DOCUMENT DATABASE

MONGODB

Yunghans Irawan (yirawan@nus.edu.sg)

Agenda

- Document Database
- MongoDB
- Data Manipulation
- Data Modeling
- Replication and Sharding
- Summary

Document Database

- Non relational database that stores data as structured document
- Usually in XML or JSON formats
  - Document – Object conversion is easy
  - JSON format particularly work well with AJAX
- Usually schemaless – no enforcement of schema
  - Data model versioning become the responsibility of the application
- Usually no join operation

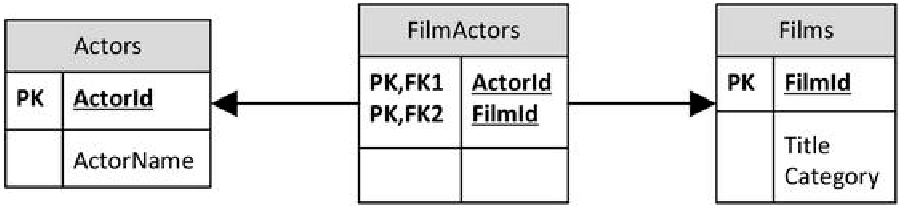
Document Databases

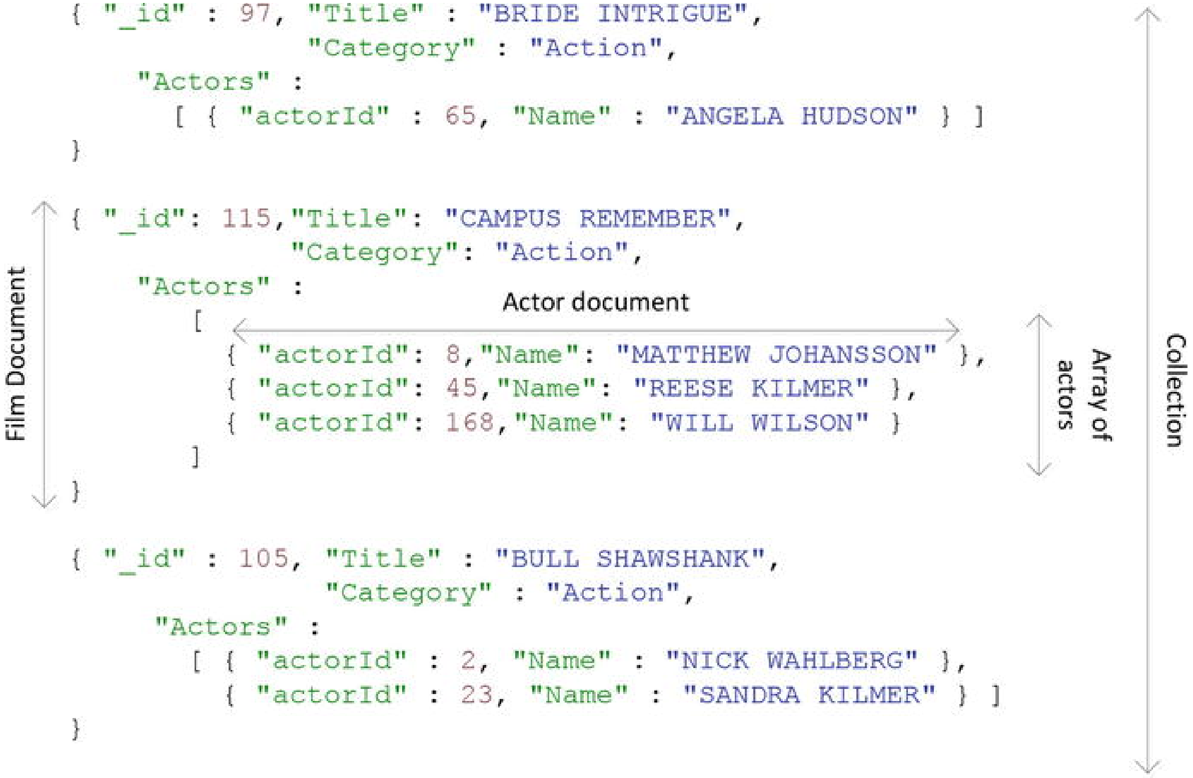
Name	Initial Release	Latest Version	License
Couchbase	2011	5.1.0, Feb 2018	Open Source
MongoDB	2009	4.0.0 June 2018	Open Source
CouchDB	2005	2.1.2 July 2018	Open Source
MarkLogic	2001	9.0 2017	Commercial
Amazon DynamoDB	2012	-	Commercial (Cloud based)

Data Modeling

- No standard of “correctness” like the normal forms for RDBMS
- Driven by the nature of the data and the nature of query

Data Modeling Example



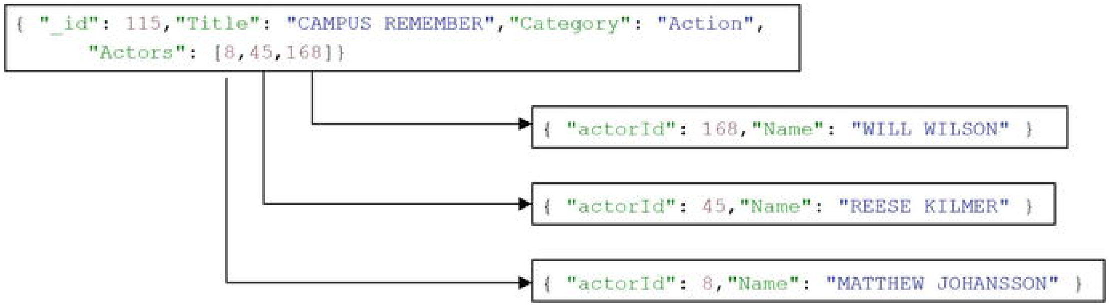


```
{ "_id" : 97, "Title" : "BRIDE INTRIGUE",
              "Category" : "Action",
    "Actors" :
      [ { "actorId" : 65, "Name" : "ANGELA HUDSON" } ]
}

{ "_id": 115,"Title": "CAMPUS REMEMBER",
             "Category": "Action",
    "Actors" :
       [
         { "actorId": 8,"Name": "MATTHEW JOHANSSON" },
         { "actorId": 45,"Name": "REESE KILMER" },
         { "actorId": 168,"Name": "WILL WILSON" }
       ]
}

{ "_id" : 105, "Title" : "BULL SHAWSHANK",
               "Category" : "Action",
     "Actors" :
       [ { "actorId" : 2, "Name" : "NICK WAHLBERG" },
         { "actorId" : 23, "Name" : "SANDRA KILMER" } ]
}
```

Data Modeling Example

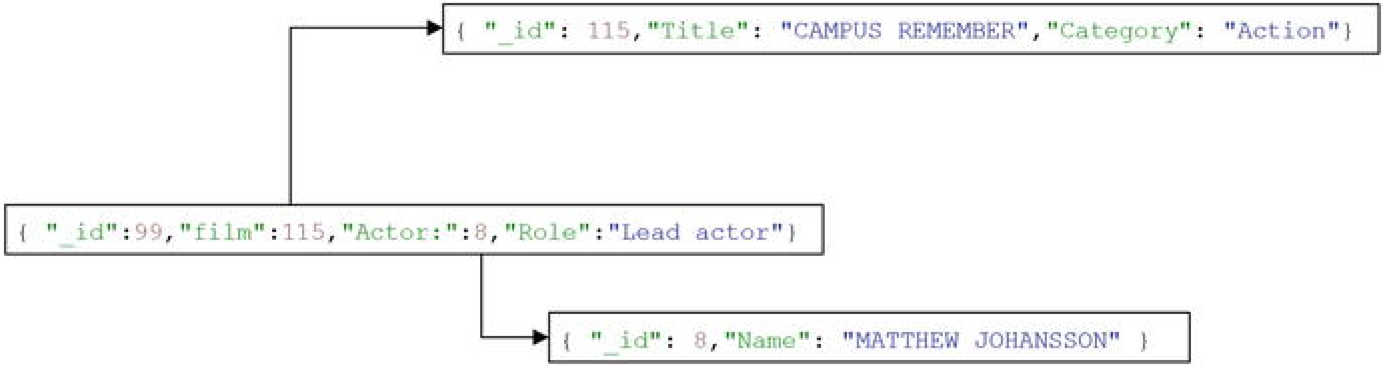


```
{ "_id": 115,"Title": "CAMPUS REMEMBER","Category": "Action",
      "Actors": [8,45,168]}
```

```
{ "actorId": 168,"Name": "WILL WILSON" }
```

```
{ "actorId": 45,"Name": "REESE KILMER" }
```

```
{ "actorId": 8,"Name": "MATTHEW JOHANSSON" }
```



```
{ "_id": 115,"Title": "CAMPUS REMEMBER","Category": "Action"}
```

```
{ "_id":99,"film":115,"Actor:":8,"Role":"Lead actor"}
```

