MongoDB – Part 2

Abdu Alawini

University of Illinois at Urbana-Champaign

CS411: Database Systems

Some slides were adopted from D. Maier. S. Davidson with permission

Learning Objectives

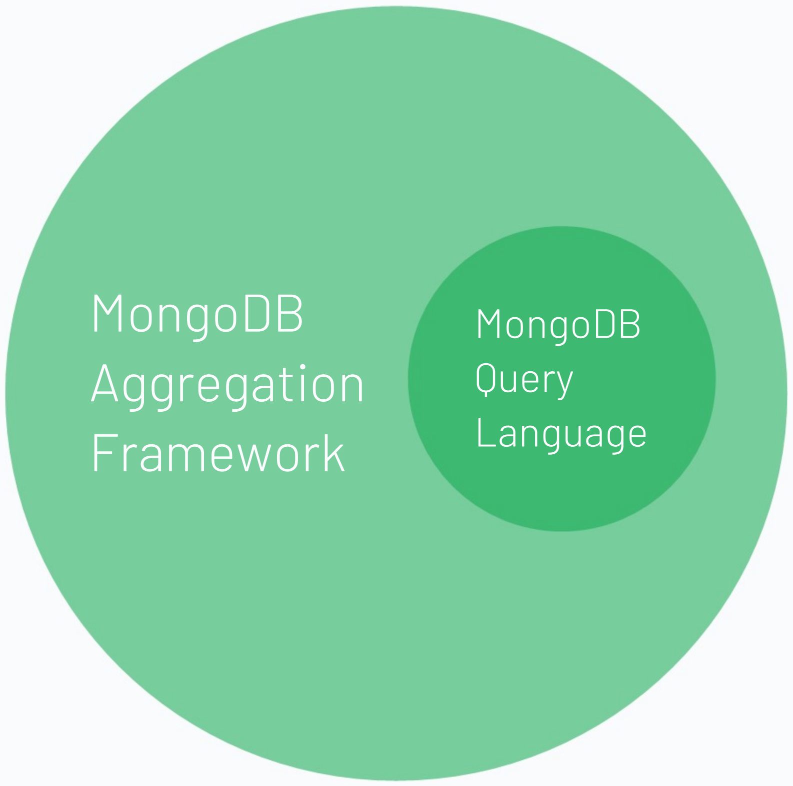
After this lecture, you should be able to:

Write MongoDB cursor queries.

Write aggregation queries

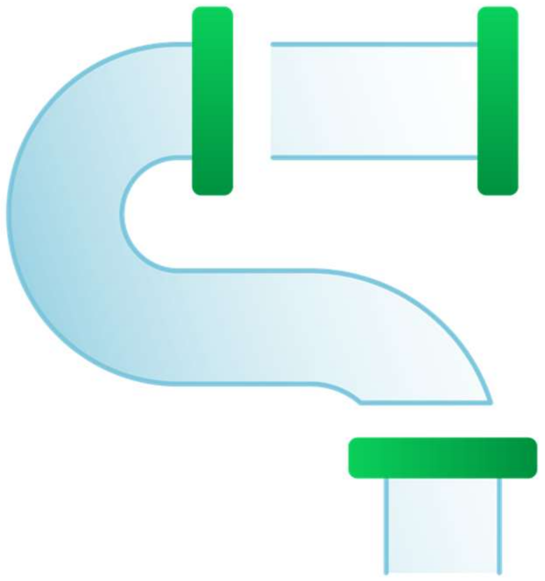
Query documents by reference.

The MongoDB Aggregation Framework



Aggregation Framework

Extends what can be done with data in MongoDB beyond MQL.



Why Aggregation?

Process documents and return computed results

Wider set of functionality than available in MQL

Applies a sequence of query operations that can reduce and transform the documents

What is the Aggregation Framework?

