Mongodb:Aggregation

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.

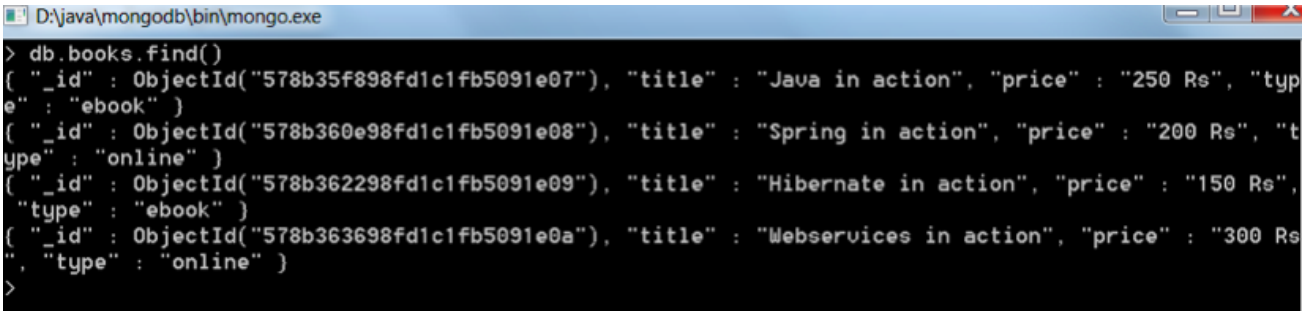
Aggregate function groups the records in a collection, and can be used to provide total number(sum), average, minimum, maximum etc out of the group selected.

In order to perform the aggregate function in MongoDB, `aggregate ()` is the function to be used. Following is the syntax for aggregation :

```
db.collection_name.aggregate(aggregate_operation)
```

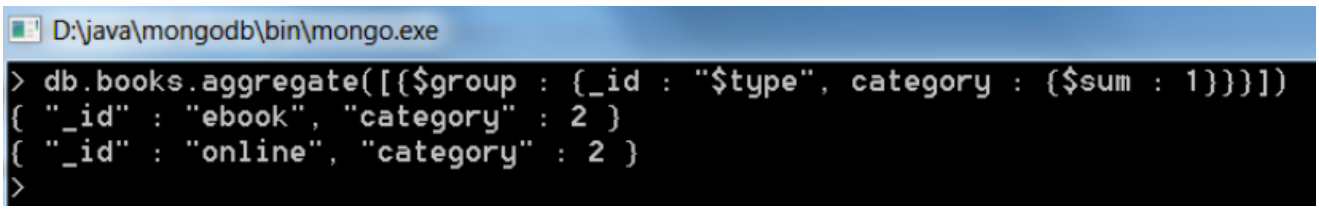
Example:

Consider a collection named **books**, with fieldnames - title, price and type. Use `db.books.find()` to list down the contents of the collection.



From the above collection, use the aggregate function to **group** the books which are of the type **ebook** and **online**.

```
db.books.aggregate([{$group : {_id: "$type", category: {$sum : 1}}}])
```



The above aggregate function will give result, which says there are 2 records of the type *ebook* and 2 records of the type *online*. So the above aggregation command, has grouped our collection data, based on their **type**.

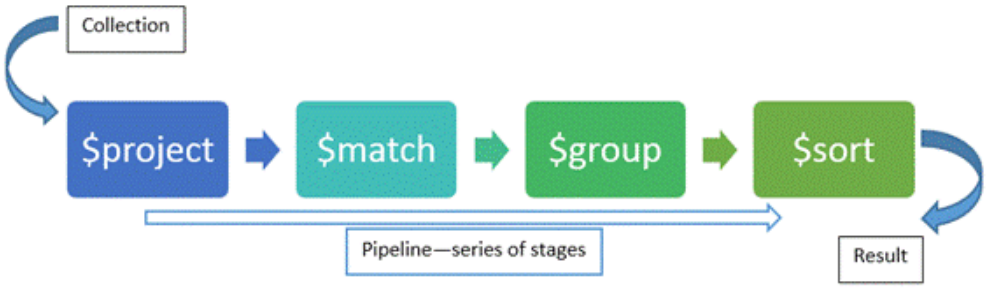
Different expressions used by Aggregate function

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
$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

Aggregations work as a pipeline, or a list of operators/filters applied to the data. We can pipe a collection into the top and transform it through a series of operations, eventually popping a result out the bottom.

Operators come in three varieties: stages, expressions, and accumulators.

When calling aggregate on a collection, we pass a list of stage operators. Documents are processed through the stages in sequence, with each stage applying to each document individually. Most importantly remember, **that aggregation is a "pipeline" and is just exactly that, being "piped" processes that feed input into the each stage as it goes along**. The output from the first thing goes to the next thing and then that manipulates to give input to the next thing and so on.



Matching Documents

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. To find all users that live in Beverly Hills (or more specifically, the *90210* area code), we'll add a match stage to our aggregation pipeline:

```
> db.customers.aggregate([
```

```
  { $match: { "zip": 90210 }}
]);
```

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.

Before we dive into more complex operations, let's start with something simple: counting the documents we matched in the previous section:

```
> db.customers.aggregate([
  { $match: {"zip": "90210"}},
  {
    $group: {
      _id: null,
      count: {
        $sum: 1
      }
    }
  }
]);
```

The `$group` transformation allows us to group documents together and performs transformations or operations across all of those grouped documents. In this case, we're creating a new field in the results called *count* which adds 1 to a running sum for every document. The `_id` field is required for grouping and would normally contain fields from each document that we'd like to preserve (ie: phoneNumber). Since we're just looking for the count of every document, we can make it null here.

```
{ "_id" : null, "count" : 24 }
```

Here we saw the use of the `$sum` arithmetic operator, which sums a field in all of the documents in a collection. We can group our documents together on any fields we'd like and perform other types of computations as well. Let's take a look at some of the other operators we can use and how we can use them.

Sum, Min, Max, and Avg

Let us take example of a transaction model looks like this:

```
{
  "id": "1",
  "productId": "1",
  "customerId": "1",
  "amount": 20.00,
  "transactionDate": ISODate("2017-02-23T15:25:56.314Z")
```

```
}
```

Let's start by calculating the total amount of sales made for the month of January. We'll start by matching only transactions that occurred between January 1 and January 31.

```
{
   $match: {
    transactionDate: {
        $gte: ISODate("2017-01-01T00:00:00.000Z"),
        $lt: ISODate("2017-02-01T00:00:00.000Z")
    }
  }
}
```

The next stage of the pipeline is summing the transaction amounts and putting that amount in a new field called `total`:

```
{
  $group: {
    _id: null,
    total: {
      $sum: "$amount"
    }
  }
}
```

The final query looks something like this:

```
> 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"
      }
    }
  }
]);
```

The final result is a transaction amount that looks like the following:

```
{ _id: null, total: 20333.00 }
```

Some other helpful monthly metrics we might want are the average price of each transaction, and the minimum and maximum transaction in the month. Let's add those to our group so we can get a single picture of the entire month:

Combined with the `$match` statement it looks like the following:

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

Our final result gives us an interesting picture of monthly sales

```
{
  _id: null,
  total: 20333.00,
  average_transaction_amount: 8.50,
  min_transaction_amount: 2.99,
  max_transaction_amount: 347.22
}
```