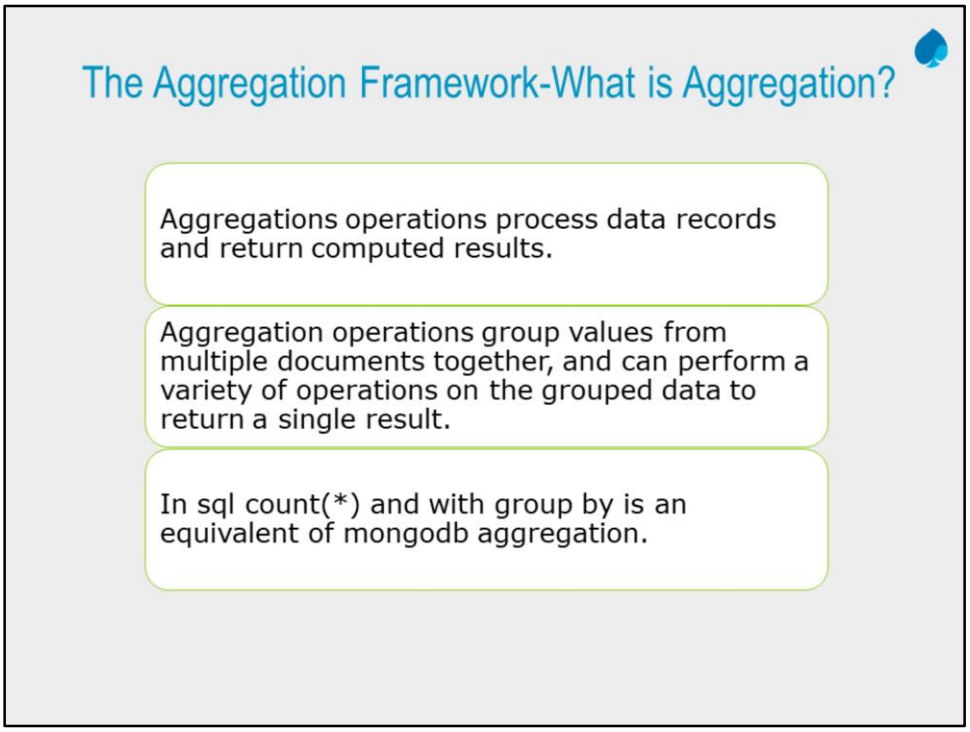

MongoDB – Aggregations
Lesson-04
Capgemini

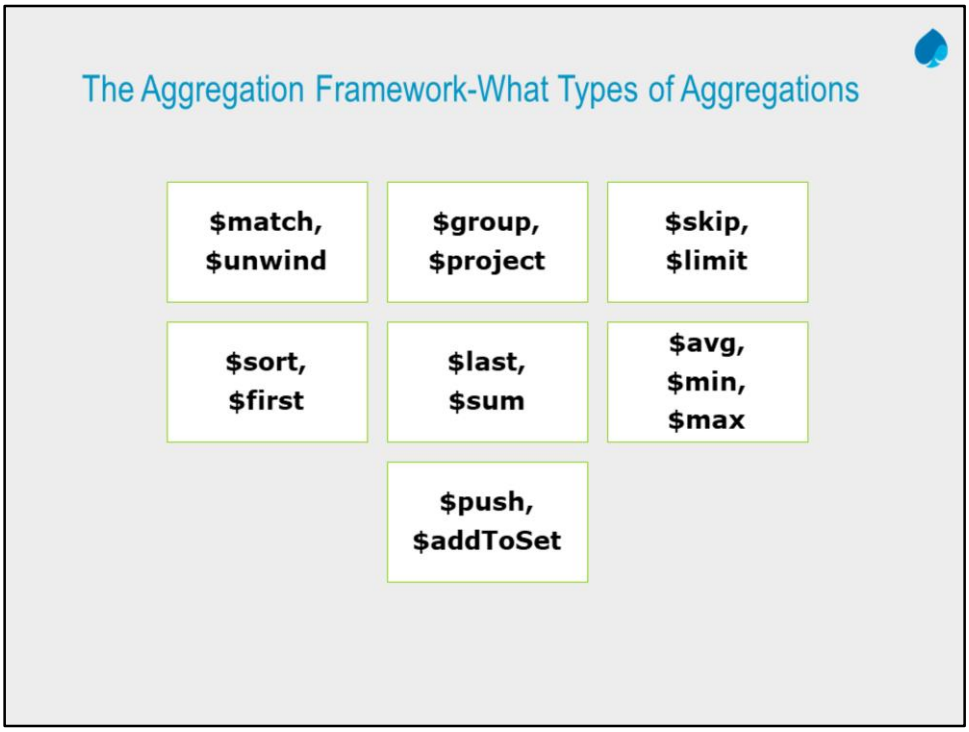


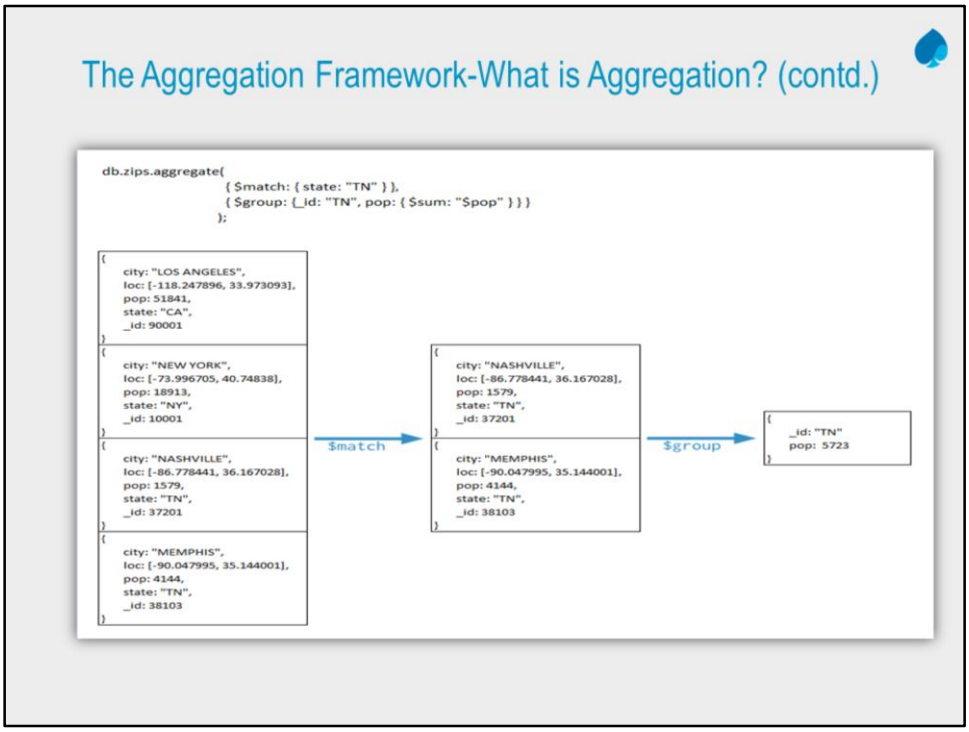
Create

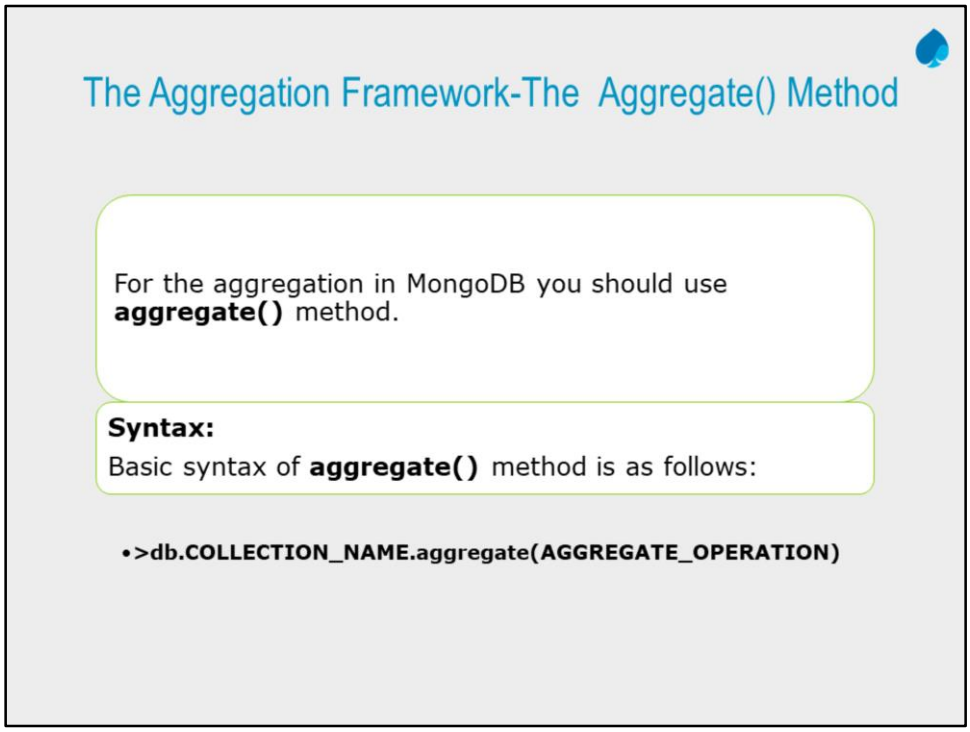
- The field name _id is reserved for use as a primary key; its value must be unique in the collection, is immutable, and may be of any type other than an array.
- The field names **cannot** start with the $ character.
- The field names **cannot** contain the . character.

Create with save

If the <document> argument does not contain the _id field or contains an _id field with a value not in the collection, the save() method performs an insert of the document.
Otherwise, the save() method performs an update.


The Aggregation Framework-What Types of Aggregations
