NoSQL Overview & MongoDB Basics

Abdu Alawini

University of Illinois at Urbana-Champaign

CS411: Database Systems

April 7, 2024

Some slides were adopted from S. Davidson and Z. Ives with permission

Learning Objectives

After this lecture, we will:

- Introduce the NoSQL paradigm
- Discuss the trade-offs between relational and non-relational(NoSQL) databases
- Introduce MongoDB, a document-oriented database
- Learn MongoDB simple queries.

Why NoSQL?

- Databases are no longer one-size-fits-all
- The needs of modern applications do not always match what relational databases provide.
- Every large web platform (e.g. Google, Facebook, LinkedIn) has developed some sort of custom solution to scale.

"Big Data" is two problems

- The **analysis** problem
  - How to extract useful info, using modeling, ML and stats.
- The **storage** problem
  - How to store and manipulate huge amounts of data to facilitate fast queries and analysis
- Problems with traditional (relational) storage
  - Not flexible
  - Hard to partition, i.e. place different segments on different machines
- NoSQL solutions address these problems.

Types of NoSQL solutions

- **Key-value stores:**

redis SimpleDB DynamoDB amazon web services

- **Column-oriented:**

APACHE HBASE cassandra Google BigTable

- **Document:**

mongoDB® CouchDB

- **Graph:** Neo4J

neo4j

Relational-NoSQL Trade-offs

Fundamentally, there are several different trade-offs

- Schema vs. no schema
  - Schema ➔ performance, no schema ➔ flexibility but parse overhead (can have partial schemas like in XML)
- Replication, data partitioning
  - Replicas mean faster queries, slower (consistent) updates
- Level of abstraction
  - High-level queries – parsing, optimization, etc. – vs. low-level operations
- Consistency
  - What does the database do on concurrent updates, especially when distributed?

Outline

- ✓ NoSQL Introduction
- MongoDB
  - Model and simple queries

Overview of mongoDB®

- MongoDB is an example of a document-oriented NoSQL solution
- The query language is limited, and oriented around "collection" (relation) at a time processing
- The power of the solution lies in the distributed, parallel nature of query processing
  - Replication and sharding

MondoDB Data Model

- A MongoDB deployment hosts several databases
  - A database holds a set of collections
    - A collection holds a set of documents
      - A document is a set of key-value pairs

RDBMS	MongoDB
Table	Collection
Row(s)	JSON Document
Index	Index
Join	Embedding & Linking
Partition	Shard
Partition Key	Shard Key

Basic data types

- Null
- Boolean
- Integer (32- and 64-bit)
- Floating point
- String
- Date
- ObjectId
- Code (JavaScript)
- Array
- Embedded document

Sample Document

```
mydoc =  {
   _id: 1,
   name: { first: “John”, last: “Backus” },
   birthyear: 1924,
  contribs: [ “Fortran”, “ALGOL”, “Backus-Naur Form”, “FP” ],
   awards: [ { award_id: “NMS001”,
                 year: 1975 },
               { award_id: “TA99”,
                 year: 1977} ]
                  }
```

Always indexed, automatically assigned unless provided

Array of documents

Core MongoDB operations

- CRUD: *create*, *read*, *update*, and *delete*
- Insert
  - One at a time: db.people.insert(mydoc)
  - New (version 3.2+):
    db.collection.insertOne(),
    db.collection.insertMany()
- Delete
  - Documents that match some predicate, e.g. to remove the document in the previous slide:
    db.people.deleteOne({"_id": 1})
    db.people.deleteMany({birthyear: 1924})
  - All documents in a collection: db.people.deleteMany()
    - The collection still remains, with indexes
  - Remove a collection (faster): db.people.drop()

Core MongoDB operations, cont.

- Update documents in a collection
  - db.collection.updateOne(), db.collection.updateMany()

```
db.people.updateMany( {birthyear: 1924}, {$set: {birthyear: 1925}})
db.people.updateMany( {birthyear: 1924}, {$set: {type: “Deceased”}})
```

  - $rename operator: change property name.

```
db.people.update({}, { $rename : {“birthyear": “birth" }})
```

Querying

- Use find( ) function and a query document
- Ranges, set inclusion, inequalities using $ conditionals
- Complex queries using $where clause
- Queries return a database cursor
- Meta-operations on cursor include skipping some number of results, limiting the number of results returned, sorting results.

