AGGREGATION IN MONGODB

- Aggregation in simple terms means **summarizing**.
- Aggregation in MongoDB is nothing but grouping a list of essential documents and then summarizing the information with a value.
- Based on these values, we can make comparisons and take decisions.
- For example – find the year where the maximum profit happened, find the category which has the highest average sales and so on.

AGGREGATION IN MONGODB

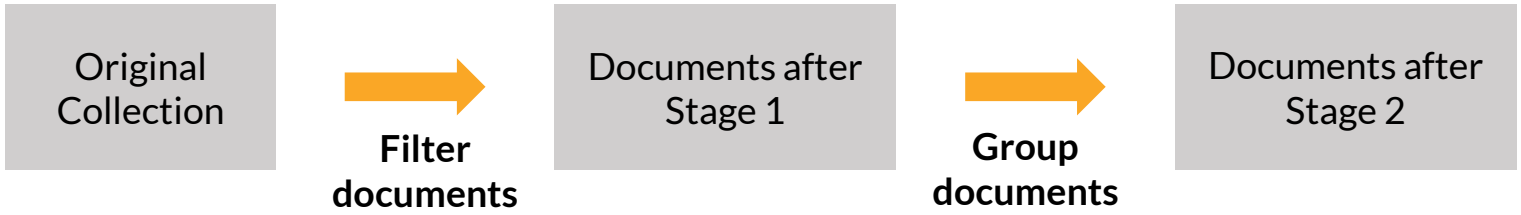
- Aggregation in MongoDB works in a series of steps called **stages** that combine to form the **aggregation pipeline**.

- Each stage consists of a specific operation that works on the documents present in the given collection and after each stage we continue working only on the processed documents from the previous stage. This way we avoid working on the entire collection's documents, and concentrate only on that specific subset which we're interested in.

AGGREGATION IN MONGODB

- Aggregation syntax – `db.Collection.aggregate([{stage1},{stage2},...])`

- Each of these stages can comprise of steps like:
  - Filtering
  - Grouping
  - Calculating summary values like sum, mean, max, etc.



AGGREGATION IN MONGODB

- For this exercise, we'll be using the same dataset from our querying exercise

- The fields present here are
  - Order ID
  - Segment
  - Product ID
  - Sales
  - Profit

- We shall use this dataset to understand some fundamental stages that are frequently used in the aggregation pipeline.

AGGREGATION IN MONGODB

- The first stage that we'll learn about is **$group**
- Let's say you want to know what the total Sales are across all the orders.
- The corresponding syntax for this would be –

```
db.purchases.aggregate([{$group:{_id:null, total: {$sum: '$Sales'}}}])
```

•The `_id` indicates by which field will the aggregation get grouped by. By keeping it as `null` we are indicating that we don't require any grouping and we just want the total Sales for all the orders.

•The `total: {$sum: '$Sales'}` indicates that we need to find the sum of all the Sales and store it in the total field in a new document.

AGGREGATION IN MONGODB

- Now let's say you want to find what the average Sales across each of the Segments – Corporate, Consumer and Home Office.

- The corresponding syntax for this would be –

```
db.purchases.aggregate([{$group:{_id:'$Segment', average: {$avg:
'$Sales'}}}])
```

  - By keeping `_id` as `'$Segment'` we're indicating that we want our final grouping to be done by the different categories present in Segment field.

  - The `average: {$avg: '$Sales'}` indicates that we need to find the average of all the Sales and store it in the average field in a new document.

AGGREGATION IN MONGODB

- We can also filter our documents by using the **$match** operator
- Let's say you want to find what the average Sales across each of the Segments – Corporate, Consumer and Home Office but for only those orders where the Profit was greater than or equal to 100
- The corresponding syntax for this would be –

```
db.purchases.aggregate
([{$match: {Profit:{$gte:100}}},{$group:{_id:'$Segment', average:
{$avg:'$Sales'}}}])
```

AGGREGATION IN MONGODB

- We can even alter the way the stages are kept in the pipeline.
- Let's say we want to get only those segment categories that have an average Sales greater than or equal to 300
- The corresponding syntax for this would be –

```
db.purchases.aggregate
([{group:{_id:'$Segment', average: {$avg:'$Sales'}}}, {$match:
{average:{$gte:300}}}])
```

QUERYING IN MONGODB

- Till now we have learnt the following CRUD operations
  - **C –** Create a new record in the database
  - **R –** Read a record and understand its contents
  - **U –** Update some values in the record
  - **D –** Delete a record from the database
- In this session we'll learn some fundamental querying techniques available in MongoDB

QUERYING IN MONGODB

- Before we go to the querying part, let's discuss the dataset that we'll be using
- The dataset comprises of order information for a retail giant and looks like this

Order ID	Segment	Product ID	Sales	Profit
IN-2014-76016	Corporate	FUR-BO-10004852	5667.87	2097.03
IN-2012-48240	Corporate	FUR-TA-10000226	1745.34	226.86
IN-2012-48240	Corporate	TEC-PH-10004664	1916.73	498.33
IN-2014-68463	Corporate	OFF-ST-10004060	1865.97	802.17
IN-2012-79439	Corporate	FUR-BO-10001372	3076.5	215.25
IN-2014-32084	Consumer	TEC-PH-10001457	2550	280.44
IN-2014-16887	Consumer	TEC-CO-10002040	1272.72	534.48

