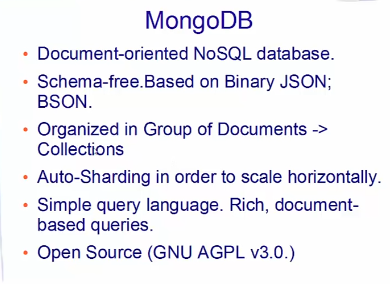
Cd C:\Program Files\MongoDB\Server\3.4\bin

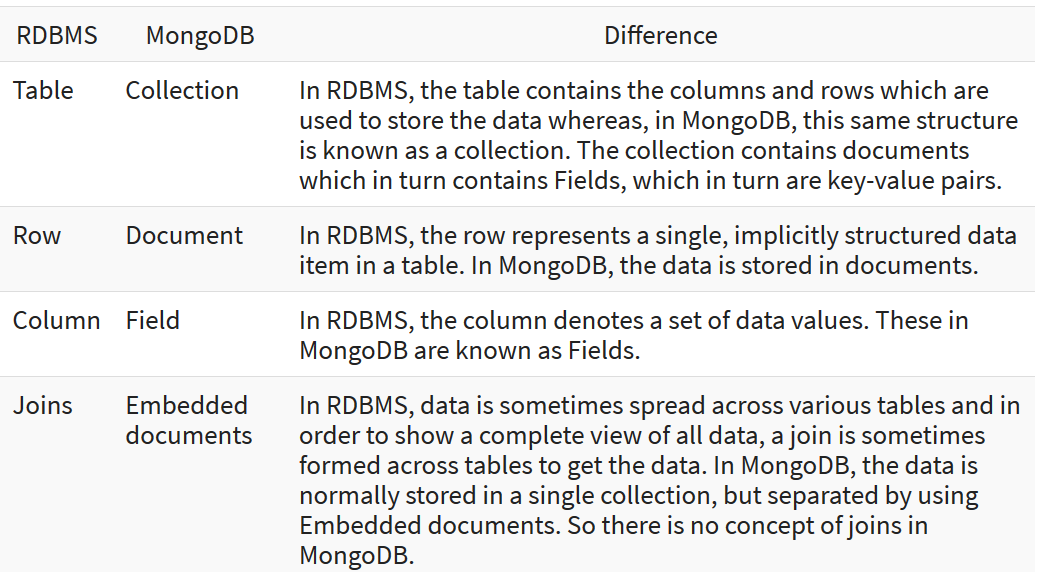
mongod --dbpath=C:/database\_mongo/

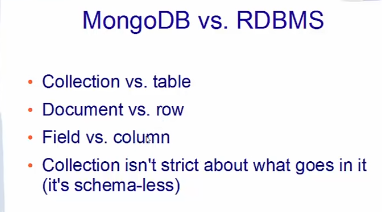
<https://docs.spring.io/spring-data/data-mongo/docs/current/reference/html/#mapping-usage-annotations>

JSON: JavaScript Object Notation

BSON : Binary JSON







**\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\***

**SQL to Aggregation Mapping Chart**

<https://docs.mongodb.com/manual/reference/sql-aggregation-comparison/>

<https://stackoverflow.com/questions/41167699/how-to-write-this-kind-of-mongo-aggregation-match-condition-in-spring>

Aggregation agg = *newAggregation*(

lookupOperation,

*unwind*("incident\_entity"),

Aggregation.*match*(**new** Criteria().orOperator(Criteria.*where*("incident\_entity.created\_timestamp").gt("1509321600000").lte("1509407999000"), Criteria.*where*("incident\_entity.incident\_state").ne("CLOSED"))),

Aggregation.*project*().and("incident\_entity.created\_timestamp").as("incident\_entity.created\_timestamp").and("incident\_id").as("incident\_id").and("customers").as("customers"),

*unwind*("customers"),

Aggregation.*sort*(Sort.Direction.***ASC***,"customers"),

Aggregation.*group*("customers.customerIdValue")

);

Mongodb expert call :

Email Id : [mark@mongodb.com](mailto:mark@mongodb.com)

<https://docs.mongodb.com/manual/reference/operator/aggregation/filter/>

house keep on the indexes

rolling indexing for secondary with primary

db.runCommand( { serverStatus: 1 } )

metrics.document.deleted

bI connector

<https://www.mongodb.com/download-center#bi-connector>

<https://docs.mongodb.com/bi-connector/master/connect/mysql/>

Some Query:

use myNewDB

db.createCollection("people")

db.myNewCollection2.insertOne( { x: 1 } )

db.myNewCollection2.insertMany([{ x: 1 } , { x: 1 }, { x: 1 }])

db.myNewCollection2.insert({ x : 1 })

db.myNewCollection3.createIndex( { y: 1 } )

db.students.count()

db.bakesales.insertMany( [ { date: new ISODate("2018-12-01"), item: "Cake - Chocolate", quantity: 2, amount: new NumberDecimal("60") },

{ date: new ISODate("2018-12-02"), item: "Cake - Peanut Butter", quantity: 5, amount: new NumberDecimal("90") }

])

db.student.aggregate([{ $group: { \_id : {}, SumStudents : {$sum :"$studentNo"} } } ])

By default, a collection does not require its documents to have the same schema; i.e. the documents in a single collection do not need to have the same set of fields and the data type for a field can differ across documents within a collection.

VIEWS:

MongoDB supports creating read-only views from existing collections or other views. For example, you can:

* Create a view that [excludes](https://docs.mongodb.com/manual/reference/operator/aggregation/project/#pipe._S_project) private or confidential data from a collection of employee data.
* Create a view that [adds](https://docs.mongodb.com/manual/reference/operator/aggregation/addFields/#pipe._S_addFields) computed fields from a collection of metrics.
* Create a view that [joins](https://docs.mongodb.com/manual/reference/operator/aggregation/lookup/#pipe._S_lookup) data from two different related collections.

db.createView(<view>, <source>, <pipeline>, <collation> )

<https://docs.mongodb.com/manual/introduction/>

MongoDB stores data records as BSON documents. BSON is a binary representation of [JSON](https://docs.mongodb.com/manual/reference/glossary/#term-json) documents, though it contains more data types than JSON

### The \_id Field

In MongoDB, each document stored in a collection requires a unique [\_id](https://docs.mongodb.com/manual/reference/glossary/#term-id) field that acts as a [primary key](https://docs.mongodb.com/manual/reference/glossary/#term-primary-key). If an inserted document omits the \_id field, the MongoDB driver automatically generates an [ObjectId](https://docs.mongodb.com/manual/reference/bson-types/#objectid) for the \_id field.

The \_id field has the following behavior and constraints:

* By default, MongoDB creates a unique index on the \_id field during the creation of a collection.
* The \_id field is always the first field in the documents. If the server receives a document that does not have the \_id field first, then the server will move the field to the beginning.
* The \_id field may contain values of any [BSON data type](https://docs.mongodb.com/manual/reference/bson-types/), other than an array.

[BSON](https://docs.mongodb.com/manual/reference/glossary/#term-bson) is a binary serialization format used to store documents and make remote procedure calls in MongoDB.

