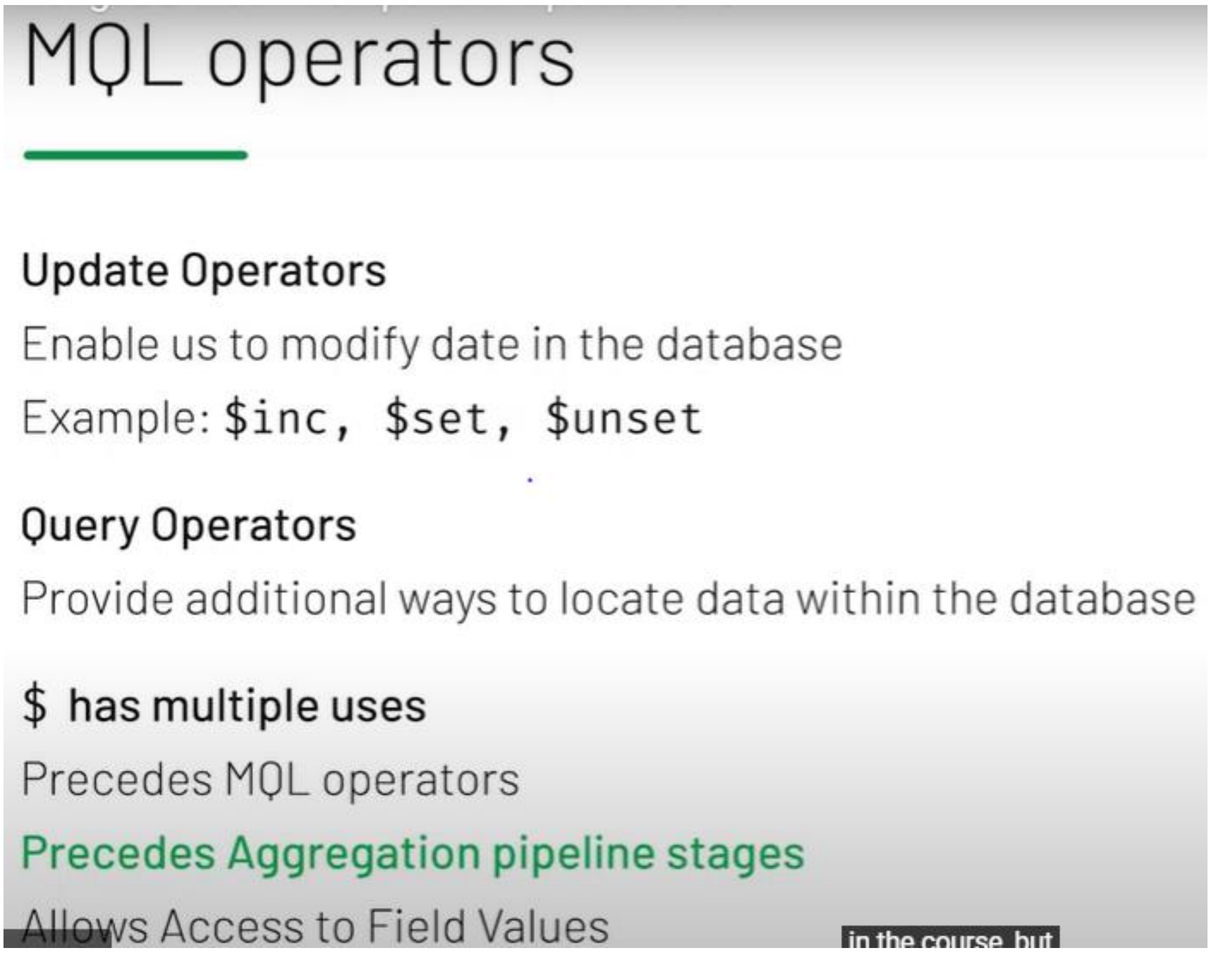


MQL operators

Update Operators

Enable us to modify date in the database

Example: `$inc, $set, $unset`

Query Operators

Provide additional ways to locate data within the database

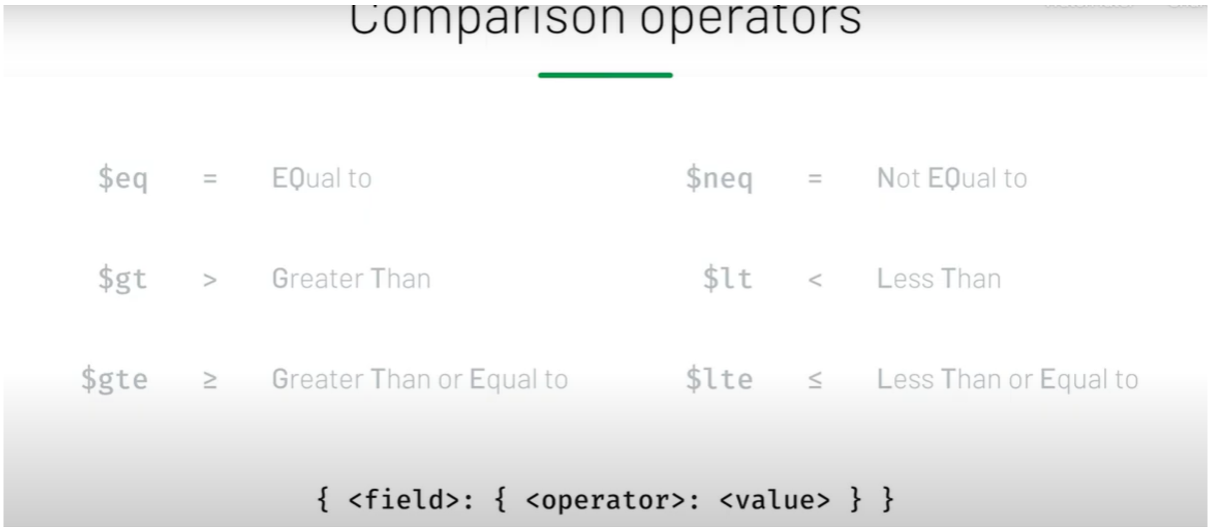
$ has multiple uses

Precedes MQL operators

Precedes Aggregation pipeline stages

Allows Access to Field Values

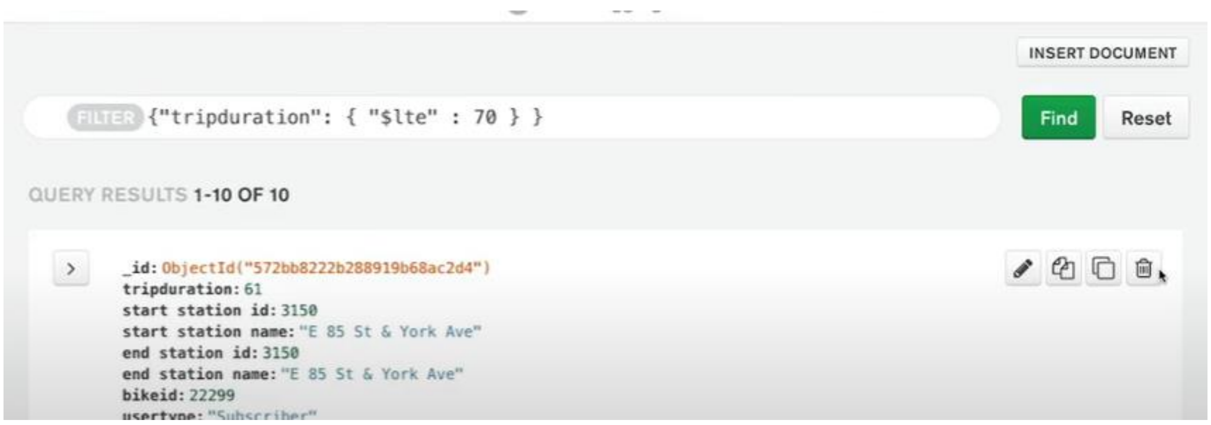
in the course, but



Comparison operators

$eq	=	EQual to	$neq	=	Not EQual to
$gt	>	Greater Than	$lt	<	Less Than
$gte	≥	Greater Than or Equal to	$lte	≤	Less Than or Equal to

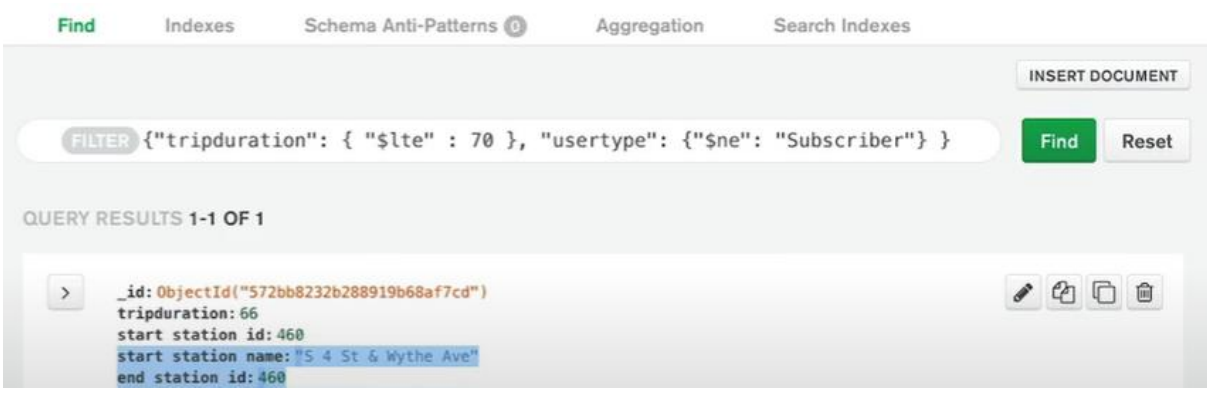
```
{ <field>: { <operator>: <value> } }
```



```
{"tripduration": { "$lte" : 70 } }
```

