```
            8096969696
        ]
}
```

comparison operators: $gt,$gte,$lt,$lte,$eq,$ne,$in,$nin
logical operators: $or, $nor, $and, $not
element query operators: $exists, $type

Evaluation Query Operators:

The operators which can be used for evaluation purposes are called Evaluation Query Operators.

1. $expr 2. $regex 3. $mod 4. $jsonSchema 5. $text 6. $where

1.$expr operator:

---

expr means expression.
Evaluate expression and select documents which satisfy that expression.
Syntax:
{ $expr: {<expression>}}

It is very helpful to compare two field values within the same document.

Case Study: Compare two field values from the same document:

```
db.homeBudget.insertOne({category:"home food", budget:1000, spent:1500})
db.homeBudget.insertOne({category:"outside food", budget:1000, spent:2000})
```

```
db.homeBudget.insertOne({category:"rent", budget:1500, spent:1500})
db.homeBudget.insertOne({category:"education", budget:2000,
spent:1000})
db.homeBudget.insertOne({category:"clothes", budget:750,
spent:1500})
db.homeBudget.insertOne({category:"entertinement", budget:1000,
spent:2500})
```

Q1. Select all documents where spent amount exceeds budget amount?

```
> db.homeBudget.find({$expr: {$gt: ["$spent","$budget"]}}).pretty()
{
        "_id" : ObjectId("5ff52c2345f8edf724d263ec"),
        "category" : "home food",
        "budget" : 1000,
        "spent" : 1500
}
{
        "_id" : ObjectId("5ff52c2345f8edf724d263ed"),
        "category" : "outside food",
        "budget" : 1000,
        "spent" : 2000
}
{
        "_id" : ObjectId("5ff52c2345f8edf724d263f0"),
        "category" : "clothes",
        "budget" : 750,
        "spent" : 1500
}
{
        "_id" : ObjectId("5ff52c2645f8edf724d263f1"),
        "category" : "entertinement",
        "budget" : 1000,
```

```
        "spent" : 2500
}
```

Q2. Select all documents where spent amount is less than or equal to budget amount?

```
> db.homeBudget.find({$expr: {$lte: ["$spent","$budget"]}}).pretty()
{
        "_id" : ObjectId("5ff52c2345f8edf724d263ee"),
        "category" : "rent",
        "budget" : 1500,
        "spent" : 1500
}
{
        "_id" : ObjectId("5ff52c2345f8edf724d263ef"),
        "category" : "education",
        "budget" : 2000,
        "spent" : 1000
}
```

Q3. Select all documents where spent amount is equal to budget amount?

```
> db.homeBudget.find({$expr: {$eq: ["$spent","$budget"]}}).pretty()
{
        "_id" : ObjectId("5ff52c2345f8edf724d263ee"),
        "category" : "rent",
        "budget" : 1500,
        "spent" : 1500
}
```

Note: $expr operator is very commonly used with aggregation expressions.

2.$regex operator:

---

regex means regular expression.

We can use $regex operator to select documents where values match a specified regular expression.

Syntax:

We can use $regex operator in any of the following styles:

```
{ field: { $regex: /pattern/, $options:'<options>'}}
{ field: { $regex: 'pattern', $options:'<options>'}}
{ field: { $regex: /pattern/<options>}}
{ field: /pattern/<options>}
```

Case Study:

```
db.homeBudget.insertOne({category:"home food", budget:1000, spent:1500})
db.homeBudget.insertOne({category:"outside food", budget:1000, spent:2000})
db.homeBudget.insertOne({category:"rent", budget:1500, spent:1500})
db.homeBudget.insertOne({category:"education", budget:2000, spent:1000})
db.homeBudget.insertOne({category:"clothes", budget:750, spent:1500})
db.homeBudget.insertOne({category:"entertinement", budget:1000, spent:2500})
```

