mongodb.org

History

- mongoDB = "Hu**mongo**us DB"
  - Open-source
  - Document-based
  - "High performance, high availability"
  - Automatic scaling (on Atlas, sharding manually)
  - C-P on CAP

Motivations

- Problems with SQL
  - Rigid schema
  - Not easily scalable (designed for 90's technology)
  - Requires unintuitive joins
- Perks of mongoDB
  - Easy interface with common languages – common drivers(Java, Javascript, PHP, 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

Data Model

Data Model

- Document-Based BSON (max size of16 MB)
- Documents are in BSON format, consisting of field-value pairs

```
{
  "name": "Tariq",
  "age": 25,
  "courses": ["DBA", "Docker", "Hadoop"]
}
```

Data Model

- Each document stored in a collection
- Collections
  - Each collection has its own index set (why)
  - Like tables
  - Documents do not need to have a uniform schema (like Hbase)

-docs.mongodb.org/manual/

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/

```
{
  "name": "Ali",
  "age": 25,
  "city": "Lahore",
  "profile_picture": "BASE64_STRING_HERE"
}
```

JSON is human-readable

Binary JSON (Decoded)

\x02 name \x00 Ali\x00
\x10 age \x00 25\x00
\x02 city \x00 Lahore\x00
\x05 profile_picture \x00 <binary data> \x00

BSON

BSON

- Binary-encoded serialization of JSON-like docs
- Allows “referencing”
- But MongoDB does not enforce foreign keys like SQL.
- Referencing is allowed but not enforced

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/

BSON

- Embedded structure reduces need for joins
- Goals
  - Lightweight
  - Traversable
  - Efficient (decoding and encoding)

http://bsonspec.org/

id	name
1	Tariq

id	user_id	city	country
101	1	Lahore	Pakistan
102	1	Karachi	Pakistan

```
SELECT *
FROM users
JOIN addresses ON users.id =
addresses.user_id
WHERE users.id = 1;
```

```
{
  "_id": 1,
  "name": "Tariq",
  "addresses": [
    { "city": "Lahore", "country": "Pakistan" },
    { "city": "Karachi", "country": "Pakistan" }
  ]
}
```

```
db.users.find({ _id: 1 });
```

BSON Example

JSON

```
{
  "product": "Laptop",
  "price": 120000,
  "inStock": true,
  "tags": ["electronics", "computer"],
  "image": "BASE64_DATA"
}
```

MongoDB style BSON

```
{
  product: "Laptop",            // String
  price: NumberInt(120000),     // int32
  inStock: true,                // boolean
  tags: ["electronics", "computer"], // array
  image: BinData(0, "\x89PNG\x0D\x0A...")
// raw binary (PNG bytes)
}
```

JSON vs BSON

Feature	**JSON**	**BSON**
Full Form	JavaScript Object Notation	Binary JSON
Format	Text (human-readable)	Binary (machine-readable)
Speed	Slower to parse	Faster to parse
Used By	Web APIs, config files	MongoDB internal storage
Type Support	Limited → string, number, boolean, array, object, null	Rich → objectId, date, int32, int64, double, binary, regex, timestamp, decimal128, etc.
Data Size	Larger	Smaller & more compact
Order of Fields	Not guaranteed	Preserved
Supports Referencing?	Conceptually yes, but no native type	Yes (ObjectId is ideal for references)
Numeric Types	Only one “number” type	Many numeric types: int32, int64, double, decimal128
Null vs Undefined	Only null	null + undefined

_id

The _id Field

- By default, each document contains an _id field.
  - Serves as primary key for collection.
  - Unique, immutable, non-array type.
  - Default: ObjectId - “small (12 bytes), likely unique (across machines), fast to generate, and ordered.”

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

Operators

Logical Operators

Operator	Description
$and	Logical AND of multiple expressions
$or	Logical OR
$not	Negates a condition
$nor	Logical NOR

Comparison Operators

Operator	Description
$eq	Equal to
$ne	Not equal
$gt	Greater than
$gte	Greater than or equal
$lt	Less than
$lte	Less than or equal
$in	Value IN array
$nin	Value NOT IN array
$cmp	Compare two values

```
{ $cmp: [ value1, value2 ] }
```

Element Operators

Operator	Description
$exists	Checks if field exists
$type	Checks BSON data type
$jsonSchema	Schema validation

```
db.users.find({
  $jsonSchema: {
    bsonType: "object",
    properties: {
      name: { bsonType: "string" }
    }
  }
})
```

This returns only documents where `name` is a string.

Evaluation Operators

Operator	Description
$regex	Regular expression
$expr	Use aggregation expressions in match
$mod	Modulo operation
$text	Text search
$where	JavaScript expression (slow, avoid)

```
db.tests.find({
  $expr: { $gt: ["$score", "$maxScore"] }
})
```

```
db.employees.find({
  $expr: {
    $gt: ["$salary", { $multiply: [2, "$experience"] }]
  }
})
```

Array Query Operators

Operator	Description
$all	Match all elements in array
$elemMatch	Match array element that matches conditions
$size	Match array size
$slice	Limit array elements (projection)

```
db.products.find(
  {},
  { tags: { $slice: 3 } }
)
```

Return first 3 items from the `tags` array:

Projection Operators

Operator	Description
$	Get the 1st matching element
$slice	Return first N array elements
$meta	Metadata (like textScore)
$elemMatch	Project array element that matches

```
{
  "_id": 1,
  "name": "Ali",
  "grades": [
    { "subject": "Math", "score": 85 },
    { "subject": "English", "score": 90 },
    { "subject": "Science", "score": 78 }
  ]
}
```

We want **only the English grade** for each student:

```
db.students.find(
  { "grades.subject": "English" },
  { "grades.$": 1, name: 1 }
)
```

Field Update Operators

Operator	Description
$set	Set field value
$unset	Remove field
$rename	Rename field
$inc	Increase/decrease number
$mul	Multiply
$currentDate	Set current date

Array Update Operators

Operator	Description
$push	Push into array
$push + $each	Push multiple
$addToSet	Push only if not exists
$pop	Pop from array
$pull	Remove elements
$pullAll	Remove all matching
$position	Insert at position
$sort	Sort array elements (when pushing)
$slice	Trim array

Update Operators for Documents

Operator	Description
$setOnInsert	Set field only on insert (not update)
$min	Keep smaller value
$max	Keep larger value

```
{ "_id": 1, "name": "Ali", "role": "user" }


db.users.updateOne(
  { _id: 2 },                    // not found
  {
    $set: { lastLogin: new Date() },
    $setOnInsert: { name: "Sara", role: "admin" }
  },
  { upsert: true }
)
```

```
{
  "_id": 2,
  "name": "Sara",        // from $setOnInsert
  "role": "admin",       // from $setOnInsert
  "lastLogin": ISODate("2025-12-03T...")  // from
$set
}
```

Aggregation Pipeline Stages

Operator	Description
$match	Filter documents
$project	Select fields
$group	Group and aggregate
$sort	Sort
$limit	Limit
$skip	Skip
$lookup	Join other collection
$unwind	Deconstruct array
$addFields	Add new fields
$set	Alias of $addFields
$unset	Remove field
$replaceRoot	Replace root document
$merge	Write results to a collection
$facet	Multi-branch pipelines
$bucket	Group ranges
$bucketAuto	Auto ranges
$count	Count documents

Aggregation Expression Operators

Operator	Description
$sum	Sum
$avg	Average
$min	Minimum
$max	Maximum
$add	Add numbers
$subtract	Subtract
$multiply	Multiply
$divide	Divide
$mod	Modulo

Aggregation String Expressions

Operator	Description
$concat	Concatenate
$substr	Substring
$toUpper	Uppercase
$toLower	Lowercase
$split	Split string
$trim	Trim
$ltrim	Left trim
$rtrim	Right trim

Aggregation Date Expressions

Operator	Description
$year, **$month**, **$dayOfWeek**, etc.	Extract date parts
$dateToString	Format date
$dateFromString	Parse date

Aggregation Boolean Expressions

Operator	Description
$and	Logical AND
$or	OR
$not	NOT
$cond	If-then-else

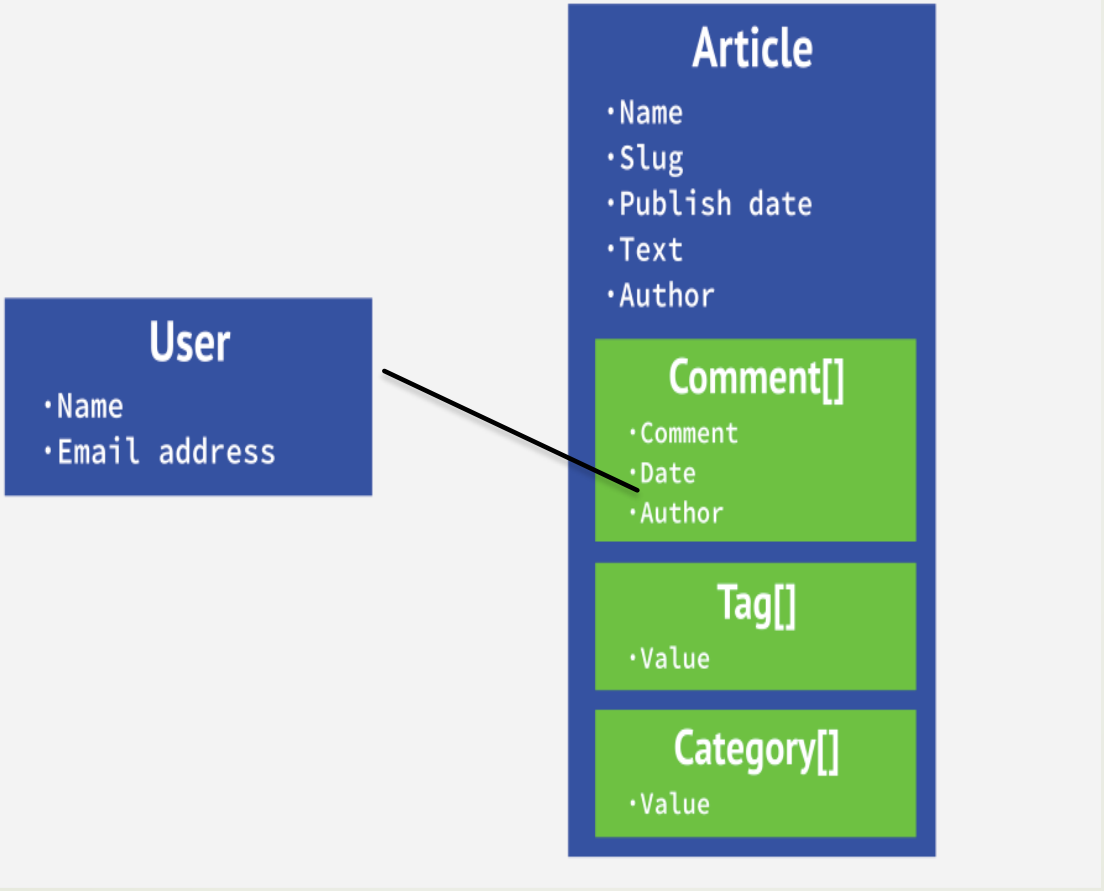
Aggregation Misc Expressions

Operator	Description
$comment	Add comment to query
$hint	Force index
$natural	Natural order

