Full Stack Web Programming

Seven Advanced Academy

MongoDB Part III

Lesson 81

MongoDB Indexing

- Indexes support the efficient resolution of queries. Without indexes, MongoDB must scan every document of a collection to select those documents that match the query statement. This scan is highly inefficient and require MongoDB to process a large volume of data.

- Indexes are special data structures, that store a small portion of the 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 as specified in the index.

MongoDB Indexing

- The ensureIndex() or createIndex() *(since mongodb v5.0)* Method
- To create an index you need to use ensureIndex() method of MongoDB.
- **Syntax**
-
  ```
  >db.COLLECTION_NAME.ensureIndex({KEY:1}, {options})
  ```
- Here key is the name of the field on which you want to create index and 1 is for ascending order. To create index in descending order you need to use -1.

MongoDB Indexing

- **Example**

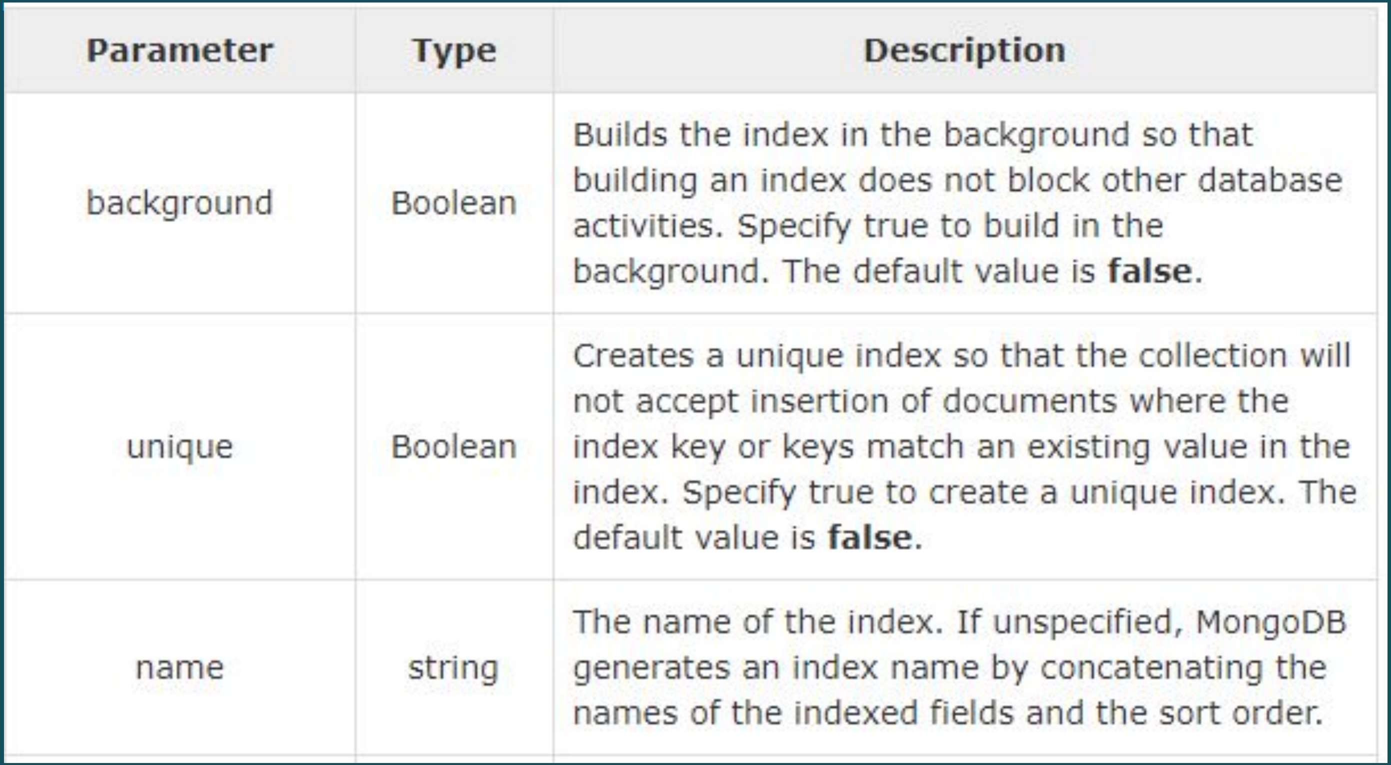
```
>db.mycol.ensureIndex({"title":1})
```

