Mongodb

Important topic :

1 . SQL TO MONGODB mapping chart :
https://docs.mongodb.com/manual/reference/sql-comparison/

2 . match , group, project , sum
https://docs.mongodb.com/manual/reference/sql-aggregation-comparison/

…………………………………………...

In smartlotto server side project :

……………………………………………………………….

Create and alter table of mysql and mongodb

1 . insert only one document with creating a new collection

```
db.people.insertOne( {
    user_id: "abc123",
    age: 55,
    status: "A"
 } )
```

However, you can also explicitly create a collection:

```
db.createCollection("people")
```

2 . we can add a column or field to a collection . here we use $set

Collections do not describe or enforce the structure of its documents; i.e. there is no structural alteration at the collection level.

However, at the document level, `updateMany()` operations can add fields to existing documents using the `$set` operator.

```
db.people.updateMany(
    { },
    { $set: { join_date: new Date() } }

)
```

3 . we can drop column or delete field from the collection or documents .

here we use $unset .

Collections do not describe or enforce the structure of its documents; i.e. there is no structural alteration at the collection level.

However, at the document level, `updateMany()` operations can remove fields from documents using the `$unset` operator.

```
db.people.updateMany(

    { },

    { $unset: { "join_date": "" } }

)
```

4 . drop a table or collection from a database .

```
    db.people.drop()
```

5 . where condition in mongodb . we can use match operations .

```
SELECT user_id, status
FROM people
WHERE status = "A"
```

```
db.people.find(
    { status: "A" },
    { user_id: 1, status: 1, _id: 0 }

)
```

6 . and operations

```
db.people.find(
    { status: "A",
      age: 50 }
)
```

7 . or operations

```
db.people.find(
    { $or: [ { status: "A" } , { age: 50 } ] }
)
```

8 . find the data which matches the words or expression

```
SELECT *
FROM people
WHERE user_id like
"%bc%"
```

```
db.people.find( { user_id: /bc/ } )
```

-or-

```
db.people.find( { user_id: { $regex: /bc/ } }
)
```

9 . find a data with matched item and then sort that data

```
db.people.find( { status: "A" } ).sort( { user_id: 1 } )
```

10 . count total data of a collection collection

```
db.people.count()
```

Or

```
db.people.find().count(
```

11 . we can count the data with two ways

```
db.people.count( { user_id: { $exists: true } } )
```

Or

```
db.people.find( { user_id: { $exists: true } } ).count()
```

12 . we can find the distinct data

`SELECT DISTINCT(status) FROM people`	`db.people.aggregate( [ { $group : { _id : "$status" } } ] )`
	or, for distinct value sets that do not exceed the BSON size limit
	`db.people.distinct( "status" )`

13 . update a value based on conditions

`UPDATE people`
`SET status = "C"`
`WHERE age > 25`
	`db.people.updateMany(`
	`   { age: { $gt: 25 } },`
	`   { $set: { status: "C" } }`
	`)`

14 . delete the documents where assign a conditions

```
db.people.deleteMany( { status: "D" } )
```

Similarity of mongodb and mysql

<table>
<tr><th>SQL Example</th><th>MongoDB Example</th><th>Description</th></tr>
<tr><td><pre>SELECT COUNT(*) AS
count
FROM orders</pre></td><td><pre>db.orders.aggregate([
   {
     $group: {
        _id: null,
        count: { $sum: 1 }
     }
   }
])</pre></td><td>Count all records from <code>orders</code></td></tr>
<tr><td><pre>SELECT SUM(price)
AS total
FROM orders</pre></td><td><pre>db.orders.aggregate([
   {
     $group: {
        _id: null,
        total: { $sum:
"$price" }
     }
   }
])</pre></td><td>Sum the <code>price</code> field from <code>orders</code></td></tr>
</table>

`SELECT cust_id,`
`       SUM(price)`
`AS total`
`FROM orders`
`GROUP BY cust_id`
	`db.orders.aggregate( [`
	`   {`
	`     $group: {`
	`        _id: "$cust_id",`
	`        total: { $sum: "$price" }`
	`     }`
	`   }`
	`] )`
		For each unique `cust_id`, sum the `price` field.