Embedding

Linking

Embedding & Linking



One to One relationship

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

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

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

 council_person: {
 name: “John Doe",
 address: “123 Fake St.”,
 Phone: 123456
}

}
```

One to many relationship - 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"
}
}
```

One to many relationship – Linking

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

Linking vs. Embedding

- Embedding is like pre-joining data
- Document level operations are easy for the server to handle
- Embed when the "many" objects always appear with their parents.
- Linking when you need more flexibility

```
{ "_id": ObjectId("u1"), "name": "Ali" }
{ "_id": ObjectId("u2"), "name": "Sara" }
```

```
{ "_id": ObjectId("o1"), "product": "Laptop", "userId": ObjectId("u1") }
{ "_id": ObjectId("o2"), "product": "Mouse", "userId": ObjectId("u1") }
{ "_id": ObjectId("o3"), "product": "Keyboard", "userId": ObjectId("u2") }
```

```
db.orders.aggregate([
  {
    $lookup: {
      from: "users",
      localField: "userId",
      foreignField: "_id",
      as: "user"
    }
  }
])
```

```
{
  "_id": "o1",
  "product": "Laptop",
  "userId": "u1",
  "user": [{ "_id": "u1", "name": "Ali" }]
}
```

```
{
  "_id": 101,
  "title": "MongoDB Tutorial",
  "content": "Learning MongoDB is fun!",
  "comments": [
    { "user": "Sara", "message": "Great post!", "date": ISODate("2025-12-03") },
    { "user": "Ali", "message": "Very helpful.", "date": ISODate("2025-12-03") }
  ]
}
```

```
{
  "_id": 1,
  "name": "Ali",
  "addresses": [
    { "city": "Lahore", "zip": 54000 },
    { "city": "Karachi", "zip": 74000 }
  ]
}
```

Many to many relationship

- Can put relation in either one of the documents (embedding in one of the documents)
- Focus how data is accessed queried

Example

```
book = {
    title: "MongoDB: The Definitive Guide",
    authors : [
        { _id: "kchodorow", name: "Kristina Chodorow" },
        { _id: "mdirolf", name: "Mike Dirolf" }
    ]
    published_date: ISODate("2010-09-24"),
    pages: 216,
    language: "English"
}

author = {
    _id: "kchodorow",
    name: "Kristina Chodorow",
    hometown: "New York"
}

db.books.find( { authors.name : "Kristina Chodorow" } )
```

Example 3

- Book can be checked out by one student at a time
- Student can check out many books

Modeling Checkouts

```
student = {
    _id: "joe"
    name: "Joe Bookreader",
    join_date: ISODate("2011-10-15"),
    address: { ... }
}

book = {
    _id: "123456789"
    title: "MongoDB: The Definitive Guide",
    authors: [ "Kristina Chodorow", "Mike Dirolf" ],
    ...
}
```

Modeling Checkouts

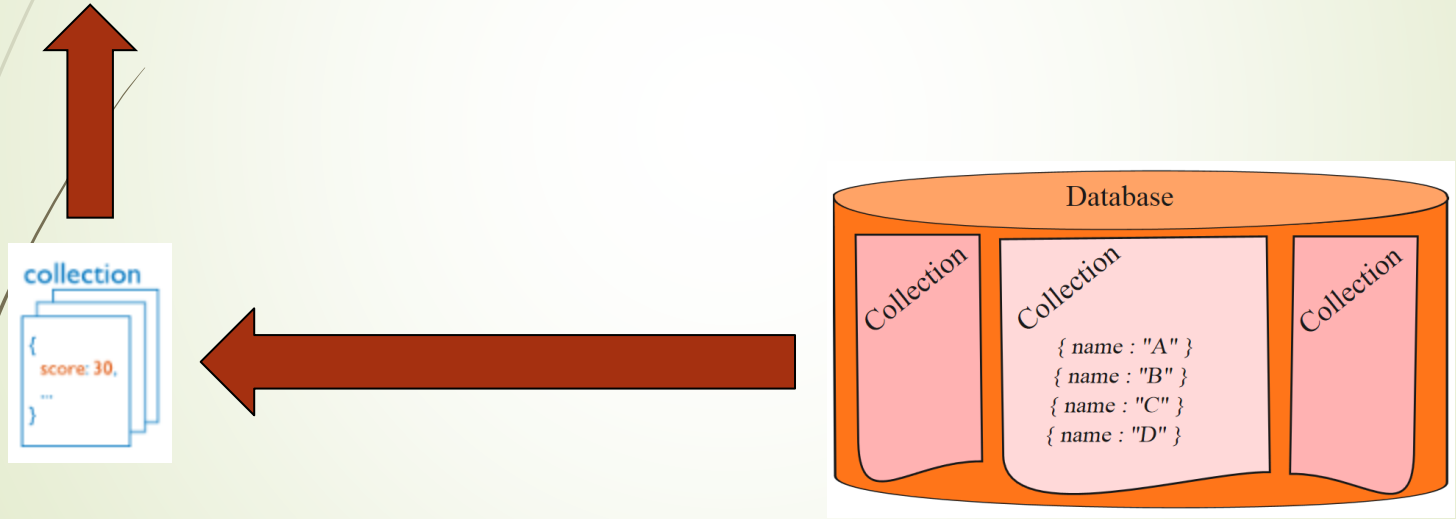
```
student = {
    _id: "joe"
    name: "Joe Bookreader",
    join_date: ISODate("2011-10-15"),
    address: { ... },
    checked_out: [
        { _id: "123456789", checked_out: "2012-10-15" },
        { _id: "987654321", checked_out: "2012-09-12" },
        ...
    ]
}
```

Indexing

Before Index

- What does database normally do when we query?
  - MongoDB must scan **every** document.
  - Inefficient because process **large volume** of data

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



Definition of Index

- **Indexes are special data structures that store a small portion of the collection's data set in an easy to traverse form.**

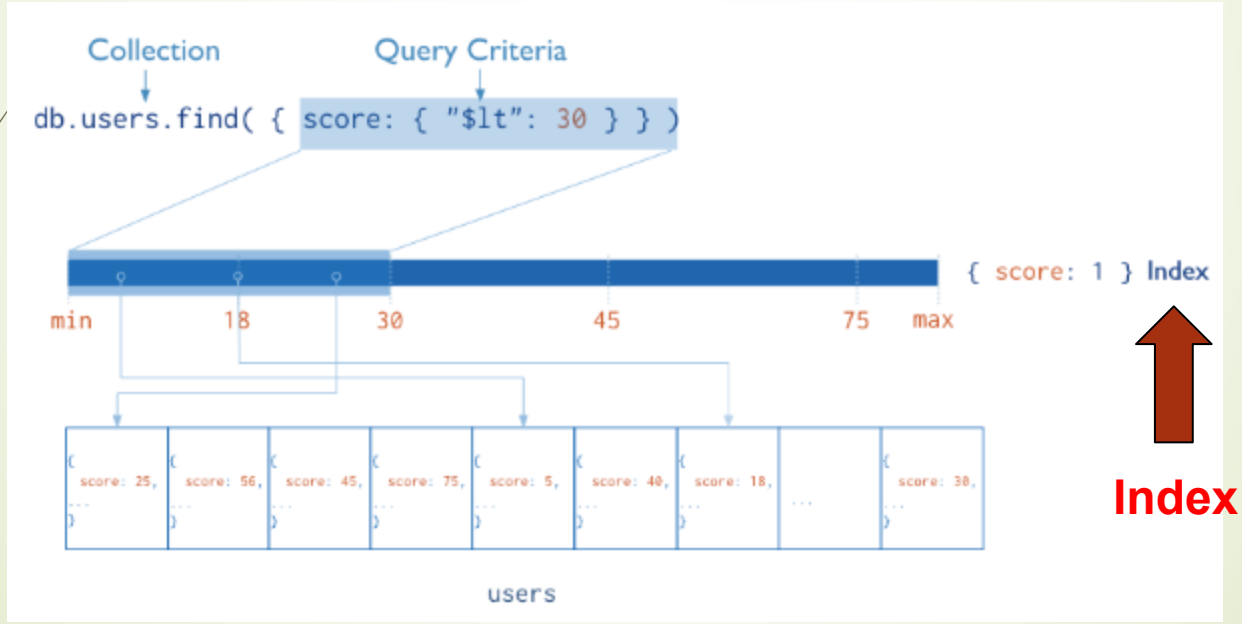


Diagram of a query that uses an index to select

Index in MongoDB

Operations

- **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()
- **Hint**
  - db.users.find().hint({score: 1})

Index in MongoDB

Types

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

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

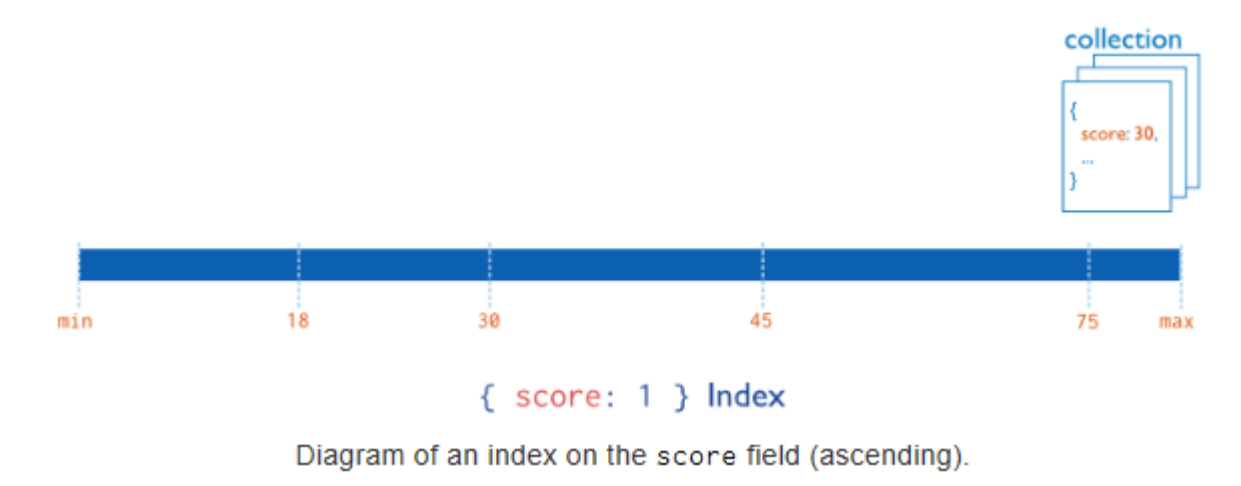


Diagram of an index on the score field (ascending).

