mongoDB

Prof. Jose F. Rodrigues Jr.
University of Sao Paulo, Brazil

Edited and extended from the slides publicly distributed by:
A. Im, G. Cai, H. Tunc, J. Stevens, Y. Barve, S. Hei
Vanderbilt University

http://conteudo.icmc.usp.br/pessoas/junio/Site/index.htm

Content

- Part 2: Advanced features
  - 3: Schema Design
  - 4: Indexes
  - 5: Aggregation
  - 6: Java + MongoDB

3. Schema Design

RDBMS		MongoDB
Database	→	Database
Table	→	Collection
Row	→	Document
Index	→	Index
Join	→	Embedded Document
Foreign Key	→	Reference

Intuition - why databases 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 via page-based memory management mechanism
- Why can't we have the database operating on the same data structures (like classes) as the ones used in programs?
  - That is where mongoDB comes in

Mongo is basically schema-free

- The purpose of schema in SQL is for meeting the requirements of tables and guide the SQL implementation

- Every "*row*" in a database "*table*" is a data structure, much like a "struct" in C, or a "class" in Java. A table is then an array (or list) of such data structures

- So what we design in mongoDB is basically the same way we design a compound data type binding in JSON

There are some patterns

- Embedding (pre-joining)
- Linking

One to One relationship

No relationship

```
zip = {
  _id: 35004,
  city: “ACMAR”,
  loc: [-86, 33],
  pop: 6065,
  State: “AL”
}

Council_person = {
  _id = 1234,
  zip_id = 35004,
  name: “John Doe",
  address: “123 Fake St.”,
  Phone: 123456
}
```

One-to-one relationship

```
zip = {
  _id: 35004 ,
  city: “ACMAR”
  loc: [-86, 33],
  pop: 6065,
  State: “AL”,

  Council_person: {
    _id = 1234,
    name: “John Doe",
    address: “123 Fake St.”,
    Phone: 123456
  }
}
```

http://conteudo.icmc.usp.br/pessoas/junio/Site/index.htm

1:N relationship via Embedding

```
book = {
    title: "MongoDB: The Definitive Guide",
    authors: [ "Kristina Chodorow", "Mike Dirolf" ]
    published_date: ISODate("2010-09-24"),
    pages: 216,
    language: "English",
            publisher: {
                name: "O'Reilly Media",
                founded: "1980",
                location: "CA"
            }
}
```

1:N relationship via Linking (referencing)

```
book = {
    title: "MongoDB: The Definitive Guide",
    authors: [ "Kristina Chodorow", "Mike Dirolf" ]
    published_date: ISODate("2010-09-24"),
    pages: 216,
    language: "English",
    publisher_id: "8798486734"
}
publisher = {
    _id: “8798486734",
    name: "O’Reilly Media",
    founded: "1980",
    location: "CA"
}
```

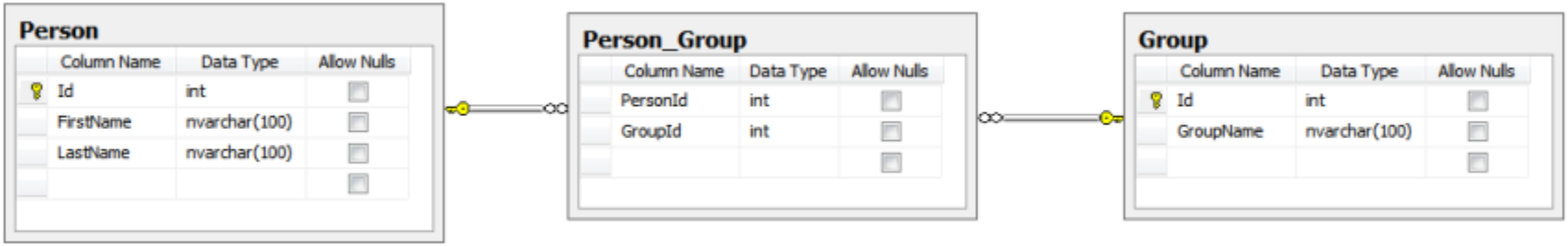
Many to many relationship

- Can put relation in either one of the documents (embedding in one of the documents)
- Unavoidable redundancy
- Possible (probable) inconsistency

- It is also possible via linking
- But, in this case, random access is necessary – and joining is necessary in case one needs all the relationships