`SELECT cust_id,`
`       SUM(price)`
`AS total`
`FROM orders`
`GROUP BY cust_id`
`ORDER BY total`
	`db.orders.aggregate( [`
	`   {`
	`     $group: {`
	`        _id: "$cust_id",`
	`        total: { $sum: "$price" }`
	`     }`
	`   },`
	`   { $sort: { total: 1 } }`
	`] )`
		For each unique `cust_id`, sum the `price` field, results sorted by sum.

SELECT cust_id,
ord_date,
SUM(price)
AS total
FROM orders
GROUP BY cust_id,
ord_date
	db.orders.aggregate([
	{
	$group: {
	_id: {
	cust_id:
	"$cust_id",
	ord_date: {
	$dateToString: {
	format:
	"%Y-%m-%d",
	date:
	"$ord_date"
	}}
	},
	total: { $sum:
	"$price" }
	}
	}
	])
		For each unique cust_id, ord_date grouping, sum the price field. Excludes the time portion of the date.

SELECT cust_id,
count(*)
FROM orders
GROUP BY cust_id
HAVING count(*) > 1
	db.orders.aggregate([
	{
	$group: {
	_id: "$cust_id",
	count: { $sum: 1 }
	}
	},
	{ $match: { count: { $gt:
	1 } } }
	])
		For cust_id with multiple records, return the cust_id and the corresponding record count.

`SELECT cust_id,`
`       ord_date,`
`       SUM(price)`
`AS total`
`FROM orders`
`GROUP BY cust_id,`
`         ord_date`
`HAVING total > 250`
	`db.orders.aggregate( [`
	`   {`
	`     $group: {`
	`        _id: {`
	`           cust_id:`
	`"$cust_id",`
	`           ord_date: {`
	`$dateToString: {`
	`              format:`
	`"%Y-%m-%d",`
	`              date:`
	`"$ord_date"`
	`           }}`
	`        },`
	`        total: { $sum:`
	`"$price" }`
	`     }`
	`   },`
	`   { $match: { total: { $gt:`
	`250 } } }`
	`] )`
		For each unique `cust_id`, `ord_date` grouping, sum the `price` field and return only where the sum is greater than 250. Excludes the time portion of the date.

`SELECT cust_id,`
`       SUM(price)`
`as total`
`FROM orders`
`WHERE status = 'A'`
`GROUP BY cust_id`
	`db.orders.aggregate( [`
	`   { $match: { status: 'A' }`
	`},`
	`   {`
	`     $group: {`
	`        _id: "$cust_id",`
	`        total: { $sum:`
	`"$price" }`
	`     }`
	`   }`
	`] )`
		For each unique `cust_id` with status `A`, sum the `price` field.

SELECT cust_id,
SUM(price)
as total
FROM orders
WHERE status = 'A'
GROUP BY cust_id
HAVING total > 250
	db.orders.aggregate([
	{ $match: { status: 'A' }
	},
	{
	$group: {
	_id: "$cust_id",
	total: { $sum:
	"$price" }
	}
	},
	{ $match: { total: { $gt:
	250 } } }
	])
		For each unique `cust_id` with status `A`, sum the `price` field and return only where the sum is greater than 250.

SELECT cust_id,
SUM(li.qty)
as qty
FROM orders o,
order_lineitem
li
WHERE li.order_id =
o.id
GROUP BY cust_id
	db.orders.aggregate([
	{ $unwind: "$items" },
	{
	$group: {
	_id: "$cust_id",
	qty: { $sum:
	"$items.qty" }
	}
	}
	])
		For each unique `cust_id`, sum the corresponding line item `qty` fields associated with the orders.