- In ensureIndex() method you can pass multiple fields, to create index on multiple fields.

```
>db.mycol.ensureIndex({"title":1,"description":-1})
```

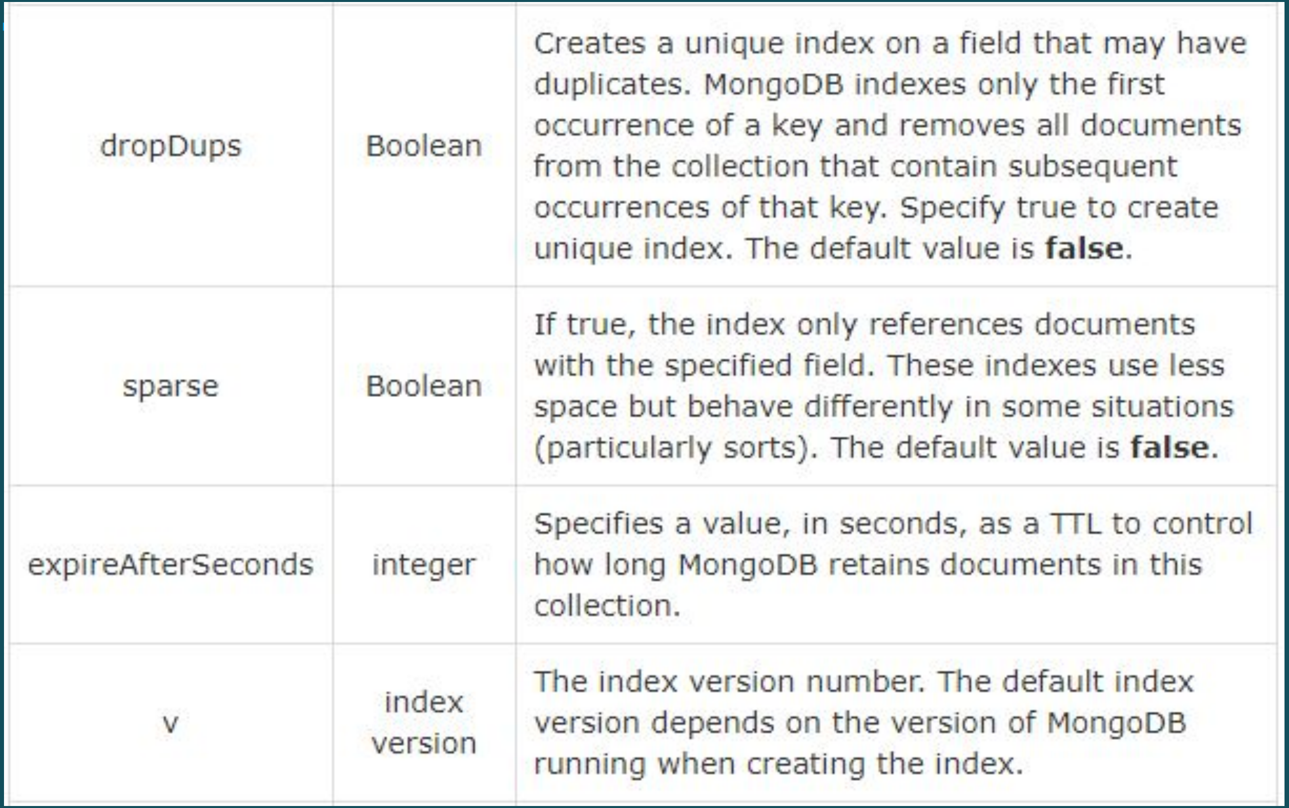
- ensureIndex() method also accepts list of options (which are optional). Following is the list –