$match, $unwind
$group, $project
$skip, $limit
$sort, $first
$last, $sum
$avg, $min, $max
$push, $addToSet


The Aggregation Framework-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
}
$match
{
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
}
$group
{
_id: "TN"
pop: 5723
}

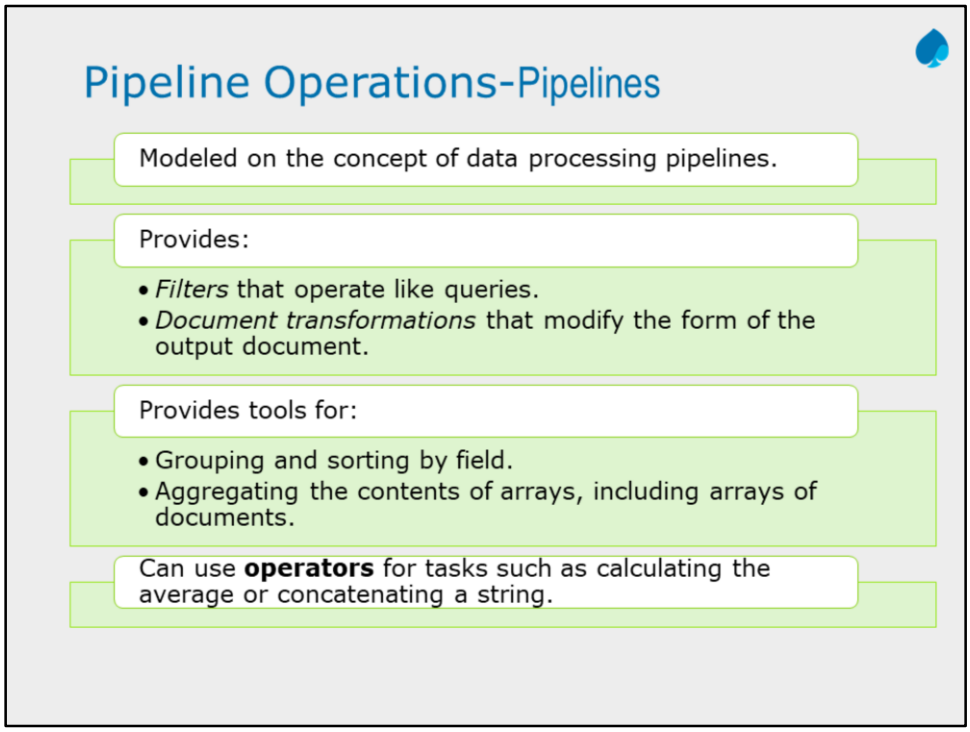

The Aggregation Framework-The Aggregate() Method
For the aggregation in MongoDB you should use **aggregate()** method.
Syntax:
Basic syntax of **aggregate()** method is as follows:
•>db.COLLECTION_NAME.aggregate(AGGREGATE_OPERATION)

Pipeline Operations-Pipeline Concept

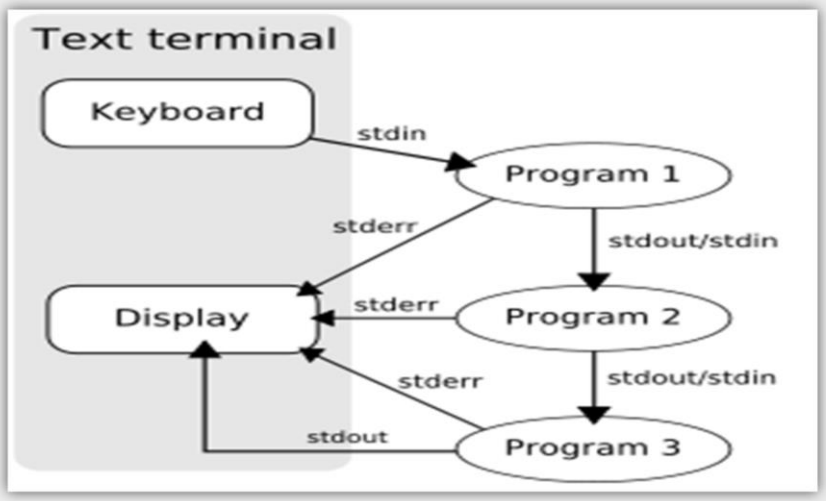
The aggregation framework is based on pipeline concept, just like Unix pipeline.

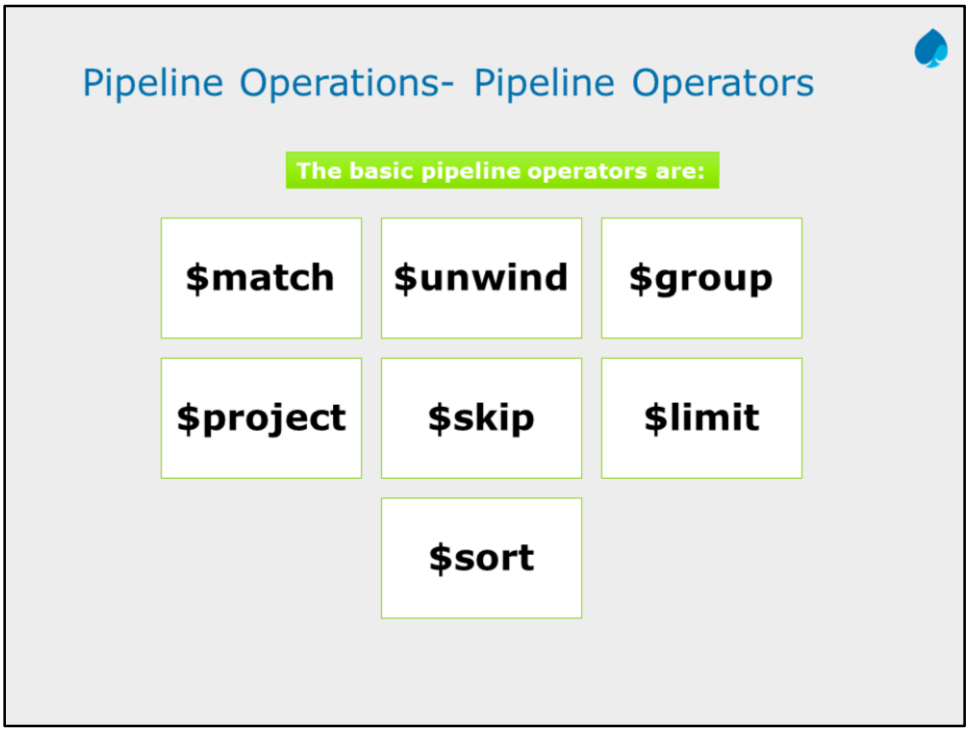
There can be N number of operators.

