Module 3

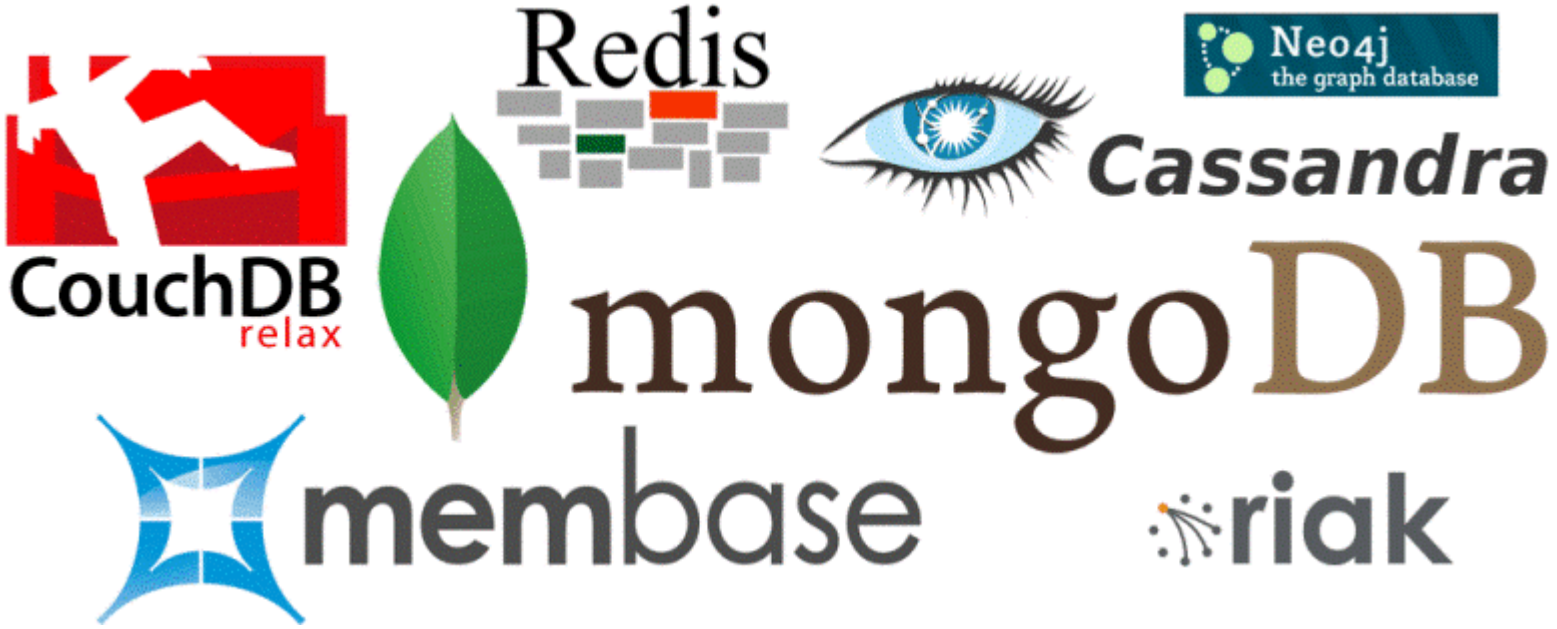
NoSQL Databases

Ismael Navas-Delgado – University of Málaga

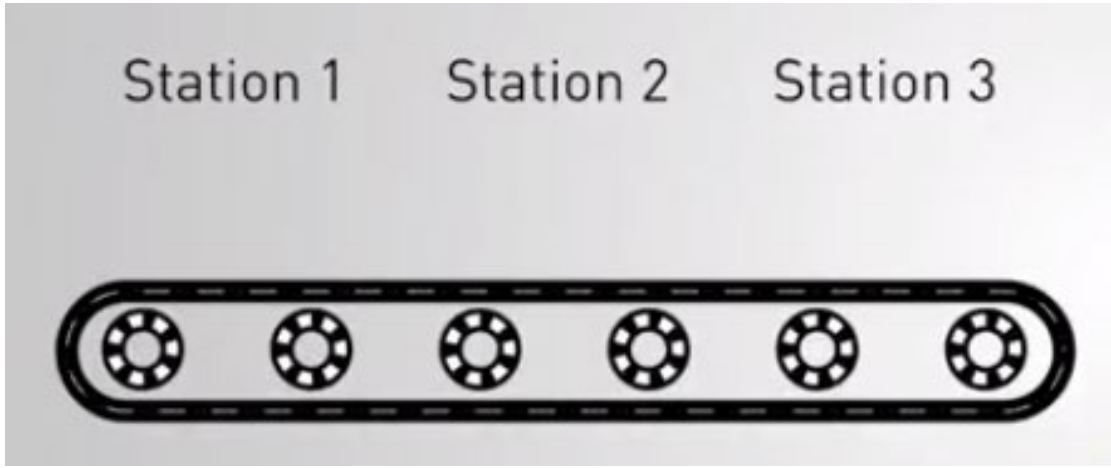
Table of contents

- C**R**UD Operations
  - Aggregate
    - Mongo Atlas
    - Mongo Compass
    - Python

