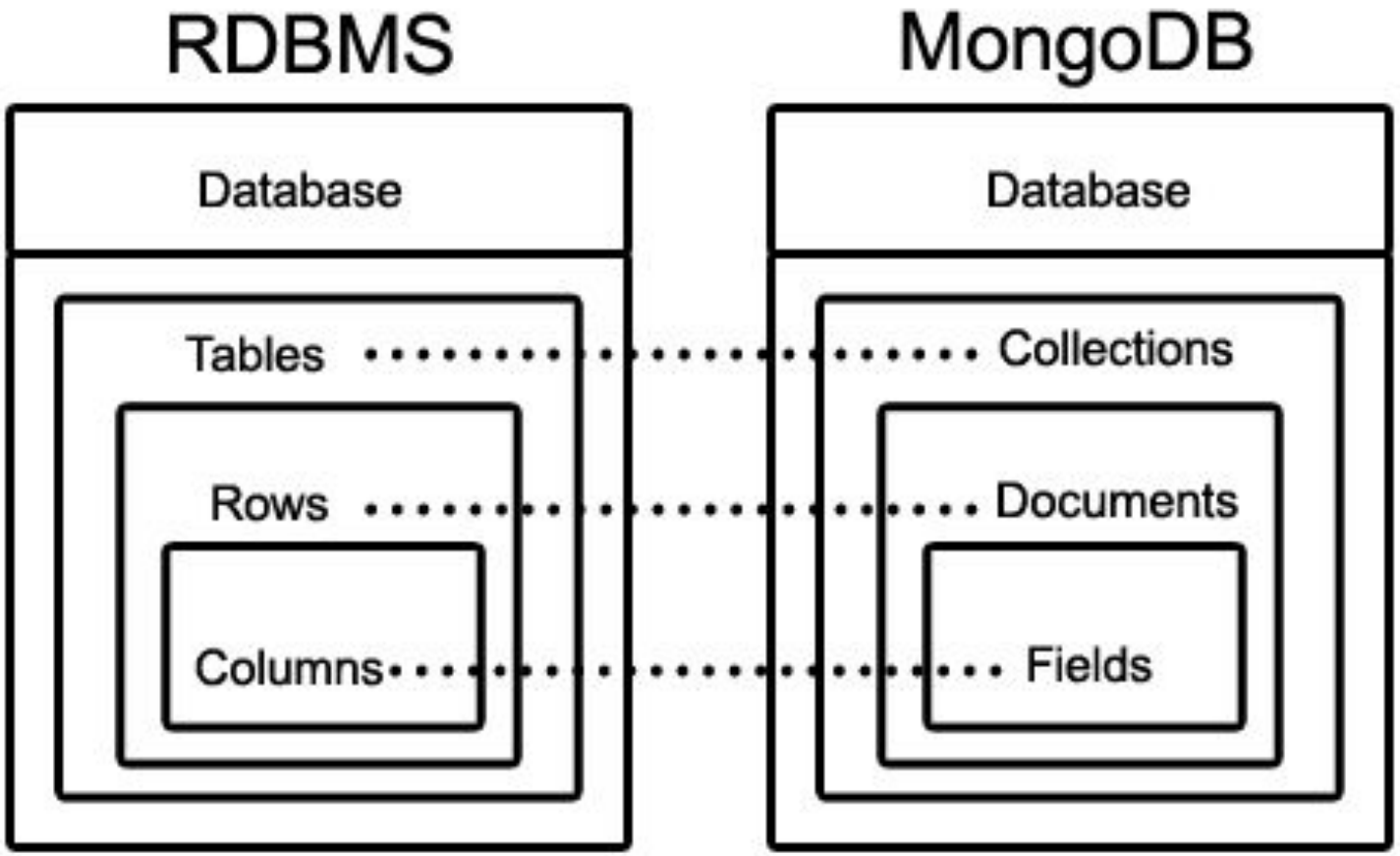
MongoDB

- MongoDB is **a document database that provides high performance, high availability and easy scalability**.
- A document database is **a type of non-relational database that is designed to store and query data as JSON-like documents**.
- MongoDB works on concept of collection and document.
- MongoDB is,
  - Cross-platform
  - Open source
  - Non-relational
  - Distributed
  - Document-oriented data store
  - NoSQL

MongoDB


RDBMS
MongoDB
Database
Database
Tables
Collections
Rows
Documents
Columns
Fields

INTRODUCTION

- MongoDB is an open source database that uses a document-oriented data model.

- MongoDB is one of several database types to arise in the mid-2000s under the NoSQLbanner. Instead of using tables and rows as in relational databases, MongoDB is built on an architecture of collections and documents.

CONTINUED....

- Documents comprise sets of key-value pairs and are the basic unit of data in MongoDB.

- Collections contain sets of documents and function as the equivalent of relational database tables.

- MongoDB was **founded in 2007 by Dwight Merriman, Eliot Horowitz and Kevin Ryan.**
- The first version of the MongoDB database shipped in **August 2009**.
- The 1.0 release focused on validating a new and larger database design — built on a JSON-like document data model and layered onto an elastic and distributed systems foundation.
- MongoDB has become one of the most wanted databases in the world because,
- **it makes it easy for developers to store, manage, and retrieve data when creating applications with most programming languages**.

MONGO DB

- Developed by 10gen
  - Founded in 2007
- A document-oriented, NoSQL database
  - Hash-based, *schema-less database*
  - No Data Definition Language
  - In practice, this means it can store hashes with any keys and values.
  - Keys are a basic data type but in reality stored as strings

Contd,

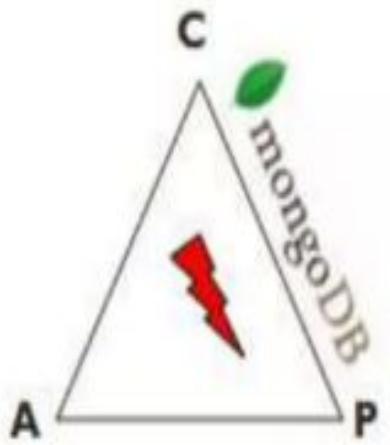
- Application tracks the schema and mapping
- Based on JSON – B, B stands for Binary
- Written in C++
- Supports APIs (drivers) in many computer languages,
  - JavaScript, Python, Ruby, Perl, Java, Scala, C#, C++, Haskell, Erlang.

Functionality of MongoDB

- Dynamic schema
- No DDL
- Document-based database
- Secondary indexes
- Query language via an API
- Atomic writes and fully-consistent reads
- Master-slave replication with automated failover (replica sets)

MongoDB: CAP approach

- Focus on Consistency and Partition tolerance
- **Consistency:**
  - All replicas contain the same version of data
- **Availability**
  - System remains operational on failing nodes
- **Partition tolarence**
  - Multiple entry points
  - System remains operational on system split



CAP Theorem:
satisfying all three at the same time is impossible

FEATURES OF MongoDB ...

1

2

3

SCHEMA FREE

Features

- Document-Oriented storege
- Open source
- Full Index Support
- Replication & High Availability
- Auto-Sharding
- Querying
- Fast In-Place Updates
- Map/Reduce functionality.

Contd,

- Easy to install and use
- Detailed documentation
- Various APIs
- Community

ySQL Vs MongoDB

MySQL

```
START TRANSACTION;
INSERT INTO contacts VALUES
    (NULL, 'joeblow');
INSERT INTO contact_emails VALUES
   ( NULL, "joe@blow.com",
      LAST_INSERT_ID() ),
   ( NULL, "Joseph@blow.com",
      LAST_INSERT_ID() );
COMMIT;
```

MongoDB

```
db.contacts.save( {
    userName: "joeblow",
    emailAddresses: [
      "joe@blow.com",
      "joseph@blow.com" ] } );
```

Why MongoDB

- MongoDB is built on a scale-out architecture that has become popular with developers of all kinds for developing scalable applications with evolving data schemas.
- As a document database, **MongoDB makes it easy for developers to store structured or unstructured data**.
- MongoDB is **a NoSQL database management application**.
- NoSQL database systems offer an alternative to traditional relational databases using SQL (Structured Query Language).
- Data is stored in tables, rows, and columns in a relational database, with relationships between entities.