## ObjectId

ObjectIds are small, likely unique, fast to generate, and ordered. ObjectId values consist of 12 bytes, where the first four bytes are a timestamp that reflect the ObjectId’s creation. Specifically:

* a 4-byte value representing the seconds since the Unix epoch,
* a 5-byte random value, and
* a 3-byte counter, starting with a random value.

show dbs

show collections

db.collection.find()

db.collection.updateOne( { \_id: 10 },{ $inc: { calc: 5 } } )

show users

show roles

show profiles

## Document Database

A record in MongoDB is a document, which is a data structure composed of field and value pairs. MongoDB documents are similar to JSON objects. The values of fields may include other documents, arrays, and arrays of documents.

### Modifying Document Structure

To change the structure of the documents in a collection, such as add new fields, remove existing fields, or change the field values to a new type, update the documents to the new structure.

The [mongo](https://docs.mongodb.com/manual/reference/program/mongo/#bin.mongo) shell is an interactive JavaScript interface to MongoDB. You can use the [mongo](https://docs.mongodb.com/manual/reference/program/mongo/#bin.mongo) shell to query and update data as well as perform administrative operations.

# MongoDB CRUD Operations

Within the [shell](https://docs.mongodb.com/manual/tutorial/getting-started/#mongo-web-shell), db refers to your current database. Type db to display the current database.

db

use myNewDB //creating and using database

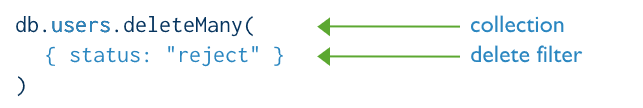
db.createCollection("people")

db.people.insertOne( { x: 1 } ) // insert aswell as create the collection if does not exist.

db.people.insertMany([{ x: 1 } , { x: 1 }, { x: 1 }])



db.user.updateOne({name:"sachin"},{$set:{status:"active"}})



db.inventory.insertMany([

{ item: "journal", qty: 25, size: { h: 14, w: 21, uom: "cm" }, status: "A" },

{ item: "notebook", qty: 50, size: { h: 8.5, w: 11, uom: "in" }, status: "A" },

{ item: "paper", qty: 100, size: { h: 8.5, w: 11, uom: "in" }, status: "D" },

{ item: "planner", qty: 75, size: { h: 22.85, w: 30, uom: "cm" }, status: "D" },

{ item: "postcard", qty: 45, size: { h: 10, w: 15.25, uom: "cm" }, status: "A" }

]);

db.inventory.find( {} )

db.inventory.find( { status: "D" } )

db.inventory.find( { status: { $in: [ "A", "D" ] } } )

db.user.find({status: { $in : ["active","pending"]}}).pretty()

db.inventory.find( { status: "A", qty: { $lt: 30 } } ) // default and operation

db.user.find({ $or : [{name:"sachin"},{age:{ $lte : 20}}]})

db.inventory.find( { $or: [ { status: "A" }, { qty: { $lt: 30 } } ] } )

db.inventory.find( {

status: "A",

$or: [ { qty: { $lt: 30 } }, { item: /^p/ } ]

} ) // item LIKE “p%”

db.inventory.find( { "size.uom": "in" } )

db.inventory.find( { dim\_cm: { $gt: 15, $lt: 20 } } )

db.inventory.insertMany( [

{ item: "journal", instock: [ { warehouse: "A", qty: 5 }, { warehouse: "C", qty: 15 } ] },

{ item: "notebook", instock: [ { warehouse: "C", qty: 5 } ] },

{ item: "paper", instock: [ { warehouse: "A", qty: 60 }, { warehouse: "B", qty: 15 } ] },

{ item: "planner", instock: [ { warehouse: "A", qty: 40 }, { warehouse: "B", qty: 5 } ] },

{ item: "postcard", instock: [ { warehouse: "B", qty: 15 }, { warehouse: "C", qty: 35 } ] }

]);

db.inventory.find( { 'instock.qty': { $lte: 20 } } )

Projection:

A projection can explicitly include several fields by setting the <field> to 1 in the projection document. The following operation returns all documents that match the query. In the result set, only the item, status and, by default, the \_id fields return in the matching documents.

db.user.find({},{name:1}) //display

db.user.find({},{name:0}) //not to display

## Existence Check

db.user.find({sports:{$exists:true}})

> var myCursor = db.user.find()

> myCursor

{ "\_id" : ObjectId("5dd668175f4be4bcbd589d94"), "name" : "sachin", "age" : 12, "status" : "pending" }

{ "\_id" : ObjectId("5dd668175f4be4bcbd589d95"), "name" : "anand", "age" : 20, "status" : "active" }

{ "\_id" : ObjectId("5dfa27168172f3dca247f726"), "name" : "anish", "sports" : [ "TT", "cricket" ] }

<https://docs.mongodb.com/manual/reference/sql-comparison/>

db.people.drop()

db.people.find( { status: "A" } ).sort( { user\_id: 1 } ) //Ascending order

db.people.find( { status: "A" } ).sort( { user\_id: -1 } ) //Des order

db.people.count() or db.people.find().count()

db.user.find().limit(1)

db.user.find().limit(1).skip(1)

db.stores.createIndex( { name: "text", description: "text" } )

db.stores.find( { $text: { $search: "java coffee shop" } } )

# Aggregation

Aggregation 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.

MongoDB provides three ways to perform aggregation:

the [aggregation pipeline](https://docs.mongodb.com/manual/aggregation/#aggregation-framework),

the [map-reduce function](https://docs.mongodb.com/manual/aggregation/#aggregation-map-reduce),

and [single purpose aggregation methods](https://docs.mongodb.com/manual/aggregation/#single-purpose-agg-operations).

## Aggregation Pipeline

MongoDB’s [aggregation framework](https://docs.mongodb.com/manual/core/aggregation-pipeline/) is modeled on the concept of data processing pipelines. Documents enter a multi-stage pipeline that transforms the documents into an aggregated result. For example:

db.orders.aggregate([

{ $match: { status: "A" } },

{ $group: { \_id: "$cust\_id", total: { $sum: "$amount" } } }

])

Other pipeline operations provide tools for grouping and sorting documents by specific field or fields as well as tools for aggregating the contents of arrays, including arrays of documents. In addition, pipeline stages can use [operators](https://docs.mongodb.com/manual/reference/operator/aggregation/#aggregation-expression-operators) for tasks such as calculating the average or concatenating a string.

The aggregation pipeline can operate on a [sharded collection](https://docs.mongodb.com/manual/sharding/).

The aggregation pipeline can use indexes to improve its performance during some of its stages.

## Pipeline

The MongoDB aggregation pipeline consists of [stages](https://docs.mongodb.com/manual/reference/operator/aggregation-pipeline/#aggregation-pipeline-operator-reference). Each stage transforms the documents as they pass through the pipeline.

## Pipeline Expressions

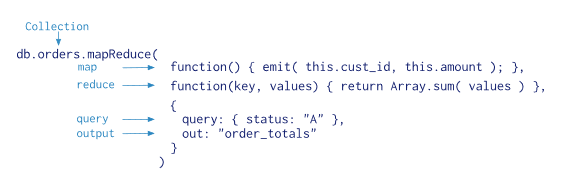
Pipeline expressions can only operate on the current document in the pipeline and cannot refer to data from other documents: expression operations provide in-memory transformation of documents.