A framework that supports complex manipulation of documents

Stages: sequentially performs an operation or set of operations within a pipeline of stages

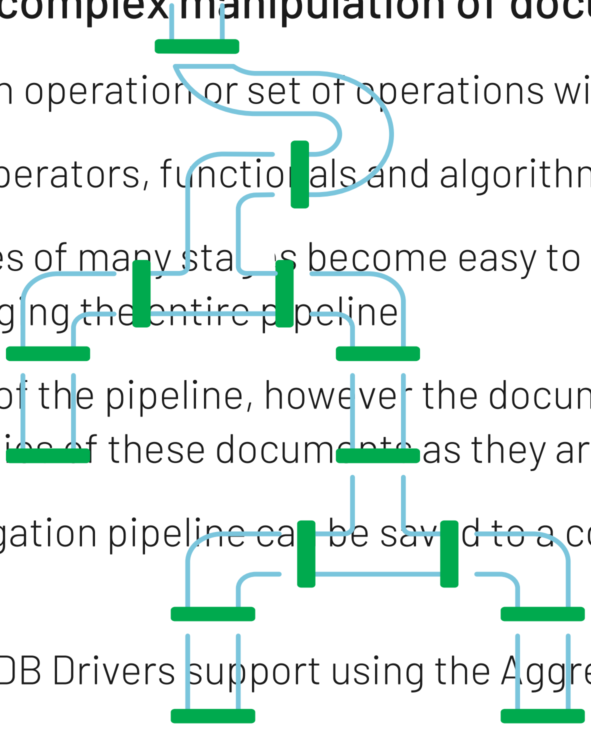
Expressions: a large toolkit of operators, functionals and algorithms that can be used

Easy to debug: complex pipelines of many stages become easy to debug as problems can be localised to a single stage rather than debugging the entire pipeline

Input: a collection is the source of the pipeline, however the documents in this collection are not modified. The pipeline holds copies of these documents as they are modified through the various stages.

Outputs: the output of an aggregation pipeline can be saved to a collection or made available to applications as a cursor

Driver support: All of the MongoDB Drivers support using the Aggregation Framework



Aggregation Framework Stages

Aggregation Stage	MQL find() equivalent
`$match`	`find(<query>)`
`$projection`	`find(<query>, projection)`
`$sort`	`find(<query>).sort(order)`
`$limit`	`find(<query>).limit(num)`
`$skip`	`find(<query>).skip(num)`
`$count`	`find(<query>).count()`

Aggregation Framework Stages

$facet

$project

$sort

$lookup

$match

$group

$unionWith

$addFields

$unwind

and more ...

One Query: Two Approaches

Let’s insert some real data on cows!

```
>>> for(c=0;c<1000;c++) {
farm_id = Math.floor((Math.random()*5)+1);
db.cowCol.insertOne({ name: "daisy", milk: c, farm: farm_id} );
}

{
          acknowledged : true,
          insertedIds :      ObjectId(5f2aefa8fde88235b959f0b1a),
}
```

See: https://docs.mongodb.com/manual/reference/method/db.collection.insertOne/

Syntax

Using the Web shell window, we will run a query to find the first ten (10) documents for farm '1' using MQL (find) and using Aggregation (aggregate).

```
db.cowCol.find(
{"farm": 1},
{"name": 1, "milk": 1, "_id": 0}).limit(10).pretty()

db.cowCol.aggregate([
{ $match: { "farm": 1 }},
{ $project:{ "name": 1, "milk": 1, "_id": 0 }},
{ $limit: 10 }
])
```

Syntax

Let's focus on the Aggregation Framework syntax:

```
cowCol.aggregate([
{ $match: { "farm": 1 }},
{ $project:{ "name": 1, "milk": 1, "_id": 0 }},
{ $limit: 10 }
])
```