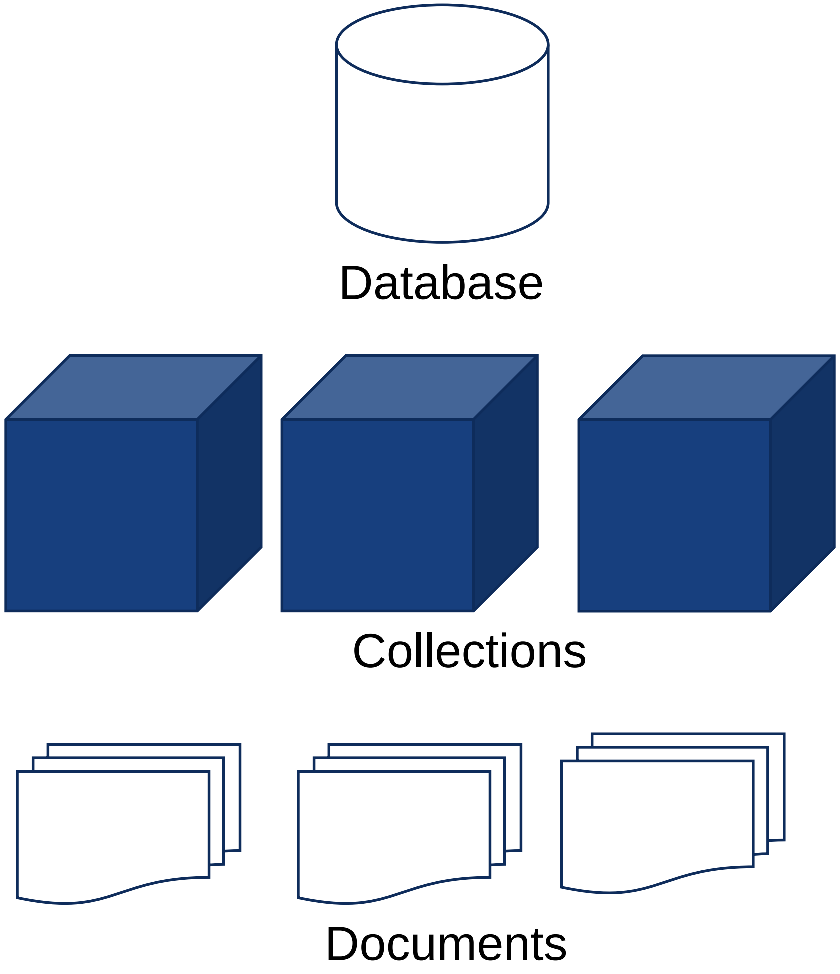
```
{ "_id": 8,"Name": "MATTHEW JOHANSSON" }
```

MongoDB

- Arguably, the most popular document database at the moment
- MEAN stack
  - MongoDB, Express, Angular, Node
- High Performance, highly available, automatic scaling
- Data are represented as JSON document
  - Behind the scene the document is stored in BSON format (BSON = Binary JSON – bsonspec.org)

Document Structure in Collection

- Collections do not enforce document structure.
- In practice, however, the documents in a collection share a similar structure.



Comparison with SQL Databases

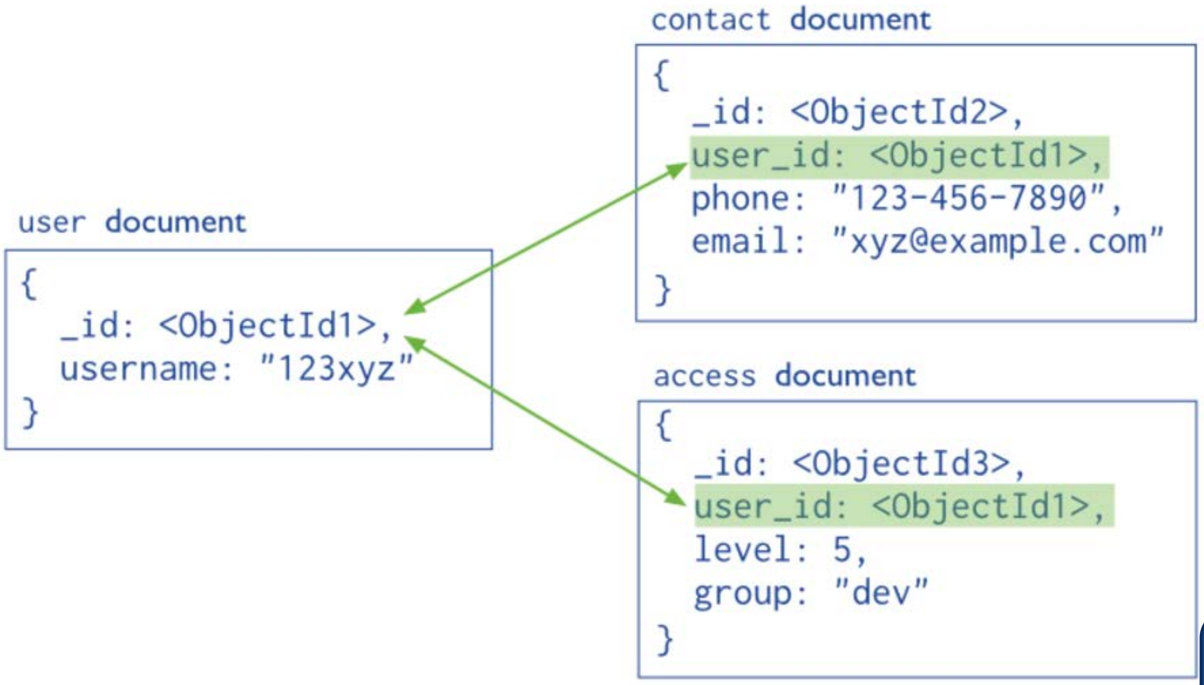
MongoDB	SQL Databases
Database	Database
Collections	Table
Documents	Records
ObjectId	Primary Key
References	Foreign Key

MongoDB Document

- Documents are uniquely identified with ObjectId
- IDs are highly likely to be unique across collections

MongoDB Document Structure - References

- References store the relationships between data by including links or *references* from one document to another.
- Applications can resolve these references to access the related data.



user document

```
{
  _id: <ObjectId1>,
  username: "123xyz"
}
```

contact document

```
{
  _id: <ObjectId2>,
  user_id: <ObjectId1>,
  phone: "123-456-7890",
  email: "xyz@example.com"
}
```

access document

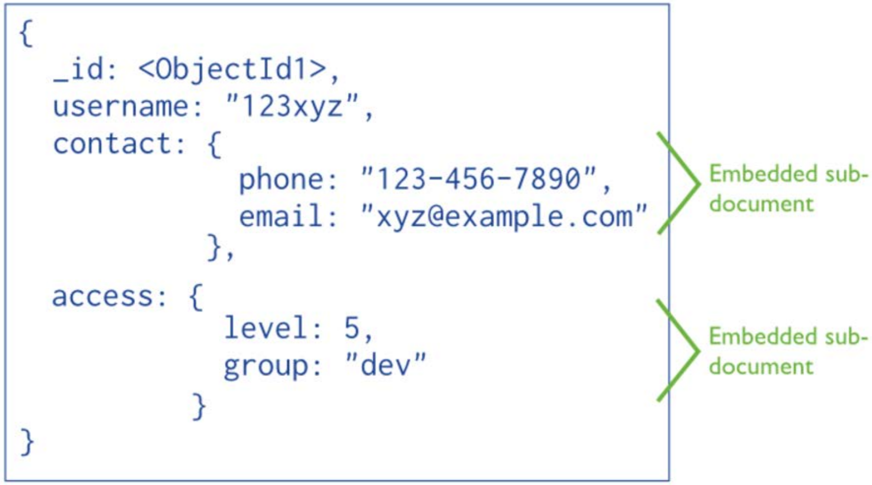
```
{
  _id: <ObjectId3>,
  user_id: <ObjectId1>,
  level: 5,
  group: "dev"
}
```

Normalized data models!