Index in MongoDB

Types

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

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

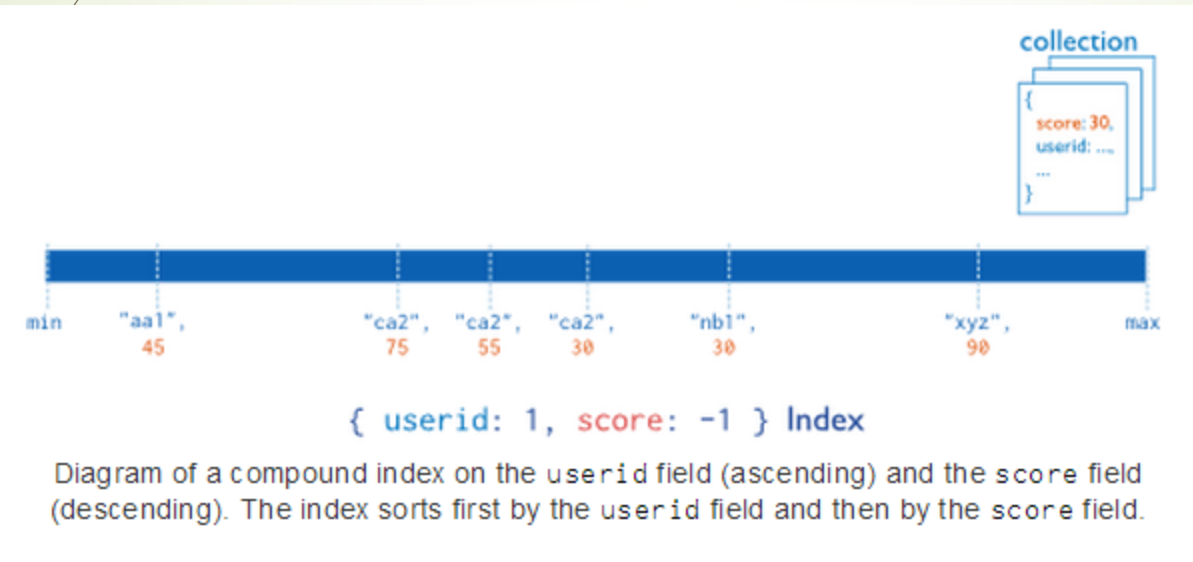


Diagram of a compound index on the `userid` field (ascending) and the `score` field (descending). The index sorts first by the `userid` field and then by the `score` field.

Index in MongoDB

Types

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

- **Multikey Indexes**
  - db.users.ensureIndex({ addr.zip:1})

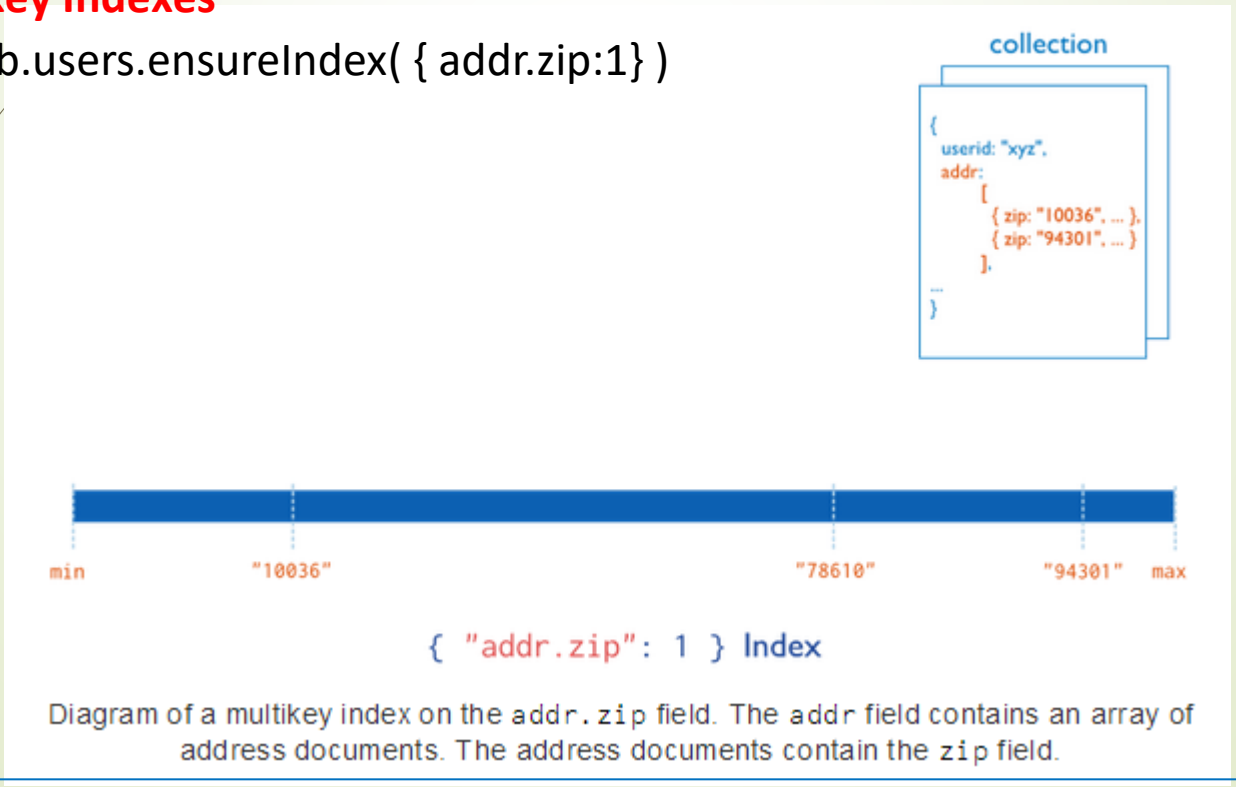


Diagram of a multikey index on the addr.zip field. The addr field contains an array of address documents. The address documents contain the zip field.

Demo of indexes in MongoDB

- Import Data
- Create Index
  - Single Field Index
  - Compound Field Indexes
  - Multikey Indexes
- Show Existing Index
- Hint
  - Single Field Index
  - Compound Field Indexes
  - Multikey Indexes
- Explain
- Compare with data without indexes

```
> db.zips.find().limit(20)
{ "city" : "ACMAR", "loc" : [ -86.51557, 33.584132 ], "pop" : 6055, "state" : "AL", "_id" : "35004" }
{ "city" : "ADAMSVILLE", "loc" : [ -86.959727, 33.588437 ], "pop" : 10616, "state" : "AL", "_id" : "35005" }
{ "city" : "ADGER", "loc" : [ -87.167455, 33.434277 ], "pop" : 3205, "state" : "AL", "_id" : "35006" }
{ "city" : "KEYSTONE", "loc" : [ -86.812861, 33.236868 ], "pop" : 14218, "state" : "AL", "_id" : "35007" }
{ "city" : "NEW SITE", "loc" : [ -85.951086, 32.941445 ], "pop" : 19942, "state" : "AL", "_id" : "35010" }
{ "city" : "ALPINE", "loc" : [ -86.208934, 33.331165 ], "pop" : 3062, "state" : "AL", "_id" : "35014" }
{ "city" : "ARAB", "loc" : [ -86.489638, 34.328339 ], "pop" : 13650, "state" : "AL", "_id" : "35016" }
{ "city" : "BAILEYTON", "loc" : [ -86.621299, 34.268298 ], "pop" : 1781, "state" : "AL", "_id" : "35019" }
{ "city" : "BESSEMER", "loc" : [ -86.947547, 33.409002 ], "pop" : 40549, "state" : "AL", "_id" : "35020" }
{ "city" : "HUEYTOWN", "loc" : [ -86.999607, 33.414625 ], "pop" : 39677, "state" : "AL", "_id" : "35023" }
{ "city" : "BLOUNTSVILLE", "loc" : [ -86.568628, 34.092937 ], "pop" : 9058, "state" : "AL", "_id" : "35031" }
{ "city" : "BREMEN", "loc" : [ -87.004281, 33.973664 ], "pop" : 3448, "state" : "AL", "_id" : "35033" }
{ "city" : "BRENT", "loc" : [ -87.211387, 32.93567 ], "pop" : 3791, "state" : "AL", "_id" : "35034" }
{ "city" : "BRIERFIELD", "loc" : [ -86.951672, 33.042747 ], "pop" : 1282, "state" : "AL", "_id" : "35035" }
{ "city" : "CALERA", "loc" : [ -86.755987, 33.1098 ], "pop" : 4675, "state" : "AL", "_id" : "35040" }
{ "city" : "CENTREVILLE", "loc" : [ -87.11924, 32.950324 ], "pop" : 4902, "state" : "AL", "_id" : "35042" }
{ "city" : "CHELSEA", "loc" : [ -86.614132, 33.371582 ], "pop" : 4781, "state" : "AL", "_id" : "35043" }
{ "city" : "COOSA PINES", "loc" : [ -86.337622, 33.266928 ], "pop" : 7985, "state" : "AL", "_id" : "35044" }
{ "city" : "CLANTON", "loc" : [ -86.642472, 32.835532 ], "pop" : 13990, "state" : "AL", "_id" : "35045" }
{ "city" : "CLEVELAND", "loc" : [ -86.559355, 33.992106 ], "pop" : 2369, "state" : "AL", "_id" : "35049" }
> db.zips.find().count()
29467
```

