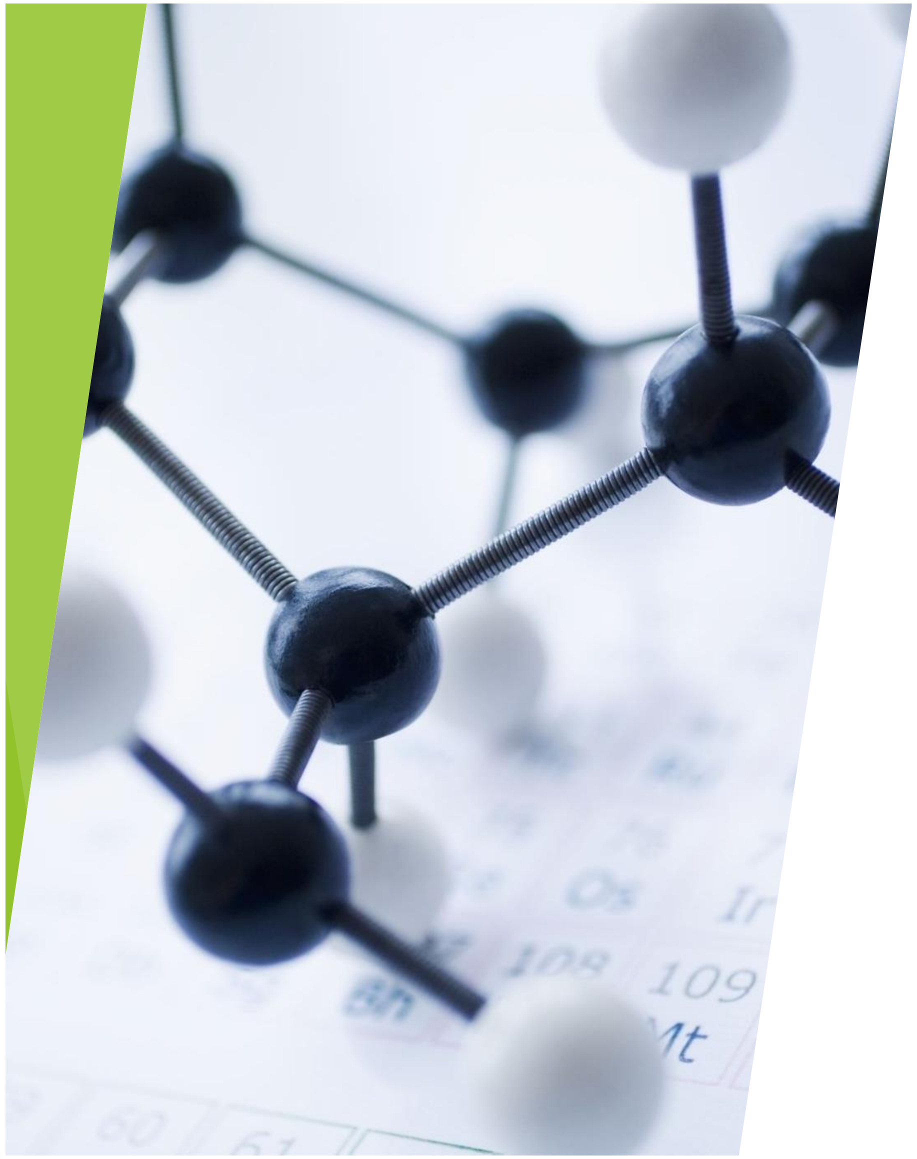


MongoDB – Aggregations

Module Objectives

What you will learn

At the end of this module, you will learn:

- What is Aggregation

What you will be able to do

At the end of this module, you be able to:

- Explain what is Aggregation
- List the various types of Aggregation
- Understand the Pipeline concept
- Describe the various types of Aggregation
- State examples for the various types of Aggregation

Types of Aggregations

$match, $unwind	$group, $project	$skip, $limit
$sort, $first	$last, $sum	$avg, $min, $max
	$push, $addToSet	