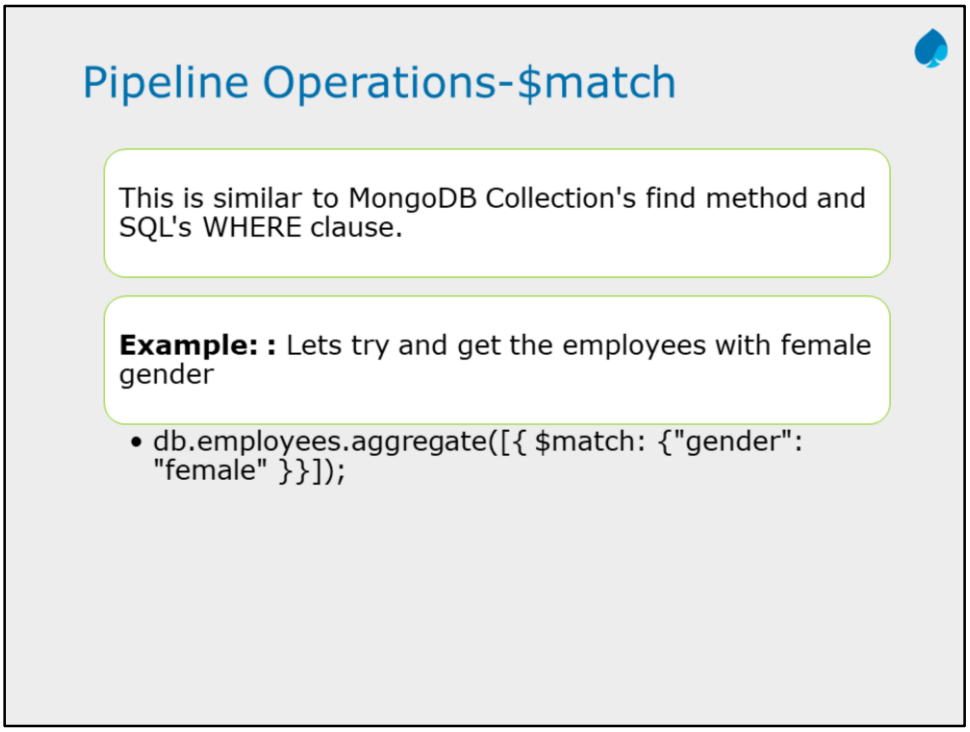
Output of first operator will be fed as input to the second operator. Output of second operator will be fed as input to the third operator and so on.


Pipeline Operations-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 Operations-Pipeline Flow




Pipeline Operations- Pipeline Operators
The basic pipeline operators are:
$match
$unwind
$group
$project
$skip
$limit
$sort



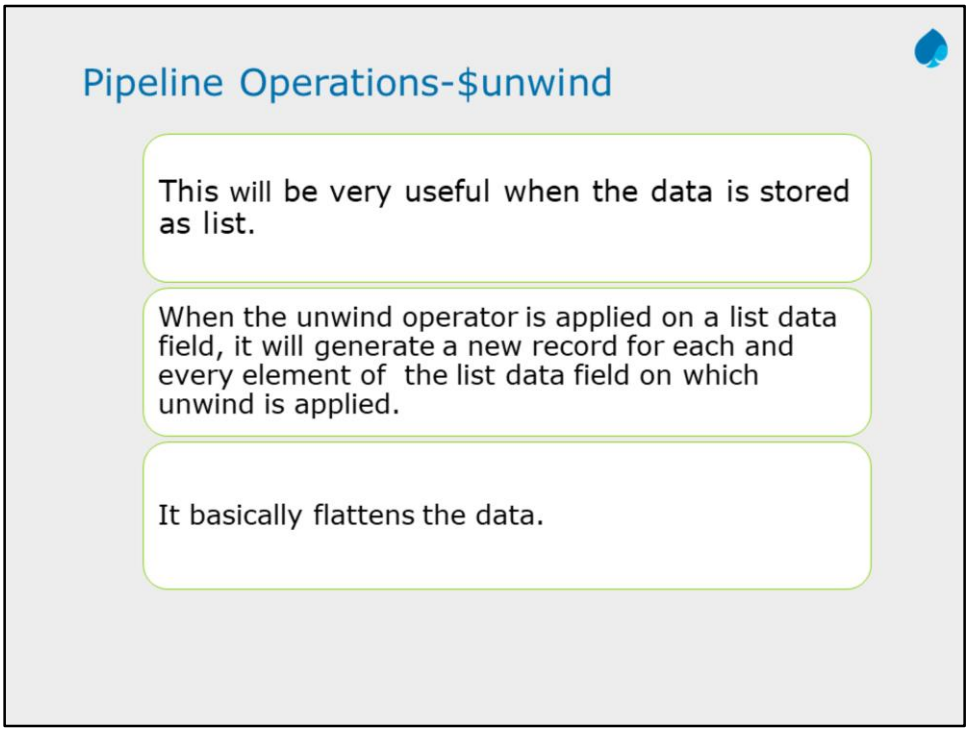
$match-
Filters the documents to pass only the documents that match the specified condition(s) to the next pipeline stage.
The $match stage has the following prototype form:

-{ $match: { <query> } }

$match takes a document that specifies the query conditions

-Find employees with female gender

> db.employees.aggregate([{ $match: {"gender": "female" }}]);



$unwind-

Deconstructs an array field from the input documents to output a document for *each* element.

Each output document is the input document with the value of the array field replaced by the element.

{ $unwind: <field path> }

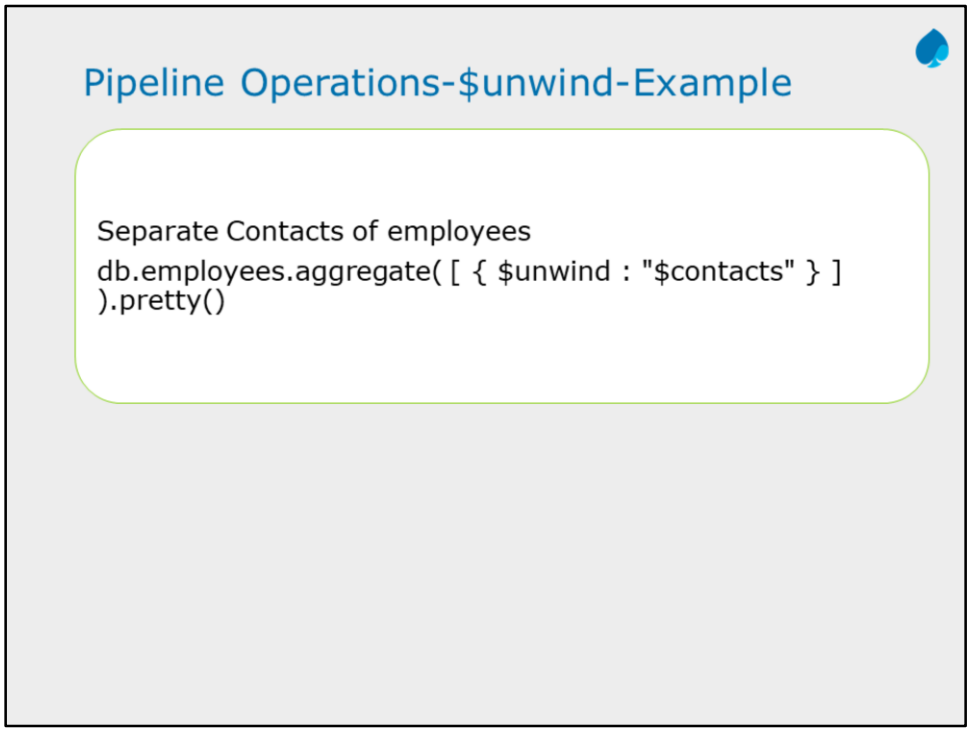
Points to remember:

If the value of a field is not an array, db.collection.aggregate() generates an error.

