**NIELIT Gorakhpur**

**Course name: A level** **SUBJECT:DATABASE TECHNOLOGIES**

**Topic: MongoDB** **DATE: 18/5/2020**

## Aggregation Framework

Aggregation is an operation used to process the data that returns the computed results. In Simple words, Aggregation groups the data from multiple documents in a collection and operates in several ways on those grouped data in order to return one combined result i.e. total number(sum), average, minimum, maximum etc out of the group selected. In SQL **count**(*) and with "**group by"** is an equivalent of MongoDB aggregation. In MongoDB, **aggregate()** method is used for the aggregation

### Syntax

**db.COLLECTION_NAME.aggregate(pipeline, options))**

where

| Name | Description | Type |
|---|---|---|
| pipeline | A sequence of data aggregation operations or stages. The method can still accept the pipeline stages as separate arguments instead of as elements in an array; however, if you do not specify the pipeline as an array, you cannot specify the options parameter. | Array (required) |
| options | Additional options that **aggregate**() passes to the aggregate command. | Document (optional) |

### Aggregation Stages or Aggregation pipeline operators

| Name | Description |
|---|---|
| $match | The $match operator filters the documents stram to pass only those documents that match the specified condition(s) to the next pipeline stage. **$match** uses standard MongoDB queries. For each input document, outputs either one document (a match) or zero documents (no match). |
| $project | The $project function in MongoDB passes along the documents with only the specified fields to the next stage in the pipeline, i.e. it Reshapes each document in the stream, such as by adding new fields or removing existing fields where the field may be the existing fields from the input documents or newly computed fields. |
| $group | In MongoDB, the $group operator groups the input documents by the specified |

|  |  |
|---|---|
|  | expression and groups the document for each distinct grouping. An identifier (**_id** ) field in the output documents contains the distinct group by key. The output documents can also contain computed fields that hold the values of some accumulator expression grouped by the $group‘s _id(identifier) field. |
| $unwind | The $unwind operator is used to deconstructing an array field from the input documents to output a document for each element. Each output document replaces the array with an element value i.e. Every output document is the input document with the value of the array field replaced by the element. |
| $sort | $ sort is used to Reorders the document stream by a specified sort key. It only changes the order not the documents. For each input document, outputs one document is there.. |
| $limit | $limit operator Passes the first n documents unmodified to the pipeline where n is the specified limit. For each input document, outputs either one document (for the first n documents) or zero documents (after the first n documents). |
| $skip | $skip operator Skips the first n documents where n is the specified skip number and passes the remaining documents unmodified to the pipeline. For each input document, outputs either zero documents (for the first n documents) or one document (if after the first n documents). |
| $set | $set, adds new fields to documents. Similar to $**operator**, $**set** reshapes each document in the stream; specifically, by adding new fields to output documents that contain both the existing fields from the input documents and the newly added fields. |
| $unset | Removes/excludes fields from documents. |

**<u>Different expressions used by Aggregate function</u>**

| **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 |
| $addToSet | Inserts values to an array but no duplicates in the resulting document |
| $push | Inserts values to an array in the resulting document |
| $first | Returns the first document from the source document |
| $last | Returns the last document from the source document |

## Aggregation Pipeline

The aggregation pipeline is a framework for data aggregation modeled on the concept of data processing pipelines. Documents enter a multi-stage pipeline that transforms the documents into aggregated results.

### Pipeline

The MongoDB aggregation pipeline consists of stages (aggregation states). Each stage transforms the documents as they pass through the pipeline. Pipeline stages do not need to produce one output document for every input document; e.g., some stages may generate new documents or filter out documents based on the various operators and functions etc.

The most basic pipeline stages provide *filters* that operate like queries and *document transformations* that modify the form of the output document.

Other pipeline operations provide tools for grouping and sorting documents by specific field or fields as well as tools for aggregating the contents of arrays, including arrays of documents. In addition, pipeline stages can use operators for tasks such as calculating the average, Sum, Min, MAX or concatenating a string also.

The pipeline provides efficient data aggregation using native operations within MongoDB, and is the preferred method for data aggregation in MongoDB.

The aggregation pipeline can use indexes to improve its performance during some of its stages. In addition, the aggregation pipeline has an internal optimization phase. Some are the pipeline stages which may take advantage of indexes are as under:

