MongoDB

Unit 2

DB Commands – Aggregate

- Calculates aggregate values for the data in a collection or a view.
  - **db.<collection_name>.aggregate(pipeline, options)**
    - **db.Semfour.aggregate([{$match: {city : "Bangalore"}},**
      {$group: {_id: "$_id", total: {$sum: "$age"}}},
      {$sort: {total: -1}}])
    - Select documents with status equal to "A", group the matching documents by the cust_id field and calculate the total for each cust_id field from the sum of the amount field, and sort the results by the total field in descending order.

```
> db.orders.find()
{ "_id" : 1, "cust_id" : "abc1", "ord_date" : ISODate("2012-11-02T17:04:11.102Z"), "status" : "A", "amount" : 50 }
{ "_id" : 2, "cust_id" : "xyz1", "ord_date" : ISODate("2013-10-01T17:04:11.102Z"), "status" : "A", "amount" : 100 }
{ "_id" : 3, "cust_id" : "xyz1", "ord_date" : ISODate("2013-10-12T17:04:11.102Z"), "status" : "D", "amount" : 25 }
{ "_id" : 4, "cust_id" : "xyz1", "ord_date" : ISODate("2013-10-11T17:04:11.102Z"), "status" : "D", "amount" : 125 }
{ "_id" : 5, "cust_id" : "abc1", "ord_date" : ISODate("2013-11-12T17:04:11.102Z"), "status" : "A", "amount" : 25 }
> db.orders.aggregate([
...                     { $match: { status: "A" } },
...                     { $group: { _id: "$cust_id", total: { $sum: "$amount" } } },
...                     { $sort: { total: -1 } }
...                   ])
{ "_id" : "xyz1", "total" : 100 }
{ "_id" : "abc1", "total" : 75 }
```

DB Aggregation Pipeline Stages

$match

- Filters the documents to pass only the documents that match the specified condition(s) to the next pipeline stage.
- The $match stage has the following prototype form:
  - **{$match: {<query>}}**
- $match takes a document that specifies the query conditions.
  - Perform a simple equality match.

```
> db.articles.find()
{ "_id" : ObjectId("512bc95fe835e68f199c8686"), "author" : "dave", "score" : 80, "views" : 100 }
{ "_id" : ObjectId("512bc962e835e68f199c8687"), "author" : "dave", "score" : 85, "views" : 521 }
{ "_id" : ObjectId("55f5a192d4bede9ac365b257"), "author" : "ahn", "score" : 60, "views" : 1000 }
{ "_id" : ObjectId("55f5a192d4bede9ac365b258"), "author" : "li", "score" : 55, "views" : 5000 }
{ "_id" : ObjectId("55f5a1d3d4bede9ac365b259"), "author" : "annT", "score" : 60, "views" : 50 }
{ "_id" : ObjectId("55f5a1d3d4bede9ac365b25a"), "author" : "li", "score" : 94, "views" : 999 }
{ "_id" : ObjectId("55f5a1d3d4bede9ac365b25b"), "author" : "ty", "score" : 95, "views" : 1000 }
> db.articles.aggregate(
...     [ { $match : { author : "dave" } } ]
... );
{ "_id" : ObjectId("512bc95fe835e68f199c8686"), "author" : "dave", "score" : 80, "views" : 100 }
{ "_id" : ObjectId("512bc962e835e68f199c8687"), "author" : "dave", "score" : 85, "views" : 521 }
```

    - Select the documents where, either the score is greater than 70 and less than 90 or the views is greater than or equal to 1000 and perform a count.

```
> db.articles.aggregate( [
...   { $match: { $or: [ { score: { $gt: 70, $lt: 90 } }, { views: { $gte: 1000 } } ] } },
...   { $group: { _id: null, count: { $sum: 1 } } }
... ] );
{ "_id" : null, "count" : 5 }
```

$group

- Groups documents by some specified expression and outputs to the next stage a document for each distinct grouping.
- The output documents contain an _id field which 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 field.
- $group does not order its output documents.
  - **{$group: {_id: <expression>, <field1>: {<accumulator1>:<expression1>}, ...}}**
    - The _id field is mandatory.
    - Group the documents by the item to retrieve the distinct item values.

```
> db.sales.find()
{ "_id" : 1, "item" : "abc", "price" : 10, "quantity" : 2, "date" : ISODate("2014-03-01T08:00:00Z") }
{ "_id" : 2, "item" : "jkl", "price" : 20, "quantity" : 1, "date" : ISODate("2014-03-01T09:00:00Z") }
{ "_id" : 3, "item" : "xyz", "price" : 5, "quantity" : 10, "date" : ISODate("2014-03-15T09:00:00Z") }
{ "_id" : 4, "item" : "xyz", "price" : 5, "quantity" : 20, "date" : ISODate("2014-04-04T11:21:39.736Z") }
{ "_id" : 5, "item" : "abc", "price" : 10, "quantity" : 10, "date" : ISODate("2014-04-04T21:23:13.331Z") }
> db.sales.aggregate( [ { $group : { _id : "$item" } } ] )
{ "_id" : "xyz" }
{ "_id" : "jkl" }
{ "_id" : "abc" }
```

    - Group the documents by the month, day, and year and calculates the total price and the average quantity as well as counts the documents per each group

```
> db.sales.find()
{ "_id" : 1, "item" : "abc", "price" : 10, "quantity" : 2, "date" : ISODate("2014-03-01T08:00:00Z") }
{ "_id" : 2, "item" : "jkl", "price" : 20, "quantity" : 1, "date" : ISODate("2014-03-01T09:00:00Z") }
{ "_id" : 3, "item" : "xyz", "price" : 5, "quantity" : 10, "date" : ISODate("2014-03-15T09:00:00Z") }
{ "_id" : 4, "item" : "xyz", "price" : 5, "quantity" : 20, "date" : ISODate("2014-04-04T11:21:39.736Z") }
{ "_id" : 5, "item" : "abc", "price" : 10, "quantity" : 10, "date" : ISODate("2014-04-04T21:23:13.331Z") }
> db.sales.aggregate([{
...     $group : { _id : { month: { $month: "$date" },
...                day: { $dayOfMonth: "$date" }, year: { $year: "$date" } },
...                totalPrice: { $sum: { $multiply: [ "$price", "$quantity" ] } },
...                averageQuantity: { $avg: "$quantity" }, count: { $sum: 1 }}}])
{ "_id" : { "month" : 4, "day" : 4, "year" : 2014 }, "totalPrice" : 200, "averageQuantity" : 15, "count" : 2 }
{ "_id" : { "month" : 3, "day" : 15, "year" : 2014 }, "totalPrice" : 50, "averageQuantity" : 10, "count" : 1 }
{ "_id" : { "month" : 3, "day" : 1, "year" : 2014 }, "totalPrice" : 40, "averageQuantity" : 1.5, "count" : 2 }
```

$project

- Pass along the documents with only the specified fields to the next stage in the pipeline.
- This may be the existing fields from the input documents or newly computed fields.
  - **{$project: {<specifications>}}**
    - The specification for $project command contains the inclusion of fields, the suppression of the _id field, the addition of new fields, and the resetting the values of existing fields.
    - Include only the title field in the embedded document in the stop field.

```
> db.Bmarks.find()
{ "_id" : 1, "user" : "1234", "stop" : { "title" : "book1", "author" : "xyz", "page" : 32 } }
{ "_id" : 2, "user" : "7890", "stop" : [ { "title" : "book2", "author" : "abc", "page" : 5 }, { "title" : "book3", "au
thor" : "ijk", "page" : 100 } ] }
> db.Bmarks.aggregate( [ { $project: { "stop.title": 1 } } ] )
{ "_id" : 1, "stop" : { "title" : "book1" } }
{ "_id" : 2, "stop" : [ { "title" : "book2" }, { "title" : "book3" } ] }
```

```
> db.Bmarks.find()
{ "_id" : 1, "user" : "1234", "stop" : { "title" : "book1", "author" : "xyz", "page" : 32 } }
{ "_id" : 2, "user" : "7890", "stop" : [ { "title" : "book2", "author" : "abc", "page" : 5 }, { "title" : "book3", "au
thor" : "ijk", "page" : 100 } ] }
> db.Bmarks.aggregate( [ { $project: { stop: { title: 1 } } } ] )
{ "_id" : 1, "stop" : { "title" : "book1" } }
{ "_id" : 2, "stop" : [ { "title" : "book2" }, { "title" : "book3" } ] }
```

- Add new fields isbn, lastName, and copiesSold

```
> db.Books.find()
{ "_id" : 1, "title" : "abc123", "isbn" : "0001122223334", "author" : { "last" : "zzz", "first" : "aaa" }, "copies" :
5, "lastModified" : "2016-07-28" }
{ "_id" : 2, "title" : "Baked Goods", "isbn" : "9999999999999", "author" : { "last" : "xyz", "first" : "abc", "middle"
 : "" }, "copies" : 2, "lastModified" : "2017-07-21" }
{ "_id" : 3, "title" : "Ice Cream Cakes", "isbn" : "8888888888888", "author" : { "last" : "xyz", "first" : "abc", "mid
dle" : "mmm" }, "copies" : 5, "lastModified" : "2017-07-22" }
> db.Books.aggregate([{$project: {title: 1, isbn: {prefix: { $substr: [ "$isbn", 0, 3 ] },group: { $substr: [ "$isbn",
 3, 2 ] },publisher: { $substr: [ "$isbn", 5, 4 ] },title: { $substr: [ "$isbn", 9, 3 ] },checkDigit: { $substr: [ "$i
sbn", 12, 1] }},lastName: "$author.last",copiesSold: "$copies"}}])
{ "_id" : 1, "title" : "abc123", "isbn" : { "prefix" : "000", "group" : "11", "publisher" : "2222", "title" : "333", "
checkDigit" : "4" }, "lastName" : "zzz", "copiesSold" : 5 }
{ "_id" : 2, "title" : "Baked Goods", "isbn" : { "prefix" : "999", "group" : "99", "publisher" : "9999", "title" : "99
9", "checkDigit" : "9" }, "lastName" : "xyz", "copiesSold" : 2 }
{ "_id" : 3, "title" : "Ice Cream Cakes", "isbn" : { "prefix" : "888", "group" : "88", "publisher" : "8888", "title" :
 "888", "checkDigit" : "8" }, "lastName" : "xyz", "copiesSold" : 5 }
```