Introduction

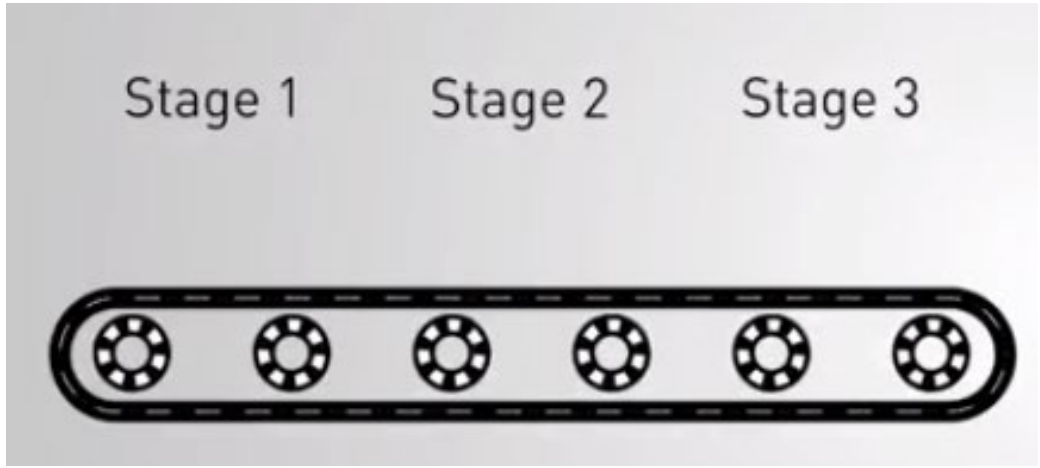


CRUD



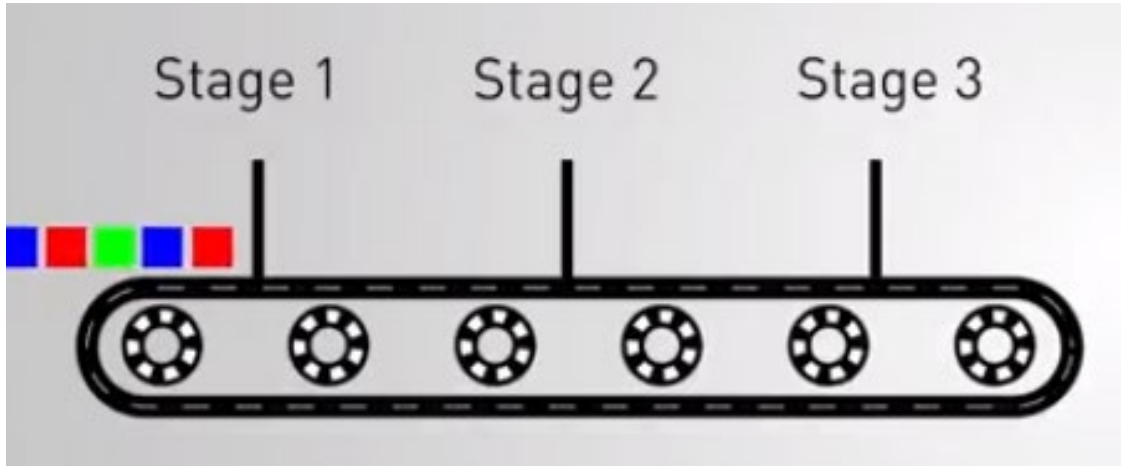
- Pipelines

CRUD



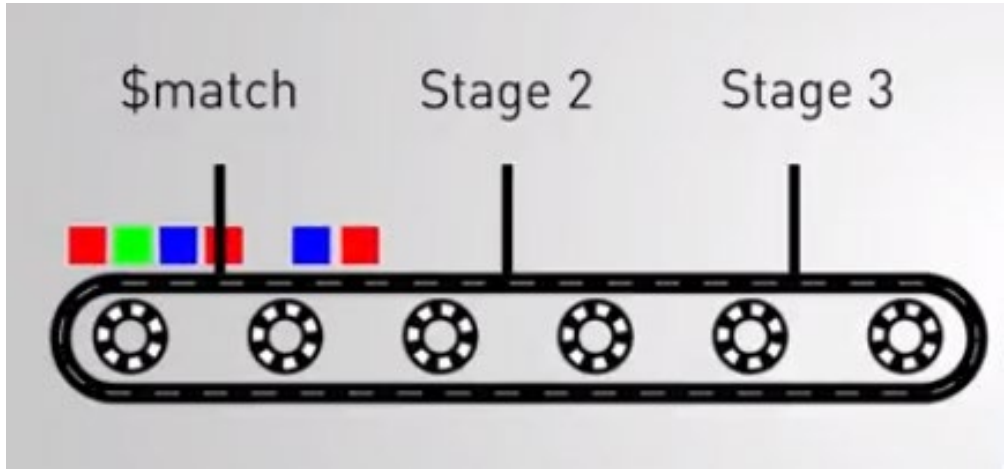
- Pipelines

CRUD



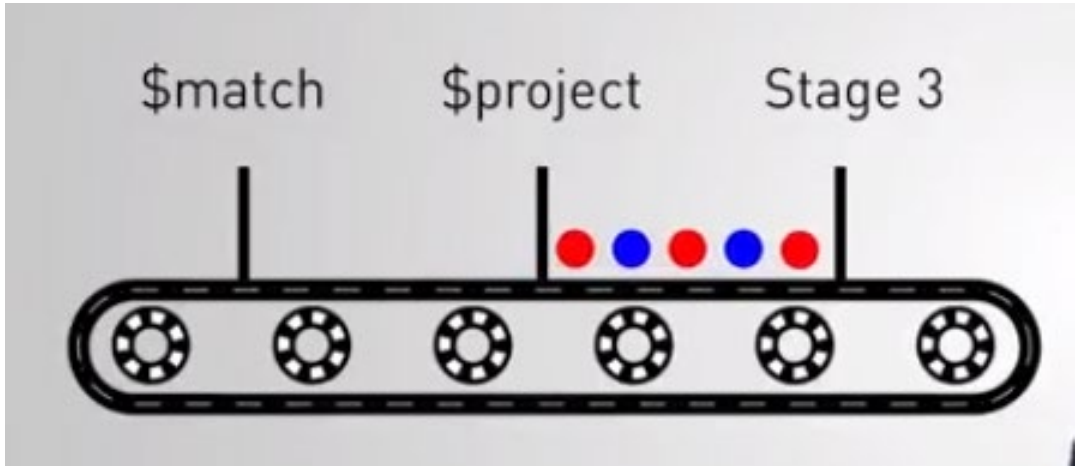
- Pipelines

CRUD



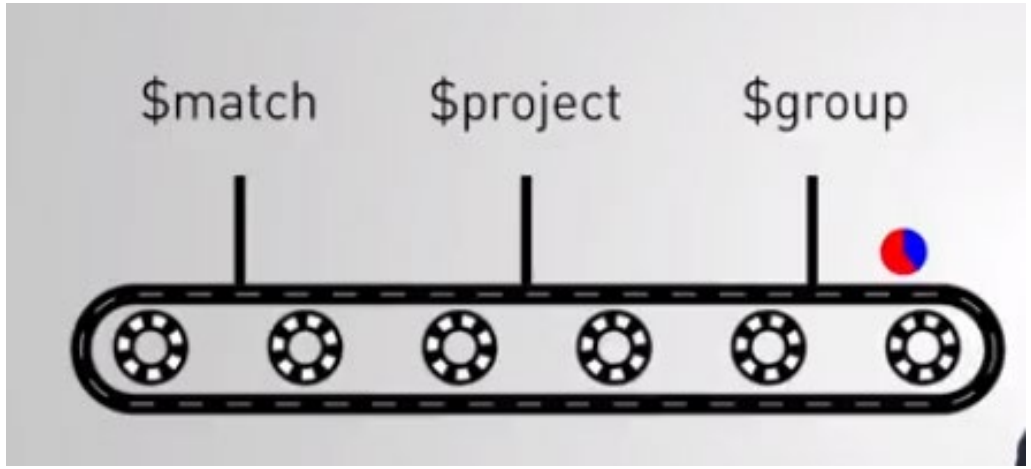
- Pipelines

CRUD



- Pipelines

CRUD



- Pipelines

CRUD

Cluster in Atlas for examples:

mongodb://m121:aggregations@cluster0-shard-00-00-jxeqq.mongodb.net:27017/?authSource=admin&readPreference=primary&ssl=true

CRUD

- Data Aggregation with the mongo Shell
  - MongoDB can perform aggregation operations, such as grouping by a specified key and evaluating a total or a count for each distinct group.
  - Use the aggregate() method to perform a stage-based aggregation. The aggregate() method accepts as its argument an array of stages, where each stage, processed sequentially, describes a data processing step.
    - db.collection.aggregate([<stage1>, <stage2>, ...])

https://docs.mongodb.com/manual/meta/aggregation-quick-reference/

CRUD

- Data Aggregation Example at Atlas

```
db.solarSystem.aggregate(
[
{  "$match":
     {    "atmosphericComposition": { "$in": [/O2/] },
     "meanTemperature": { $gte: -40, "$lte": 40 }  }
},
{  "$project":
     {    "_id": 0,    "name": 1,    "hasMoons": { "$gt": ["$numberOfMoons", 0] }  }
}
],{ "allowDiskUse": true});
```

CRUD

- Aggreation Operators (STAGES)

```
db.solarSystem.aggregate(
[
{  "$match":
     {   "atmosphericComposition": { "$in": [/O2/] },
     "meanTemperature": { $gte: -40, "$lte": 40 }  }
},
{  "$project":
     {   "_id": 0,   "name": 1,   "hasMoons": { "$gt": ["$numberOfMoons", 0] }  }
}
],{ "allowDiskUse": true});
```