QUERY RESULTS 1-10 OF 10

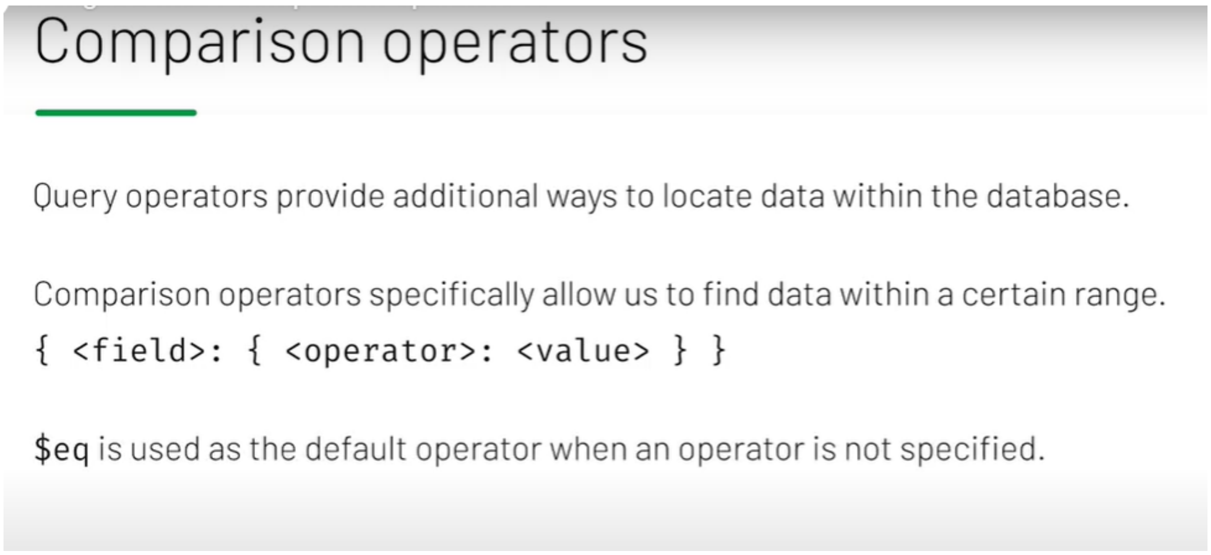
```
_id: ObjectId("572bb8222b288919b68ac2d4")
tripduration: 61
start station id: 3150
start station name: "E 85 St & York Ave"
end station id: 3150
end station name: "E 85 St & York Ave"
bikeid: 22299
usertype: "Subscriber"
```



```
{"tripduration": { "$lte" : 70 }, "usertype": {"$ne": "Subscriber"} }
```

QUERY RESULTS 1-1 OF 1

```
_id: ObjectId("572bb8232b288919b68af7cd")
tripduration: 66
start station id: 460
start station name: "S 4 St & Wythe Ave"
end station id: 460
```



Comparison operators

Query operators provide additional ways to locate data within the database.

Comparison operators specifically allow us to find data within a certain range.

```
{ <field>: { <operator>: <value> } }
```

$eq is used as the default operator when an operator is not specified.

Switch to this database:

```
use sample_training
```

Find all documents where the `tripduration` *was less than or equal to* `70` *seconds and the* `usertype` *was not* `Subscriber`:

```
db.trips.find({ "tripduration": { "$lte" : 70 },
                "usertype": { "$ne": "Subscriber" }
}).pretty()
```

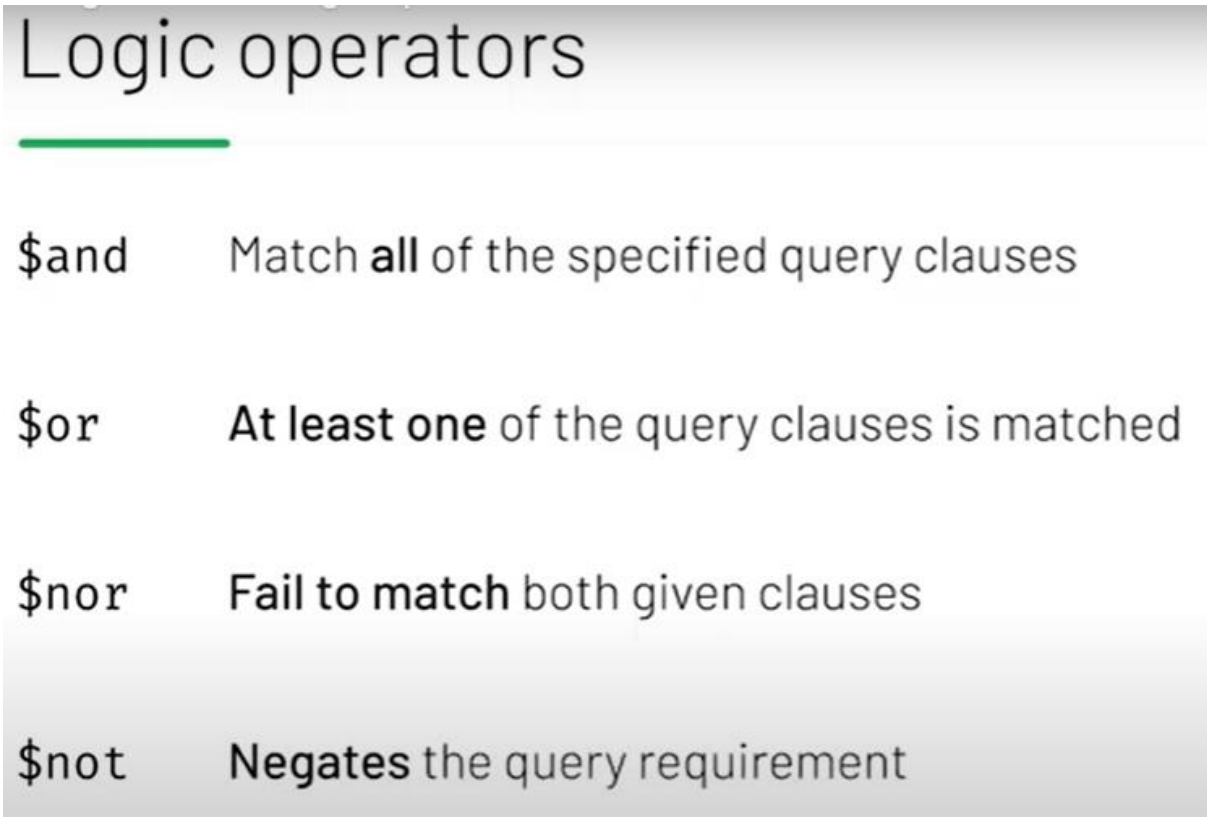
Find all documents where the `tripduration` *was less than or equal to* `70` *seconds and the* `usertype` *was* `Customer` *using a redundant equality operator:*

```
db.trips.find({ "tripduration": { "$lte" : 70 },

                "usertype": { "$eq": "Customer" }}).pretty()
```

Find all documents where the `tripduration` *was less than or equal to* `70` *seconds and the* `usertype` *was* `Customer` *using the implicit equality operator:*

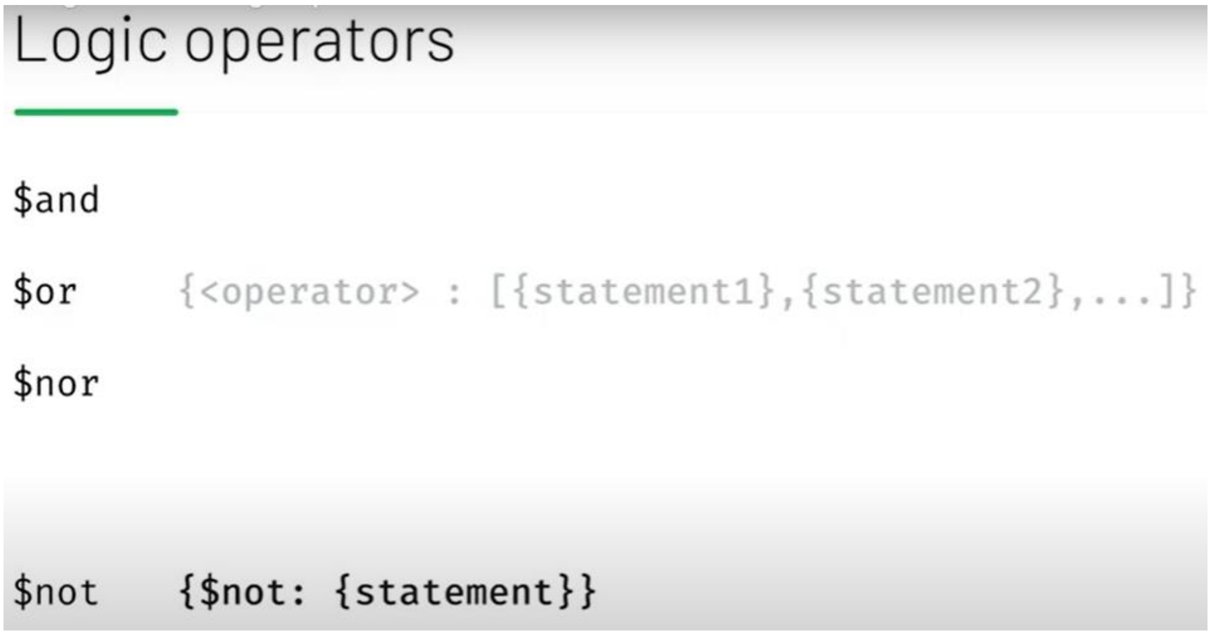
```
db.trips.find({ "tripduration": { "$lte" : 70 },

                "usertype": "Customer" }).pretty()
```

Logical operators:



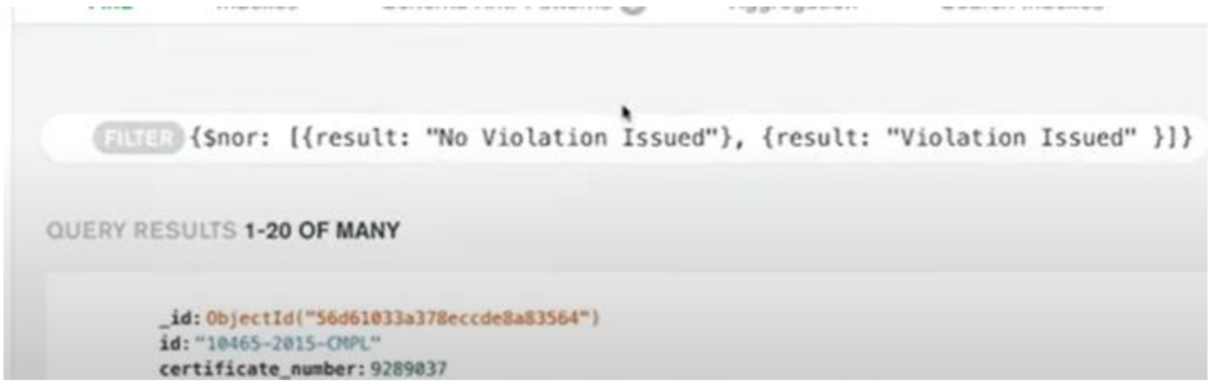
Logic operators

$and	Match **all** of the specified query clauses
$or	**At least one** of the query clauses is matched
$nor	**Fail to match** both given clauses
$not	**Negates** the query requirement