SELECT COUNT(*)
FROM (SELECT cust_id,
ord_date
FROM orders
GROUP BY cust_id,
ord_date)
as DerivedTable
	db.orders.aggregate([
	{
	$group: {
	_id: {
	cust_id: "$cust_id",
	ord_date: {
	$dateToString: {
	format: "%Y-%m-%d",
	date: "$ord_date"
	}}
	}
	}
	},
	{
	$group: {
	_id: null,
	count: { $sum: 1 }
	}
	}
	])
		Count the number of distinct `cust_id`, `ord_date` groupings. Excludes the time portion of the date.

$sortByCount

The $sortByCount stage is equivalent to the following $group + $sort sequence:

```
{ $group: { _id: <expression>, count: { $sum: 1 } } },
```

```
{ $sort: { count: -1 } }
```

```
{ "_id" : 1, "title" : "The Pillars of Society", "artist" : "Grosz", "year" : 1926, "tags" : [ "painting", "satire", "Expressionism", "caricature" ] }
{ "_id" : 2, "title" : "Melancholy III", "artist" : "Munch", "year" : 1902, "tags" : [ "woodcut", "Expressionism" ] }
{ "_id" : 3, "title" : "Dancer", "artist" : "Miro", "year" : 1925, "tags" : [ "oil", "Surrealism", "painting" ] }
{ "_id" : 4, "title" : "The Great Wave off Kanagawa", "artist" : "Hokusai", "tags" : [ "woodblock", "ukiyo-e" ] }
{ "_id" : 5, "title" : "The Persistence of Memory", "artist" : "Dali", "year" : 1931, "tags" : [ "Surrealism", "painting", "oil" ] }
{ "_id" : 6, "title" : "Composition VII", "artist" : "Kandinsky", "year" : 1913, "tags" : [ "oil", "painting", "abstract" ] }
{ "_id" : 7, "title" : "The Scream", "artist" : "Munch", "year" : 1893, "tags" : [ "Expressionism", "painting", "oil" ] }
{ "_id" : 8, "title" : "Blue Flower", "artist" : "O'Keefe", "year" : 1918, "tags" : [ "abstract", "painting" ] }
```

```
db.exhibits.aggregate(
  [
{ $unwind: "$tags" },  { $sortByCount: "$tags" }
]
)
```

```
{ "_id" : "painting", "count" : 6 }
{ "_id" : "oil", "count" : 4 }
{ "_id" : "Expressionism", "count" : 3 }
{ "_id" : "Surrealism", "count" : 2 }
{ "_id" : "abstract", "count" : 2 }
{ "_id" : "woodblock", "count" : 1 }
{ "_id" : "woodcut", "count" : 1 }
{ "_id" : "ukiyo-e", "count" : 1 }
{ "_id" : "satire", "count" : 1 }
{ "_id" : "caricature", "count" : 1 }
```

$sum

Use in $project Stage

{ "_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.students.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 }
```

$limit

```
db.article.aggregate(
[
   { $limit : 5 }
]
);
```

$sort

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

db.users.aggregate(
   [
     { $match: { $text: { $search: "operating" } } },
     { $sort: { score: { $meta: "textScore" }, posts: -1 } }
   ]
)
```

$project

```
{
  "_id" : 1,
  title: "abc123",
  isbn: "0001122223334",
  author: { last: "zzz", first: "aaa" },
  copies: 5,
  lastModified: "2016-07-28"
}
```

The following `$project` stage excludes the `author.first` and `lastModified` fields from the output:

```
db.books.aggregate( [ { $project : { "author.first" : 0, "lastModified" :
0 } } ] )
```

Alternatively, you can nest the exclusion specification in a document:

```
db.bookmarks.aggregate( [ { $project: { "author": { "first": 0},
"lastModified" : 0 } } ] )
```

You can include the $project operator on both the

```
{ _id: 1, user: "1234", stop: { title: "book1", author: "xyz", page: 32 }
}
{ _id: 2, user: "7890", stop: [ { title: "book2", author: "abc", page: 5
}, { title: "book3", author: "ijk", page: 100 } ] }
```