Ref: https://www.mongodb.com/

MongoDB Document Structure – Embedded Data

- Embedded documents capture relationships between data by storing related data in a single document structure.
- Possible to embed document structures in a field or array
- Allow applications to retrieve and manipulate related data in a single database operation.



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

Denormalized data models!

Data Types

Type	Number	Alias
Double	1	“double”
String	2	“string”
Object	3	“object”
Array	4	“array”
Binary data	5	“binData”
Undefined (Deprecated)	6	“undefined”
ObjectId	7	“objectId”
Boolean	8	“bool”
Date	9	“date”
Null	10	“null”

Type	Number	Alias
Regular Expression	11	“regex”
DBPointer (Deprecated)	12	“dbPointer”
JavaScript	13	“javascript”
Symbol (Deprecated)	14	“symbol”
JavaScript (with scope)	15	“javascriptWithScope”
32-bit integer	16	“int”
Timestamp	17	“timestamp”
64-bit integer	18	“long”
Min key	-1	“minKey”
Max key	127	“maxKey”

Database Commands

`show dbs`	Shows the names of the available databases
`db`	Check current database
`use <db name>`	Sets the current database to <db name>. If the database does not exist, the database will be created automatically. *(Note. There must be at least one document in order to appear in database list)*
`show collections`	Shows the collections in the current database

Insert Operation -1

- Creation of a document into the collection
  - Collection media is created once the data is saved (if the collection was exist previously)

```
db.media.insert( { "Type" : "CD", "Artist" :
"Nirvana", "Title" : "Nevermind" })
```

- Create a document, then add to the collection

```
documentMedia = ({“Type":“CD", “Artist":“John
Tan", "Title" : “Good Day"})
```

```
db.media.insert(documentMedia)
```

Insert Operation -2

- Bulk Insert

```
db.post.insert([
   {  title: 'MongoDB Overview',
      description: 'MongoDB is no sql database'},
   {  title: 'NoSQL Database',
      description: 'NoSQL database doesn't have tables',
      comments: [
         {  user:'user1',
            message: 'My first comment', } ]
   }
])
```

Query Operation

- Query a document

```
db.media.find()
```

- To query a document with the results displayed in a formatted way

```
db.media.find().pretty()
```

More criteria: refer to https://docs.mongodb.com/getting-started/shell/query/

Update Operation

- Query a document

```
db.media.update(
  { “Title" : “Nevermind" },
  { $set: { “Artist": “Chia" } }
)
```

- To update all document (MongoDB > >= 2.2)

```
db.media.update(
      {},
      { $set: {"Artist" : "ChiaYK" } },
      {multi : true}
)
```

Replace a Document

- Replace a document

```
db.media.update(
  { “Title" : “Nevermind" },
  { “Album": “The best of You" },
    “Track” : “1”}
)
```

Delete Operation

- Remove documents that satisfy the criteria

```
db.media.remove(
      {“Artist” :  “ChiaYK” }
)
```

- Remove all documents

```
db.media.remove( {} )
```

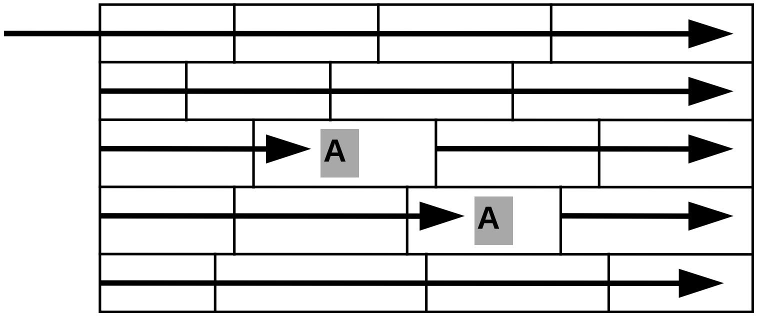
Drop Collection

- Drop the complete collection

```
db.media.drop()
```

Indexes -1

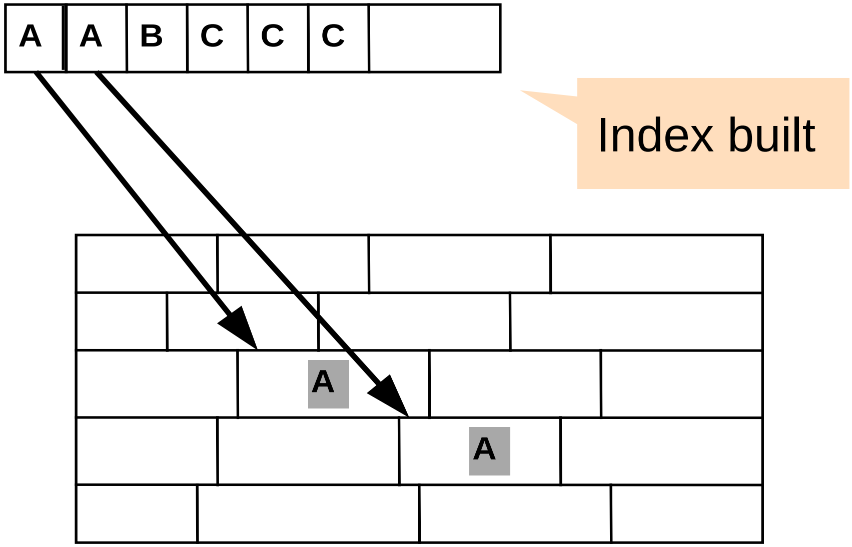
- Indexes support the efficient execution of queries in MongoDB.



- Without indexes, MongoDB must perform a *collection scan*, i.e. scan every document in a collection, to select those documents that match the query statement.

Indexes -2

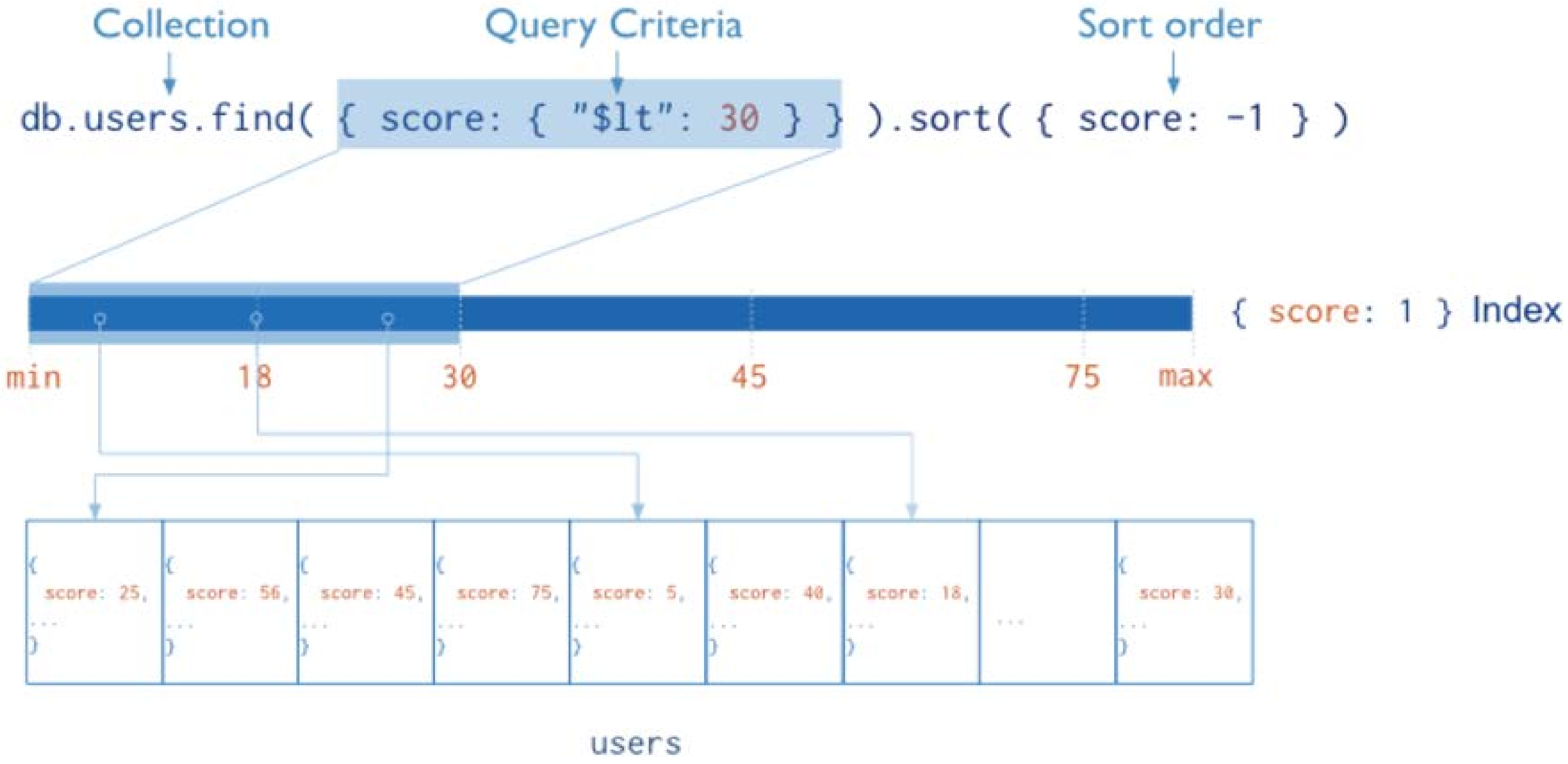
- If an appropriate index exists for a query, MongoDB can use the index to limit the number of documents it must inspect.