MongoDB Indexing



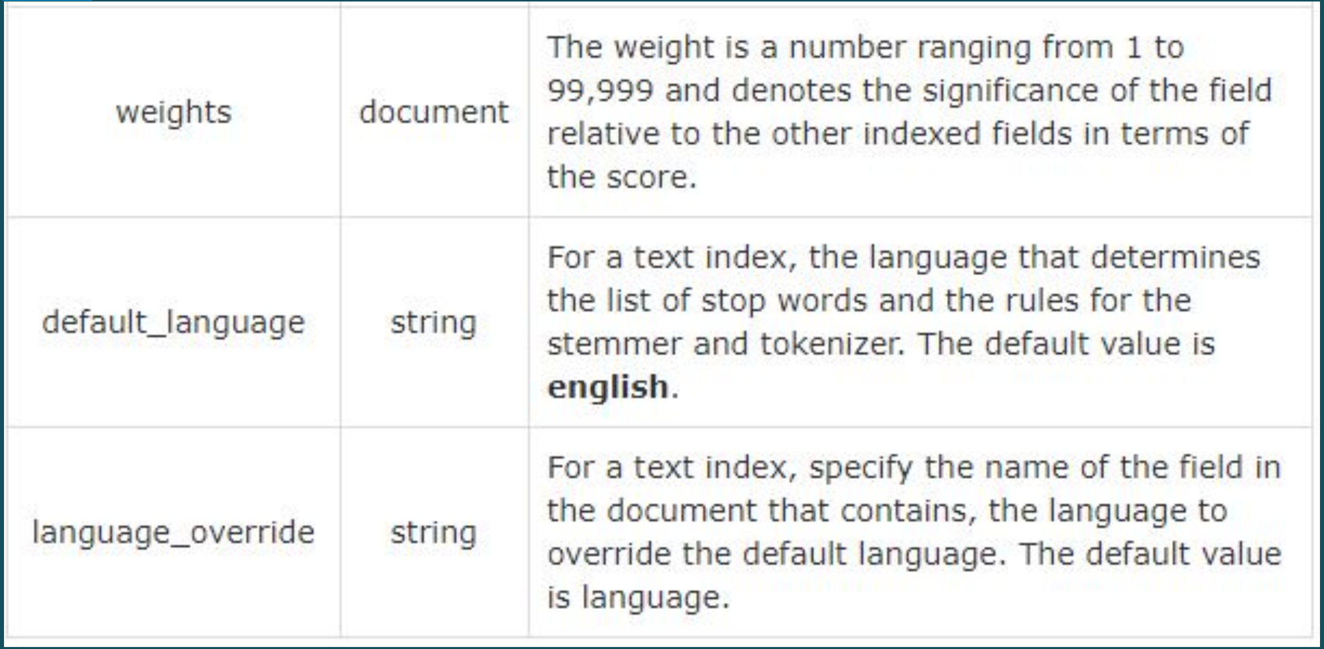
Parameter	Type	Description
background	Boolean	Builds the index in the background so that building an index does not block other database activities. Specify true to build in the background. The default value is **false**.
unique	Boolean	Creates a unique index so that the collection will not accept insertion of documents where the index key or keys match an existing value in the index. Specify true to create a unique index. The default value is **false**.
name	string	The name of the index. If unspecified, MongoDB generates an index name by concatenating the names of the indexed fields and the sort order.

MongoDB Indexing



dropDups	Boolean	Creates a unique index on a field that may have duplicates. MongoDB indexes only the first occurrence of a key and removes all documents from the collection that contain subsequent occurrences of that key. Specify true to create unique index. The default value is **false**.
sparse	Boolean	If true, the index only references documents with the specified field. These indexes use less space but behave differently in some situations (particularly sorts). The default value is **false**.
expireAfterSeconds	integer	Specifies a value, in seconds, as a TTL to control how long MongoDB retains documents in this collection.
v	index version	The index version number. The default index version depends on the version of MongoDB running when creating the index.

MongoDB Indexing



weights	document	The weight is a number ranging from 1 to 99,999 and denotes the significance of the field relative to the other indexed fields in terms of the score.
default_language	string	For a text index, the language that determines the list of stop words and the rules for the stemmer and tokenizer. The default value is **english**.
language_override	string	For a text index, specify the name of the field in the document that contains, the language to override the default language. The default value is language.

MongoDB Aggregation

- Aggregations operations process data records and return computed results. Aggregation operations group values from multiple documents together, and can perform a variety of operations on the grouped data to return a single result. In SQL count(*) and with group by is an equivalent of mongodb aggregation.

- The aggregate() Method
- For the aggregation in MongoDB, you should use aggregate() method.

