MongoDB Lab

MongoDB

Connecting to MongoDB Server

Lab

Modify Docker Build

- We will be using some Javascript files to bootstrap and create MongoDB collections and documents
- These files must be accessible from within the MongoDB server docker container
- The easiest way is to place all the Javascript files inside the mongo directory and modify our build so that all files in the mongo directory are copied into the container environment

- Challenge:
  - What changes do you need?

Lab

Modify Dockerfile

- Step 1
  - Copy all the Javascript files provided by the instructor to the ./mongo directory
- Step 2
  - Modify the Dockerfile under ./mongo
  - This file instructs docker-composite, how to build the the mongo container
  - Add the following:
- Step 3
  - Run docker-compose down
  - Run docker-compose up --build

```
FROM mongo

RUN apt-get update \
    && apt-get install -y mongodb-org-shell \
    && rm -rf /var/lib/apt/lists/*


# Copy local code to the container image.
COPY . .
```

Lab

Connect to MongoDB Container From Terminal

- Use `docker ps` to verify that the container is up and to get the container id

```
hwpark@wiken2 myapp % docker ps
CONTAINER ID   IMAGE         COMMAND                  CREATED         STATUS         PORTS
NAMES
029ca2f407eb   myapp-app     "docker-entrypoint.s…"   9 seconds ago   Up 8 seconds   0.0.0.0:3000->3000/tcp
myapp-app-1
b8c638e25355   myapp-mongo   "docker-entrypoint.s…"   9 seconds ago   Up 8 seconds   0.0.0.0:27017->27017/tcp
myapp-mongo-1
7a4a8ecd1cdf   redis         "docker-entrypoint.s…"   9 seconds ago   Up 8 seconds   0.0.0.0:6379->6379/tcp
myapp-redis-1
```

- Use the container id with command below to open bash shell

```
docker exec –it <container_id> bash
```

```
hwpark@wiken2 myapp % docker exec -it b8c638e25355 bash
root@b8c638e25355:/#
```

- Hint: It is often easier to use a container name. In our case:

```
docker exec –it myapp-mongo-1 bash
```

Lab

Login to MongoDB Server

- Use mongosh to login to the MongoDB Server
  - We will have to provide credentials
    - user = admin
    - password = password
  - Name of the host and port
    - host = localhost
    - port = 27017
  - Authenticating Database
    - Name of DB used to authenticate = admin

```
mongosh --host localhost --port 27017 --authenticationDatabase
admin -u admin -p password
```