**$match** : The $match stage can use an index to filter documents if it occurs at the beginning of a pipeline.

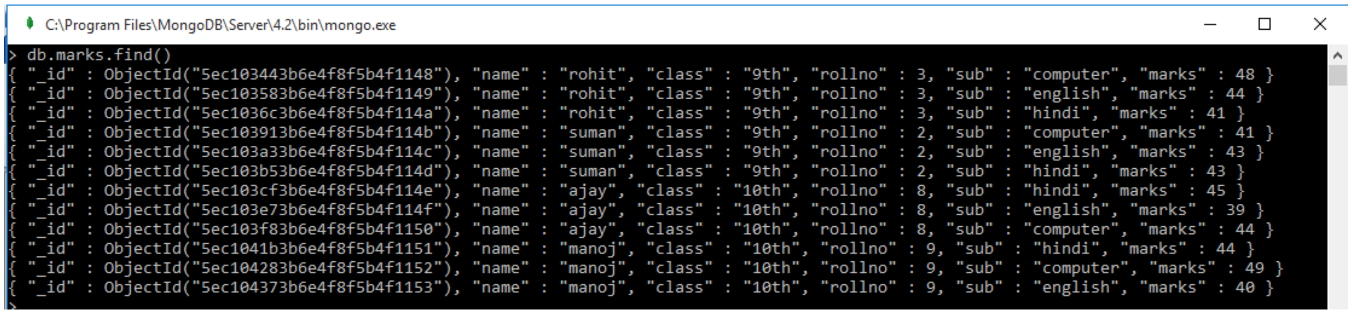
**$sort** : The $sort stage can use an index as long as it is not preceded by a **$project, $unwind** or **$group** stage.

**$group** : The $group stage may sometimes be used as an index to find the first document in each group if all of the following criteria are met:

- The $group stage is preceded by a $sort stage that sorts the field to group by,
- There is an index on the grouped field which matches the sort order and
- The only accumulator used in the $group stage is $first.

**Example:** lets take a collection marks, having marks of various subjects for each students in various class given as under:

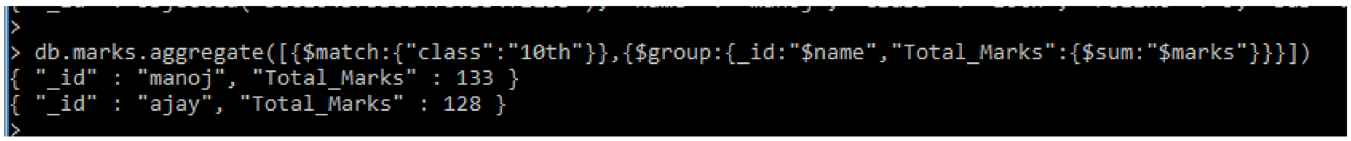
{ "_id" : ObjectId("5ec103443b6e4f8f5b4f1148"), "name" : "rohit", "class" : "9th", "rollno" : 3, "sub" : "computer", "marks" : 48 }
{ "_id" : ObjectId("5ec103583b6e4f8f5b4f1149"), "name" : "rohit", "class" : "9th", "rollno" : 3, "sub" : "english", "marks" : 44 }
{ "_id" : ObjectId("5ec1036c3b6e4f8f5b4f114a"), "name" : "rohit", "class" : "9th", "rollno" : 3, "sub" : "hindi", "marks" : 41 }
{ "_id" : ObjectId("5ec103913b6e4f8f5b4f114b"), "name" : "suman", "class" : "9th", "rollno" : 2, "sub" : "computer", "marks" : 41 }
{ "_id" : ObjectId("5ec103a33b6e4f8f5b4f114c"), "name" : "suman", "class" : "9th", "rollno" : 2, "sub" : "english", "marks" : 43 }
{ "_id" : ObjectId("5ec103b53b6e4f8f5b4f114d"), "name" : "suman", "class" : "9th", "rollno" : 2, "sub" : "hindi", "marks" : 43 }
{ "_id" : ObjectId("5ec103cf3b6e4f8f5b4f114e"), "name" : "ajay", "class" : "10th", "rollno" : 8, "sub" : "hindi", "marks" : 45 }
{ "_id" : ObjectId("5ec103e73b6e4f8f5b4f114f"), "name" : "ajay", "class" : "10th", "rollno" : 8, "sub" : "english", "marks" : 39 }
{ "_id" : ObjectId("5ec103f83b6e4f8f5b4f1150"), "name" : "ajay", "class" : "10th", "rollno" : 8, "sub" : "computer", "marks" : 44 }
{ "_id" : ObjectId("5ec1041b3b6e4f8f5b4f1151"), "name" : "manoj", "class" : "10th", "rollno" : 9, "sub" : "hindi", "marks" : 44 }
{ "_id" : ObjectId("5ec104283b6e4f8f5b4f1152"), "name" : "manoj", "class" : "10th", "rollno" : 9, "sub" : "computer", "marks" : 49 }
{ "_id" : ObjectId("5ec104373b6e4f8f5b4f1153"), "name" : "manoj", "class" : "10th", "rollno" : 9, "sub" : "english", "marks" : 40 }