Why MongoDB


Aggregation
High Performance
GridFS
Ad Hoc Queries
Sharding
mongoDB
Indexing
Document oriented
Schema Less Database
Replication

DOCUMENT

MongoDB is a **document**-oriented database. Instead of storing your data in tables **made** out of individual rows, like a relational database does, it stores your data in collections **made** out of individual **documents**.

In **MongoDB**, a **document** is a big JSON blob with no particular format or schema.N

A document in MongoDB is like a JSON.
Example:

```
{'name': 'Christiano',
 'language': 'Python',
 'country': 'Brazil'}
```

YOU CAN DYNAMICALLY UPDATE YOUR DATA

, example:

```
{'name': 'Christiano',
  'language': 'Python',
  'country': 'Brazil'}

{'name': 'Christiano',
  'language': 'Python',
  'country': 'Brazil',
  'event': 'PyConAr'}
```

Databases

In MongoDB, **databases** hold collections of documents.

`use` <db> statement,

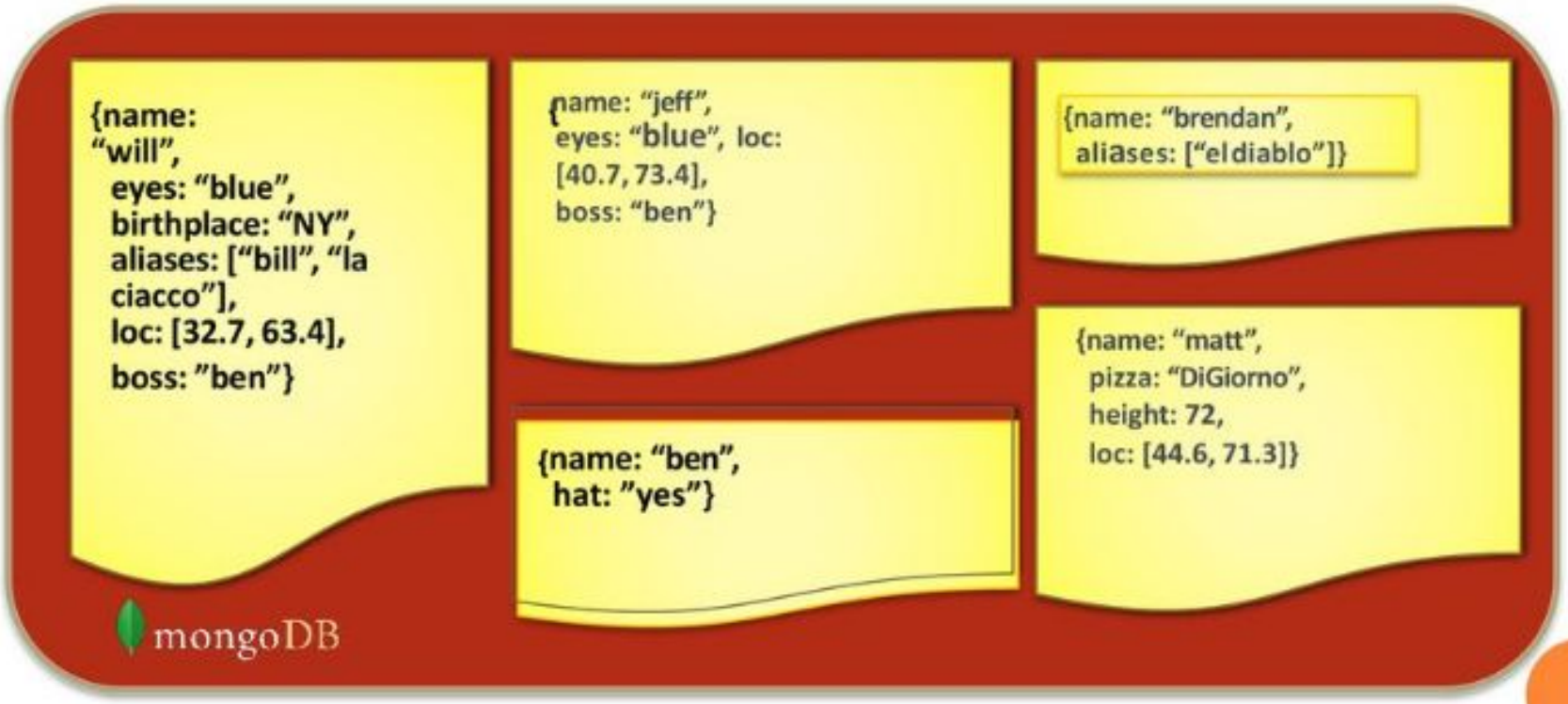
use myDB

Creation of Database

- If a database does not exist, MongoDB creates the database when you first store data for that database.

SCHEMA FREE

- MongoDB does not need any pre-defined data schema
- Every document in a collection could have different data
  - Addresses NULL data fields



JSON FORMAT

- Data is in name / value pairs
- A name/value pair consists of a field name followed by a colon, followed by a value:
  - Example: “name”: “R2-D2”
- Data is separated by commas
  - Example: “name”: “R2-D2”, race : “Droid”
- Curly braces hold objects
  - Example: {“name”: “R2-D2”, race : “Droid”, affiliation: “rebels”}
- An array is stored in brackets []
  - Example [{“name”: “R2-D2”, race : “Droid”, affiliation: “rebels”},
  - {“name”: “Yoda”, affiliation: “rebels”}]

INSERTION OF DOCUMENTS

Inserting Single Documents

```
db.inventory.insertOne
({ item: "canvas", qty: 100,tags: ["cotton"],
size:{h: 28, w: 35.5, unit: "cm" } } )
```

Inserting Multiple Documents

db.collection.insertMany():

can insert *multiple* documents into a collection. Pass an array of documents to the method.

EXAMPLE : INSERT MULTIPLE DOCUMENTS

```
db.inventory.insertMany
([{ item: “Freeze“,qty: 25, tags: ["blank", "red"],size:{ h: 14, w: 21, unit: "cm" }},
  { item: “TV“,qty: 85, tags: ["gray"],size:{ h: 27.9, w: 35.5, unit : "cm" } },
  { item: “AC“,qty:25,tags: [“white”], "blue"],size:{ h: 19, w: 22.85, unit:"cm" } }
])
```

Query Operators

Name	Description
$eq	Matches value that are equal to a specified value
$gt, $gte	Matches values that are greater than (or equal to a specified value
$lt, $lte	Matches values less than or (equal to) a specified value
$ne	Matches values that are not equal to a specified value
$in	Matches any of the values specified in an array
$nin	Matches none of the values specified in an array
$or	Joins query clauses with a logical OR returns all
$and	Join query clauses with a loginal AND
$not	Inverts the effect of a query expression
$nor	Join query clauses with a logical NOR
$exists	Matches documents that have a specified field

MongoDB sample Commands:

Ques. Show all databases:

```
show dbs
```

Ques. Create a Database Named 'workshop':

```
use workshop
```

Ques. Creating a collection/table named 'student' in 'workshop' database:

```
db.createCollection("student")
```

Ques. Inserting a document/record in 'student':

```
db.student.insertOne({RollNo:1, Name:"Aashna", Marks:80})
```

Ques. Inserting multiple documents/records in 'student':

```
db.student.insertMany( [{RollNo:2, Name:"Abhinav", Marks:70},
{RollNo:3, Name:"Ayesha", Marks:85},
{RollNo:4, Name:"Mohit", Marks:72} ] )
```

Ques. Display all records/documents from 'student':

```
db.student.find()
```

Ques. Display details of students having Marks more than 80:

```
db.student.find( { Marks : { $gt : 80} } )
```

Lets Relate the MongoDB queries with SQL

- db.users.insert({a:1,b:2})

- Insert into Users Values(1,2)

- db.users.find( )

- Select * from users;

- db.users.find({age:33})

- Select * from users where age=33

- db.users.find({'name':'Manasvi',Age:12})

- Select * From Users where name='Manasvi' and Age=12;

- db.users.find({‘a:{$gt:22}})

- Select * from users where a>22;

- db.users.sort({name:-1})

- Select * from users order by name desc;

