Aggregation stages

Count,group,limit,lookup,match,merge,sort,project,unwind,unset

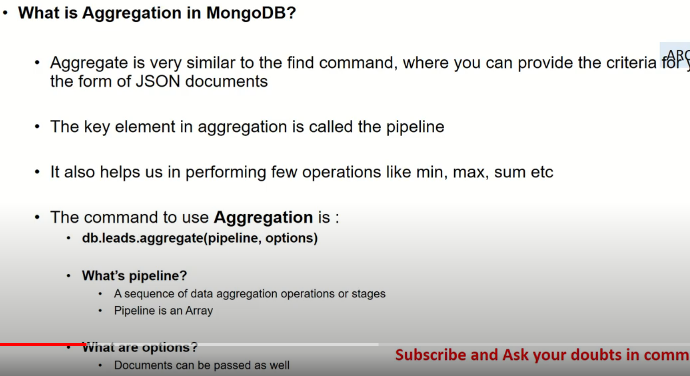
When to use this aggregation? -

If u have sequence of operations, then use aggregations

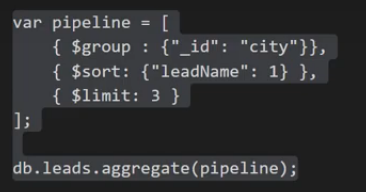
Like :1st if u want to filter ,then sort and limit only for 10 records

Means here u are using sequence of operations then aggregations is the best choice

**Pipeline is a sequence of operations**



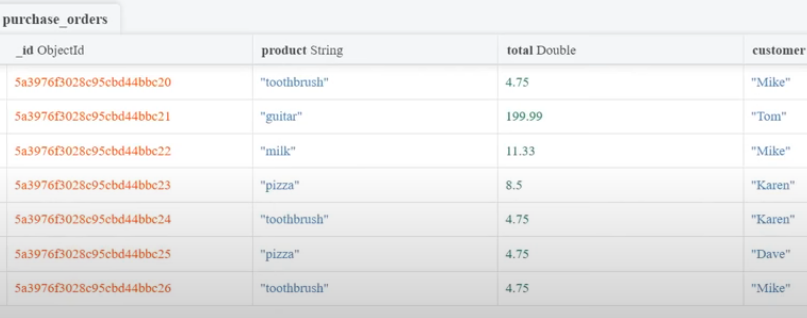
Examples



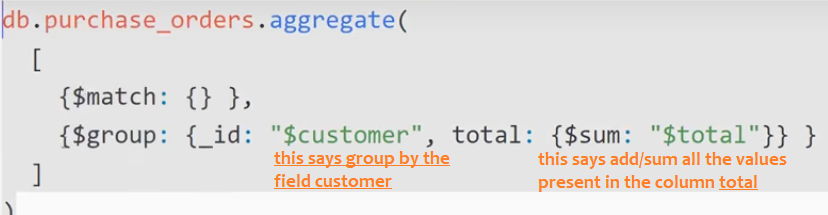
See here ur are 3 operations, so instead of using find by operations better use aggregations

## Sample aggregations

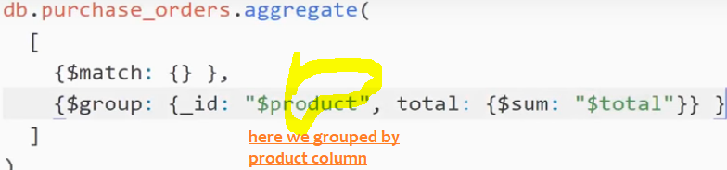
Here we want to group by customer and want to find out, how much he spent



Group by column customer



Group by column product



### Group by sort, filter , sum



### Aggregations examples

{$match:{ Sport:{$in:[“ Swimming”]}}}

The below example says group by category and count the number of rows and sort by count

db.applications.aggregate([

{"$group" : {\_id:"$Category", count:{$sum:1}}},

{$sort:{count:-1}}

])

#### Group by job title and count

db.hdfstomongo.aggregate([

{"$group" : {\_id:"$JOB\_TITLE", count:{$sum:1}}},

{$sort:{count:-1}}

])

BELOW Are the results

{ "\_id" : "PROGRAMMER ANALYST", "count" : 94 }

{ "\_id" : "SOFTWARE ENGINEER", "count" : 32 }

{ "\_id" : "SOFTWARE DEVELOPER", "count" : 29 }

{ "\_id" : "SYSTEMS ANALYST", "count" : 20 }

{ "\_id" : "SENIOR SOFTWARE ENGINEER", "count" : 13 }

db.applications.aggregate([

{"$group" : {\_id:"$Type", count:{$sum:1}}},

{$sort:{count:-1}}

])