CRUD

- Query Operators

```
db.solarSystem.aggregate(
[
{  "$match":
     {   "atmosphericComposition": { "$in": [/O2/] },
     "meanTemperature": { $gte: -40, "$lte": 40 }  }
},
{  "$project":
     {   "_id": 0,   "name": 1,   "hasMoons": { "$gt": ["$numberOfMoons", 0] }  }
}
],{ "allowDiskUse": true});
```

CRUD

- Operators → Key Position

```
db.solarSystem.aggregate(
[
{  "$match":
     {   "atmosphericComposition": { "$in": [/O2/] },
     "meanTemperature": { $gte: -40, "$lte": 40 }  }
},
{  "$project":
     {   "_id": 0,   "name": 1,   "hasMoons": { "$gt": ["$numberOfMoons", 0] }  }
}
],{ "allowDiskUse": true});
```

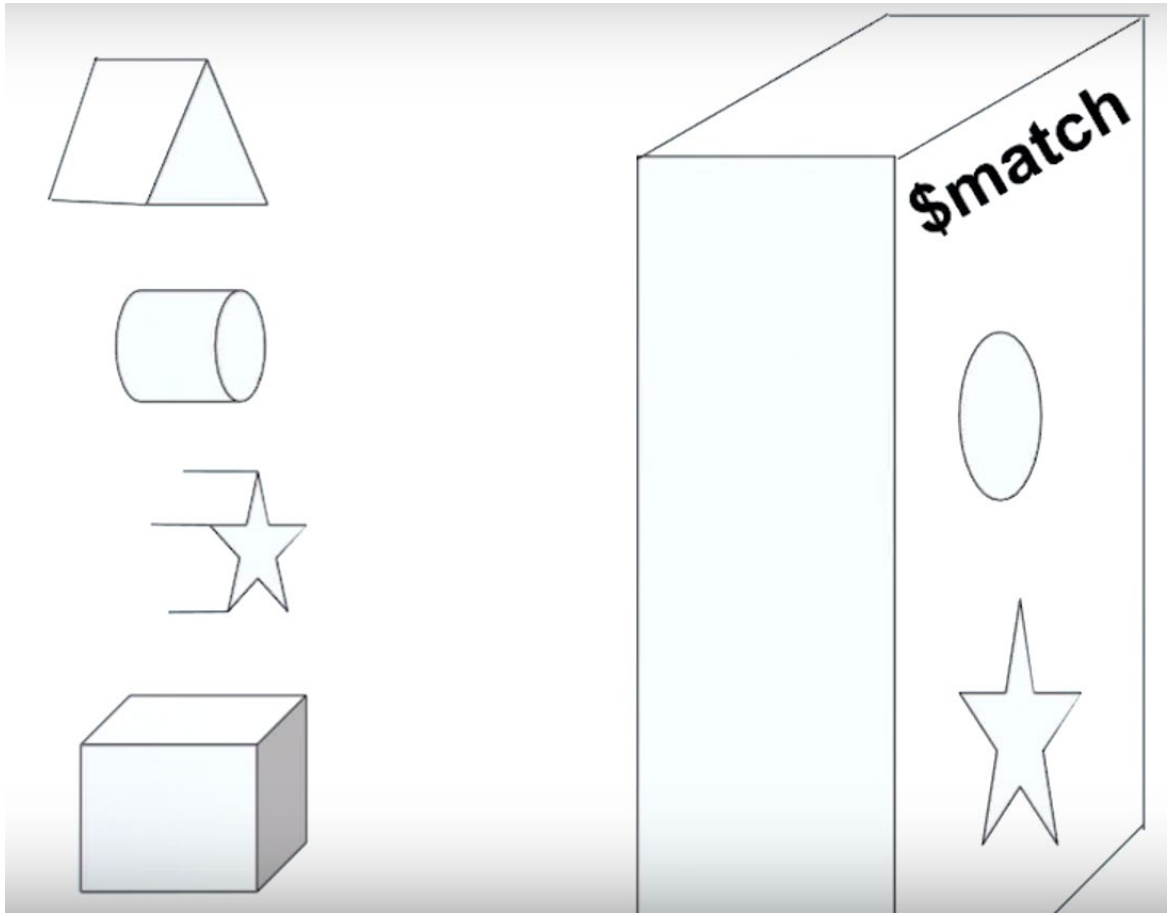
CRUD

- (Expressions) Field in a document → $field

```
db.solarSystem.aggregate(
[
{  "$match":
     {   "atmosphericComposition": { "$in": [/O2/] },
     "meanTemperature": { $gte: -40, "$lte": 40 }  }
},
{  "$project":
     {   "_id": 0,   "name": 1,   "hasMoons": { "$gt": ["$numberOfMoons", 0] }  }
}
],{ "allowDiskUse": true});
```

CRUD

- Filtering Documents → $match
  - Can have several $match stages
  - After $match you can use almost any stage
  - Same syntax as find



CRUD

- Filtering Documents → $match
  - Can have several $match stages
  - After $match you can use almost any stage

```
// $match all celestial bodies, not equal to Star
db.solarSystem.aggregate([
      {  "$match": { "type": { "$ne": "Star" } }}
]).pretty()

// same query using find command
db.solarSystem.find({ "type": { "$ne": "Star" } }).pretty();
```

CRUD

- Filtering Documents → $match
  - Limitations:
    - You cannot use $where operator
    - If you use $test operator, $match has to be the first stage
    - If $match is the first stage in the pipeline you can take advantage of indexes to accelerate queries
    - You should put $match at early stages of pipelines
    - It does not have projection (you can not directly select which fields to show in the results). But, we can do this and much more with other operators

    - https://docs.mongodb.com/manual/reference/operator/aggregation/match?jmp=university

CRUD

- Filtering Documents → $count

```
// count the number of matching documents
db.solarSystem.count();


// using $count
db.solarSystem.aggregate([
        {  "$match": { "type": { "$ne": "Star"} }},
        {  "$count": "planets"}
]);
```

CRUD

- Exercise
  - After connecting to the cluster, ensure you can see the movies collection by typing show collections and then run the command db.movies.findOne(). Take a moment to familiarize yourself with the schema.
  - Help MongoDB pick a movie our next movie night! Based on employee polling, we've decided that potential movies must meet the following criteria.
    - imdb.rating is at least 7
    - genres does not contain "Crime" or "Horror"
    - rated is either "PG" or "G"
    - languages contains "English" and "Japanese"

CRUD

- Exercise
  - Help MongoDB pick a movie our next movie night! Based on employee polling, we've decided that potential movies must meet the following criteria.
    - imdb.rating is at least 7
    - genres does not contain "Crime" or "Horror"
    - rated is either "PG" or "G"
    - languages contains "English" and "Japanese“
  - Help
    - https://docs.mongodb.com/manual/reference/operator/aggregation/in/
    - $nin
    - https://docs.mongodb.com/manual/reference/operator/query/all/

CRUD

Stage 1 $match

```
{
  "imdb.rating": {
    $gte: 7,
  },
  genres: {
    $nin: ["Crime", "Horror"],
  },
  rated: {
    $in: ["PG", "G"],
  },
  languages: {
    $all: ["English", "Japanese"],
  },
}
```

CRUD

- Shaping Documents → $project
  - Like a map function
  - Not only projects
    - Create new fields
    - Modify fields
    - Project fields
  - https://docs.mongodb.com/manual/reference/operator/aggregation/project?jmp=university

CRUD

- Shaping Documents → $project
  - Not only projects
    - Create new fields
    - Modify fields
    - **Project fields**

```
// project ``name`` and remove ``_id``
db.solarSystem.aggregate([
        { "$project": { "_id": 0, "name": 1 } }
]);
```

Do it also with find!