Index built

Search with Index

- Example – Search on Collection *Users*



```
db.users.find( { score: { "$lt": 30 } } ).sort( { score: -1 } )
```

Creating Index - 1

- Creating an index for title (ascending order)

```
db.media.ensureIndex( { Title :1 } )
```

- Creating an index for title (descending order)

```
db.media.ensureIndex( { Title :-1 } )
```

Creating Index - 2

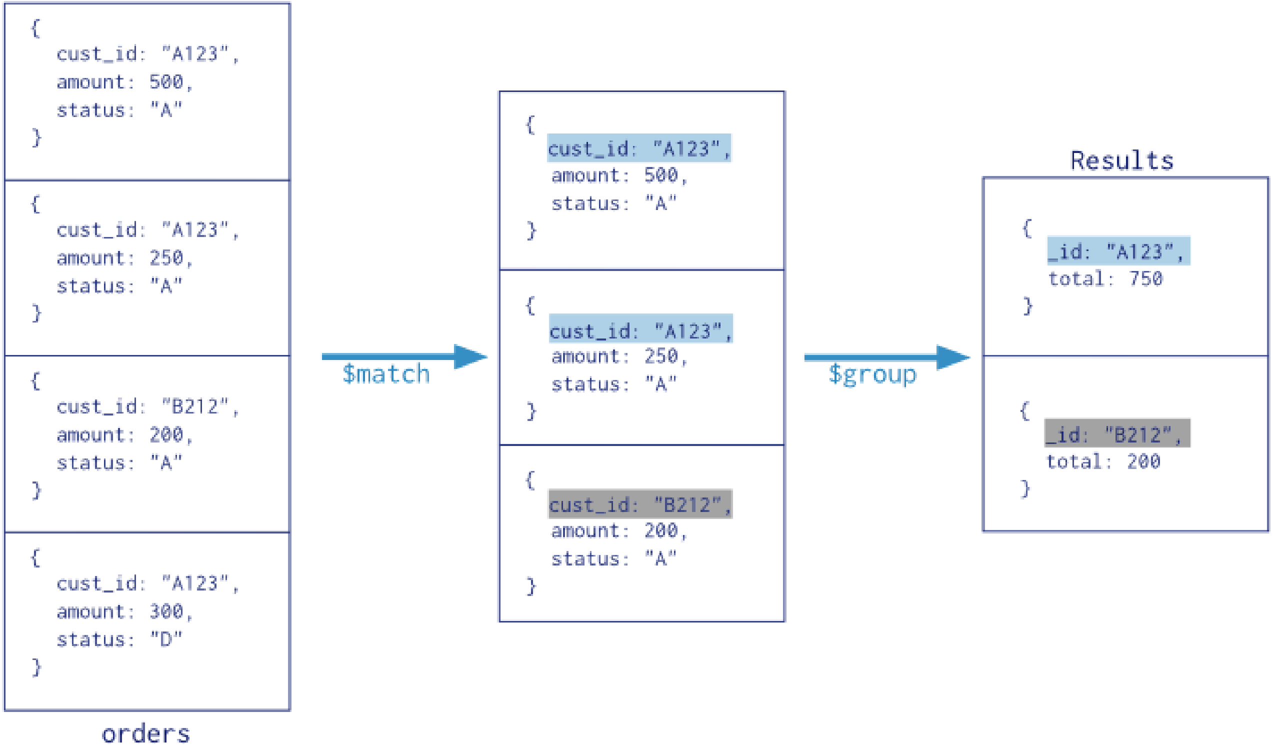
- Building index with embedded key

```
db.media.insert( { "Type" : "CD", "Artist" :
 "Nirvana","Title" : "Nevermind",
  "Tracklist"
   : [ {"Track" : "1",
        "Title" : "Smells like teen spirit",
       "Length“ : "5:02" },
       {"Track"  : "2",
        "Title"  : "In Bloom",
        "Length“ : "4:15" } ] } )
```

```
 db.media.ensureIndex( { "Tracklist.Title" :
      1 } )
```

Aggregation

```
   Collection
       ↓
db.orders.aggregate( [
     $match stage ──→   { $match: { status: "A" } },
     $group stage ──→   { $group: { _id: "$cust_id",total: { $sum: "$amount" } } }
                    ] )
```



```
{
   cust_id: "A123",
   amount: 500,
   status: "A"
}
{
   cust_id: "A123",
   amount: 250,
   status: "A"
}
{
   cust_id: "B212",
   amount: 200,
   status: "A"
}
{
   cust_id: "A123",
   amount: 300,
   status: "D"
}
```

orders

$match

```
{
   cust_id: "A123",
   amount: 500,
   status: "A"
}
{
   cust_id: "A123",
   amount: 250,
   status: "A"
}
{
   cust_id: "B212",
   amount: 200,
   status: "A"
}
```