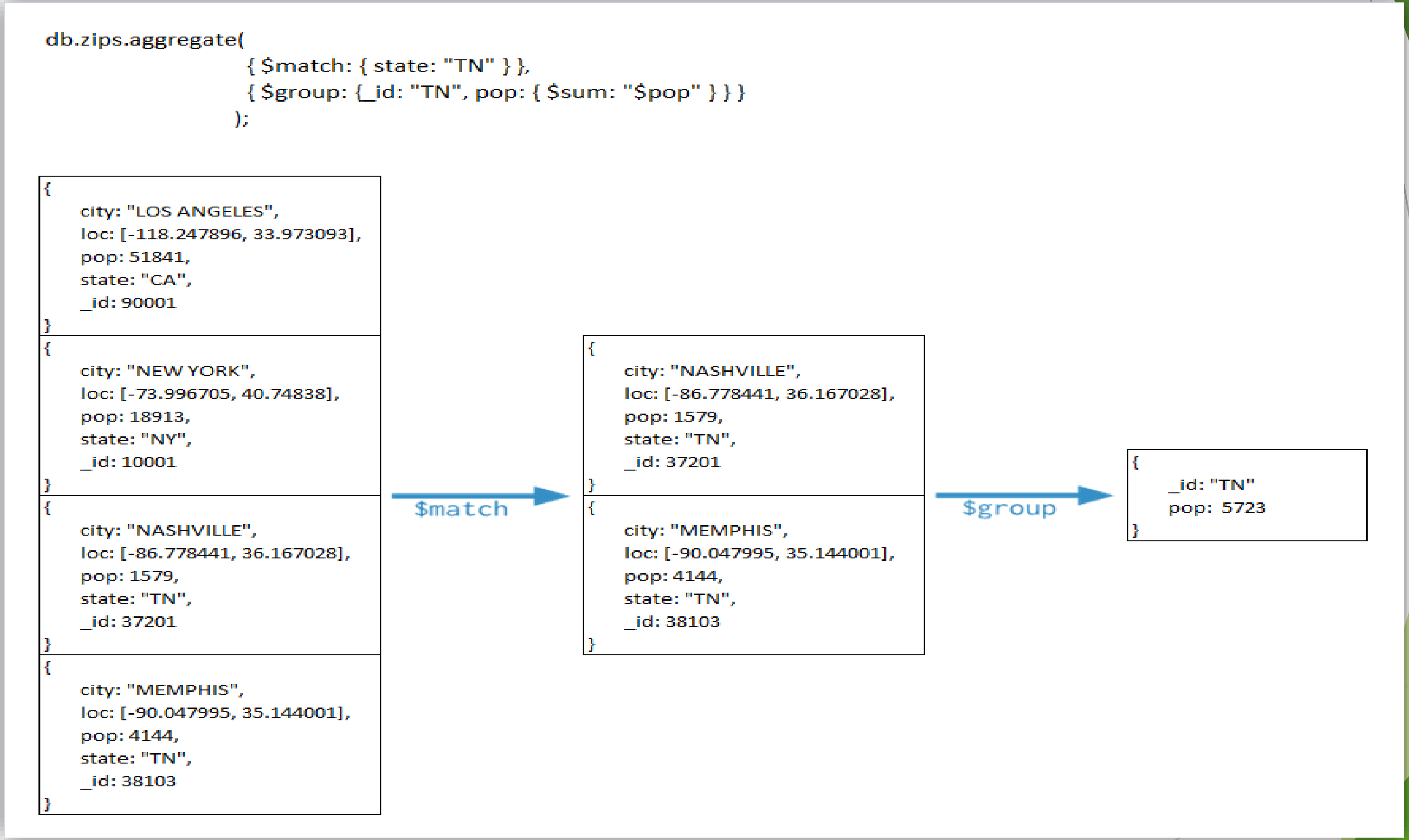
What is Aggregation?

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.