CRUD

- Shaping Documents → $project
  - Not only projects
    - Create new fields
    - Modify fields
    - **Project fields**

```
// project ``name`` and ``gravity`` fields, including default ``_id``
db.solarSystem.aggregate([
        { "$project": { "name": 1, "gravity": 1 } }
]);
```

Do it also with find!

CRUD

- Shaping Documents → $project
  - Not only projects
    - Create new fields
    - Modify fields
    - **Project fields**

```
// using dot-notation to express the projection fields
db.solarSystem.aggregate([
        { "$project": { "_id": 0, "name": 1, "gravity.value": 1 } }
]);
```

Do it also with find!

CRUD

- Shaping Documents → $project
  - Not only projects
    - **Create new fields**
    - Modify fields
    - **Project fields**

```
// reassing ``gravity`` field with value from ``gravity.value`` embeded field
db.solarSystem.aggregate([
        {"$project": { "_id": 0, "name": 1, "gravity": "$gravity.value" }}
]);
```

Do it also with find?

CRUD

- Shaping Documents → $project
  - Not only projects
    - **Create new fields**
    - Modify fields
    - **Project fields**

```
// creating a document new field ``surfaceGravity``
db.solarSystem.aggregate([
        {"$project": { "_id": 0, "name": 1, "surfaceGravity": "$gravity.value" }}
]);
```

CRUD

- Shaping Documents → $project
  - Not only projects
    - **Create new fields**
    - Modify fields
    - **Project fields**

```
// creating a new field ``myWeight`` using expressions
db.solarSystem.aggregate([
        {"$project": { "_id": 0, "name": 1,
                "myWeight": { "$multiply": [ { "$divide": [ "$gravity.value", 9.8 ] }, 75 ] } }}
]);
```

CRUD

- Shaping Documents → $project
  - Exercise 1:
    - Our first movie night was a success. Unfortunately, our ISP called to let us know we're close to our bandwidth quota, but we need another movie recommendation!
    - Using the same $match stage from the previous lab, add a $project stage to only display the the title and film rating (title and rated fields).

CRUD

- Shaping Documents → $project
  - Exercise 1:
    - Our first movie night was a success. Unfortunately, our ISP called to let us know we're close to our bandwidth quota, but we need another movie recommendation!
    - Using the same $match stage from the previous lab, add a $project stage to only display the the title and film rating (title and rated fields).

CRUD

- Shaping Documents → $project
  - Exercise 2:
    - Our movies dataset has a lot of different documents, some with more convoluted titles than others. If we'd like to analyze our collection to find movie titles that are composed of only one word, we could fetch all the movies in the dataset and do some processing in a client application, but the Aggregation Framework allows us to do this on the server!
    - Using the Aggregation Framework, **find a count of the number of movies that have a title composed of one word**. To clarify, "Cinderella" and "3-25" should count, where as "Cast Away" would not.
    - Make sure you look into the $split String expression and the $size Array expression
    - To get the count, you can append itcount() to the end of your pipeline
    - db.movies.aggregate([...]).itcount()

CRUD

Stage 1 $match

```
{
  $expr: {
    $eq: [
      {
        $size: {
          $split: ["$title", " "],
        },
      },
      1,
    ],
  },
}
```

Stage 2 $group

```
{
    _id: null,
    count: {
      $sum: 1,
    },
}
```

CRUD

- Shaping Documents → $project
  - Exercise 3:
    - This lab will have you work with data within arrays, a common operation.
    - Specifically, one of the arrays you'll work with is writers, from the movies collection.
    - There are times when we want to make sure that the field is an array, and that it is not empty. We can do this within $match
      - { $match: { writers: { $elemMatch: { $exists: true } } }
    - However, the entries within writers presents another problem. A good amount of entries in writers look something like the following, where the writer is attributed with their specific contribution
      - "writers" : ["Vincenzo Cerami (story)", "Roberto Benigni (story)"]
    - But the writer also appears in the cast array as "Roberto Benigni"!
      - db.movies.findOne({title: "Life Is Beautiful"}, { _id: 0, cast: 1, writers: 1})

CRUD

- Shaping Documents → $project
  - Exercise 3:
    - This presents a problem, since comparing "Roberto Benigni" to "Roberto Benigni (story)" will definitely result in a difference.
    - Thankfully there is a powerful expression to help us, $map. $map lets us iterate over an array, element by element, performing some transformation on each element. The result of that transformation will be returned in the same place as the original element.
    - Within $map, the argument to input can be any expression as long as it resolves to an array. The argument to as is the name we want to use to refer to each element of the array when performing whatever logic we want, surrounding it with quotes and prepending two $ signs. The field as is optional, and if omitted each element must be referred to as "$$this"

```
writers: {
  $map: {
    input: "$writers",
    as: "writer",
    in: "$$writer"
```

    - in is where the work is performed. Here, we use the $arrayElemAt expression, which takes two arguments, the array and the index of the element we want. We use the $split expression, splitting the values on " (".

CRUD

- Shaping Documents → $project
  - Exercise 3:
    - If the string did not contain the pattern specified, the only modification is it is wrapped in an array, so $arrayElemAt will always work

```
writers: {
 $map: {
  input: "$writers",
  as: "writer",
  in: {
   $arrayElemAt: [
    {
     $split: [ "$$writer", " (" ]
    },
    0
   ]
  }
 }
}
```

CRUD

- Shaping Documents → $project
  - Let's find how many movies in our movies collection are a "labor of love", where **the same person appears in cast, directors, and writers**
  - Note that you may have a dataset that has duplicate entries for some films. Don't worry if you count them few times, meaning you should not try to find those duplicates.

CRUD

- Shaping Documents → $project
  - To get a count after you have defined your pipeline, there are two simple methods.

```
// add the $count stage to the end of your pipeline
// you will learn about this stage shortly!
db.movies.aggregate([
  {$stage1},
  {$stage2},
  ...$stageN,
  { $count: "labors of love" }
])

// or use itcount()
db.movies.aggregate([
  {$stage1},
  {$stage2},
  {...$stageN},
]).itcount()
```

  - How many movies are "labors of love"?

CRUD

- Adding Fields → $addFields
  - Similar to project
  - Simplest to add fields in some cases than using project

```
// reassign ``gravity`` field value
db.solarSystem.aggregate([{"$project": { "gravity": "$gravity.value" } }]);

// adding ``name`` and removing ``_id`` from projection
db.solarSystem.aggregate([{"$project": { "_id": 0, "name": 1, "gravity": "$gravity.value" } }])‘’
```

But, what happens if I want to retain dome other fields?