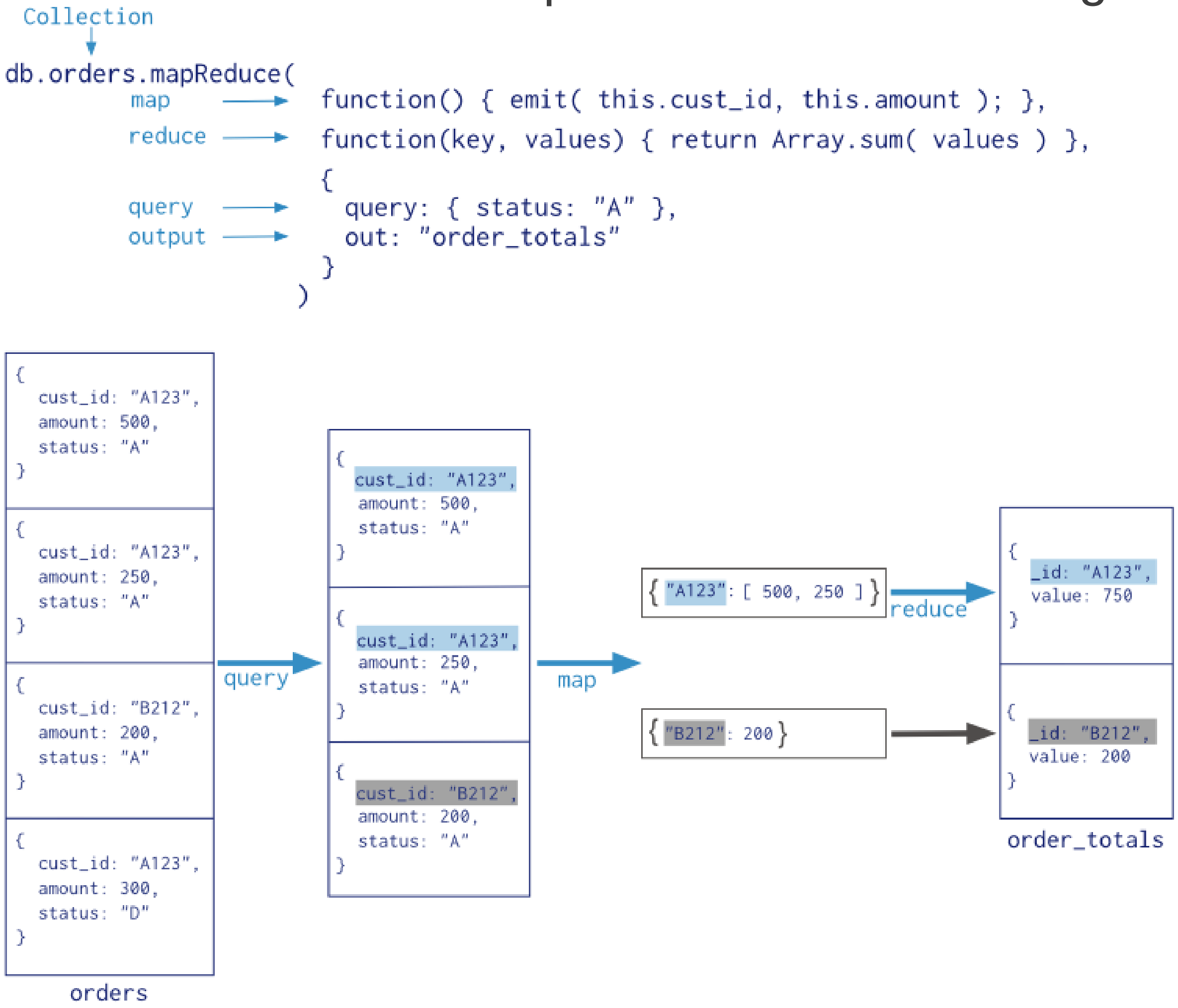
$group

Results

```
{
   _id: "A123",
   total: 750
}
{
   _id: "B212",
   total: 200
}
```

Built-in Map Reduce

- For more complex aggregation
  - Outside of the intended scope of this course – but good to know



```
db.orders.mapReduce(
                 function() { emit( this.cust_id, this.amount ); },
                 function(key, values) { return Array.sum( values ) },
                 {
                   query: { status: "A" },
                   out: "order_totals"
                 }
)
```

orders

```
{
  cust_id: "A123",
  amount: 500,
  status: "A"
}
{
  cust_id: "A123",
  amount: 250,
  status: "A"
}
{
  cust_id: "B212",
  amount: 200,
  status: "A"
}
{
  cust_id: "A123",
  amount: 300,
  status: "D"
}
```

```
{
  cust_id: "A123",
  amount: 500,
  status: "A"
}
{
  cust_id: "A123",
  amount: 250,
  status: "A"
}
{
  cust_id: "B212",
  amount: 200,
  status: "A"
}
```

```
{ "A123": [ 500, 250 ] }
{ "B212": 200 }
```

```
{
  _id: "A123",
  value: 750
}
{
  _id: "B212",
  value: 200
}
```

order_totals

Relational DB vs MongoDB

Relational DB	MongoDB
Table	Collection
Row	Document
Column	Field
Joins	Embedded documents; or linking (through foreign key)

Ref: *https://docs.mongodb.com/v3.0/reference/sql-comparison/*

Design with MongoDB

- Depending on the nature of data and frequency of different access, different design approaches can be adopted
  - 1-to-1 relations to embedded document
  - 1-to-many relations to embedded document
  - 1-to-Many to embedded model
  - Rolling up Documents

Model Relationships - 1

- 1-to-1 to embedded model
  - If data is frequently retrieve, apply embedded data model

```
{ _id: "joe",
 name: "Joe Bookreader" }

{ patron_id: "joe",
 street: "123 Clementi Road",
 city: “Singapore“,
 postalCode: “223450" }
```

1-to-1

Embedded document

```
{ _id: "joe",
 name: "Joe Bookreader",
  address:
         {street: " 123 Clementi Road ",
          city: “Singapore“,
               postalCode: “223450"  } }
```

Model Relationships - 2

- 1-to-Many to embedded model
  - If data is frequently retrieved, apply embedded data model

```
{ _id: "joe",
  name: "Joe  Bookreader" }

{ patron_id: "joe",
 street: "123 Clementi Road",
 city: “Singapore“,
 postalCode: “223450" }

{ patron_id: "joe",
 street: “18 Eden Lane ",
 city: “Singapore“,
 postalCode: “213456" }
```

1-to-Many

Embedded document

```
{ _id: "joe",
 name: "Joe Bookreader",
 addresses: [
         {street: "123 Clementi Road",
          city: “Singapore“,
          postalCode: “223450" },
         {street: “18 Eden Lane ",
          city: “Singapore”,
          postalCode: “213456”} ]  }
```

Model Relationships - 3

- 1-to-Many to reference model
  - Avoid growing array

```
{ name: "O'Reilly Media",
  founded: 1980,
  location: "CA",
  books: [12346789,
          234567890, ...] }
 { _id: 123456789,
  title: "MongoDB: The Definitive Guide",
   author: [ "Kristina Chodorow", "Mike Dirolf" ],
   published_date:
    ISODate("2010-09-24"),
    pages: 216,
    language: "English" }
 { _id: 234567890,
  title: "50 Tips and Tricks for MongoDB Developer",
  author: "Kristina Chodorow",
  published_date: ISODate("2011-05-06"),
  pages: 68,
  language: "English" }
```

Growing array

Model Relationships – 3 (cont’d)

- 1-to-Many to reference model
  - Avoid growing array : model with reference

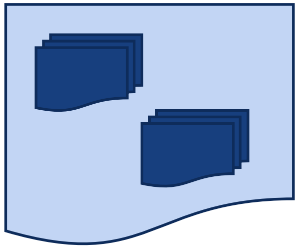
```
{ _id: "oreilly",
  name: "O'Reilly Media",
  founded: 1980,
  location: "CA" }
```

```
{ _id: 123456789,
  title: "MongoDB: The Definitive Guide",
  author: [ "Kristina Chodorow", "Mike Dirolf" ],
  published_date: ISODate("2010-09-24"),
  pages: 216,
  language: "English",
  publisher_id: "oreilly" }

{ _id: 234567890,
 title: "50 Tips and Tricks for MongoDB Developer",
 author: "Kristina Chodorow",
 published_date: ISODate("2011-05-06"),
 pages: 68, language: "English",
 publisher_id: "oreilly" }
```

Encapsulating Documents

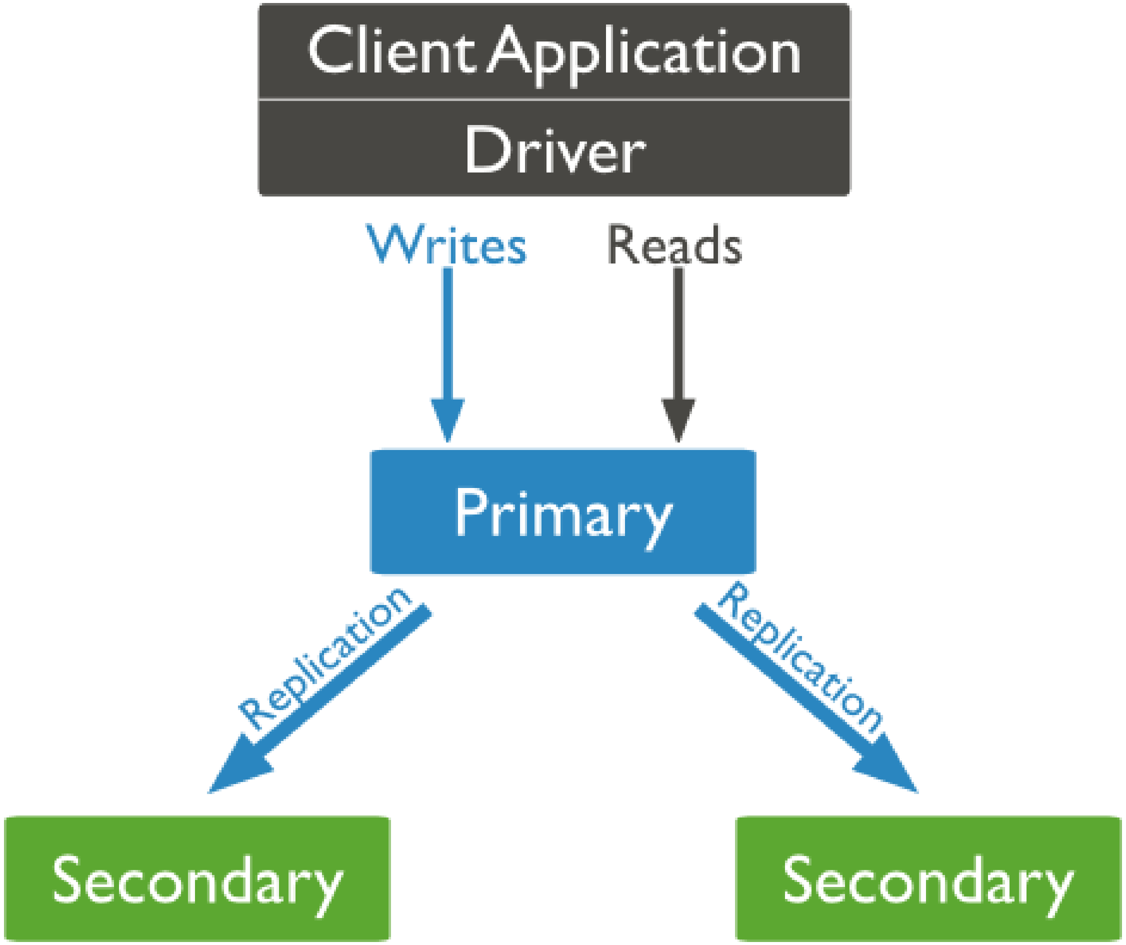
- Rolling-up documents
  - Logically group small documents
- Advantages
  - Improve retrieval performance
  - Common fields moved to larger document
    - Fewer copies of common fields
- Caveats
  - May not provide better performance for frequent retrieval of a subset of documents
- Tips
  - model small, separate documents that represent the <u>natural model of the data</u>