#### Sample -4

\* Which Employer has the maximum number of PROGRAMMER ANALYST job roles

PROGRAMMER ANALYST

db.hdfstomongo.aggregate([

{$match:{ JOB\_TITLE:{$in:["PROGRAMMER ANALYST"]}}},

{"$group" : {\_id:"$EMPLOYER\_NAME", count:{$sum:1}}},

{$sort:{count:-1}}

])

result

{ "\_id" : "CITADEL INFORMATION SERVICES INC.", "count" : 7 }

{ "\_id" : "MYTHRI CONSULTING LLC", "count" : 6 }

{ "\_id" : "SYNTEL CONSULTING INC.", "count" : 5 }

Sample -2

\* Query top 10 employers who files more number of cases along with the count of number of cases

Group by company whereas 1 record for 1 case

db.hdfstomongo.aggregate([

{"$group" : {\_id:"$EMPLOYER\_NAME", count:{$sum:1}}},

{$sort:{count:-1}}

])

{ "\_id" : "ERNST & YOUNG U.S. LLP", "count" : 20 }

{ "\_id" : "ACCENTURE LLP", "count" : 19 }

{ "\_id" : "DELOITTE CONSULTING LLP", "count" : 18 }

{ "\_id" : "HCL AMERICA, INC.", "count" : 16 }

{ "\_id" : "INFOSYS LIMITED", "count" : 16 }

{ "\_id" : "TATA CONSULTANCY SERVICES LIMITED", "count" : 16 }

{ "\_id" : "MICROSOFT CORPORATION", "count" : 13 }

{ "\_id" : "TECHTOMIX LLC", "count" : 13 }

Indexing

**Index Introduction**

An **index** is required for faster retrieval of data. Indexes in MongoDB are sorted and stored as B-tree structure. There should be a balance between Indexes and queries.

Following are the index types supported in MongoDB.

* Default\_id: Each collection **contains an index named default\_id**
* **Single Field: Indexes can be either in ascending order or descending order**
* **Compound Index: used for multiple fields**
* **Text indexes: To support text search queries on string content**
* **Multikey Index: These are used to index array data**
* **Geospatial Index: Indexes used are of 2d and 2d spheres**

What happens Without index

When query in MongoDB is not indexed, a **full collection** scan will be performed. The absence of index can cause significant database performance degradation..

Documents inspected in memory should be reduced. The need to perform in-memory sorts must be removed.

Following are some of the factors to be considered for an index selection include:

What data is written to the database

What kind of data is read-only

What piece of data are used together

Rich Documents

How to create index

We can use **ensureIndex()** or **createIndex()** method to create an index in MongoDB.

**Syntax:**

db.collection.createIndex({KEY:1})

db.Player.insert(

{

"\_id": "1",

"score": 10340,

"location": { state: "NSW", city: "Sydney" }

})

**Create Index on score**

db.Player.createIndex( { score: 1 } )

* **1** for index indicates scores are sorted in ascending order
* **-1** for index indicates scores are sorted in descending orde

Since version 3.0 ensureIndex() method is deprecated.

**Example :**

db.file.createIndex({tags: 1});

**Drop Index:**

we can use **dropIndex()** command to remove or drop index from collection. Default index on the \_id field cannot be removed with dropIndexes() command.

**Example**

db.file.dropIndex({tags: 1});

**Default index**

During the creation of a collection,

* MongoDB will create a **unique index** on the ***\_id***.
* The \_id index will restrict clients from inserting two documents with the same values (duplicates) for the \_id field.

Compound Indexes

**Compound indexes:**

* On compound indexes, there will be **single index structure** that holds references to multiple fields.
* MongoDB has to limit restriction of **31 fields** for any compound index.

db.collectionname.createIndex( { <field1>: <type>, <field2>: <type2>, ... } )

db.school.createIndex({ subject: 1 ,score: 1})

db.school.createIndex({ score: 1,subject: 1 })

**Performance will vary based on the *order of fields* mentioned on Compound index**

Multi key indexes

In this index will be applied on array field

Multikey indexes are used to make efficient queries against array fields. This can be created over arrays which hold both scalar values and nested documents.