## Map-Reduce

MongoDB also provides [map-reduce](https://docs.mongodb.com/manual/core/map-reduce/) operations to perform aggregation. In general, map-reduce operations have two phases: a map stage that processes each document and emits one or more objects for each input document, and reduce phase that combines the output of the map operation. Optionally, map-reduce can have a finalize stage to make final modifications to the result. Like other aggregation operations, map-reduce can specify a query condition to select the input documents as well as sort and limit the results.

in general, map-reduce is less efficient and more complex than the aggregation pipeline.

Map-reduce can operate on a [sharded collection](https://docs.mongodb.com/manual/sharding/).



## Single Purpose Aggregation Operations

MongoDB also provides [db.collection.estimatedDocumentCount()](https://docs.mongodb.com/manual/reference/method/db.collection.estimatedDocumentCount/#db.collection.estimatedDocumentCount), [db.collection.count()](https://docs.mongodb.com/manual/reference/method/db.collection.count/#db.collection.count) and [db.collection.distinct()](https://docs.mongodb.com/manual/reference/method/db.collection.distinct/#db.collection.distinct).

All of these operations aggregate documents from a single collection. While these operations provide simple access to common aggregation processes, they lack the flexibility and capabilities of the aggregation pipeline and map-reduce.

db.orders.distinct(“customer\_id”)

db.orders.count()

# Aggregation Pipeline Optimization

Aggregation pipeline operations have an optimization phase which attempts to reshape the pipeline for improved performance.

## Projection Optimization

The aggregation pipeline can determine if it requires only a subset of the fields in the documents to obtain the results. If so, the pipeline will only use those required fields, reducing the amount of data passing through the pipeline.

## Pipeline Sequence Optimization

### ($project or $unset or $addFields or $set) + $match Sequence Optimization

For an aggregation pipeline that contains a projection stage ([$project](https://docs.mongodb.com/manual/reference/operator/aggregation/project/#pipe._S_project) or [$unset](https://docs.mongodb.com/manual/reference/operator/aggregation/unset/#pipe._S_unset) or [$addFields](https://docs.mongodb.com/manual/reference/operator/aggregation/addFields/#pipe._S_addFields) or [$set](https://docs.mongodb.com/manual/reference/operator/aggregation/set/#pipe._S_set)) followed by a [$match](https://docs.mongodb.com/manual/reference/operator/aggregation/match/#pipe._S_match) stage, MongoDB moves any filters in the [$match](https://docs.mongodb.com/manual/reference/operator/aggregation/match/#pipe._S_match) stage that do not require values computed in the projection stage to a new [$match](https://docs.mongodb.com/manual/reference/operator/aggregation/match/#pipe._S_match) stage before the projection.

### $sort + $match Sequence Optimization

When you have a sequence with [$sort](https://docs.mongodb.com/manual/reference/operator/aggregation/sort/#pipe._S_sort) followed by a [$match](https://docs.mongodb.com/manual/reference/operator/aggregation/match/#pipe._S_match), the [$match](https://docs.mongodb.com/manual/reference/operator/aggregation/match/#pipe._S_match) moves before the [$sort](https://docs.mongodb.com/manual/reference/operator/aggregation/sort/#pipe._S_sort) to minimize the number of objects to sort.

{ $match: { status: 'A' } },

{ $sort: { age : -1 } }

### $redact + $match Sequence Optimization

When possible, when the pipeline has the [$redact](https://docs.mongodb.com/manual/reference/operator/aggregation/redact/#pipe._S_redact) stage immediately followed by the [$match](https://docs.mongodb.com/manual/reference/operator/aggregation/match/#pipe._S_match) stage, the aggregation can sometimes add a portion of the [$match](https://docs.mongodb.com/manual/reference/operator/aggregation/match/#pipe._S_match) stage before the [$redact](https://docs.mongodb.com/manual/reference/operator/aggregation/redact/#pipe._S_redact) stage. If the added [$match](https://docs.mongodb.com/manual/reference/operator/aggregation/match/#pipe._S_match) stage is at the start of a pipeline, the aggregation can use an index as well as query the collection to limit the number of documents that enter the pipeline.

{ $match: { year: 2014 } },

{ $redact: { $cond: { if: { $eq: [ "$level", 5 ] }, then: "$$PRUNE", else: "$$DESCEND" } } },

{ $match: { year: 2014, category: { $ne: "Z" } } }

### $project/$unset + $skip Sequence Optimization

When you have a sequence with [$project](https://docs.mongodb.com/manual/reference/operator/aggregation/project/#pipe._S_project) or [$unset](https://docs.mongodb.com/manual/reference/operator/aggregation/unset/#pipe._S_unset) followed by [$skip](https://docs.mongodb.com/manual/reference/operator/aggregation/skip/#pipe._S_skip), the [$skip](https://docs.mongodb.com/manual/reference/operator/aggregation/skip/#pipe._S_skip) moves before [$project](https://docs.mongodb.com/manual/reference/operator/aggregation/project/#pipe._S_project)

{ $sort: { age : -1 } },

{ $skip: 5 },

{ $project: { status: 1, name: 1 } }

## Pipeline Coalescence Optimization

When possible, the optimization phase coalesces a pipeline stage into its predecessor. Generally, coalescence occurs after any sequence reordering optimization.

### $sort + $limit Coalescence

When a [$sort](https://docs.mongodb.com/manual/reference/operator/aggregation/sort/#pipe._S_sort) precedes a [$limit](https://docs.mongodb.com/manual/reference/operator/aggregation/limit/#pipe._S_limit), the optimizer can coalesce the [$limit](https://docs.mongodb.com/manual/reference/operator/aggregation/limit/#pipe._S_limit) into the [$sort](https://docs.mongodb.com/manual/reference/operator/aggregation/sort/#pipe._S_sort) if no intervening stages modify the number of documents (e.g. [$unwind](https://docs.mongodb.com/manual/reference/operator/aggregation/unwind/#pipe._S_unwind), [$group](https://docs.mongodb.com/manual/reference/operator/aggregation/group/#pipe._S_group)).

For example, if the pipeline consists of the following stages:

{ $sort : { age : -1 } },

{ $project : { age : 1, status : 1, name : 1 } },

{ $limit: 5 }

During the optimization phase, the optimizer coalesces the sequence to the following:

{

"$sort" : {

"sortKey" : {

"age" : -1

},

"limit" : NumberLong(5)

}

},

{ "$project" : {

"age" : 1,

"status" : 1,

"name" : 1

}

}

### $limit + $limit Coalescence

When a [$limit](https://docs.mongodb.com/manual/reference/operator/aggregation/limit/#pipe._S_limit) immediately follows another [$limit](https://docs.mongodb.com/manual/reference/operator/aggregation/limit/#pipe._S_limit), the two stages can coalesce into a single [$limit](https://docs.mongodb.com/manual/reference/operator/aggregation/limit/#pipe._S_limit) where the limit amount is the smaller of the two initial limit amounts.

