



MONGODB AGGREGATES:

Aggregation operations in MongoDB process data records and return computed results. These operations group values from multiple documents together and can perform various kinds of transformations on the data. The aggregation framework is powerful and flexible, allowing for complex data processing and transformation.

MongoDB Aggregation Operators:

- **\$sum:**

Description: Computes the sum of numeric values. Can also count the number of documents when used with a constant value of 1.

- **\$avg:**

Description: Computes the average of numeric values.

- **\$minL:**

Description: Returns the minimum value.

- **\$max:**

Description: Returns the maximum value.

- **\$push:**

Description: Adds values to an array.

- **\$addToSet**

Description: Adds unique values to an array.

- **\$first**

Description: Returns the first value in a group.

- **\$last:**

Description: Returns the last value in a group.

These operators are essential for performing complex aggregations and data transformations in MongoDB, enabling you to analyze and manipulate data effectively.

\$SUM OPERATION IN MONGODB

The \$sum operator in MongoDB is used to calculate the sum of numeric values. It is often used within the aggregation framework.

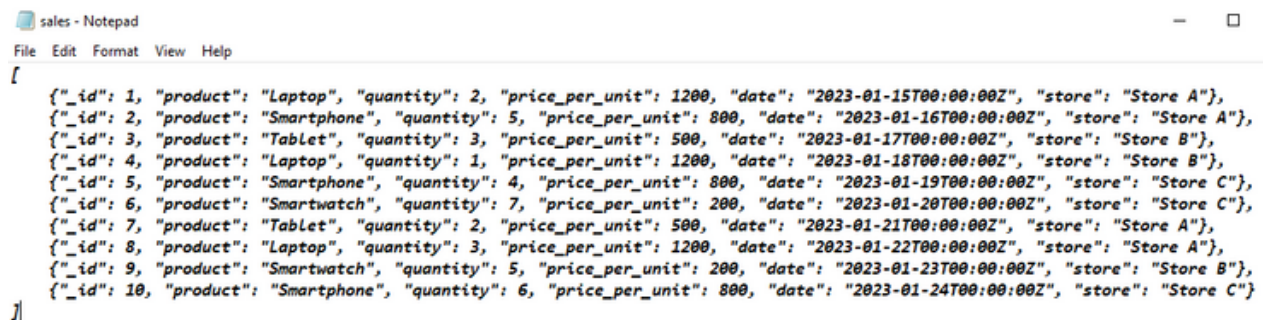
Syntax:

```
{
  $group: {
    _id: <expression>, // Field by which to group documents
    total: { $sum: <expression> } // Calculate the sum
  }
}
```

- <expression>: The field or computed value to sum.

Example Scenario:

Consider a collection sales with the following documents:



```
sales - Notepad
File Edit Format View Help

[
  { "_id": 1, "product": "Laptop", "quantity": 2, "price_per_unit": 1200, "date": "2023-01-15T00:00:00Z", "store": "Store A"},
  { "_id": 2, "product": "Smartphone", "quantity": 5, "price_per_unit": 800, "date": "2023-01-16T00:00:00Z", "store": "Store A"},
  { "_id": 3, "product": "Tablet", "quantity": 3, "price_per_unit": 500, "date": "2023-01-17T00:00:00Z", "store": "Store B"},
  { "_id": 4, "product": "Laptop", "quantity": 1, "price_per_unit": 1200, "date": "2023-01-18T00:00:00Z", "store": "Store B"},
  { "_id": 5, "product": "Smartphone", "quantity": 4, "price_per_unit": 800, "date": "2023-01-19T00:00:00Z", "store": "Store C"},
  { "_id": 6, "product": "Smartwatch", "quantity": 7, "price_per_unit": 200, "date": "2023-01-20T00:00:00Z", "store": "Store C"},
  { "_id": 7, "product": "Tablet", "quantity": 2, "price_per_unit": 500, "date": "2023-01-21T00:00:00Z", "store": "Store A"},
  { "_id": 8, "product": "Laptop", "quantity": 3, "price_per_unit": 1200, "date": "2023-01-22T00:00:00Z", "store": "Store A"},
  { "_id": 9, "product": "Smartwatch", "quantity": 5, "price_per_unit": 200, "date": "2023-01-23T00:00:00Z", "store": "Store B"},
  { "_id": 10, "product": "Smartphone", "quantity": 6, "price_per_unit": 800, "date": "2023-01-24T00:00:00Z", "store": "Store C"}
]
```

To calculate the total quantity for each product:

```
db.orders.aggregate([
  {
    $group: {
      _id: "$product",
      totalQuantity: { $sum: "$quantity" }
    }
  }
])
```

Explanation:

1. \$group Stage: This stage groups the documents by the product field. The _id field in the \$group stage specifies the field by which to group the documents.

