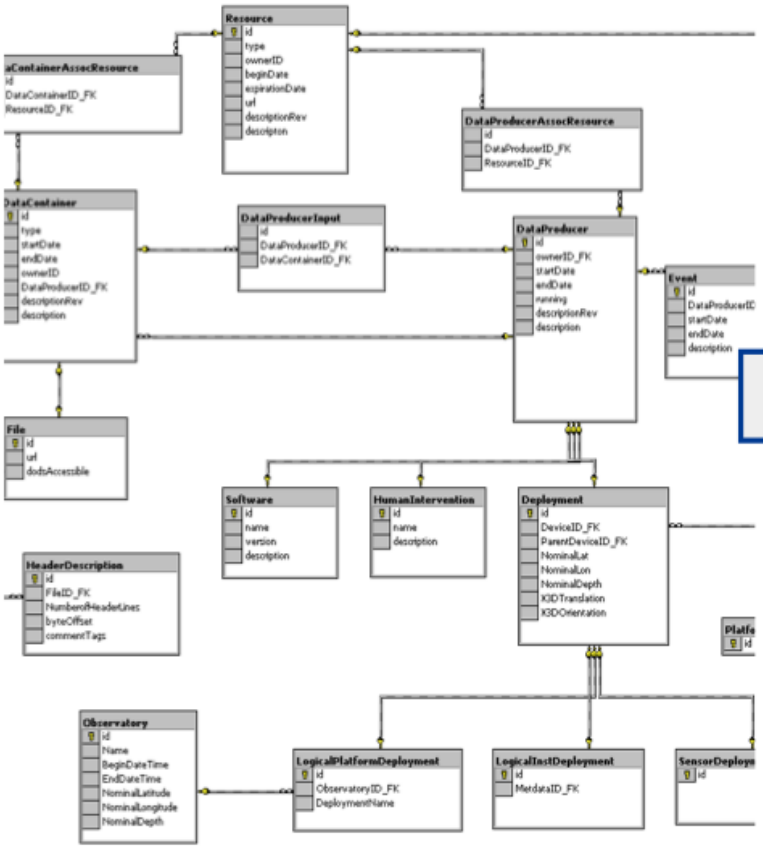
- **N-dimensional** storage
- Field can contain **many** values and **embedded** values
- Query on **any field & level**
- **Flexible** schema
- Optimal data locality requires fewer **indexes** and provides better **performance**



Document vs. Relational Models

- **Relational**
  - Focus on data storage
  - At query time ➔ build your business objects
- **Document**
  - Focus on data usage
  - Maintain your business object in storage