Demo of indexes in MongoDB

- Import Data
- Create Index
  - Single Field Index
  - Compound Field Indexes
  - Multikey Indexes
- Show Existing Index
- Hint
  - Single Field Index
  - Compound Field Indexes
  - Multikey Indexes
- Explain
- Compare with data without indexes

```
db.zips.ensureIndex({pop: -1})
db.zips.ensureIndex({state: 1, city: 1})
db.zips.ensureIndex({loc: -1})
```

Demo of indexes in MongoDB

- Import Data
- Create Index
  - Single Field Index
  - Compound Field Indexes
  - Multikey Indexes
- **Show Existing Index**
- Hint
  - Single Field Index
  - Compound Field Indexes
  - Multikey Indexes
- Explain
- Compare with data without indexes

```
> db.zips.getIndexes()
[
        {
                "v" : 1,
                "key" : {
                        "_id" : 1
                },
                "ns" : "blog.zips",
                "name" : "_id_"
        },
        {
                "v" : 1,
                "key" : {
                        "pop" : 1
                },
                "ns" : "blog.zips",
                "name" : "pop_1"
        },
        {
                "v" : 1,
                "key" : {
                        "state" : 1,
                        "city" : 1
                },
                "ns" : "blog.zips",
                "name" : "state_1_city_1"
        },
        {
                "v" : 1,
                "key" : {
                        "loc" : 1
                },
                "ns" : "blog.zips",
                "name" : "loc_1"
        }
]
```

Demo of indexes in MongoDB

- Import Data
- Create Index
  - Single Field Index
  - Compound Field Indexes
  - Multikey Indexes
- Show Existing Index
- Hint
  - Single Field Index
  - Compound Field Indexes
  - Multikey Indexes
- Explain
- Compare with data without indexes

```
> db.zips.find().limit(20).hint({pop: -1})
{ "city" : "CHICAGO", "loc" : [  -87.7157,  41.849015 ], "pop" : 112047, "state" : "IL", "_id" : "60623" }
{ "city" : "BROOKLYN", "loc" : [  -73.956985,  40.646694 ], "pop" : 111396, "state" : "NY", "_id" : "11226" }
{ "city" : "NEW YORK", "loc" : [  -73.958805,  40.768476 ], "pop" : 106564, "state" : "NY", "_id" : "10021" }
{ "city" : "NEW YORK", "loc" : [  -73.968312,  40.797466 ], "pop" : 100027, "state" : "NY", "_id" : "10025" }
{ "city" : "BELL GARDENS", "loc" : [  -118.17205,  33.969177 ], "pop" : 99568, "state" : "CA", "_id" : "90201" }
{ "city" : "CHICAGO", "loc" : [  -87.556012,  41.725743 ], "pop" : 98612, "state" : "IL", "_id" : "60617" }
{ "city" : "LOS ANGELES", "loc" : [  -118.258189,  34.007856 ], "pop" : 96074, "state" : "CA", "_id" : "90011" }
{ "city" : "CHICAGO", "loc" : [  -87.704322,  41.920903 ], "pop" : 95971, "state" : "IL", "_id" : "60647" }
{ "city" : "CHICAGO", "loc" : [  -87.624277,  41.693443 ], "pop" : 94317, "state" : "IL", "_id" : "60628" }
{ "city" : "NORWALK", "loc" : [  -118.081767,  33.90564 ], "pop" : 94188, "state" : "CA", "_id" : "90650" }
{ "city" : "CHICAGO", "loc" : [  -87.654251,  41.741119 ], "pop" : 92005, "state" : "IL", "_id" : "60620" }
{ "city" : "CHICAGO", "loc" : [  -87.706936,  41.778149 ], "pop" : 91814, "state" : "IL", "_id" : "60629" }
{ "city" : "CHICAGO", "loc" : [  -87.653279,  41.809721 ], "pop" : 89762, "state" : "IL", "_id" : "60609" }
{ "city" : "CHICAGO", "loc" : [  -87.704214,  41.946401 ], "pop" : 88377, "state" : "IL", "_id" : "60618" }
{ "city" : "JACKSON HEIGHTS", "loc" : [  -73.878551,  40.740388 ], "pop" : 88241, "state" : "NY", "_id" : "11373" }
{ "city" : "ARLETA", "loc" : [  -118.420692,  34.258081 ], "pop" : 88114, "state" : "CA", "_id" : "91331" }
{ "city" : "BROOKLYN", "loc" : [  -73.914483,  40.662474 ], "pop" : 87079, "state" : "NY", "_id" : "11212" }
{ "city" : "SOUTH GATE", "loc" : [  -118.201349,  33.94617 ], "pop" : 87026, "state" : "CA", "_id" : "90280" }
{ "city" : "RIDGEWOOD", "loc" : [  -73.896122,  40.703613 ], "pop" : 85732, "state" : "NY", "_id" : "11385" }
{ "city" : "BRONX", "loc" : [  -73.871242,  40.873671 ], "pop" : 85710, "state" : "NY", "_id" : "10467" }
```

Demo of indexes in MongoDB

- Import Data
- Create Index
  - Single Field Index
  - Compound Field Indexes
  - Multikey Indexes
- Show Existing Index
- Hint
  - Single Field Index
  - Compound Field Indexes
  - Multikey Indexes
- Explain
- Compare with data without indexes

```
> db.zips.find().limit(20).hint({state: 1, city: 1})
{ "city" : "98791", "loc" : [  -176.310048,  51.938901 ], "pop" : 5345, "state" : "AK", "_id" : "98791" }
{ "city" : "AKHIOK", "loc" : [  -152.500169,  57.781967 ], "pop" : 13309, "state" : "AK", "_id" : "99615" }
{ "city" : "AKIACHAK", "loc" : [  -161.39233,  60.891854 ], "pop" : 481, "state" : "AK", "_id" : "99551" }
{ "city" : "AKIAK", "loc" : [  -161.199325,  60.890632 ], "pop" : 285, "state" : "AK", "_id" : "99552" }
{ "city" : "AKUTAN", "loc" : [  -165.785368,  54.143012 ], "pop" : 589, "state" : "AK", "_id" : "99553" }
{ "city" : "ALAKANUK", "loc" : [  -164.60228,  62.746967 ], "pop" : 1186, "state" : "AK", "_id" : "99554" }
{ "city" : "ALEKNAGIK", "loc" : [  -158.619882,  59.269688 ], "pop" : 185, "state" : "AK", "_id" : "99555" }
{ "city" : "ALLAKAKET", "loc" : [  -152.712155,  66.543197 ], "pop" : 170, "state" : "AK", "_id" : "99720" }
{ "city" : "AMBLER", "loc" : [  -156.455652,  67.46951 ], "pop" : 8, "state" : "AK", "_id" : "99786" }
{ "city" : "ANAKTUVUK PASS", "loc" : [  -151.679005,  68.11878 ], "pop" : 260, "state" : "AK", "_id" : "99721" }
{ "city" : "ANCHORAGE", "loc" : [  -149.876077,  61.211571 ], "pop" : 14436, "state" : "AK", "_id" : "99501" }
{ "city" : "ANCHORAGE", "loc" : [  -150.093943,  61.096163 ], "pop" : 15891, "state" : "AK", "_id" : "99502" }
{ "city" : "ANCHORAGE", "loc" : [  -149.893844,  61.189953 ], "pop" : 12534, "state" : "AK", "_id" : "99503" }
{ "city" : "ANCHORAGE", "loc" : [  -149.74467,  61.203696 ], "pop" : 32383, "state" : "AK", "_id" : "99504" }
{ "city" : "ANCHORAGE", "loc" : [  -149.828912,  61.153543 ], "pop" : 20128, "state" : "AK", "_id" : "99507" }
{ "city" : "ANCHORAGE", "loc" : [  -149.810085,  61.205959 ], "pop" : 29857, "state" : "AK", "_id" : "99508" }
{ "city" : "ANCHORAGE", "loc" : [  -149.897401,  61.119381 ], "pop" : 17094, "state" : "AK", "_id" : "99515" }
{ "city" : "ANCHORAGE", "loc" : [  -149.779998,  61.10541 ], "pop" : 18356, "state" : "AK", "_id" : "99516" }
{ "city" : "ANCHORAGE", "loc" : [  -149.936111,  61.190136 ], "pop" : 15192, "state" : "AK", "_id" : "99517" }
{ "city" : "ANCHORAGE", "loc" : [  -149.886571,  61.154862 ], "pop" : 8116, "state" : "AK", "_id" : "99518" }
```

Demo of indexes in MongoDB

- Import Data
- Create Index
  - Single Field Index
  - Compound Field Indexes
  - Multikey Indexes
- Show Existing Index
- Hint
  - Single Field Index
  - Compound Field Indexes
  - Multikey Indexes
- Explain
- Compare with data without indexes

```
> db.zips.find().limit(20).hint({loc: -1})
{ "city" : "BARROW", "loc" : [ -156.817409, 71.234637 ], "pop" : 3696, "state" : "AK", "_id" : "99723" }
{ "city" : "WAINWRIGHT", "loc" : [ -160.012532, 70.620064 ], "pop" : 492, "state" : "AK", "_id" : "99782" }
{ "city" : "NUIQSUT", "loc" : [ -150.997119, 70.192737 ], "pop" : 354, "state" : "AK", "_id" : "99789" }
{ "city" : "PRUDHOE BAY", "loc" : [ -148.559636, 70.070057 ], "pop" : 153, "state" : "AK", "_id" : "99734" }
{ "city" : "KAKTOVIK", "loc" : [ -143.631329, 70.042889 ], "pop" : 245, "state" : "AK", "_id" : "99747" }
{ "city" : "POINT LAY", "loc" : [ -162.906148, 69.705626 ], "pop" : 139, "state" : "AK", "_id" : "99759" }
{ "city" : "POINT HOPE", "loc" : [ -166.72618, 68.312058 ], "pop" : 640, "state" : "AK", "_id" : "99766" }
{ "city" : "ANAKTUVUK PASS", "loc" : [ -151.679005, 68.11878 ], "pop" : 260, "state" : "AK", "_id" : "99721" }
{ "city" : "ARCTIC VILLAGE", "loc" : [ -145.423115, 68.077395 ], "pop" : 107, "state" : "AK", "_id" : "99722" }
{ "city" : "KIVALINA", "loc" : [ -163.733617, 67.665859 ], "pop" : 689, "state" : "AK", "_id" : "99750" }
{ "city" : "AMBLER", "loc" : [ -156.455652, 67.46951 ], "pop" : 8, "state" : "AK", "_id" : "99786" }
{ "city" : "KIANA", "loc" : [ -158.152204, 67.18026 ], "pop" : 349, "state" : "AK", "_id" : "99749" }
{ "city" : "BETTLES FIELD", "loc" : [ -151.062414, 67.100495 ], "pop" : 156, "state" : "AK", "_id" : "99726" }
{ "city" : "VENETIE", "loc" : [ -146.413723, 67.010446 ], "pop" : 184, "state" : "AK", "_id" : "99781" }
{ "city" : "NOATAK", "loc" : [ -160.509453, 66.97553 ], "pop" : 395, "state" : "AK", "_id" : "99761" }
{ "city" : "SHUNGNAK", "loc" : [ -157.613496, 66.958141 ], "pop" : 0, "state" : "AK", "_id" : "99773" }
{ "city" : "KOBUK", "loc" : [ -157.066864, 66.912253 ], "pop" : 306, "state" : "AK", "_id" : "99751" }
{ "city" : "KOTZEBUE", "loc" : [ -162.126493, 66.846459 ], "pop" : 3347, "state" : "AK", "_id" : "99752" }
{ "city" : "NOORVIK", "loc" : [ -161.044132, 66.836353 ], "pop" : 534, "state" : "AK", "_id" : "99763" }
{ "city" : "CHALKYITSIK", "loc" : [ -143.638121, 66.719 ], "pop" : 99, "state" : "AK", "_id" : "99788" }
```