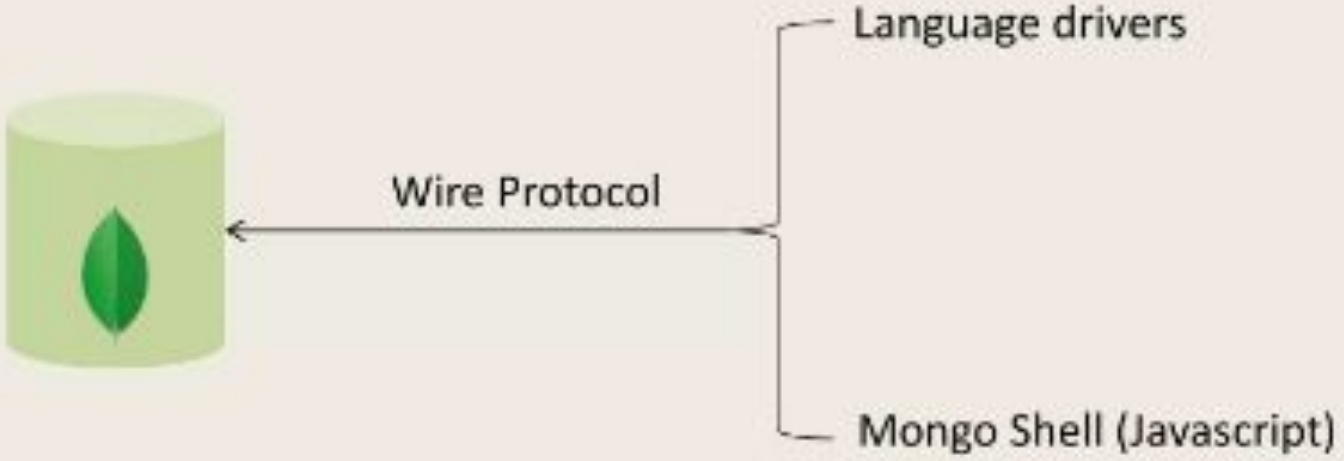
WHAT IS JSON?

JSON is acronym of Java Object Notation which is a format for structuring data to transmit data between server and web application

CRUD in MongoDB

Operation	MongoDB	SQL
Create	Insert	Insert
Read	Find	Select
Update	Update	Update
Delete	Remove	Delete

MongoDB's CRUD operations exist as methods/functions in programming languages, **not as** a separate query language like SQL.



CRUD

- Create
  - db.collection.insert(<document>)
  - db.collection.save(<document>)
- Read
  - db.collection.find(<query>, <projection>)
  - db.collection.findOne(<query>, <projection>)

Contd,

- Update
  - db.collection.update(<query>, <update>, <options>)
- Delete
  - db.collection.remove(<query>, <justOne>)

Create Database

- The **use** Command,

  MongoDB **use DATABASE_NAME** is used to create database. The command will create a new database if it doesn't exist, otherwise it will return the existing database.

- **Syntax**

```
use DATABASE_NAME
```

Contd,

- To check the currently selected database, use the command **db.**
  - **Syntax**

    >db
- To check all database lists use the command
  - **Syntax**

    >show dbs
- In MongoDB default database is **test**. If you didn't create any database, then collections will be stored in test database.

Drop Database

- MongoDB **dropDatabase()** command is used to drop a existing database.
- **Syntax**

  db.dropDatabase()

- This will delete the selected database.
- If any database is not selected, then it will delete default 'test' database.

```
> db.stats()
{
        "db" : "test",
        "collections" : 0,
        "objects" : 0,
        "avgObjSize" : 0,
        "dataSize" : 0,
        "storageSize" : 0,
        "numExtents" : 0,
        "indexes" : 0,
        "indexSize" : 0,
        "fileSize" : 0,
        "ok" : 1
}
>
> use student
switched to db student
> db
student
> show dbs
local  0.000GB
> db.dropDatabase()
{ "ok" : 1 }
> show dbs
local  0.000GB
```

Create Collection

- MongoDB **createCollection** is used to create collection.
  - **Syntax**

    db.createCollection(name, options)

    - ✓ **name** is name of collection to be created.
    - ✓ **Options** is a document and is used to specify configuration of collection like memory size and indexing.

- To check the created collection by using the command **show collections**.

Method 1:


```
> use student
switched to db student
> db.create
student.create
> db.createCollection("MyClass")
{ "ok" : 1 }
> show collections
MyClass
```

Method 2:


```
> db.NameList.insert({"name": "mca"})
WriteResult({ "nInserted" : 1 })
> show collections
MyClass
NameList
```

Drop a Collection

- MongoDB's **collection.drop()** is used to drop a collection from the database.

- **Syntax**

  db.COLLECTION_NAME.drop()


```
> use student
switched to db student
> db.MyClass.drop()
true
> show collections
NameList
>
```

Dealing with NULL values

```
/* 1 */
{
    "_id" : ObjectId("5ee2444070dd9ee380ce40a5"),
    "name" : "Amy",
    "age" : NumberLong(52)
}

/* 2 */
{
    "_id" : ObjectId("5ee2444070dd9ee380ce40a6"),
    "name" : "Hannah",
    "age" : null
}
```

SQL vs NoSQL

- ✔ NoSQL (often interpreted as Not only SQL) database

- ✔ It provides a mechanism for storage and retrieval of data that is modeled in means other than the tabular relations used in relational databases.

SQL	NoSQL
Relational Database Management System (RDBMS)	Non-relational or distributed database system.
These databases have fixed or static or predefined schema	They have dynamic schema
These databases are best suited for complex queries	These databases are not so good for complex queries
Vertically Scalable	Horizontally scalable
Follows ACID property	Follows BASE property

MongoDB is Easy to Use

Relational

Person:

Pers_ID	Surname	First_Name	City
0	Miller	Paul	London
1	Ortega	Alvaro	Valencia
2	Huber	Urs	Zurich
3	Blanc	Gaston	Paris
4	Bertolini	Fabrizio	Rom

no relation

Car:

Car_ID	Model	Year	Value	Pers_ID
101	Bentley	1973	100000	0
102	Rolls Royce	1965	330000	0
103	Peugeot	1993	500	3
104	Ferrari	2005	150000	4
105	Renault	1998	2000	3
106	Renault	2001	7000	3
107	Smart	1999	2000	2

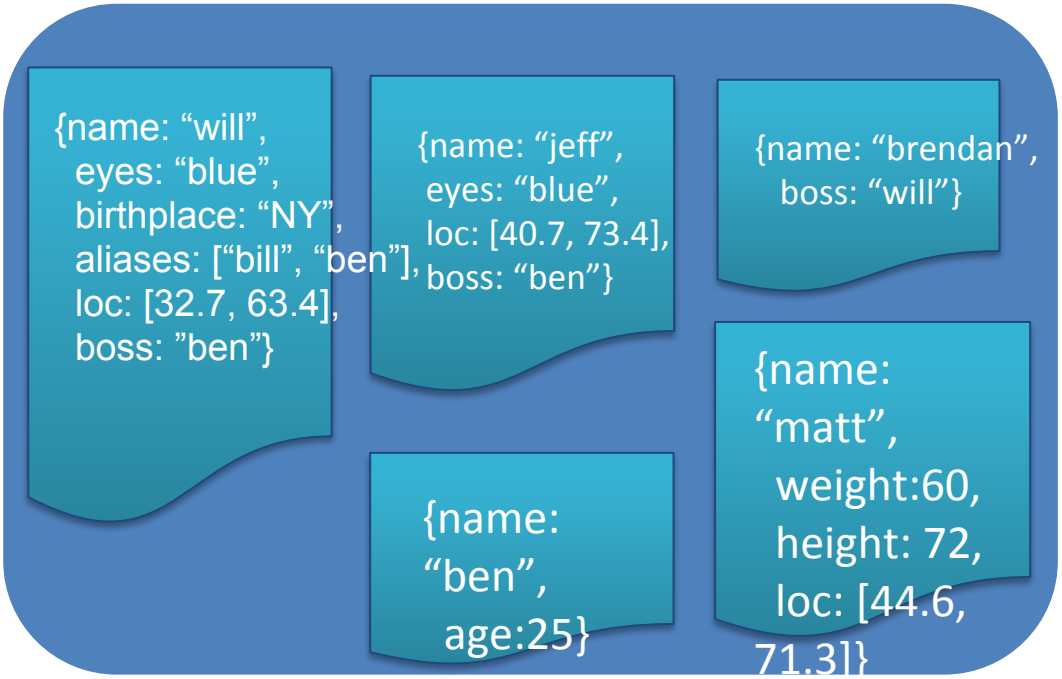
MongoDB Document

```
{
  first_name: 'Paul',
  surname: 'Miller'
  city: 'London',
  location: [45.123,47.232],
  cars: [
    { model: 'Bentley',
      year: 1973,
      value: 100000, … },
    { model: 'Rolls Royce',
      year: 1965,
      value: 330000, … }
  ]
}
```

Scheme Free

MongoDB does not need any pre-defined data schema

Every document could have different data!