To include only the `title` field in the embedded document in the `stop` field, you can use the dot notation:

```
db.bookmarks.aggregate( [ { $project: { "stop.title": 1 } } ] )
```

Or, you can nest the inclusion specification in a document:

```
db.bookmarks.aggregate( [ { $project: { stop: { title: 1 } } } ] )
```

Both specifications result in the following documents:

```
{ "_id" : 1, "stop" : { "title" : "book1" } }

{ "_id" : 2, "stop" : [ { "title" : "book2" }, { "title" : "book3" } ] }
```

$group

```
db.sales.insertMany([

  { "_id" : 1, "item" : "abc", "price" : NumberDecimal("10"), "quantity" :
NumberInt("2"), "date" : ISODate("2014-03-01T08:00:00Z") },

  { "_id" : 2, "item" : "jkl", "price" : NumberDecimal("20"), "quantity" :
NumberInt("1"), "date" : ISODate("2014-03-01T09:00:00Z") },

  { "_id" : 3, "item" : "xyz", "price" : NumberDecimal("5"), "quantity" :
NumberInt( "10"), "date" : ISODate("2014-03-15T09:00:00Z") },

  { "_id" : 4, "item" : "xyz", "price" : NumberDecimal("5"), "quantity" :
NumberInt("20") , "date" : ISODate("2014-04-04T11:21:39.736Z") },

  { "_id" : 5, "item" : "abc", "price" : NumberDecimal("10"), "quantity" :
NumberInt("10") , "date" : ISODate("2014-04-04T21:23:13.331Z") },

  { "_id" : 6, "item" : "def", "price" : NumberDecimal("7.5"), "quantity":
NumberInt("5" ) , "date" : ISODate("2015-06-04T05:08:13Z") },

  { "_id" : 7, "item" : "def", "price" : NumberDecimal("7.5"), "quantity":
NumberInt("10") , "date" : ISODate("2015-09-10T08:43:00Z") },

  { "_id" : 8, "item" : "abc", "price" : NumberDecimal("10"), "quantity" :
NumberInt("5" ) , "date" : ISODate("2016-02-06T20:20:13Z") },

])

db.sales.aggregate( [

  {
    $group: {
        _id: null,
        count: { $sum: 1 }
    }
  }
] )
```

The operation returns the following result:

```
{ "_id" : null, "count" : 8 }
```

Retrieve Distinct Values

The following aggregation operation uses the `$group` stage to retrieve the distinct item values from the `sales` collection:

```
db.sales.aggregate( [ { $group : { _id : "$item" } } ] )
```

The operation returns the following result:

```
{ "_id" : "abc" }
{ "_id" : "jkl" }
{ "_id" : "def" }
{ "_id" : "xyz" }
```

Summation and multiplications of price and quantity fields

```
db.sales.aggregate(
  [
    // First Stage
    {
      $group :
        {
          _id : "$item",
          totalSaleAmount: { $sum: { $multiply: [ "$price", "$quantity" ] } }
        }
     },
     // Second Stage
     {
       $match: { "totalSaleAmount": { $gte: 100 } }
     }
   ]
```

```
 )
```

First Stage:

The `$group` stage groups the documents by `item` to retrieve the distinct item values. This stage returns the `totalSaleAmount` for each item.

Second Stage:

The `$match` stage filters the resulting documents to only return items with a `totalSaleAmount` greater than or equal to 100.

