Aggregate Functions in MongoDB

- 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.
- Aggregation in MongoDB is nothing but an operation used to process the data that returns the computed results. Aggregation basically groups the data from multiple documents and operates in many ways on those grouped data in order to return one combined result. In sql count(*) and with group by is an equivalent of MongoDB aggregation.

- Aggregate function groups the records in a collection, and can be used to provide total number(sum), average, minimum, maximum etc out of the group selected.

Aggregate Functions in MongoDB

- **Example 1** : Find out How many documents are inside the customer Collection

db.customers.count({})

- **Example 2** : Find out How many male students are in the Customer Collection

db.student.count({Gender : "Male"})

Different expressions used by Aggregate function

Expression	Description
$sum	Summates the defined values from all the documents in a collection
$avg	Calculates the average values from all the documents in a collection
$min	Return the minimum of all values of documents in a collection
$max	Return the maximum of all values of documents in a collection

Gaining Insights with Sum, Min, Max, and Avg

Example 3 : $sum

Aggregation Imagine if we had male and female students in a recordBook collection and we want a total count on each of them. In order to get the sum of males and females, we could use the $group aggregate function.

```
db.recordBook.aggregate([
 { $group : {_id : "$gender", result: {$sum: 1}}
}
]);
```

Gaining Insights with Sum

Example 4 : $sum

```
db.transactions.insert(
 {
  id: "1",
  productId: "1",
  customerId: "1",
  amount: 20,
  transactionDate: ISODate("2017-01-23T15:25:56.314Z")
}
)
```

Gaining Insights with Sum

Example 4 : $sum

```
db.transactions.insert(
 {
 id: "2",
 productId: "2",
 customerId: "2",
 amount: 40,
 transactionDate: ISODate("2017-01-23T15:25:56.314Z")
}
)
```

Gaining Insights with Sum

Example 4 : $sum

```
db.transactions.aggregate([
 {
   $match: {
     transactionDate: {
       $gte: ISODate("2017-01-01T00:00:00.000Z"),
       $lt: ISODate("2017-01-31T23:59:59.000Z")
     }
   }
 }, {
   $group: {
     _id: null,
     total: {
       $sum: "$amount"
     }
   }
 }
]);
```

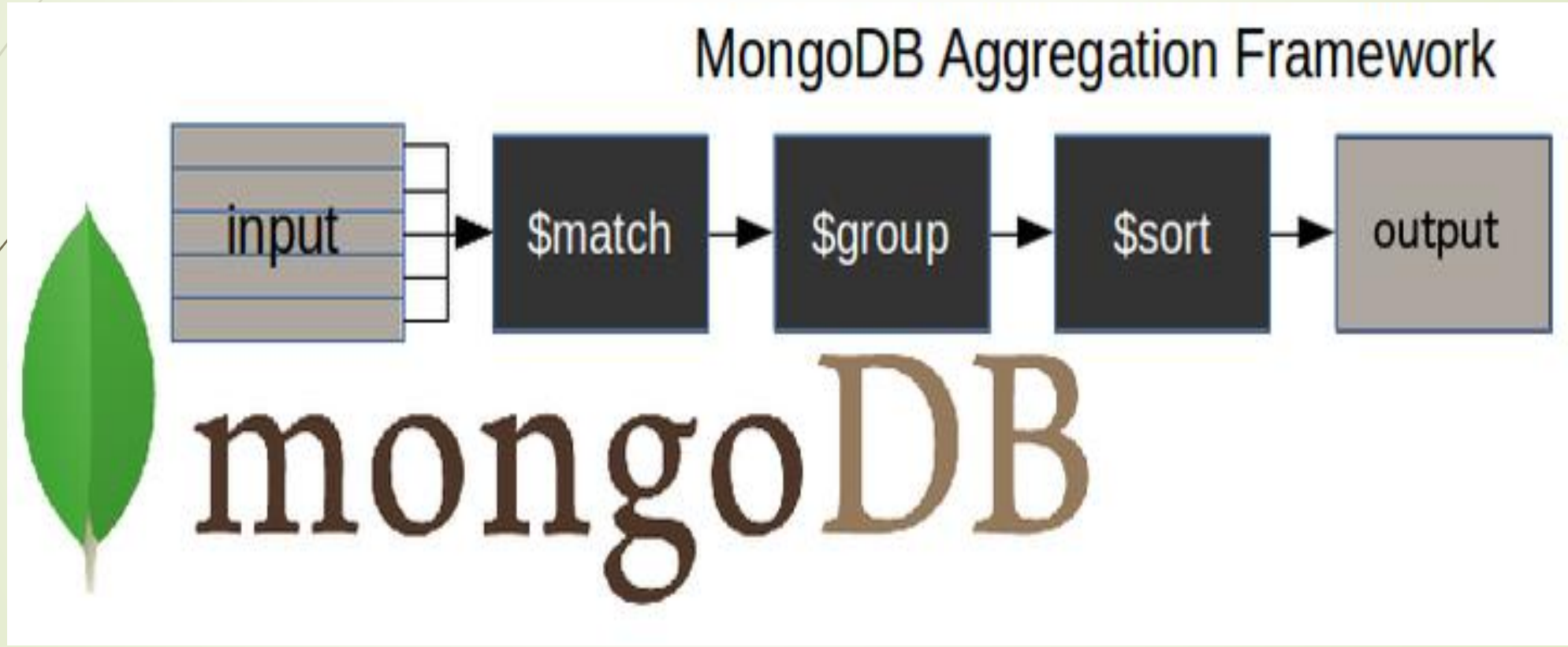
Gaining Insights with Sum

```
db.transactions.aggregate([
 {
   $match: {
     transactionDate: {
       $gte: ISODate("2017-01-01T00:00:00.000Z"),
       $lt: ISODate("2017-01-31T23:59:59.000Z")
     }
   }
 }, {
   $group: {
     _id: null,
     total: {
       $sum: "$amount"
     },
```