MongoDB Aggregation

- **Syntax**
- Basic syntax of aggregate() method is as follows –

```
>db.COLLECTION_NAME.aggregate(AGGREGATE_OPERATION)
```

- **Example**
- In the collection you have the following data –

```
{
   _id: ObjectId(7df78ad8902c)
   title: 'MongoDB Overview',
   description: 'MongoDB is no sql database',
   by_user: 'Seven Academy',
   url: 'http://www.sevenadvancedacademy.com',
   tags: ['mongodb', 'database', 'NoSQL'],
   likes: 100
},
{
   _id: ObjectId(7df78ad8902d)
   title: 'NoSQL Overview',
   description: 'No sql database is very fast',
   by_user: 'Seven Academy',
   url: 'http://www.sevenadvancedacademy.com',
   tags: ['mongodb', 'database', 'NoSQL'],
   likes: 10
},
{
   _id: ObjectId(7df78ad8902e)
   title: 'Neo4j Overview',
   description: 'Neo4j is no sql database',
   by_user: 'Neo4j',
   url: 'http://www.neo4j.com',
   tags: ['neo4j', 'database', 'NoSQL'],
   likes: 750
},
```

MongoDB Aggregation

- Now from the above collection, if you want to display a list stating how many tutorials are written by each user, then you will use the following aggregate() method –

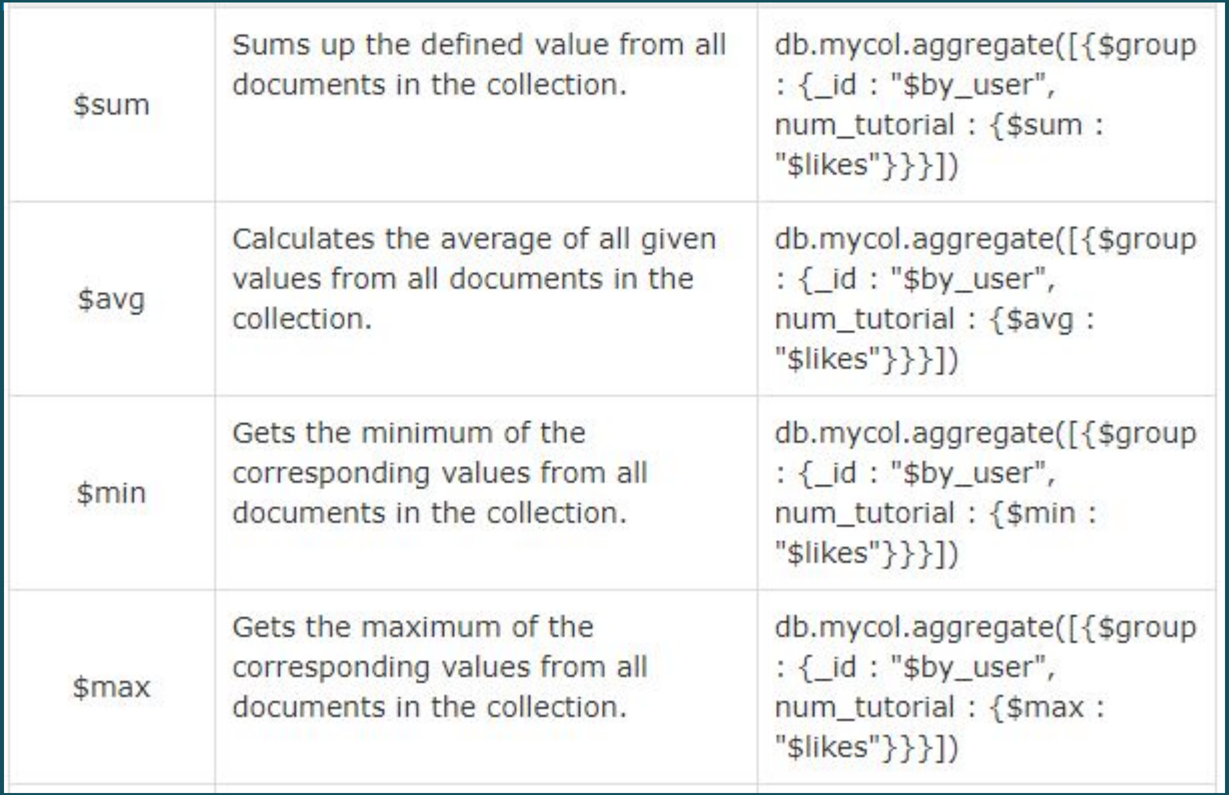
```
> db.mycol.aggregate([{$group : {_id : "$by_user", num_tutorial : {$sum : 1}}}])
{
   "result" : [
      {
         "_id" : "Seven Academy",
         "num_tutorial" : 2
      },
      {
         "_id" : "Neo4j",
         "num_tutorial" : 1
      }
   ],
   "ok" : 1
}
```