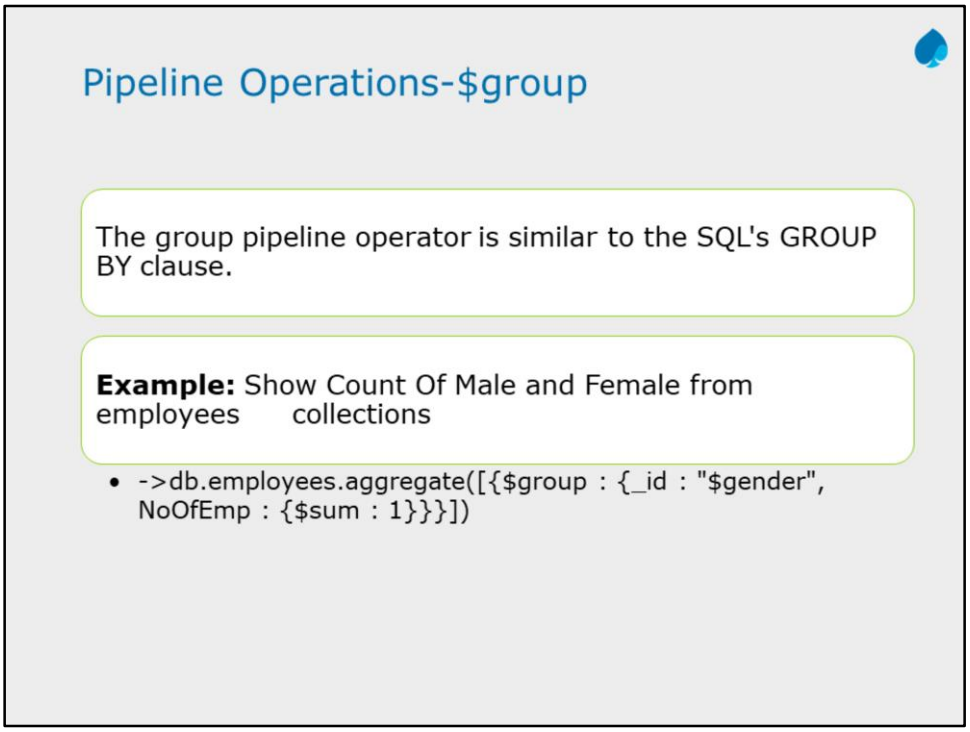
If the specified path for a field does not exist in an input document, the pipeline ignores the input document and displaying no output.

If the array is empty in an input document, the pipeline ignores the input document and displaying no output.

db.employees.aggregate([{ $unwind : "$contacts" }]).pretty()




```
Separate Contacts of employees
db.employees.aggregate( [ { $unwind : "$contacts" } ] ).pretty()
------------------OutPut-----------------------------------------------
{
        "_id" : 6,
        "firstname" : "Uma",
        "lastname" : "P",
        "deptinfo" : {
                "_id" : 30,
                "deptname" : "java"
        },
        "mgrcode" : 8,
        "gender" : "female",
        "age" : 53,
        "salary" : 300000,
        "address" : {
                "street" : "MgRoad",
                "city" : "Mysoor",
                "state" : "KA"
        },
        "contacts" : "9430087878",
        "annlGrossSal" : [
                {
                        "year" : 2001,
                        "gross" : "900000
                },
                {
                        "year" : 2002,
                        "gross" : "300000
                },
                {
                        "year" : 2003,
                        "gross" : "200000
                }
        ]
}
{
        "_id" : 6,
        "firstname" : "Uma",
        "lastname" : "P",
        "deptinfo" : {
                "_id" : 30,
                "deptname" : "java"
        },
        "mgrcode" : 8,
        "gender" : "female",
        "age" : 53,
        "salary" : 300000,
        "address" : {
                "street" : "MgRoad",
                "city" : "Mysoor",
                "state" : "KA"
        },
        "contacts" : "011-888888",
        "annlGrossSal" : [
                {
                        "year" : 2001,
                        "gross" : "900000
                },
                {
                        "year" : 2002,
                        "gross" : "300000
                },
                {
                        "year" : 2003,
                        "gross" : "200000
                }
        ]
}
```



Grouping Documents
Once we've filtered out the documents we don't want, we can start grouping together the ones that
we do into useful subsets. We can also use groups to perform operations across a common field in all documents,
such as calculating the sum of a set of transactions and counting documents.

-Show Count Of Male and Female from employees
->db.employees.aggregate([{$group : {_id : "$gender", NoOfEmp : {$sum : 1}}}])

-Show Count Of Only Female from employees collections

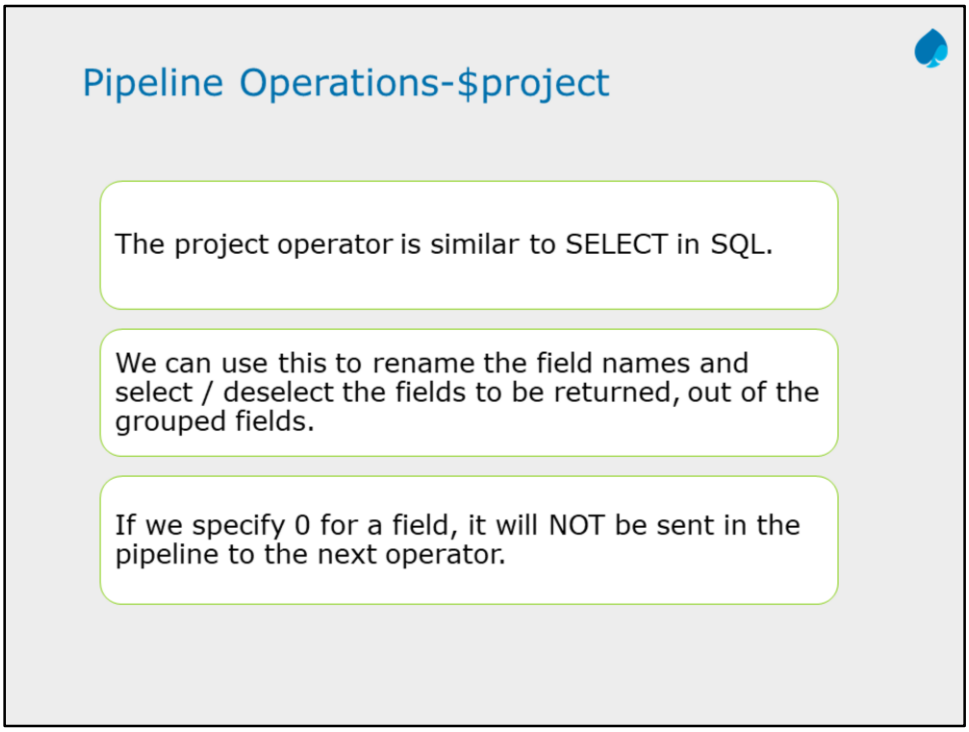
db.employees.aggregate([{$match:{gender:"female"}},{$group: {_id: null,count:{$sum:1}}}])

Pipeline Operations-$group (contd.)