Another sample document

```
d={
    _id : ObjectId(“4c4ba5c0672c685e5e8aabf3”),
    author : “Kevin”,
    date : new Date(“February 2, 2012”),
    text : “About MongoDB...”,
    birthyear:  1980,
    tags : [ "tech", "databases" ]
    }

> db.posts.insert(d)
```

Find

Return entire collection in posts:

```
db.posts.find( )
```

Return posts that match condition (conjunction):

```
db.posts.find({author: “Kevin”, birthyear: 1980})
```

```
{ _id : ObjectId("4c4ba5c0672c685e5e8aabf3"), author : "Kevin",
date : Date(“February 2, 2012”), birthyear:  1980,
 text : "About MongoDB...",  tags : [ "tech", "databases" ]}
```

“Pretty” format

- If you want to be able to read the result:

```
db.posts.find({author: “Kevin”,  birthyear: 1980}).pretty()
```

```
{
      _id : ObjectId("4c4ba5c0672c685e5e8aabf3"),
      author : "Kevin",
      date : Date(“February 2, 2012”),
      birthyear:  1980,
      text : "About MongoDB...",
      tags : [ "tech", "databases" ]
 }
```

Specifying which keys to return

```
db.people.find({}, {name:1, contribs:1})
```

```
{
   _id: 1,
  name: { first: "John", last: "Backus" },
  contribs: [ "Fortran", "ALGOL", "Backus-Naur Form", "FP" ]
}
```

```
db.people.find({}, {_id: 0, name:1})
```

```
{
  name: { first: "John", last: "Backus" }
}
```

Ranges, Negation, OR-clauses

- Comparison operators: $lt, $lte, $gt, $gte
  - db.posts.find({birthyear: {$gte: 1970, $lte: 1990}})
- Negation: $ne
  - db.posts.find({birthyear: {$ne: 1982}})
- Or queries: $in (single key), $or (different keys)
  - db.posts.find({birthyear: {$in: [1982, 1985]}})
  - db.posts.find({$or: [{birthyear: 1982}, {author:“John”}], name:”abdu”})

Embedded/Nested Documents

Embedded/Nested Documents

Embedded data into a single document

Can be viewed as “denormalized” model

Use dot notation to access data within the nested/embedded document

Provides better read performance, allows for all related data to be request and retrieved in one database operation.

Provides for the updates on related data to be done in a single *atomic* write.

Embedded data in a single document

```
{
  _id: <ObjectId1>,
  username: “123xyz”,
  contact: {
              phone: “123-456-7890”,
              email: xyz@example.com
            },
  access: {
              level: 5
              group: “dev”
            }
}
```

Embedded sub-document

Embedded sub-document

Let’s insert a document with embedded data

```
>>> db.sportsCol = db.getCollection("sports")
>>> db.sportsCol.insertOne({"sport" : "tennis", "athlete_first_name" : "Martina",
"athlete_surname" : "Navratilova", "athlete_full_name" : "Martina Navratilova",
"competition_earnings" : { "value" : NumberDecimal("216226089"), "currency" : "USD" },
"number_of_tournaments" : 390, "number_of_titles" : 177, "event" : [ { "type" : "singles",
"number_of_tournaments" : 390, "number_of_titles" : 167 }, { "type" : "doubles",
"number_of_tournaments" : 233, "number_of_titles" : 177, "partner_full_name" : [ "Renáta
Tomanová", "Beatriz Fernández", "Olga Morozova", "Chris Evert" ] } ] })
…
{
    acknowledged : true,
    insertedId : ObjectId(5f3a594ae90262aae835efba)
}
```

See: https://docs.mongodb.com/manual/reference/method/db.collection.insertOne/

Let’s use find and project on embedded data

```
>>> sportsCol.find({"athlete_full_name": "Martina
Navratilova"},{"event.type": 1})

{ "_id": ObjectId(5f3a594ae90262aae835efba), "event" : [ { "type" :
"singles" }, { "type" : "doubles" } ] }
```

See: https://docs.mongodb.com/manual/reference/method/db.collection.find/

Arrays

Arrays

Query to match the entire array

Query to match for a specific element in the array

Query such that either a single array element meets these condition or any combination of array elements meets the conditions