2.\$sum Operator: Within the \$group stage, the \$sum operator is used to calculate the total quantity for each product.

Result:

The query will produce a result:

```
test> use db
switched to db db
db> show collections
candidates
locations
movies
sales
students
students_permission
db> db.sales.aggregate([
... {
... $group:{
... _id:"$product",
... totalQuantity:{$sum:"$quantity"}
... }
... }
... ])
[
  { _id: 'Smartwatch', totalQuantity: 12 },
  { _id: 'Tablet', totalQuantity: 5 },
  { _id: 'Laptop', totalQuantity: 6 },
  { _id: 'Smartphone', totalQuantity: 15 }
]
```

Breakdown of the Result

- Laptop:
 - Quantity from document 1: 2
 - Quantity from document 4: 1
 - Quantity from document 8: 3
 - Total Quantity: $2 + 1 + 3 = 6$
- Smartphone:
 - Quantity from document 2: 5
 - Quantity from document 5: 4
 - Quantity from document 10: 6
 - Total Quantity: $5 + 4 + 6 = 15$
- Tablet:
 - Quantity from document 3: 3
 - Quantity from document 7: 2
 - Total Quantity: $3 + 2 = 5$
- Smartwatch:
 - Quantity from document 6: 7
 - Quantity from document 9: 5
 - Total Quantity: $7 + 5 = 12$

Calculate Total Revenue for Each Product:

To calculate the total revenue for each product, we will multiply the quantity by the price_per_unit and then sum these values for each product.

```
db.sales.aggregate([
  {
    $group: {
      _id: "$product",
      totalRevenue: { $sum: { $multiply: ["$quantity", "$price_per_unit"] } }
    }
  }
])
```

Explanation:

1. \$group Stage: Groups the documents by the product field. The _id field in the \$group stage specifies the field by which to group the documents.
2. \$sum Operator: Used within the \$group stage to sum the total revenue for each product.
3. \$multiply Operator: Multiplies quantity by price_per_unit for each document to calculate the revenue for that sale.

Result:

The query will produce a result:

```
db> db.sales.aggregate([
... {
...   $group:{
...     _id:"$product",
...     totalRevenue:{$sum:{$multiply:["$quantity","$price_per_unit"]}}
...   }
... }
... ])
[
  { _id: 'Laptop', totalRevenue: 7200 },
  { _id: 'Smartwatch', totalRevenue: 2400 },
  { _id: 'Smartphone', totalRevenue: 12000 },
  { _id: 'Tablet', totalRevenue: 2500 }
]
db> _
```

Breakdown of the Result

- Laptop:
 - Revenue from document 1: $2 * 1200 = 2400$
 - Revenue from document 4: $1 * 1200 = 1200$
 - Revenue from document 8: $3 * 1200 = 3600$
 - Total Revenue: $2400 + 1200 + 3600 = 7200$

- Smartphone:
 - Revenue from document 2: $5 * 800 = 4000$
 - Revenue from document 5: $4 * 800 = 3200$
 - Revenue from document 10: $6 * 800 = 4800$
 - Total Revenue: $4000 + 3200 + 4800 = 12000$
- Tablet:
 - Revenue from document 3: $3 * 500 = 1500$
 - Revenue from document 7: $2 * 500 = 1000$
 - Total Revenue: $1500 + 1000 = 2500$
- Smartwatch:
 - Revenue from document 6: $7 * 200 = 1400$
 - Revenue from document 9: $5 * 200 = 1000$
 - Total Revenue: $1400 + 1000 = 2400$

\$AVG OPERATION IN MONGODB

The \$avg operator in MongoDB is used within the aggregation framework to calculate the average of numeric values. This operator is typically used within the \$group stage to compute the average value of a specified field or expression across grouped documents.

Syntax:

```
{
  $group: {
    _id: <expression>, // Field by which to group documents
    averageValue: { $avg: <expression> } // Calculate the average
  }
}
```

- <expression>: The field or computed value to sum.

Example Scenario:

Consider a collection sales to perform this operation:

To calculate the average quantity for each product:

```
db.sales.aggregate([
  {
    $group: {
      _id: "$product",
      averageQuantity: { $avg: "$quantity" }
    }
  }
])
```

Explanation:

1. `db.sales.aggregate([])`:
 - This function is used to perform aggregation operations on the sales collection.
2. `$group`:
 - This is an aggregation stage that groups the documents by a specified field. In this case, it groups by the product field.
3. `_id: "$product"`:
 - The `_id` field in the `$group` stage is used to specify the field by which the documents should be grouped. Here, the documents are grouped by the product field.
4. `averageQuantity: { $avg: "$quantity" }`:
 - This creates a new field `averageQuantity` in the output. The `$avg` operator is used to calculate the average value of the `quantity` field for each group of documents that have the same product value.

```
db> db.sales.aggregate([
... {
...   $group:{
...     _id:"$product",
...     averageQuantity:{ $avg:"$quantity"}
...   }
... }
... ])
db>
{ _id: 'Tablet', averageQuantity: 2.5 },
{ _id: 'Smartwatch', averageQuantity: 6 },
{ _id: 'Smartphone', averageQuantity: 5 },
{ _id: 'Laptop', averageQuantity: 2 }
```

Interpretation of the Output

- Laptop: The average quantity sold is 2.
- Smartphone: The average quantity sold is 5.
- Tablet: The average quantity sold is 2.5.
- Smartwatch: The average quantity sold is 6.