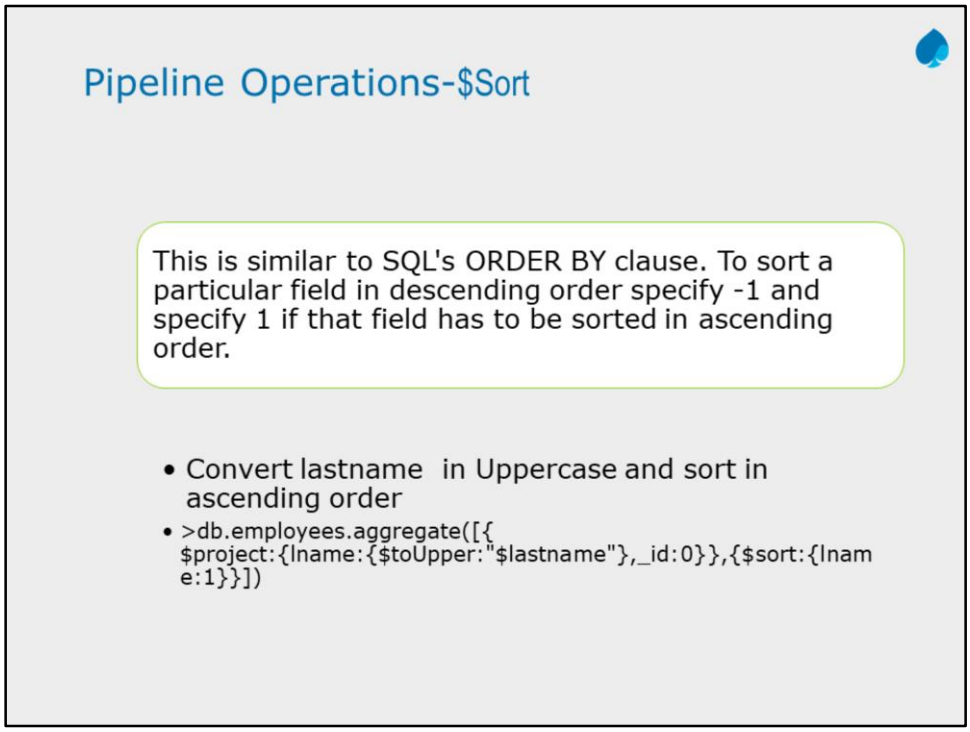
In this aggregation example, we have specified an _id element .

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

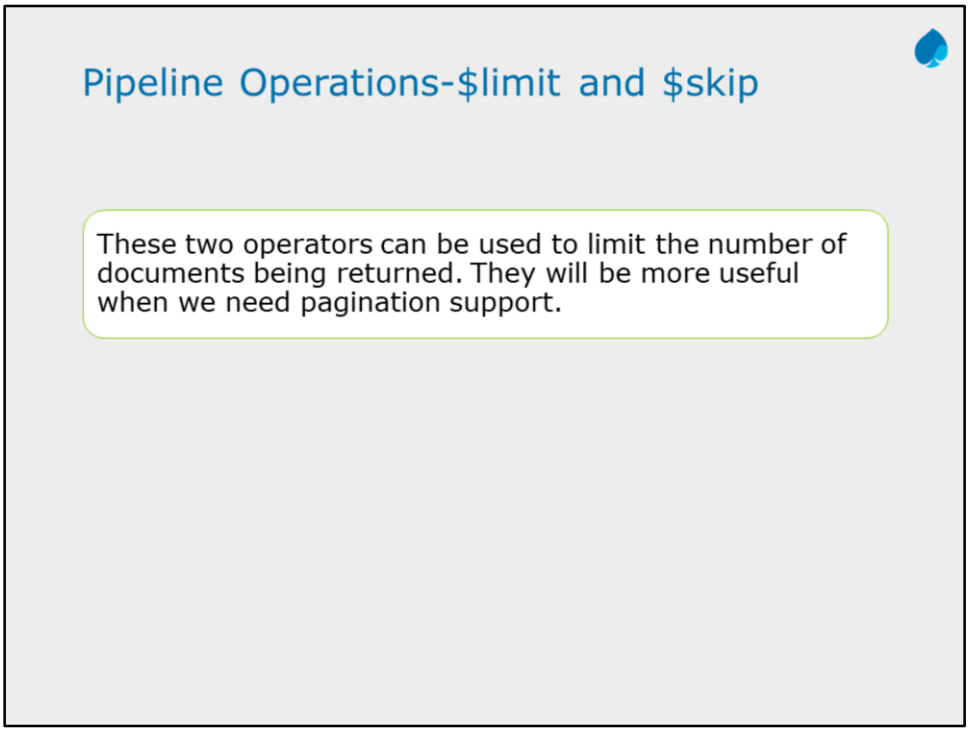
The NoOfEmp uses an aggregation function **$sum**, which basically counts up both gender and returns the sum.



$project − Used to select some specific fields from a collection.
The $project function in MongoDB passes along the documents with only the specified fields to the next stage in the pipeline. This may be the existing fields from the input documents or newly computed fields.
Syntax:
{ $project: { <specifications>
The specification for $project command contain the inclusion of fields, the suppression of the _id field, the addition of new fields, and the resetting the values of existing fields.
Parameters:
Specification Description<field>: <1 or true>Specify the inclusion of a field. _id: <0 or false>Specify the suppression of the _id field.<field>: <expression>Add a new field or reset the value of an existing field.
Points to remember:
By default, the _id field is included in the output documents. To exclude the _id field from the output documents, you must explicitly specify the suppression of the _id field in $project.
A field that does not exist in the document are going to include, the $project ignores that field inclusion;
A new field can be added and value of an existing field can be reset by specifying the field name and set



-Convert lastname in Uppercase and sort in ascending order
>db.employees.aggregate([{
$project:{lname:{$toUpper:"$lastname"},_id:0}},{$sort:{lname:1}}])



$skip-

Skips over the specified number of documents that pass into the stage and

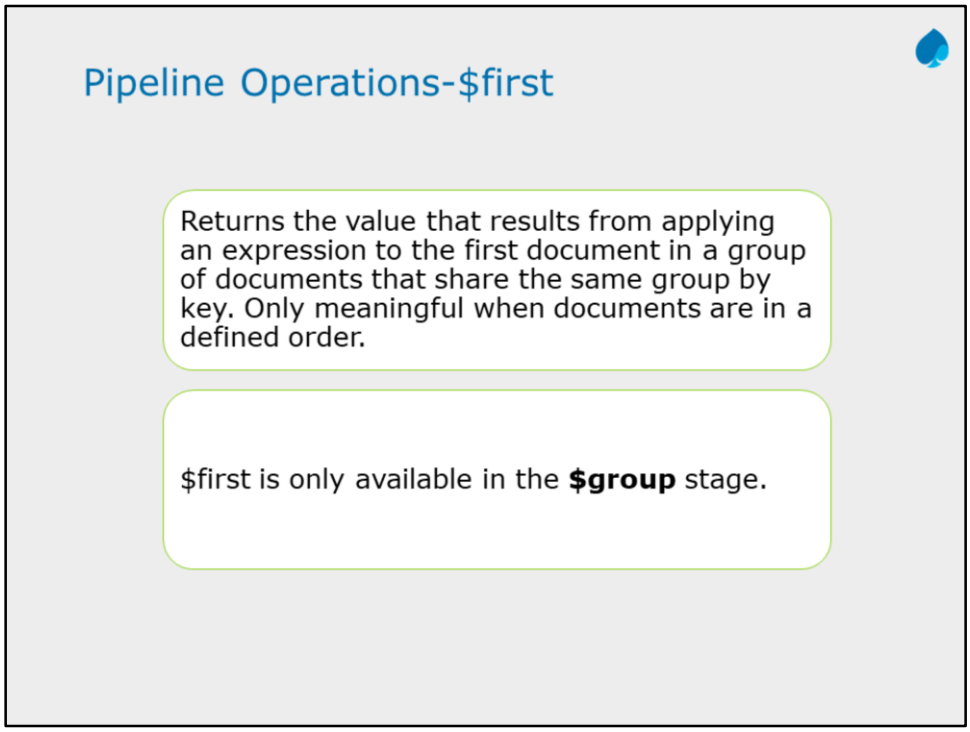
passes the remaining documents to the next stage in the pipeline.