- Using defaults since we are on the Server container

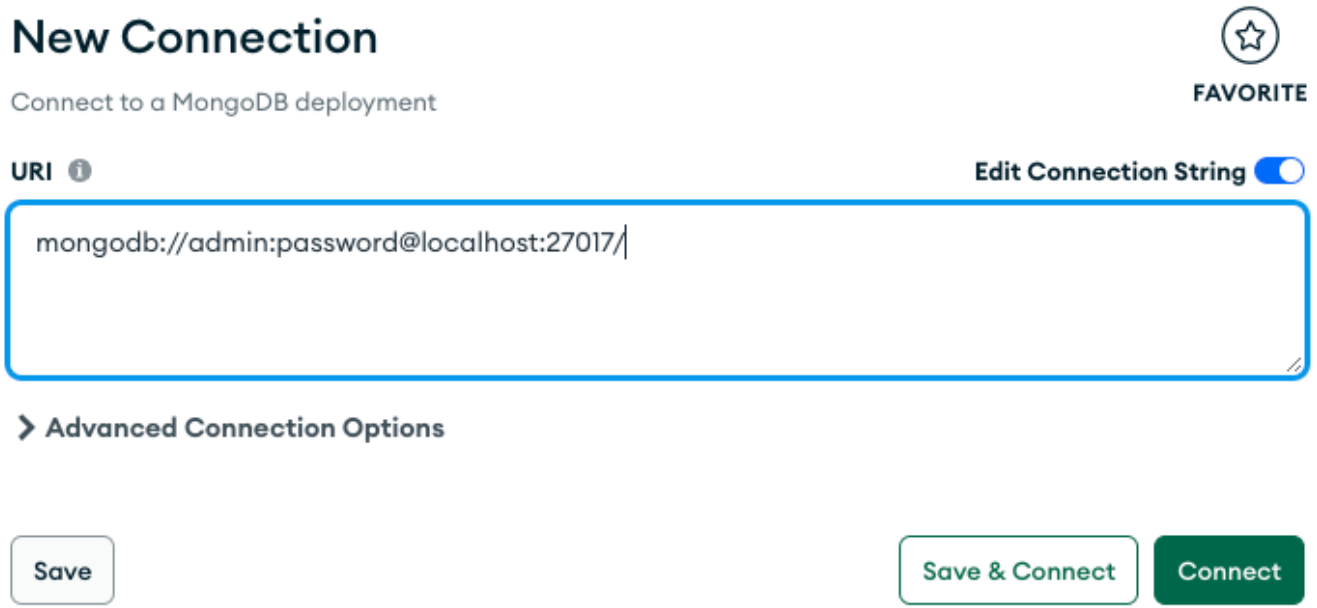
```
mongosh -u admin -p password
```

Verify Connection to MongoDB Server

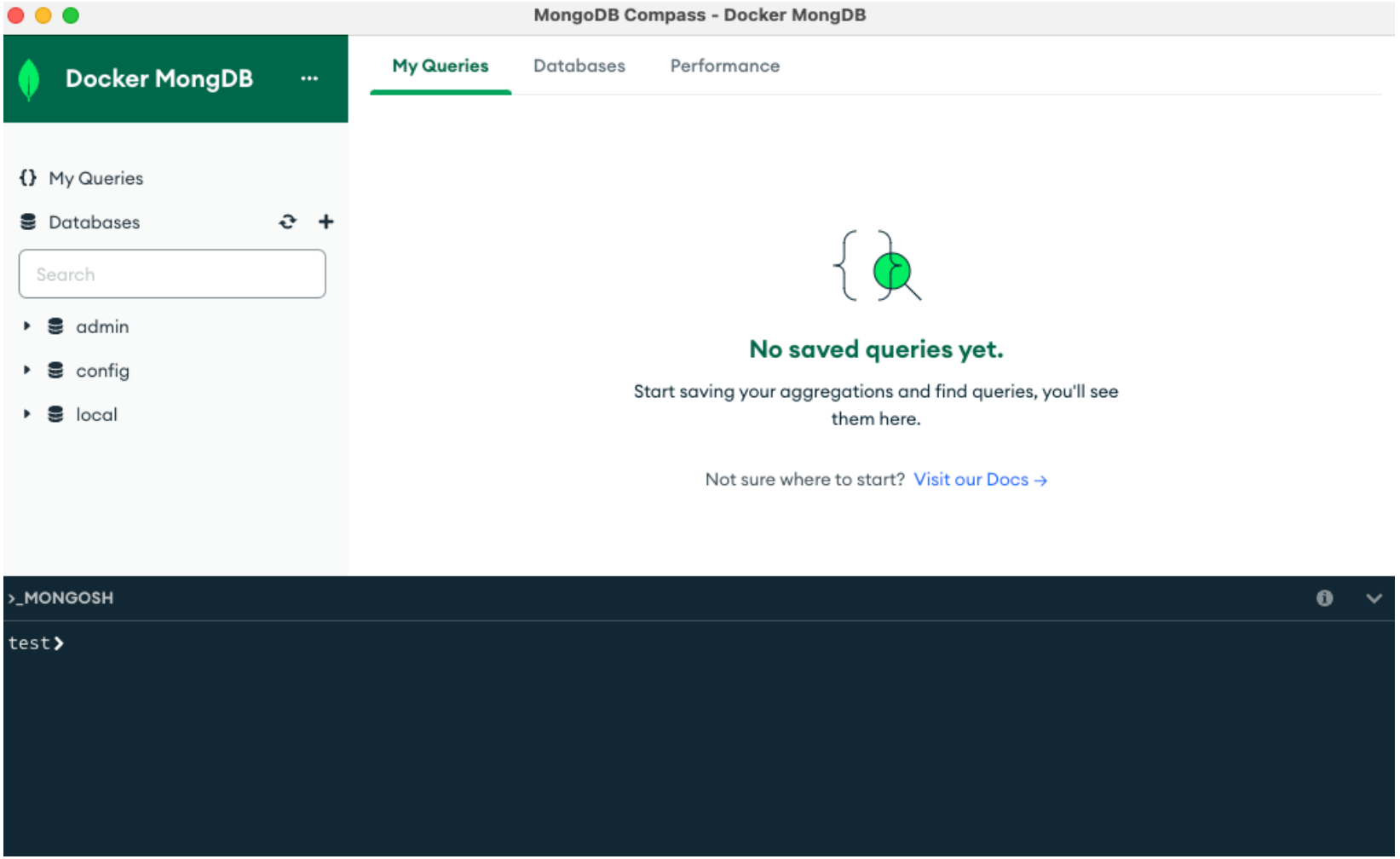
```
root@b8c638e25355:/# mongosh -u admin -p password
Current Mongosh Log ID: 648dd4831207b2f168fa906a
Connecting to:          mongodb://<credentials>@127.0.0.1:27017/?directConnection=tru
e&serverSelectionTimeoutMS=2000&appName=mongosh+1.9.1
Using MongoDB:          6.0.6
Using Mongosh:          1.9.1

For mongosh info see: https://docs.mongodb.com/mongodb-shell/

------
   The server generated these startup warnings when booting
   2023-06-17T10:53:42.604+00:00: Using the XFS filesystem is strongly recommended wi
th the WiredTiger storage engine. See http://dochub.mongodb.org/core/prodnotes-filesy
stem
   2023-06-17T10:53:43.504+00:00: vm.max_map_count is too low
------

test>
```