The operation returns the following result:

```
{ "_id" : "abc", "totalSaleAmount" : NumberDecimal("170") }

{ "_id" : "xyz", "totalSaleAmount" : NumberDecimal("150") }

{ "_id" : "def", "totalSaleAmount" : NumberDecimal("112.5") }
```

$lookup

```
db.orders.insert([
   { "_id" : 1, "item" : "almonds", "price" : 12, "quantity" : 2 },
   { "_id" : 2, "item" : "pecans", "price" : 20, "quantity" : 1 },
   { "_id" : 3  }
])
```

```
db.inventory.insert([
   { "_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 }
])
```

```
db.orders.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", "description" : "product 1",
"instock" : 120 }
   ]
}
{
   "_id" : 2,
   "item" : "pecans",
   "price" : 20,
   "quantity" : 1,
   "inventory_docs" : [
      { "_id" : 4, "sku" : "pecans", "description" : "product 4",
"instock" : 70 }
   ]
}
{
   "_id" : 3,
   "inventory_docs" : [
      { "_id" : 5, "sku" : null, "description" : "Incomplete" },
      { "_id" : 6 }
   ]
}
```

$match

```
{ "_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 }
```

In the aggregation pipeline, `$match` selects the documents where either the `score` is greater than `70` and less than `90` or the `views` is greater than or equal to `1000`. These documents are then piped to the `$group` to perform a count. The aggregation returns the following:

```
db.articles.aggregate(
 [
  {
$match: {
      $or:
            [
            { score: { $gt: 70, $lt: 90 } },
            { views: { $gte: 1000 } }
            ]
        }
},

              {
                 $group:
                 { _id: null, count: { $sum: 1 } }
              }
   }
 ]
);
```

```
{ "_id" : null, "count" : 5 }
```

…………………………..

Query :

Greater than :

```
db.getCollection('balances').find(
{
  "balance" : {$gt:12}
} )
```

Less than :

```
db.getCollection('balances').find(
{
   "balance" : {$lt:12}
}
 )
```

Not queal :

```
db.getCollection('balances').find(
{
   "balance" : {$ne:12}
}
 )
```

And operation :

```
db.getCollection('balances').find(
{
   "balance" : 12 ,
    "purchase": 18
}
 )
```

Or operation :

```
db.getCollection('balances').find(
{
    $or: [{"balance" : 12},{"purchase": 18}]
}
 )
```

Update operation :

```
db.getCollection('balances').update(
{ “_id”: ObjectId("5edc9027816a445e94645be3") } ,
{  $set: { “balance”: “12” } }
 )
```

Update a value with age and update the name (here the one value will update , among all values the first value will change)

```
db.getCollection('balances').update(
{ “age”: “20” } ,
{  $set: { “balance”: “12” } }
 )
```

For updating multiple values use the multi syntax

```
db.getCollection('balances').update(
{ “age”: “20” } ,
{  $set: { “balance”: “12” } },
{ mutli : true }
 )
```

To update the data with a save command with a json , then the data will change frequently based on that id .

```
db.getCollection('balances').save(
{
    "_id" : ObjectId("5e29aa7e5ea7f5581250b8be"),
    "balance" : 21,
    "purchase" : 0,
    "is_transaction_in_process" : true
}

 )
```

Removes all the documents of a collections

```
db.getCollection('balances').remove()
```

To remove a specific id we can type

```
db.getCollection('balances').remove(
{    "_id" : ObjectId("5e29aa7e5ea7f5581250b8be")}
)
```

To remove documents whose age is 16 or you can use gt , gte , lt ;

```
db.getCollection('balances').remove(
{    "age" : 16 }
)
```

Projections :

```
db.getCollection('balances').find(
{}, {"balance" : 1 , "purchase" : 1}
 )
```

Or

```
db.getCollection('balances').find(
{}, {"balance" : 0 , "purchase" : 0}
 )
```

To fetch a limited document we use limit

```
db.getCollection('balances').find(
{}, {"balance" : 1 , "purchase" : 1}
 ).limit(4)
```

We can skip the first two documents from the tables

```
db.getCollection('balances').find(
{}, {"balance" : 1 , "purchase" : 1}
 ).skip(4)
```

We can use skip , sort and limit together

```
db.getCollection('balances').find(
{}, {"balance" : 1 , "purchase" : 1}
 ).skip(2).limit(10).sort({"balance": 1})
```