The $skip stage has the following prototype form:
{ $skip: <positive integer> }

Ex.-db.employees.aggregate({ $skip : 5 });

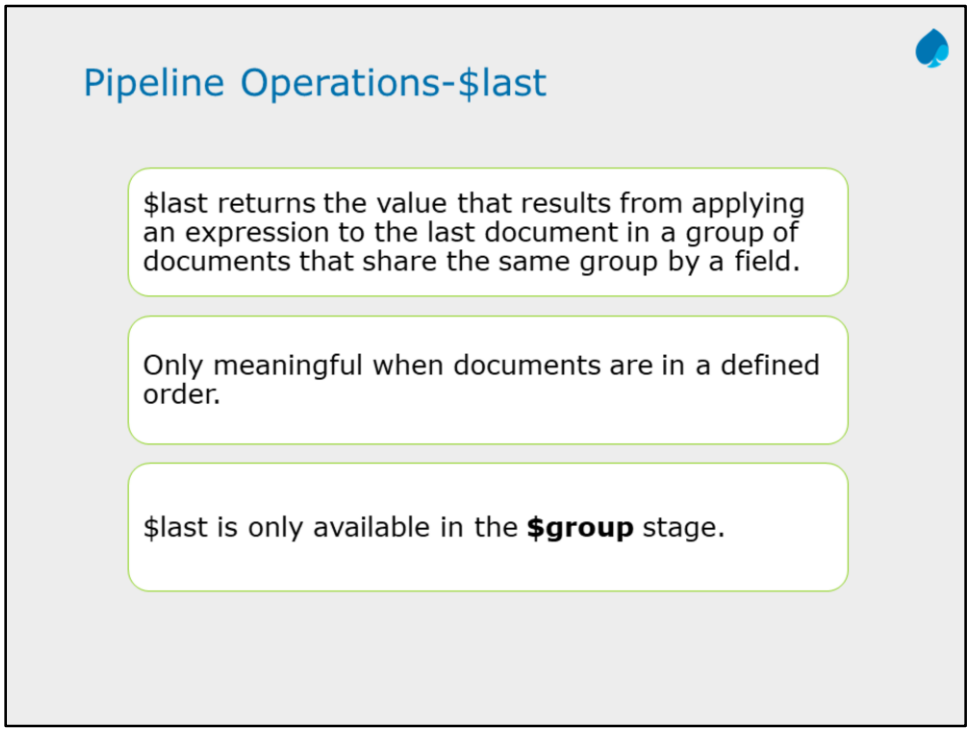
The limit() function in MongoDB is used to specify the maximum number of results to be returned.
Only one parameter is required for this function.to return the number of the desired result.
Sometimes it is required to return a certain number of results after a certain number of documents.
The skip() can do this job.

> db.employees.aggregate({$limit : 5 }).pretty()


Pipeline Operations-$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.

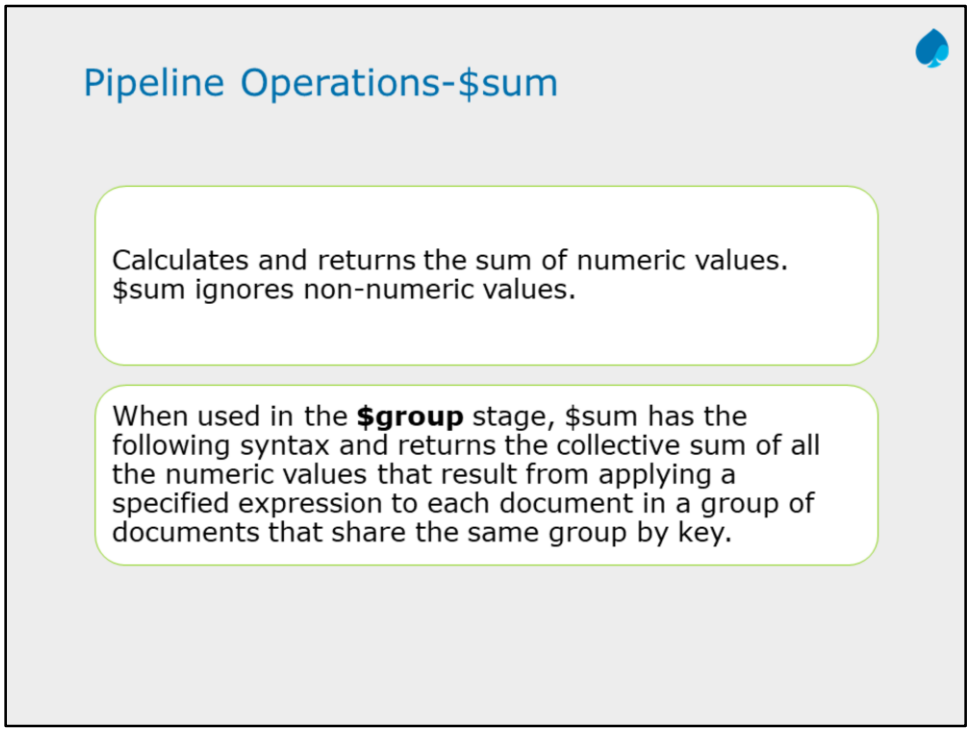
Pipeline Operations-$first- Example

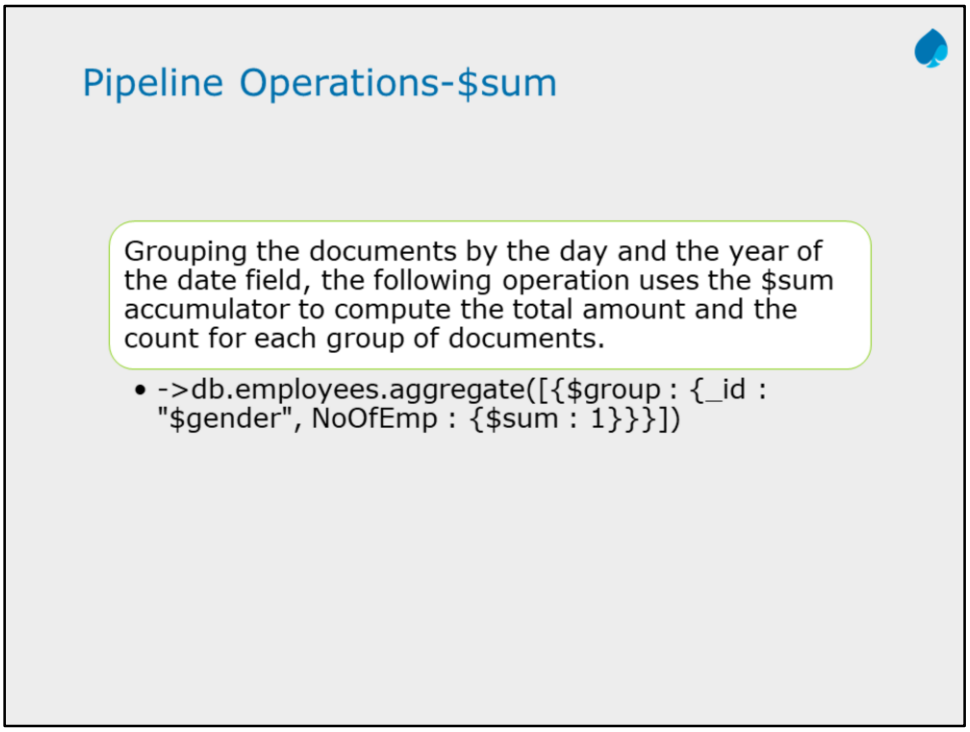
Grouping the documents by the item field, the following operation uses the $first accumulator to compute the first sales date for each item.


Pipeline Operations-$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.

Pipeline Operations-$last (contd.)