MongoDB Aggregation

- **Sql** equivalent query for the above use case will be: **select by_user, count(*) from mycol group by by_user.**

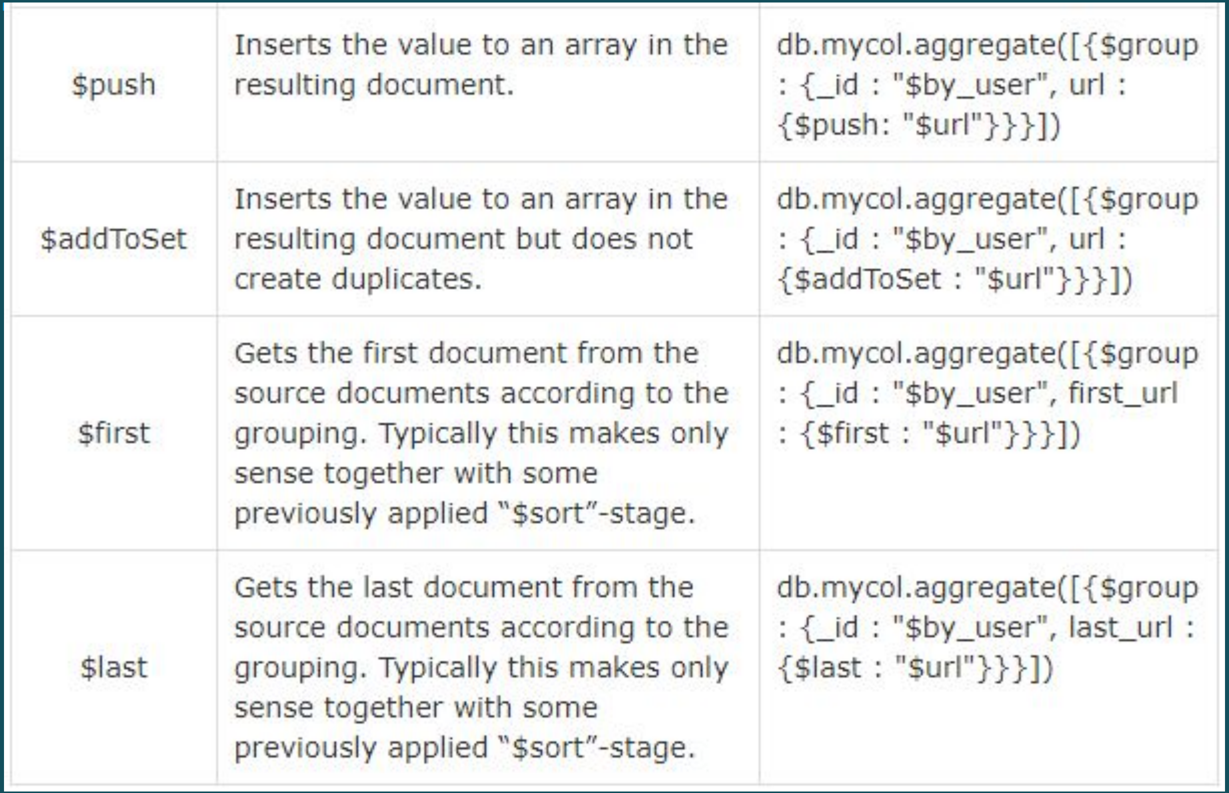
- In the above example, we have grouped documents by field by_user and on each occurrence of by_user previous value of sum is incremented. Following is a list of available aggregation expressions.