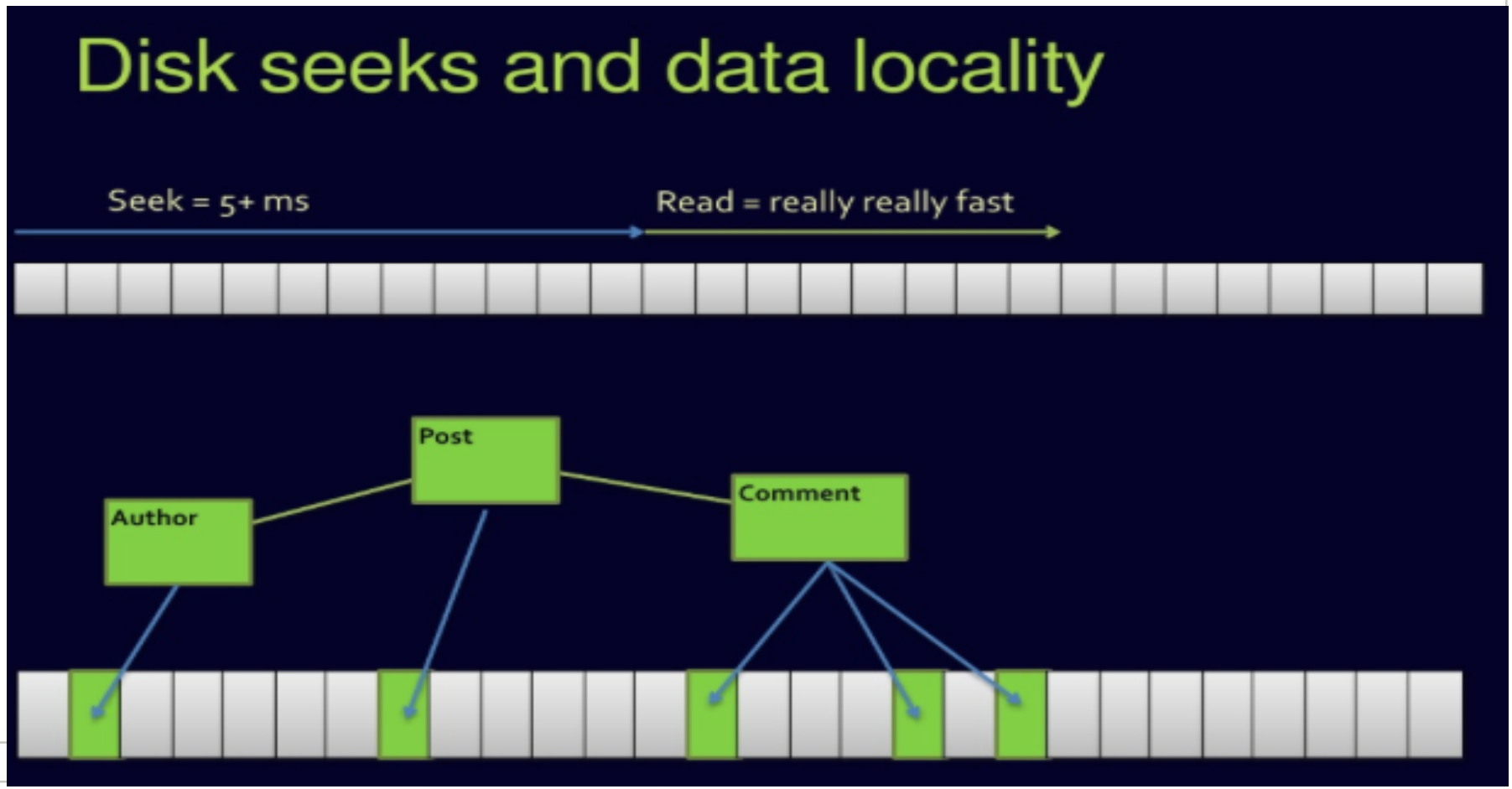
Tradeoff: Normalization vs. Predetermined Usage