Many to many relationship

- Example:




```
1  db.person.insert({
2    "_id": ObjectId("4e54ed9f48dc5922c0094a43"),
3    "firstName": "Joe",
4    "lastName": "Mongo",
5    "groups": [
6      ObjectId("4e54ed9f48dc5922c0094a42"),
7      ObjectId("4e54ed9f48dc5922c0094a41")
8    ]
9  });
10
11 db.person.insert({
12   "_id": ObjectId("4e54ed9f48dc5922c0094a40"),
13   "firstName": "Sally",
14   "lastName": "Mongo",
15   "groups": [
16     ObjectId("4e54ed9f48dc5922c0094a42")
17   ]
18 });
```

```
1  db.groups.insert({
2    "_id": ObjectId("4e54ed9f48dc5922c0094a42"),
3    "groupName": "mongoDB User",
4    "persons": [
5      ObjectId("4e54ed9f48dc5922c0094a43"),
6      ObjectId("4e54ed9f48dc5922c0094a40")
7    ]
8  });
9
10 db.groups.insert({
11   "_id": ObjectId("4e54ed9f48dc5922c0094a41"),
12   "groupName": "mongoDB Administrator",
13   "persons": [
14     ObjectId("4e54ed9f48dc5922c0094a43")
15   ]
16 });
```

Many to many relationship

- Example:

```
// Get all persons in the "mongoDB User" group
db.person.find({"groups": ObjectId("4e54ed9f48dc5922c0094a42")});

// Get all persons in the "mongoDB Administrator" group
db.person.find({"groups": ObjectId("4e54ed9f48dc5922c0094a41")});

// Get all groups for "Joe Mongo"
db.groups.find({"persons": ObjectId("4e54ed9f48dc5922c0094a43")});

// Get all groups for "Sally Mongo"
db.groups.find({"persons": ObjectId("4e54ed9f48dc5922c0094a40")});
```

```
  _id : ObjectId("4e54ed9f48dc5922c0094a40"),
  "firstName": "Sally",
  "lastName": "Mongo",
  "groups": [
    ObjectId("4e54ed9f48dc5922c0094a42")
  ]
});
```

```
  _id : ObjectId("4e54ed9f48dc5922c0094a41"),
  "groupName": "mongoDB Administrator",
  "persons": [
    ObjectId("4e54ed9f48dc5922c0094a43")
  ]
});
```

Many to many relationship

- Example:
- Inserting a reference

1) Find the document to be referenced

```
var doc = db.courses.findOne("name": "Data Bases");
```

2) Insert with _id field

```
db.students.insert("name": "Chris", "courses":[doc]);
```

- Examp
- Inserti

Atenção: precisa usar o findOne() pois ele retorna a cópia de um documento; o find() retorna um iterador para um conjunto de documentos.

1) Find t d

```
var doc = db.courses.findOne("name": "Data Bases");
```

2) Insert with _id field

```
db.students.insert("name": "Chris", "courses":[doc]);
```

Joins

- MongoDB enthusiasts say that it avoids joining by pre-joining (embedding documents). Is it true?
  - Well, not
- Embedding is also supported in RDBMs with a more technical name: **denormalization**
- In many situations, embedding is not applicable:
  - M:N relationships
  - 1:N relationships in which the left side (the 1) is related to other entities

Joins

- When embedding is not the case, it is possible to do linking….and pay the same price as RDBMs pay
- In other cases, when the relationship was not modeled into the data (usual, since MongoDB does not have schema), an explicit join is necessary:
  - **$lookup aggregation**

Joins

- $lookup example

```
db.disciplina.aggregate(
  {$lookup:
    {from: ”professor”,
     localField: “Prof”,
     foreignField: “ProfCPF”,
     as: “Ministrantes”}
  }
)
```

➔ **SELECT ***

FROM disciplina, professor

WHERE diciplina.Prof = Professor.ProfCPF

Checks

- MongoDb also takes checks
- They come in the form of “Document Validation”
- The overall sintax is as follows:

```
db.runCommand( {
   collMod: “<collection_name>",
   validator: <boolean expression>,
   validationAction: “error"|“warn" – warn or issues error
} )
```

Example: ensure that either phone or mail are provided

“NOT NULL”

```
db.runCommand( {
   collMod: "contacts",
   validator: { $or: [ { phone: { $exists: true } }, { email: { $exists: true } } ] },
 validationAction : "warn"
} )
```