Demo of indexes in MongoDB

- Import Data
- Create Index
  - Single Field Index
  - Compound Field Indexes
  - Multikey Indexes
- Show Existing Index
- Hint
  - Single Field Index
  - Compound Field Indexes
  - Multikey Indexes
- Explain
- Compare with data without indexes

```
> db.zips.find({city: 'NASHVILLE', state: 'TN'}).explain()
{
        "cursor" : "BasicCursor",
        "isMultiKey" : false,
        "n" : 19,
        "nscannedObjects" : 29467,
        "nscanned" : 29467,
        "nscannedObjectsAllPlans" : 29467,
        "nscannedAllPlans" : 29467,
        "scanAndOrder" : false,
        "indexOnly" : false,
        "nYields" : 0,
        "nChunkSkips" : 0,
        "millis" : 33,
        "indexBounds" : {

        },
        "server" : "g:27017"
}
```

Demo of indexes in MongoDB

- Import Data
- Create Index
  - Single Field Index
  - Compound Field Indexes
  - Multikey Indexes
- Show Existing Index
- Hint
  - Single Field Index
  - Compound Field Indexes
  - Multikey Indexes
- Explain
- Compare with data without indexes

```
> db.zips.dropIndexes()
{
        "nIndexesWas" : 4,
        "msg" : "non-_id indexes dropped for collection",
        "ok" : 1
}
> db.zips.find({city: 'NASHVILLE', state: 'TN'}).explain()
{
        "cursor" : "BasicCursor",
        "isMultiKey" : false,
        "n" : 19,
        "nscannedObjects" : 29467,
        "nscanned" : 29467,
        "nscannedObjectsAllPlans" : 29467,
        "nscannedAllPlans" : 29467,
        "scanAndOrder" : false,
        "indexOnly" : false,
        "nYields" : 0,
        "nChunkSkips" : 0,
        "millis" : 33,
        "indexBounds" : {

        },
        "server" : "g:27017"
}
```

Without Index

```
> db.zips.find({city: 'NASHVILLE', state: 'TN'}).explain()
{
        "cursor" : "BtreeCursor state_1_city_1",
        "isMultiKey" : false,
        "n" : 19,
        "nscannedObjects" : 19,
        "nscanned" : 19,
        "nscannedObjectsAllPlans" : 19,
        "nscannedAllPlans" : 19,
        "scanAndOrder" : false,
        "indexOnly" : false,
        "nYields" : 0,
        "nChunkSkips" : 0,
        "millis" : 0,
        "indexBounds" : {
                "state" : [
                        [
                                "TN",
                                "TN"
                        ]
                ],
                "city" : [
                        [
                                "NASHVILLE",
                                "NASHVILLE"
                        ]
                ]
        },
        "server" : "g:27017"
}
```

With Index

Aggregation

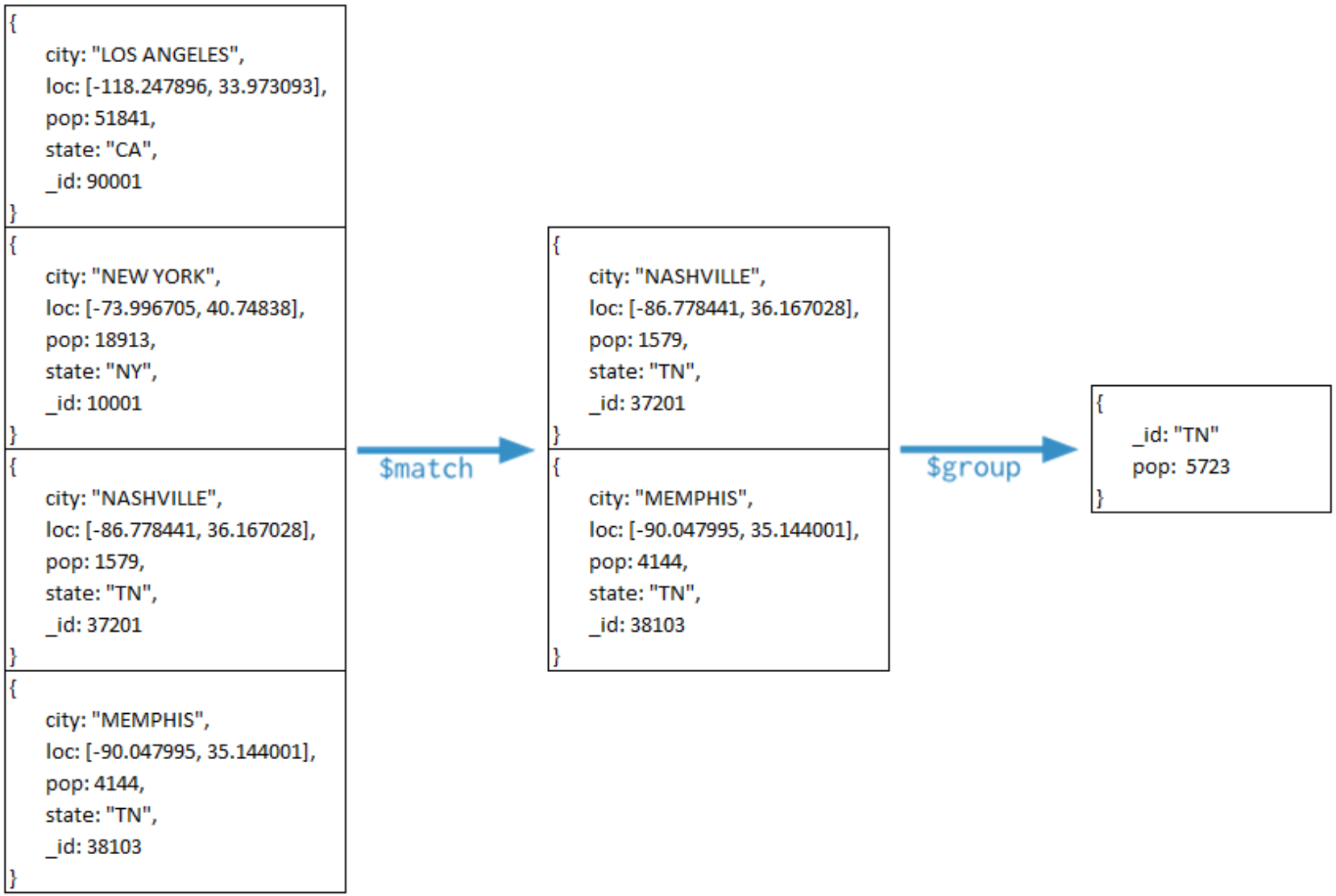
Aggregation

- Operations that process records and return results.
- Aggregation operations
- Running aggregation on mongod instance simplifies app code and limits resource requirements.

Pipelines

- Modeled on 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.

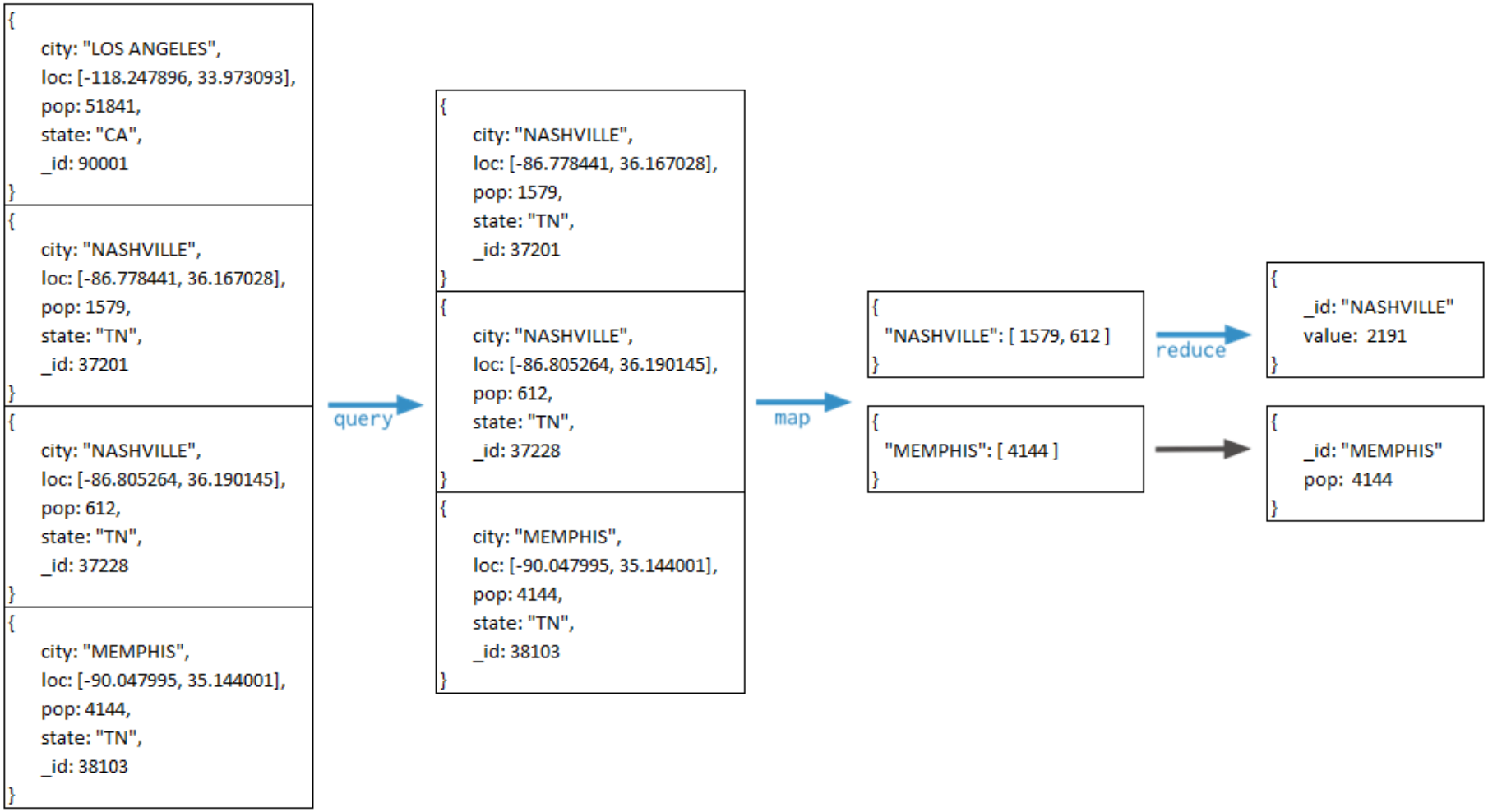
```
db.zips.aggregate(
                    { $match: { state: "TN" } },
                    { $group: {_id: "TN", pop: { $sum: "$pop" } } }
                  );
```