Q1. Select all documents where category value contains food?

```
> db.homeBudget.find({ category: { $regex: /food/}}).pretty()
> db.homeBudget.find({ category: { $regex: 'food'}}).pretty()
> db.homeBudget.find({ category: /food/}).pretty()
{
        "_id" : ObjectId("5ff52c2345f8edf724d263ec"),
        "category" : "home food",
        "budget" : 1000,
        "spent" : 1500
}
{
        "_id" : ObjectId("5ff52c2345f8edf724d263ed"),
        "category" : "outside food",
        "budget" : 1000,
        "spent" : 2000
}
```

Note: It is something like 'like operator': '%xxx' or 'xxx%' or '%xxx%' in relational databases.

Note: We can use ^ symbol in regular expressions to indicate starts with.

Q2. Select all documents where category value starts with 'e'?

```
> db.homeBudget.find({ category: { $regex: /^e/}}).pretty()
> db.homeBudget.find({ category: { $regex: '^e'}}).pretty()
> db.homeBudget.find({ category: /^e/}).pretty()
{
        "_id" : ObjectId("5ff52c2345f8edf724d263ef"),
        "category" : "education",
        "budget" : 2000,
        "spent" : 1000
}
```

```
{
        "_id" : ObjectId("5ff52c2645f8edf724d263f1"),
        "category" : "entertinement",
        "budget" : 1000,
        "spent" : 2500
}
```

Note:

[abc] --->either a or b or c

[ec] --->either e or c

^[ec] ---> starts with either e or c

Q3. Select all documents where category value starts with either 'e' or 'c'?

```
> db.homeBudget.find({ category: { $regex: /^[ec]/}}).pretty()
> db.homeBudget.find({ category: { $regex: '^[ec]'}}).pretty()
> db.homeBudget.find({ category: /^[ec]/}).pretty()
{
        "_id" : ObjectId("5ff52c2345f8edf724d263ef"),
        "category" : "education",
        "budget" : 2000,
        "spent" : 1000
}
{
        "_id" : ObjectId("5ff52c2345f8edf724d263f0"),
        "category" : "clothes",
        "budget" : 750,
        "spent" : 1500
}
{
        "_id" : ObjectId("5ff52c2645f8edf724d263f1"),
        "category" : "entertinement",
```

```
        "budget" : 1000,
        "spent" : 2500
}
```

Note:

^t ---> starts with t

t$ ---> ends with t

Q4. Select all documents where category value ends with 't'?

```
> db.homeBudget.find({ category: { $regex: /t$/}}).pretty()
> db.homeBudget.find({ category: { $regex: 't$'}}).pretty()
> db.homeBudget.find({ category: /t$/}).pretty()

{
        "_id" : ObjectId("5ff52c2345f8edf724d263ee"),
        "category" : "rent",
        "budget" : 1500,
        "spent" : 1500
}
{
        "_id" : ObjectId("5ff52c2645f8edf724d263f1"),
        "category" : "entertinement",
        "budget" : 1000,
        "spent" : 2500
}
```

Q. Select all documents where category value ends with either 't' or 'n'?

```
> db.homeBudget.find({ category: { $regex: /[tn]$/}}).pretty()
> db.homeBudget.find({ category: { $regex: '[tn]$'}}).pretty()
> db.homeBudget.find({ category: /[tn]$/}).pretty()
{
```

```
        "_id" : ObjectId("5ff52c2345f8edf724d263ee"),
        "category" : "rent",
        "budget" : 1500,
        "spent" : 1500
}
{
        "_id" : ObjectId("5ff52c2345f8edf724d263ef"),
        "category" : "education",
        "budget" : 2000,
        "spent" : 1000
}
{
        "_id" : ObjectId("5ff52c2645f8edf724d263f1"),
        "category" : "entertinement",
        "budget" : 1000,
        "spent" : 2500
}
```

How to check case insensitity:

Bydefault case will be considered. If we want to ignore case, ie if we want case insensitivity then we should use 'i' option.

i means case insensitive.