QUERYING IN MONGODB

- Data dictionary:
  - **Order ID** - Each order has a unique Order ID which is stored in the following pattern

    Country Code - *Year of Purchase* - *Order No*

    Ex - IN-2014-76016

QUERYING IN MONGODB

- Data dictionary:
  - **Order ID** - Each order has a unique Order ID which is stored in the following pattern

    Country Code - *Year of Purchase* - *Order No*

    Ex - IN-2014-76016
  - **Segment** – Corporate, Consumer or Home Office

QUERYING IN MONGODB

- Data dictionary:
  - **Order ID** - Each order has a unique Order ID which is stored in the following pattern

    Country Code - *Year of Purchase* - *Order No*

    Ex - IN-2014-76016
  - **Segment** – Corporate, Consumer or Home Office
  - **Product ID** – Each product being purchased has a unique pattern as shown

    Product Category - *Product Sub-Category* - *Product No*

    Ex - FUR-BO-10004852

QUERYING IN MONGODB

- Data dictionary:
  - **Order ID** - Each order has a unique Order ID which is stored in the following pattern

    Country Code - *Year of Purchase* - *Order No*

    Ex - IN-2014-76016
  - **Segment** – Corporate, Consumer or Home Office
  - **Product ID** – Each product being purchased has a unique pattern as shown

    Product Category - *Product Sub-Category* - *Product No*

    Ex - FUR-BO-10004852
  - **Sales** - Sales made by the order
  - **Profit** – Profit made by the order

QUERYING IN MONGODB

- Importing the dataset:
  - For importing the dataset we shall use the **mongoimport** utility
  - As per the mongodb documentation - *"The **mongoimport** tool imports content from an Extended JSON, CSV, or TSV export created by mongoexport, or potentially, another third-party export tool."*
  - MongoDB predominantly works with **json** files but in this case, we'll be using a **csv** file to do import the collection that we'll be using for our query analysis

QUERYING IN MONGODB

- Importing the dataset:
  - For importing the dataset we shall use the **mongoimport** utility
  - As per the mongodb documentation - *"The **mongoimport** tool imports content from an Extended JSON, CSV, or TSV export created by mongoexport, or potentially, another third-party export tool."*
  - MongoDB predominantly works with **json** files but in this case, we'll be using a **csv** file to do import the collection that we'll be using for our query analysis
  - The command for importing the dataset is as follows

```
mongoimport --db <database name> --collection <collection name> --
type csv --file <filepath> --headerline
```

QUERYING IN MONGODB

- **Basic Comparison Operators**
  - Equal to ($eq)
  - Greater than ($gt)
  - Greater than or equal to ($gte)
  - Less than ($lt)
  - Less than or equal to ($lte)
  - Not equal to ($neq)
  - In ($in)

QUERYING IN MONGODB

- **Logical operators** – If we want to combine multiple queries and retrieve documents which satisfy all or some of them, then you need to use Logical operators. The main logical operators are:

  - AND
  - OR
  - NOT

QUERYING IN MONGODB

- **Logical operators** – If we want to combine multiple queries and retrieve documents which satisfy all or some of them, then you need to use Logical operators. The main logical operators are:

  - AND
  - OR
  - NOT

- AND – `db.collection.find({$and: [{condition 1},{condition 2}...]})`

QUERYING IN MONGODB

- **Logical operators** – If we want to combine multiple queries and retrieve documents which satisfy all or some of them, then you need to use Logical operators. The main logical operators are:

  - AND
  - OR
  - NOT

- AND – `db.collection.find({$and: [{condition 1},{condition 2}...]})`

- OR – `db.collection.find({$or: [{condition 1},{condition 2}...]})`

QUERYING IN MONGODB

- **Logical operators** – If we want to combine multiple queries and retrieve documents which satisfy all or some of them, then you need to use Logical operators. The main logical operators are:
  - AND
  - OR
  - NOT
- AND – `db.collection.find({$and: [{condition 1},{condition 2}...]})`
- OR – `db.collection.find({$or: [{condition 1},{condition 2}...]})`
- NOT – `db.collection.find({field: {$not: {condition}}})`

QUERYING IN MONGODB

- **Regular expressions**– Regular expressions or regex help us in matching patterns in the data. Some common regular expression symbols are as follows:
  - `[]` : a set of characters
  - `\` : special sequence
  - `^` : starts with
  - `$` : ends with
  - `.` : any character except new line
  - Similarly we have several other regular expressions like `*` , `+` , `{}` , etc.

QUERYING IN MONGODB

- Let's take an example to understand where regular expressions are used
- Recall that the Product ID comprises of
  - Product Category – Office Supplies (OFF), Furniture (FUR) and Technology (TEC)
  - Product Sub Category
  - Product Number
- Let's say we want to find all the orders where the product category is Office Supplies. Here, we need to use regular expressions to identify which orders' Product ID starts with '**OFF**' pattern in order to perform the query.