- Use the REMOVE variable to excludes the author.middle field only if it equals ""

```
> db.Books.aggregate( [
...    {
...       $project: {
...          title: 1,
...          "author.first": 1,
...          "author.last" : 1,
...          "author.middle": {
...             $cond: {
...                if: { $eq: [ "", "$author.middle" ] },
...                then: "$$REMOVE",
...                else: "$author.middle"
...             }
...          }
...       }
...    }
... ] )
{ "_id" : 1, "title" : "abc123", "author" : { "last" : "zzz", "first" : "aaa" } }
{ "_id" : 2, "title" : "Baked Goods", "author" : { "last" : "xyz", "first" : "abc" } }
{ "_id" : 3, "title" : "Ice Cream Cakes", "author" : { "last" : "xyz", "first" : "abc", "middle" : "mmm" } }
```

- Project the fields x and y as elements in a new field myArray.

```
> db.coll.find()
{ "_id" : ObjectId("55ad167f320c6be244eb3b95"), "x" : 1, "y" : 1 }
{ "_id" : ObjectId("56ad167f320c6be244eb3b95"), "x" : 2, "y" : 2 }
> db.coll.aggregate( [ { $project: { myArray: [ "$x", "$y" ] } } ] )
{ "_id" : ObjectId("55ad167f320c6be244eb3b95"), "myArray" : [ 1, 1 ] }
{ "_id" : ObjectId("56ad167f320c6be244eb3b95"), "myArray" : [ 2, 2 ] }
```

$count

- Passes a document to the next stage that contains a count of the number of documents input to the stage.
  - **{$count: <string>}**

- Exclude documents that have a score value of less than or equal to 80 to pass along the documents with score greater than 80 to the next stage and returns count of the remaining documents in the aggregation pipeline and assigns the value to a field called passing_scores.

```
> db.scores.find()
{ "_id" : 1, "subject" : "History", "score" : 88 }
{ "_id" : 2, "subject" : "History", "score" : 92 }
{ "_id" : 3, "subject" : "History", "score" : 97 }
{ "_id" : 4, "subject" : "History", "score" : 71 }
{ "_id" : 5, "subject" : "History", "score" : 79 }
{ "_id" : 6, "subject" : "History", "score" : 83 }
> db.scores.aggregate([{$match: {score: {$gt: 80}}},{$count: "passing_scores"}])
{ "passing_scores" : 4 }
```

$out

- Create a new collection in the current database if the specified one does not already exist based on an aggregation.
  - **{ $out: "<output-collection>" }**
    - It should be the last stage in the pipeline operator.
      - Pivot the data in the books collection to have titles grouped by authors and then write the results to the authors collection.

```
> db.books.aggregate([{ $group : { _id : "$author", books: { $push: "$title" } } },{ $out : "authors" }] )
> db.books.find()
{ "_id" : 8751, "title" : "The Banquet", "author" : "Dante", "copies" : 2 }
{ "_id" : 8752, "title" : "Divine Comedy", "author" : "Dante", "copies" : 1 }
{ "_id" : 8645, "title" : "Eclogues", "author" : "Dante", "copies" : 2 }
{ "_id" : 7000, "title" : "The Odyssey", "author" : "Homer", "copies" : 10 }
{ "_id" : 7020, "title" : "Iliad", "author" : "Homer", "copies" : 10 }
> db.authors.find()
{ "_id" : "Homer", "books" : [ "The Odyssey", "Iliad" ] }
{ "_id" : "Dante", "books" : [ "The Banquet", "Divine Comedy", "Eclogues" ] }
```

      - Return the data of the invoice collection to have invoice date grouped by the item and then writes the results to the newinvoice collection

```
> db.invoice.find()
{ "_id" : 1, "item" : "doz", "qty" : 20, "rate" : 10, "inv_date" : "02/02/2014" }
{ "_id" : 2, "item" : "sam", "qty" : 15, "rate" : 8, "inv_date" : "05/12/2014" }
{ "_id" : 3, "item" : "amp", "qty" : 25, "rate" : 8, "inv_date" : "07/02/2014" }
{ "_id" : 4, "item" : "doz", "qty" : 20, "rate" : 10, "inv_date" : "02/02/2014" }
{ "_id" : 5, "item" : "amp", "qty" : 10, "rate" : 8, "inv_date" : "05/12/2014" }
{ "_id" : 6, "item" : "doz", "qty" : 30, "rate" : 10, "inv_date" : "13/04/2014" }
{ "_id" : 7, "item" : "sam", "qty" : 15, "rate" : 8, "inv_date" : "05/12/2014" }
{ "_id" : null, "item" : "mks", "qty" : 10, "rate" : 20, "inv_date" : "17/12/2014" }
```

```
> db.invoice.aggregate(
... [
... { $group : { _id : "$item", invoiceDate: { $push: "$inv_date" } } },
...         { $out : "newinvoice" }
... ] )
```

```
> db.newinvoice.find()
{ "_id" : "mks", "invoiceDate" : [ "17/12/2014" ] }
{ "_id" : "amp", "invoiceDate" : [ "07/02/2014", "05/12/2014" ] }
{ "_id" : "sam", "invoiceDate" : [ "05/12/2014", "05/12/2014" ] }
{ "_id" : "doz", "invoiceDate" : [ "02/02/2014", "02/02/2014", "13/04/2014" ] }
```

$skip

- Skips over the specified number of documents that pass into the stage and passes the remaining documents to the next stage in the pipeline.
  - {$skip: <positive integer>}
- $skip takes a positive integer that specifies the maximum number of documents to skip.

```
> db.Mycol.aggregate( {$skip : 5} )
{ "_id" : 5 }
{ "_id" : 6, "a" : ISODate("2019-03-05T03:49:51.314Z") }
{ "_id" : 7, "a" : 9.9090909 }
```

$limit

- Limits the number of documents passed to the next stage in the pipeline.
  - {$limit: <positive integer>}
- $limit takes a positive integer that specifies the maximum number of documents to pass along.

```
> db.Mycol.aggregate(
... {$limit : 4}
... )
{ "_id" : 0, "a" : 8 }
{ "_id" : 1, "a" : [ 41.63, 88.19 ] }
{ "_id" : 2, "a" : { "a" : "apple", "b" : "banana", "c" : "carrot" } }
{ "_id" : 3, "a" : "caribou" }
```

$sample

- Randomly selects the specified number of documents from its input.
  - {$sample: {size: <positive integer>}}

```
> db.Mycol.aggregate(
...    [ { $sample: { size: 3 } } ]
... )
{ "_id" : 5 }
{ "_id" : 2, "a" : { "a" : "apple", "b" : "banana", "c" : "carrot" } }
{ "_id" : 6, "a" : ISODate("2019-03-05T03:49:51.314Z") }
```

$lookup

- Performs a left outer join to an unsharded collection in the same database to filter in documents from the "joined" collection for processing.
- To each input document, the $lookup stage adds a new array field whose elements are the matching documents from the "joined" collection.
- The $lookup stage passes these reshaped documents to the next stage.
  - Equality Match

```
{
   $lookup:
     {
       from: <collection to join>,
       localField: <field from the input documents>,
       foreignField: <field from the documents of the "from" collection>,
       as: <output array field>
     }
}
```

  - Join Conditions and Uncorrelated Sub-queries

```
{
   $lookup:
     {
       from: <collection to join>,
       let: { <var_1>: <expression>, …, <var_n>: <expression> },
       pipeline: [ <pipeline to execute on the collection to join> ],
       as: <output array field>
     }
}
```

```
> db.Order.find()
{ "_id" : 1, "item" : "almonds", "price" : 12, "quantity" : 2 }
{ "_id" : 2, "item" : "pecans", "price" : 20, "quantity" : 1 }
{ "_id" : 3 }
> db.Inventory.find()
{ "_id" : 1, "sku" : "almonds", "description" : "product 1", "instock" : 120 }
{ "_id" : 2, "sku" : "bread", "description" : "product 2", "instock" : 80 }
{ "_id" : 3, "sku" : "cashews", "description" : "product 3", "instock" : 60 }
{ "_id" : 4, "sku" : "pecans", "description" : "product 4", "instock" : 70 }
{ "_id" : 5, "sku" : null, "description" : "Incomplete" }
{ "_id" : 6 }
```

  - Joins the documents from Order with the documents from the Inventory collection using the fields item from the orders collection and the sku field from the inventory collection

```
> db.Order.aggregate([
...    {
...      $lookup:
...        {
...          from: "Inventory",
...          localField: "item",
...          foreignField: "sku",
...          as: "inventory_docs"
...        }
...    }
... ])
{ "_id" : 1, "item" : "almonds", "price" : 12, "quantity" : 2, "inventory_docs" : [ { "_id" : 1, "sku" : "almonds", "d
escription" : "product 1", "instock" : 120 } ] }
{ "_id" : 2, "item" : "pecans", "price" : 20, "quantity" : 1, "inventory_docs" : [ { "_id" : 4, "sku" : "pecans", "des
cription" : "product 4", "instock" : 70 } ] }
{ "_id" : 3, "inventory_docs" : [ { "_id" : 5, "sku" : null, "description" : "Incomplete" }, { "_id" : 6 } ] }
```