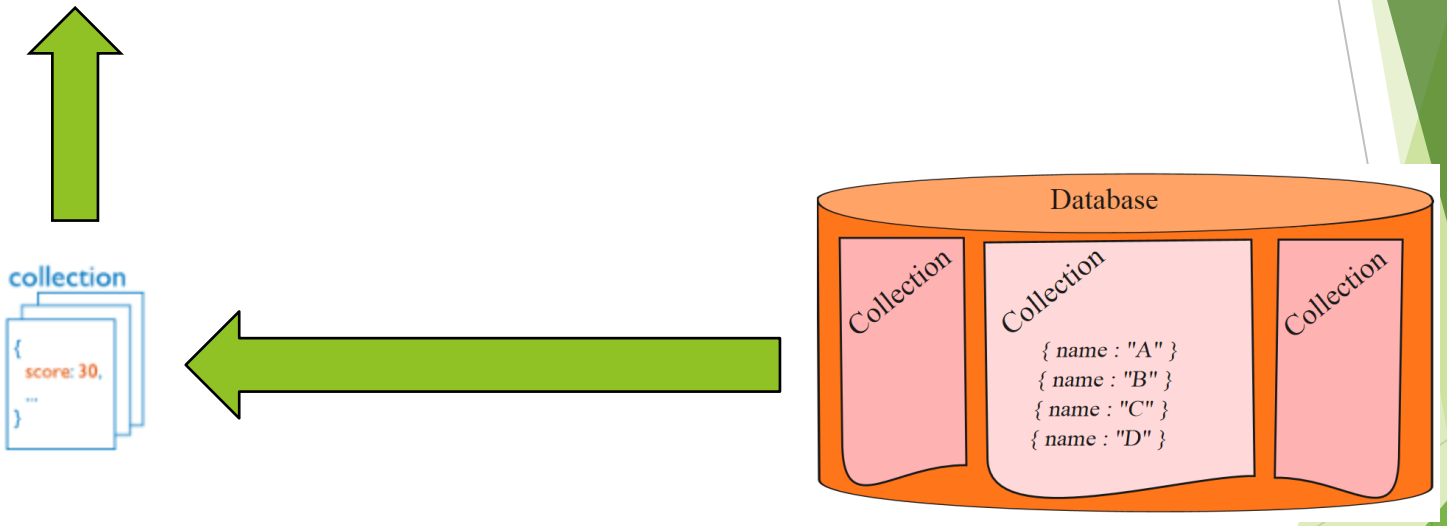
http://conteudo.icmc.usp.br/pessoas/junio/Site/index.htm

4. Index in MongoDB

Before Index

- What does database normally do when we query?
  - MongoDB must scan **every** document
  - Inefficient due to **large volumes** of data

db.users.find({ score: { “$lt” : 30} })



http://conteudo.icmc.usp.br/pessoas/junio/Site/index.htm

Definition of Index

- Definition
  - Indexes are special data structures (B-Trees by default in MogoDB - just as in RDBMSs) that store a small portion of the **collection**'s data set in an easy to traverse form.