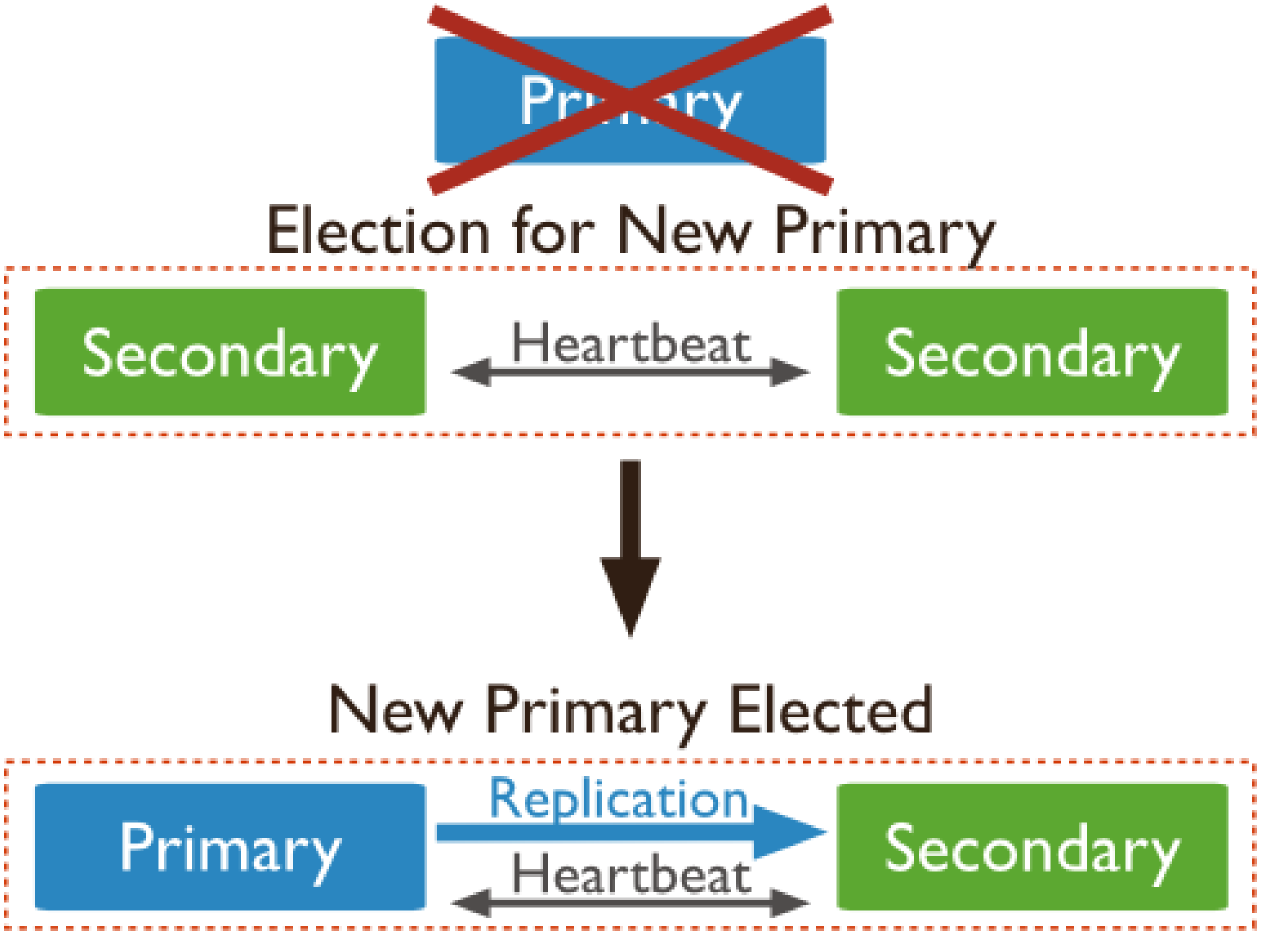
Replication

- Replication allow redundancy, failover and load balancing
- Replica sets is the recommended strategy
- Minimum recommended replica set consists of three nodes
- Primary is the only member in the set that can accept write operations
- Secondary replicates data from primary and can process read operations