What is Aggregation? (contd.)



```
db.zips.aggregate(
                    { $match: { state: "TN" } },
                    { $group: {_id: "TN", pop: { $sum: "$pop" } } }
                  );
```

```
{
    city: "LOS ANGELES",
    loc: [-118.247896, 33.973093],
    pop: 51841,
    state: "CA",
    _id: 90001
}
{
    city: "NEW YORK",
    loc: [-73.996705, 40.74838],
    pop: 18913,
    state: "NY",
    _id: 10001
}
{
    city: "NASHVILLE",
    loc: [-86.778441, 36.167028],
    pop: 1579,
    state: "TN",
    _id: 37201
}
{
    city: "MEMPHIS",
    loc: [-90.047995, 35.144001],
    pop: 4144,
    state: "TN",
    _id: 38103
}
```

```
{
    city: "NASHVILLE",
    loc: [-86.778441, 36.167028],
    pop: 1579,
    state: "TN",
    _id: 37201
}
{
    city: "MEMPHIS",
    loc: [-90.047995, 35.144001],
    pop: 4144,
    state: "TN",
    _id: 38103
}
```

```
{
    _id: "TN"
    pop:  5723
}
```

The Aggregate() Method

For the aggregation in Mongodb you should use **aggregate()** method.

Syntax:

Basic syntax of **aggregate()** method is as follows:

- **>db.COLLECTION_NAME.aggregate(AGGREGATE_OPERATION)**

Pipeline Concept

The aggregation framework is based on pipeline concept, just like Unix pipeline.

There can be N number of operators.

Output of first operator will be fed as input to the second operator. Output of second operator will be fed as input to the third operator and so on.

Pipelines

Modeled on the concept of data processing pipelines.

Provides:

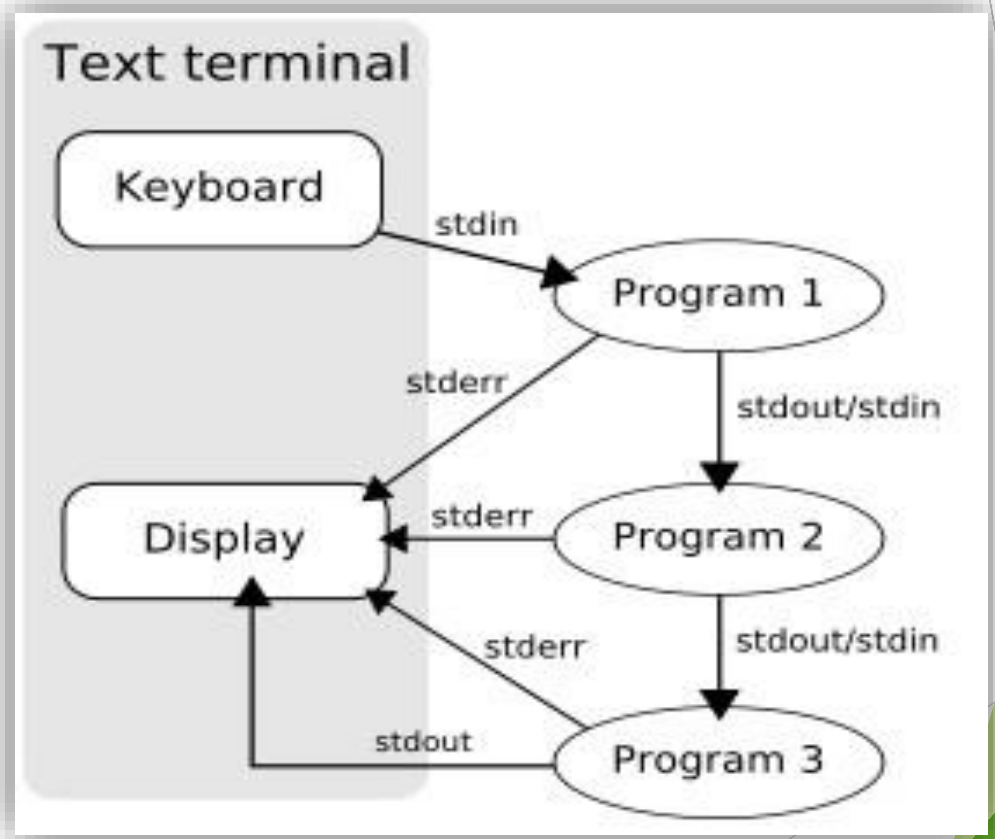
- *Filters* that operate like queries.
- *Document transformations* that modify the form of the output document.