{ $limit: 100 },

{ $limit: 10 }

Then the second [$limit](https://docs.mongodb.com/manual/reference/operator/aggregation/limit/#pipe._S_limit) stage can coalesce into the first [$limit](https://docs.mongodb.com/manual/reference/operator/aggregation/limit/#pipe._S_limit) stage and result in a single [$limit](https://docs.mongodb.com/manual/reference/operator/aggregation/limit/#pipe._S_limit) stage where the limit amount 10 is the minimum of the two initial limits 100 and 10.

{ $limit: 10 }

### $skip + $skip Coalescence

When a [$skip](https://docs.mongodb.com/manual/reference/operator/aggregation/skip/#pipe._S_skip) immediately follows another [$skip](https://docs.mongodb.com/manual/reference/operator/aggregation/skip/#pipe._S_skip), the two stages can coalesce into a single [$skip](https://docs.mongodb.com/manual/reference/operator/aggregation/skip/#pipe._S_skip) where the skip amount is the *sum* of the two initial skip amounts. For example, a pipeline contains the following sequence:

{ $skip: 5 },

{ $skip: 2 }

Then the second [$skip](https://docs.mongodb.com/manual/reference/operator/aggregation/skip/#pipe._S_skip) stage can coalesce into the first [$skip](https://docs.mongodb.com/manual/reference/operator/aggregation/skip/#pipe._S_skip) stage and result in a single [$skip](https://docs.mongodb.com/manual/reference/operator/aggregation/skip/#pipe._S_skip) stage where the skip amount 7 is the sum of the two initial limits 5 and 2.

{ $skip: 7 }

### $match + $match Coalescence

When a [$match](https://docs.mongodb.com/manual/reference/operator/aggregation/match/#pipe._S_match) immediately follows another [$match](https://docs.mongodb.com/manual/reference/operator/aggregation/match/#pipe._S_match), the two stages can coalesce into a single [$match](https://docs.mongodb.com/manual/reference/operator/aggregation/match/#pipe._S_match) combining the conditions with an [$and](https://docs.mongodb.com/manual/reference/operator/aggregation/and/#exp._S_and). For example, a pipeline contains the following sequence:

{ $match: { year: 2014 } },

{ $match: { status: "A" } }

Then the second [$match](https://docs.mongodb.com/manual/reference/operator/aggregation/match/#pipe._S_match) stage can coalesce into the first [$match](https://docs.mongodb.com/manual/reference/operator/aggregation/match/#pipe._S_match) stage and result in a single [$match](https://docs.mongodb.com/manual/reference/operator/aggregation/match/#pipe._S_match) stage

{ $match: { $and: [ { "year" : 2014 }, { "status" : "A" } ] } }

### $lookup + $unwind Coalescence

When a [$unwind](https://docs.mongodb.com/manual/reference/operator/aggregation/unwind/#pipe._S_unwind) immediately follows another [$lookup](https://docs.mongodb.com/manual/reference/operator/aggregation/lookup/#pipe._S_lookup), and the [$unwind](https://docs.mongodb.com/manual/reference/operator/aggregation/unwind/#pipe._S_unwind) operates on the as field of the [$lookup](https://docs.mongodb.com/manual/reference/operator/aggregation/lookup/#pipe._S_lookup), the optimizer can coalesce the [$unwind](https://docs.mongodb.com/manual/reference/operator/aggregation/unwind/#pipe._S_unwind) into the [$lookup](https://docs.mongodb.com/manual/reference/operator/aggregation/lookup/#pipe._S_lookup) stage. This avoids creating large intermediate documents.

For example, a pipeline contains the following sequence:

{

$lookup: {

from: "otherCollection",

as: "resultingArray",

localField: "x",

foreignField: "y"

}

},

{ $unwind: "$resultingArray"}

The optimizer can coalesce the [$unwind](https://docs.mongodb.com/manual/reference/operator/aggregation/unwind/#pipe._S_unwind) stage into the [$lookup](https://docs.mongodb.com/manual/reference/operator/aggregation/lookup/#pipe._S_lookup) stage. If you run the aggregation with explain option, the explain output shows the coalesced stage:

{

$lookup: {

from: "otherCollection",

as: "resultingArray",

localField: "x",

foreignField: "y",

unwinding: { preserveNullAndEmptyArrays: false }

}

}

### $sort + $skip + $limit Sequence

A pipeline contains a sequence of [$sort](https://docs.mongodb.com/manual/reference/operator/aggregation/sort/#pipe._S_sort) followed by a [$skip](https://docs.mongodb.com/manual/reference/operator/aggregation/skip/#pipe._S_skip) followed by a [$limit](https://docs.mongodb.com/manual/reference/operator/aggregation/limit/#pipe._S_limit):

{ $sort: { age : -1 } },

{ $skip: 10 },

{ $limit: 5 }

The optimizer performs [$sort + $limit Coalescence](https://docs.mongodb.com/manual/core/aggregation-pipeline-optimization/#agg-sort-limit-coalescence) to transforms the sequence to the following:

{

"$sort" : {

"sortKey" : {

"age" : -1

},

"limit" : NumberLong(15)

}

},

{

"$skip" : NumberLong(10)

}

## Result Size Restrictions

The [aggregate](https://docs.mongodb.com/manual/reference/command/aggregate/#dbcmd.aggregate) command can return either a cursor or store the results in a collection. When returning a cursor or storing the results in a collection, each document in the result set is subject to the [BSON Document Size](https://docs.mongodb.com/manual/reference/limits/#BSON-Document-Size) limit, currently 16 megabytes; if any single document that exceeds the [BSON Document Size](https://docs.mongodb.com/manual/reference/limits/#BSON-Document-Size) limit, the command will produce an error. The limit only applies to the returned documents; during the pipeline processing, the documents may exceed this size. The [db.collection.aggregate()](https://docs.mongodb.com/manual/reference/method/db.collection.aggregate/#db.collection.aggregate) method returns a cursor by default.

## Memory Restrictions

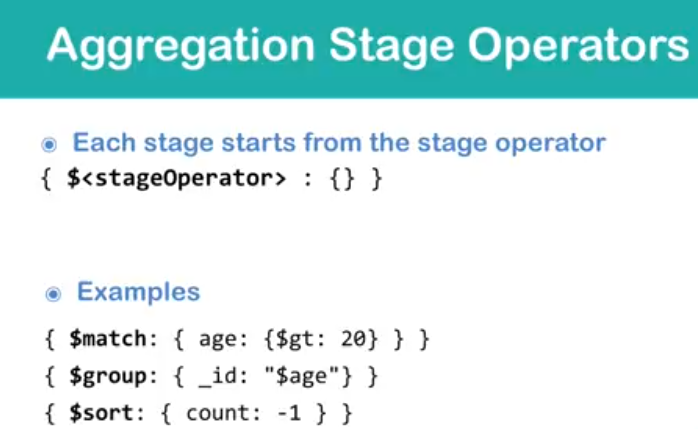
Pipeline stages have a limit of 100 megabytes of RAM. If a stage exceeds this limit, MongoDB will produce an error. To allow for the handling of large datasets, use the allowDiskUse option to enable aggregation pipeline stages to write data to temporary files.

# MongoDB Aggregation

Documents during aggregation pass through the stage.

db.user.aggregate([]) – without any stages works same as that of find function.

# Aggregation Stages



Some of aggregation operator:

$match – match for the particular value.

$group – group based on particular criteria.