This query helps in understanding the average number of units sold for each type of product in the 'sales' collection.

\$MIN AND \$MAX AGGREGATION OPERATORS:

In MongoDB, the `$min` and `$max` aggregation operators are used to calculate the minimum and maximum values of a specified field across grouped documents. These operators are typically used within the `$group` stage of the aggregation pipeline.

- `$min`: Returns the minimum value of the specified field from the grouped documents.
- `$max`: Returns the maximum value of the specified field from the grouped documents.

Syntax:

```
db.collection.aggregate([
  {
    $group: {
      _id: "$fieldToGroupBy",
      minField: { $min: "$fieldToCalculateMin" },
      maxField: { $max: "$fieldToCalculateMax" }
    }
  }
])
```

Consider the dataset in the sales collection:

To find the minimum and maximum quantity of each product, we use the following aggregation pipeline:

```
db.sales.aggregate([
  {
    $group: {
      _id: "$product",
      minQuantity: { $min: "$quantity" },
      maxQuantity: { $max: "$quantity" }
    }
  }
])
```

Explanation of the Code

1. `db.sales.aggregate([])`:
 - Initiates an aggregation operation on the sales collection.
2. `$group`:
 - This stage groups documents by a specified field. Here, we are grouping documents by the product field.
3. `_id: "$product"`:
 - Specifies the field to group by, which is the product field in this case.
4. `minQuantity: { $min: "$quantity" }`:
 - Calculates the minimum value of the quantity field for each group of documents that have the same product.
5. `maxQuantity: { $max: "$quantity" }`:
 - Calculates the maximum value of the quantity field for each group of documents that have the same product.

The query output will be:

```
db> db.sales.aggregate([
... {
...   $group:{
...     _id:"$product",
...     minQuantity:{$min:"$quantity"},
...     maxQuantity:{$max:"$quantity"}
...   }
... }
... ])
[
  { _id: 'Smartwatch', minQuantity: 5, maxQuantity: 7 },
  { _id: 'Laptop', minQuantity: 1, maxQuantity: 3 },
  { _id: 'Smartphone', minQuantity: 4, maxQuantity: 6 },
  { _id: 'Tablet', minQuantity: 2, maxQuantity: 3 }
]
db>
```

Interpretation of the Output:

- Laptop:
 - Minimum Quantity: 1
 - Maximum Quantity: 3
- Smartphone:
 - Minimum Quantity: 4
 - Maximum Quantity: 6
- Tablet:
 - Minimum Quantity: 2
 - Maximum Quantity: 3
- Smartwatch:
 - Minimum Quantity: 5
 - Maximum Quantity: 7

This aggregation query groups the documents by product and calculates both the minimum and maximum quantity for each product in the sales collection, providing valuable insights into the range of quantities sold for each product.

\$FIRST AND \$LAST OPERATORS :

The \$first and \$last aggregation operators are used to get the first and last document in each group of documents, respectively. These operators are commonly used within the \$group stage of the aggregation pipeline.

Syntax:

```
db.collection.aggregate([
{
  $group: {
    _id: "$fieldToGroupBy",
    firstDocumentField: { $first: "$fieldToGetFirstValue" },
    lastDocumentField: { $last: "$fieldToGetLastValue" }
  }
}]
```


Using the same sales dataset:

To find the first and last quantities for each product:

```
db.sales.aggregate([
  {
    $group: {
      _id: "$product",
      firstQuantity: { $first: "$quantity" },
      lastQuantity: { $last: "$quantity" }
    }
  }
])
```

```
db> db.sales.aggregate([
... {
... $group:{
...   _id:"$product",
...   firstQuantity:{ $first:"$quantity"},
...   lastQuantity:{ $last:"$quantity"}
... }
... }
... ])
[
  { _id: 'Smartwatch', firstQuantity: 7, lastQuantity: 5 },
  { _id: 'Laptop', firstQuantity: 2, lastQuantity: 3 },
  { _id: 'Smartphone', firstQuantity: 5, lastQuantity: 6 },
  { _id: 'Tablet', firstQuantity: 3, lastQuantity: 2 }
]
```

Explanation of the Code:

1. `db.sales.aggregate([])`:
 - Initiates an aggregation operation on the sales collection.
2. `$group`:
 - This stage groups documents by a specified field. Here, we are grouping documents by the product field.
3. `_id: "$product"`:
 - Specifies the field to group by, which is the product field in this case.
4. `firstQuantity: { $first: "$quantity" }`:
 - This operator retrieves the value of the quantity field from the first document in each group of documents that have the same product.
5. `lastQuantity: { $last: "$quantity" }`:
 - This operator retrieves the value of the quantity field from the last document in each group of documents that have the same product.