RDBMS vs MongoDB

RDBMS		MongoDB
Database	⇨	Database
Table	⇨	Collection
Row	⇨	Document (JSON, BSON)
Column	⇨	Field
Index	⇨	Index
Join	⇨	Embedded Document
Partition		Shard

limit, skip, sort and count the results of the find() method

- Similar to aggregation methods also by the find() method you have the possibility to limit, skip, sort and count the results. Let say we have following collection:
- db.test.insertMany([
- {name:"Anu", age:"21", status:"busy"},
- {name:“Banu", age:"25", status:"busy"},
- {name:“Chitra", age:"28", status:"online"},
- {name:“Deva", age:"28", status:"away"},
- {name:“Eshwari", age:"20", status:"online"}
-])

limit, skip, sort and count the results of the find() method

- To list the collection:
- db.test.find({})
- Will return:
- { "_id" : ObjectId("592516d7fbd5b591f53237b0"), "name" : "Anu", "age" : "21", "status" : "busy" }
- { "_id" : ObjectId("592516d7fbd5b591f53237b1"), "name" : " Banu ", "age" : "25", "status" : "busy" }
- { "_id" : ObjectId("592516d7fbd5b591f53237b2"), "name" : " Chitra ", "age" : "28", "status" : "online" }
- { "_id" : ObjectId("592516d7fbd5b591f53237b3"), "name" : " Deva ", "age" : "28", "status" : "away" }
- { "_id" : ObjectId("592516d7fbd5b591f53237b4"), "name" : " Eshwari ", "age" : "20", "status" : "online" }

limit, skip, sort and count the results of the find() method

- To skip first 3 documents:
- db.test.find({}).skip(3)

Will return as,

- { "_id" : ObjectId("592516d7fbd5b591f53237b3"), "name" : " Deva ", "age" : "28", "status" : "away" }
- { "_id" : ObjectId("592516d7fbd5b591f53237b4"), "name" : " Eshwari ", "age" : "20", "status" : "online" }

limit, skip, sort and count the results of the find() method

- To sort descending by the field name:
- db.test.find({}).sort({ "name" : -1})
- { "_id" : ObjectId("592516d7fbd5b591f53237b1"), "name" : " Eshwari ", "age" : "25", "status" : "busy" }
- { "_id" : ObjectId("592516d7fbd5b591f53237b3"), "name" : " Deva ", "age" : "28", "status" : "away" }
- { "_id" : ObjectId("592516d7fbd5b591f53237b4"), "name" : " Chitra ", "age" : "20", "status" : "online" }
- { "_id" : ObjectId("592516d7fbd5b591f53237b2"), "name" : " Banu ", "age" : "28", "status" : "online" }
- { "_id" : ObjectId("592516d7fbd5b591f53237b0"), "name" : " Anu ", "age" : "21", "status" : "busy" }Will return:

limit, skip, sort and count the results of the find() method

- If you want to sort ascending just replace -1 with 1
- To count the results:
- db.test.find({}).count()
- Will return:
- 5
- Also combinations of this methods are allowed. For example get 2 documents from descending sorted collection skipping the first 1:
- db.test.find({}).sort({ "name" : -1}).skip(1).limit(2)

limit, skip, sort and count the results of the find() method

- Will return:
- { "_id" : ObjectId("592516d7fbd5b591f53237b3"), "name" : " Deva ", "age" : "28", "status" : "away" }
- { "_id" : ObjectId("592516d7fbd5b591f53237b4"), "name" : " Chitra ", "age" : "20", "status" : "online" }

Disadvantages

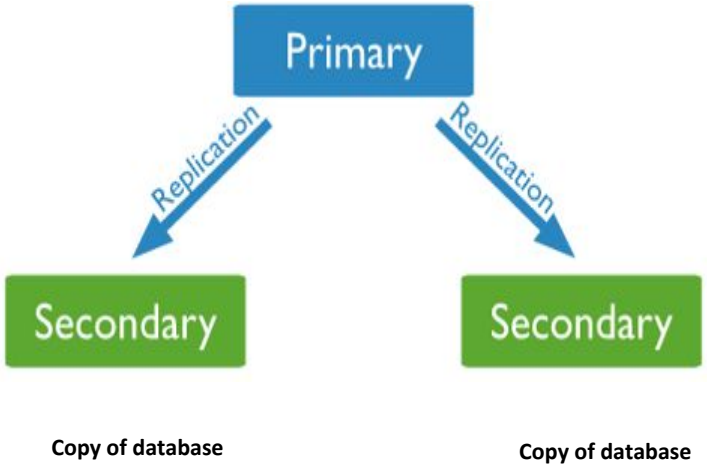
- Usage of memory- server will need lot of RAM
- Less flexibility with querying (eg. No joins)
- No Support for transactions.
- Concurrency issues

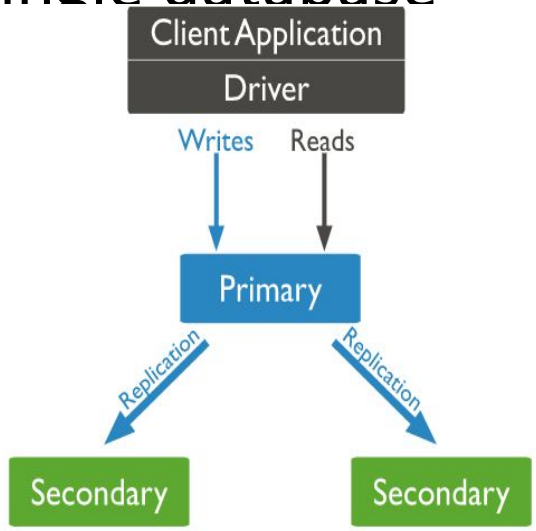
The process of REPLICATION in Mongodb

- In simple terms, MongoDB replication is the process of **creating a copy of the same data set in more than one MongoDB server**.
- This can be achieved by using a Replica Set. A replica set is a group of MongoDB instances that maintain the same data set and pertain to any mongodb process.
- In MongoDB, **copyTo() method** is used to copies all the documents from one collection(Source collection) to another collection(Target collection) using server-side JavaScript and if that other collection(Target collection) is not present then MongoDB creates a new collection with that name.
- Database replication is **the process of copying or transferring data from a database on one server to a database on another server to improve data availability and accessibility**.
- The process facilitates data sharing and data recovery.