Q. Select all documents where category value starts with either e or E?

```
> db.homeBudget.find({ category: {$regex: /^E/, $options: 'i'}}).pretty()
> db.homeBudget.find({ category: {$regex: '^E', $options: 'i'}}).pretty()
> db.homeBudget.find({ category: {$regex: /^E/i}}).pretty()
> db.homeBudget.find({ category: /^E/i}).pretty()
```

```
{
        "_id" : ObjectId("5ff52c2345f8edf724d263ef"),
```

```
        "category" : "education",
        "budget" : 2000,
        "spent" : 1000
}
{
        "_id" : ObjectId("5ff52c2645f8edf724d263f1"),
        "category" : "entertinement",
        "budget" : 1000,
        "spent" : 2500
}
```

3.$mod operator:

---

mod means modulo operator or remainder operator.
It is very helpful to select documents based on modulo operation.

We can use $mod operator to select documents where the value of the field divided by a divisor has a specified remainder.

Syntax: { field: {$mod: [divisor, remainder]}}

Case Study:

```
db.shop.insertOne({_id: 1, item: "soaps", quantity: 13})
db.shop.insertOne({_id: 2, item: "books", quantity: 10})
db.shop.insertOne({_id: 3, item: "pens", quantity: 15})
db.shop.insertOne({_id: 4, item: "pencils", quantity: 17})
```

Q1. Select all documents of shop collection where quantity value is divisible by 5?

```
> db.shop.find({ quantity: {$mod: [5, 0]}}).pretty()
{ "_id" : 2, "item" : "books", "quantity" : 10 }
```

```
{ "_id" : 3, "item" : "pens", "quantity" : 15 }
```

Q2. Select all documents of shop collection where quantity value is divisible by 4 and has remainder 1.

```
> db.shop.find({ quantity: {$mod: [4, 1]}}).pretty()
{ "_id" : 1, "item" : "soaps", "quantity" : 13 }
{ "_id" : 4, "item" : "pencils", "quantity" : 17 }
```

Note: { field: {$mod: [divisor, remainder]}}
Compulsory we have to provide both divisor and remainder, otherwise we will get error.

eg1:

```
> db.shop.find({ quantity: {$mod: [4]}}).pretty()
Error: error: {
        "ok" : 0,
        "errmsg" : "malformed mod, not enough elements",
        "code" : 2,
        "codeName" : "BadValue"
}
```

eg2:

```
> db.shop.find({ quantity: {$mod: [4,1,2]}}).pretty()
Error: error: {
        "ok" : 0,
        "errmsg" : "malformed mod, too many elements",
        "code" : 2,
        "codeName" : "BadValue"
}
```

4.$jsonSchema:

We can use this operator to select documents based on given jsonSchema.

5.$text:

It is related to indexes concept, will be discussed soon in INDEXES

6.$where:

It is deprecated and replaced $expr.

Array Query Operators:

---

1. $all 2. $elemMatch 3. $size

1. $all operator:

We can use $all operator to select documents where array contains all specified elements.

Syntax:

```
{ field: { $all: [value1, value2, value3,...]}}
```

We can write equaivalent query by using $and operator also.

```
{ $and: [{field: value1},{field: value2},{field: value3},...]}
```

Case Study:

```
db.courses.insertOne({_id:1,
name:"java",tags:["language","programming","easy","ocean"]})
```

```
db.courses.insertOne({_id:2,
name:"python",tags:["language","programming","easy"]})
db.courses.insertOne({_id:3,
name:"C",tags:["language","performance"]})
db.courses.insertOne({_id:4,
name:"Oracle",tags:["database","sql","cloud"]})
db.courses.insertOne({_id:5,
name:"MongoDB",tags:["database","nosql","cloud"]})
db.courses.insertOne({_id:6, name:"Devops",tags:["culture"]})
```