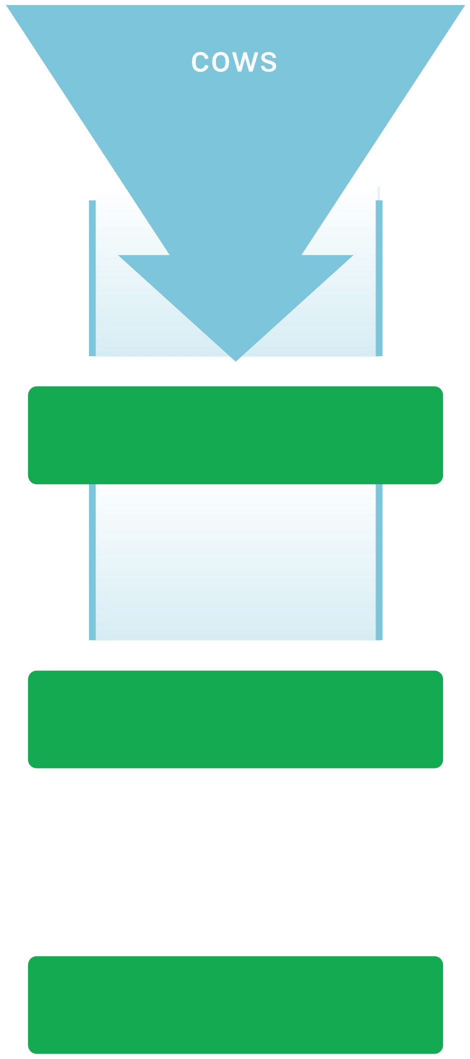
Array

Syntax

Let's focus on the Aggregation Framework syntax:

```
cowCol.aggregate([
{ $match: { "farm": 1 }},
{ $project:{ "name": 1, "milk": 1,
"_id": 0 }},
{ $limit: 10 }
])
```

Documents



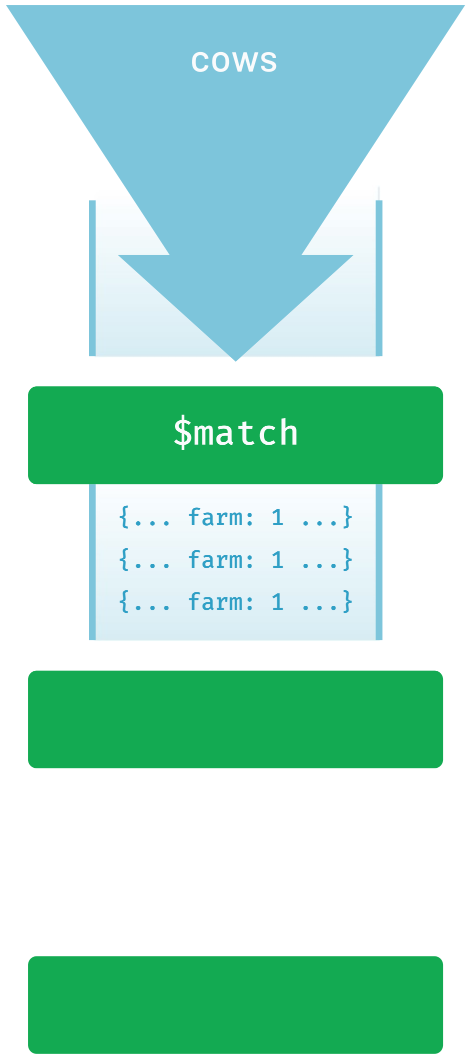
```
[

  { $match:
   {"farm":"1}   },

  { $project:
    {"name": 1,
        "milk": 1,
        "_id": 0 }
  },

  { $limit: 10 }

]
```



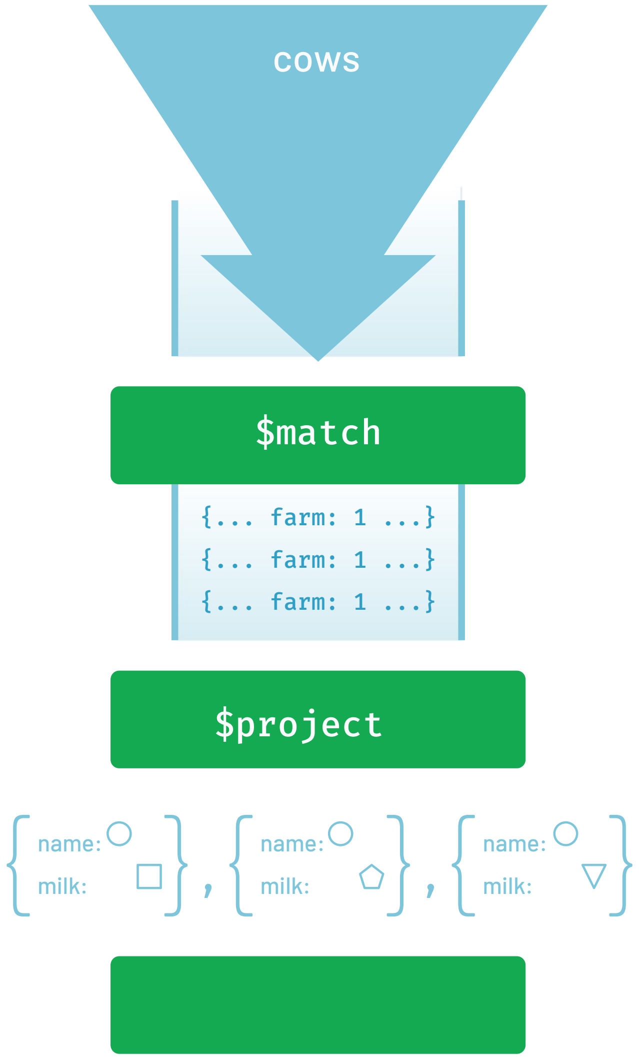
```
[

  { $match:
   {"farm":"1}   },

  { $project:
    {"name": 1,
        "milk": 1,
        "_id": 0 }
  },

  { $limit: 10 }

]
```