Map-Reduce

- Has two phases:
  - A *map* stage that processes each document and *emits* one or more objects for each input document
  - A *reduce* phase that combines the output of the map operation.
  - An optional *finalize* stage for final modifications to the result
- Uses Custom JavaScript functions
  - Provides greater flexibility but is less efficient and more complex than the aggregation pipeline
- Can have output sets that exceed the 16 MB output limitation of the aggregation pipeline.

```
db.zips.mapReduce(
                    function() { emit( this.city, this.pop ); },
                    function(key, values) { return Array.sum( values ) },
                    {
                      query: { state: "TN" },
                      out: "city_pop_totals"
                    }
                  );
```

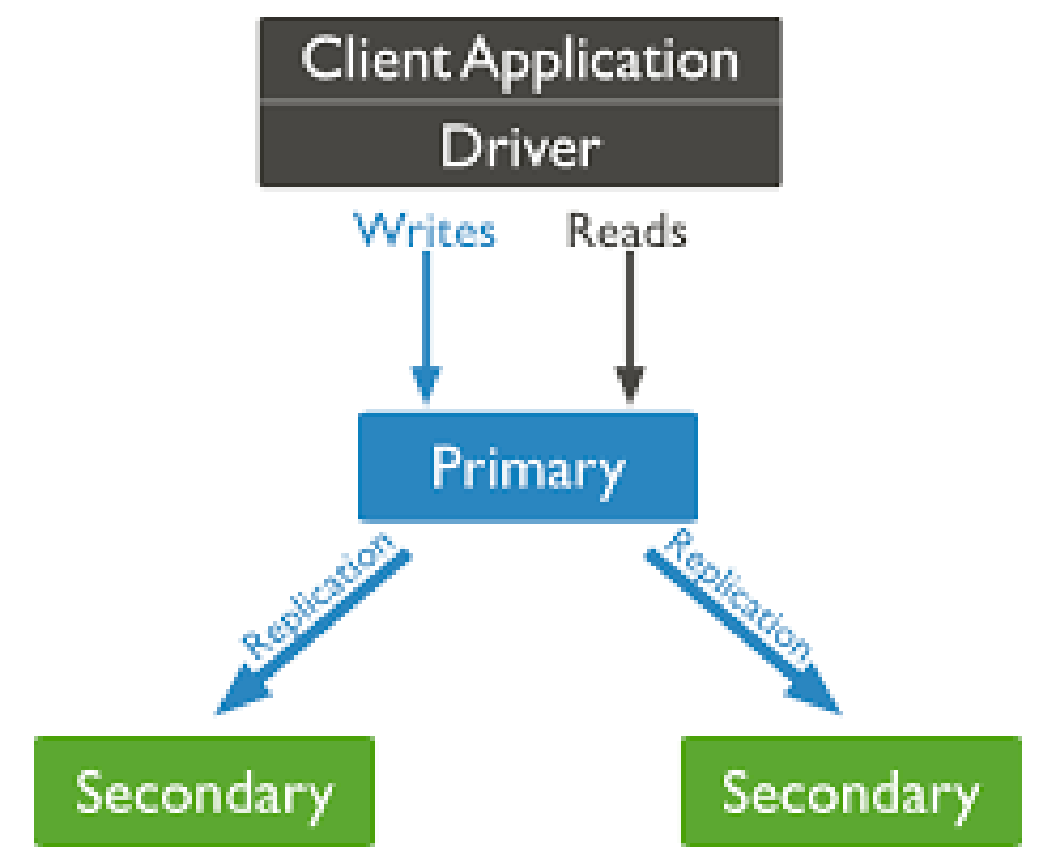


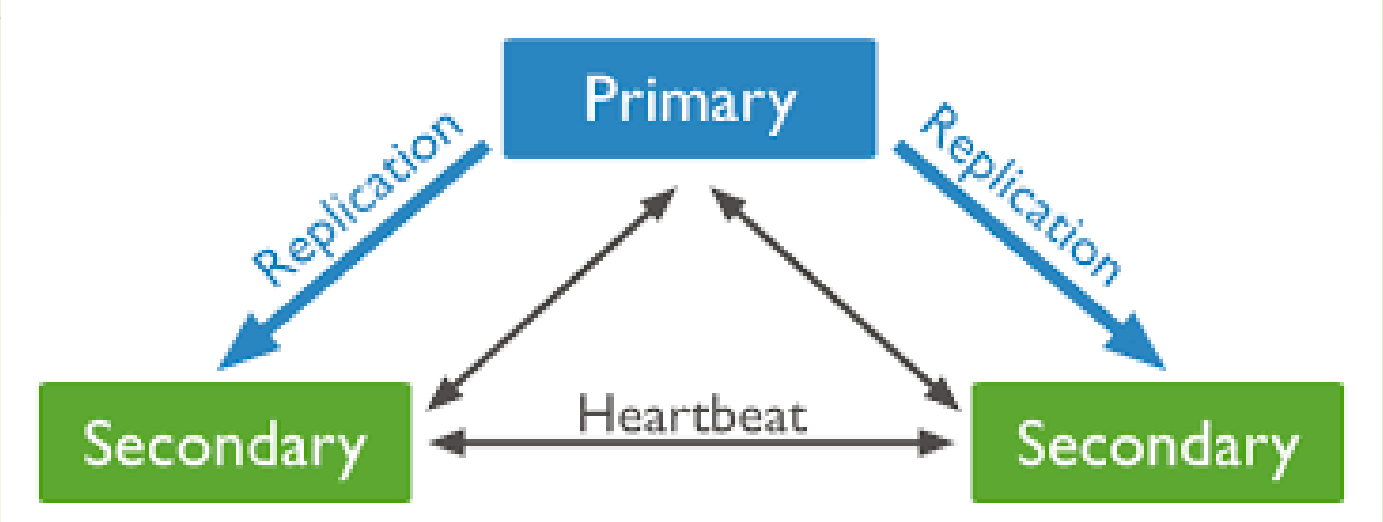
```
db.students.aggregate([
  {
    $group: {
      _id: "$class",           // group by class
      totalScore: { $sum: "$score" },
      avgScore: { $avg: "$score" }
    }
  }
])
```

```
db.students.aggregate([
  { $match: { score: { $gte: 80 } } },  // only students with score >= 80
  { $project: { name: 1, score: 1, _id: 0 } } // show only name & score
])
```

```
db.students.aggregate([
  { $sort: { score: -1 } },  // sort descending by score
  { $limit: 3 }              // top 3 students
])
```

Replication


Client Application
Driver
Writes
Reads
Primary
Replication
Replication
Secondary
Secondary


Primary
Replication
Replication
Secondary
Heartbeat
Secondary

Replication

- What is replication?
- Fault tolerance
- Availability: Increase read capacity

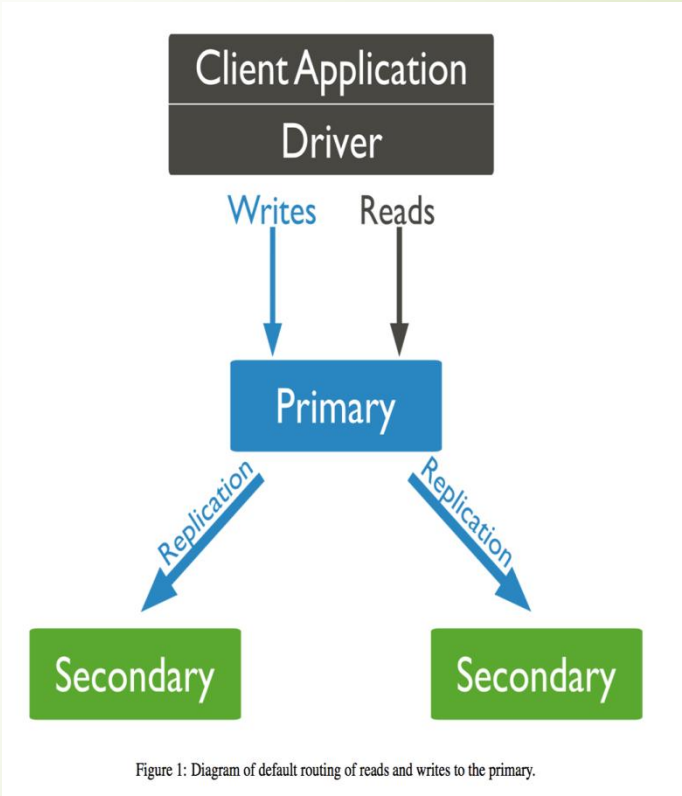


Figure 1: Diagram of default routing of reads and writes to the primary.

A replica set is a group of MongoDB servers that maintain the same data for high availability and redundancy.