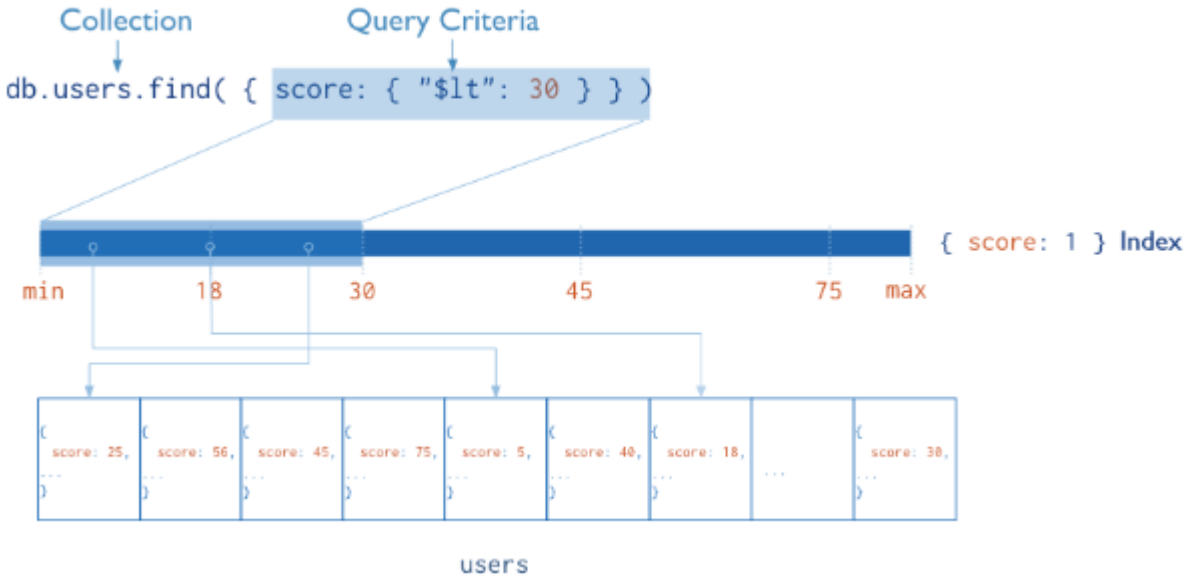


Diagram of a query that uses an index to select

http://conteudo.icmc.usp.br/pessoas/junio/Site/index.htm

Index in MongoDB

Operations

an attribute of the collection

ascending

- **Creation index**
  - db.users.ensureIndex({ score: 1 })
- **Show existing indexes**
  - db.users.getIndexes()
- **Drop index**
  - db.users.dropIndex({score: 1})
- **Explain—Explain**
  - db.users.find().explain()
  - Returns a document that describes the access and its indexes

Index in MongoDB

Types

- **Single Field Indexes**
- **Compound Field Indexes**

- **Single Field Indexes**
  - db.users.ensureIndex({ score: 1 })

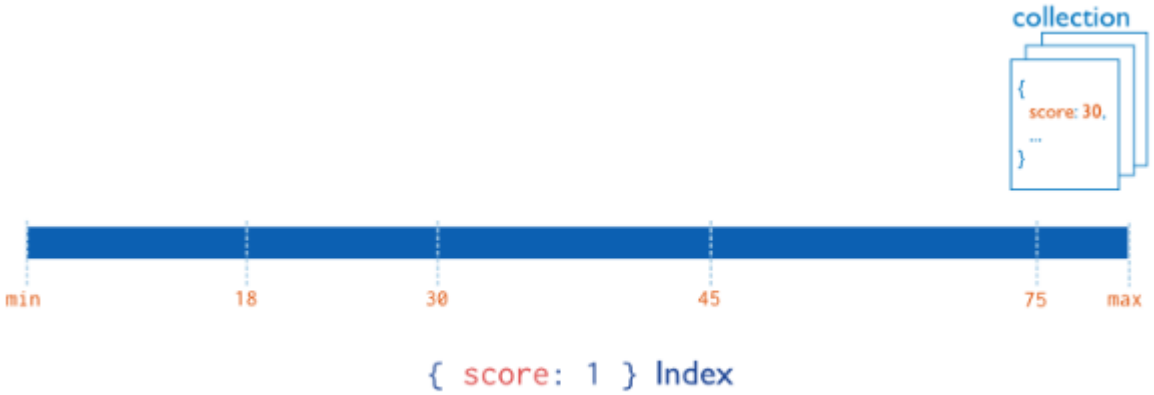


Diagram of an index on the score field (ascending).

http://conteudo.icmc.usp.br/pessoas/junio/Site/index.htm

Index in MongoDB

Types

- **Single Field Indexes**
- **Compound Field Indexes**

ascending descending

- **Compound Field Indexes**
  - db.users.ensureIndex({ userid: 1, score: -1 })

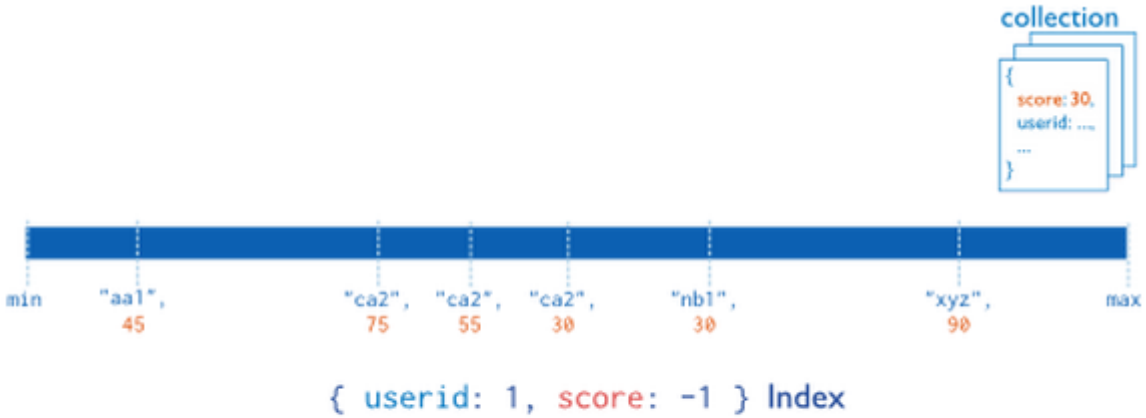


Diagram of a compound index on the `userid` field (ascending) and the `score` field (descending). The index sorts first by the `userid` field and then by the `score` field.