- The equivalent SQL code for the same is
  - SELECT *, inventory_docs
  - FROM Order
  - WHERE inventory_docs IN (SELECT *
  - FROM Inventory
  - WHERE sku= orders.item);

```
> db.NutOrder.find()
{ "_id" : 1, "item" : "almonds", "price" : 12, "ordered" : 2 }
{ "_id" : 2, "item" : "pecans", "price" : 20, "ordered" : 1 }
{ "_id" : 3, "item" : "cookies", "price" : 10, "ordered" : 60 }
> db.NutWarehouse.find()
{ "_id" : 1, "stock_item" : "almonds", "warehouse" : "A", "instock" : 120 }
{ "_id" : 2, "stock_item" : "pecans", "warehouse" : "A", "instock" : 80 }
{ "_id" : 3, "stock_item" : "almonds", "warehouse" : "B", "instock" : 60 }
{ "_id" : 4, "stock_item" : "cookies", "warehouse" : "B", "instock" : 40 }
{ "_id" : 5, "stock_item" : "cookies", "warehouse" : "A", "instock" : 80 }
```

- Join the orders collection with the warehouse collection by the item and whether the quantity in stock is sufficient to cover the ordered quantity.

```
> db.NutOrder.aggregate([
...    {
...       $lookup:
...          {
...            from: "NutWarehouse",
...            let: { order_item: "$item", order_qty: "$ordered" },
...            pipeline: [
...               { $match:
...                  { $expr:
...                     { $and:
...                        [
...                          { $eq: [ "$stock_item",  "$$order_item" ] },
...                          { $gte: [ "$instock", "$$order_qty" ] }
...                        ]
...                     }
...                  }
...               },
...               { $project: { stock_item: 0, _id: 0 } }
...            ],
...            as: "stockdata"
...          }
...    }
... ])
```

```
{ "_id" : 1, "item" : "almonds", "price" : 12, "ordered" : 2, "stockdata" : [ { "warehouse" : "A", "instock" : 120 },
{ "warehouse" : "B", "instock" : 60 } ] }
{ "_id" : 2, "item" : "pecans", "price" : 20, "ordered" : 1, "stockdata" : [ { "warehouse" : "A", "instock" : 80 } ] }

{ "_id" : 3, "item" : "cookies", "price" : 10, "ordered" : 60, "stockdata" : [ { "warehouse" : "A", "instock" : 80 } ]
 }
```

- The equivalent SQL code for the same is
  - SELECT *, stockdata
  - FROM NutOrder
  - WHERE stockdata IN (SELECT warehouse, instock
  - FROM NutWarehouse
  - WHERE stock_item= NutOrder.item
  - AND instock >= NutOrder.ordered);

```
> db.Absences.find()
{ "_id" : 1, "student" : "Ann Aardvark", "sickdays" : [ ISODate("2018-05-01T00:00:00Z"), ISODate("2018-08-23T00:00:00Z
") ] }
{ "_id" : 2, "student" : "Zoe Zebra", "sickdays" : [ ISODate("2018-02-01T00:00:00Z"), ISODate("2018-05-23T00:00:00Z")
] }
> db.Holidays.find()
{ "_id" : 1, "year" : 2018, "name" : "New Years", "date" : ISODate("2018-01-01T00:00:00Z") }
{ "_id" : 2, "year" : 2018, "name" : "Pi Day", "date" : ISODate("2018-03-14T00:00:00Z") }
{ "_id" : 3, "year" : 2018, "name" : "Ice Cream Day", "date" : ISODate("2018-07-15T00:00:00Z") }
{ "_id" : 4, "year" : 2017, "name" : "New Years", "date" : ISODate("2017-01-01T00:00:00Z") }
{ "_id" : 5, "year" : 2017, "name" : "Ice Cream Day", "date" : ISODate("2017-07-16T00:00:00Z") }
```

- Join the Absences collection with 2018 holiday information from the Holidays collection

```
> db.Absences.aggregate([
...     {
...       $lookup:
...         {
...           from: "Holidays",
...           pipeline: [
...              { $match: { year: 2018 } },
...              { $project: { _id: 0, date: { name: "$name", date: "$date" } } },
...              { $replaceRoot: { newRoot: "$date" } }
...           ],
...           as: "holidays"
...         }
...     }
... ])
{ "_id" : 1, "student" : "Ann Aardvark", "sickdays" : [ ISODate("2018-05-01T00:00:00Z"), ISODate("2018-08-23T00:00:00Z
") ], "holidays" : [ { "name" : "New Years", "date" : ISODate("2018-01-01T00:00:00Z") }, { "name" : "Pi Day", "date" :
 ISODate("2018-03-14T00:00:00Z") }, { "name" : "Ice Cream Day", "date" : ISODate("2018-07-15T00:00:00Z") } ] }
{ "_id" : 2, "student" : "Zoe Zebra", "sickdays" : [ ISODate("2018-02-01T00:00:00Z"), ISODate("2018-05-23T00:00:00Z")
], "holidays" : [ { "name" : "New Years", "date" : ISODate("2018-01-01T00:00:00Z") }, { "name" : "Pi Day", "date" : IS
ODate("2018-03-14T00:00:00Z") }, { "name" : "Ice Cream Day", "date" : ISODate("2018-07-15T00:00:00Z") } ] }
```

- The equivalent SQL code for the same is
  - SELECT *, Holidays
  - FROM Absences
  - WHERE Holidays IN (SELECT name, date
  - FROM Holidays
  - WHERE year = 2018);

$sort

- Sorts all input documents and returns them to the pipeline in sorted order.
  - {$sort: {<field1>: <sort order>, <field2>: <sort order> ...}}
- $sort takes a document that specifies the field(s) to sort by and the respective sort order.
- <sort order> can have one of the following values:

- 1 to specify ascending order
- -1 to specify descending order

```
> db.sales.find()
{ "_id" : 1, "item" : "abc", "price" : 10, "quantity" : 2, "date" : ISODate("2014-03-01T08:00:00Z") }
{ "_id" : 2, "item" : "jkl", "price" : 20, "quantity" : 1, "date" : ISODate("2014-03-01T09:00:00Z") }
{ "_id" : 3, "item" : "xyz", "price" : 5, "quantity" : 10, "date" : ISODate("2014-03-15T09:00:00Z") }
{ "_id" : 4, "item" : "xyz", "price" : 5, "quantity" : 20, "date" : ISODate("2014-04-04T11:21:39.736Z") }
{ "_id" : 5, "item" : "abc", "price" : 10, "quantity" : 10, "date" : ISODate("2014-04-04T21:23:13.331Z") }
{ "_id" : 7, "item" : "jkl", "price" : 100, "quantity" : 10, "date" : ISODate("2019-02-06T04:15:06.041Z") }
{ "_id" : 6, "item" : "lmn", "price" : 20, "quantity" : 100, "date" : ISODate("2019-02-06T06:27:54.811Z") }
{ "_id" : 8, "item" : "abc", "price" : 10, "quantity" : 100, "date" : ISODate("2019-02-13T10:03:48.489Z") }
```

- Sort the documents in the sales collection, in descending order according to the price field and then in ascending order according to the value in the quantity field.

```
> db.sales.aggregate(
... [
... {$sort: {price: -1, quantity : 1}}
... ]
... )
{ "_id" : 7, "item" : "jkl", "price" : 100, "quantity" : 10, "date" : ISODate("2019-02-06T04:15:06.041Z") }
{ "_id" : 2, "item" : "jkl", "price" : 20, "quantity" : 1, "date" : ISODate("2014-03-01T09:00:00Z") }
{ "_id" : 6, "item" : "lmn", "price" : 20, "quantity" : 100, "date" : ISODate("2019-02-06T06:27:54.811Z") }
{ "_id" : 1, "item" : "abc", "price" : 10, "quantity" : 2, "date" : ISODate("2014-03-01T08:00:00Z") }
{ "_id" : 5, "item" : "abc", "price" : 10, "quantity" : 10, "date" : ISODate("2014-04-04T21:23:13.331Z") }
{ "_id" : 8, "item" : "abc", "price" : 10, "quantity" : 100, "date" : ISODate("2019-02-13T10:03:48.489Z") }
{ "_id" : 3, "item" : "xyz", "price" : 5, "quantity" : 10, "date" : ISODate("2014-03-15T09:00:00Z") }
{ "_id" : 4, "item" : "xyz", "price" : 5, "quantity" : 20, "date" : ISODate("2014-04-04T11:21:39.736Z") }
```