Aggregate operations

```
{ "_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.students.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 }
```

```
db.getCollection('balances').aggregate([
{ $group: { _id : null, sum : { $sum: "$balance" } } }]);
```

The maximum and minimum number :

```
db.getCollection('balances').aggregate([
{ $group: { _id : null, sum : { $max: "$balance" } } }]);

db.getCollection('balances').aggregate([
{ $group: { _id : null, sum : { $min: "$balance" } } }]);
```

Aggregation

```
{
   _id: ObjectId(7df78ad8902c)
   title: 'MongoDB Overview',
   description: 'MongoDB is no sql database',
   by_user: 'tutorials point',
   url: 'http://www.tutorialspoint.com',
   tags: ['mongodb', 'database', 'NoSQL'],
   likes: 100
},
{
   _id: ObjectId(7df78ad8902d)
   title: 'NoSQL Overview',
   description: 'No sql database is very fast',
   by_user: 'tutorials point',
   url: 'http://www.tutorialspoint.com',
   tags: ['mongodb', 'database', 'NoSQL'],
   likes: 10
},
{
   _id: ObjectId(7df78ad8902e)
   title: 'Neo4j Overview',
   description: 'Neo4j is no sql database',
   by_user: 'Neo4j',
   url: 'http://www.neo4j.com',
   tags: ['neo4j', 'database', 'NoSQL'],
   likes: 750
},
```

$sum	Sums up the defined value from all documents in the collection.	db.mycol.aggregate([{$group : {_id : "$by_user", num_tutorial : {$sum : "$likes"}}}])
$avg	Calculates the average of all given values from all documents in the collection.	db.mycol.aggregate([{$group : {_id : "$by_user", num_tutorial : {$avg : "$likes"}}}])
$min	Gets the minimum of the corresponding values from all documents in the collection.	db.mycol.aggregate([{$group : {_id : "$by_user", num_tutorial : {$min : "$likes"}}}])

$max	Gets the maximum of the corresponding values from all documents in the collection.	db.mycol.aggregate([{$group : {_id : "$by_user", num_tutorial : {$max : "$likes"}}}])
$push	Inserts the value to an array in the resulting document.	db.mycol.aggregate([{$group : {_id : "$by_user", url : {$push: "$url"}}}])
$addToSet	Inserts the value to an array in the resulting document but does not create duplicates.	db.mycol.aggregate([{$group : {_id : "$by_user", url : {$addToSet : "$url"}}}])
$first	Gets the first document from the source documents according to the grouping. Typically this makes only sense together with some previously applied “$sort”-stage.	db.mycol.aggregate([{$group : {_id : "$by_user", first_url : {$first : "$url"}}}])
$last	Gets the last document from the source documents according to the grouping. Typically this makes only sense together with some previously applied “$sort”-stage.	db.mycol.aggregate([{$group : {_id : "$by_user", last_url : {$last : "$url"}}}])

Pipeline Concept

In UNIX command, shell pipeline means the possibility to execute an operation on some input and use the output as the input for the next command and so on. MongoDB also supports same concept in aggregation framework. There is a set of possible stages and each of those is taken as a set of documents as an input and produces a resulting set of documents (or the final resulting JSON document at the end of the pipeline). This can then in turn be used for the next stage and so on.

Following are the possible stages in aggregation framework −

- $project − Used to select some specific fields from a collection.
- $match − This is a filtering operation and thus this can reduce the amount of documents that are given as input to the next stage.
- $group − This does the actual aggregation as discussed above.
- $sort − Sorts the documents.
- $skip − With this, it is possible to skip forward in the list of documents for a given amount of documents.
- $limit − This limits the amount of documents to look at, by the given number starting from the current positions.
- $unwind − This is used to unwind document that are using arrays. When using an array, the data is kind of pre-joined and this operation will be undone with this to have individual documents again. Thus with this stage we will increase the amount of documents for the next stage.

In project : smartlotto :