Logic operators

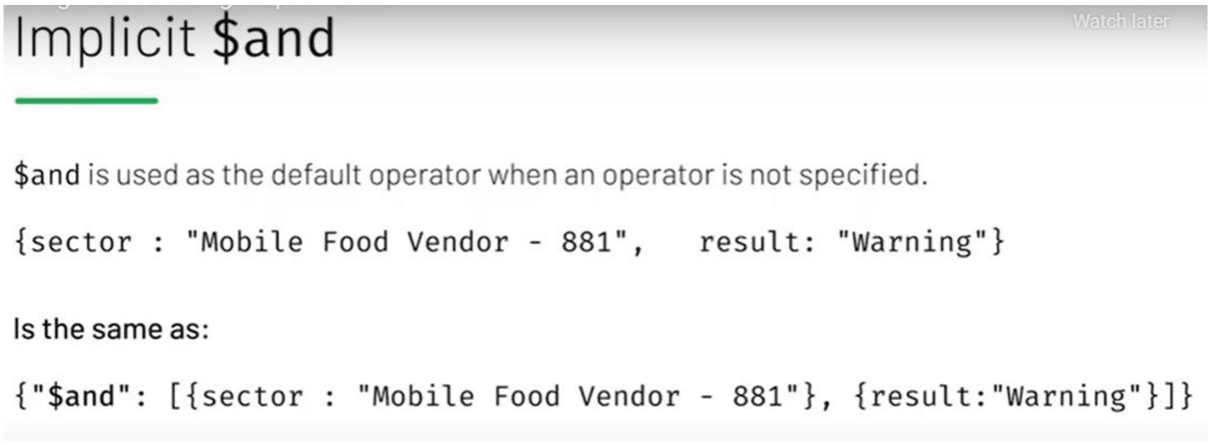
```
$and

$or     {<operator> : [{statement1},{statement2},...]}

$nor


$not    {$not: {statement}}
```



```
{$nor: [{result: "No Violation Issued"}, {result: "Violation Issued" }]}
```

QUERY RESULTS 1-20 OF MANY

```
_id: ObjectId("56d61033a378eccde8a83564")
id: "10465-2015-CMPL"
certificate_number: 9289037
```



Watch later

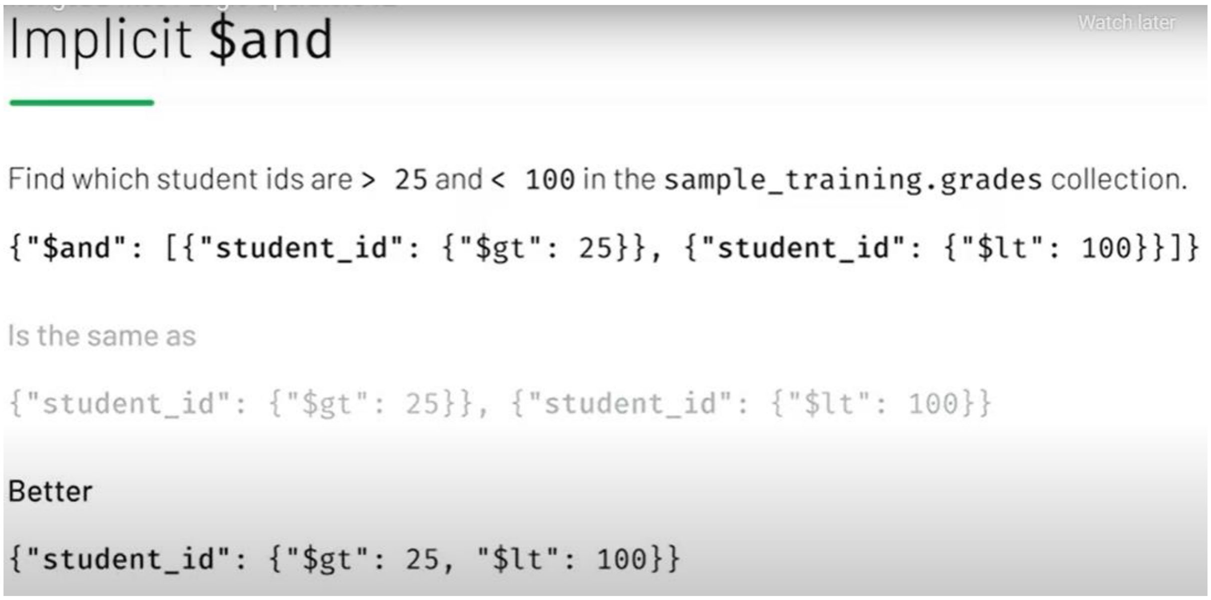
Implicit $and

$and is used as the default operator when an operator is not specified.

```
{sector : "Mobile Food Vendor - 881",   result: "Warning"}
```

Is the same as:

```
{"$and": [{sector : "Mobile Food Vendor - 881"}, {result:"Warning"}]}
```



Implicit $and

Find which student ids are `>` `25` and `<` `100` in the `sample_training.grades` collection.

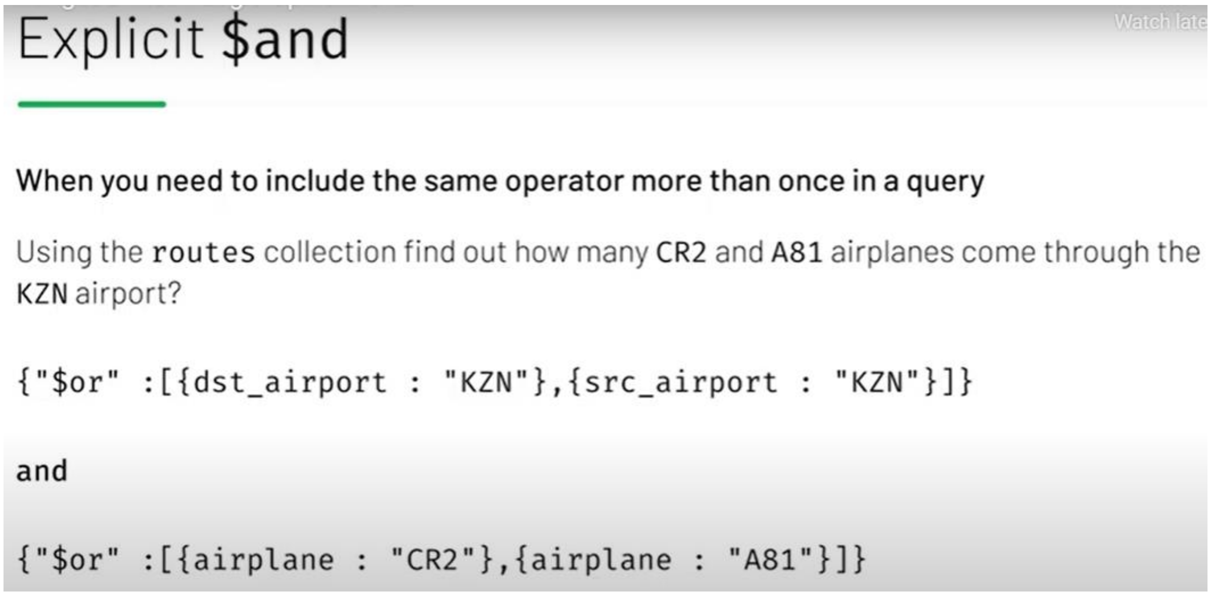
```
{"$and": [{"student_id": {"$gt": 25}}, {"student_id": {"$lt": 100}}]}
```

Is the same as

```
{"student_id": {"$gt": 25}}, {"student_id": {"$lt": 100}}
```

Better

```
{"student_id": {"$gt": 25, "$lt": 100}}
```



Explicit $and

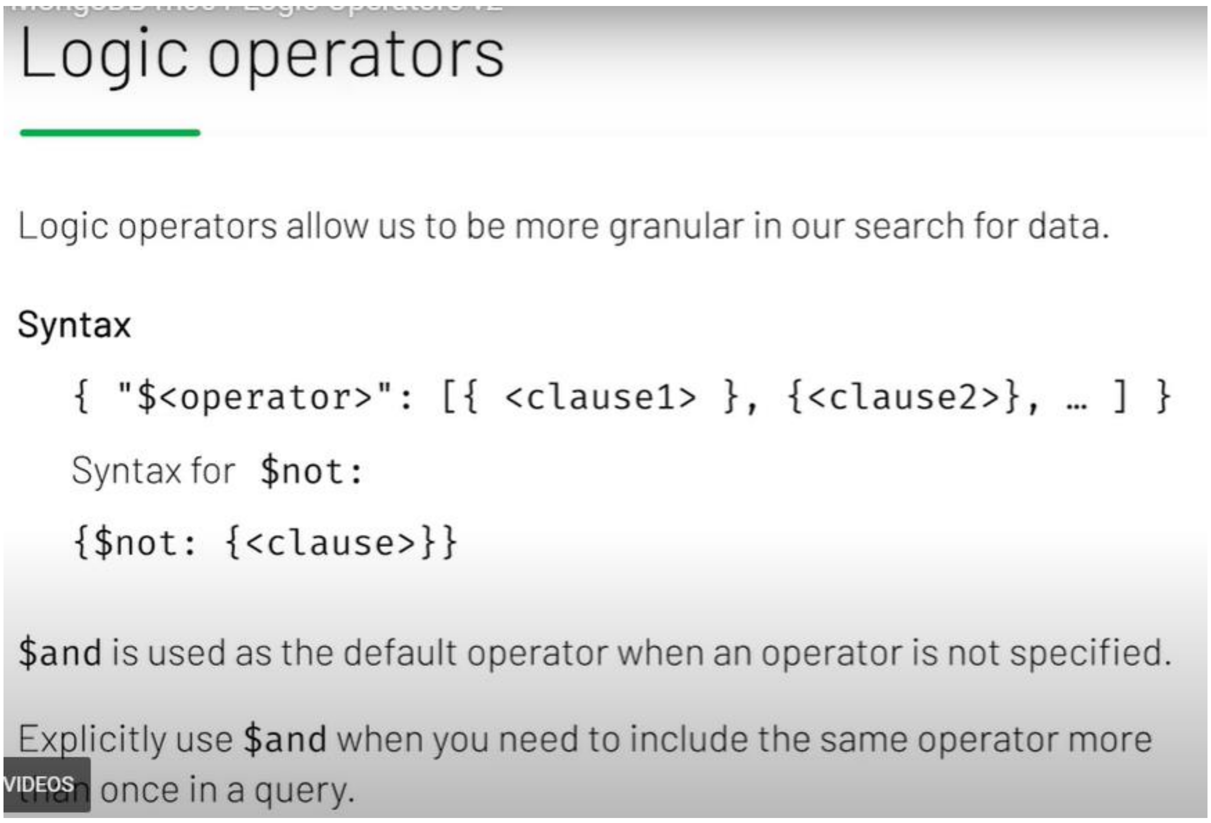
When you need to include the same operator more than once in a query

Using the `routes` collection find out how many `CR2` and `A81` airplanes come through the `KZN` airport?

```
{"$or" :[{dst_airport : "KZN"},{src_airport : "KZN"}]}
```

and

```
{"$or" :[{airplane : "CR2"},{airplane : "A81"}]}
```



Logic operators

Logic operators allow us to be more granular in our search for data.