Replication



New Primary Election



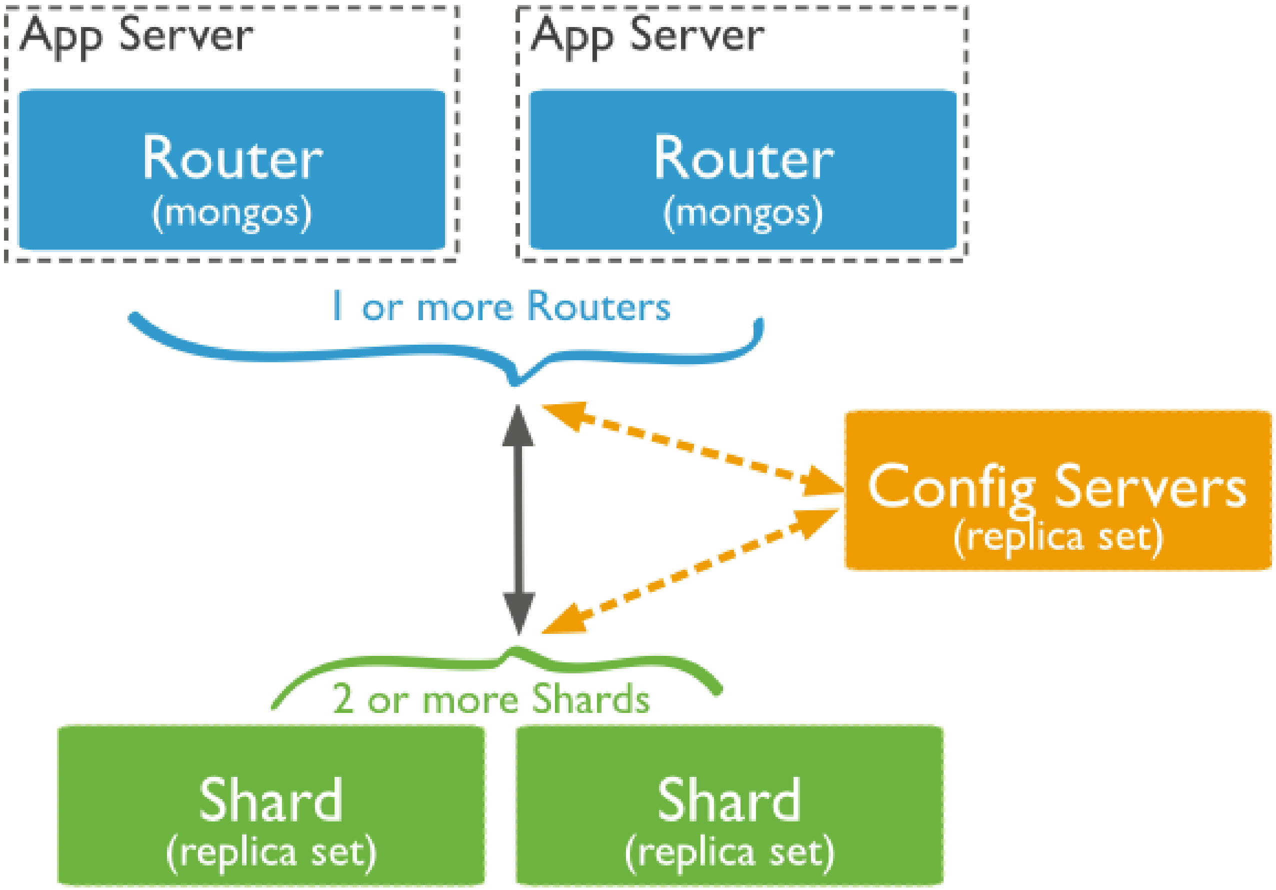
Minority and Majority Rule

- Network partitioning may occurs
  - The nodes are separated into 2 or more connected network
- When the primary detects that it can only see a minority of nodes in the replica set,
  - the primary steps down as primary and becomes a secondary.
- Independently, a member in the partition that can communicate with a majority of the nodes (including itself) holds an election to become the new primary.

Sharding

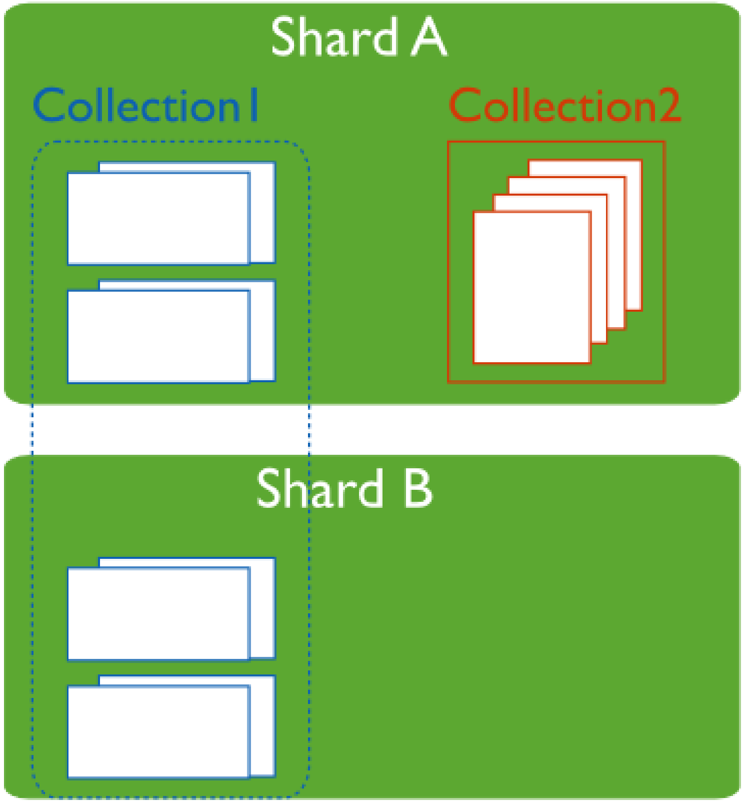
- Sharding is distributing data across multiple machine
- Typically done in deployments with very large data sets and/or very high throughput operations
- Requires 3 components
  - Shard: contain the data – can be deployed as replica set
  - Mongos: query router
  - Config servers: store metadata and setting for the cluster – can be deployed as replica set

Sharding


App Server
Router
(mongos)
App Server
Router
(mongos)
1 or more Routers
Config Servers
(replica set)
2 or more Shards
Shard
(replica set)
Shard
(replica set)

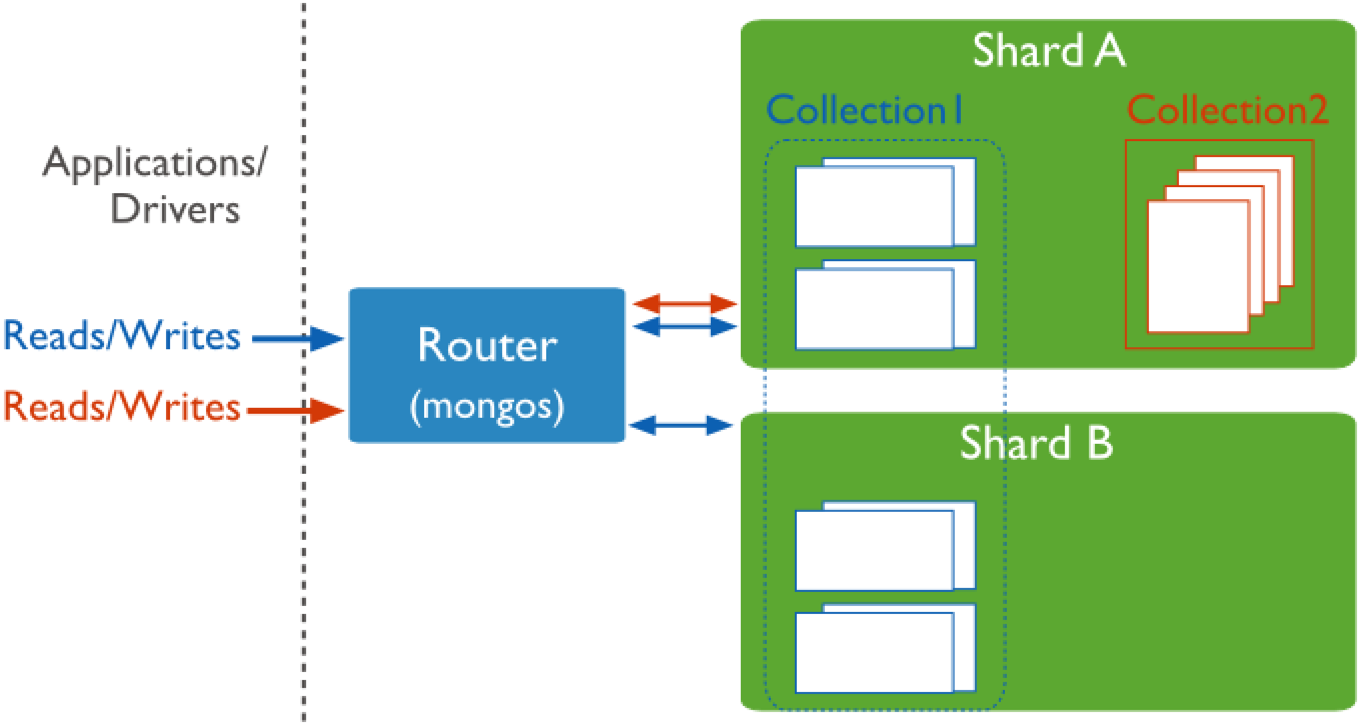
Sharded collections

- Sharding is done at collection level – so you can have sharded and unsharded collections



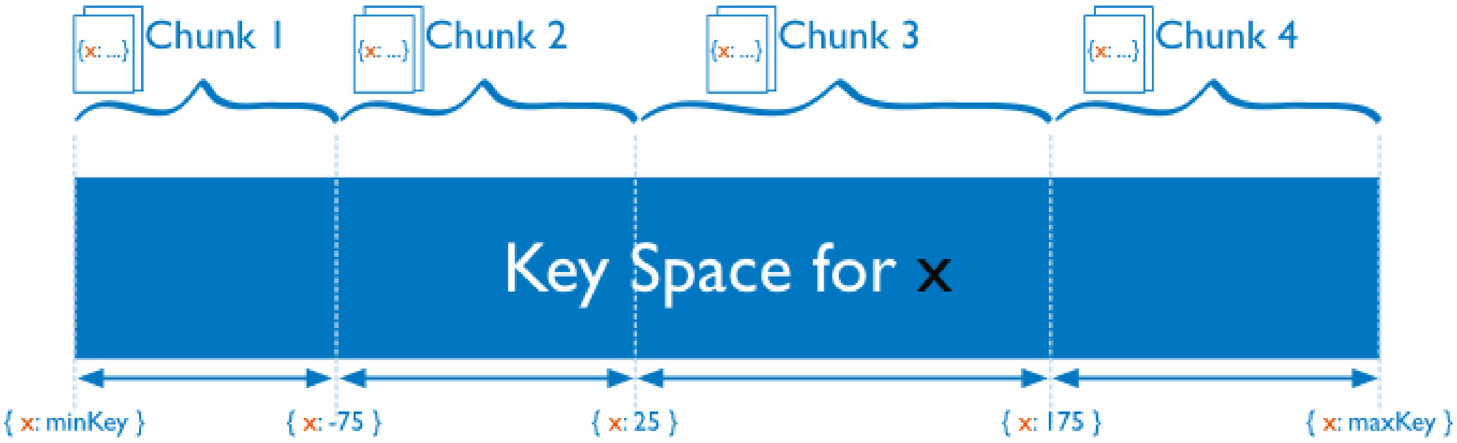
Accessing sharded data

- Client should access data through mongos router to ensure that the query is routed to the right instance