CRUD

- Adding Fields → $addFields
  - But, what happens if I want to retain dome other fields?

```
// adding more fields to the projected document
db.solarSystem.aggregate([
      {"$project":{
      "_id": 0, "name": 1, "gravity": "$gravity.value", "meanTemperature": 1,
   "density": 1,   "mass": "$mass.value",   "radius": "$radius.value",   "sma": "$sma.value" }}
]);
```

CRUD

- Adding Fields → $addFields
  - But, what happens if I want to retain dome other fields?

```
// using ``$addFields`` to generate the new computed field values
db.solarSystem.aggregate([
    {"$addFields":{   "gravity": "$gravity.value",   "mass": "$mass.value",   "radius": "$radius.value",   "sma": "$sma.value"}}]);
```

CRUD

- Adding Fields → $addFields
  - // combining ``$project`` with ``$addFields``

```
db.solarSystem.aggregate([
    {"$project": {  "_id": 0,  "name": 1,  "gravity": 1,  "mass": 1,  "radius": 1,  "sma": 1}},
    {"$addFields": {  "gravity": "$gravity.value",  "mass": "$mass.value",  "radius": "$radius.value",  "sma": "$sma.value"}}
]);
```

CRUD

- Cursor Like Stages → $sort, $count, $skip, $limit

```
// project fields ``numberOfMoons`` and ``name``
db.solarSystem.find({}, {"_id": 0, "name": 1, "numberOfMoons": 1}).pretty();
// count the number of documents
db.solarSystem.find({}, {"_id": 0, "name": 1, "numberOfMoons": 1}).count();
// skip documents
db.solarSystem.find({}, {"_id": 0, "name": 1, "numberOfMoons": 1}).skip(5).pretty();
// limit documents
db.solarSystem.find({}, {"_id": 0, "name": 1, "numberOfMoons": 1}).limit(5).pretty();
// sort documents
db.solarSystem.find({}, { "_id": 0, "name": 1, "numberOfMoons": 1 }).sort(
{"numberOfMoons": -1 } ).pretty();
```

CRUD

- Cursor Like Stages → $sort, $count, $skip, $limit

```
$limit: { <integer> }

$skip: { <integer> }

$count: { <name we want the count called> }

$sort: { <field we want to sort on>: <integer, direction to sort> }
```

CRUD

- Cursor Like Stages → $sort, $count, $skip, $limit

```
// ``$limit`` stage
db.solarSystem.aggregate([
      {  "$project": {    "_id": 0,    "name": 1,    "numberOfMoons": 1  }},
      { "$limit": 5  }]).pretty();
// ``skip`` stage
db.solarSystem.aggregate([
      {  "$project": {    "_id": 0,    "name": 1,    "numberOfMoons": 1  }},
      {  "$skip": 1}]).pretty()
// ``$count`` stage
db.solarSystem.aggregate([
      {  "$match": {    "type": "Terrestrial planet"  }},
      {  "$project": {    "_id": 0,    "name": 1,    "numberOfMoons": 1  }},
      {  "$count": "terrestrial planets"}]).pretty();
```

CRUD

- Cursor Like Stages → $sort, $count, $skip, $limit

```
//removing ``$project`` stage since it does not interfere with our count
db.solarSystem.aggregate([
      {  "$match": {    "type": "Terrestrial planet"  }},
      {  "$count": "terrestrial planets"}]).pretty();
// ``$sort`` stage
db.solarSystem.aggregate([
      {  "$project": {    "_id": 0,    "name": 1,    "numberOfMoons": 1  }},
      {  "$sort": { "numberOfMoons": -1 }}]).pretty();
// sorting on more than one field
db.solarSystem.aggregate([
      {  "$project": {    "_id": 0,    "name": 1,    "hasMagneticField": 1,    "numberOfMoons": 1  }},
      {  "$sort": { "hasMagneticField": -1, "numberOfMoons": -1 }}]).pretty();
```

CRUD

- Cursor Like Stages → $sort, $count, $skip, $limit
  - Sort at the beginning of the pipeline will use indexes
  - If it is not at the beginning, then sort will be done in memory
  - There is a limit of 100MB for this operation
    - In some cases this will result in an out of memory error
    - We can set up this stage to be done in disk if needed

```
// setting ``allowDiskUse`` option
db.solarSystem.aggregate([
    { "$project": {  "_id": 0,  "name": 1,  "hasMagneticField": 1,  "numberOfMoons": 1 }},
    { "$sort": { "hasMagneticField": -1, "numberOfMoons": -1 }}], { "allowDiskUse": true }).pretty();
```

CRUD

- Cursor Like Stages → $sort, $count, $skip, $limit (Exercise 1)
  - MongoDB has another movie night scheduled. This time, we polled employees for their favorite actress or actor, and got these results

```
favorites = [
  "Sandra Bullock",
  "Tom Hanks",
  "Julia Roberts",
  "Kevin Spacey",
  "George Clooney"]
```

  - For movies released in the USA with a tomatoes.viewer.rating greater than or equal to 3, calculate a new field called num_favs that represets how many favorites appear in the cast field of the movie.
  - Sort your results by num_favs, tomatoes.viewer.rating, and title, all in descending order.
  - What is the title of the 25th film in the aggregation result?

CRUD

- Cursor Like Stages → $sort, $count, $skip, $limit (Exercise 2)
  - Calculate an average rating for each movie in our collection where English is an available language, the minimum imdb.rating is at least 1, the minimum imdb.votes is at least 1, and it was released in 1990 or after. You'll be required to rescale (or normalize) imdb.votes. The formula to rescale imdb.votes and calculate normalized_rating is:

  - What film has the lowest normalized_rating?

CRUD

- Cursor Like Stages → $sort, $count, $skip, $limit (Exercise 2)
  - The formula to rescale imdb.votes and calculate normalized_rating is:
    - // general scaling
      - min + (max - min) * ((x - x_min) / (x_max - x_min))
    - // we will use 1 as the minimum value and 10 as the maximum value for scaling,
    - // so all scaled votes will fall into the range [1,10]
      - scaled_votes = 1 + 9 * ((x - x_min) / (x_max - x_min))//
    - NOTE: We CANNOT simply do 10 * ((x - x_min))..., results will be wrong
  - What film has the lowest normalized_rating?

CRUD

- Cursor Like Stages → $sort, $count, $skip, $limit (Exercise 2)
  - The formula to rescale imdb.votes and calculate normalized_rating is:
    - // Order of operations is important!
    - // use these values for scaling imdb.votes
      - x_max = 1521105
      - x_min = 5
      - min = 1
      - max = 10
      - x = imdb.votes
  - What film has the lowest normalized_rating?

CRUD

- Cursor Like Stages → $sort, $count, $skip, $limit (Exercise 2)
  - The formula to rescale imdb.votes and calculate normalized_rating is:
    - // within a pipeline, it should look something like the following

```
{$add: [     1,
      {$multiply: [        9,
                  {$divide: [{ $subtract: [<x>, <x_min>] }, { $subtract: [<x_max>, <x_min>] }
                  ]
      }]
}   ] }
// given we have the numbers, this is how to calculated normalized_rating
// yes, you can use $avg in $project and $addFields!
normalized_rating = average(scaled_votes, imdb.rating)
```

  - What film has the lowest normalized_rating?

CRUD

- Grouping information → $group

```
{
  $group: {
    _id: <matching/grouping criteria>,
    fieldName: <accumulator expression>,
    … <as many fieldName:expressions as required>
  }
}
```

```
coins: [
  { denomination: 0.01 },
  { denomination: 0.25 },
  { denomination: 0.10 },
  { denomination: 0.05 },
  { denomination: 0.25 },
  …
]
```

```
$group: { _id: "$denomination" }
```

CRUD

- Grouping information → $group

```
// grouping by year and getting a count per year using the { $sum: 1 } pattern
db.movies.aggregate([
    {   "$group":
            {     "_id": "$year",
            "numFilmsThisYear": { "$sum": 1 }    }
}])
```

CRUD

- Grouping information → $group

```
// grouping by year and getting a count per year using the { $sum: 1 } pattern
db.movies.aggregate([
    {   "$group":
            {     "_id": "$year",
            "numFilmsThisYear": { "$sum": 1 }    }
}])
```

Sort the results in descending order!