Query for an element by the array index position

Query to match arrays of a given size

Let’s insert some data with arrays!

```
>>> db.inventory.drop()
>>> db.inventory.insertMany([
   { item: "journal", qty: 25, tags: ["blank", "red"], dim_cm: [ 14, 21 ] },
   { item: "notebook", qty: 50, tags: ["red", "blank"], dim_cm: [ 14, 21 ] },
   { item: "paper", qty: 100, tags: ["red", "blank", "plain"], dim_cm: [ 14, 21 ] },
   { item: "planner", qty: 75, tags: ["blank", "red"], dim_cm: [ 22.85, 30 ] },
   { item: "postcard", qty: 45, tags: ["blue"], dim_cm: [ 10, 15.25 ] }
]);
…
{
    acknowledged : true,
    insertedId : ObjectId(5f3b939f63a92a8719c01239)
}
```

See: https://docs.mongodb.com/manual/reference/method/db.collection.insertMany/

Let’s match a complete exact array

```
>>> db.inventory.find( { tags: ["red", "blank"] } )

{ "_id" : ObjectId("5f3b939f63a92a8719c01236"), "item" : "notebook", "qty" : 50,
"tags" : [ "red", "blank" ], "dim_cm" : [ 14, 21 ] }
```

See: https://docs.mongodb.com/manual/reference/method/db.collection.find/

Let’s match elements of an array using $all

```
>>> db.inventory.find( { tags: ["red", "blank"] } )

{ "_id" : ObjectId("5f3b939f63a92a8719c01236"), "item" : "notebook", "qty" : 50,
"tags" : [ "red", "blank" ], "dim_cm" : [ 14, 21 ] }
```

See: https://docs.mongodb.com/manual/reference/method/db.collection.find/

Let's match a complete exact array

```
>>> db.inventory.find( { tags: {$all:["red", "blank"] }} )

{ "_id" : ObjectId("5f3b939f63a92a8719c01236"), "item" : "notebook", "qty" : 50,
"tags" : [ "red", "blank" ], "dim_cm" : [ 14, 21 ] }
```

See: https://docs.mongodb.com/manual/reference/method/db.collection.find/

Let’s match a specific element in the array

```
>>> db.inventory.find( { tags: "red" } )

{ "_id" : ObjectId("5f3b939f63a92a8719c01235"), "item" : "journal", "qty" : 25,
"tags" : [ "blank", "red" ], "dim_cm" : [ 14, 21 ] }

{ "_id" : ObjectId("5f3b939f63a92a8719c01236"), "item" : "notebook", "qty" : 50,
"tags" : [ "red", "blank" ], "dim_cm" : [ 14, 21 ] }

{ "_id" : ObjectId("5f3b939f63a92a8719c01237"), "item" : "paper", "qty" : 100,
"tags" : [ "red", "blank", "plain" ], "dim_cm" : [ 14, 21 ] }

{ "_id" : ObjectId("5f3b939f63a92a8719c01238"), "item" : "planner", "qty" : 75,
"tags" : [ "blank", "red" ], "dim_cm" : [ 22.85, 30 ] }
```

See: https://docs.mongodb.com/manual/reference/method/db.collection.find/

Let’s match on conditions for the array

```
>>> db.inventory.find( { dim_cm: { $gt: 15, $lt: 20 } } )

{ "_id" : ObjectId("5f3b939f63a92a8719c01235"), "item" : "journal", "qty" : 25,
"tags" : [ "blank", "red" ], "dim_cm" : [ 14, 21 ] }

{ "_id" : ObjectId("5f3b939f63a92a8719c01236"), "item" : "notebook", "qty" : 50,
"tags" : [ "red", "blank" ], "dim_cm" : [ 14, 21 ] }

{ "_id" : ObjectId("5f3b939f63a92a8719c01237"), "item" : "paper", "qty" : 100,
"tags" : [ "red", "blank", "plain" ], "dim_cm" : [ 14, 21 ] }

{ "_id" : ObjectId("5f3b939f63a92a8719c01239"), "item" : "postcard", "qty" : 45,
"tags" : [ "blue" ], "dim_cm" : [ 10, 15.25 ] }
```

See: https://docs.mongodb.com/manual/reference/method/db.collection.find/

Let’s match by array index position

```
>>> db.inventory.find( { “dim_cm.1”: { $gt: 25 } } )

{ "_id" : ObjectId("5f3b939f63a92a8719c01238"), "item" : "planner", "qty" : 75,
"tags" : [ "blank", "red" ], "dim_cm" : [ 22.85, 30 ] }

>>> db.inventory.find( { “dim_cm.0”: { $lt: 14 } } )

{ "_id" : ObjectId("5f3b939f63a92a8719c01239"), "item" : "postcard", "qty" : 45,
"tags" : [ "blue" ], "dim_cm" : [ 10, 15.25 ] }
```