$project – views particular fields of documents

$sort – sort the document

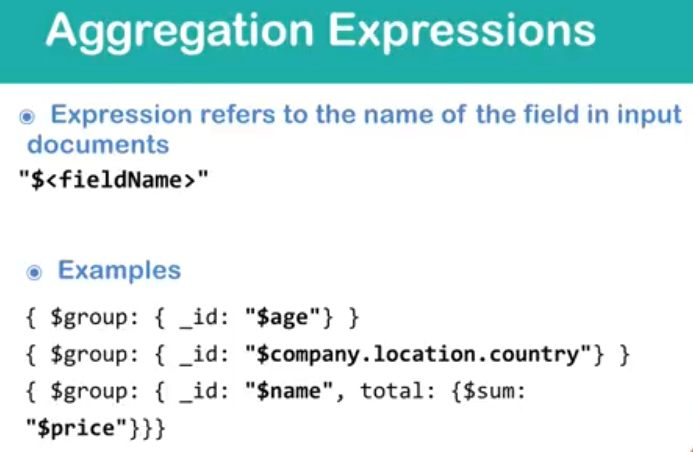
$count – count the document

$limit – limit no of doc

$skip – skip no of doc

$out – out the result for another document.

# Aggregation Expressions



$match

{$match:{ <query> }}

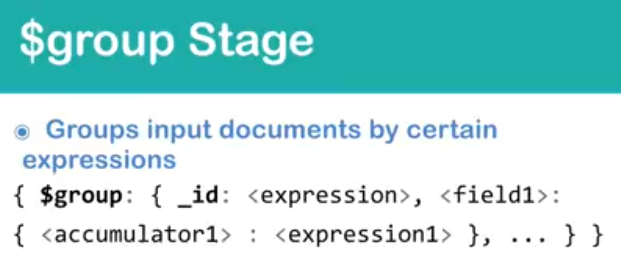
> db.user.aggregate([{$match : {name:"Anish"}}])

{ "\_id" : 2, "name" : "Anish", "last" : "Shah", "birthDate" : ISODate("2019-12-30T12:46:40.972Z"), "setting" : { }, "createdOn" : NumberLong("1577710000972"), "\_class" : "com.sachin.mongotempleate.mongotemplateexample.document.User" }

{ "\_id" : 7, "name" : "Anish", "last" : "Shah", "birthDate" : ISODate("2019-12-30T12:47:21.520Z"), "setting" : { "job" : "ATT", "friend" : "Anish", "other" : "Shah" }, "createdOn" : NumberLong("1577710041520"), "\_class" : "com.sachin.mongotempleate.mongotemplateexample.document.User" }

db.user.aggregate([{$match : {$or : [{name:"Anish"},{name:"sachin"}]}}]);

$group



> db.user.aggregate([{$group : {\_id:"$name"}}])

{ "\_id" : "Anish" }

{ "\_id" : "sachin" }

> db.user.aggregate([{$group : {\_id:{name:"$name"}}}])

{ "\_id" : { "name" : "Anish" } }

{ "\_id" : { "name" : "sachin" } }

//multiple field group gives all combination value

> db.user.aggregate([{$group : {\_id:{name:"$name",last : "$last"}}}])

{ "\_id" : { "name" : "Anish", "last" : "Shah" } }

{ "\_id" : { "name" : "sachin", "last" : "chippalkatti" } }

//nested documents

> db.user.aggregate([{$group : {\_id:"$setting.job"}}])

{ "\_id" : null }

{ "\_id" : "ATT" }

> db.user.aggregate([{$match:{name:"sachin"}}, {$group :{\_id:"$birthDate"}}])

{ "\_id" : null }

{ "\_id" : ISODate("2019-12-30T10:51:31.717Z") }

{ "\_id" : ISODate("2019-12-30T11:00:56.038Z") }

{ "\_id" : ISODate("2019-12-30T11:07:14.769Z") }

$count

{$count:<title>}

Count will be always at the last command

db.user.aggregate([{$count:"allDoc"}])

Other count operation

db.user.aggregate([]).toArray().length

db.user.aggregate([]).itcount()

db.user.aggregate([{$group : {\_id:{name:"$name",last : "$last"},count : {$sum:1}}}])

{ "\_id" : { "name" : "Anish", "last" : "Shah" }, "count" : 2 }

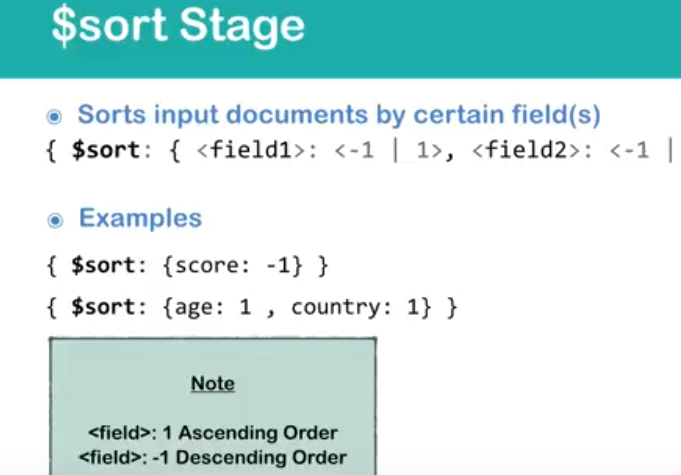
{ "\_id" : { "name" : "sachin", "last" : "chippalkatti" }, "count" : 5 }

$sort

db.user.aggregate([{$group : {\_id:{name:"$name",last : "$last"},count : {$sum:1}}},{$sort : {"\_id.last":-1}}])

{ "\_id" : { "name" : "sachin", "last" : "chippalkatti" }, "count" : 5 }

{ "\_id" : { "name" : "Anish", "last" : "Shah" }, "count" : 2 }



> db.user.aggregate([{$group : {\_id:{name:"$name",last : "$last"},count : {$sum:1}}},{$sort : {"\_id.last":-1,"\_id.count":-1}}])

{ "\_id" : { "name" : "sachin", "last" : "chippalkatti" }, "count" : 5 }

{ "\_id" : { "name" : "Anish", "last" : "Shah" }, "count" : 2 }

$project

Include, exclude or adds new field

{$project :{<field1> : 1, <field2>:0}}

> db.user.aggregate([{$group : {\_id:{name:"$name",last : "$last"},count : {$sum:1}}},{$project:{"\_id.name":1}}])

{ "\_id" : { "name" : "Anish" } }

{ "\_id" : { "name" : "sachin" } }

// creating the new filed and renaming the existing and restructure

db.user.aggregate([{$group : {\_id:{name:"$name",last : "$last"},count : {$sum:1}}},{$project:{name:"$\_id.name",last:"$\_id.last",\_id:0}}])