Suppose we have **Employees** collection that contains details of employees with multiple skills

try {

db.Employees.insertMany( [

{ "\_id" :"1", "Name" :"Mridhula", "EmployeeCode" : "EC01", "Country" : "IND" ,"Skills": ["java", "oracle", "Informatica"]},

{ "\_id" :"2", "Name" : "Akhila", "EmployeeCode" : "EC02", "Country" : "US","Skills": ["java", "oracle", "Informatica"]},

We can create a MultiKey Index of Skills as below.

db.Employees.createIndex({"Skills":1});

sharding

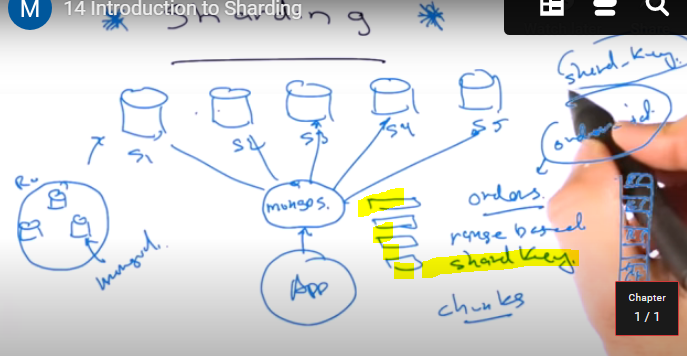
like instead of storing all aadhar card data in 1 table , it says split and store the data in diff servers in same table name

**Sharding** can be termed as the processes for distributing data across various servers for storage.

A key from the collection will be identified as shard key and splits data using that specific key.

Factors to be considered for selecting Shard Key:

* Good Cardinality/Granularity which means selecting key should have enough values to spread the collections.
* Common data in queries for the collection.
* Based on the schema of data.
* Based on Database applications query and perform write operations to be performed.
* **Shards** − Shards are useful for storing data. This will ensure **high availability and data consistency**.
* Config Servers − This will be storing cluster's metadata. Data consist of mapping between cluster's data set with shards. This metadata will be taken by query router to target operations to specific shards.
* Query Routers − Those are mongo instances, which interfaces between direct operations to the appropriate shard client applications.
* Router processes and combines the operations to shards and then returns results to the clients.

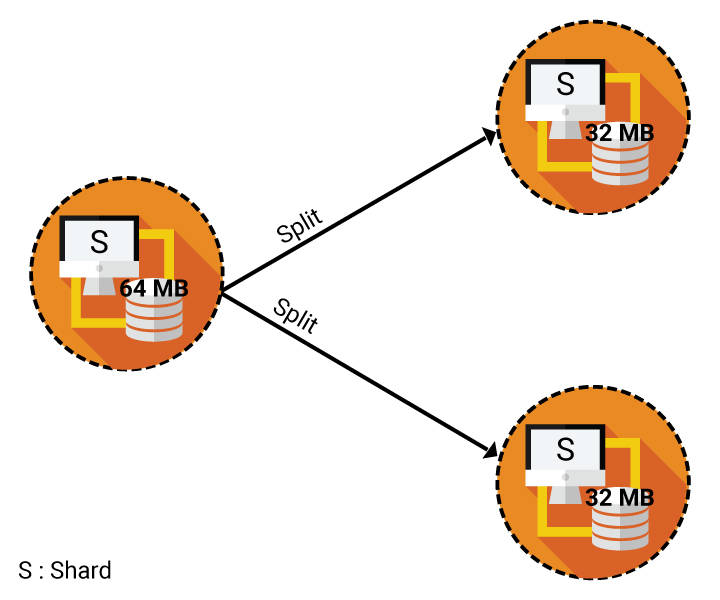


Mongo knows in which shard the data is 🡪 shard is nothing but a piece of data- in telugu it’s a mukka

In MongoDB, you have to know that data distribution is based on ShardKey.

* While storing itself , it will store based on the shard key – like all 1 crore records will not be stored in a single collection- based on shard key it will be split and stored across multiple servers
* Shard key values will be divided into chunk that are evenly distributed across the shard.
* MongoDB divides shard key values by
  + Range based Partitioning
  + Hash based Partitioning
* Consider the below example with a collection "User" that contains name {.....name...}. Here ***name*** is the sharding key.
* **Metadata** looks like below.
* Shard nameLow - nameHigh
* Shard 0 (s0) : range [jane - jose]
* Shard 1 (s1) : range [joe - kyle ]
* Shard 3 (s2) : range [kyle - matt]
* Shard 4 (s3) : range [Robert- Zzzzz]
* when the user runs the below command
* db.users.find({name:/^jo/})

**in that case, the search will go to only two shards s0 and s1.**



**Example**: This picture shows how split works. Consider when a chunk grows beyond 64 MB limit, it will be split into **two 32 MB** chunks.