See: https://docs.mongodb.com/manual/reference/method/db.collection.find/

Let’s match by the size/length of the array

```
>>> db.inventory.find( { "tags": { $size: 3 } } )

{ "_id" : ObjectId("5f3b939f63a92a8719c01237"), "item" : "paper", "qty" : 100,
"tags" : [ "red", "blank", "plain" ], "dim_cm" : [ 14, 21 ] }
```

See: https://docs.mongodb.com/manual/reference/method/db.collection.find/

Cursors

Cursors

MongoDB queries return a cursor, a pointer to the result set of documents

Cursors can be iterated through to retrieve the results

10 minutes of inactivity before timing out

Various options available within the MongoDB Shell or via MongoDB's drivers

count(), limit(), skip(), sort(), etc.

Let’s insert a document

```
>>> db.topProfs.insertOne({  _id: 1, name: { first: "John", last: "Backus" }, birthyear: 1924,
contribs: [ "Fortran", "ALGOL", "Backus Naur Form", "FP" ], awards: [ {title:  "National Medal
of Science" , by: "National Science Foundation", year: 1975 }, {title:  "Turing Award", by:
"ACM" , year: 1977} , {title:  "Career Award", by: "NIH" , year: 1980} , {title:  "Career
Success Award", by: "NASA", year: 1982}]})
…
{
    acknowledged : true,
    insertedId : 1
}
```

See: https://docs.mongodb.com/manual/reference/method/db.collection.insertOne/

Let's insert a document

```
>>> db.topProfs.insertOne({  _id: 2, name: { first: "Abdu", last: "Alawini" }, birthyear: 2000,
contribs: [ "SQL", "MongoDB", "Neo4J", "CoffeeDB" ], awards: [ {title:  "National Medal of
Science" , by: "National Science Foundation", year: 2001 }, {title:  "Turing Award", by: "ACM"
, year: 2002} , {title:  "Career Award", by: "NIH" , year: 2003} , {title:  "Career Success
Award", by: "NASA", year: 2004}]})
…
{
    acknowledged : true,
    insertedId : 2
}
```

See: https://docs.mongodb.com/manual/reference/method/db.collection.insertOne/

Let’s insert a document

```
>>> db.topProfs.insertOne({  _id: 3, name: { first: "John", last: "McAlawini" }, birthyear:
2002, contribs: [ "Java", "JS" ], awards: [ {title:  "National Medal of Science" , by:
"National Science Foundation", year: 2012 }, {title:  "Turing Award", by: "ACM" , year: 2015} ,
{title:  "Career Award", by: "NIH" , year: 2017} , {title:  "Career Success Award", by: "NASA",
year: 2020}]})
…
{
    acknowledged : true,
    insertedId : 3
}
```

See: https://docs.mongodb.com/manual/reference/method/db.collection.insertOne/

Limits, Skips, Sort, Count

- db.topProfs.find().limit(2)
  - Limits the number of results to 2
- db. topProfs.find().skip(1)
  - Skips the first result and returns the rest
- db. topProfs.find().sort({birthyear:1, _id: -1})
  - Sorts by birthyear ascending (1) and _id descending (-1)
- db. topProfs.find({birthyear:{$gt: 1999}}).count()
  - Counts the number of documents in the awards collection matching the find(…)

Cursor methods

- **.toArray** returns an array that contains all documents from a cursor
- **.forEach** applies a JavaScript function to each document from the cursor (similar to .map)

Let’s try the toArray() cursor method

```
>>> nsf= db.topProfs.find({"awards.by": "National Science
  Foundation"}).toArray();
>>>nsf
>>> if (nsf.length >0) {printjson (nsf[0])};

How many documents returned by the last command?
```

See: https://docs.mongodb.com/manual/reference/method/db.collection.insertOne/

Let's try the forEach() cursor method

```
>>>db.topProfs.find().forEach(
  function(myDoc) {
    printjson ("Professor: " + myDoc.name.last);
   });
```

```
How many names returned?
```

See: https://docs.mongodb.com/manual/reference/method/db.collection.insertOne/