{ "name" : "Anish", "last" : "Shah" }

{ "name" : "sachin", "last" : "chippalkatti" }

$limit

{$limit: <number>}

This will result first n documents

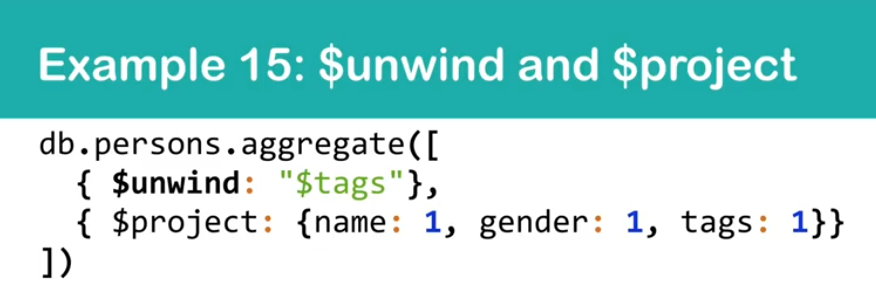
> db.user.aggregate([{$group : {\_id:{name:"$name",last : "$last"},count : {$sum:1}}},{$project:{name:"$\_id.name",last:"$\_id.last",\_id:0}},{$limit:1}])

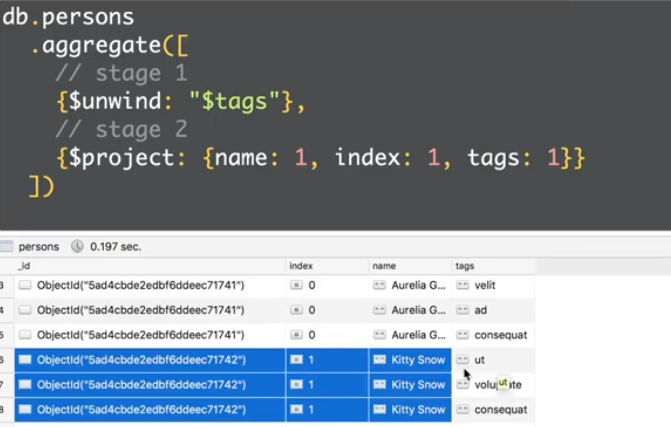
{ "name" : "Anish", "last" : "Shah" }

$unwind

Is used on array of fields.

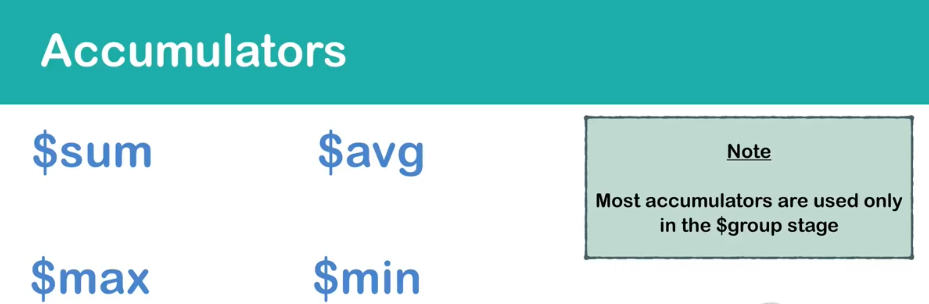
Unwind create separate document for each of array value of fields.

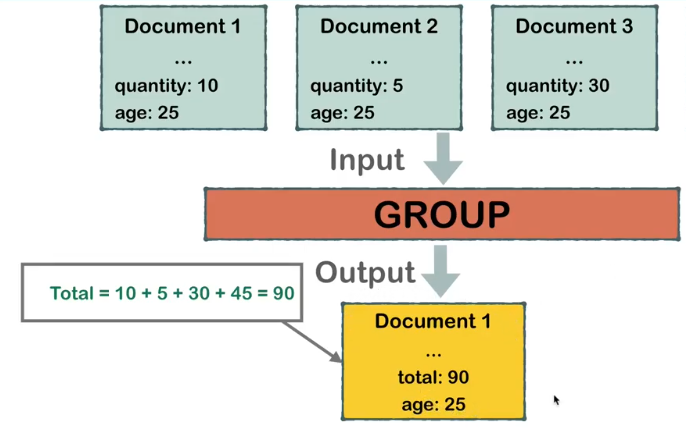




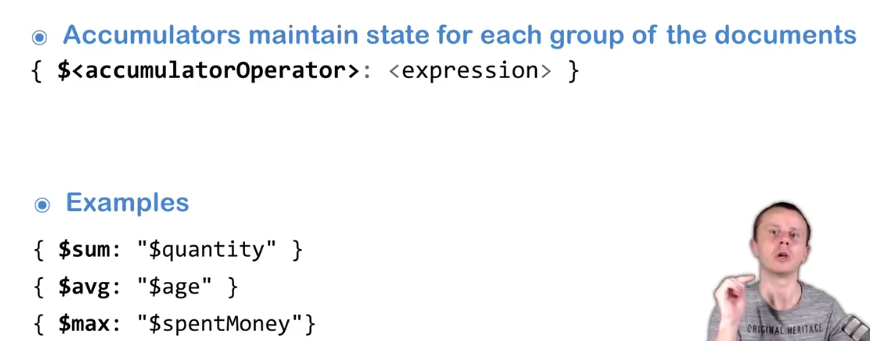


# Accumulators - MongoDB Aggregation

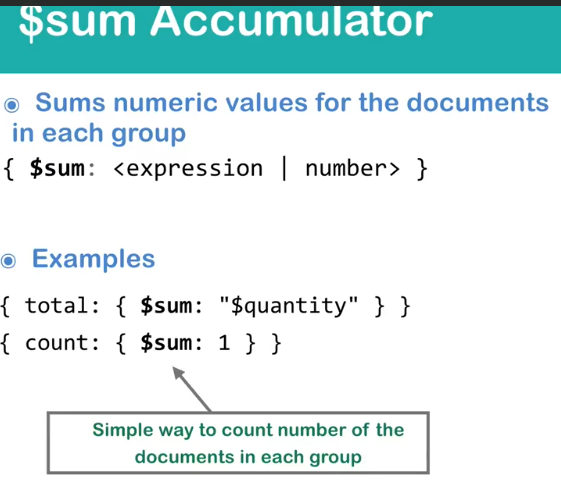




Syntax:



$sum



> db.user.aggregate([{$group:{\_id:"$name"}}])

{ "\_id" : "Anish" }

{ "\_id" : "sachin" }

> db.user.aggregate([{$group:{\_id:"$name",count:{$sum:1}}}])

{ "\_id" : "Anish", "count" : 2 }

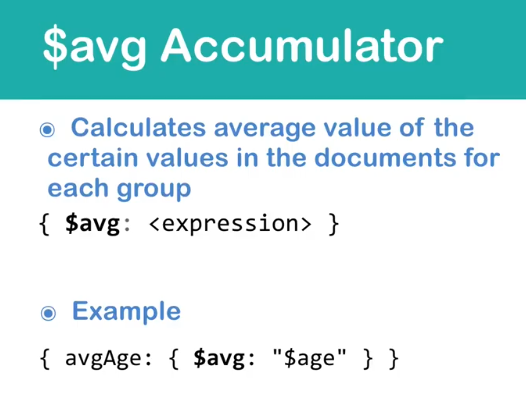
{ "\_id" : "sachin", "count" : 5 }

db.user.aggregate([{$group:{\_id:"$name",count:{$sum:NumberInt(1)}}}])