Interpretation of the Output:

- Laptop:
 - First Quantity: 2
 - Last Quantity: 3
- Smartphone:
 - First Quantity: 5
 - Last Quantity: 6
- Tablet:
 - First Quantity: 3
 - Last Quantity: 2
- Smartwatch:
 - First Quantity: 7
 - Last Quantity: 5

This aggregation query groups the documents by product and calculates both the first and last quantity for each product in the sales collection. This can be useful to understand the range of quantities over time for each product.

\$PUSH AND \$ADDTOSET OPERATORS IN MONGODB :

The \$push and \$addToSet aggregation operators are used to add elements to an array in the resulting documents of an aggregation pipeline. They are commonly used within the \$group stage to accumulate values from multiple documents into an array.

- \$push: Adds an element to an array. It includes all values, including duplicates.
- \$addToSet: Adds an element to a set (array) but only if the element is not already present, ensuring no duplicates.

Syntax:

- \$push operator:

```
db.collection.aggregate([
  {
    $group: {
      _id: "$fieldToGroupBy",
      arrayField: { $push: "$fieldToPush" }
    }
  }
])
```

Syntax:

- **\$addToSet operator:**

```
db.collection.aggregate([
  {
    $group: {
      _id: "$fieldToGroupBy",
      arrayField: { $addToSet: "$fieldToAddToSet" }
    }
  }
])
```

Using the same sales dataset: To gather all quantities of each product into an array using \$push:

```
db.sales.aggregate([
  {
    $group: {
      _id: "$product",
      quantities: { $push: "$quantity" }
    }
  }
])
```

- **Explanation of the Code:**

- db.sales.aggregate([]):
 - Initiates an aggregation operation on the sales collection.
- \$group:
 - This stage groups documents by a specified field. Here, we are grouping documents by the product field.
- _id: "\$product":
 - Specifies the field to group by, which is the product field in this case.
- uniqueQuantities: { \$addToSet: "\$quantity" }:
 - Adds the value of the quantity field from each document in the group to a set (array), ensuring no duplicates.
- quantities: { \$push: "\$quantity" }:
 - Adds the value of the quantity field from each document in the group to an array, including duplicates.

```

db> db.salesnew.aggregate([ { $group: { _id: "$product", uniqueQuantities: { $addToSet: "$quantity" } } } ] )
[
  { _id: 'Smartphone', uniqueQuantities: [ 5 ] },
  { _id: 'Smartwatch', uniqueQuantities: [ 7 ] },
  { _id: 'Tablet', uniqueQuantities: [ 3 ] },
  { _id: 'Laptop', uniqueQuantities: [ 2, 1 ] }
]
db> db.salesnew.aggregate([ { $group: { _id: "$product", quantites: { $push: "$quantity" } } } ] )
[
  { _id: 'Smartphone', quantites: [ 5, 5, 5 ] },
  { _id: 'Smartwatch', quantites: [ 7, 7 ] },
  { _id: 'Tablet', quantites: [ 3, 3 ] },
  { _id: 'Laptop', quantites: [ 2, 1, 2 ] }
]

```

Interpretation of the Output:

- Laptop:
 - quantities using \$push: [2, 1, 2]
 - uniqueQuantities using \$addToSet: [2, 1]
- Smartphone:
 - quantities using \$push: [5, 5, 5]
 - uniqueQuantities using \$addToSet: [5]
- Tablet:
 - quantities using \$push: [3, 3]
 - uniqueQuantities using \$addToSet: [3]
- Smartwatch:
 - quantities using \$push: [7, 7]
 - uniqueQuantities using \$addToSet: [7]

This aggregation query demonstrates how to accumulate values into an array using \$push and \$addToSet. The \$push operator includes all values, while \$addToSet includes only unique values, ensuring no duplicates. These examples illustrate the difference between the two operators clearly, especially when there are repeated values in the dataset.

By understanding and leveraging these operators, you can perform complex data transformations and aggregations with ease. The distinction between \$push and \$addToSet is crucial for effectively managing data that contains duplicates. Using the provided examples, the practical differences between these operators are clearly demonstrated, highlighting their respective uses and advantages in various aggregation scenarios.

In conclusion, mastering the use of \$push and \$addToSet within MongoDB's aggregation framework enhances your ability to perform detailed and sophisticated data analyses, making it an essential skill for working with large datasets and extracting meaningful insights.