DB Aggregate Commands

$substrBytes

- Return the substring of a string.
  - **{$substrBytes: [<string expression>, <byte index>, <byte count>] }**
    - Create a three-byte menuCode from the name value

```
> db.food.find()
{ "_id" : 1, "name" : "apple" }
{ "_id" : 2, "name" : "banana" }
{ "_id" : 3, "name" : "éclair" }
{ "_id" : 4, "name" : "hamburger" }
{ "_id" : 5, "name" : "jalapeño" }
{ "_id" : 6, "name" : "pizza" }
{ "_id" : 7, "name" : "tacos" }
{ "_id" : 8, "name" : "□ □ sushi" }
> db.food.aggregate([{$project: {"name": 1,"menuCode": { $substrBytes: [ "$name", 0, 3]}}}])
{ "_id" : 1, "name" : "apple", "menuCode" : "app" }
{ "_id" : 2, "name" : "banana", "menuCode" : "ban" }
{ "_id" : 3, "name" : "éclair", "menuCode" : "éc" }
{ "_id" : 4, "name" : "hamburger", "menuCode" : "ham" }
{ "_id" : 5, "name" : "jalapeño", "menuCode" : "jal" }
{ "_id" : 6, "name" : "pizza", "menuCode" : "piz" }
{ "_id" : 7, "name" : "tacos", "menuCode" : "tac" }
{ "_id" : 8, "name" : "□ □ sushi", "menuCode" : "□ " }
```

    - Separate the quarter value (containing only single byte US-ASCII characters) into a yearSubstring and a quarterSubstring.

```
> db.items.find()
{ "_id" : 1, "item" : "ABC1", "quarter" : "13Q1", "description" : "product 1" }
{ "_id" : 2, "item" : "ABC2", "quarter" : "13Q4", "description" : "product 2" }
{ "_id" : 3, "item" : "XYZ1", "quarter" : "14Q2", "description" : null }
> db.items.aggregate([{
...       $project: {item: 1, yearSubstring: { $substrBytes: [ "$quarter", 0, 2 ] },
...         quarterSubtring: {$substrBytes: [
...             "$quarter", 2, { $subtract: [ { $strLenBytes: "$quarter" }, 2 ] }
...           ]}}}])
{ "_id" : 1, "item" : "ABC1", "yearSubstring" : "13", "quarterSubtring" : "Q1" }
{ "_id" : 2, "item" : "ABC2", "yearSubstring" : "13", "quarterSubtring" : "Q4" }
{ "_id" : 3, "item" : "XYZ1", "yearSubstring" : "14", "quarterSubtring" : "Q2" }
```

$ceil

- Return the smallest integer greater than or equal to the specified number.
  - **{$ceil: <number>}**
    - Return both the original value and the ceiling value

```
> db.samp.find()
{ "_id" : 1, "value" : 9.25 }
{ "_id" : 2, "value" : 8.73 }
{ "_id" : 3, "value" : 4.32 }
{ "_id" : 4, "value" : -5.34 }
> db.samp.aggregate([{ $project: { value: 1, ceilingValue: { $ceil: "$value" } } }])
{ "_id" : 1, "value" : 9.25, "ceilingValue" : 10 }
{ "_id" : 2, "value" : 8.73, "ceilingValue" : 9 }
{ "_id" : 3, "value" : 4.32, "ceilingValue" : 5 }
{ "_id" : 4, "value" : -5.34, "ceilingValue" : -5 }
```

$floor

- Returns the largest integer less than or equal to the specified number.
  - **{$floor: <number>}**
    - Return both the original value and the floored value

```
> db.samp.find()
{ "_id" : 1, "value" : 9.25 }
{ "_id" : 2, "value" : 8.73 }
{ "_id" : 3, "value" : 4.32 }
{ "_id" : 4, "value" : -5.34 }
{ "_id" : 5, "value" : -9.87 }
{ "_id" : 6, "value" : 8.9 }
{ "_id" : 7, "value" : -98.9 }
> db.samp.aggregate([{$project : {value : 1, floorvalue : {$floor : "$value"}}}])
{ "_id" : 1, "value" : 9.25, "floorvalue" : 9 }
{ "_id" : 2, "value" : 8.73, "floorvalue" : 8 }
{ "_id" : 3, "value" : 4.32, "floorvalue" : 4 }
{ "_id" : 4, "value" : -5.34, "floorvalue" : -6 }
{ "_id" : 5, "value" : -9.87, "floorvalue" : -10 }
{ "_id" : 6, "value" : 8.9, "floorvalue" : 8 }
{ "_id" : 7, "value" : -98.9, "floorvalue" : -99 }
```

$cmp

- Compares two values and returns
  - -1 if the first value is less than the second.
  - 1 if the first value is greater than the second.
  - 0 if the two values are equivalent.
    - **{$cmp: [<expression1>, <expression2>]}**
      - Compare the qty value with 250

```
> db.ord.find()
{ "_id" : 1, "item" : "abc1", "description" : "product 1", "qty" : 300 }
{ "_id" : 2, "item" : "abc2", "description" : "product 2", "qty" : 200 }
{ "_id" : 3, "item" : "xyz1", "description" : "product 3", "qty" : 250 }
{ "_id" : 4, "item" : "VWZ1", "description" : "product 4", "qty" : 300 }
{ "_id" : 5, "item" : "VWZ2", "description" : "product 5", "qty" : 180 }
> db.ord.aggregate([{
...         $project:{item: 1, qty: 1, cmpTo250: { $cmp: [ "$qty", 250 ] },_id: 0 }}])
{ "item" : "abc1", "qty" : 300, "cmpTo250" : 1 }
{ "item" : "abc2", "qty" : 200, "cmpTo250" : -1 }
{ "item" : "xyz1", "qty" : 250, "cmpTo250" : 0 }
{ "item" : "VWZ1", "qty" : 300, "cmpTo250" : 1 }
{ "item" : "VWZ2", "qty" : 180, "cmpTo250" : -1 }
```

$sum

- Calculates and returns the sum of numeric values. $sum ignores non-numeric values.
- $sum is available in the $group and $project stages.
- When used in the $group stage, it returns the collective sum of all the numeric values that result from applying a specified expression to each document in a group of documents that share the same group by key:
  - **{$sum: <expression>}**
- When used in the $project stage, $sum returns the sum of the specified expression or list of expressions for each document and has one of two syntaxes:
- $sum has one specified expression as its operand:
  - **{$sum: <expression>}**
- $sum has a list of specified expressions as its operand:
  - **{$sum: [<expression1>, <expression2> ...]}**
    - Group the documents by the day and the year of the date field, compute the total amount of quantity and the count for each group of documents.

```
> db.sales.aggregate(
...     [
...       {
...         $group:
...           {
...             _id: { day: { $dayOfYear: "$date"}, year: { $year: "$date" } },
...             totalAmount: { $sum: "$quantity" },
...             count: { $sum: 1 }
...           }
...       }
...     ]
... )
{ "_id" : { "day" : 37, "year" : 2019 }, "totalAmount" : 110, "count" : 2 }
{ "_id" : { "day" : 94, "year" : 2014 }, "totalAmount" : 30, "count" : 2 }
{ "_id" : { "day" : 74, "year" : 2014 }, "totalAmount" : 10, "count" : 1 }
{ "_id" : { "day" : 60, "year" : 2014 }, "totalAmount" : 3, "count" : 2 }
```

- Group the documents by the year of the date field, compute the total amount of quantity for each group of documents.

```
> db.sales.aggregate(
...     [
...       {
...         $group:
...           { _id: {year : {$year : "$date"}},
...             totalAmount: { $sum: "$quantity"}
...           }
...       }
...     ]
... )
{ "_id" : { "year" : 2019 }, "totalAmount" : 110 }
{ "_id" : { "year" : 2014 }, "totalAmount" : 43 }
```

- Calculate the total quiz scores, the total lab scores, and the total of the final and the midterm.

```
> db.marks.find()
{ "_id" : 1, "quizzes" : [ 10, 6, 7 ], "labs" : [ 5, 8 ], "final" : 80, "midterm" : 75 }
{ "_id" : 2, "quizzes" : [ 9, 10 ], "labs" : [ 8, 8 ], "final" : 95, "midterm" : 80 }
{ "_id" : 3, "quizzes" : [ 4, 5, 5 ], "labs" : [ 6, 5 ], "final" : 78, "midterm" : 70 }
> db.marks.aggregate([
...     {
...       $project: {
...         quizTotal: { $sum: "$quizzes"},
...         labTotal: { $sum: "$labs" },
...         examTotal: { $sum: [ "$final", "$midterm" ] }
...       }
...     }
... ])
{ "_id" : 1, "quizTotal" : 23, "labTotal" : 13, "examTotal" : 155 }
{ "_id" : 2, "quizTotal" : 19, "labTotal" : 16, "examTotal" : 175 }
{ "_id" : 3, "quizTotal" : 14, "labTotal" : 11, "examTotal" : 148 }
```

- Calculate the total quiz scores, the total lab scores, and the total of the final and the midterm with the total quiz scores and lab scores.

```
> db.marks.aggregate([
...   {$project :
...             {quiztotal : {$sum : "$quizzes"},
...             labtotal : {$sum : "$labs"},
...     midterm : 1,
...             final : 1,
...             total : {$sum : [{$sum : "$quizzes"}, {$sum: "$labs"}, "$midterm", "$final"]}
...            }
...   }
...  ]
... )
{ "_id" : 1, "final" : 80, "midterm" : 75, "quiztotal" : 23, "labtotal" : 13, "total" : 191 }
{ "_id" : 2, "final" : 95, "midterm" : 80, "quiztotal" : 19, "labtotal" : 16, "total" : 210 }
{ "_id" : 3, "final" : 78, "midterm" : 70, "quiztotal" : 14, "labtotal" : 11, "total" : 173 }
{ "_id" : 4, "midterm" : 90, "final" : 45, "quiztotal" : 19, "labtotal" : 19, "total" : 173 }
{ "_id" : 5, "midterm" : 89, "final" : 20, "quiztotal" : 19, "labtotal" : 9, "total" : 137 }
```