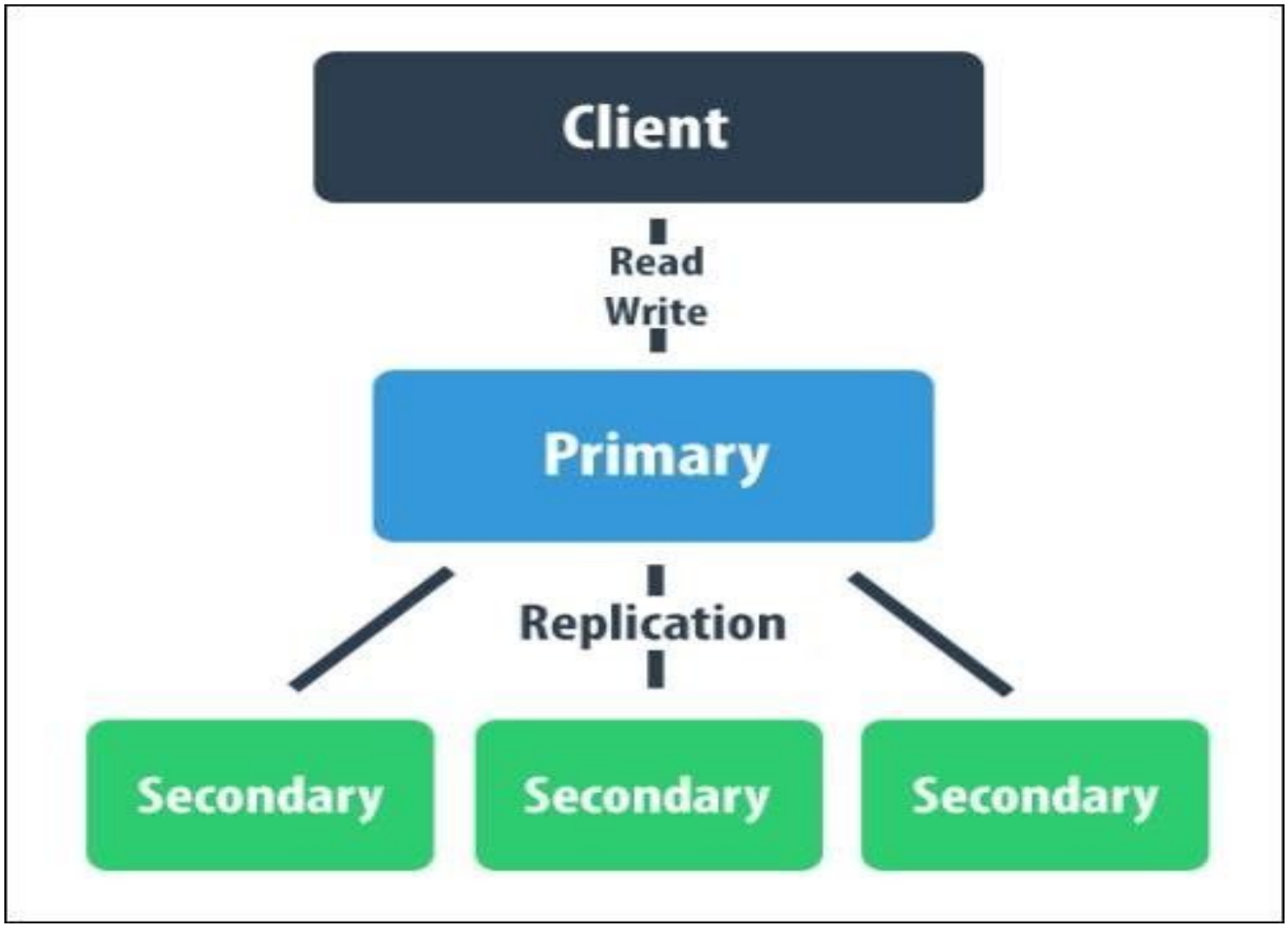
Replication

- Replication provides redundancy and increases data availability.
- With multiple copies of data on different database servers, replication provides a level of fault tolerance against the loss of a single database server.





The process of REPLICATION in Mongodb

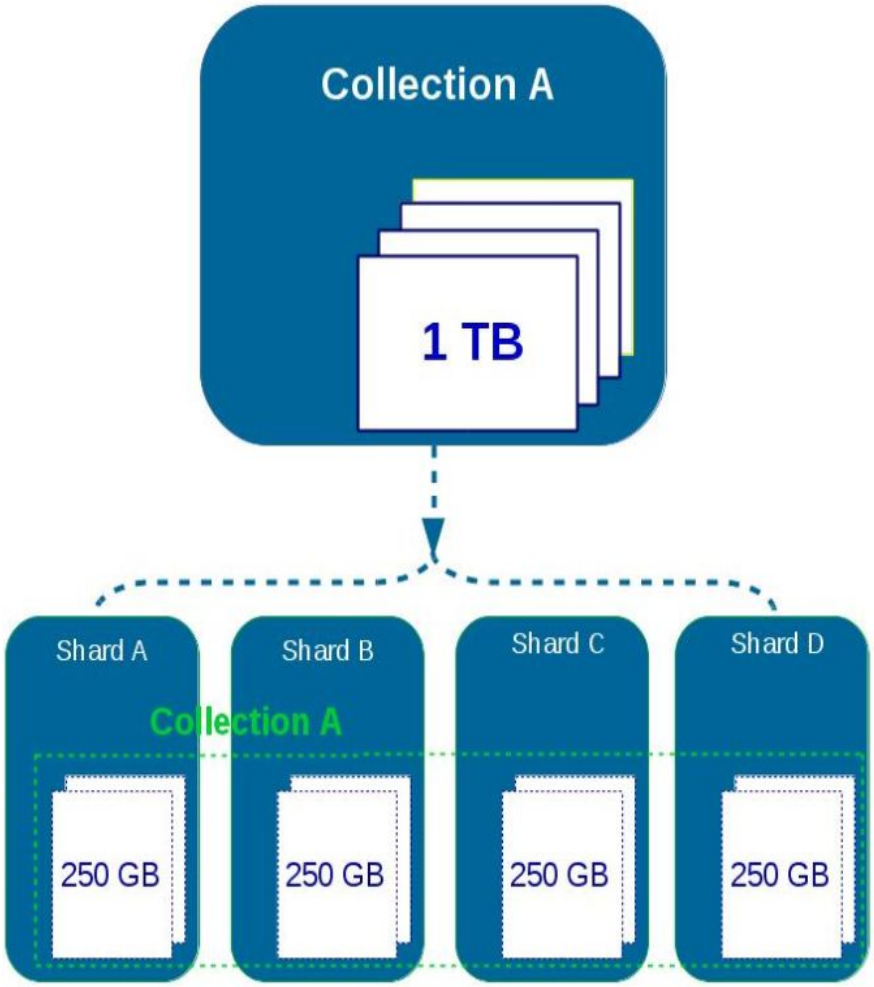


The process of SHARDING in Mongodb

- MongoDB uses the shard key, the large dataset is divided and distributed over multiple servers or shards.
- Each shards is an independent database and collectively they would continue a logical database.
- Database sharing **splits a single dataset into partitions or shards**.
- Each shard contains unique rows of information that you can store separately across multiple computers, called nodes.
- All shards run on separate nodes but share the original database's schema or design.
- **Sharing is a type of database partitioning that separates very large databases the into smaller, faster, more easily managed parts called data shards**.

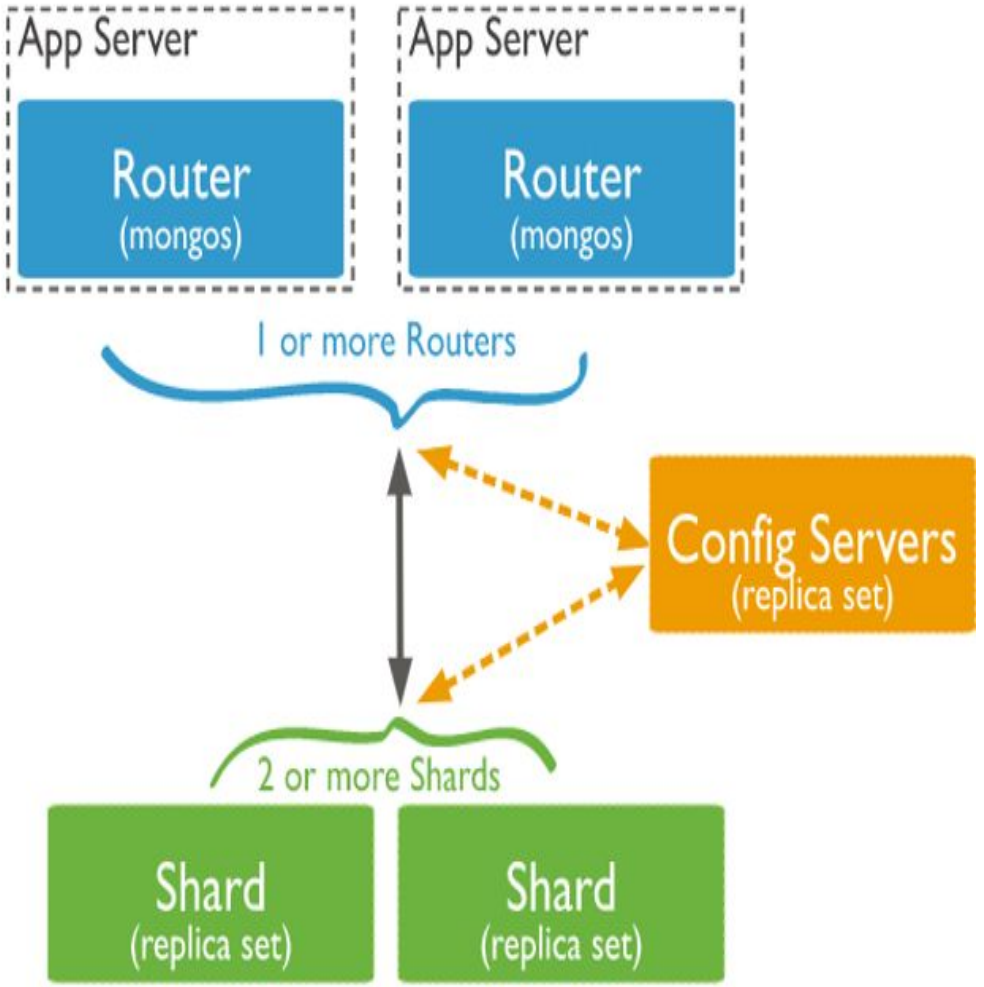
Sharding

- Sharding is a method for distributing data across multiple machines.
- MongoDB uses sharing to support deployments with very large data sets and **high throughput operations.**