Connect to MongoDB Container From Compass

- Start MongoDB Compass and add New Connection
- Use the following connection string
  - mongdb://admin:password@localhost:27017/
- Save & Connect



Verify Connection to MongoDb



MongoDB

Introduction to CRUD
(Create, Read, Update, Delete)

MongoDB

Create

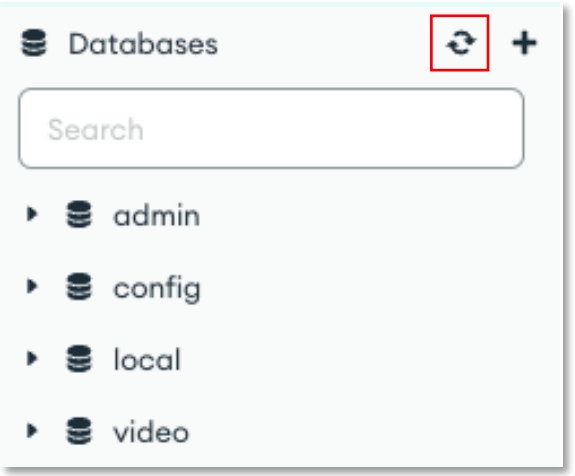
Create loadMovieDetail Dataset

- Inspect loadMovieDetailsDataset.js
  - db.getSiblingDB(<database>)
    - Allows changing to another database without establishing a new connection to Server
  - db.<collection>.drop
    - Drop a collection
  - db.<collection>.insertMany
    - Insert multiple documents into the collection
    - The documents are presented as {key:value} objects in a List
- Use load("myscript.js") to execute it
- Review Insert Methods
  - https://www.mongodb.com/docs/manual/reference/insert-methods/

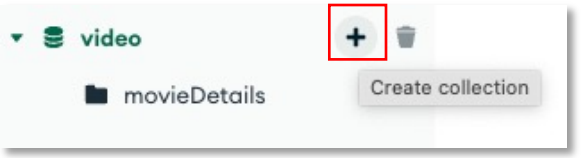
```
test> show dbs
admin   100.00 KiB
config  108.00 KiB
local    80.00 KiB
test> load("loadMovieDetailsDataset.js")
true
video> show dbs
admin   100.00 KiB
config  108.00 KiB
local    80.00 KiB
video     8.00 KiB
video> use video
already on db video
video> show collections
movieDetails
video>
```

Create Collection from Compass

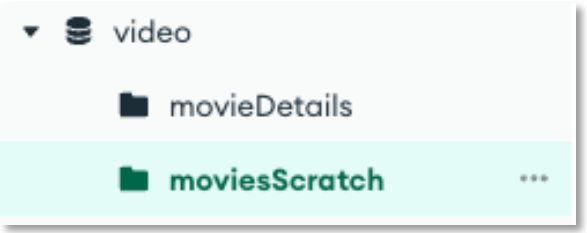
- Refresh Databases if you do not see the newly created video DB