MongoDB Aggregation



$sum	Sums up the defined value from all documents in the collection.	db.mycol.aggregate([{$group : {_id : "$by_user", num_tutorial : {$sum : "$likes"}}}])
$avg	Calculates the average of all given values from all documents in the collection.	db.mycol.aggregate([{$group : {_id : "$by_user", num_tutorial : {$avg : "$likes"}}}])
$min	Gets the minimum of the corresponding values from all documents in the collection.	db.mycol.aggregate([{$group : {_id : "$by_user", num_tutorial : {$min : "$likes"}}}])
$max	Gets the maximum of the corresponding values from all documents in the collection.	db.mycol.aggregate([{$group : {_id : "$by_user", num_tutorial : {$max : "$likes"}}}])

MongoDB Aggregation



$push	Inserts the value to an array in the resulting document.	db.mycol.aggregate([{$group : {_id : "$by_user", url : {$push: "$url"}}}])
$addToSet	Inserts the value to an array in the resulting document but does not create duplicates.	db.mycol.aggregate([{$group : {_id : "$by_user", url : {$addToSet : "$url"}}}])
$first	Gets the first document from the source documents according to the grouping. Typically this makes only sense together with some previously applied "$sort"-stage.	db.mycol.aggregate([{$group : {_id : "$by_user", first_url : {$first : "$url"}}}])
$last	Gets the last document from the source documents according to the grouping. Typically this makes only sense together with some previously applied "$sort"-stage.	db.mycol.aggregate([{$group : {_id : "$by_user", last_url : {$last : "$url"}}}])

MongoDB Aggregation

- **Pipeline Concept**
- In UNIX command, shell pipeline means the possibility to execute an operation on some input and use the output as the input for the next command and so on. MongoDB also supports same concept in aggregation framework. There is a set of possible stages and each of those is taken as a set of documents as an input and produces a resulting set of documents (or the final resulting JSON document at the end of the pipeline). This can then in turn be used for the next stage and so on.

MongoDB Aggregation

- Following are the possible stages in aggregation framework –
- **$project** – Used to select some specific fields from a collection.
- **$match** – This is a filtering operation and thus this can reduce the amount of documents that are given as input to the next stage.
- **$group** – This does the actual aggregation as discussed above.

MongoDB Aggregation

- **$sort** – Sorts the documents.
- **$skip** – With this, it is possible to skip forward in the list of documents for a given amount of documents.
- **$limit** – This limits the amount of documents to look at, by the given number starting from the current positions.
- **$unwind** – This is used to unwind document that are using arrays. When using an array, the data is kind of pre-joined and this operation will be undone with this to have individual documents again. Thus with this stage we will increase the amount of documents for the next stage.

MongoDB Replication

- Replication is the process of synchronizing data across multiple servers. Replication provides redundancy and increases data availability with multiple copies of data on different database servers. Replication protects a database from the loss of a single server. Replication also allows you to recover from hardware failure and service interruptions. With additional copies of the data, you can dedicate one to disaster recovery, reporting, or backup.

Why Replication ?

- To keep your data safe
- High (24*7) availability of data
- Disaster recovery
- No downtime for maintenance (like backups, index rebuilds, compaction)
- Read scaling (extra copies to read from)
- Replica set is transparent to the application

How Replication Works in MongoDB ?

- MongoDB achieves replication by the use of replica set. A replica set is a group of mongod instances that host the same data set. In a replica, one node is primary node that receives all write operations. All other instances, such as secondaries, apply operations from the primary so that they have the same data set. Replica set can have only one primary node.