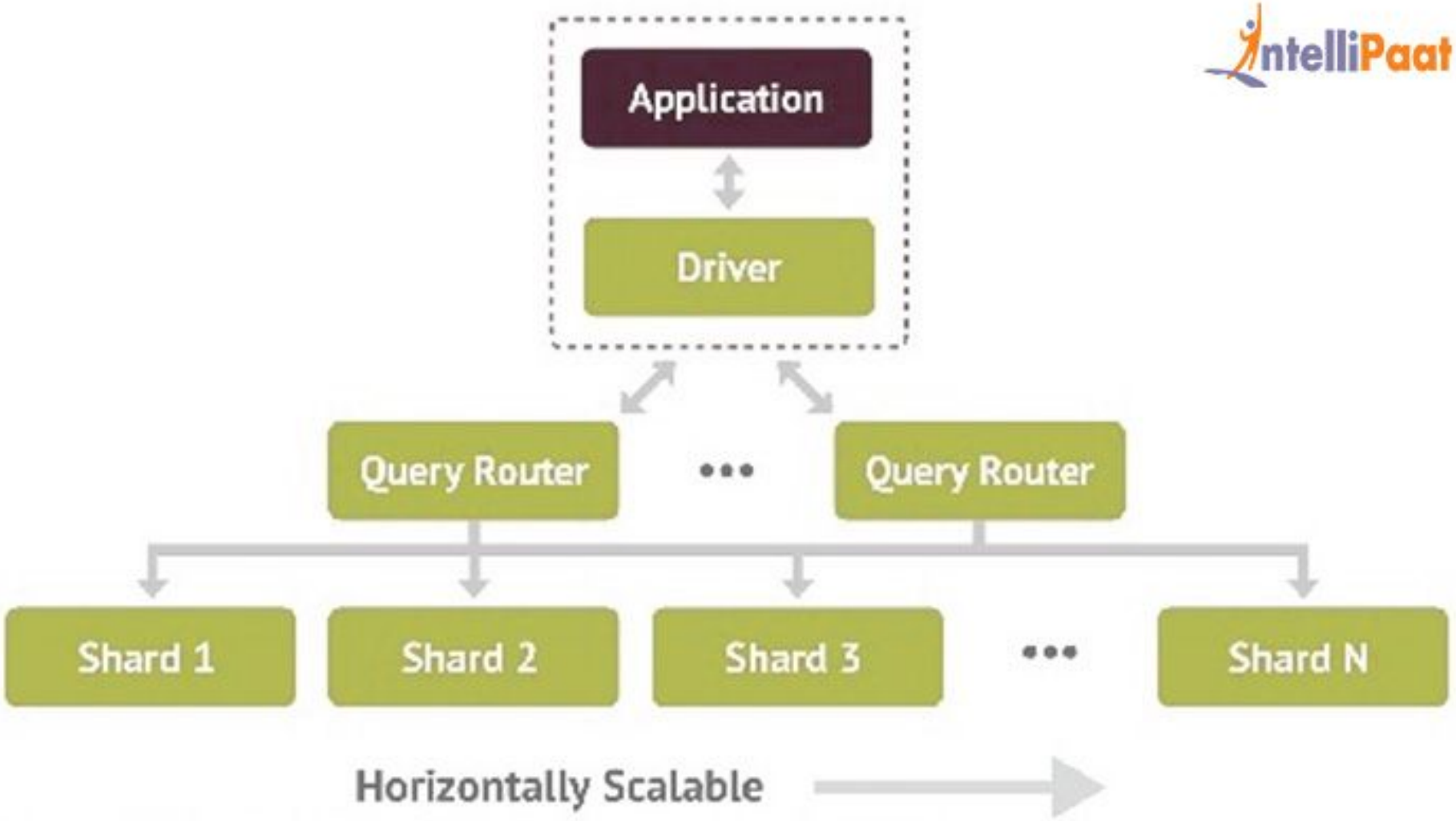


Sharding Architecture

- **Shard** is a Mongo instance to handle a subset of original data.
- **Mongos** is a query router to shards.
- **Config Server** is a Mongo instance which stores metadata information and configuration details of cluster.



The process of SHARDING in Mongodb



Document Store

RDBMS	MongoDB
Database	Database
Table, View	Collection
Row	Document
Column	Field
Index	Index
Join	Embedded Document
Foreign Key	Reference

Various data types in Mongodb

Type	Number	Alias	Notes
Double	1	"double"	
String	2	"string"	
Object	3	"object"	
Array	4	"array"	
Binary Data	5	"binData"	
Undefined	6	"undefined"	Deprecated
ObjectId	7	"objectId"	
Boolean	8	"bool"	
Date	9	"date"	
Null	10	"null"	
Regular Expression	11	"regex"	
DBPointer	12	"dbPointer"	Deprecated
JavaScript	13	"javascript"	
Symbol	14	"symbol"	Deprecated
Javascript (with scope)	15	"javascriptWithScope"	
32-bit integer	16	"int"	
Timestamp	17	"timestamp"	
64-bit Integer	18	"long"	
Decimal128	19	"decimal	New in Version 3.4
Min Key	-1	"minKey"	
Max Key	127	"maxKey"	

Arrays in Mongodb

- MongoDB Array is **a flexible document structure**; it will make it possible to have a field with array as a value in MongoDB. This is nothing but a simple list of values, and it can take many forms in MongoDB.
- We can define arrays are as follows.
- **{< array field >: {< operator1> : <value1>,**
  < operator2> : <value2>,
  < operator3> : <value3>, ….. }}

Parameters to MongoDB Array

- **1. Array field:** Array field is defined as the field name of the collection on which we create or define array values. The array field is significant while defining any array.
- **2. Operator:** The operator is defined as which values we have to create the array. The operator name is specific to the value name in the array.
- **3. Value:** Value in the array is defined as the actual value of array on which we have defining array in MongoDB. Value is significant while defining an array.

- we have to define an array of emp_skills after defining we have inserted an array of documents in emp_count collection:
- **Code:**
- db.emp_count.find()
- **Code:**
- db.emp_count.insertOne({"emp_name": "ABC", "emp_skills": [["PostgreSQL", "MongoDB", "MySQL", "Perl","ORACLE"]]});

Example

- Below example shows that define a multiple array field in the single collection are as follows:
- **Code:**
- db.emp_countmultiple.insertOne({"emp_name": "ABC", "emp_skills":[["PostgreSQL", "MongoDB", "MySQL", "Perl", "ORACLE"]], "emp_address":[["Pune", "Mumbai", "Delhi"]]});
- we have defined multiple array fields in one collection. We have to define emp_skills and emp_address array in a single collection.