CRUD

- Grouping information → $group

```
// grouping as before, then sorting in descending order based on the count
db.movies.aggregate([
    {   "$group": {     "_id": "$year",     "count": { "$sum": 1 }   } },
    {   "$sort": { "count": -1 } }])
```

CRUD

- Grouping information → $group

```
// grouping on the number of directors a film has, demonstrating that we have to
// validate types to protect some expressions
db.movies.aggregate([
    {   "$group": {
            "_id": {"numDirectors": {"$cond": [{ "$isArray": "$directors" }, { "$size":
"$directors" }, 0]}     },
            "numFilms": { "$sum": 1 },
            "averageMetacritic": { "$avg": "$metacritic" }    }  },
    {   "$sort": { "_id.numDirectors": -1 }  }])
```

CRUD

- Grouping information → $group

```
// showing how to group all documents together. By convention, we use null or an
// empty string, ""
db.movies.aggregate([ {   "$group": {     "_id": null,     "count": { "$sum": 1 }   } }])
```

CRUD

- Grouping information → $group

```
// filtering results to only get documents with a numeric metacritic value
db.movies.aggregate([
    {   "$match": { "metacritic": { "$gte": 0 } }  },
    {   "$group": {     "_id": null,     "averageMetacritic": { "$avg": "$metacritic" }    } }
])
```

CRUD

- Grouping information → $bucket

```
{
$bucket: {
groupBy: <expression>,
boundaries: [ <lowerbound1>, <lowerbound2>, ... ],
default: <literal>,
output: {
<output1>: { <$accumulator expression> },
...
<outputN>: { <$accumulator expression> }
}
}
}
```

CRUD

- Grouping information → $bucket

```
// grouping by year and getting a count per year using the { $sum: 1 } pattern
db.movies.aggregate([
    {   "$bucket":
            {       groupBy: "$year",
                    boundaries: [ 1900, 1995, 1998, 2000 ],
                    default: "other",
                    output: {   "numFilmsThisYear": { "$sum": 1 }   }
            }
    }])
```

CRUD

- Grouping information → $bucketAuto

CRUD

- Grouping information → $bucketAuto

```
{
$bucketAuto: {
groupBy: <expression>,
buckets: <number>,
output: {
<output1>: { <$accumulator expression> },
...
}
granularity: <string>
}
}
```

CRUD

- Grouping information → $bucketAuto

```
// grouping by year and getting a count per year using the { $sum: 1 } pattern
db.movies.aggregate([
    {   "$bucketAuto":
            {       groupBy: "$year",
                    buckets: 5,
                    output: {   "numFilmsThisYear": { "$sum": 1 }   }
            }
    }])
```

CRUD

- Accumulator expressions → $project

Accumulator Expressions in $project operate over an array in the current document, they do not carry values over all documents!

```
db.example.find()
```

```
{ _id: 0, data: [ 1, 2, 3, 4, 5  ] }
{ _id: 1, data: [ 1, 3, 5, 7, 9  ] }
{ _id: 2, data: [ 2, 4, 6, 8, 10 ] }
```

```
db.example.aggregate([{
  $project: { dataAverage: { $avg: "$data" } }
}])
```

```
{ _id: 0, dataAverage: 3 }
{ _id: 1, dataAverage: 5 }
{ _id: 2, dataAverage: 6 }
```

CRUD

- Accumulator expressions → $project

```
// using $reduce to get the highest temperature
db.icecream_data.aggregate([
    {   "$project": {"_id": 0,
            "max_high": {"$reduce": {
                "input": "$trends",
                "initialValue": -Infinity,
                "in": {"$cond": [{
                    "$gt": ["$$this.avg_high_tmp", "$$value"] },
                    "$$this.avg_high_tmp",
                    "$$value"       ]       }       }     }   } }])
```

CRUD

- Accumulator expressions → $project

```
// note that these two operations can be done with the following operations can
// be done more simply. The following two expressions are functionally identical
db.icecream_data.aggregate([  { "$project": { "_id": 0,
                                              "max_high": { "$max": "$trends.avg_high_tmp" } } }])

db.icecream_data.aggregate([  { "$project": { "_id": 0,
                                              "min_low": { "$min": "$trends.avg_low_tmp" } } }])
```

CRUD

- Accumulator expressions → $project

```
// getting the average and standard deviations of the consumer price index
db.icecream_data.aggregate([  {    "$project": {      "_id": 0,
                        "average_cpi": { "$avg": "$trends.icecream_cpi" },
                        "cpi_deviation": { "$stdDevPop": "$trends.icecream_cpi" }    }  }])

// using the $sum expression to get total yearly sales
db.icecream_data.aggregate([  {    "$project": {      "_id": 0,
     "yearly_sales (millions)": { "$sum": "$trends.icecream_sales_in_millions" }    }  }])
```

CRUD

- Accumulator expressions → $project

  Exercise:

  In the last lab, we calculated a normalized rating that required us to know what the minimum and maximum values for imdb.votes were. These values were found using the $group stage!

  For all films that won at least 1 Oscar, calculate the standard deviation, highest, lowest, and average imdb.rating. Use the sample standard deviation expression.

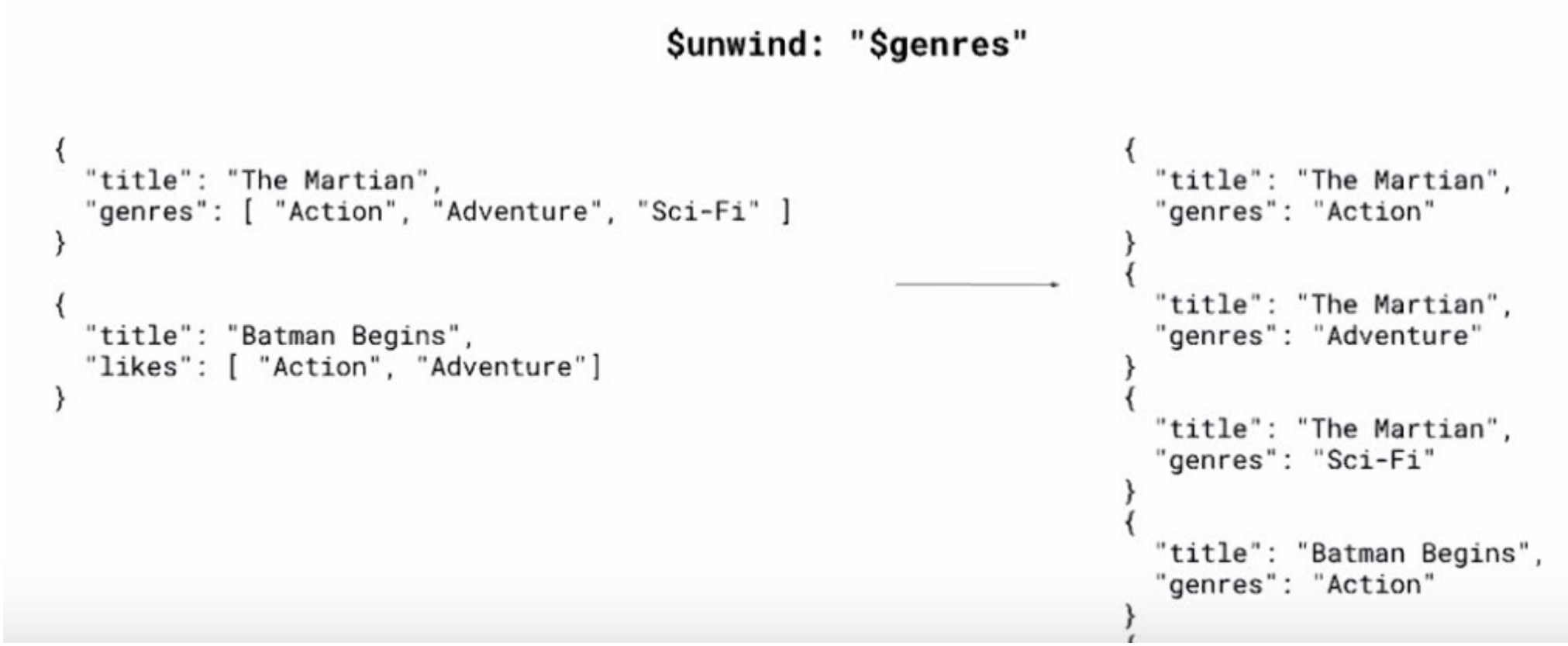
  HINT - All movies in the collection that won an Oscar begin with a string resembling one of the following in their awards field