- Select the video Database

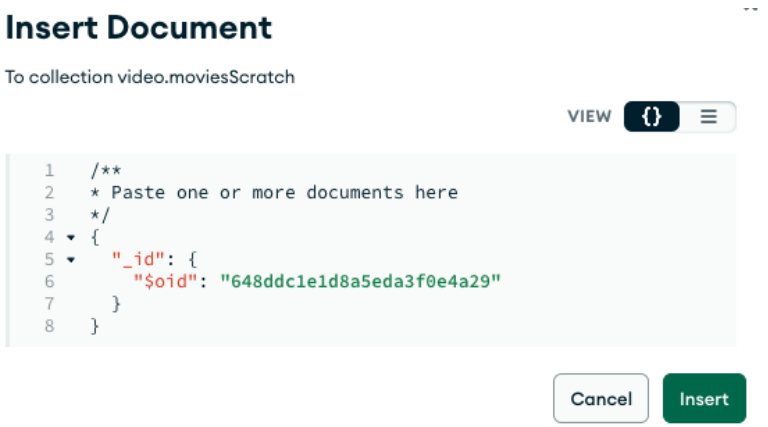


- Create a new collection
  - moviesScratch
  - Do not select time-series
  - Keep all defaults



Add Documents from Compass

- Insert a document by selecting ADD DATA
- Select Insert document Insert document
- Insert a new Document
  - Documents are in JSON format



- Add the following documents
  - {"title":"Rocky", "year":1976, "imdb":"tt0075148"}
  - {"title" :"Creed", "year" : 2015, "imdb":"tt3076658"}

Add Document from Shell

- Add a new document to the moviesScratch collection
  - use insertOne API

```
video> show collections
movieDetails
moviesScratch
video> db.moviesScratch.insertOne({"title":"Star Trek: The Wrath of Khan", "year":198
2, "imdb":"tt0084726"})
{
  acknowledged: true,
  insertedId: ObjectId("648de0471207b2f168fa9962")
}
```

  - Notice that in Compass, you must put the Keys in quotes but in the Shell, you don't have to

Shell

insertMany – Insert Multiple Documents

- There are two types of insertMany() operations
  - Ordered – Stops when an error is encountered
  - Ordered inserts are the default