Q1. Select all documents where tags array contains "database" and "cloud" elements?

```
> db.courses.find({$and: [{tags: "database"}, {tags: "cloud"}]}).pretty()
> db.courses.find({tags: {$all: ["database","cloud"]}}).pretty()
{
        "_id" : 4,
        "name" : "Oracle",
        "tags" : [
                "database",
                "sql",
                "cloud"
        ]
}
{
        "_id" : 5,
        "name" : "MongoDB",
        "tags" : [
                "database",
                "nosql",
                "cloud"
        ]
}
```

Note: Order of elements is not important and it is not exact match.
> db.courses.find({tags: ["database","cloud"]}).pretty()===>Here order is important and Exact Match

Q2. Select all documents where tags array contains "language" and "programming" elements?

```
> db.courses.find({$and: [{tags: "language"}, {tags: "programming"}]}).pretty()
> db.courses.find({tags: {$all: ["language","programming"]}}).pretty()
{
        "_id" : 1,
        "name" : "java",
        "tags" : [
                "language",
                "programming",
                "easy",
                "ocean"
        ]
}
{
        "_id" : 2,
        "name" : "python",
        "tags" : [
                "language",
                "programming",
                "easy"
        ]
}
```

2. $elemMatch Operator:

---

elemMatch means element Match.

We can use $elemMatch operator to select documents where atleast one element of the array matches the specified query criteria.

Syntax:

{field: {$elemMatch: {<query1>,<query2>,<query3>,...}}}

Case Study:

```
db.students.insertOne({_id:1,name:"Durga",marks:[82,35,99]})
db.students.insertOne({_id:2,name:"Ravi",marks:[75,90,95]})
```

Q1. Select documents where student has atleast one subject marks greater than or equal to 80 but less than 90?

```
> db.students.find({marks: {$elemMatch: {$gte: 80, $lt: 90}}}).pretty()
{ "_id" : 1, "name" : "Durga", "marks" : [ 82, 35, 99 ] }
```

82 is greater than or equal to 80 but less than 90.

3. $size operator:

We can use $size operator to select documents based on specified array size.

Syntax: { field: {$size: n} }

Case Study:

```
db.courses.insertOne({_id:1,
name:"java",tags:["language","programming","easy","ocean"]})
db.courses.insertOne({_id:2,
name:"python",tags:["language","programming","easy"]})
db.courses.insertOne({_id:3,
name:"C",tags:["language","performance"]})
db.courses.insertOne({_id:4,
name:"Oracle",tags:["database","sql","cloud"]})
db.courses.insertOne({_id:5,
name:"MongoDB",tags:["database","nosql","cloud"]})
db.courses.insertOne({_id:6, name:"Devops",tags:["culture"]})
```

Q1. Select all documents where tags array contains exactly 4 elements?

```
> db.courses.find({tags: {$size: 4}}).pretty()
{
        "_id" : 1,
        "name" : "java",
        "tags" : [
                "language",
                "programming",
                "easy",
                "ocean"
        ]
}
```

Q2. Select all documents where tags array contains exactly 3 elements?

```
> db.courses.find({tags: {$size: 3}}).pretty()
{
        "_id" : 2,
        "name" : "python",
        "tags" : [
```

```
            "language",
            "programming",
            "easy"
        ]
}
{
        "_id" : 4,
        "name" : "Oracle",
        "tags" : [
            "database",
            "sql",
            "cloud"
        ]
}
{
        "_id" : 5,
        "name" : "MongoDB",
        "tags" : [
            "database",
            "nosql",
            "cloud"
        ]
}
```

Q3. Select all documents where tags array contains exactly 1 element?

```
> db.courses.find({tags: {$size: 1}}).pretty()
{ "_id" : 6, "name" : "Devops", "tags" : [ "culture" ] }
```

Note: $size does not accept range of values.

How to import data from csv file to MongoDB?