  Won 13 Oscars

  Won 1 Oscar

CRUD

- Accumulator expressions → $unwind



CRUD

- Accumulator expressions → $unwind

```
$group: {
  _id: {
    title: "$title",
    genre: "$genres"
  }
}
```

```
{
  "title": "Star Trek",
  "genre": [
    "Adventure",
    "Action"
  ]
}
```

≠

```
{
  "title": "Star Trek",
  "genres": [
    "Action",
    "Adventure"
  ]
}
```

CRUD

- Accumulator expressions → $unwind

```
// finding the top rated genres per year from 2010 to 2015...
db.movies.aggregate([
    {"$match": { "imdb.rating": { "$gt": 0 }, "year": { "$gte": 2010, "$lte": 2015 },
"runtime": { "$gte": 90 }    } },
    {"$unwind": "$genres"  },
    {"$group": {"_id": {"year": "$year", "genre": "$genres"},
                "average_rating": { "$avg": "$imdb.rating" }    } },
    {"$sort": { "_id.year": -1, "average_rating": -1 }  }
])
```

CRUD

- Accumulator expressions → $unwind

```
// unfortunately we got too many results per year back. Rather than peform some
// other complex grouping and matching, we just append a simple group and sort
// stage, taking advantage of the fact the documents are in the order we wantç
db.movies.aggregate([{"$match": {"imdb.rating": { "$gt": 0 },"year": { "$gte": 2010, "$lte": 2015 },"runtime": { "$gte": 90 }}},{"$unwind": "$genres"}, {    "$group": {"_id": {"year": "$year","genre": "$genres"},"average_rating": { "$avg": "$imdb.rating" }}}, {"$sort": { "_id.year": -1, "average_rating": -1 }  },
{"$group": {"_id": "$_id.year",
        "genre": { "$first": "$_id.genre" },
        "average_rating": { "$first": "$average_rating" }}},  {"$sort": { "_id": -1 }  }])
```

CRUD

- Accumulator expressions → $unwind

Exercise:

Let's use our increasing knowledge of the Aggregation Framework to explore our movies collection in more detail. We'd like to calculate how many movies every cast member has been in and get an average imdb.rating for each cast member.

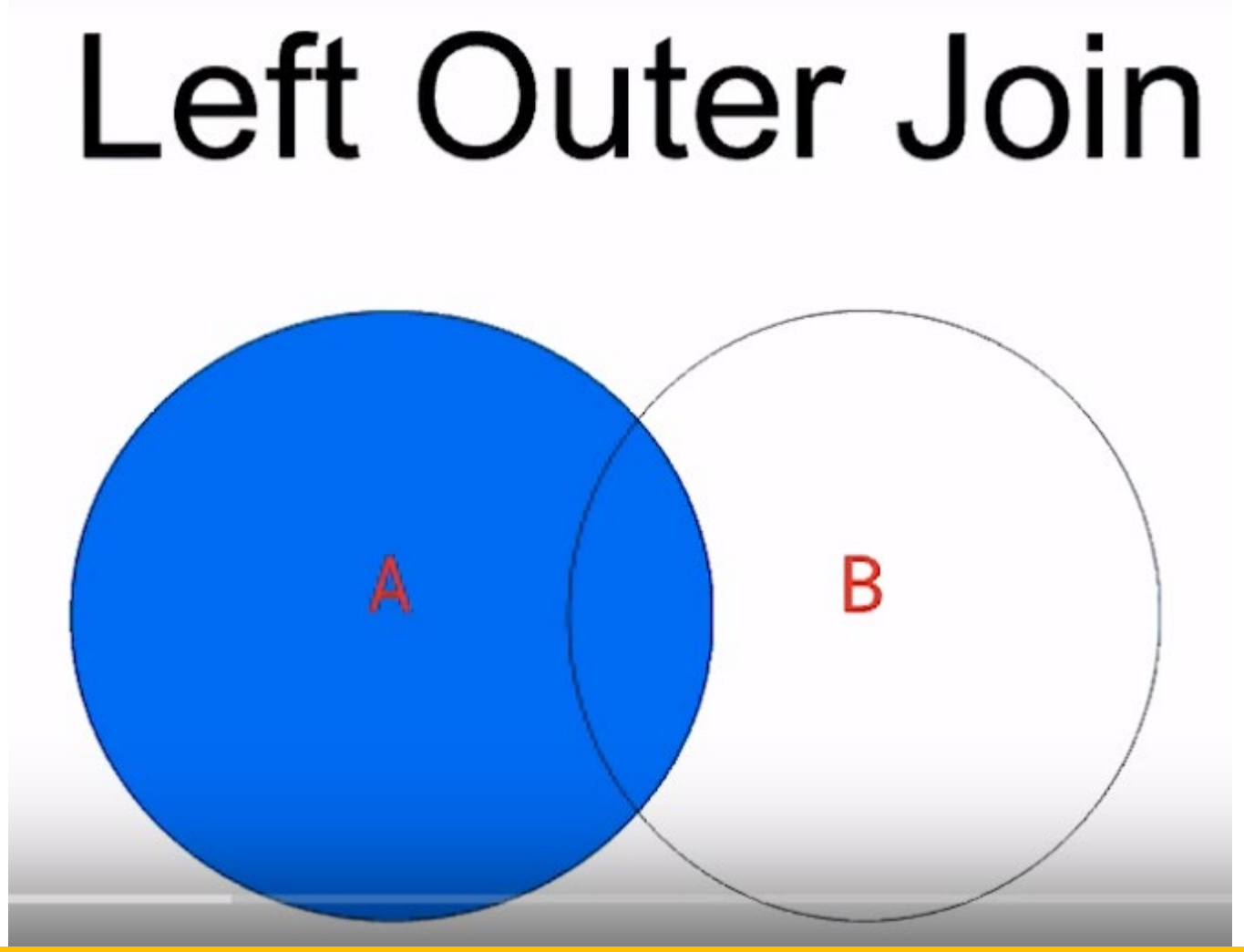
What is the name, number of movies, and average rating (truncated to one decimal) for the cast member that has been in the most number of movies with English as an available language?