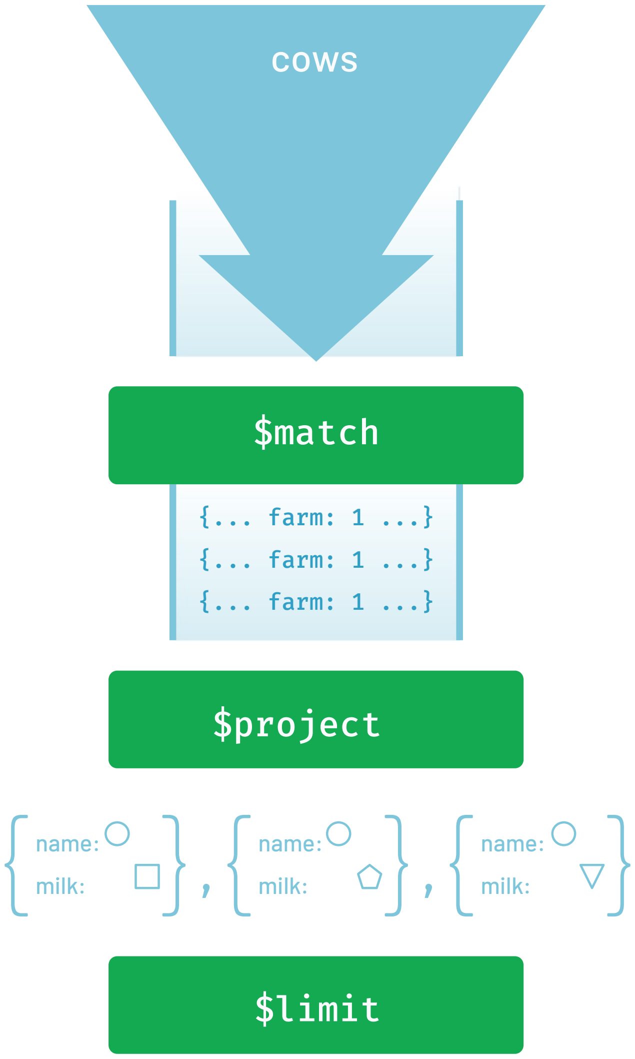
```
[

  { $match:
   {"farm":"1}   },

  { $project:
    {"name": 1,
         "milk": 1,
         "_id": 0 }
  },

  { $limit: 10 }

]
```



```
[

  { $match:
   {"farm":"1}   },

  { $project:
    {"name": 1,
         "milk": 1,
         "_id": 0 }
  },

  { $limit: 10 }

]
```