{ "\_id" : "Anish", "count" : 2 }

{ "\_id" : "sachin", "count" : 5 }

$avg

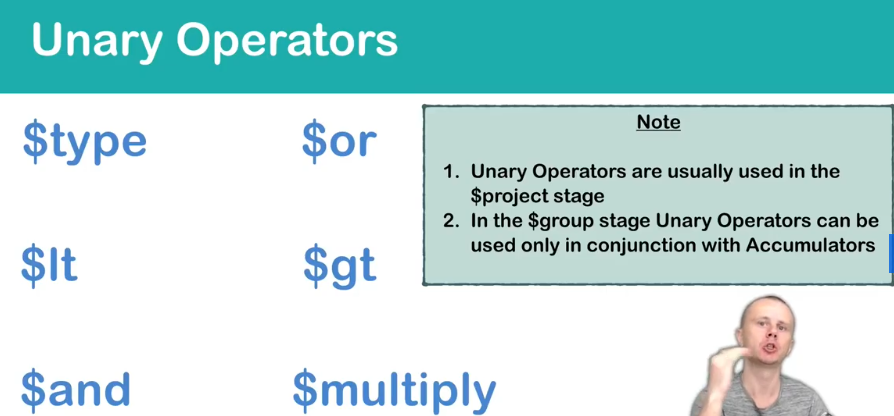


> db.user.aggregate([{$group:{\_id:"$name",avg:{$avg:"$\_id"}}}])

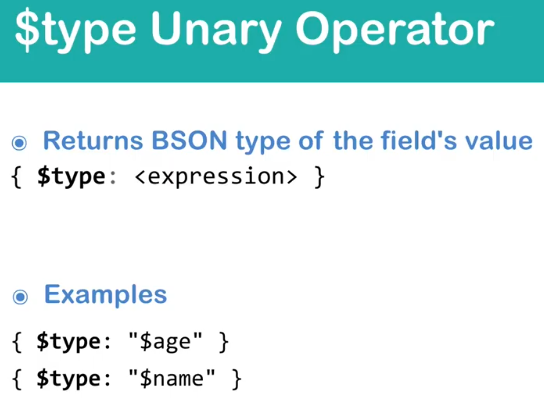
{ "\_id" : "Anish", "avg" : 4.5 }

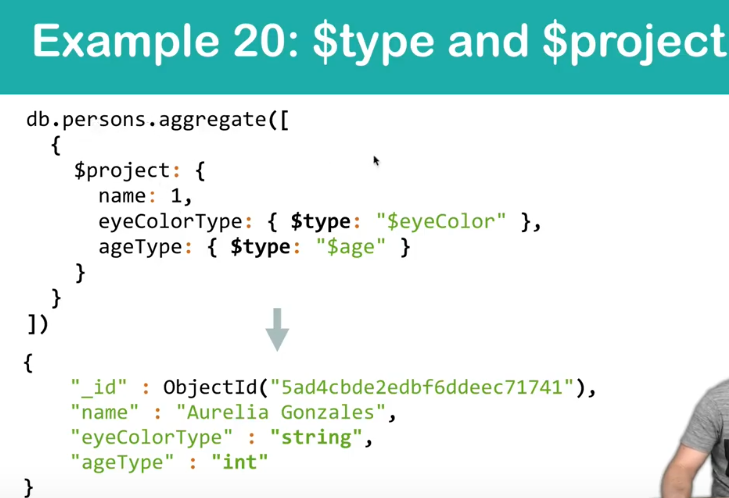
{ "\_id" : "sachin", "avg" : 3.8 }

# Unary Operators - MongoDB Aggregation

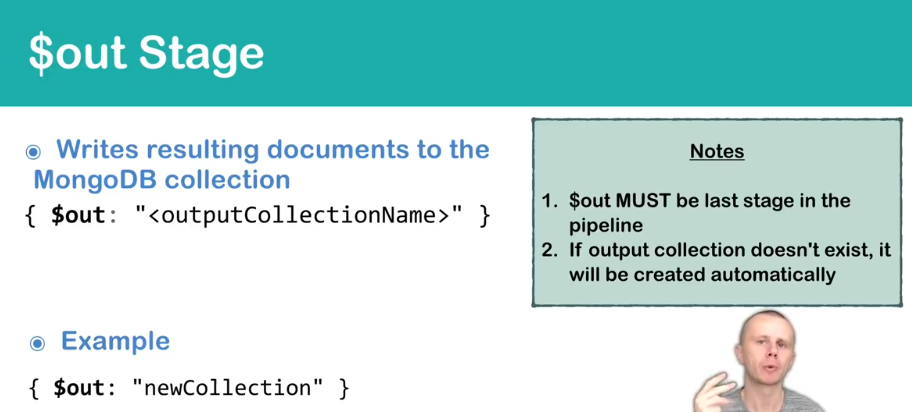


$type

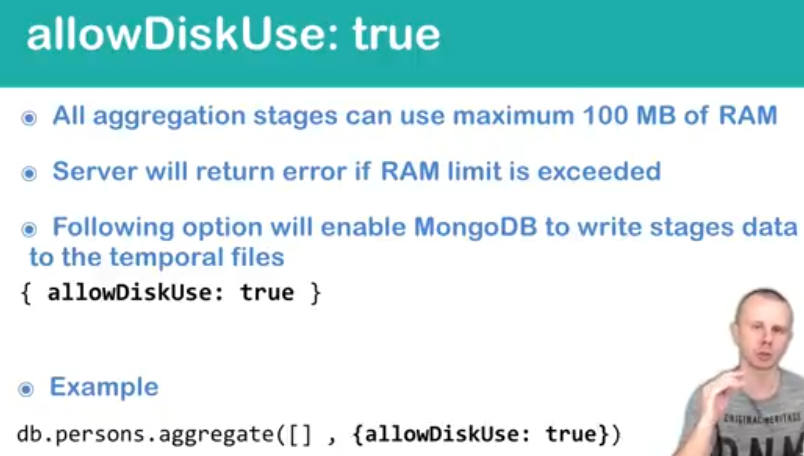




# $out - MongoDB Aggregation



# allowDiskUse Option - MongoDB Aggregation



db.user.aggregate([{$project:{name:1,last:1,setting:1}}],{allowDiskUse:true})

{ "\_id" : 2, "name" : "Anish", "last" : "Shah", "setting" : { } }

{ "\_id" : 1, "name" : "sachin", "last" : "chippalkatti" }

{ "\_id" : 3, "name" : "sachin", "last" : "chippalkatti" }

{ "\_id" : 4, "name" : "sachin", "last" : "chippalkatti", "setting" : { "job" : "ATT" } }

{ "\_id" : 5, "name" : "sachin", "last" : "chippalkatti", "setting" : { "job" : "ATT" } }

{ "\_id" : 6, "name" : "sachin", "last" : "chippalkatti", "setting" : { "job" : "ATT" } }

{ "\_id" : 7, "name" : "Anish", "last" : "Shah", "setting" : { "job" : "ATT", "friend" : "Anish", "other" : "Shah" } }

# $lookup(similar to Left Outer Join in RDBMS) in MongoDB

db.user.aggregate([

{

$lookup:{

from:"userDetails",

localField:"name",

foreignField:"name",

as:"userfull"}

}])