Balancer acts as a background process that helps to manage chunk migrations.

* This can be performed from any of the mongo instances in a cluster.
* When the distribution of sharded collection in a cluster became uneven, balancer migrates chunks from the largest number of chunks to least number of chunks still the collection balances.

Replication

So never think mongo will go down

In MongoDB Replication helps to synchronize data across multiple servers.

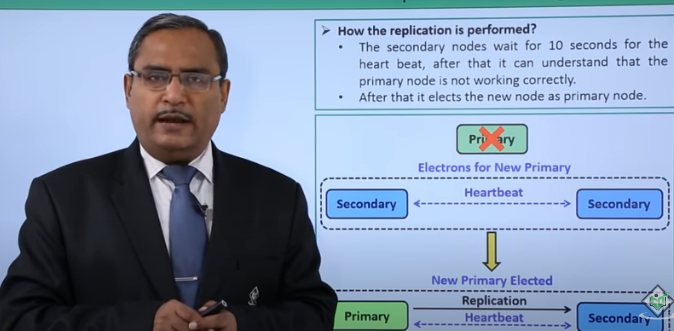
* Replication is achieved by placing multiple copies of data on different database servers.
* Replication assists by
  + **providing Redundancy**
  + **increased Data availability**

**Benefits**

* Protects the database from the **loss of a single server failure**.
* Helps to keep data safe with higher availability of data.
* Downtime is not required for maintenance and Disaster Recovery (like backups, index rebuilds, compaction).

##### How Replication Works?

In MongoDB replication,

* Insertion will occur in the primary node.
* An operation will get tracked in oplog (which is part of local db – **oplog.$main**)
* Secondary nodes will read data from oplog and update the respective node.
* A group of **mongod** instances that host the same data set can be termed as a **replica set**.
* On replica set, one node will act as a primary node, and remaining nodes will become secondary nodes.
* ItS RESPONSIBILITY OF primary node to send the heart beat signal to all the secondary nodes
* 

##### Replica Set Data Synchronization

MongoDB supports two forms of data synchronization:

Initial sync to copy data to new members with the full data set, and replication for ongoing changes to the entire data set.

Initial Sync:

* To copies data to new members with the full data set, MongoDB clones all databases except the local database.
* To perform Clones mongod will scan every collection in each source database and inserts all data into its copies of these collections.

Fault Tolerance:

* Recover when there are network or operation failures.

##### **Set Up a Replica Set**

Following is the basic syntax for replica set.

mongod --port "portname" --dbpath "

Databasedatapath" --replSet

"Replicasetname"

&nbsp;

**Example:**

mongod --port 27018 --dbpath "C:\mongodb\data1" --replSet rs0

* Mongo client will generate **rs.initiate()** command to initiate a new replica set.
* **rs.conf()** can be used to check the replica set configuration.
* **rs.status()** can be used to check the status of the replica set.
* **db.isMaster()** can be used to check connected node is primary or not.

creating user

he db.createUser() method is used to create a new user.

**Example:**

**Create user with roles**

use sample

db.createUser(

{

user: "usertest",

pwd: "usertest123",

roles: [ "readWrite", "dbAdmin" ]

}

);

db.getUsers();

db.getUser("mynewuser");

db.dropUser()

db.dropUser("User1", {w: "majority",

wtimeout: 2000})

db.grantRolesToUser( "<username>", [ <roles> ], { <writeConcern> } )

db.grantRolesToUser(

"Usr01",

[ "readWrite" ,{ role: "read",

db: "Film" } ],

{ w: "majority" , wtimeout: 2000 }

)

db.revokeRolesFromUser() removes one or more roles from a user on the current

db.revokeRolesFromUser( "<username>", [ <roles> ], { <writeConcern> } )

db.revokeRolesFromUser( "accountUser01",

[ { role: "read", db: "film"

},

"readWrite"

],

{ w: "majority" }

)

Connecting form java

MongoClient mongoClient = new MongoClient();

MongoClient mongoClient = new MongoClient("abcd.server.com", 27017);

List<ServerAddress> servername = new ArrayList<ServerAddress>();

servername.add(new ServerAddress("ab.server.com", 27017));

MongoClient mongoClient = new MongoClient(servername);