Results

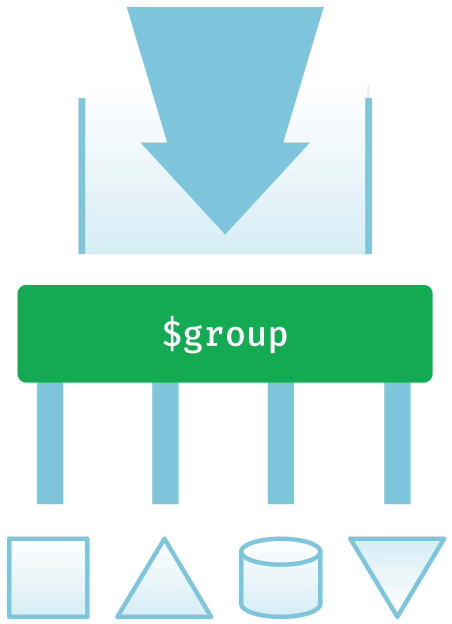
```
cowCol.find(
{"farm": 1},
{"name": 1, "milk": 1, "_id": 0}).limit(10).pretty()
```

```
{ "name" : "daisy", "milk" : 4 }
{ "name" : "daisy", "milk" : 6 }
{ "name" : "daisy", "milk" : 14 }
{ "name" : "daisy", "milk" : 20 }
{ "name" : "daisy", "milk" : 31 }
{ "name" : "daisy", "milk" : 34 }
{ "name" : "daisy", "milk" : 43 }
{ "name" : "daisy", "milk" : 44 }
{ "name" : "daisy", "milk" : 55 }
{ "name" : "daisy", "milk" : 71 }
```

```
cowCol.aggregate([{ $match: { "farm": 1 }},
{ $project:{ "name": 1, "milk": 1, "_id": 0 }},
{ $limit: 10 }])
```

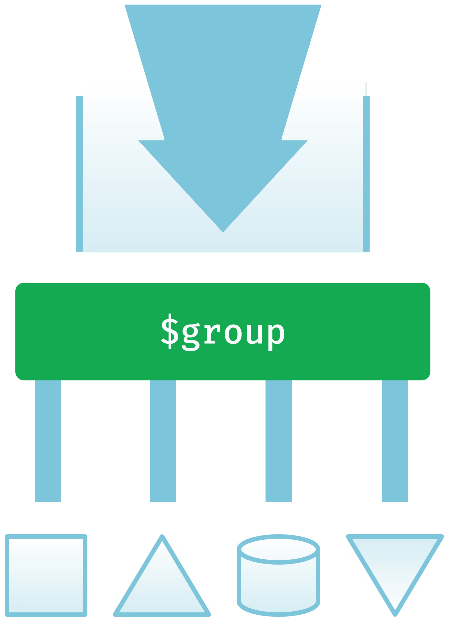
```
{ "name" : "daisy", "milk" : 4 }
{ "name" : "daisy", "milk" : 6 }
{ "name" : "daisy", "milk" : 14 }
{ "name" : "daisy", "milk" : 20 }
{ "name" : "daisy", "milk" : 31 }
{ "name" : "daisy", "milk" : 34 }
{ "name" : "daisy", "milk" : 43 }
{ "name" : "daisy", "milk" : 44 }
{ "name" : "daisy", "milk" : 55 }
{ "name" : "daisy", "milk" : 71 }
```

$group



$group

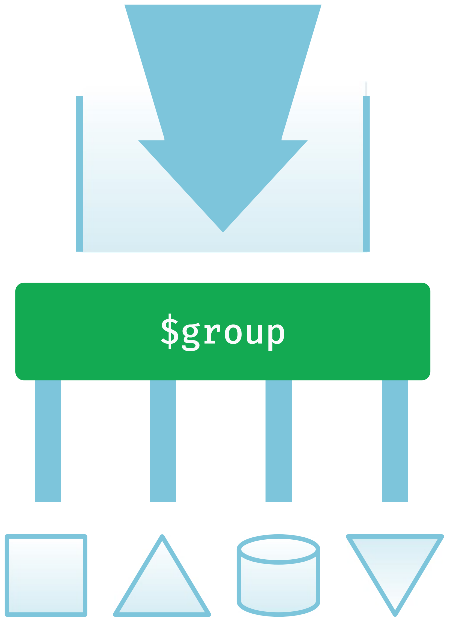
This stage that takes the incoming stream of documents, and segments it. Each group is represented by a single document.



```
[

  { $group:
   { _id: "$farm",
     total_milk:
     { $sum: "$milk" }
   }
  }

]
```



```
[

  { $group:
   { _id: "$farm",
     total_milk:
     { $sum: "$milk" }
   }
  }

]
```

Results

```
cowCol.aggregate([{ $group:
{ _id: "$farm", total_milk: {$sum: "$milk"} }}])
```

```
{ "_id" : 4, "total_milk" : 96562 }
{ "_id" : 1, "total_milk" : 110104 }
{ "_id" : 2, "total_milk" : 99335 }
{ "_id" : 5, "total_milk" : 108357 }
{ "_id" : 3, "total_milk" : 85142 }
```