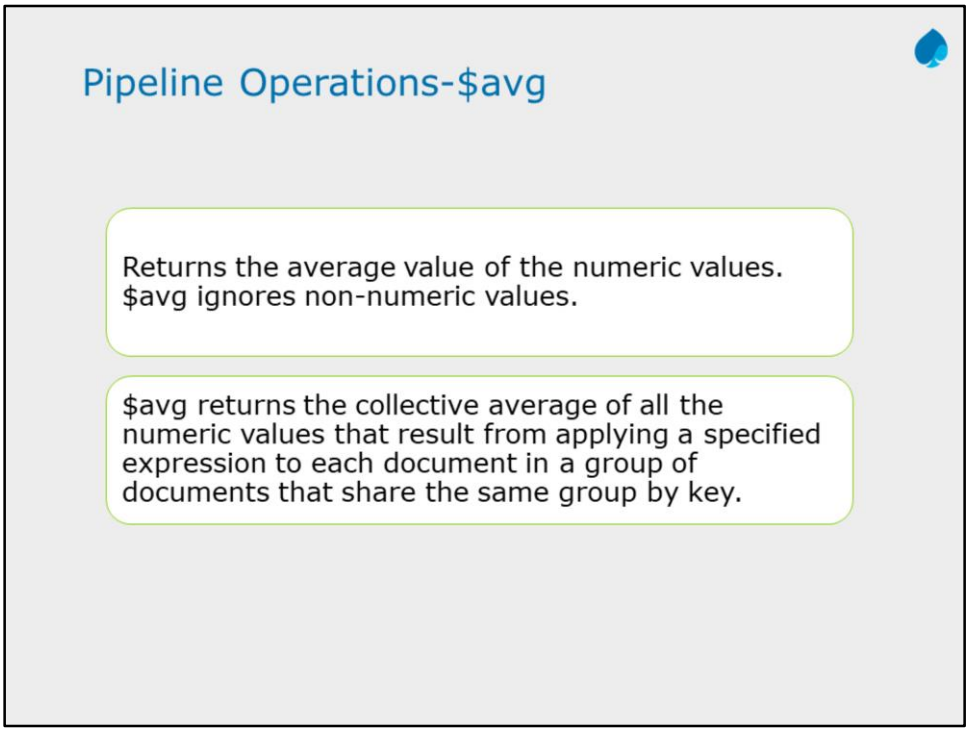
The following operation first sorts the documents by item and date, and then in the following **$group** stage, groups the now sorted documents by the item field and uses the $last accumulator to compute the last sales date for each item:


Pipeline Operations-$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.



Show Count Of Male and Female from employees

- >db.employees.aggregate([{$group : {_id : "$gender", NoOfEmp : {$sum : 1}}}])

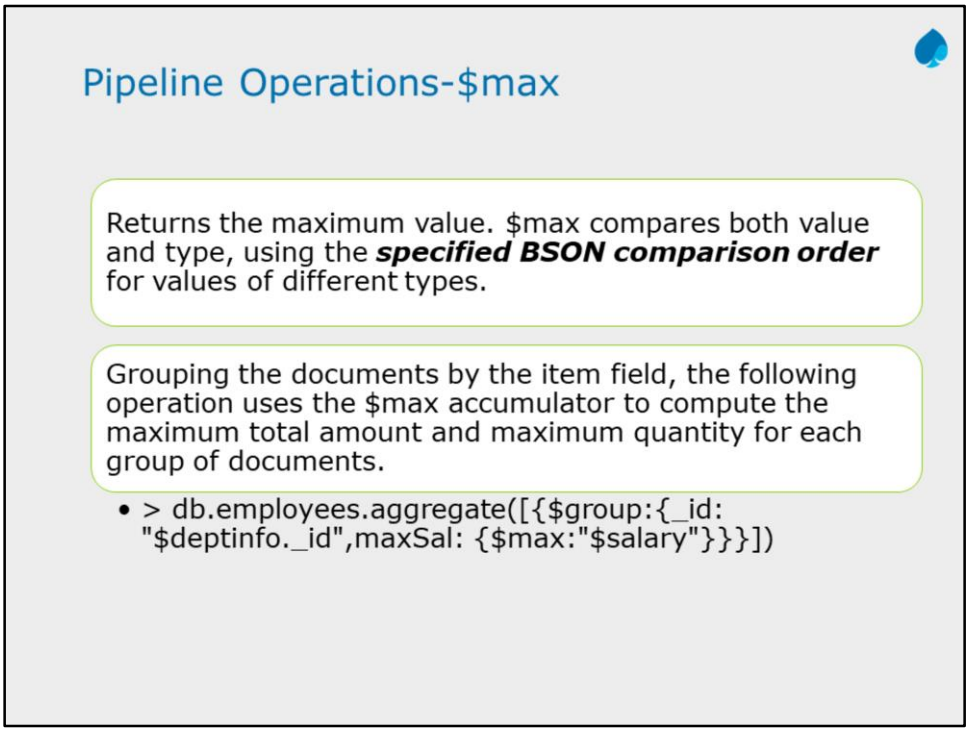


Q1)-Find Department wise average salary
Ans-db.employees.aggregate([{$group : {_id : "$deptinfo._id", AvgSal : {$avg : "$salary"}}}])

Pipeline Operations-$avg -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.employees.aggregate([{$group : {_id : "$deptinfo._id", AvgSal : {$avg : "$salary"}}}])



$max¶-

Returns the maximum value. $max compares both value and type, using the specified BSON comparison order for values of different types.

Changed in version 3.2: $max is available in the $group and $project stages.

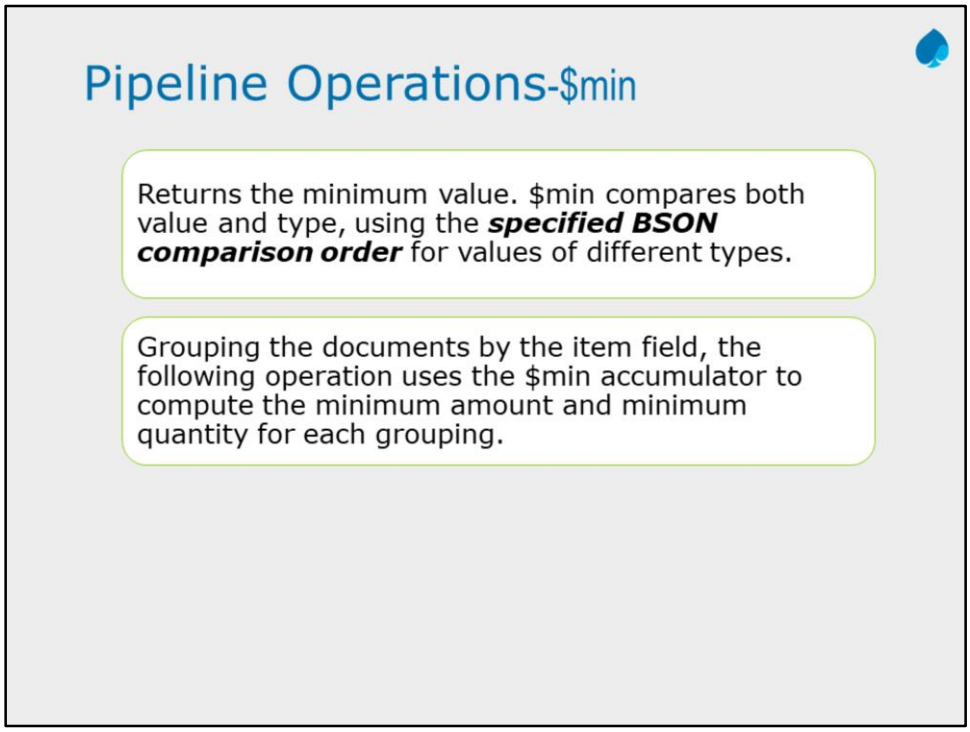
In previous versions of MongoDB, $max is available in the $group stage only.