C:\Program Files\MongoDB\Server\4.2\bin\mongo.exe

```
> db.marks.find()
{ "_id" : ObjectId("5ec103443b6e4f8f5b4f1148"), "name" : "rohit", "class" : "9th", "rollno" : 3, "sub" : "computer", "marks" : 48 }
{ "_id" : ObjectId("5ec103583b6e4f8f5b4f1149"), "name" : "rohit", "class" : "9th", "rollno" : 3, "sub" : "english", "marks" : 44 }
{ "_id" : ObjectId("5ec1036c3b6e4f8f5b4f114a"), "name" : "rohit", "class" : "9th", "rollno" : 3, "sub" : "hindi", "marks" : 41 }
{ "_id" : ObjectId("5ec103913b6e4f8f5b4f114b"), "name" : "suman", "class" : "9th", "rollno" : 2, "sub" : "computer", "marks" : 41 }
{ "_id" : ObjectId("5ec103a33b6e4f8f5b4f114c"), "name" : "suman", "class" : "9th", "rollno" : 2, "sub" : "english", "marks" : 43 }
{ "_id" : ObjectId("5ec103b53b6e4f8f5b4f114d"), "name" : "suman", "class" : "9th", "rollno" : 2, "sub" : "hindi", "marks" : 43 }
{ "_id" : ObjectId("5ec103cf3b6e4f8f5b4f114e"), "name" : "ajay", "class" : "10th", "rollno" : 8, "sub" : "hindi", "marks" : 45 }
{ "_id" : ObjectId("5ec103e73b6e4f8f5b4f114f"), "name" : "ajay", "class" : "10th", "rollno" : 8, "sub" : "english", "marks" : 39 }
{ "_id" : ObjectId("5ec103f83b6e4f8f5b4f1150"), "name" : "ajay", "class" : "10th", "rollno" : 8, "sub" : "computer", "marks" : 44 }
{ "_id" : ObjectId("5ec1041b3b6e4f8f5b4f1151"), "name" : "manoj", "class" : "10th", "rollno" : 9, "sub" : "hindi", "marks" : 44 }
{ "_id" : ObjectId("5ec104283b6e4f8f5b4f1152"), "name" : "manoj", "class" : "10th", "rollno" : 9, "sub" : "computer", "marks" : 49 }
{ "_id" : ObjectId("5ec104373b6e4f8f5b4f1153"), "name" : "manoj", "class" : "10th", "rollno" : 9, "sub" : "english", "marks" : 40 }
>
```

1. **Now, lets execute the following aggregate command:**

db.marks.aggregate**([{$match:{"class":"10th"}}**,**{$group:{_id:"$name","Total_Marks":{$sum: "$marks"}}}**])



```
>
> db.marks.aggregate([{$match:{"class":"10th"}},{$group:{_id:"$name","Total_Marks":{$sum:"$marks"}}}])
{ "_id" : "manoj", "Total_Marks" : 133 }
{ "_id" : "ajay", "Total_Marks" : 128 }
>
```

This command executed in two states,

**first Stage**: The **$match** stage filters the documents by the status field and passes to the next stage those documents that have class equal to "10th".

**Second Stage**: The **$group** stage groups the documents by the sub field to calculate the sum of the amount for marks.

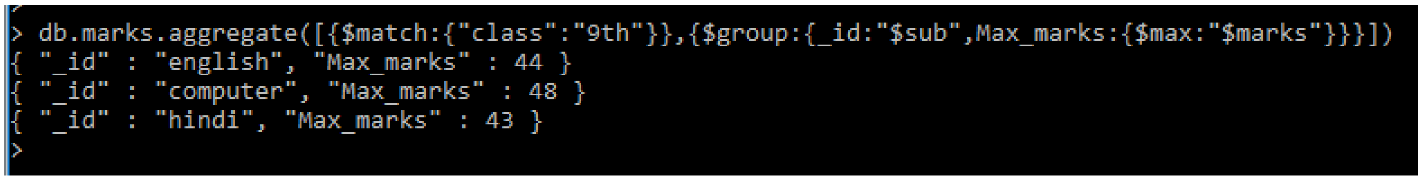
This resulted into:

**{ "_id" : "manoj", "Total_Marks" : 133 }**

**{ "_id" : "ajay", "Total_Marks" : 128 }**

**2. Now Run another, aggregate command and see the output**

db.marks.aggregate([**{$match:{"class":"9th"}},{$group:{_id:"$sub",Max_marks:{$max:"$marks"}}}]**)



```
> db.marks.aggregate([{$match:{"class":"9th"}},{$group:{_id:"$sub",Max_marks:{$max:"$marks"}}}])
{ "_id" : "english", "Max_marks" : 44 }
{ "_id" : "computer", "Max_marks" : 48 }
{ "_id" : "hindi", "Max_marks" : 43 }
>
```

Here, in state 1, students of class 10th are filtered out and then Maximum marks of each subject has been computed (filtered). The output is:

**{ "_id" : "english", "Max_marks" : 44 }**

**{ "_id" : "computer", "Max_marks" : 48 }**

**{ "_id" : "hindi", "Max_marks" : 43 }**

**Assignment**

1. What is aggregation framework? List some aggregation stage operators with their usage.

2. What is aggregation pipeline?