Syntax

```
{ "$<operator>": [{ <clause1> }, {<clause2>}, … ] }
```

Syntax for `$not:`

```
{$not: {<clause>}}
```

$and is used as the default operator when an operator is not specified.

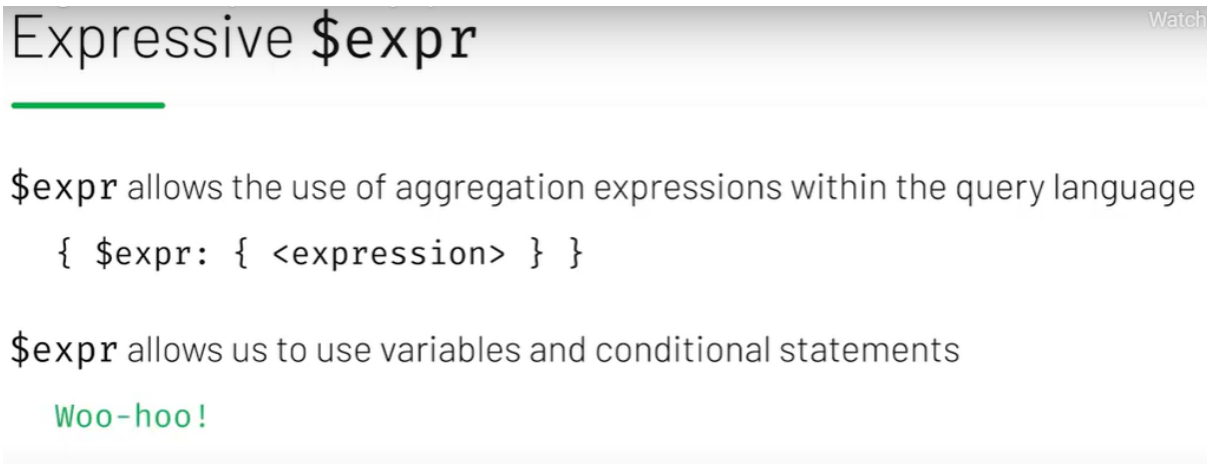
Explicitly use **$and** when you need to include the same operator more than once in a query.

Switch to this database:

```
use sample_training
```

Find all documents where airplanes `CR2` *or* `A81` *left or landed in the* `KZN` *airport:*

```
db.routes.find({ "$and": [ { "$or" :[ { "dst_airport": "KZN" },
                                        { "src_airport": "KZN" }
                                      ] },
                           { "$or" :[ { "airplane": "CR2" },
                                       { "airplane": "A81" } ] }
                         ]}).pretty()
```



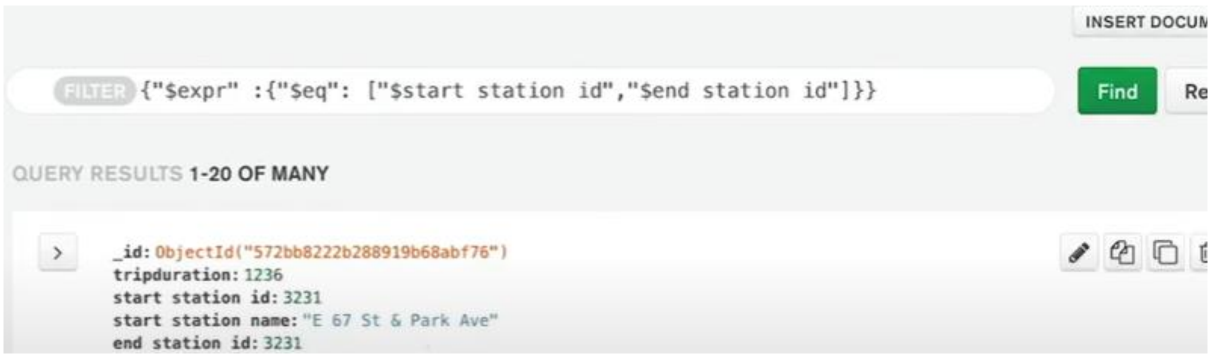
Expressive `$expr`

`$expr` allows the use of aggregation expressions within the query language

```
{ $expr: { <expression> } }
```

`$expr` allows us to use variables and conditional statements

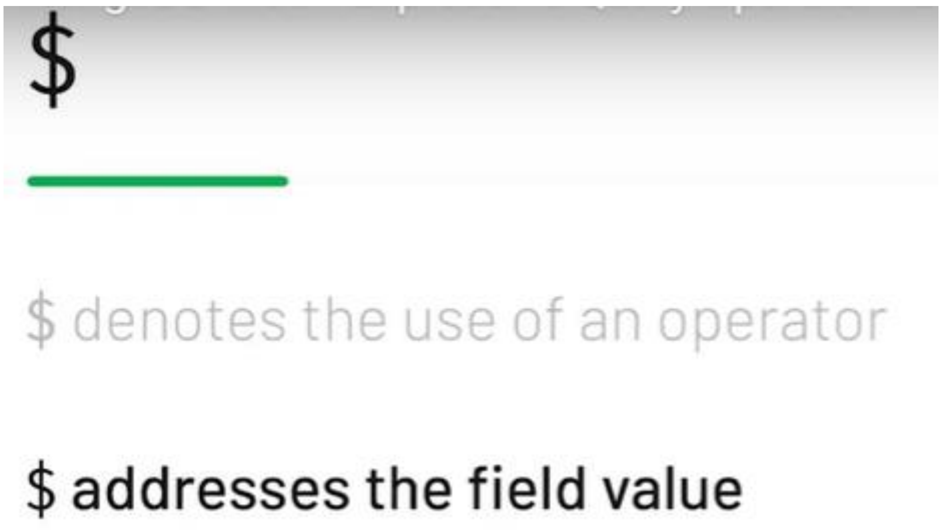
`Woo-hoo!`



```
{"$expr" :{"$eq": ["$start station id","$end station id"]}}
```

QUERY RESULTS 1-20 OF MANY

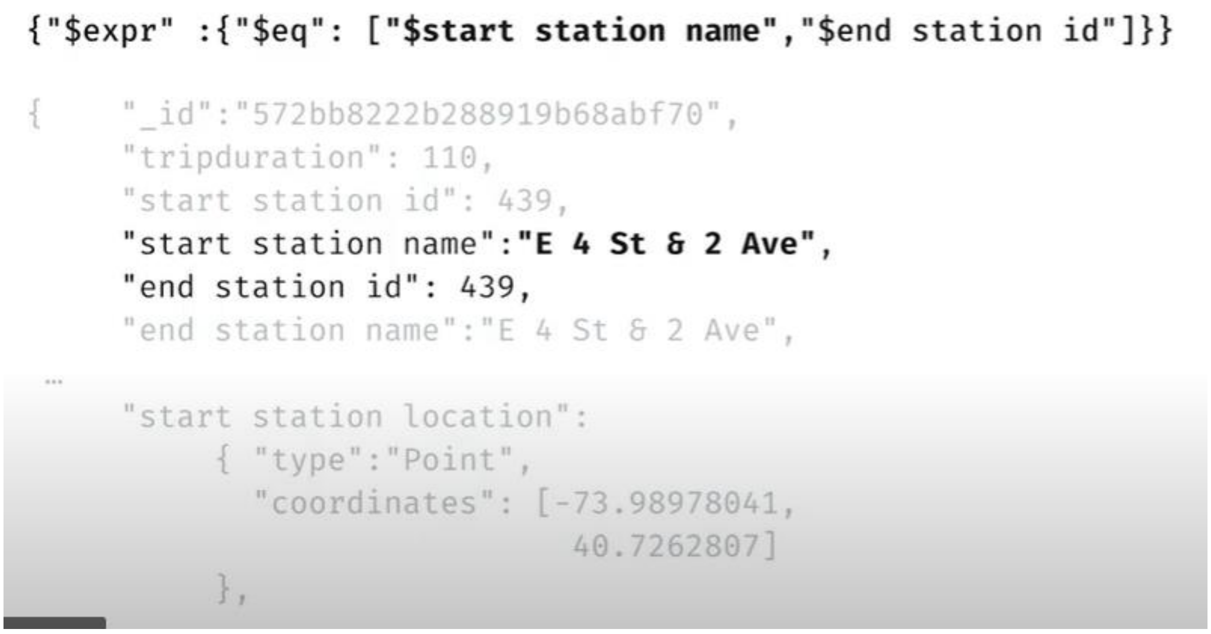
```
_id: ObjectId("572bb8222b288919b68abf76")
tripduration: 1236
start station id: 3231
start station name: "E 67 St & Park Ave"
end station id: 3231
```