$add

- Adds numbers together or adds numbers and a date.
- If one of the arguments is a date, $add treats the other arguments as milliseconds to add to the date.
- It will not add array of numbers.
  - **{$add: [<expression1>, <expression2>, ...]}**
    - Calculate the total quiz scores, the total lab scores, and the total of the final and the midterm with the total quiz scores and lab scores.

```
> db.marks.find()
{ "_id" : 1, "quizzes" : [ 10, 6, 7 ], "labs" : [ 5, 8 ], "final" : 80, "midterm" : 75 }
{ "_id" : 2, "quizzes" : [ 9, 10 ], "labs" : [ 8, 8 ], "final" : 95, "midterm" : 80 }
{ "_id" : 3, "quizzes" : [ 4, 5, 5 ], "labs" : [ 6, 5 ], "final" : 78, "midterm" : 70 }
{ "_id" : 4, "quizzes" : [ 10, 9, 0 ], "labs" : [ 10, 9 ], "midterm" : 90, "final" : 45 }
{ "_id" : 5, "quizzes" : [ 10, 9 ], "labs" : [ 9 ], "midterm" : 89, "final" : 20 }
```

```
> db.marks.aggregate([
...   {$project :
...             {quiztotal : {$sum : "$quizzes"},
...             labtotal : {$sum : "$labs"},
...     midterm : 1,
...             final : 1,
...             total : {$add : [{$sum : "$quizzes"}, {$sum: "$labs"}, "$midterm", "$final"]}
...            }
...   }
...  ]
... )
{ "_id" : 1, "final" : 80, "midterm" : 75, "quiztotal" : 23, "labtotal" : 13, "total" : 191 }
{ "_id" : 2, "final" : 95, "midterm" : 80, "quiztotal" : 19, "labtotal" : 16, "total" : 210 }
{ "_id" : 3, "final" : 78, "midterm" : 70, "quiztotal" : 14, "labtotal" : 11, "total" : 173 }
{ "_id" : 4, "midterm" : 90, "final" : 45, "quiztotal" : 19, "labtotal" : 19, "total" : 173 }
{ "_id" : 5, "midterm" : 89, "final" : 20, "quiztotal" : 19, "labtotal" : 9, "total" : 137 }
```

$subtract

- Subtracts two numbers to return the difference, or two dates to return the difference in milliseconds, or a date and a number in milliseconds to return the resulting date.
  - **{$subtract: [<expression1>, <expression2>] }**
    - Compute the total by subtracting the discount from the subtotal of price and fee.

```
> db.sale.find()
{ "_id" : 1, "item" : "abc", "price" : 10, "fee" : 2, "discount" : 5, "date" : ISODate("2014-03-01T08:00:00Z") }
{ "_id" : 2, "item" : "jkl", "price" : 20, "fee" : 1, "discount" : 2, "date" : ISODate("2014-03-01T09:00:00Z") }
{ "_id" : 3, "item" : "abc", "price" : 120, "fee" : 3, "discount" : 15, "date" : ISODate("2015-03-02T08:00:00Z") }
{ "_id" : 4, "item" : "jkl", "price" : 200, "fee" : 4, "discount" : 12, "date" : ISODate("2016-03-04T09:00:00Z") }
{ "_id" : 5, "item" : "abc", "price" : 190, "fee" : 21, "discount" : 50, "date" : ISODate("2017-05-12T08:00:00Z") }
{ "_id" : 6, "item" : "jkl", "price" : 220, "fee" : 11, "discount" : 20, "date" : ISODate("2018-07-02T09:00:00Z") }
{ "_id" : 7, "item" : "abc", "price" : 170, "fee" : 200, "discount" : 5, "date" : ISODate("2019-02-13T07:00:25.583Z")
}
```

```
> db.sale.aggregate( [
...          { $project: { item: 1,
...                        total:
...                             { $subtract: [ { $add: [ "$price", "$fee" ] }, "$discount" ] }
...                      }
...          }
... ]
... )
{ "_id" : 1, "item" : "abc", "total" : 7 }
{ "_id" : 2, "item" : "jkl", "total" : 19 }
{ "_id" : 3, "item" : "abc", "total" : 108 }
{ "_id" : 4, "item" : "jkl", "total" : 192 }
{ "_id" : 5, "item" : "abc", "total" : 161 }
{ "_id" : 6, "item" : "jkl", "total" : 211 }
{ "_id" : 7, "item" : "abc", "total" : 365 }
```

- Subtract date from the current date and returns the difference in milliseconds

```
> db.sales.aggregate(
... [
...     {$project: {
...                 item: 1,
...                 dateDifference:
...                 { $subtract: [ new Date(), "$date" ] }
...                   }
...      }
... ]
... )
```

```
{ "_id" : 1, "item" : "abc", "dateDifference" : NumberLong("156392108994") }
{ "_id" : 2, "item" : "jkl", "dateDifference" : NumberLong("156388508994") }
{ "_id" : 3, "item" : "xyz", "dateDifference" : NumberLong("155178908994") }
{ "_id" : 4, "item" : "xyz", "dateDifference" : NumberLong("153442409258") }
{ "_id" : 5, "item" : "abc", "dateDifference" : NumberLong("153406315663") }
{ "_id" : 7, "item" : "jkl", "dateDifference" : NumberLong(626402953) }
{ "_id" : 6, "item" : "lmn", "dateDifference" : NumberLong(618434183) }
{ "_id" : 8, "item" : "abc", "dateDifference" : NumberLong(680505) }
```

$multiply

- Multiplies numbers together and returns the result.
- Pass the arguments to $multiply in an array.
  - **{$multiply: [<expression1>, <expression2>, ...]}**

- Calculate the total amount of every item.

```
> db.sales.find()
{ "_id" : 1, "item" : "abc", "price" : 10, "quantity" : 2, "date" : ISODate("2014-03-01T08:00:00Z") }
{ "_id" : 2, "item" : "jkl", "price" : 20, "quantity" : 1, "date" : ISODate("2014-03-01T09:00:00Z") }
{ "_id" : 3, "item" : "xyz", "price" : 5, "quantity" : 10, "date" : ISODate("2014-03-15T09:00:00Z") }
{ "_id" : 4, "item" : "xyz", "price" : 5, "quantity" : 20, "date" : ISODate("2014-04-04T11:21:39.736Z") }
{ "_id" : 5, "item" : "abc", "price" : 10, "quantity" : 10, "date" : ISODate("2014-04-04T21:23:13.331Z") }
{ "_id" : 7, "item" : "jkl", "price" : 100, "quantity" : 10, "date" : ISODate("2019-02-06T04:15:06.041Z") }
> db.sales.aggregate(    [      { $project: { date: 1, item: 1, total: { $multiply: [ "$price", "$quantity" ] } } }
 ] )
{ "_id" : 1, "item" : "abc", "date" : ISODate("2014-03-01T08:00:00Z"), "total" : 20 }
{ "_id" : 2, "item" : "jkl", "date" : ISODate("2014-03-01T09:00:00Z"), "total" : 20 }
{ "_id" : 3, "item" : "xyz", "date" : ISODate("2014-03-15T09:00:00Z"), "total" : 50 }
{ "_id" : 4, "item" : "xyz", "date" : ISODate("2014-04-04T11:21:39.736Z"), "total" : 100 }
{ "_id" : 5, "item" : "abc", "date" : ISODate("2014-04-04T21:23:13.331Z"), "total" : 100 }
{ "_id" : 7, "item" : "jkl", "date" : ISODate("2019-02-06T04:15:06.041Z"), "total" : 1000 }
```

```
> db.sales.find()
{ "_id" : 1, "item" : "abc", "price" : 10, "quantity" : 2, "date" : ISODate("2014-03-01T08:00:00Z") }
{ "_id" : 2, "item" : "jkl", "price" : 20, "quantity" : 1, "date" : ISODate("2014-03-01T09:00:00Z") }
{ "_id" : 3, "item" : "xyz", "price" : 5, "quantity" : 10, "date" : ISODate("2014-03-15T09:00:00Z") }
{ "_id" : 4, "item" : "xyz", "price" : 5, "quantity" : 20, "date" : ISODate("2014-04-04T11:21:39.736Z") }
{ "_id" : 5, "item" : "abc", "price" : 10, "quantity" : 10, "date" : ISODate("2014-04-04T21:23:13.331Z") }
{ "_id" : 7, "item" : "jkl", "price" : 100, "quantity" : 10, "date" : ISODate("2019-02-06T04:15:06.041Z") }
{ "_id" : 6, "item" : "lmn", "price" : 20, "quantity" : 100, "date" : ISODate("2019-02-06T06:27:54.811Z") }
> db.sales.aggregate([  {$group :{_id :"$_id",             cost: {$sum : {$multiply : ["$price", "$quantity"]}}  }}])
{ "_id" : 6, "cost" : 2000 }
{ "_id" : 7, "cost" : 1000 }
{ "_id" : 3, "cost" : 50 }
{ "_id" : 2, "cost" : 20 }
{ "_id" : 5, "cost" : 100 }
{ "_id" : 1, "cost" : 20 }
{ "_id" : 4, "cost" : 100 }
```