- One primary node → handles all writes and reads by default
- Secondary nodes → replicate data from primary and can serve reads (optional)

Primary Node

Role: Accepts writes and sends data changes (oplog) to secondaries.

Election: Only one primary exists at a time; if it fails, a new primary is elected.

Reads: By default, clients read from primary (strong consistency).

Failover: If primary goes down, a secondary becomes the new primary automatically.

Secondary Node

Role: Replicates data from the primary (using **oplog**)

Reads: Can serve reads if read preference allows

Writes: Cannot accept writes (unless promoted to primary)

Use case: Backup, failover, and scaling read operations

Replication in MongoDB

- Arbiter
  - Participates in Voting – helps maintain odd # votes in quorum
  - Can't be primary
  - Does not store data
  - Used when Replica sets are even number of data-bearing nodes to avoid ties

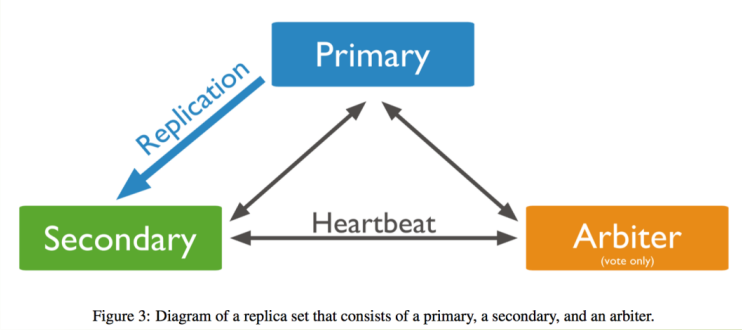


Figure 3: Diagram of a replica set that consists of a primary, a secondary, and an arbiter.

Replication in MongoDB

- Delayed Secondary
  - A normal secondary **replicating data from primary with a delay** (usually in seconds or hours).
  - Stores data
  - Used to protect against accidental deletions/updates
  - Can serve reads if read preference allows

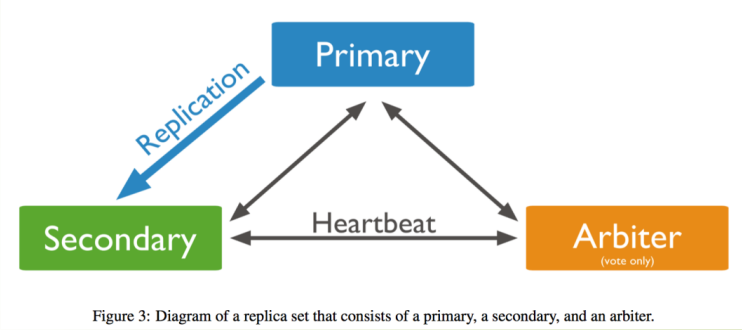


Figure 3: Diagram of a replica set that consists of a primary, a secondary, and an arbiter.

Replication in MongoDB

- Automatic Failover (primary fails)
  - Heartbeats
  - Elections
- Ensures high availability without manual intervention
- Happens within seconds in most cases

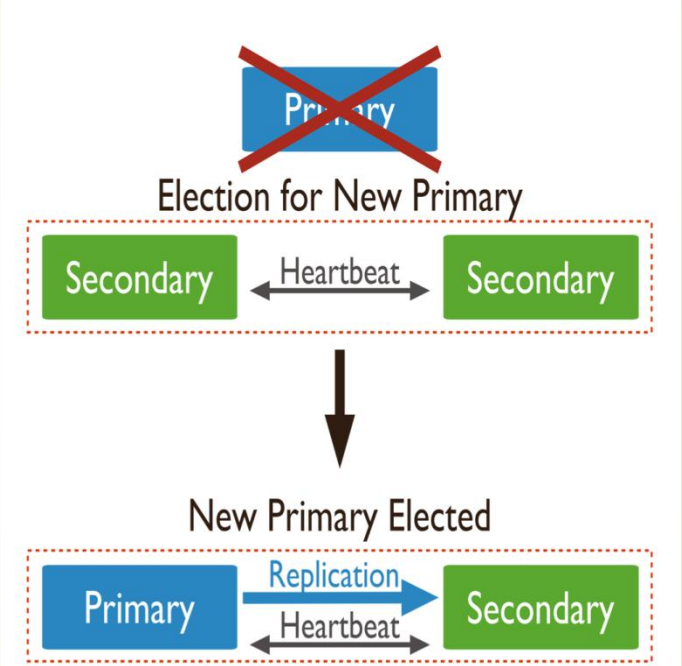


Figure 21: Diagram of an election of a new primary. In a three member replica set with two secondaries, the primary becomes unreachable. The loss of a primary triggers an election where one of the secondaries becomes the new primary

Replication in MongoDB

- The Standard Replica Set Deployment
- Deploy an Odd Number of Members
- Rollback
- Security: SSL/TLS

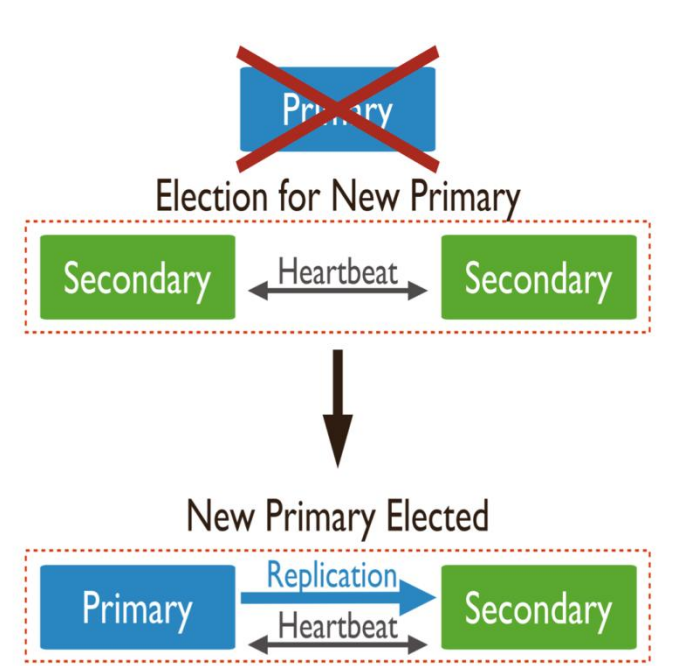


Figure 21: Diagram of an election of a new primary. In a three member replica set with two secondaries, the primary becomes unreachable. The loss of a primary triggers an election where one of the secondaries becomes the new primary

Replication in MongoDB

- **Election timeout:** default 10s (settings.electionTimeoutMillis)
- **Priority:** Secondary with higher priority is more likely to become primary
- **Arbiters:** Help maintain odd number of votes for election quorum

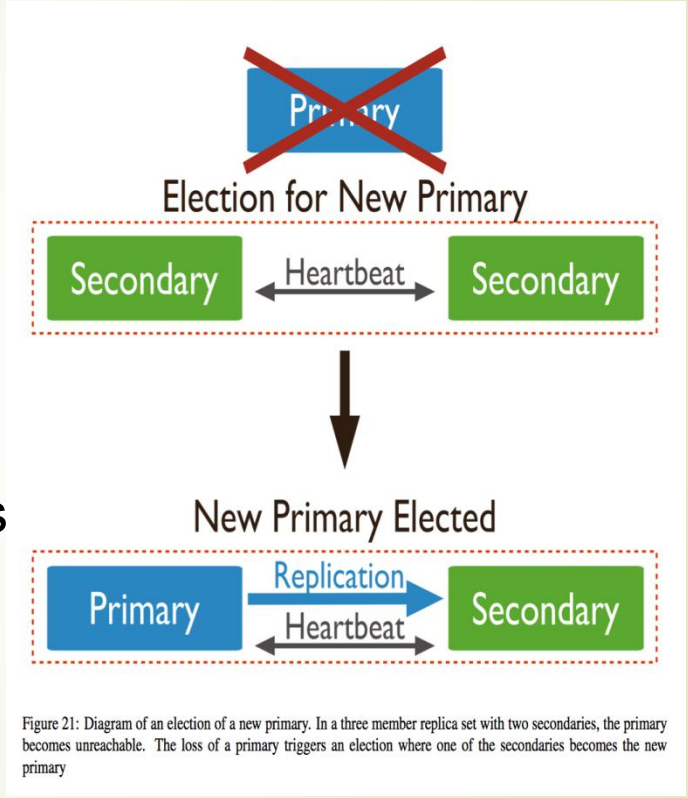
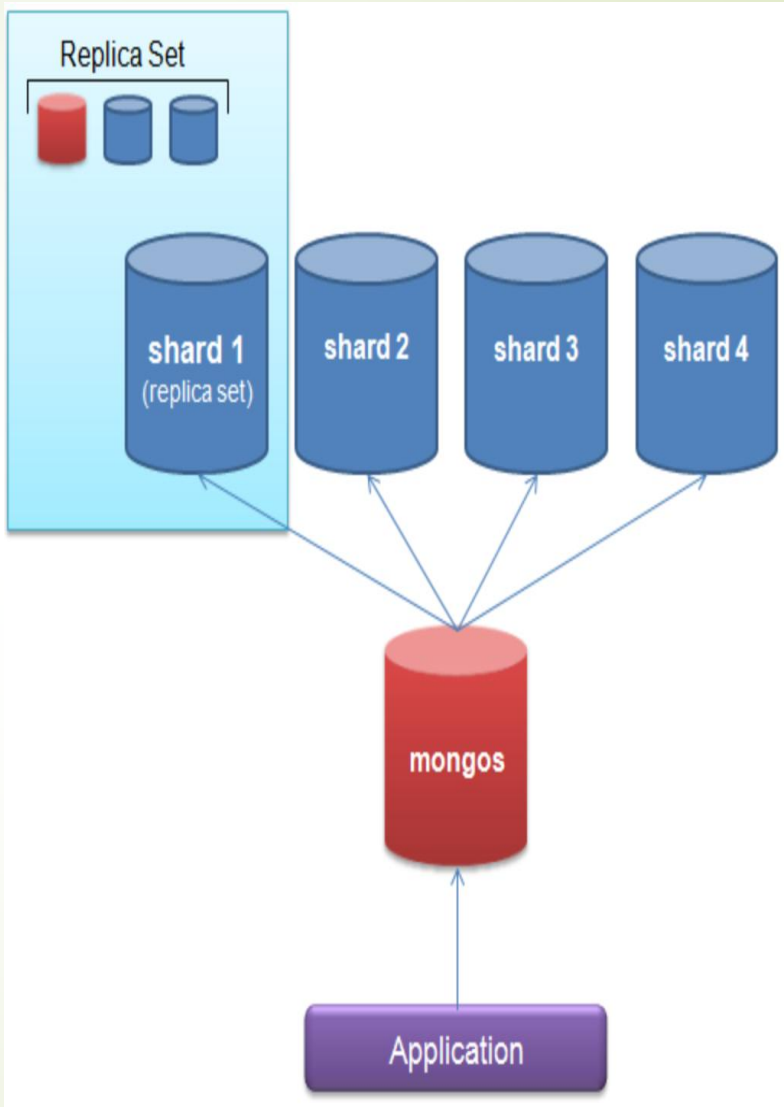