$

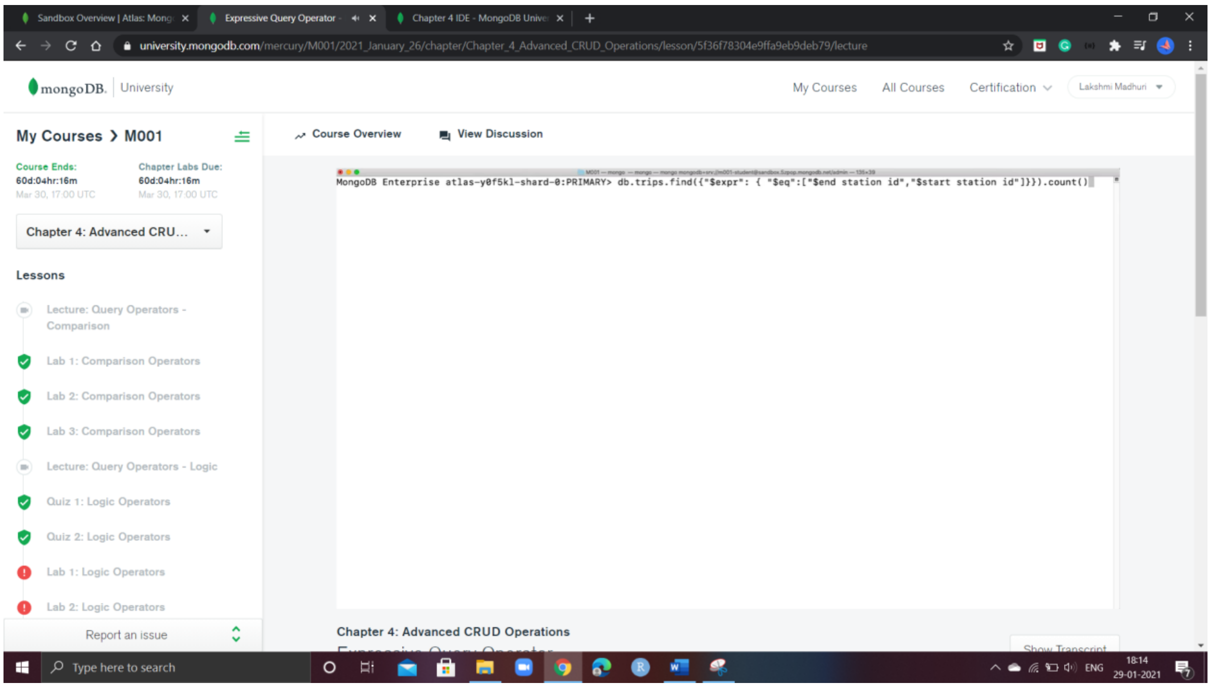
$ denotes the use of an operator

$ addresses the field value



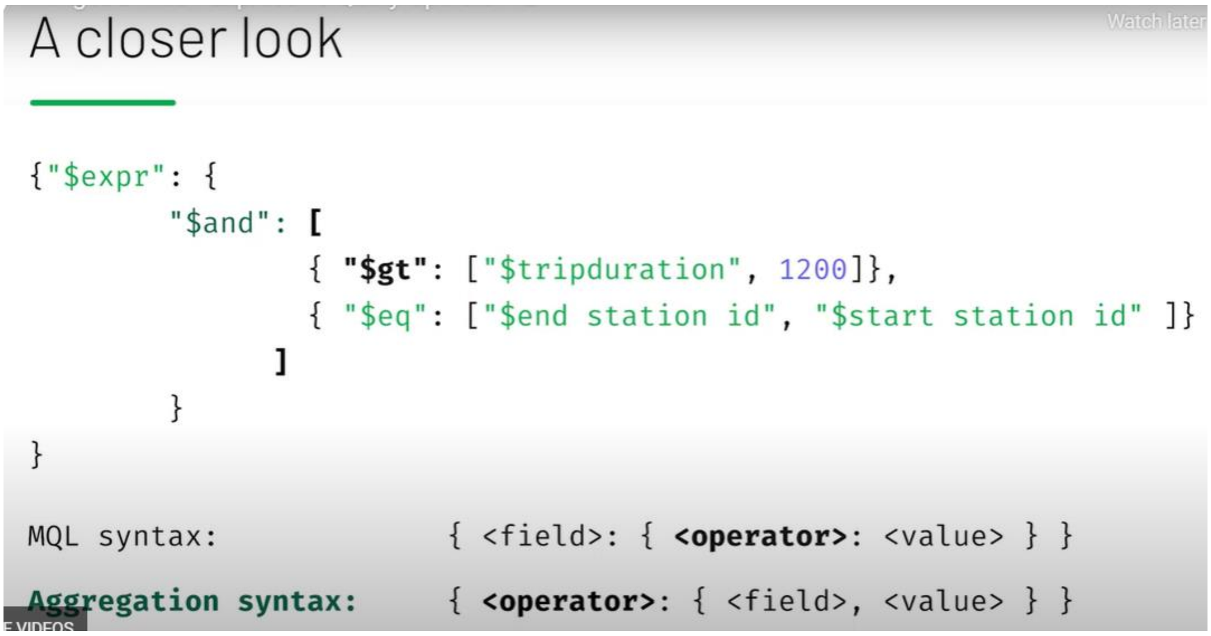
```
{"$expr" :{"$eq": ["$start station name","$end station id"]}}

{     "_id":"572bb8222b288919b68abf70",
      "tripduration": 110,
      "start station id": 439,
      "start station name":"E 4 St & 2 Ave",
      "end station id": 439,
      "end station name":"E 4 St & 2 Ave",
...
      "start station location":
          { "type":"Point",
            "coordinates": [-73.98978041,
                             40.7262807]
          },
```

$ specifies value of that field



My Courses > M001
Course Overview
View Discussion
Chapter 4: Advanced CRUD Operations
Lessons
Lecture: Query Operators - Comparison
Lab 1: Comparison Operators
Lab 2: Comparison Operators
Lab 3: Comparison Operators
Lecture: Query Operators - Logic
Quiz 1: Logic Operators
Quiz 2: Logic Operators
Lab 1: Logic Operators
Lab 2: Logic Operators
Report an issue

```
MongoDB Enterprise atlas-y0f5kl-shard-0:PRIMARY> db.trips.find({"$expr": { "$eq":["$end station id","$start station id"]}}).count()
```



A closer look

```
{"$expr": {
        "$and": [
                { "$gt": ["$tripduration", 1200]},
                { "$eq": ["$end station id", "$start station id" ]}
              ]
        }
}
```

```
MQL syntax:              { <field>: { <operator>: <value> } }

Aggregation syntax:      { <operator>: { <field>, <value> } }
```

```
use sample_training

 COPY
```

Find all documents where the trip started and ended at the same station:

```
db.trips.find({ "$expr": { "$eq": [ "$end station id",
"$start station id"] }

              }).count()

 COPY
```

Find all documents where the trip lasted longer than `1200` *seconds, and started and ended at the same station:*

```
db.trips.find({ "$expr": { "$and": [ { "$gt": [
"$tripduration", 1200 ]},

                         { "$eq": [ "$end station id",
"$start station id" ]}

                       ]}}).count()
```

Array Operators:

Switch to this database:

```
use sample_airbnb
```

Find all documents with exactly `20` *amenities which include all the amenities listed in the query array, and display their* `price` *and* `address`:

```
db.listingsAndReviews.find({ "amenities":{ "$size": 20,
"$all": [ "Internet", "Wifi",  "Kitchen",
"Heating","Family/kid friendly", "Washer",
"Dryer","Essentials", "Shampoo", "Hangers","Hair dryer",
"Iron","Laptop friendly workspace" ] } },{"price": 1,
"address": 1}).pretty()
```

Find all documents that have `Wifi` *as one of the amenities only include* `price` *and* `address` *in the resulting cursor:*

```
db.listingsAndReviews.find({ "amenities": "Wifi" },{
"price": 1, "address": 1, "_id": 0 }).pretty()
```

Find all documents that have `Wifi` *as one of the amenities only include price and address in the resulting cursor, also exclude* ``"maximum_nights"``. ***This will be an error:**

```
db.listingsAndReviews.find({ "amenities": "Wifi" },{
"price": 1, "address": 1,"_id": 0, "maximum_nights":0
}).pretty()
```

Switch to this database:

```
use sample_training
```

Get one document from the collection:

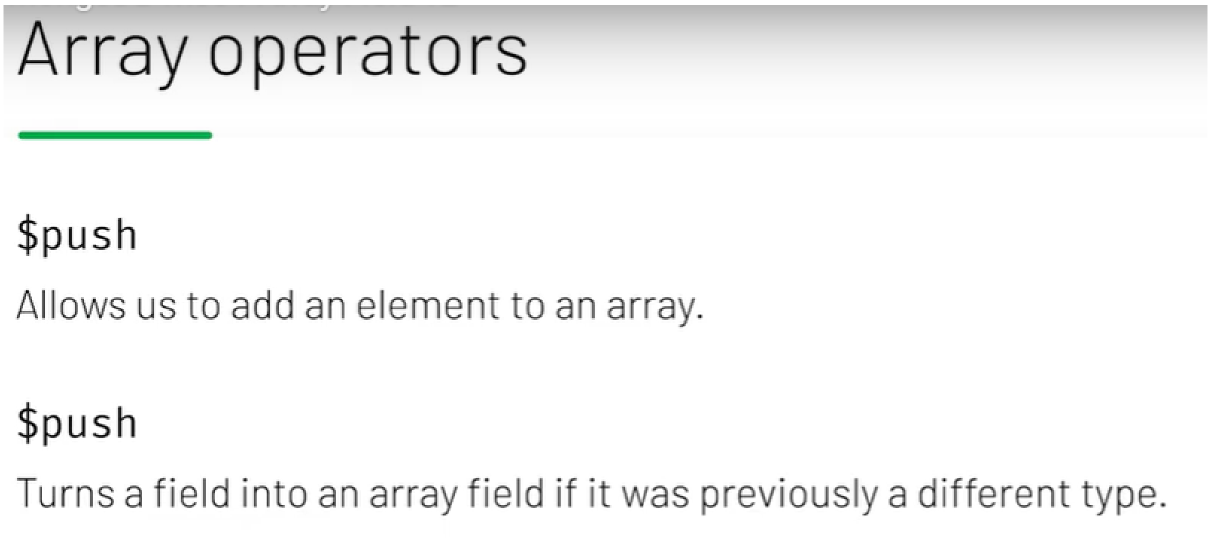
```
db.grades.findOne()
```

Find all documents where the student in class `431` *received a grade higher than* `85` *for any type of assignment:*

```
db.grades.find({ "class_id": 431 },{ "scores": {
"$elemMatch": { "score": { "$gt": 85 } } } }).pretty()
```

Find all documents where the student had an extra credit score:

```
db.grades.find({ "scores": { "$elemMatch": { "type": "extra
credit" } } }).pretty()
```



Array operators

$push

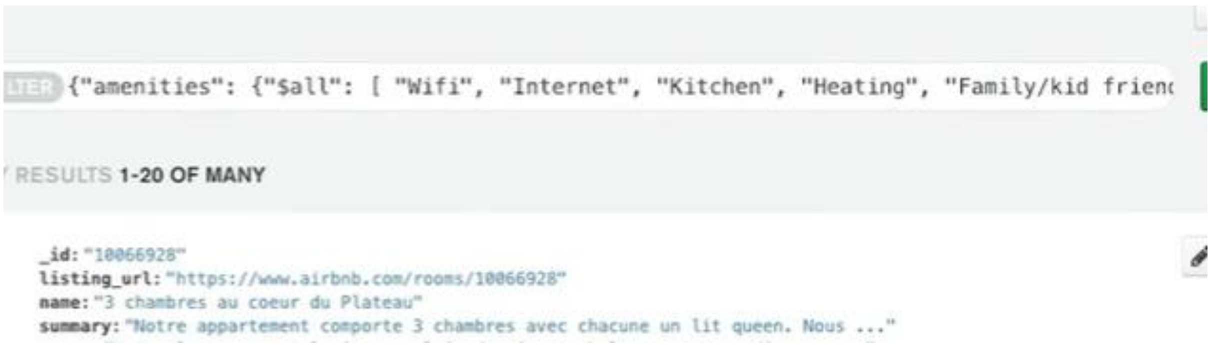
Allows us to add an element to an array.