{ "\_id" : 2, "name" : "Anish", "last" : "Shah", "birthDate" : ISODate("2019-12-30T12:46:40.972Z"), "setting" : { }, "createdOn" : NumberLong("1577710000972"), "\_class" : "com.sachin.mongotempleate.mongotemplateexample.document.User", "userfull" : [ { "\_id" : 2, "name" : "Anish", "last" : "Shah", "setting" : { } }, { "\_id" : 7, "name" : "Anish", "last" : "Shah", "setting" : { "job" : "ATT", "friend" : "Anish", "other" : "Shah" } } ] }

{ "\_id" : 1, "name" : "sachin", "last" : "chippalkatti", "\_class" : "com.sachin.mongodb.mongorepository.document.User", "userfull" : [ { "\_id" : 1, "name" : "sachin", "last" : "chippalkatti" }, { "\_id" : 3, "name" : "sachin", "last" : "chippalkatti" }, { "\_id" : 4, "name" : "sachin", "last" : "chippalkatti", "setting" : { "job" : "ATT" } }, { "\_id" : 5, "name" : "sachin", "last" : "chippalkatti", "setting" : { "job" : "ATT" } }, { "\_id" : 6, "name" : "sachin", "last" : "chippalkatti", "setting" : { "job" : "ATT" } } ] }

{ "\_id" : 3, "name" : "sachin", "last" : "chippalkatti", "\_class" : "com.sachin.mongodb.mongorepository.document.User", "userfull" : [ { "\_id" : 1, "name" : "sachin", "last" : "chippalkatti" }, { "\_id" : 3, "name" : "sachin", "last" : "chippalkatti" }, { "\_id" : 4, "name" : "sachin", "last" : "chippalkatti", "setting" : { "job" : "ATT" } }, { "\_id" : 5, "name" : "sachin", "last" : "chippalkatti", "setting" : { "job" : "ATT" } }, { "\_id" : 6, "name" : "sachin", "last" : "chippalkatti", "setting" : { "job" : "ATT" } } ] }

{ "\_id" : 4, "name" : "sachin", "last" : "chippalkatti", "birthDate" : ISODate("2019-12-30T10:51:31.717Z"), "setting" : { "job" : "ATT" }, "\_class" : "com.sachin.mongodb.mongorepository.document.User", "userfull" : [ { "\_id" : 1, "name" : "sachin", "last" : "chippalkatti" }, { "\_id" : 3, "name" : "sachin", "last" : "chippalkatti" }, { "\_id" : 4, "name" : "sachin", "last" : "chippalkatti", "setting" : { "job" : "ATT" } }, { "\_id" : 5, "name" : "sachin", "last" : "chippalkatti", "setting" : { "job" : "ATT" } }, { "\_id" : 6, "name" : "sachin", "last" : "chippalkatti", "setting" : { "job" : "ATT" } } ] }

{ "\_id" : 5, "name" : "sachin", "last" : "chippalkatti", "birthDate" : ISODate("2019-12-30T11:00:56.038Z"), "setting" : { "job" : "ATT" }, "\_class" : "com.sachin.mongodb.mongorepository.document.User", "userfull" : [ { "\_id" : 1, "name" : "sachin", "last" : "chippalkatti" }, { "\_id" : 3, "name" : "sachin", "last" : "chippalkatti" }, { "\_id" : 4, "name" : "sachin", "last" : "chippalkatti", "setting" : { "job" : "ATT" } }, { "\_id" : 5, "name" : "sachin", "last" : "chippalkatti", "setting" : { "job" : "ATT" } }, { "\_id" : 6, "name" : "sachin", "last" : "chippalkatti", "setting" : { "job" : "ATT" } } ] }

{ "\_id" : 6, "name" : "sachin", "last" : "chippalkatti", "birthDate" : ISODate("2019-12-30T11:07:14.769Z"), "setting" : { "job" : "ATT" }, "createdOn" : NumberLong("1577704034769"), "\_class" : "com.sachin.mongodb.mongorepository.document.User", "userfull" : [ { "\_id" : 1, "name" : "sachin", "last" : "chippalkatti" }, { "\_id" : 3, "name" : "sachin", "last" : "chippalkatti" }, { "\_id" : 4, "name" : "sachin", "last" : "chippalkatti", "setting" : { "job" : "ATT" } }, { "\_id" : 5, "name" : "sachin", "last" : "chippalkatti", "setting" : { "job" : "ATT" } }, { "\_id" : 6, "name" : "sachin", "last" : "chippalkatti", "setting" : { "job" : "ATT" } } ] }

{ "\_id" : 7, "name" : "Anish", "last" : "Shah", "birthDate" : ISODate("2019-12-30T12:47:21.520Z"), "setting" : { "job" : "ATT", "friend" : "Anish", "other" : "Shah" }, "createdOn" : NumberLong("1577710041520"), "\_class" : "com.sachin.mongotempleate.mongotemplateexample.document.User", "userfull" : [ { "\_id" : 2, "name" : "Anish", "last" : "Shah", "setting" : { } }, { "\_id" : 7, "name" : "Anish", "last" : "Shah", "setting" : { "job" : "ATT", "friend" : "Anish", "other" : "Shah" } } ] }

db.user.aggregate([

{$match:{name:"Anish"}},

{$lookup:{ from:"userDetails",localField:"name",foreignField:"name", as:"userfull"}}]).pretty()

> db.user.aggregate([

{$match:{name:"Anish"}},

{$lookup:{ from:"userDetails",localField:"name",foreignField:"name", as:"userfull"}},

{$project:{name:1,last:1}}]).pretty()

{ "\_id" : 2, "name" : "Anish", "last" : "Shah" }

{ "\_id" : 7, "name" : "Anish", "last" : "Shah" }

# [Poor lookup aggregation performance](https://stackoverflow.com/questions/43742635/poor-lookup-aggregation-performance)

Posts:

{

"\_Id": "1",

"\_PostTypeId": "1",

"\_AcceptedAnswerId": "192",

"\_CreationDate": "2012-02-08T20:02:48.790",

"\_Score": "10",

...

"\_OwnerUserId": "6",

...

},

...

and users:

{

"\_Id": "1",

"\_Reputation": "101",

"\_CreationDate": "2012-02-08T19:45:13.447",

"\_DisplayName": "Geoff Dalgas",

...

"\_AccountId": "2"

},

...

db.posts.aggregate([

{

$lookup: {

from: "users",

localField: "\_OwnerUserId",

foreignField: "\_AccountId",

as: "X"

}

},

{

$group: {

\_id: "$X.\_AccountId",

posts: { $sum: 1 }

}

},

{

$match : {posts: {$gte: 5, $lte: 15}}

},

{

$sort: {posts: -1 }

},

{

$project : {posts: 1}

}

])

foreignField Specifies the field from the documents in the from collection. **$lookup performs an equality match on the foreignField** to the localField from the input documents. If a document in the from collection does not contain the foreignField, the $lookup treats the value as null for matching purposes.

This will be performed the same as any other query.

If you don't have an index on the field \_AccountId, it will do a full tablescan query for each one of the 10,000 posts. The bulk of the time will be spent in that tablescan.

db.users.ensureIndex("\_AccountId", 1)

speeds up the process so it's doing 10,000 index hits instead of 10,000 table scans.