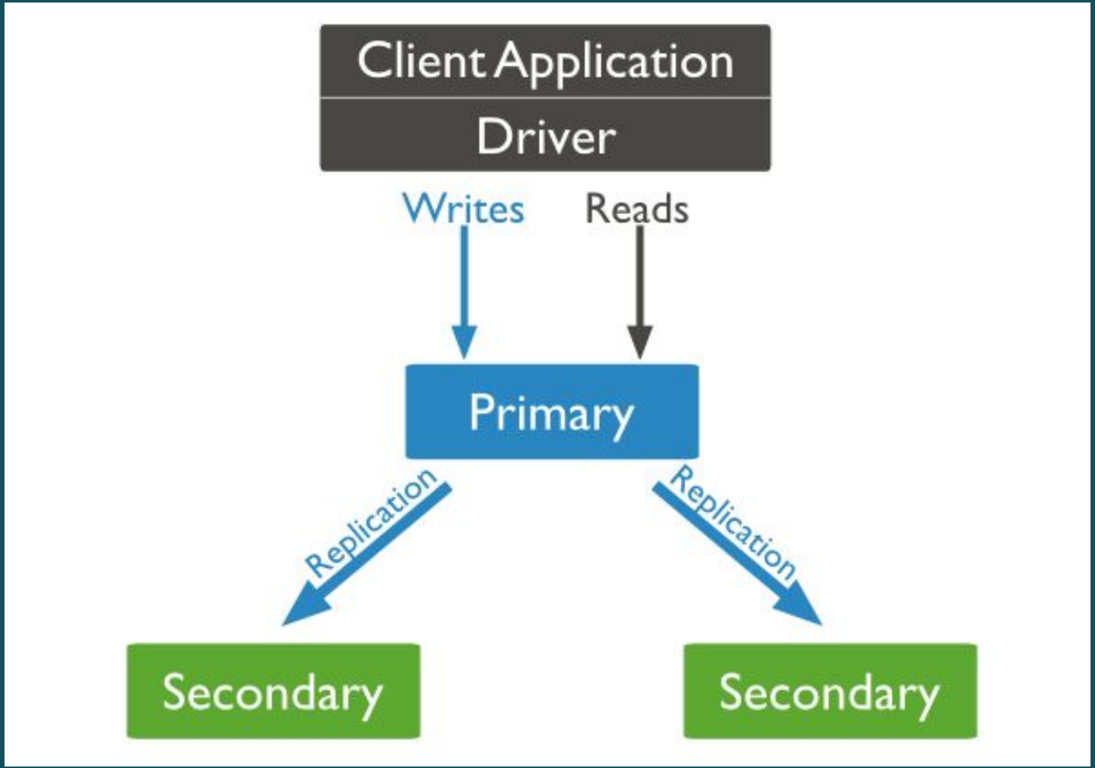
How Replication Works in MongoDB ?

- Replica set is a group of two or more nodes (generally minimum 3 nodes are required).
- In a replica set, one node is primary node and remaining nodes are secondary.
- All data replicates from primary to secondary node.
- At the time of automatic failover or maintenance, election establishes for primary and a new primary node is elected.
- After the recovery of failed node, it again join the replica set and works as a secondary node.

How Replication Works in MongoDB ?

- A typical diagram of MongoDB replication is shown in which client application always interact with the primary node and the primary node then replicates the data to the secondary nodes.

How Replication Works in MongoDB ?



Replicat Set Features

- A cluster of N nodes
- Any one node can be primary
- All write operations go to primary
- Automatic failover
- Automatic recovery
- Consensus election of primary

Setup a Replicat Set

- In this tutorial, we will convert standalone MongoDB instance to a replica set. To convert to replica set, the following are the steps –
- Shutdown already running MongoDB server.
- Start the MongoDB server by specifying -- replSet option.
- The following is the basic syntax of --replSet –

```
mongod --port "PORT" --dbpath "YOUR_DB_DATA_PATH" --replSet "REPLICA_SET_INSTANCE_NAME"
```

Setup a Replicat Set

- **Example**

```
mongod --port 27017 --dbpath "D:\set up\mongodb\data" --replSet rs0
```

- It will start a mongod instance with the name rs0, on port 27017.
- Now start the command prompt and connect to this mongod instance.
- In Mongo client, issue the command rs.initiate() to initiate a new replica set.
- To check the replica set configuration, issue the command rs.conf(). To check the status of replica set issue the command rs.status().

Add Members to Replica Set

- To add members to replica set, start mongod instances on multiple machines. Now start a mongo client and issue a command rs.add().
- **Syntax**
- The basic syntax of rs.add() command is as follows –

```
>rs.add(HOST_NAME:PORT)
```

- **Example**
- Suppose your mongod instance name is mongod1.net and it is running on port 27017. To add this instance to replica set, issue the command rs.add() in Mongo client.

Add Members to Replica Set

```
>rs.add("mongod1.net:27017")
```

- You can add mongod instance to replica set only when you are connected to primary node. To check whether you are connected to primary or not, issue the command db.isMaster() in mongo client.

Congratulation!