$push

Turns a field into an array field if it was previously a different type.

order matters in array to find something {amenties:[array elemts]}

if we want to list all the array elements we can check using “$all”

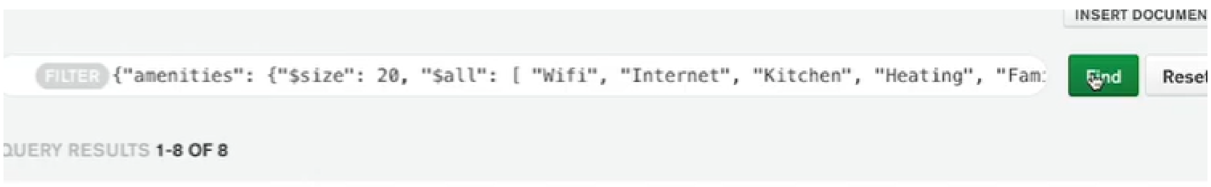


```
LTER {"amenities": {"$all": [ "Wifi", "Internet", "Kitchen", "Heating", "Family/kid frienc
```

RESULTS 1-20 OF MANY

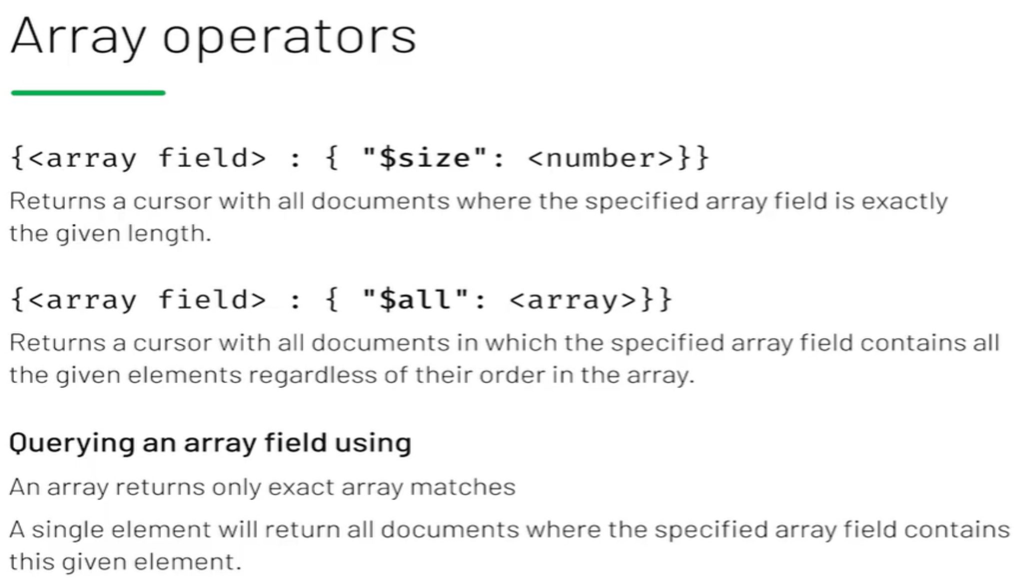
```
_id: "10066928"
listing_url: "https://www.airbnb.com/rooms/10066928"
name: "3 chambres au coeur du Plateau"
summary: "Notre appartement comporte 3 chambres avec chacune un lit queen. Nous ..."
```

Limiting results can be done by specifying size



```
FILTER {"amenities": {"$size": 20, "$all": [ "Wifi", "Internet", "Kitchen", "Heating", "Fam:
```

QUERY RESULTS 1-8 OF 8



Array operators

{<array field> : { "$size": <number>}}

Returns a cursor with all documents where the specified array field is exactly the given length.

{<array field> : { "$all": <array>}}

Returns a cursor with all documents in which the specified array field contains all the given elements regardless of their order in the array.

Querying an array field using

An array returns only exact array matches

A single element will return all documents where the specified array field contains this given element.

Array Operators and projections:

Switch to this database:

```
use sample_airbnb
```

Find all documents with exactly `20` *amenities which include all the amenities listed in the query array, and display their* `price` *and* `address`:

```
db.listingsAndReviews.find({ "amenities":

        { "$size": 20, "$all": [ "Internet", "Wifi",
"Kitchen", "Heating",

                                 "Family/kid friendly",
"Washer", "Dryer",

                                 "Essentials", "Shampoo",
"Hangers",

                                 "Hair dryer", "Iron",

                                 "Laptop friendly workspace"
] } },

                            {"price": 1, "address":
1}).pretty()
```

Find all documents that have `Wifi` as one of the amenities only include `price` and `address` in the resulting cursor:

```
db.listingsAndReviews.find({ "amenities": "Wifi" },

                           { "price": 1, "address": 1,
"_id": 0 }).pretty()
```

Find all documents that have `Wifi` as one of the amenities only include price and address in the resulting cursor, also exclude ``"maximum_nights"``. ***This will be an error:**

```
db.listingsAndReviews.find({ "amenities": "Wifi" },

                           { "price": 1, "address": 1,

                             "_id": 0, "maximum_nights":0
}).pretty()
```

Switch to this database:

```
use sample_training
```

Get one document from the collection:

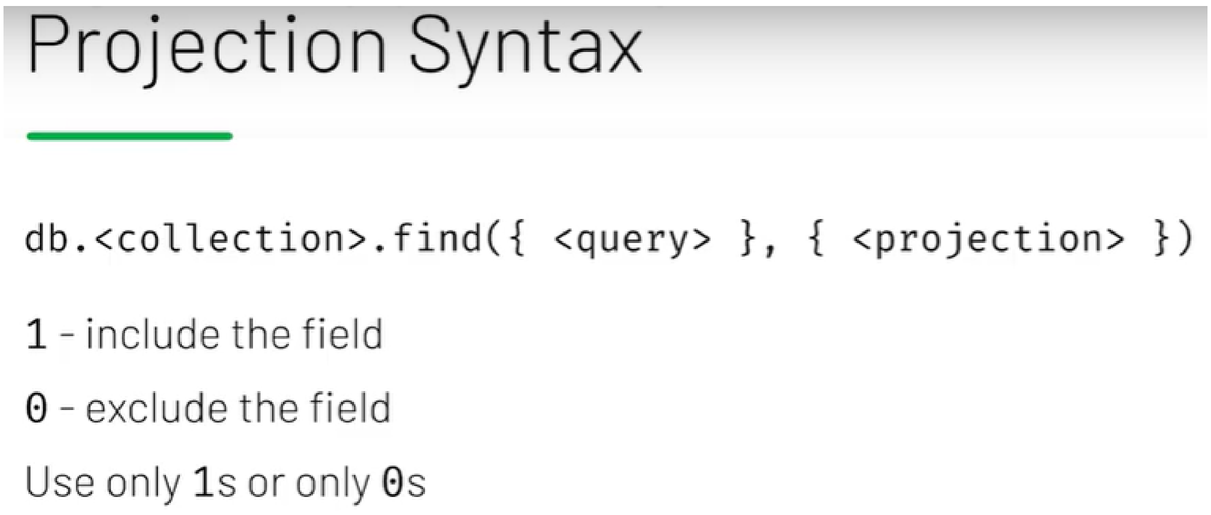
```
db.grades.findOne()
```

Find all documents where the student in class `431` received a grade higher than `85` for any type of assignment:

```
db.grades.find({ "class_id": 431 },

               { "scores": { "$elemMatch": { "score": {
"$gt": 85 } } }

             }).pretty()
```

Find all documents where the student had an extra credit score:

```
db.grades.find({ "scores": { "$elemMatch": { "type": "extra
credit" } }

               }).pretty()
```



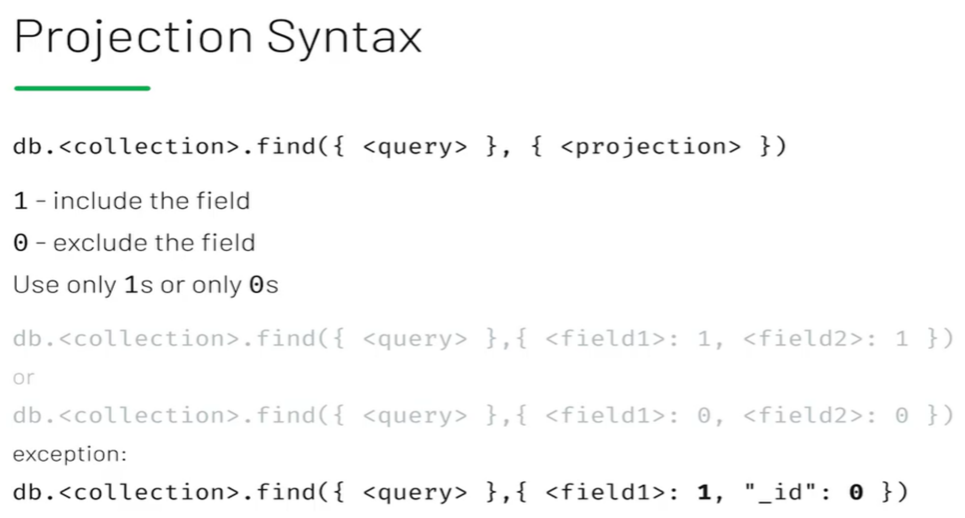
Projection Syntax

`db.<collection>.find({ <query> }, { <projection> })`

`1` - include the field

`0` - exclude the field

Use only `1`s or only `0`s



Projection Syntax

`db.<collection>.find({ <query> }, { <projection> })`

`1` - include the field

`0` - exclude the field

Use only `1`s or only `0`s

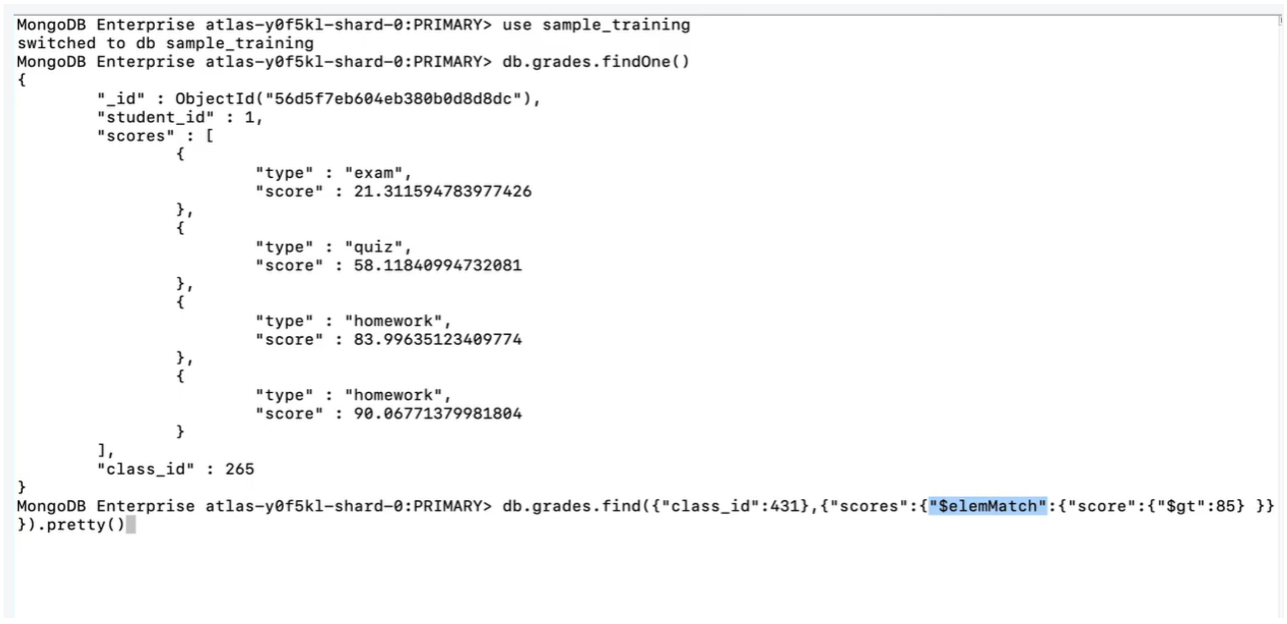
`db.<collection>.find({ <query> },{ <field1>: 1, <field2>: 1 })`