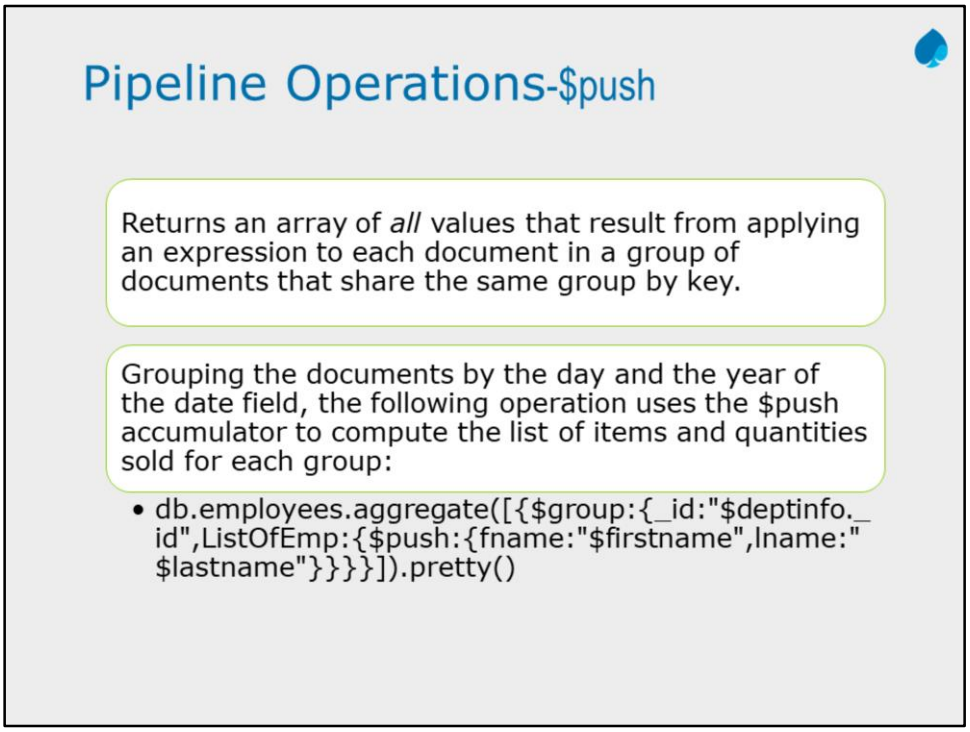
When used in the $group stage, $max has the following syntax and returns the maximum value that

results from applying an expression to each document in a group of documents that share the same group by key:

Ex-Find Department wise Maximum salary

```
> db.employees.aggregate([{$group:{_id: "$deptinfo._id",maxSal: {$max:"$salary"}}}])
{ "_id" : null, "maxSal" : 1200000 }
{ "_id" : 20, "maxSal" : 47000 }
{ "_id" : 40, "maxSal" : 20000 }
{ "_id" : 30, "maxSal" : 300000 }
>
```


Pipeline Operations-$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.



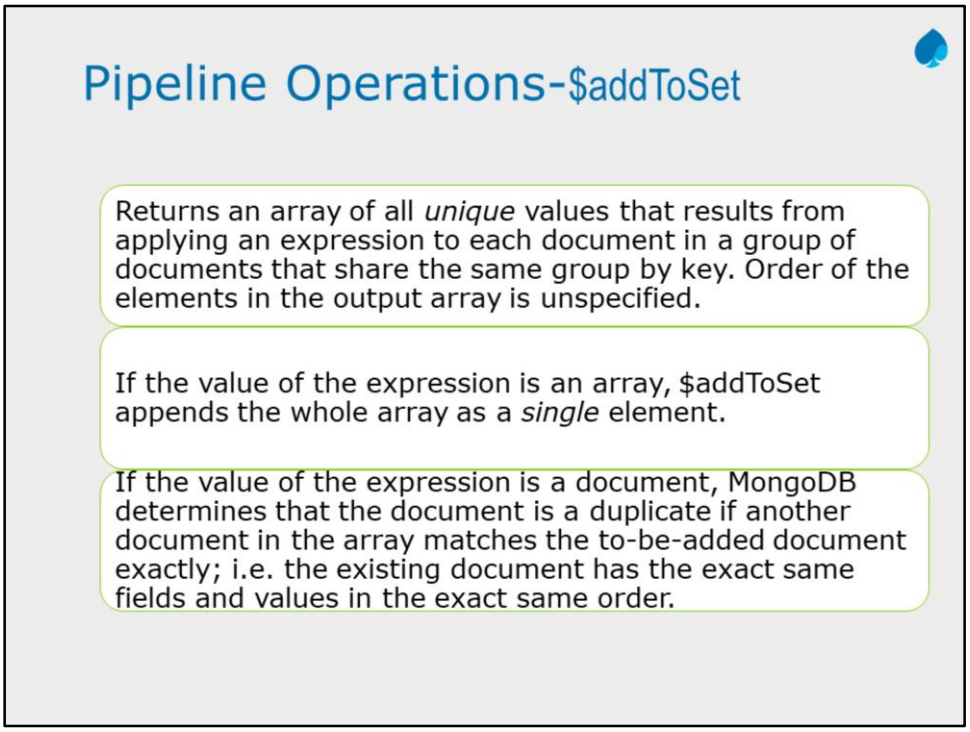
$push -

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.
$push is only available in the $group stage.
$push has the following syntax:
{ $push: <expression> }
For more information on expressions, see Expressions

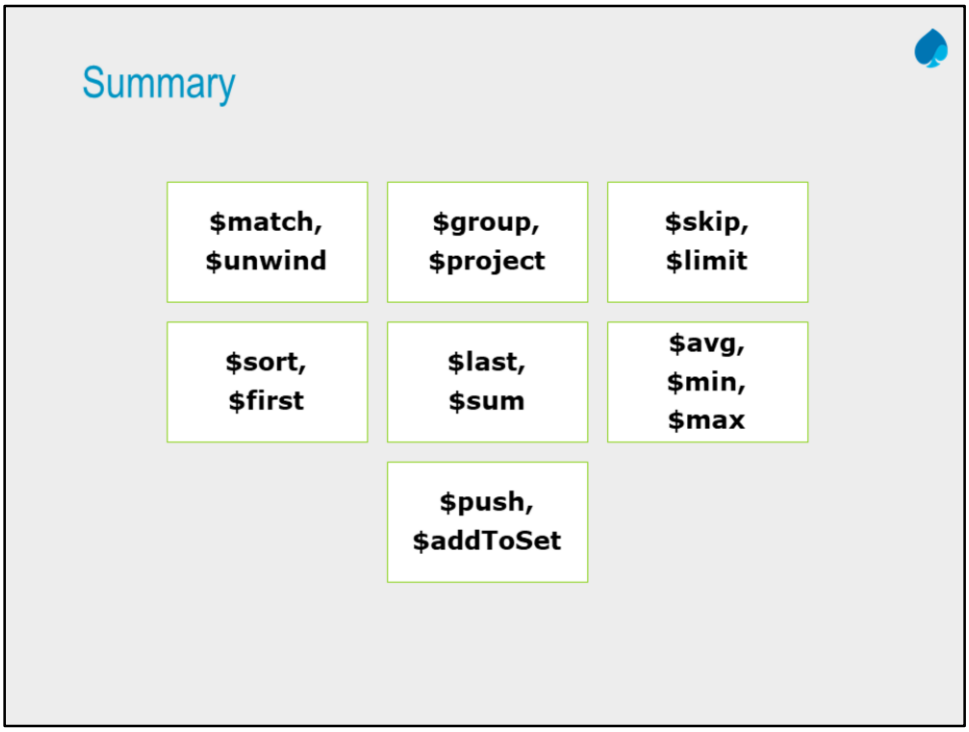
Print Depart wise first and last name

-

>db.employees.aggregate([{$group:{_id:"$deptinfo._id",ListOfEmp:{$push:{fname:"$firstname",lname:"$lastname"}}}}])



$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.
$addToSet is only available in the $group stage.


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