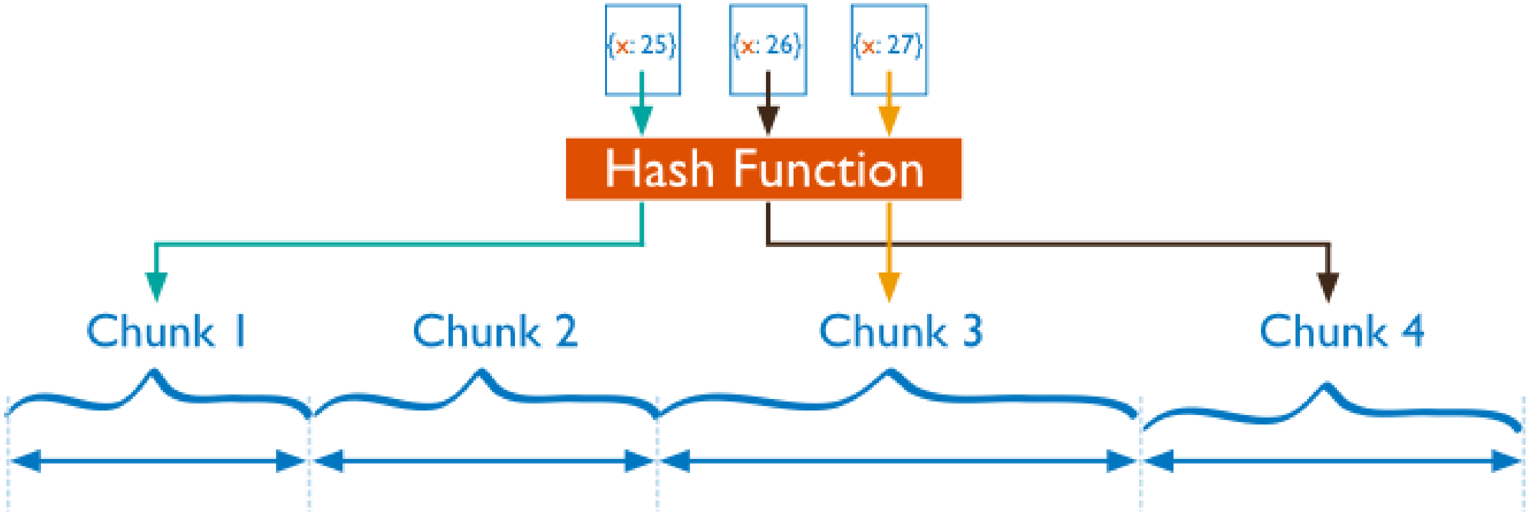
Ranged Sharding

- Default methodology
- Data is divided into contiguous ranges based on the value of the shard key



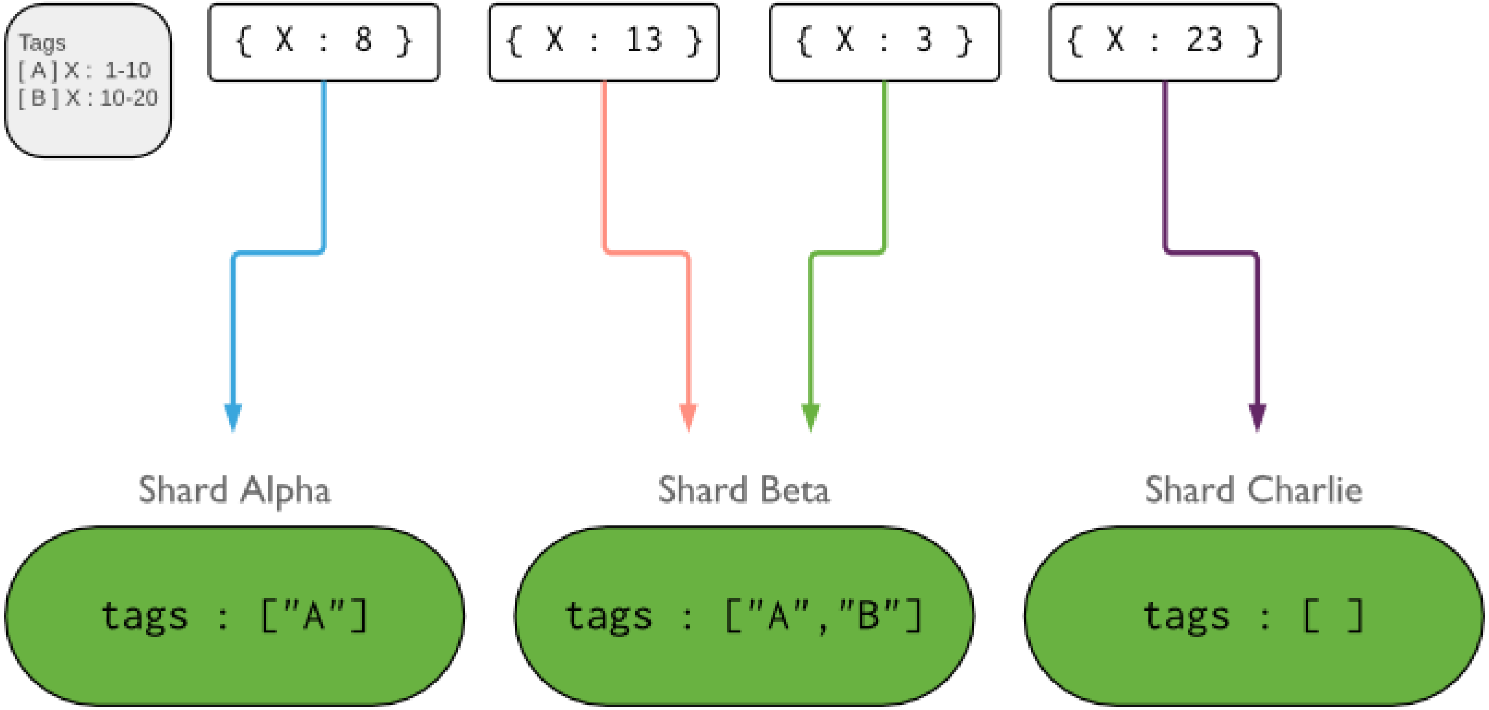
Sharding Strategy

- Data is divided based on the hashed value of the shard key
- Result in a more even data distribution across cluster
  - Data distribution will affect the throughput distribution



Tag Aware Sharding

- Allow for more finer control on the location of the data
- E.g. for ensure geographical closeness of the data



Atomicity and Transactions

- MongoDB provides features to implement two-phase commit in your **application** for atomic multi-document updates.
  - Not as convenient as transaction support in traditional RDBMS

Consistency

- In a multi instances deployment, it is possible to read “stale” data from one of the replica
- MongoDB allow some settings for
  - readConcern=majority query option
    - Return only data that has been written to majority of the members in the replica set
  - writeConcern=majority query option
    - Acknowledge the write only after the write has been accepted by majority of the members in the replica set

Summary

- MongoDB is one example of document database
- Support more complex document structure compared to a typical RDBMS record
- More flexibility – more design decisions
- High availability and redundancy is achieved through sharding and replication

References

- MongoDB Documentation
  - https://docs.mongodb.com
- MongoDB Reference Card
  - https://www.mongodb.com/collateral/quick-reference-cards
- MongoDB in Action, Second Edition
  - Kyle Banker, Peter Bakkum, Shaun Verch, Doug Garret, Tim Hawkins, Manning, 2016