Various Array Operators in MongoDB

- Below are the types of array Operators available in MongoDB.
  - $all
  - $elemMatch
  - $size
  - $
  - $pull
  - $push
  - $pop
- We have taken an example of the stud_test table to describe the various operator's example as follows.
- db.stud_test.find ()

Example

- **1. $all**
- $all array operator is used to display all the value from the array field. Below example show $all array operator:
- **Code:**
- db.stud_test.find ({results: {$all: [88]}})
- **2. $elemMatch**
- $elemMatch array operator is used to match the document which contains the array field and contains only one filed to match our given criteria:
- **Code:**
- db.stud_test.find({results: {$elemMatch: {$gte: 80, $lt: 95}}})

- **3. $size**
- The size array operator in MongoDB will match any array with the number of elements specified by the argument. The below example shows a $size array operator:
- **Code:**
- db.stud_test.find ({“results": {$size: 3}});
- **Output:**

```
>>> db.stud_test.find ({"results": {$size: 3}});
{ "_id" : ObjectId("5e969e3b10f10b41c6e5a8ff"), "stud_id" : 1, "results" : [ 82, 85, 88 ] }
{ "_id" : ObjectId("5e969e5710f10b41c6e5a900"), "stud_id" : 2, "results" : [ 88, 88, 90 ] }
{ "_id" : ObjectId("5e969e7210f10b41c6e5a901"), "stud_id" : 3, "results" : [ 85, 89, 95 ] }
>>>
```

- **4. $**
- The below example shows an $ array operator. $ Array operator is used to identify array element and update the same into the collection:
- **Code:**
- db.stud_test.updateOne ({stud_id: 1, results: 85},{$set: {"results.$" : 95 }})
- **Output:**

```
>>> db.stud_test.updateOne ({stud_id: 1, results: 85},{$set: {"results.$" : 95 }})
{ "acknowledged" : true, "matchedCount" : 1, "modifiedCount" : 1 }
>>>
```

- **5. $pop**
- $pop array operator is used to remove the first and last element from an array. The below example shows a $pop array operator:
- **Code:**
- db.stud_test.updateOne({ stud_id: 1 }, { $pop: { results: -1 } })
- **Output:**

```
>>> db.stud_test.updateOne( { stud_id: 1 }, { $pop: { results: -1 } } )
{ "acknowledged" : true, "matchedCount" : 1, "modifiedCount" : 1 }
>>>
```

db.stud_test.find()

```
>>> db.stud_test.find()
{ "_id" : ObjectId("5e969e3b10f10b41c6e5a8ff"), "stud_id" : 1, "results" : [ 95, 88 ] }
{ "_id" : ObjectId("5e969e5710f10b41c6e5a900"), "stud_id" : 2, "results" : [ 88, 88, 90 ] }
{ "_id" : ObjectId("5e969e7210f10b41c6e5a901"), "stud_id" : 3, "results" : [ 85, 89, 95 ] }
>>>
```

- **6. $pull**
- A pull array operator is used to remove elements from an existing array. The below example shows a $pull array operator are as follows:
- **Code:**
- db.stud_test.update({ stud_id: 1 }, { $pull: { results: { $gte: 95 } } })
- **Output:**

```
>>> db.stud_test.find()
{ "_id" : ObjectId("5e969e3b10f10b41c6e5a8ff"), "stud_id" : 1, "results" : [ 88 ] }
{ "_id" : ObjectId("5e969e5710f10b41c6e5a900"), "stud_id" : 2, "results" : [ 88, 88, 90 ] }
{ "_id" : ObjectId("5e969e7210f10b41c6e5a901"), "stud_id" : 3, "results" : [ 85, 89, 95 ] }
>>>
```

- **7. $push**
- A push array operator is used to append the value into the existing collection. The below example shows the push array operator:
- **Code:**
- db.stud_test.update ({stud_id: 1}, { $push: { results: 101 }})
- **Output:**

```
>>> db.stud_test.find()
{ "_id" : ObjectId("5e969e3b10f10b41c6e5a8ff"), "stud_id" : 1, "results" : [ 88, 101 ] }
{ "_id" : ObjectId("5e969e5710f10b41c6e5a900"), "stud_id" : 2, "results" : [ 88, 88, 90 ] }
{ "_id" : ObjectId("5e969e7210f10b41c6e5a901"), "stud_id" : 3, "results" : [ 85, 89, 95 ] }
>>>
```

Array Example

```
db.students.insertMany([
{
"_id": 1,
"name":"Ram",
"age": 24
"hobbies": ["cricket", "tennis"]
},
{
"_id": 2,
"name":"Rahul",
"age":20,
"hobbies": ["cricket"]
},
{
"_id": 3,
"name":"Riya",
"age":23,
"hobbies": ["FootBall", "tennis"]
}
])
```

Types of Arrays

- Arrays are classified into two types based on their dimensions :
- **single-dimensional and multi-dimensional**.
- Logically, a single-dimensional array represents a linear collection of data, and a two-dimensional array represents a mathematical matrix.
- Similarly, a multidimensional array has multiple dimensions.

Updating Documents

- Once a document is stored in the database, it can be changed using the **`update`** method
- **`update`** takes two parameters: a query document, which locates documents to update, and a modifier document, which describes the changes to make to the documents found.
- Updates are atomic: if two updates happen at the same time, whichever one reaches the server first will be applied, and then the next one will be applied. Thus, conflicting
- updates can safely be sent in rapid-fire succession without any documents being corrupted: the last update will "win."

Updating data in MongoDB

How: `db.collection_name.update(query,update)`

```
db.scores.update({"student":0, "type":"essay"},{ "type":"exam"})})
//Will not really update just the type field. Will Replace the whole
document
```

Set

```
db.scores.update({"student":0, "type":"essay"},{ $set :
{"type":"exam"}})})
//Will update only the type field. If the field doesn't exist will
add it.
```

Incrementing a field that has number value

```
db.scores.update({"student":0, "type":"essay"},{ $inc :
{"score":1}})})
//Will increment the score of the student by 1. If field doesn't exist
Will create the field with value 1
```

Using Modifiers

- Usually only certain portions of a document need to be updated.
- Partial updates can be done extremely efficiently by using atomic *update modifiers*
- Update modifiers are special keys that can be used to specify complex update operations, such as altering, adding, or removing keys, and even manipulating arrays and embedded documents.

Update Operator Modifiers

- Name Description
- $each Modifies the $push and $addToSet operators to append multiple items for array updates.
- $position Modifies the $push operator to specify the position in the array to add elements.
- $slice Modifies the $push operator to limit the size of updated arrays.
- $sort Modifies the $push operator to reorder documents stored in an array.

Removing data in MongoDB

How: `db.collection_name.remove(query)`

Remove all documents

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

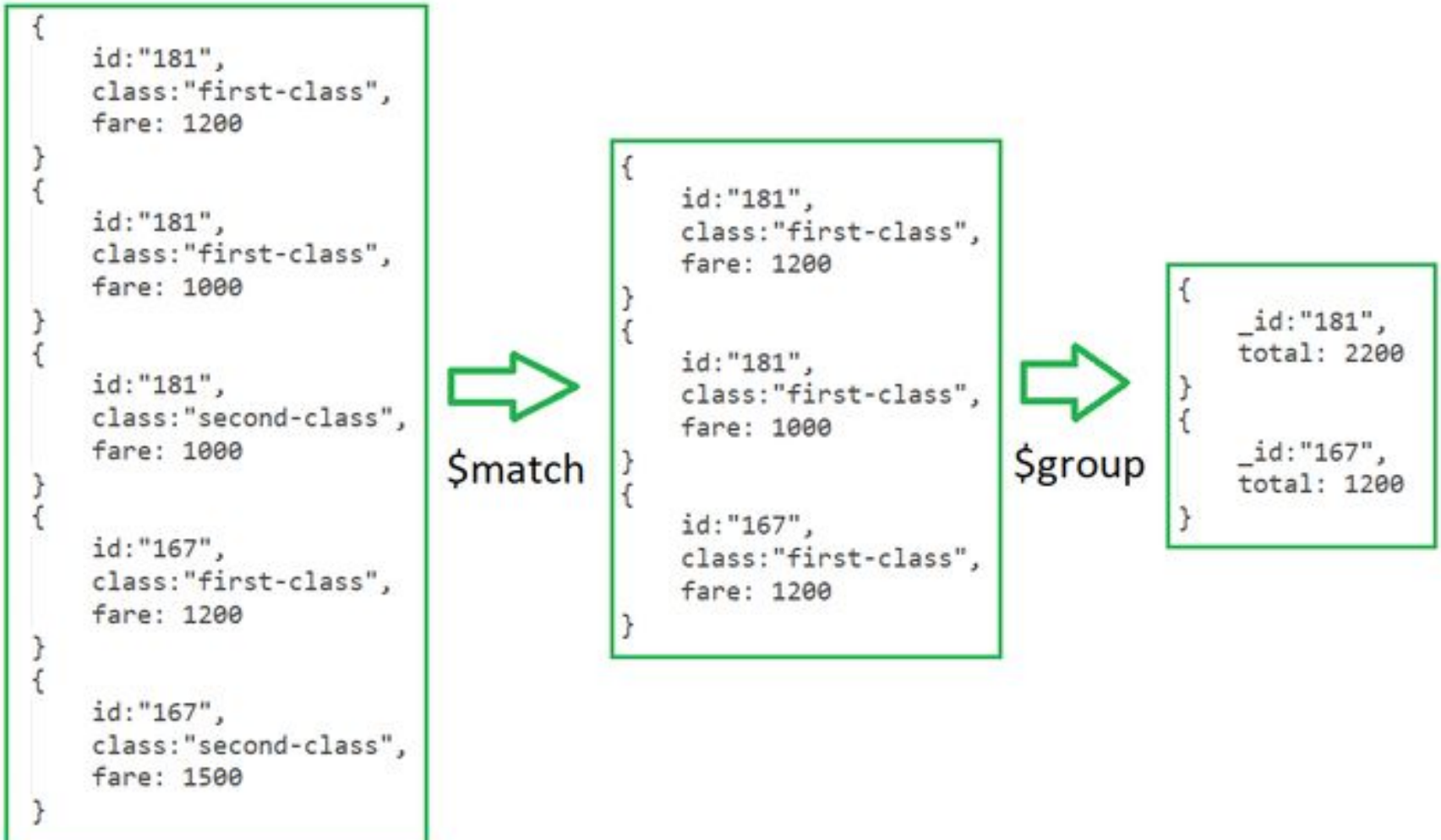
Remove all documents that fulfill a condition

```
db.scores.remove({ "score" : {$gt : 500}})
```

