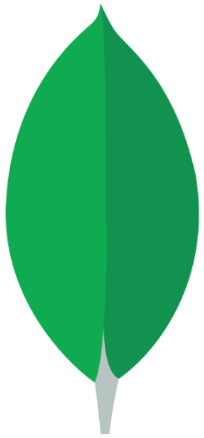


mongoDB®

Mrihan Mohamed Ahmed

Teaching Assistant – ITI

mrihan.m.ahmed@gmail.com

01094984105

FB: Mrihan Mohamed

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.
- Indexes are special data structures, that store a small portion of the data set in an easy-to-traverse form.
- To create an index you need to use createIndex() method of MongoDB.

```
db.Authors.createIndex({Name:1})
```

Indexing

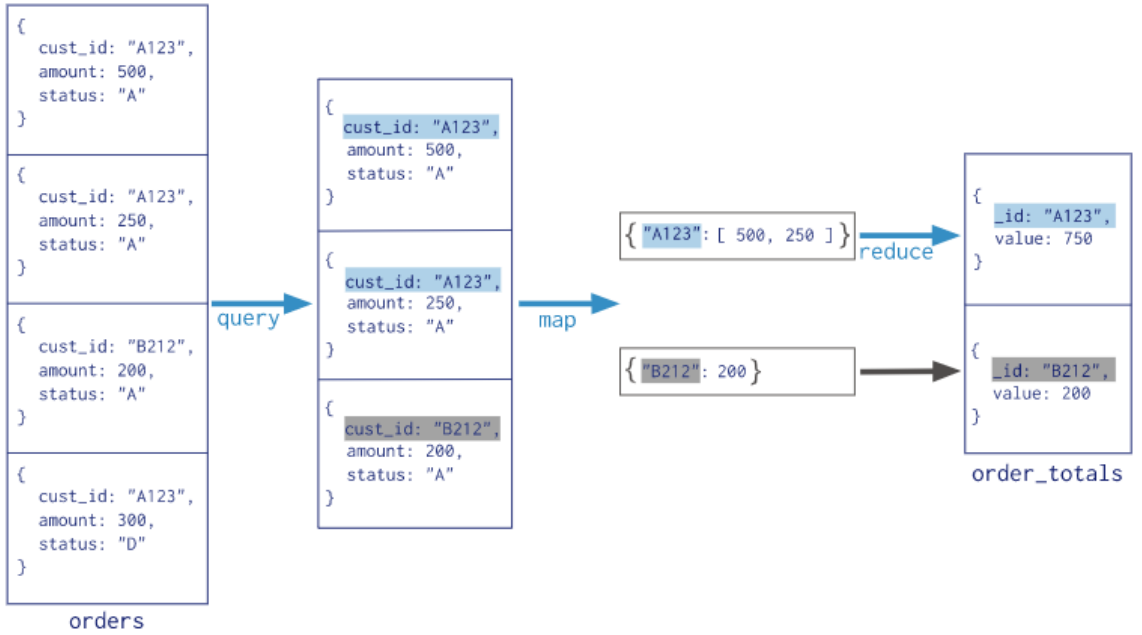
Parameter	Type	Description
background	Boolean	Builds the index in the background so that building an index does not block other database activities. Specify true to build in the background. The default value is **false**.
unique	Boolean	Creates a unique index so that the collection will not accept insertion 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.
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.

Demo

MapReduce

- **Map-reduce** is a data processing paradigm for condensing large volumes of data into useful aggregated results. MongoDB provides the mapReduce database command.

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



Demo

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

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

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"}}}])

Aggregation

$first	Gets the first document from the source documents according to the grouping. Typically this makes only sense together with some previously applied “$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"}}}])

Aggregation

- **$project** – Used to select some specific fields from a collection.
- **$group** – This does the actual aggregation as discussed above.
- **$sort** – Sorts the documents.
- **$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.

Demo

Relationships

- Relationships in MongoDB represent how various documents are logically related to each other.
- There are 3 ways to make relations in MongoDB:
  - Embedded Documents.
  - ObjectId() Reference.
  - DBRef.
- Such relationships can be either 1:1, 1:N, N:1 or N:N.

Relationships (Embedded Docs)

```
{
   "_id":ObjectId("52ffc33cd85242f436000001"),
   "contact": "987654321",
   "dob": "01-01-1991",
   "name": "Tom Benzamin",
   "address": [
      {
         "building": "170 A, Acropolis Apt",
         "pincode": 456789,
         "city": "Chicago",
         "state": "Illinois"
      }
   ]
}
```

```
>db.users.findOne({"name":"Tom Benzamin"},{"address":1})
```

Relationships (ObjectId() Reference)

```
{
   "_id":ObjectId("52ffc33cd85242f436000001"),
   "contact": "987654321",
   "dob": "01-01-1991",
   "name": "Tom Benzamin",
   "address_ids": [
      ObjectId("52ffc4a5d85242602e000000"),
      ObjectId("52ffc4a5d85242602e000001")
   ]
}
```

```
>var result = db.users.findOne({"name":"Tom Benzamin"},{"address_ids":1})
>var addresses = db.address.find({"_id":{"$in":result["address_ids"]}})
```

Relationships (DBRef)

```
{
   "_id":ObjectId("53402597d852426020000002"),
   "address": {
   "$ref": "address_home",
   "$id": ObjectId("534009e4d852427820000002"),
   "$db": "tutorialspoint"},
   "contact": "987654321",
   "dob": "01-01-1991",
   "name": "Tom Benzamin"
}
```

```
>var user = db.users.findOne({"name":"Tom Benzamin"})
>var dbRef = user.address
>db[dbRef.$ref].findOne({"_id":(dbRef.$id)})
```

Demo

Lookup

- Performs a left outer join to an unsharded collection in the same database to filter in documents from the "joined" collection for processing.

```
{
   $lookup:
     {
       from: <collection to join>,
       localField: <field from the input documents>,
       foreignField: <field from the documents of the "from" collection>,
       as: <output array field>
     }
}
```

Atlas

- MongoDB Atlas is the global cloud database service for modern applications. Used for deploying fully managed MongoDB across AWS, Google Cloud, and Azure.
- handles all the complexity of deploying, managing, and healing your deployments on the cloud service provider.
- Atlas comes:
  - MongoDB Atlas Search.
  - MongoDB Atlas Data Lake.
  - Realm Database.

Cluster

- Clusters are Atlas-managed MongoDB deployments.
- A cluster can be either a replica set or a sharded cluster.
- The choice of cloud provider and region affects the configuration options for the available cluster tiers, network latency for clients accessing your cluster, and the cost of running the cluster.
- The region refers to the physical location of your MongoDB cluster.
- The cluster tier dictates the memory, storage, and IOPS specification

Compass

- As the GUI for MongoDB, MongoDB Compass allows you to make decisions about document structure, querying, indexing, document validation, and more.
- Download Compass form here:
  - https://www.mongodb.com/try/download/compass
- Click on connect in your cluster and follow the steps.
- Now you can preform the CRUD operation on your server.

Thanks