http://conteudo.icmc.usp.br/pessoas/junio/Site/index.htm

5. Aggregation

Aggregation

- Complex syntax
- Tricky enclosing of brackets
- Sum:

```
db.collection.aggregate([
    {$group: {_id:"$group_by_attribute", title_of_new_column:{$sum:"$attribute"}}}
])
```

Exemplo:

```
db.funcionarios.aggregate([
    {$group:{_id:$cargo, soma_sal_por_cargo:{$sum:"$salario"}}}
])
```

- Count:

```
db.collection.aggregate([
    {$group: {_id:"$group_by_attribute", title_of_new_column:{$sum:1}}}
])
```

Exemplo:

```
db.funcionarios.aggregate([
    {$group:{_id:$cargo, soma_function_por_cargo:{$sum:1}}}
])
```

Pipelines

- Modeled on the concept of data processing pipelines.
- Provides:
  - *filters* that operate like queries
  - *document transformations* that modify the form of the output document
- Provides tools for:
  - grouping and sorting by field
  - aggregating the contents of arrays, including arrays of documents
- Can use operators for tasks such as calculating the average or concatenating a string.

```
db.zips.aggregate(
        { $match: { state: "TN" } },
        { $group: {_id: "TN", pop: { $sum: "$pop" } } }
        );
```

SELECT SUM(POP) AS pop
FROM ZIPS
WHERE STATE = “TN”

```
{
    city: "LOS ANGELES",
    loc: [-118.247896, 33.973093],
    pop: 51841,
    state: "CA",
    _id: 90001
}
{
    city: "NEW YORK",
    loc: [-73.996705, 40.74838],
    pop: 18913,
    state: "NY",
    _id: 10001
}
{
    city: "NASHVILLE",
    loc: [-86.778441, 36.167028],
    pop: 1579,
    state: "TN",
    _id: 37201
}
{
    city: "MEMPHIS",
    loc: [-90.047995, 35.144001],
    pop: 4144,
    state: "TN",
    _id: 38103
}
```

$match →

```
{
    city: "NASHVILLE",
    loc: [-86.778441, 36.167028],
    pop: 1579,
    state: "TN",
    _id: 37201
}
{
    city: "MEMPHIS",
    loc: [-90.047995, 35.144001],
    pop: 4144,
    state: "TN",
    _id: 38103
}
```

$group →

```
{
    _id: "TN"
    pop:  5723
}
```

http://conteudo.icmc.usp.br/pessoas/junio/Site/index.htm

Pipelines

- $limit
- $sort

```
db.zips.aggregate(

    {$group: {_id:{state:"$state“, pop:{$sum:"$pop"}}},      -- group by
    {$sort{pop,-1}                                            -- sort descending
    {$limit: 3},                                              -- only 3 first states
  }
)
```

```
SELECT * FROM
  (SELECT state, SUM(POP) AS pop
  FROM ZIPS
  GROUP BY state
ORDER BY pop DESC)
WHERE ROWNUM <= 3
```

Pipelines

- $limit
- $sort

Notice the difference to the former example

```
db.zips.aggregate(

    {$group: {_id:{state:"$state“, pop:{$sum:"$pop"}}},      -- group by
    {$sort{pop,-1}                                            -- sort descending
    {$limit: 3},                                              -- only 3 first states
  }
)
```

```
SELECT * FROM
  (SELECT state, SUM(POP) AS pop
  FROM ZIPS
  GROUP BY state
ORDER BY pop DESC)
WHERE ROWNUM <= 3
```

Pipelines

- $limit
- $sort

Group-by on multiple fields

```
db.zips.aggregate(

{$group: {_id:{state:"$state“, city:“$city"}, pop:{$sum:"$pop"}}}},
   {$sort{pop,-1}                                   -- sort descending
   {$limit: 3},                                     -- only 3 first states-cities
 }
)
```

```
SELECT * FROM
  (SELECT state, city, SUM(POP) AS pop
  FROM ZIPS
  GROUP BY state, city
ORDER BY pop DESC)
WHERE ROWNUM <= 3
```