Aggregation: $unwind

Deconstructs an array field to output a document for each element.

```
Posts:    {
          _id :
ObjectId("4c4ba5c0672c685e5e8aabf3"),
          author : "Kevin",
          date : new Date("February 2, 2012"),
          text : "About MongoDB...",
          birthyear:  1980,
          tags : [ "tech", "databases" ]
          }
```

```
>db.posts.aggregate( { $project : { author : 1, tags : 1 }}, { $unwind : "$tags" } )
```

Result of unwind

```
>db.posts.aggregate( { $project : { author : 1, tags : 1 }}, { $unwind : "$tags" } )
```

```
{ "_id" : ObjectId("4c4ba5c0672c685e5e8aabf3"),
                 "author" : "Kevin",
                 "tags" : ”tech" },
{ "_id" : ObjectId("4c4ba5c0672c685e5e8aabf3"),
                 "author" : "Kevin",
                 "tags" : ”databases" }
```

$lookup

Aggregation Framework $unwind

Performs a left outer join to a 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.

$lookup Syntax

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

$lookup Example

customers

```
{
        _id: 1
        "CustID" : "FRANR",
         "Address" : 54,
        "City" : "rue Royale",
         …
        "Fax" : "40.32.21.21",
        "field11" : "40.32.21.20”  }
}
```

orders

```
{
       "_id" : 10,
       "OrderID" : 10268,
       "CustomerID" : " FRANR",
       "EmployeeID" : 8,
       "OrderDate" : "1996-07-30",
       …
```

Example Query

```
db.customers.aggregate([
  {$lookup:
      {
      from:"orders",
      localField:"CustID",
      foreignField:"CustomerID",
      as:"Cust_Orders“
      }
  }
])
```

Results

```
{
        "_id" : 1,
        "CustID" : "FRANR",
         "Address" : 54,
        "City" : "rue Royale",
         …
        "Fax" : "40.32.21.21",
        "field11" : "40.32.21.20“,
        "Cust_Orders" : [
                            {"_id" : 10,
                            "OrderID" : 10268,
                            "CustomerID" : " FRANR",
                            "EmployeeID" : 8,
                            "OrderDate" : "1996-07-30",
           }
         ]
 }
```

Relationships: Referenced

People:

```
{  _id: 1,
    name: { first: “John”, last: “Backus” },
    birthyear: 1924,
   contribs: [ “Fortran”, “ALGOL”,
                      “Backus Naur Form”, “FP” ],
    awards: [ { award_id: “NMS001”, year: 1975 },
                  { award_id: “TA99”,  year: 1977} ]   }
```

Awards:

```
 {_id: “NMS001”,
  title:  “National Medal of Science” ,
  by: “National Science Foundation”},
 {_id: “TA99”,
  title:  “Turing Award”,
  by: “ACM” }
```

Let’s create a new collection for people

```
    db.people.insertOne(
        {  _id: 1, name: { first: "John", last:
                     "Backus" },
            birthyear: 1924,
            contribs: [ "Fortran", "ALGOL", "Backus
                  Naur Form", "FP" ],
            awards: [
                     { award_id: "NMS001", year:
               1975 },
                     { award_id: "TA99",  year: 1977} ]   }
     )
```

Let’s create another collection for awards

```
db.awards.insertMany([
  {   _id: "NMS001", title:  "National Medal of
      Science" ,
      by: "National Science Foundation"},
  {   _id: "TA99", title:  "Turing Award",
      by: "ACM" },
      {_id:"CA85", title:"Career Award",
      by:"NIH" , year: 1980},
  {   _id:"CSA19", title:"Career Success Award",
      by: "NASA" , year: 1982}])
```

Cursor Way

```
>>>let n_award= db.awards.find ({title: /^n/i},{_id:1});

>>>let award_Ids = new Array();

>>>while (n_award.hasNext())
     {
     let tmp = award_Ids.push(n_award.next()._id)
     }

>>>db.people.find({"awards.award_id":{$in: award_Ids}})
```

$lookup Way

```
db.people.aggregate([
  {$lookup: {
                 from: "awards",
                 localField:"awards.award_id",
                 foreignField: "_id",
                 as: "awardsInfo"} },
  {$unwind: "$awardsInfo"},
  {$match:{"awardsInfo.title":/^n/i}}
])
```

Further Learning on MongoDB University

M121 MongoDB Aggregation