Provides tools for:

- Grouping and sorting by field.
- Aggregating the contents of arrays, including arrays of documents.

Can use **operators** for tasks such as calculating the average or concatenating a string.

Pipeline Flow



Pipeline Operators

The basic pipeline operators are:

$match	**$unwind**	**$group**
$project	**$skip**	**$limit**
	$sort	

$match

This is similar to MongoDB Collection's find method and SQL's WHERE clause.

Example: We want to consider only the marks of the students who study in Class “2”.

- db.Student.aggregate ([{ "$match": { "Class":"2" } }])

$unwind

This will be very useful when the data is stored as list.

When the unwind operator is applied on a list data field, it will generate a new record for each and every element of the list data field on which unwind is applied.

It basically flattens the data.

Example

```
db.Student.aggregate ([ { "$match": { "Student_Name": "Kalki", } }, { "$unwind": "$Subject" } ])
```

```
{ "result" : [ { "_id" : ObjectId("517cbb98eccb9ee3d000fa5c"), "Student_Name" : "Kalki", "Class" : "2", "Mark_Scored" : 100, "Subject" : "Tamil" }, { "_id" : ObjectId("517cbb98eccb9ee3d000fa5c"), "Student_Name" : "Kalki", "Class" : "2", "Mark_Scored" : 100, "Subject" : "English" }, { "_id" : ObjectId("517cbb98eccb9ee3d000fa5c"), "Student_Name" : "Kalki", "Class" : "2", "Mark_Scored" : 100, "Subject" : "Maths" } ], "ok" : 1 }
```

$group

The group pipeline operator is similar to the SQL's GROUP BY clause.

Example: Lets try and get the sum of all the marks scored by each and every student, in Class "2".

- db.Student.aggregate ([{ "$match": { "Class": "2" } }, { "$unwind": "$Subject" }, { "$group": { "_id": { "Student_Name" : "$Student_Name" }, "Total_Marks": { "$sum": "$Mark_Scored" } } }])

$group (contd.)

In this aggregation example, we have specified an _id element and Total_Marks element.

The _id element tells MongoDB to group the documents based on Student_Name field.

The Total_Marks uses an aggregation function **$sum**, which basically adds up all the marks and returns the sum.

$project

The project operator is similar to SELECT in SQL.

We can use this to rename the field names and select / deselect the fields to be returned, out of the grouped fields.

If we specify 0 for a field, it will NOT be sent in the pipeline to the next operator.

$Sort

This is similar to SQL's ORDER BY clause. To sort a particular field in descending order specify -1 and specify 1 if that field has to be sorted in ascending order.

- db.Student.aggregate ([{ "$match": { "Class": "2" } }, { "$unwind": "$Subject" }, { "$group": { "_id": { "Student_Name" : "$Student_Name" }, "Total_Marks": { "$sum": "$Mark_Scored" } } },
- { "$project": { "_id":0, "Name": "$_id.Student_Name", "Total":
- "$Total_Marks" } }, { "$sort": { "Total":-1, "Name":1 } }])

17

$limit and $skip

These two operators can be used to limit the number of documents being returned. They will be more useful when we need pagination support.

$first

Returns the value that results from applying an expression to the first document in a group of documents that share the same group by key. Only meaningful when documents are in a defined order.

$first is only available in the **$group** stage.