Figure 21: Diagram of an election of a new primary. In a three member replica set with two secondaries, the primary becomes unreachable. The loss of a primary triggers an election where one of the secondaries becomes the new primary

Sharding

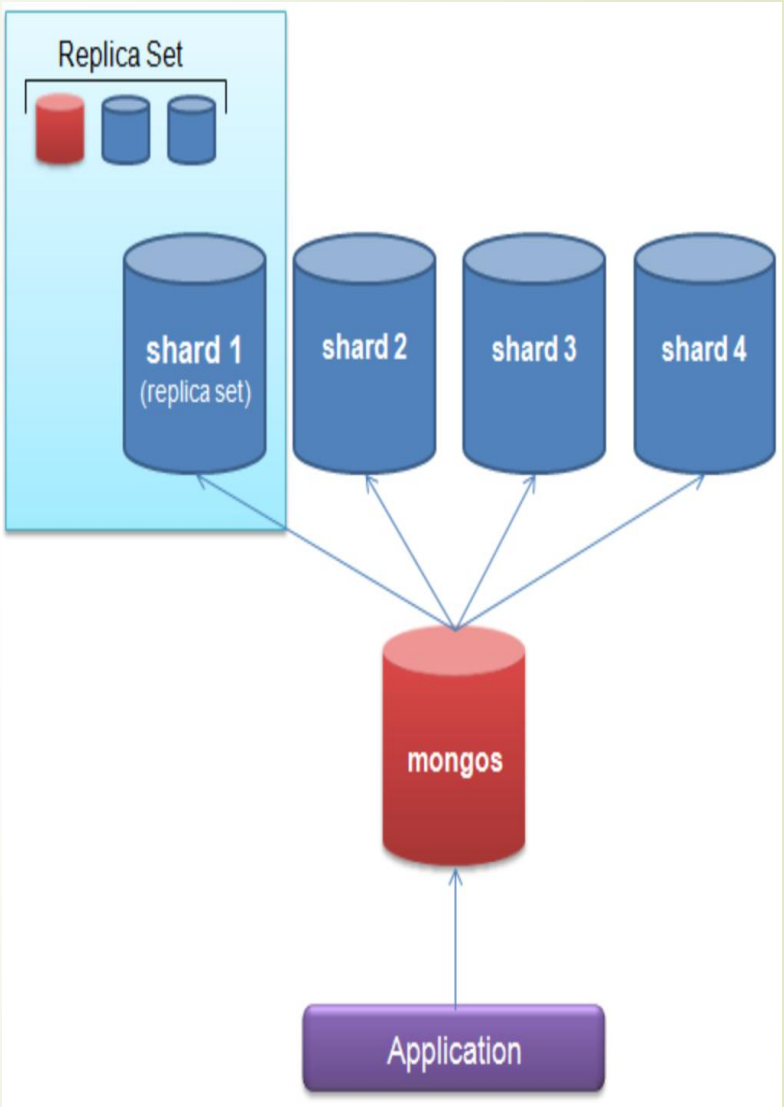
Sharding

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



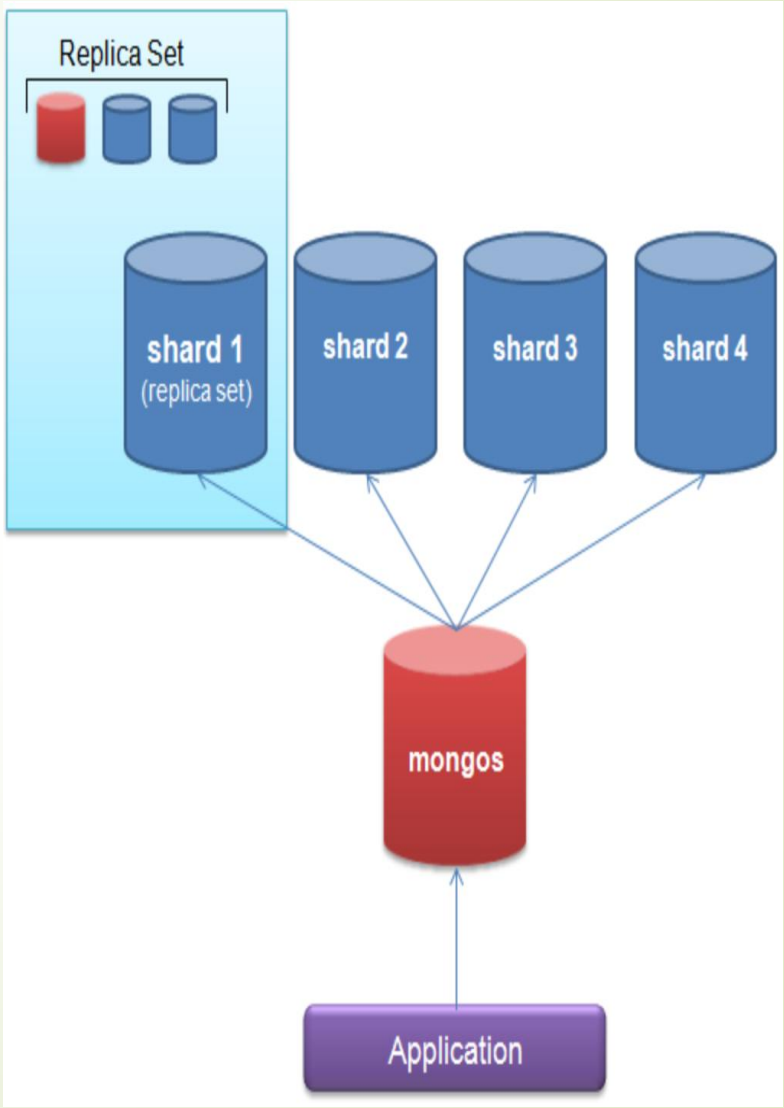
Hotspotting

- When too many reads/writes keep hitting the *same shard*, zone, or chunk, causing that shard to become overloaded while others remain idle.
- It is a performance imbalance.



Why Sharding

- MongoDB workloads become slow when:
- Data > RAM
- Data > Single machine disk
- High write/transaction rate
- Users want linear scalability



Why MongoDB is Suitable for Big Data

```
{
  "_id": 1,
  "customer": "Ali",
  "orders": [
    { "item": "Laptop", "price": 100000 },
    { "item": "Mouse", "price": 1500 }
  ]
}
```

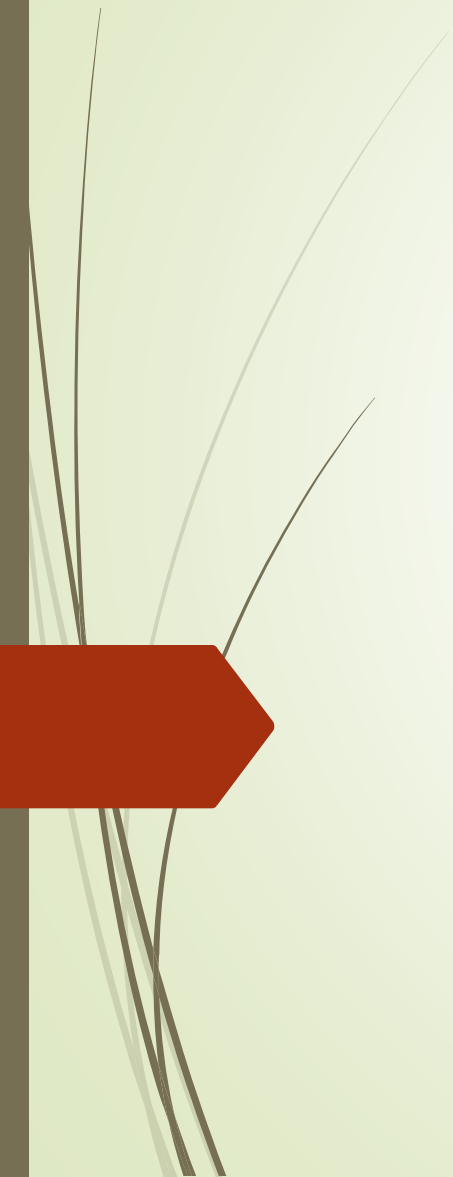
Document Model

- One read loads ALL relevant information
- No expensive JOINs across 3–4 tables
- Faster in distributed systems where JOINs are costly over the network
- Embedding reduces JOIN cost → critical in Big Data systems.

```
{
  "name": "Ali",
  "skills": ["Python", "ML"],
  "address": { "city": "Lahore" }
}
```

Flexible Schema

- New fields can appear anytime
- Different documents can have different structures
- Ideal for sensor data, clickstream, logs, IoT, social media
- Supports huge, fast-changing data without schema migrations.



{ "country": 1 }

Sharding: Horizontal Scaling

- MongoDB can split a collection across many machines.
- Automatic data distribution
- Load balancing across nodes
- Query routing to correct shard only → massive performance gains
- Solves TB/PB scale data where RDBMS collapses.

```
db.users.find({
  $and: [
    { age: { $gt: 20 } },
    { city: { $in: ["Lahore", "Karachi"] } }
  ]
});
```

Operators

- Works directly on BSON documents
- Can query nested fields
- Can search arrays, objects, text, geospatial data
- Optimized for distributed searches
- (WHERE clauses slow down on huge tables)

Indexing

- Query huge text documents instantly
- Find nearest locations with $geoNear
- Index elements inside arrays
- Index types: Single field, Compound, Multi-key (arrays),Text search, Geospatial search, Partial, TTL docs

Operators

- Non-blocking pipeline stages
- Built for parallel execution
- Efficient map/shuffle operations
- Can process large-scale analytics inside the DB
- Solves analytical workloads that normally need Hadoop/Spark.

Thanks