$divide

- Divide one number by another and return the result.
- Pass the arguments to $divide in an array.
  - **{$divide: [<expression1>, <expression2>]}**
    - Divide the hours field by a literal 8 to compute the number of work days.

```
> db.plan.find()
{ "_id" : 1, "name" : "A", "hours" : 80, "resources" : 7 }
{ "_id" : 2, "name" : "B", "hours" : 40, "resources" : 4 }
{ "_id" : 3, "name" : "C", "hours" : 180, "resources" : 3 }
{ "_id" : 4, "name" : "D", "hours" : 45, "resources" : 5 }
{ "_id" : 5, "name" : "E", "hours" : 90, "resources" : 5 }
{ "_id" : 6, "name" : "F", "hours" : 30, "resources" : 4 }
{ "_id" : 7, "name" : "G", "hours" : 280, "resources" : 9 }
{ "_id" : 8, "name" : "H", "hours" : 55, "resources" : 6 }
> db.plan.aggregate([{ $project: { name: 1, workdays: { $divide: [ "$hours", 8 ] } } }])
{ "_id" : 1, "name" : "A", "workdays" : 10 }
{ "_id" : 2, "name" : "B", "workdays" : 5 }
{ "_id" : 3, "name" : "C", "workdays" : 22.5 }
{ "_id" : 4, "name" : "D", "workdays" : 5.625 }
{ "_id" : 5, "name" : "E", "workdays" : 11.25 }
{ "_id" : 6, "name" : "F", "workdays" : 3.75 }
{ "_id" : 7, "name" : "G", "workdays" : 35 }
{ "_id" : 8, "name" : "H", "workdays" : 6.875 }
```

- Subtract date from the current date and returns the difference in hours

```
> db.sales.find()
{ "_id" : 1, "item" : "abc", "price" : 10, "quantity" : 2, "date" : ISODate("2014-03-01T08:00:00Z") }
{ "_id" : 2, "item" : "jkl", "price" : 20, "quantity" : 1, "date" : ISODate("2014-03-01T09:00:00Z") }
{ "_id" : 3, "item" : "xyz", "price" : 5, "quantity" : 10, "date" : ISODate("2014-03-15T09:00:00Z") }
{ "_id" : 4, "item" : "xyz", "price" : 5, "quantity" : 20, "date" : ISODate("2014-04-04T11:21:39.736Z") }
{ "_id" : 5, "item" : "abc", "price" : 10, "quantity" : 10, "date" : ISODate("2014-04-04T21:23:13.331Z") }
{ "_id" : 7, "item" : "jkl", "price" : 100, "quantity" : 10, "date" : ISODate("2019-02-06T04:15:06.041Z") }
{ "_id" : 6, "item" : "lmn", "price" : 20, "quantity" : 100, "date" : ISODate("2019-02-06T06:27:54.811Z") }
{ "_id" : 8, "item" : "abc", "price" : 10, "quantity" : 100, "date" : ISODate("2019-02-13T10:03:48.489Z") }
```

```
> db.sales.aggregate([
... {$project : {
...     item : 1,
...     datediff :
...        {$divide: [{$subtract : [ISODate(), "$date"]}, 360000]}
... }
... }
... ])
{ "_id" : 1, "item" : "abc", "datediff" : 434420.6712222222 }
{ "_id" : 2, "item" : "jkl", "datediff" : 434410.6712222222 }
{ "_id" : 3, "item" : "xyz", "datediff" : 431050.6712222222 }
{ "_id" : 4, "item" : "xyz", "datediff" : 426227.06084444444 }
{ "_id" : 5, "item" : "abc", "datediff" : 426126.80085833336 }
{ "_id" : 7, "item" : "jkl", "datediff" : 1738.1544416666666 }
{ "_id" : 6, "item" : "lmn", "datediff" : 1716.0189694444443 }
{ "_id" : 8, "item" : "abc", "datediff" : 0.03653055555555555 }
```

$filter

- Select a subset of an array to return based on the specified condition.
  - **{$filter: {input: <array>, as: <string>, cond: <expression>}}**
    - input - An expression that resolves to an array.
    - as - A name for the variable that represents each individual element of the input array.
      - If no name is specified, the variable name defaults to this.
    - cond - Expression that resolves to a boolean value.
      - Used to determine if an element should be included in the output array.
      - Expression references each element of the input array individually with the variable name specified in as.
  - Returns an array with only those elements that match the condition.
  - The returned elements are in the original order.
    - Filter the items array to only include documents that have a price greater than or equal to 100

```
> db.Saldat.find()
{ "_id" : 0, "items" : [ { "item_id" : 43, "quantity" : 2, "price" : 10 }, { "item_id" : 2, "quantity" : 1, "price" :
240 } ] }
{ "_id" : 1, "items" : [ { "item_id" : 23, "quantity" : 3, "price" : 110 }, { "item_id" : 103, "quantity" : 4, "price"
 : 5 }, { "item_id" : 38, "quantity" : 1, "price" : 300 } ] }
{ "_id" : 2, "items" : [ { "item_id" : 4, "quantity" : 1, "price" : 23 } ] }
> db.Saldat.aggregate([{$project: {items: {$filter: {input: "$items", as: "item", cond: { $gte: [ "$$item.price", 100
] }}}}}])
{ "_id" : 0, "items" : [ { "item_id" : 2, "quantity" : 1, "price" : 240 } ] }
{ "_id" : 1, "items" : [ { "item_id" : 23, "quantity" : 3, "price" : 110 }, { "item_id" : 38, "quantity" : 1, "price"
: 300 } ] }
{ "_id" : 2, "items" : [ ] }
```

- Filter the marks array to only include documents that have a mark lesser than 40.

```
> db.grade.aggregate(
... [
...  {$project :
...  {name: 1,
...  scores :
... {$filter :
... {input: "$scores", as: "score",
... cond: {$lt: ["$$score" , 40] }
... }
... }
... }
... }])
```

```
{ "_id" : 1, "name" : "Akshay K", "scores" : [ ] }
{ "_id" : 2, "name" : "Bobby D", "scores" : [ ] }
{ "_id" : 3, "name" : "Charles D", "scores" : [ ] }
{ "_id" : 4, "name" : "Dharma E", "scores" : [ ] }
{ "_id" : 5, "name" : "Eshwar K", "scores" : [ 31 ] }
{ "_id" : 6, "name" : "Fanna L", "scores" : [ 21, 24 ] }
{ "_id" : 7, "name" : "Ganga G", "scores" : [ ] }
{ "_id" : 8, "name" : "Hastha S", "scores" : [ ] }
{ "_id" : 9, "name" : "Ion M", "scores" : [ 14 ] }
{ "_id" : 10, "name" : "Jack N", "scores" : [ 10, 10, 10 ] }
```

- Filter the marks array to only include documents that have a mark lesser than 40 and create a new collection called student_fail.

```
> db.grade.aggregate(
... [
...  {$project :
...   {name: 1,
...    scores :
...     {$filter :
...       {input: "$scores", as: "score",
... cond: {$lt: ["$$score" , 40] }
...       }
...     }
...   }
... },
... {$out: "student_fail"}
... ]
... )
```

```
> db.student_fail.find()
{ "_id" : 1, "name" : "Akshay K", "scores" : [ ] }
{ "_id" : 2, "name" : "Bobby D", "scores" : [ ] }
{ "_id" : 3, "name" : "Charles D", "scores" : [ ] }
{ "_id" : 4, "name" : "Dharma E", "scores" : [ ] }
{ "_id" : 5, "name" : "Eshwar K", "scores" : [ 31 ] }
{ "_id" : 6, "name" : "Fanna L", "scores" : [ 21, 24 ] }
{ "_id" : 7, "name" : "Ganga G", "scores" : [ ] }
{ "_id" : 8, "name" : "Hastha S", "scores" : [ ] }
{ "_id" : 9, "name" : "Ion M", "scores" : [ 14 ] }
{ "_id" : 10, "name" : "Jack N", "scores" : [ 10, 10, 10 ] }
```

Date Options

- Return the specific option for a date
  - year: {$year: "$<date expression>"}
  - month: {$month: "$<date expression>"}
  - day: {$dayOfMonth: "$<date expression>"}
  - hour: {$hour: "$<date expression>"}
  - minutes: {$minute: "$<date expression>"}
  - seconds: {$second: "$<date expression>"}
  - milliseconds: {$millisecond: "$<date expression>"}
  - dayOfYear: {$dayOfYear: "$<date expression>"}
  - dayOfWeek: {$dayOfWeek: "$<date expression>"}
  - week: {$week: "$<date expression>"}

```
> db.sales.aggregate(
...   [
...     {
...       $project:
...         {
...           year: { $year: "$date" },
...           month: { $month: "$date" },
...           day: { $dayOfMonth: "$date" },
...           hour: { $hour: "$date" },
...           minutes: { $minute: "$date" },
...           seconds: { $second: "$date" },
...           milliseconds: { $millisecond: "$date" },
...           dayOfYear: { $dayOfYear: "$date" },
...           dayOfWeek: { $dayOfWeek: "$date" },
...           week: { $week: "$date" }
...         }
...     }
...   ]
... )
```

$type

- Returns a string that specifies the BSON type of the argument.
  - { $type: <expression> }
    - If the argument is a field that is missing in the input document, $type returns the string "missing".

```
> db.Mycol.find()
{ "_id" : 0, "a" : 8 }
{ "_id" : 1, "a" : [ 41.63, 88.19 ] }
{ "_id" : 2, "a" : { "a" : "apple", "b" : "banana", "c" : "carrot" } }
{ "_id" : 3, "a" : "caribou" }
{ "_id" : 4, "a" : NumberLong(71) }
{ "_id" : 5 }
> db.Mycol.aggregate([{
...     $project: {
...        a : { $type: "$a" }
...     }
... }])
{ "_id" : 0, "a" : "double" }
{ "_id" : 1, "a" : "array" }
{ "_id" : 2, "a" : "object" }
{ "_id" : 3, "a" : "string" }
{ "_id" : 4, "a" : "long" }
{ "_id" : 5, "a" : "missing" }
```