Example

Grouping the documents by the item field, the following operation uses the $first accumulator to compute the first sales date for each item.

```
db.sales.aggregate( [ { $sort: { item: 1, date: 1 } }, { $group: { _id:
"$item", firstSalesDate: { $first: "$date" } } } ] )
```

$last

$last returns the value that results from applying an expression to the last document in a group of documents that share the same group by a field.

Only meaningful when documents are in a defined order.

$last is only available in the **$group** stage.

$last (contd.)

The following operation first sorts the documents by item and date, and then in the following **$group** stage, groups the now sorted documents by the item field and uses the $last accumulator to compute the last sales date for each item:

- db.sales.aggregate([{ $sort: { item: 1, date: 1 } }, { $group: { _id: "$item", lastSalesDate: { $last: "$date" } } }])

$sum

Calculates and returns the sum of numeric values. $sum ignores non-numeric values.

When used in the **$group** stage, $sum has the following syntax and returns the collective sum of all the numeric values that result from applying a specified expression to each document in a group of documents that share the same group by key.

Example

Grouping the documents by the day and the year of the date field, the following operation uses the $sum accumulator to compute the total amount and the count for each group of documents.

- db.sales.aggregate([{ $group: { _id: { day: { $dayOfYear: "$date"}, year: { $year: "$date" } }, totalAmount: { $sum: { $multiply: ["$price", "$quantity"] } }, count: { $sum: 1 } } }])

$avg

Returns the average value of the numeric values. $avg ignores non-numeric values.

$avg returns the collective average of all the numeric values that result from applying a specified expression to each document in a group of documents that share the same group by key.

Example

Grouping the documents by the item field, the following operation uses the $avg accumulator to compute the average amount and average quantity for each grouping.

- db.sales.aggregate([{ $group: { _id: "$item", avgAmount: { $avg: { $multiply: ["$price", "$quantity"] } }, avgQuantity: { $avg: "$quantity" } } }])

$max

Returns the maximum value. $max compares both value and type, using the

specified BSON comparison order for values of different types.

Grouping the documents by the item field, the following operation uses the

$max accumulator to compute the maximum total amount and maximum quantity for each group of documents.

- db.sales.aggregate([{ $group: { _id: "$item", maxTotalAmount: { $max: {
- $multiply: ["$price", "$quantity"] } }, maxQuantity: { $max: "$quantity" } } }
-])

$min

Returns the minimum value. $min compares both value and type, using

the ***specified BSON comparison order*** for values of different types.

Grouping the documents by the item field, the following operation uses the $min accumulator to compute the minimum amount and minimum quantity for each grouping.

- db.sales.aggregate([{ $group: { _id: "$item", minQuantity: { $min:
- "$quantity" } } }])

$push

Returns an array of *all* values that result from applying an expression to each document in a group of documents that share the same group by key.

Grouping the documents by the day and the year of the date field, the following operation uses the $push accumulator to compute the list of items and quantities sold for each group:

- db.sales.aggregate([{ $group: { _id: { day: { $dayOfYear: "$date"}, year:
- { $year: "$date" } }, itemsSold: { $push: { item: "$item", quantity: "$quantity" } } } }])

$addToSet

Returns an array of all *unique* values that results from applying an expression to each document in a group of documents that share the same group by key. Order of the elements in the output array is unspecified.

If the value of the expression is an array, $addToSet appends the whole array as a *single* element.

If the value of the expression is a document, MongoDB determines that the document is a duplicate if another document in the array matches the to-be-added document exactly; i.e. the existing document has the exact same fields and values in the exact same order.

Example

- Grouping the documents by the day and the year of the date field, the following operation uses the $addToSet accumulator to compute the list of unique items sold for each group:

- db.sales.aggregate([{ $group: { _id: { day: { $dayOfYear: "$date"},
    - year: { $year: "$date" } }, itemsSold: { $addToSet: "$item" } } }])

31

Summary

$match, $unwind	$group, $project	$skip, $limit
$sort, $first	$last, $sum	$avg, $min, $max
	$push, $addToSet	

32