or

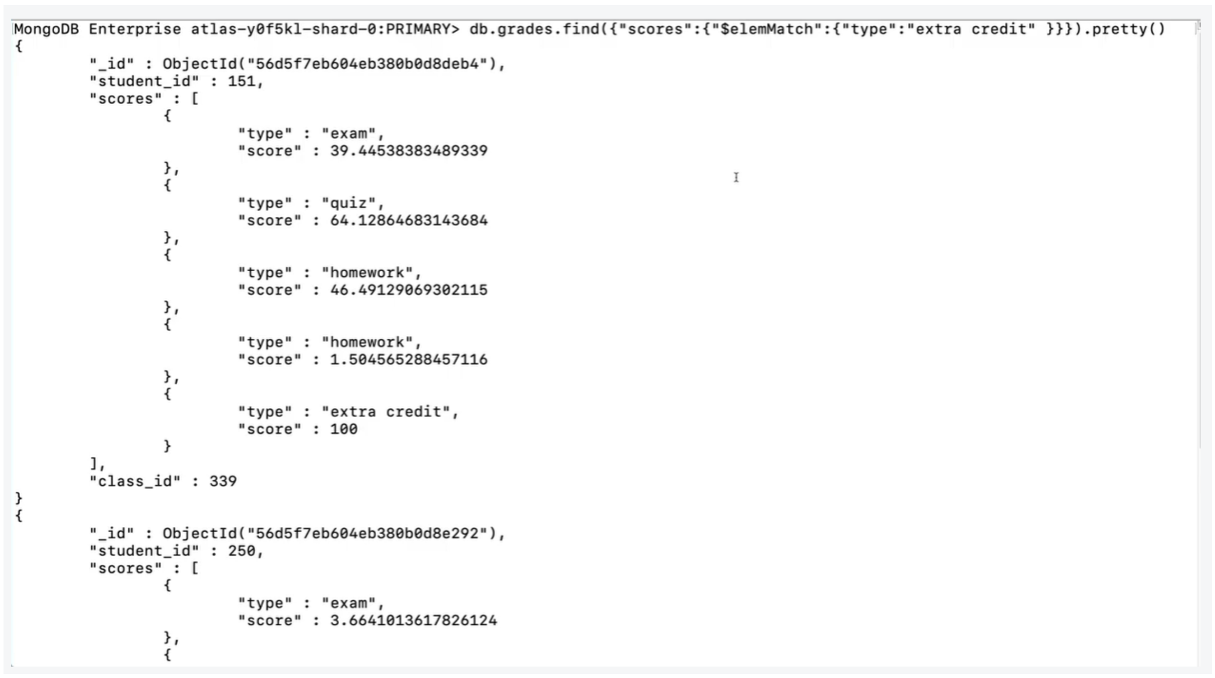
`db.<collection>.find({ <query> },{ <field1>: 0, <field2>: 0 })`

exception:

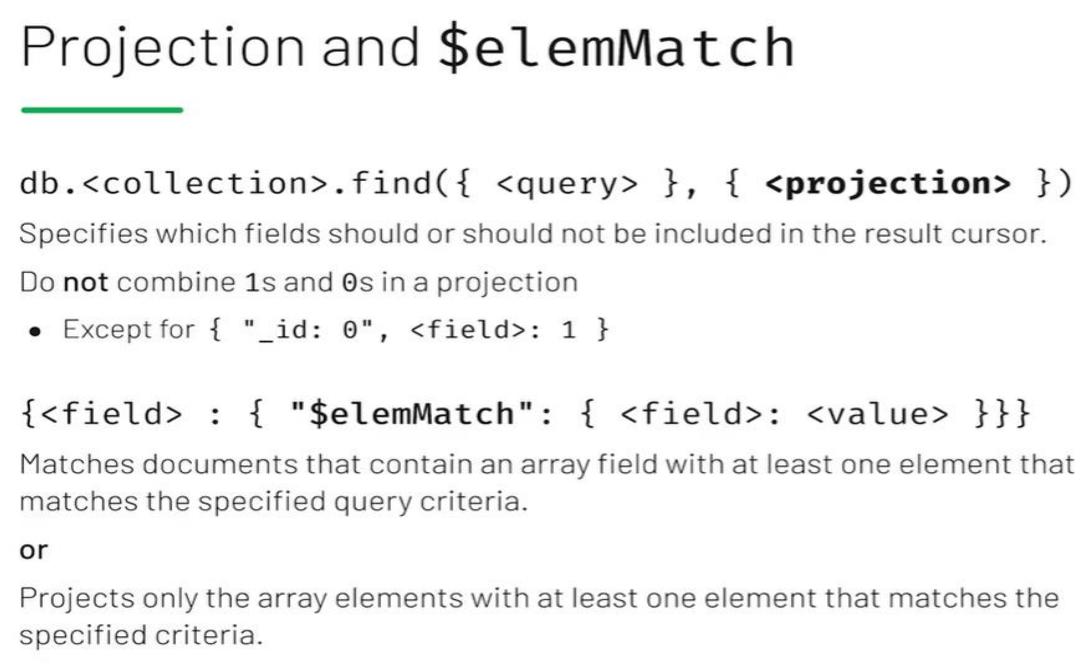
`db.<collection>.find({ <query> },{ <field1>: **1**, "_id": **0** })`



```
MongoDB Enterprise atlas-y0f5kl-shard-0:PRIMARY> use sample_training
switched to db sample_training
MongoDB Enterprise atlas-y0f5kl-shard-0:PRIMARY> db.grades.findOne()
{
        "_id" : ObjectId("56d5f7eb604eb380b0d8d8dc"),
        "student_id" : 1,
        "scores" : [
                {
                        "type" : "exam",
                        "score" : 21.311594783977426
                },
                {
                        "type" : "quiz",
                        "score" : 58.11840994732081
                },
                {
                        "type" : "homework",
                        "score" : 83.99635123409774
                },
                {
                        "type" : "homework",
                        "score" : 90.06771379981804
                }
        ],
        "class_id" : 265
}
MongoDB Enterprise atlas-y0f5kl-shard-0:PRIMARY> db.grades.find({"class_id":431},{"scores":{"$elemMatch":{"score":{"$gt":85} }}
}).pretty()
```



```
MongoDB Enterprise atlas-y0f5kl-shard-0:PRIMARY> db.grades.find({"scores":{"$elemMatch":{"type":"extra credit" }}}).pretty()
{
        "_id" : ObjectId("56d5f7eb604eb380b0d8deb4"),
        "student_id" : 151,
        "scores" : [
                {
                        "type" : "exam",
                        "score" : 39.44538383489339
                },
                {
                        "type" : "quiz",
                        "score" : 64.12864683143684
                },
                {
                        "type" : "homework",
                        "score" : 46.49129069302115
                },
                {
                        "type" : "homework",
                        "score" : 1.504565288457116
                },
                {
                        "type" : "extra credit",
                        "score" : 100
                }
        ],
        "class_id" : 339
}
{
        "_id" : ObjectId("56d5f7eb604eb380b0d8e292"),
        "student_id" : 250,
        "scores" : [
                {
                        "type" : "exam",
                        "score" : 3.6641013617826124
                },
                {
```



Projection and `$elemMatch`

`db.<collection>.find({ <query> }, { **<projection>** })`

Specifies which fields should or should not be included in the result cursor.

Do **not** combine `1`s and `0`s in a projection

- Except for `{ "_id: 0", <field>: 1 }`

`{<field> : { "$elemMatch": { <field>: <value> }}}`

Matches documents that contain an array field with at least one element that matches the specified query criteria.

or

Projects only the array elements with at least one element that matches the specified criteria.

Sub-Documents:

`use sample_training`

`db.trips.findOne({ "start station location.type": "Point" })`

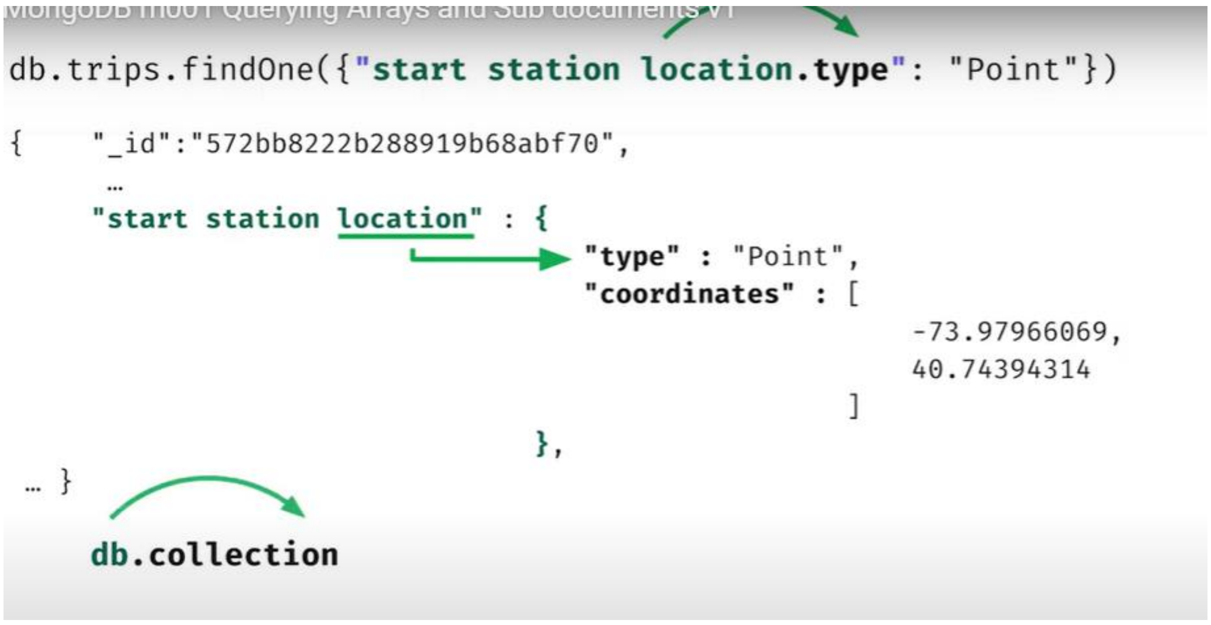
`db.companies.find({ "relationships.0.person.last_name": "Zuckerberg" }, { "name": 1 }).pretty()`