Pipelines

- Outro exemplo:
- Média salarial por função:

```
db.Salario.aggregate([
  {$group:{'_id':'$funcao',media:{$avg:'$valor'}}}])
```

- Médias salarias maiores que 20000:

```
db.Salario.aggregate([
  {$group:{'_id':'$funcao',media:{$avg:'$valor'}}},
  {$match:{media:{$gt:20000}}}])
```

- Médias salarias maiores que 20000 ordenadas:

```
db.Salario.aggregate([
  {$group:{'_id':'$funcao',media:{$avg:'$valor'}}},
  {$match:{media:{$gt:20000}}},
  {$sort:{media:1}}])
```

- Contagem das médias salarias maiores que 20000 – novo group:

```
db.Salario.aggregate([
  {$group:{'_id':'$funcao',media:{$avg:'$valor'}}},
  {$match:{media:{$gt:20000}}},
  {$sort:{media:1}},
  {$group:{'_id':'$soma',total:{$sum:1}}}
])
```

Single Purpose Aggregation Operations

- Special purpose database commands:
  - returning a count of matching documents
  - returning the distinct values for a field
  - grouping data based on the values of a field
- Aggregate documents from a single collection.

- https://docs.mongodb.com/manual/reference/method/js-collection/

db.zips.distinct("state");

```
{
    city: "LOS ANGELES",
    loc: [-118.247896, 33.973093],
    pop: 51841,
    state: "CA",
    _id: 90001
}
{
    city: "NEW YORK",
    loc: [-73.996705, 40.74838],
    pop: 18913,
    state: "NY",
    _id: 10001
}
{
    city: "NASHVILLE",
    loc: [-86.778441, 36.167028],
    pop: 1579,
    state: "TN",
    _id: 37201
}
{
    city: "MEMPHIS",
    loc: [-90.047995, 35.144001],
    pop: 4144,
    state: "TN",
    _id: 38103
}
```

distinct → ["CA", "NY", "TN"]

```
db.zips.distinct( "state" );
```

```
{
    city: "LOS ANGELES",
    loc: [-118.247896, 33.973093],
```

Other commands:

-count documents of a collection

db.mycollection.count()

-count documents left after a predicate

db.mycollection.count({ "name" : "John", "age" : { "$lte" : 32", "$gte" : 24 } })

```
    city: "MEMPHIS",
    loc: [-90.047995, 35.144001],
    pop: 4144,
    state: "TN",
    _id: 38103
}
```

6. Java + MongoDB

Install

- Add the following libraries to your project :
  - *mongodb-driver*

https://oss.sonatype.org/content/repositories/releases/org/mongodb/mongodb-driver/

  - *mongodb-driver-core*

https://oss.sonatype.org/content/repositories/releases/org/mongodb/mongodb-driver-core

  - *bson*

https://oss.sonatype.org/content/repositories/releases/org/mongodb/bson

➔ *Make sure you use the same release for all of them*

Hello World

```
import com.mongodb.*;
public class App {
 public static void main(String[] args) {
  try {		/*Connect*/
		MongoClient mongo = new MongoClient("localhost", 27017);
		DB db = mongo.getDB("testdb");
		DBCollection table = db.getCollection("user");
		/*Insert*/
		BasicDBObject document = new BasicDBObject();
		document.put("name", "joao");
		document.put("age", 30);
		document.put("createdDate", new Date());
		table.insert(document);
		/*Find*/
		BasicDBObject searchQuery = new BasicDBObject();
		searchQuery.put("name", "joao");
		DBCursor cursor = table.find(searchQuery);
		while (cursor.hasNext()) {
			System.out.println(cursor.next());
		}
 } catch (Exception e) { e.printStackTrace(); }}}
```

Discussion

Cons

- No schema ➔ No project (tempting to start right away)
- No schema ➔ More expensive application-level management later on
- Joins might be necessary after all; RDBMs do it better
- Consistency is in risk when denormalization is accepted by default
- Very limited transaction support → do not put your bank account on MongoDB

For More Information

Resource	Location
MongoDB Downloads	mongodb.com/download
Free Online Training	education.mongodb.com
Webinars and Events	mongodb.com/events
White Papers	mongodb.com/white-papers
Case Studies	mongodb.com/customers
Presentations	mongodb.com/presentations
Documentation	docs.mongodb.org
Additional Info	info@mongodb.com

mongoDB

MongoDB: The Definitive Guide,

By Kristina Chodorow and Mike Dirolf

Published: 9/24/2010

Pages: 216

Language: English

Publisher: O'Reilly Media, CA