$max

- Returns the maximum value. $max compares both value and type, using the specified BSON comparison order for values of different types.
- When used in the $group stage, $max has the following syntax and returns the maximum value that results from applying an expression to each document in a group of documents that share the same group by key:
  - {$max: <expression>}
- When used in the $project stage, $max returns the maximum of the specified expression or list of expressions for each document and has one of two syntaxes:
  - $max has one specified expression as its operand:

- {$max: <expression>}
- $max has a list of specified expressions as its operand:
  - {max: [<expression1>, <expression2> ...]}

- Calculate the maximum quiz scores, the maximum lab scores, and the maximum of the final and the midterm

```
> db.marks.aggregate([
...     {
...       $project: {
...         quizMax: { $max: "$quizzes"},
...         labMax: { $max: "$labs" },
...         examMax: { $max: [ "$final", "$midterm" ] }
...       }
...     }
... ])
{ "_id" : 1, "quizMax" : 10, "labMax" : 8, "examMax" : 80 }
{ "_id" : 2, "quizMax" : 10, "labMax" : 8, "examMax" : 95 }
{ "_id" : 3, "quizMax" : 5, "labMax" : 6, "examMax" : 78 }
{ "_id" : 4, "quizMax" : 10, "labMax" : 10, "examMax" : 90 }
{ "_id" : 5, "quizMax" : 10, "labMax" : 9, "examMax" : 89 }
```

- Compute the maximum amount and maximum quantity for each grouping

```
> db.sales.aggregate(
...     [
...       {
...         $group:
...           {
...             _id: "$item",
...             maxTotalAmount: { $max: { $multiply: [ "$price", "$quantity" ] } },
...             maxQuantity: { $max: "$quantity" }
...           }
...       }
...     ]
... )
{ "_id" : "lmn", "maxTotalAmount" : 2000, "maxQuantity" : 100 }
{ "_id" : "xyz", "maxTotalAmount" : 100, "maxQuantity" : 20 }
{ "_id" : "jkl", "maxTotalAmount" : 1000, "maxQuantity" : 10 }
{ "_id" : "abc", "maxTotalAmount" : 1000, "maxQuantity" : 100 }
```

$min

- Returns the minimum value. $min compares both value and type, using the specified BSON comparison order for values of different types.
- When used in the $group stage, $min has the following syntax and returns the minimum value that results from applying an expression to each document in a group of documents that share the same group by key
  - {$min: <expression>}
- When used in the $project stage, $min returns the minimum of the specified expression or list of expressions for each document and has one of two syntaxes:
  - $min has one specified expression as its operand
    - {$min: <expression>}
  - $min has a list of specified expressions as its operand

- {$min: [<expression1>, <expression2> ...]}

- Calculate the minimum quiz scores, the minimum lab scores, and the minimum of the final and the midterm.

```
> db.marks.aggregate([
...     {
...       $project: {
...         quizMin: { $min: "$quizzes"},
...         labMin: { $min: "$labs" },
...         examMin: { $min: [ "$final", "$midterm" ] }
...       }
...     }
... ])
{ "_id" : 1, "quizMin" : 6, "labMin" : 5, "examMin" : 75 }
{ "_id" : 2, "quizMin" : 9, "labMin" : 8, "examMin" : 80 }
{ "_id" : 3, "quizMin" : 4, "labMin" : 5, "examMin" : 70 }
{ "_id" : 4, "quizMin" : 0, "labMin" : 9, "examMin" : 45 }
{ "_id" : 5, "quizMin" : 9, "labMin" : 9, "examMin" : 20 }
```

- Compute the minimum amount and minimum quantity for each grouping

```
> db.sales.aggregate(
...     [
...       {
...         $group:
...           {
...             _id: "$item",
...             minQuantity: { $min: "$quantity" }
...           }
...       }
...     ]
... )
{ "_id" : "lmn", "minQuantity" : 100 }
{ "_id" : "xyz", "minQuantity" : 10 }
{ "_id" : "jkl", "minQuantity" : 1 }
{ "_id" : "abc", "minQuantity" : 2 }
```

$switch

- Evaluates a series of case expressions.
- When it finds an expression which evaluates to true, $switch executes a specified expression and breaks out of the control flow.
- $switch has the following syntax

```
$switch: {
   branches: [
      { case: <expression>, then: <expression> },
      { case: <expression>, then: <expression> },
      ...
   ],
   default: <expression>
}
```

- Display a particular message based on each student's average score.

```
> db.grade.find()
{ "_id" : 1, "name" : "Akshay K", "scores" : [ 87, 86, 78 ] }
{ "_id" : 2, "name" : "Bobby D", "scores" : [ 71, 64, 81 ] }
{ "_id" : 3, "name" : "Charles D", "scores" : [ 91, 84, 97 ] }
{ "_id" : 4, "name" : "Dharma E", "scores" : [ 57, 56, 78 ] }
{ "_id" : 5, "name" : "Eshwar K", "scores" : [ 31, 84, 81 ] }
{ "_id" : 6, "name" : "Fanna L", "scores" : [ 21, 24, 97 ] }
{ "_id" : 7, "name" : "Ganga G", "scores" : [ 67, 96, 78 ] }
{ "_id" : 8, "name" : "Hastha S", "scores" : [ 81, 64, 81 ] }
{ "_id" : 9, "name" : "Ion M", "scores" : [ 91, 14, 97 ] }
```

```
> db.grade.aggregate( [
...   {
...     $project:
...       {
...         "name" : 1,
...         "summary" :
...         {
...           $switch:
...             {
...               branches: [
...                 { case: { $gte : [ { $avg : "$scores" }, 90 ] },
...                   then: "Doing great!" },
...                 { case: { $and : [ { $gte : [ { $avg : "$scores" }, 80 ] },
...                                    { $lt : [ { $avg : "$scores" }, 90 ] } ] },
...                   then: "Doing pretty well." },
...                 { case: { $lt : [ { $avg : "$scores" }, 80 ] },
...                   then: "Needs improvement." }
...               ],
...               default: "No scores found."
...             }
...         }
...       }
...   }
... ] )
```

```
{ "_id" : 1, "name" : "Akshay K", "summary" : "Doing pretty well." }
{ "_id" : 2, "name" : "Bobby D", "summary" : "Needs improvement." }
{ "_id" : 3, "name" : "Charles D", "summary" : "Doing great!" }
{ "_id" : 4, "name" : "Dharma E", "summary" : "Needs improvement." }
{ "_id" : 5, "name" : "Eshwar K", "summary" : "Needs improvement." }
{ "_id" : 6, "name" : "Fanna L", "summary" : "Needs improvement." }
{ "_id" : 7, "name" : "Ganga G", "summary" : "Doing pretty well." }
{ "_id" : 8, "name" : "Hastha S", "summary" : "Needs improvement." }
{ "_id" : 9, "name" : "Ion M", "summary" : "Needs improvement." }
```

- Display a particular message based on the total amount of sales.

- Create a collection called sales_details that consists of the id, item name and total_amt.

```
> db.sales.find()
{ "_id" : 1, "item" : "abc", "price" : 10, "quantity" : 2, "date" : ISODate("2014-03-01T08:00:00Z") }
{ "_id" : 2, "item" : "jkl", "price" : 20, "quantity" : 1, "date" : ISODate("2014-03-01T09:00:00Z") }
{ "_id" : 3, "item" : "xyz", "price" : 5, "quantity" : 10, "date" : ISODate("2014-03-15T09:00:00Z") }
{ "_id" : 4, "item" : "xyz", "price" : 5, "quantity" : 20, "date" : ISODate("2014-04-04T11:21:39.736Z") }
{ "_id" : 5, "item" : "abc", "price" : 10, "quantity" : 10, "date" : ISODate("2014-04-04T21:23:13.331Z") }
{ "_id" : 7, "item" : "jkl", "price" : 100, "quantity" : 10, "date" : ISODate("2019-02-06T04:15:06.041Z") }
{ "_id" : 6, "item" : "lmn", "price" : 20, "quantity" : 100, "date" : ISODate("2019-02-06T06:27:54.811Z") }
{ "_id" : 8, "item" : "abc", "price" : 10, "quantity" : 100, "date" : ISODate("2019-02-13T10:03:48.489Z") }
```

```
> db.sales.aggregate([ {$project: {item:1, total_amt : {$multiply: ["$price", "$quantity"]}}}, {$out: "sales_details"}
])
```

```
> db.sales_details.find()
{ "_id" : 1, "item" : "abc", "total_amt" : 20 }
{ "_id" : 2, "item" : "jkl", "total_amt" : 20 }
{ "_id" : 3, "item" : "xyz", "total_amt" : 50 }
{ "_id" : 4, "item" : "xyz", "total_amt" : 100 }
{ "_id" : 5, "item" : "abc", "total_amt" : 100 }
{ "_id" : 7, "item" : "jkl", "total_amt" : 1000 }
{ "_id" : 6, "item" : "lmn", "total_amt" : 2000 }
{ "_id" : 8, "item" : "abc", "total_amt" : 1000 }
```

```
> db.sales_details.aggregate([
... {$project:
... {item : 1,
... summary :
... {
... $switch:
... {
...  branches : [
... {case : {$gte : ["$total_amt", 1000]},
...  then : "Too Costly"},
... {case : {$and : [{$gte : ["$total_amt", 750]},
...                  {$lt : ["$total_amt", 1000]}]},
...  then : "Should be reduced"},
... {case : {$and: [{$gte: ["$total_amt", 500]},
...                 {$lt: ["$total_amt", 750]}]},
...  then : "Can be reduced"},
... {case: {$and: [{$gte : ["$total_amt",1]},
...                {$lt : ["$total_amt", 500]}]},
...  then: "Affordable"}
... ],
... default: "Cannot Exist"
... }
... }
... }
... }
... ])
```

```
{ "_id" : 1, "item" : "abc", "summary" : "Affordable" }
{ "_id" : 2, "item" : "jkl", "summary" : "Affordable" }
{ "_id" : 3, "item" : "xyz", "summary" : "Affordable" }
{ "_id" : 4, "item" : "xyz", "summary" : "Affordable" }
{ "_id" : 5, "item" : "abc", "summary" : "Affordable" }
{ "_id" : 7, "item" : "jkl", "summary" : "Too Costly" }
{ "_id" : 6, "item" : "lmn", "summary" : "Too Costly" }
{ "_id" : 8, "item" : "abc", "summary" : "Too Costly" }
```