`db.companies.find({ "relationships.0.person.first_name": "Mark", "relationships.0.title": { "$regex": "CEO" } }, { "name": 1 }).count()`

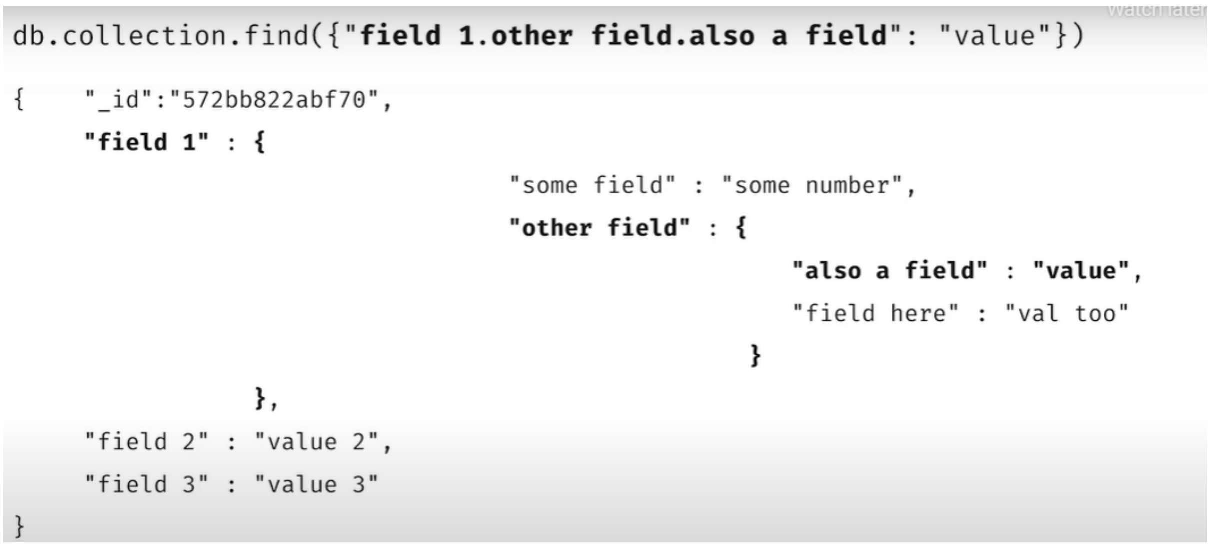
```
db.companies.find({ "relationships.0.person.first_name": "Mark",
"relationships.0.title": {"$regex": "CEO" } }, { "name": 1 }).pretty()
```

```
db.companies.find({ "relationships": { "$elemMatch": { "is_past": true,
"person.first_name": "Mark" } } }, { "name": 1 }).pretty()
```

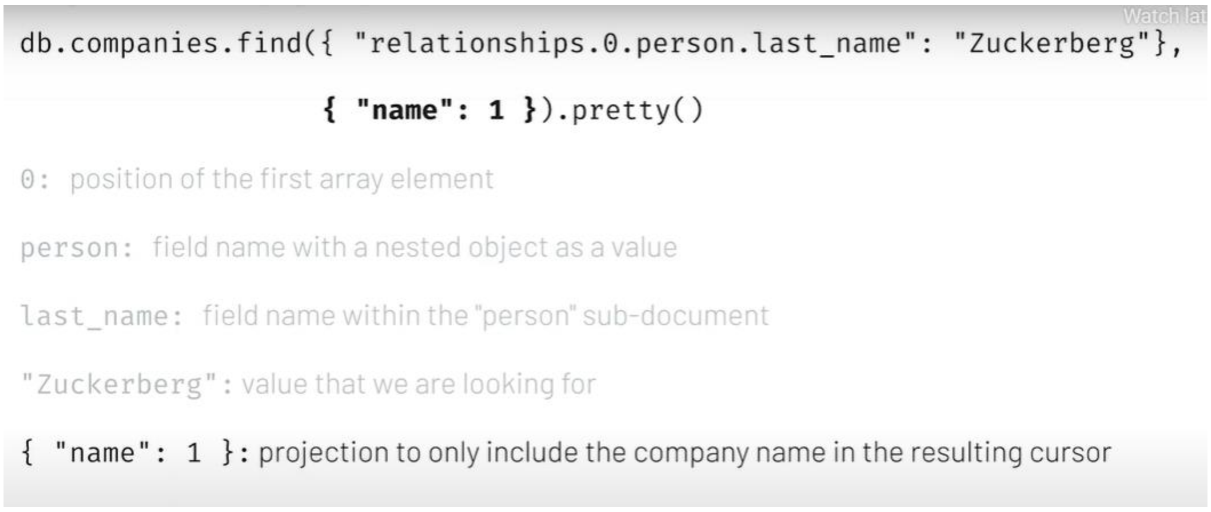
```
db.companies.find({ "relationships": { "$elemMatch": { "is_past": true,
"person.first_name": "Mark" } } }, { "name": 1 }).count()
```



```
db.trips.findOne({"start station location.type": "Point"})

{     "_id":"572bb8222b288919b68abf70",
      …
      "start station location" : {
                                    "type" : "Point",
                                    "coordinates" : [
                                                      -73.97966069,
                                                      40.74394314
                                                    ]
                                  },
 … }
```



```
db.collection.find({"field 1.other field.also a field": "value"})

{     "_id":"572bb822abf70",
      "field 1" : {
                          "some field" : "some number",
                          "other field" : {
                                              "also a field" : "value",
                                              "field here" : "val too"
                                            }
                  },
      "field 2" : "value 2",
      "field 3" : "value 3"
}
```



```
db.companies.find({ "relationships.0.person.last_name": "Zuckerberg"},
                     { "name": 1 }).pretty()
```

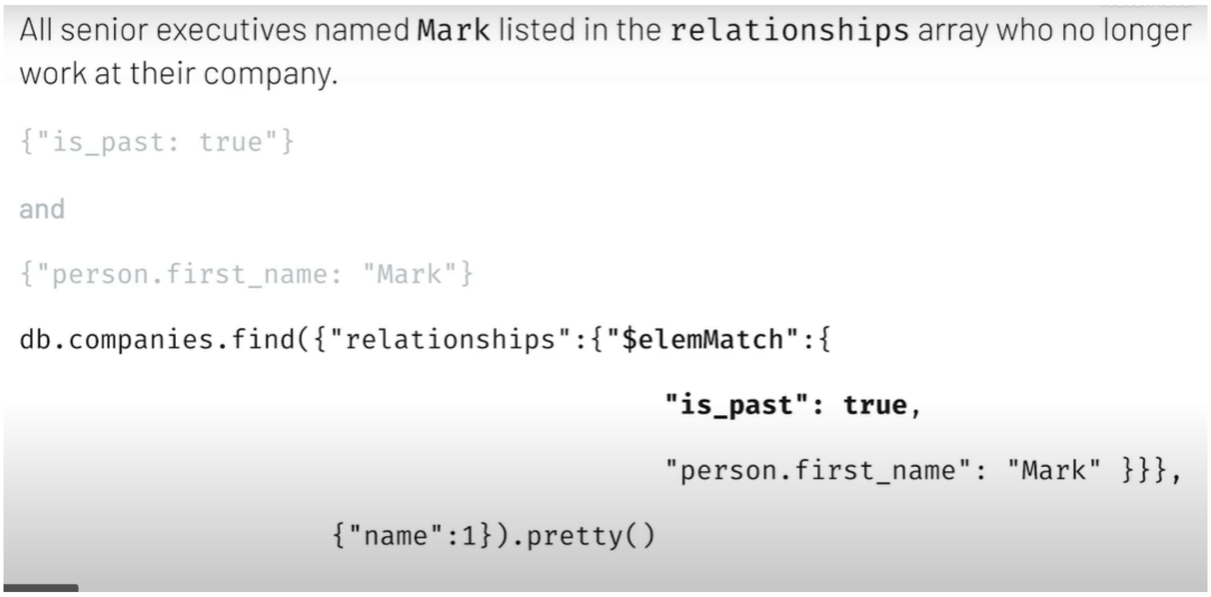
`0:` position of the first array element

`person:` field name with a nested object as a value

`last_name:` field name within the "person" sub-document

`"Zuckerberg":` value that we are looking for

`{ "name": 1 }:` projection to only include the company name in the resulting cursor



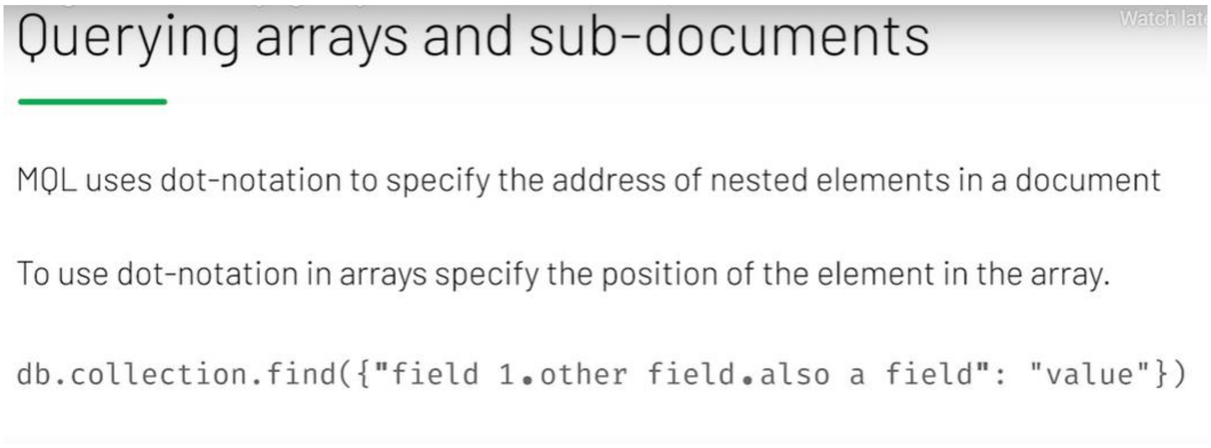
All senior executives named `Mark` listed in the `relationships` array who no longer work at their company.

`{"is_past: true"}`

and

`{"person.first_name: "Mark"}`

```
db.companies.find({"relationships":{"$elemMatch":{
                                        "is_past": true,
                                        "person.first_name": "Mark" }}},
                    {"name":1}).pretty()
```



Querying arrays and sub-documents

MQL uses dot-notation to specify the address of nested elements in a document

To use dot-notation in arrays specify the position of the element in the array.

```
db.collection.find({"field 1.other field.also a field": "value"})
```