Gaining Insights with Sum

```
average_transaction_amount: {
      $avg: "$amount"
    },
    min_transaction_amount: {
      $min: "$amount"
    },
    max_transaction_amount: {
      $max: "$amount"
    }
  }
 }
]);
```

What is the MongoDB aggregation pipeline?



What is the MongoDB aggregation pipeline?

$match() stage – filters those documents we need to work with, those that fit our needs
$group() stage – does the aggregation job
$sort() stage – sorts the resulting documents the way we require (ascending or descending)

Aggregate Functions Examples

```
db.customers1.insert(
{
  id: "1",
  firstName: "Jane",
  lastName: "Doe",
  phoneNumber: "555-555-1212",
  city: "Beverly Hills",
  state: "CA",
  zip: 90210,
  email: "Jane.Doe@compose.io"
}
)
```

Aggregate Functions Examples

Example 1 : Matching Documents

db.customers1.aggregate**([**

{ $match: { "zip": 90210 }}

Definition :

The first stage of the pipeline is matching, and that allows us to filter out documents so that we're only manipulating the documents we care about. The matching expression looks and acts much like the MongoDB *find* function or a *SQL* **WHERE** clause.**]);**

This will return the array of customers that live in the 90210 zip code

Aggregate Functions Examples

Example 2 : Grouping Documents

Aggregation Imagine if we had male and female students in a recordBook collection and we want a total count on each of them. In order to get the sum of males and females, we could use the $group aggregate function.

```
db.recordBook.aggregate([
 { $group : {_id : "$gender", result: {$sum: 1}}
}
]);
```

Using sort

Consider the following restaurant collection:

```
db.restaurants.insertMany( [
  { "_id" : 1, "name" : "Central Park Cafe", "borough" : "Manhattan"},
  { "_id" : 2, "name" : "Rock A Feller Bar and Grill", "borough" : "Queens"},
  { "_id" : 3, "name" : "Empire State Pub", "borough" : "Brooklyn"},
  { "_id" : 4, "name" : "Stan's Pizzaria", "borough" : "Manhattan"},
  { "_id" : 5, "name" : "Jane's Deli", "borough" : "Brooklyn"},
] );
```

The following command uses the $sort stage to sort on the borough field:

```
db.restaurants.aggregate(
  [
    { $sort : { borough : 1 } }
  ]
)
```

Using sort

Consider the following restaurant collection:

```
db.restaurants.insertMany( [
  { "_id" : 1, "name" : "Central Park Cafe", "borough" : "Manhattan"},
  { "_id" : 2, "name" : "Rock A Feller Bar and Grill", "borough" : "Queens"},
  { "_id" : 3, "name" : "Empire State Pub", "borough" : "Brooklyn"},
  { "_id" : 4, "name" : "Stan's Pizzaria", "borough" : "Manhattan"},
  { "_id" : 5, "name" : "Jane's Deli", "borough" : "Brooklyn"},
] );
```

The following command uses the $sort to sort the restaurant name in ascending order:

```
db.users.aggregate(
  [
    { $sort : { name : 1} }
  ]
)
```

Using sort

Consider the following restaurant collection:

```
db.restaurants.insertMany( [
  { "_id" : 1, "name" : "Central Park Cafe", "borough" : "Manhattan"},
  { "_id" : 2, "name" : "Rock A Feller Bar and Grill", "borough" : "Queens"},
  { "_id" : 3, "name" : "Empire State Pub", "borough" : "Brooklyn"},
  { "_id" : 4, "name" : "Stan's Pizzaria", "borough" : "Manhattan"},
  { "_id" : 5, "name" : "Jane's Deli", "borough" : "Brooklyn"},
] );
```

The following command uses the $sort to sort the restaurant name in descending order:

```
db.users.aggregate(
  [
    { $sort : { name : -1} }
  ]
)
```