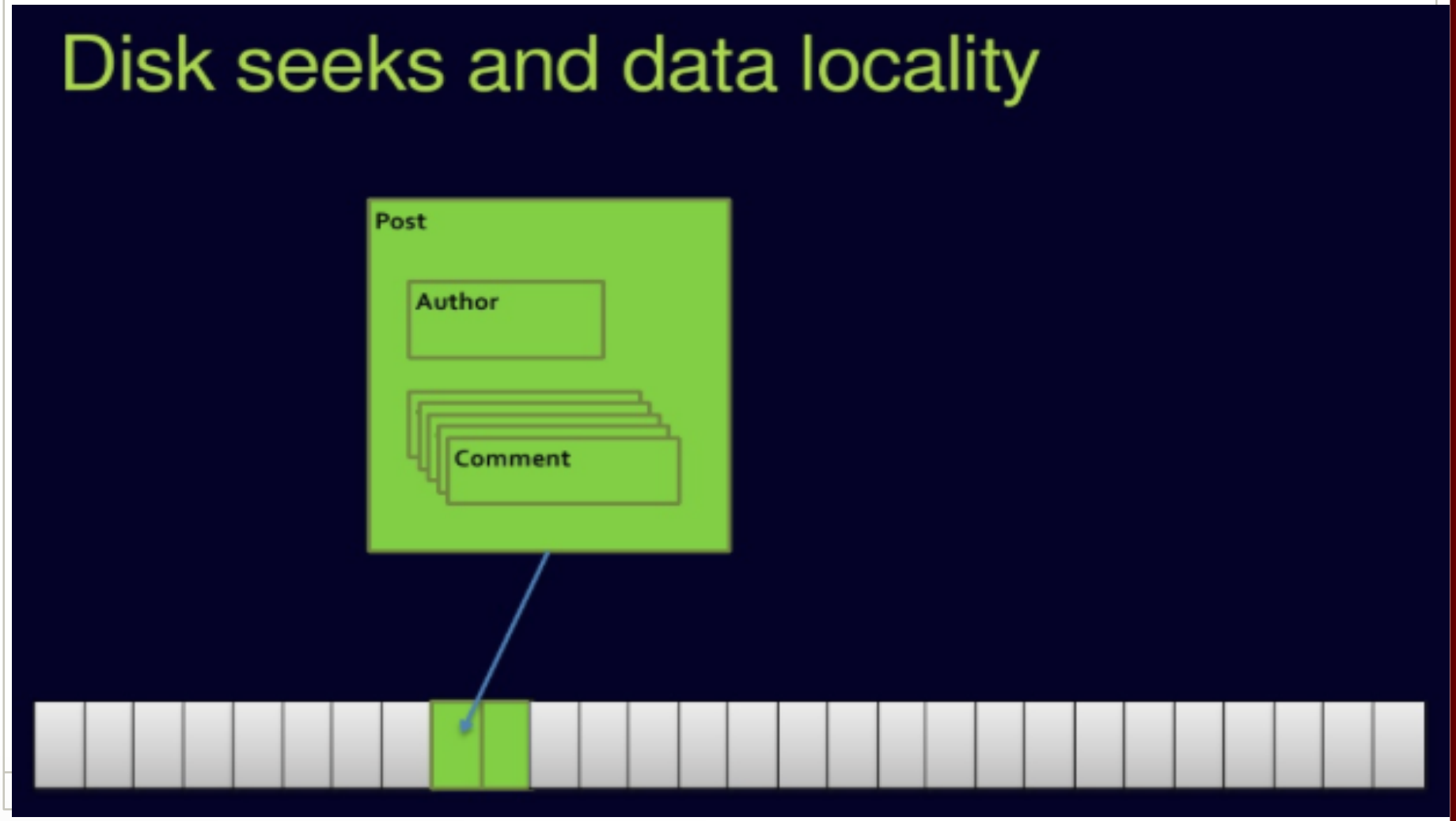

```
{
  title: 'MongoDB',
  contributors: [
    { name: 'Eliot Horowitz',
      email: 'eliot@10gen.com' },
    { name: 'Dwight Merriman',
      email: 'dwight@10gen.com' }
  ],
  model: {
    relational: false,
    awesome: true
  }
}
```

Complex Join Queries

Relational DBs



No Joins in MongoDB



Updating & Querying

CRUD

- **C**reate
  - db.collection.insert(<document>)
  - db.collection.update(<query>, <update>, { upsert: true })
- **R**ead
  - db.collection.find(<query>, <projection>)
  - db.collection.findOne(<query>, <projection>)
- **U**pdate
  - 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

Create "Users" collection explicitly

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

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

Examples

In RDBMS

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

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")
```

```
DROP TABLE users
```

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

Insertion (Basic Operation)

Collection

Document

```
db.users.insert(
                 {
                    name: "sue",
                     age: 26,
                  status: "A",
                  groups: [ "news", "sports" ]
                 }
                )
```

Document

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

insert

Collection

{ name: "al", age: 18, ... }
{ name: "lee", age: 28, ... }
{ name: "jan", age: 21, ... }
{ name: "kai", age: 38, ... }
{ name: "sam", age: 18, ... }
{ name: "mel", age: 38, ... }
{ name: "ryan", age: 31, ... }
{ name: "sue", age: 26, ... }

users

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 documents are inserted at once into inventory

Multi-Document Insertion (Bulk Operation)

- Temporary object in memory to hold your insertions and upload them at once into storage

There is a *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(          ← collection
   { status: "D" }        ← remove criteria
)
```

The following diagram shows the same query in SQL:

```
DELETE FROM users         ← table
WHERE  status = 'D'       ← delete criteria
```

db.users.remove ( ) → Removes all documents from *users* collection

Update

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

Otherwise, it will update only 1st matching document

Equivalent to SQL:

```
UPDATE  users          ← table
SET     status = 'A'   ← update action
WHERE   age > 18       ← update criteria
```

Update (Cont'd)

Two update 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

Query Condition

New doc

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

Because there is no $SET operator;
We replace the document having item = "BE10", 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.

REMINDER

Installation: in your VM

Manual: https://docs.mongodb.com/manual

Data examples:
https://docs.mongodb.com/manual/reference/bios-example-collection/

Online Terminal:
http://www.tutorialspoint.com/mongodb_terminal_online.php