- Display a particular message based on the total amount of sales and create another collection.

```
> db.sales_details.aggregate([
... {$project:
...  {item : 1,
...  summary :
...  {
...  $switch:
...  {
...   branches : [
...  {case : {$gte : ["$total_amt", 1000]},
...   then : "Too Costly"},
...  {case : {$and : [{$gte : ["$total_amt", 750]},
...                   {$lt : ["$total_amt", 1000]}]},
...   then : "Should be reduced"},
...  {case : {$and: [{$gte: ["$total_amt", 500]},
...                  {$lt: ["$total_amt", 750]}]},
...   then : "Can be reduced"},
...  {case: {$and: [{$gte : ["$total_amt",1]},
...                 {$lt : ["$total_amt", 500]}]},
...   then: "Affordable"}
...   ],
...  default: "Cannot Exist"
...  }
...  }
... }
... }, {$out: "sales_details"}])
```

```
> db.sales_details.find()
{ "_id" : 1, "item" : "abc", "summary" : "Affordable" }
{ "_id" : 2, "item" : "jkl", "summary" : "Affordable" }
{ "_id" : 3, "item" : "xyz", "summary" : "Affordable" }
{ "_id" : 4, "item" : "xyz", "summary" : "Affordable" }
{ "_id" : 5, "item" : "abc", "summary" : "Affordable" }
{ "_id" : 7, "item" : "jkl", "summary" : "Too Costly" }
{ "_id" : 6, "item" : "lmn", "summary" : "Too Costly" }
{ "_id" : 8, "item" : "abc", "summary" : "Too Costly" }
```

$cond

- Evaluates a boolean expression to return one of the two specified return expressions.
- The $cond expression has one of two syntaxes:
  - **{$cond: {if: <boolean-expression>, then: <true-case>, else: <false-case->}}**
- or
  - **{$cond: [<boolean-expression>, <true-case>, <false-case>]}**
- If the <boolean-expression> evaluates to true, then $cond evaluates and returns the value of the <true-case> expression.
- Otherwise, $cond evaluates and returns the value of the <false-case> expression.
  - Set the discount value to 30 if qty value is greater than or equal to 250 and to 20 if qty value is less than 250.

```
> db.inven.find()
{ "_id" : 1, "item" : "abc1", "description" : "product 1", "qty" : 300 }
{ "_id" : 2, "item" : "abc2", "description" : "product 2", "qty" : 200 }
{ "_id" : 3, "item" : "xyz1", "description" : "product 3", "qty" : 250 }
{ "_id" : 4, "item" : "VWZ1", "description" : "product 4", "qty" : 300 }
{ "_id" : 5, "item" : "VWZ2", "description" : "product 5", "qty" : 180 }
```

```
> db.inven.aggregate(
...    [
...       {
...          $project:
...            {
...              item: 1,
...              discount:
...                {
...                  $cond: { if: { $gte: [ "$qty", 250 ] }, then: 30, else: 20 }
...                }
...            }
...       }
...    ]
... )
```

```
{ "_id" : 1, "item" : "abc1", "discount" : 30 }
{ "_id" : 2, "item" : "abc2", "discount" : 20 }
{ "_id" : 3, "item" : "xyz1", "discount" : 30 }
{ "_id" : 4, "item" : "VWZ1", "discount" : 30 }
{ "_id" : 5, "item" : "VWZ2", "discount" : 20 }
```

DB Commands – Count

- Returns the count of documents that would match a find() query for the collection or view.
  - db.<collection_name>.count(query, options)
    - The db.<collection_name>.count() method does not perform the find() operation but instead counts and returns the number of results that match a query.
    - The options can be
      - limit – the maximum number of documents to count
      - skip – the number of documents to skip before counting

- hint – an index name hint
- maxTimeMS – the maximum amount of time to allow the query to run

- Count the number of all documents

```
> db.sales.find()
{ "_id" : 1, "item" : "abc", "price" : 10, "quantity" : 2, "date" : ISODate("2014-03-01T08:00:00Z") }
{ "_id" : 2, "item" : "jkl", "price" : 20, "quantity" : 1, "date" : ISODate("2014-03-01T09:00:00Z") }
{ "_id" : 3, "item" : "xyz", "price" : 5, "quantity" : 10, "date" : ISODate("2014-03-15T09:00:00Z") }
{ "_id" : 4, "item" : "xyz", "price" : 5, "quantity" : 20, "date" : ISODate("2014-04-04T11:21:39.736Z") }
{ "_id" : 5, "item" : "abc", "price" : 10, "quantity" : 10, "date" : ISODate("2014-04-04T21:23:13.331Z") }
> db.sales.count()
5
```

- Count the number of the documents with the field ord_dt greater than new Date('01/01/2013')

```
> db.orders.find()
{ "_id" : 1, "cust_id" : "abc1", "ord_date" : ISODate("2012-11-02T17:04:11.102Z"), "status" : "A", "amount" : 50 }
{ "_id" : 2, "cust_id" : "xyz1", "ord_date" : ISODate("2013-10-01T17:04:11.102Z"), "status" : "A", "amount" : 100 }
{ "_id" : 3, "cust_id" : "xyz1", "ord_date" : ISODate("2013-10-12T17:04:11.102Z"), "status" : "D", "amount" : 25 }
{ "_id" : 4, "cust_id" : "xyz1", "ord_date" : ISODate("2013-10-11T17:04:11.102Z"), "status" : "D", "amount" : 125 }
{ "_id" : 5, "cust_id" : "abc1", "ord_date" : ISODate("2013-11-12T17:04:11.102Z"), "status" : "A", "amount" : 25 }
> db.orders.count( { ord_date: { $gt: new Date('01/11/2013') } } )
4
```

DB Commands – distinct

- Finds the distinct values for a specified field across a single collection or view and return the results in an array.
  - **db.<collection_name>.distinct(field, query, options)**
    - Return the distinct values for the field dept from all documents

```
> db.cloth.find()
{ "_id" : 1, "dept" : "A", "item" : { "sku" : "111", "color" : "red" }, "sizes" : [ "S", "M" ] }
{ "_id" : 2, "dept" : "A", "item" : { "sku" : "111", "color" : "blue" }, "sizes" : [ "M", "L" ] }
{ "_id" : 3, "dept" : "B", "item" : { "sku" : "222", "color" : "blue" }, "sizes" : "S" }
{ "_id" : 4, "dept" : "A", "item" : { "sku" : "333", "color" : "black" }, "sizes" : [ "S" ] }
> db.cloth.distinct( "dept" )
[ "A", "B" ]
```

- Embedded documents
  - Return the distinct values for the field sku, embedded in the item field

```
> db.cloth.distinct( "item.sku" )
[ "111", "222", "333" ]
```

  - Return the distinct values for the field sizes from all documents in the cloth collection.

```
> db.cloth.find()
{ "_id" : 1, "dept" : "A", "item" : { "sku" : "111", "color" : "red" }, "sizes" : [ "S", "M" ] }
{ "_id" : 2, "dept" : "A", "item" : { "sku" : "111", "color" : "blue" }, "sizes" : [ "M", "L" ] }
{ "_id" : 3, "dept" : "B", "item" : { "sku" : "222", "color" : "blue" }, "sizes" : "S" }
{ "_id" : 4, "dept" : "A", "item" : { "sku" : "333", "color" : "black" }, "sizes" : [ "S" ] }
> db.cloth.distinct( "sizes" )
[ "M", "S", "L" ]
```

  - Return the distinct values for the field sku, embedded in the item field, from the documents whose dept is equal to "A"

```
> db.cloth.find()
{ "_id" : 1, "dept" : "A", "item" : { "sku" : "111", "color" : "red" }, "sizes" : [ "S", "M" ] }
{ "_id" : 2, "dept" : "A", "item" : { "sku" : "111", "color" : "blue" }, "sizes" : [ "M", "L" ] }
{ "_id" : 3, "dept" : "B", "item" : { "sku" : "222", "color" : "blue" }, "sizes" : "S" }
{ "_id" : 4, "dept" : "A", "item" : { "sku" : "333", "color" : "black" }, "sizes" : [ "S" ] }
> db.cloth.distinct( "item.sku", { dept: "A" } )
[ "111", "333" ]
```

Administrative Commands

- Rename an existing collection
  - **db.adminCommand({ renameCollection: "<db_name>.<collection_name>", to: "<db>.<newcollection_name>" })**
  - **db.collection.renameCollection("<new_name>")**
- Drop a database
  - **use <database_name>**
  - **db.runCommand({dropDatabase: 1})**
- Create a collection
  - **db.runCommand({ create: "<collection_name", capped : <Boolean>, size: <bytes>})**
- Drop a collection given the collection_name
  - **db.<collection_name>. drop()**