Aggregate Function

```
db.train.aggregate( [
                    {$match:{class:"first-class"}},
                    {$group:{_id:"id",total:{$sum:"$fare"}}}
                   ])
```

} pipeline stages



MapReduce Function

- Using query – match operation select the status ‘A’ – it is same as Aggregate function.
- Next operation is map
- Result

Java script programming

- Javascript is used by programmers across the world **to create dynamic and interactive web content like applications and browsers**.
- JavaScript is so popular that it's the most used programming language in the world, used as a client-side programming language by 97.0% of all websites.
- Example : factorial program

cursors in Mongodb

- In MongoDB, when the find() method is used to find the documents present in the given collection.
- Then this method returned **a pointer which will points to the documents of the collection**, now this pointer is known as cursor.

Concepts: Cursors

- The find() function returns a cursor object

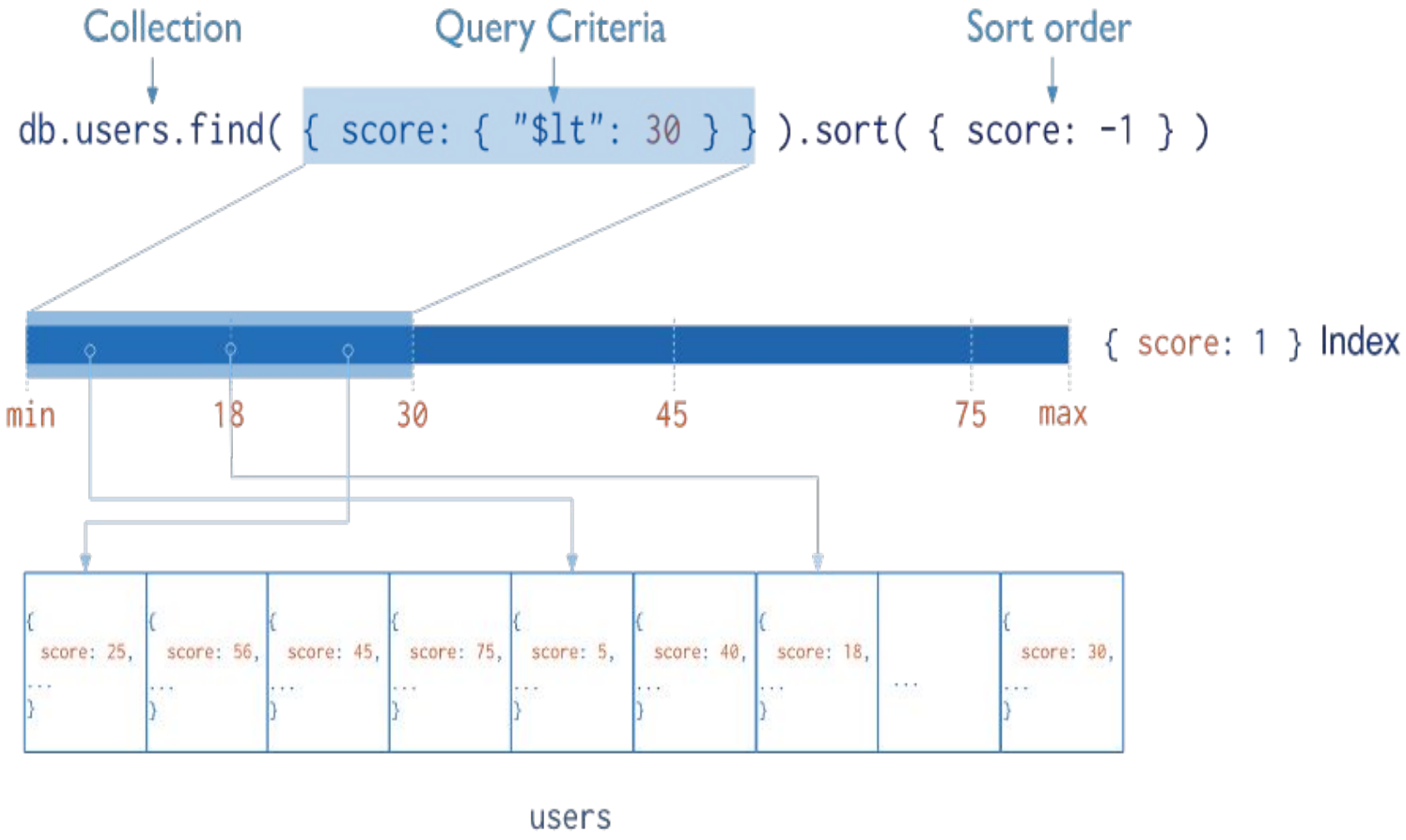
```
var cursor = db.logged_requests.find({ 'status_code' : 200 })

cursor.hasNext() // "true"

cursor.forEach(
  function(item) {
    print(tojson(item))
  }
);

cursor.hasNext() // "false"
```

Indexes

- Indexes are special data structures that store a small portion of the collection's data set in an easy to traverse form.
- The index stores the value of a specific field or set of fields, ordered by the value of the field.
- The ordering of the index entries supports efficient equality matches and range-based query operations.
- In addition, MongoDB can return sorted results by using the ordering in the index.

- The following diagram illustrates a query that selects and orders the matching documents using an index:

- Diagram of a query that uses an index to select and return sorted results. The index stores ``score`` values in ascending order.
- MongoDB can traverse the index in either ascending or descending order to return sorted results.
- Fundamentally, indexes in MongoDB are similar to indexes in other database systems.
- MongoDB defines indexes at the collection level and supports indexes on any field or sub-field of the documents in a MongoDB collection.


Collection
Query Criteria
Sort order
db.users.find({ score: { "$lt": 30 } }).sort({ score: -1 })
{ score: 1 } Index
min
18
30
45
75
max
{ score: 25, ... }
{ score: 56, ... }
{ score: 45, ... }
{ score: 75, ... }
{ score: 5, ... }
{ score: 40, ... }
{ score: 18, ... }
...
{ score: 30, ... }
users

MongoImport and MongoExport in big data

- **MongoImport**
- The mongoimport tool **imports content from an Extended JSON, CSV, or TSV export created by mongoexport , or potentially, another third-party export tool**.
- Run mongoimport from the system command line, not the mongo shell.
- **MongoExport**
- mongoexport is **a command-line tool that produces a JSON or CSV export of data stored in a MongoDB instance**.
- Run mongoexport from the system command line, not the mongo shell.