Provide the input in the following order and format

```
{ "_id": "First Last", "numFilms": 1, "average": 1.1 }
```

CRUD

- Accumulator expressions → $lookup



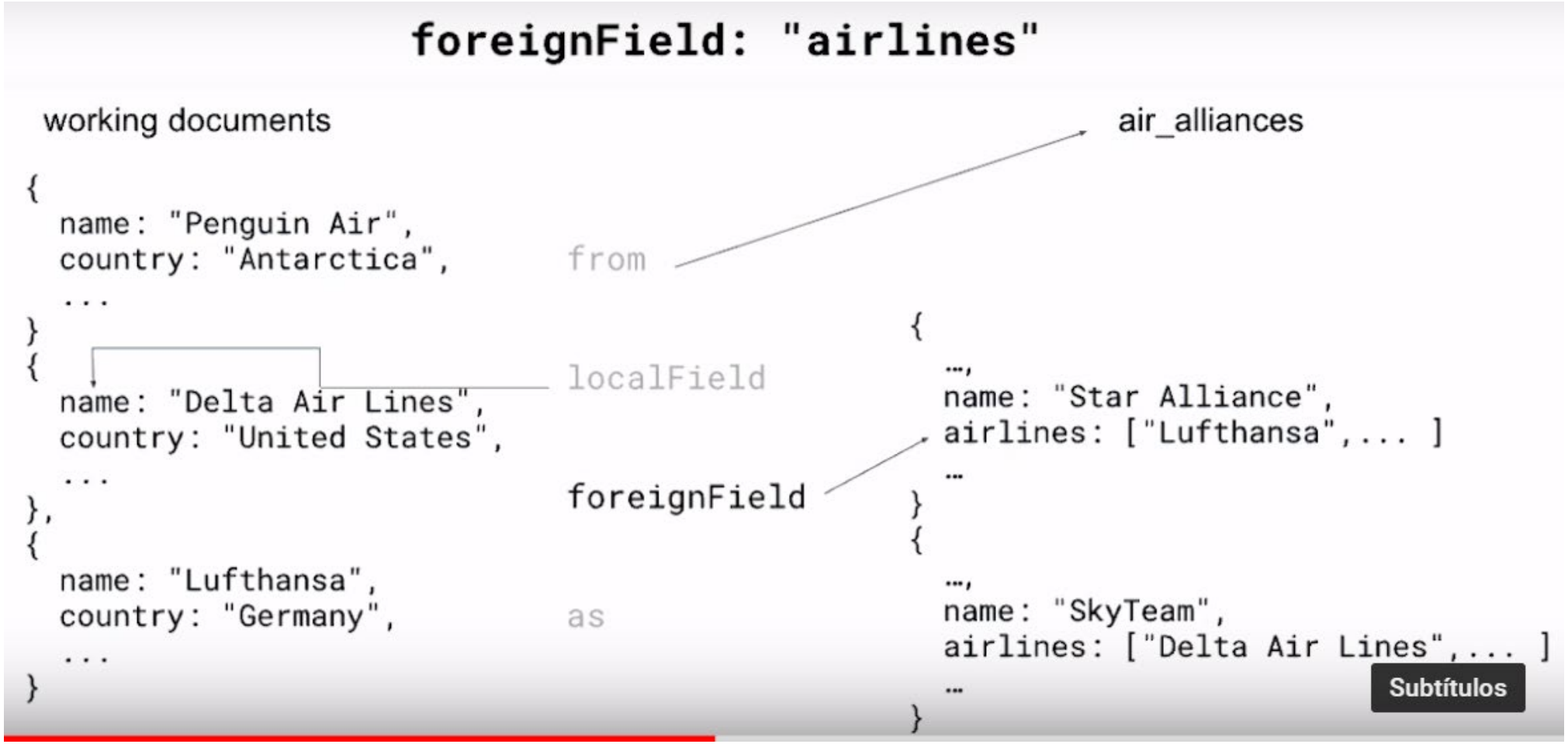
CRUD

- Accumulator expressions → $lookup

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

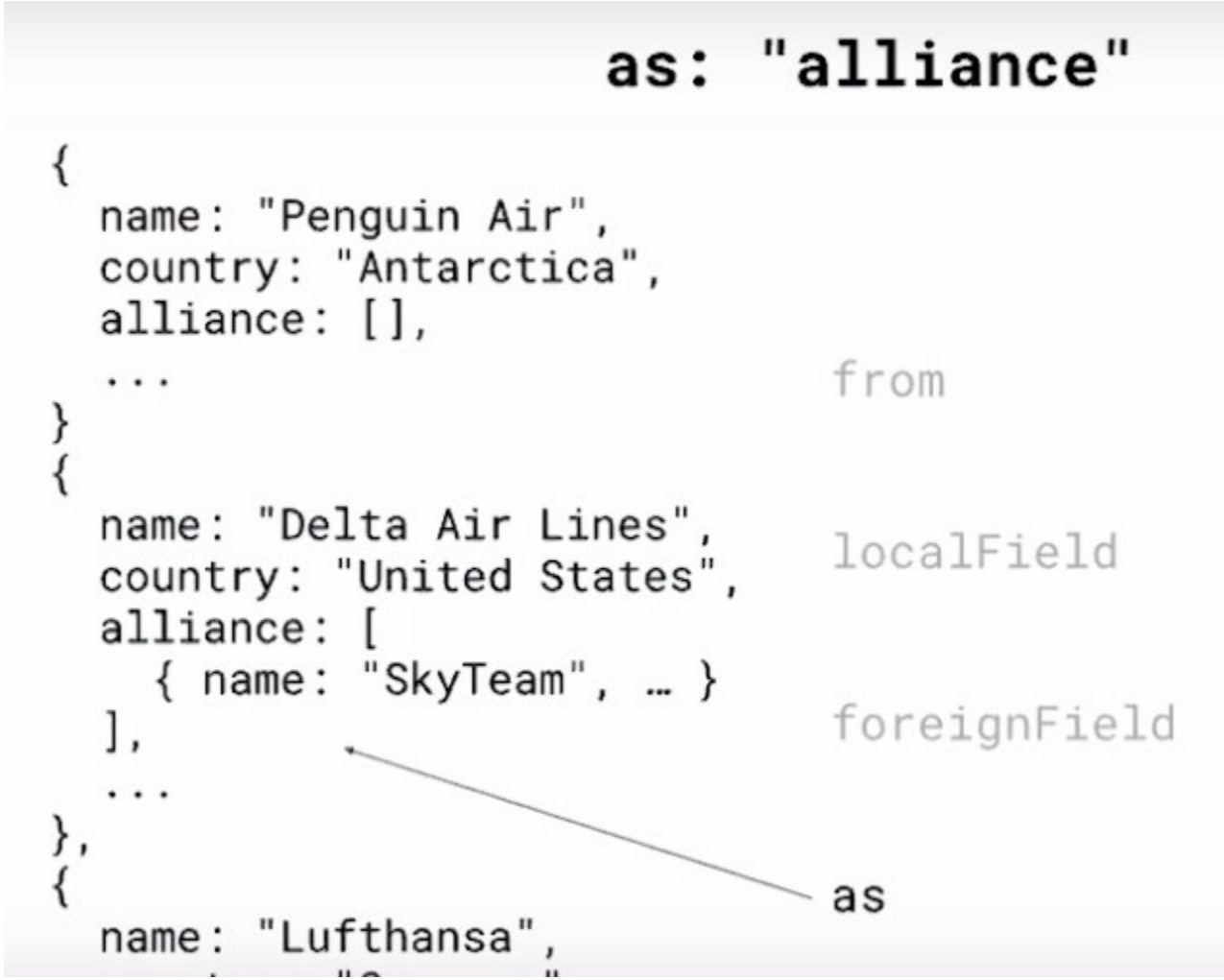
CRUD

- Accumulator expressions → $lookup



CRUD

- Accumulator expressions → $lookup



CRUD

- Accumulator expressions → $lookup

```
// familiarizing with the air_alliances schema
db.air_alliances.findOne()
// familiarizing with the air_airlines schema
db.air_airlines.findOne()
// performing a lookup, joining air_alliances with air_airlines and replacing the current airlines information with the new values
db.air_alliances  .aggregate([
        {"$lookup": {"from": "air_airlines", "localField": "airlines",
                "foreignField": "name", "as": "airlines"}}]).pretty()
```

CRUD

- Accumulator expressions → $lookup

Exercise:

Which alliance from air_alliances flies the most routes with either a Boeing 747 or an Airbus A380 (abbreviated 747 and 380 in air_routes)?