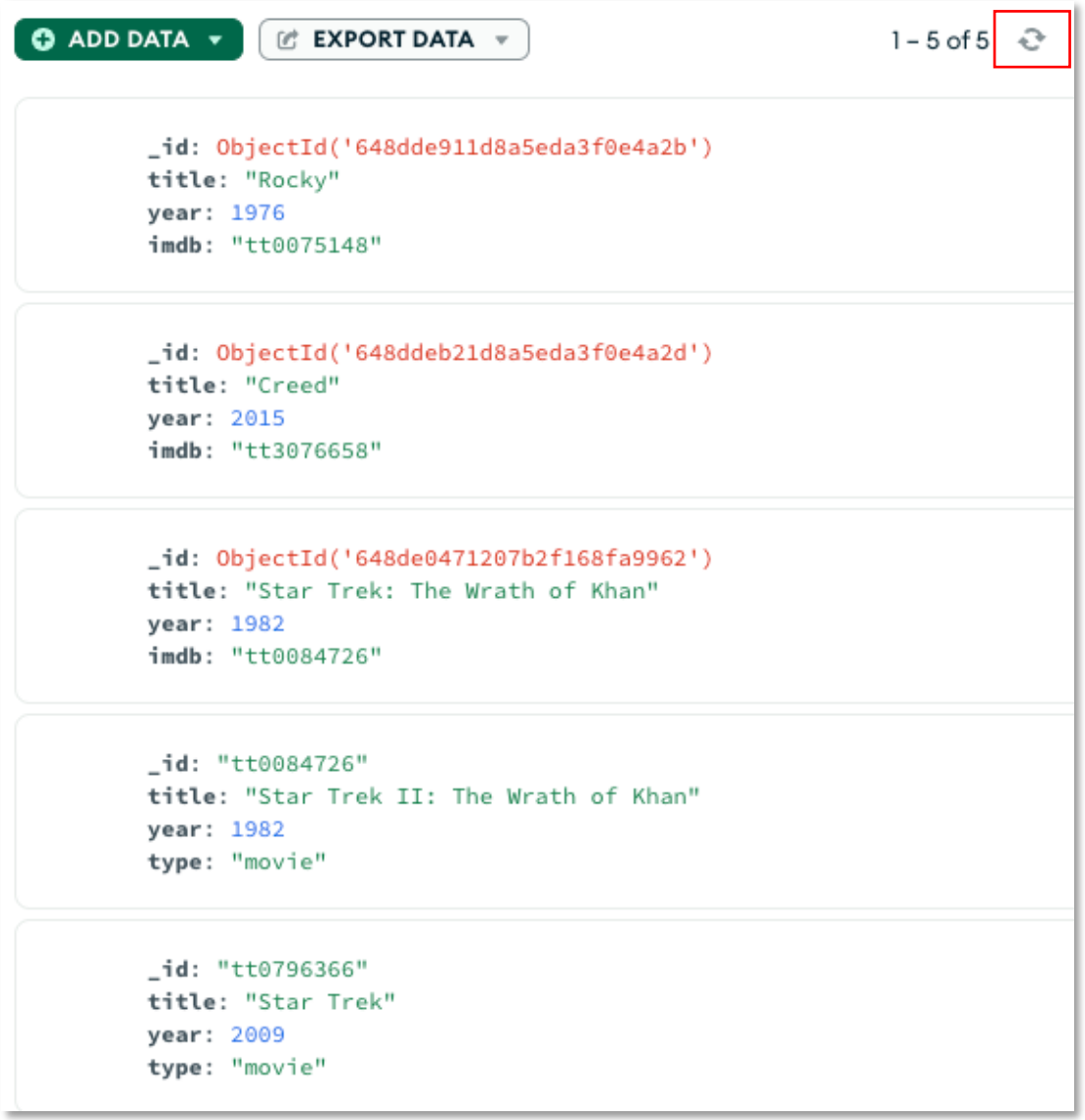
```
db.moviesScratch.insertMany(
    [
        {
      "_id" : "tt0084726",
      "title" : "Star Trek II: The Wrath of Khan",
      "year" : 1982,
      "type" : "movie"
        },
        {
      "_id" : "tt0796366",
      "title" : "Star Trek",
      "year" : 2009,
      "type" : "movie"
        },
        {
      "_id" : "tt0084726",
      "title" : "Star Trek II: The Wrath of Khan",
      "year" : 1982,
      "type" : "movie"
        },
        {
      "_id" : "tt1408101",
      "title" : "Star Trek Into Darkness",
      "year" : 2013,
      "type" : "movie"
        },
        {
      "_id" : "tt0117731",
      "title" : "Star Trek: First Contact",
      "year" : 1996,
      "type" : "movie"
        }
    ]
);
```

```
Uncaught:
MongoBulkWriteError: E11000 duplicate key er
ror collection: video.moviesScratch index: _
id_ dup key: { _id: "tt0084726" }
Result: BulkWriteResult {
  insertedCount: 2,
  matchedCount: 0,
  modifiedCount: 0,
  deletedCount: 0,
  upsertedCount: 0,
  upsertedIds: {},
  insertedIds: {
    '0': 'tt0084726',
    '1': 'tt0796366',
    '2': 'tt0084726',
    '3': 'tt1408101',
    '4': 'tt0117731'
  }
}
Write Errors: [
  WriteError {
    err: {
      index: 2,
      code: 11000,
      errmsg: 'E11000 duplicate key error co
llection: video.moviesScratch index: _id_ du
p key: { _id: "tt0084726" }',
      errInfo: undefined,
      op: {
        _id: 'tt0084726',
        title: 'Star Trek II: The Wrath of K
han',
        year: 1982,
        type: 'movie'
      }
```

Inspect Added Documents

- Refresh moviesScratch collection

- Notice that only 2 additional documents have been added
  - Notice that the documents we added from compass where we did not specify a _id has been assigned an ObjectId



insertMany – Insert Multiple Documents

- There are two types of insertMany() operations
  - Unordered – Keeps going with the rest even after an error
  - Add {"ordered": false }

```
db.moviesScratch.insertMany(
    [
        {
    "_id" : "tt0084726",
    "title" : "Star Trek II: The Wrath of Khan",
    "year" : 1982,
    "type" : "movie"
        },
        {
    "_id" : "tt0796366",
    "title" : "Star Trek",
    "year" : 2009,
    "type" : "movie"
        },
        {
    "_id" : "tt0084726",
    "title" : "Star Trek II: The Wrath of Khan",
    "year" : 1982,
    "type" : "movie"
        },
        {
    "_id" : "tt1408101",
    "title" : "Star Trek Into Darkness",
    "year" : 2013,
    "type" : "movie"
        },
        {
    "_id" : "tt0117731",
    "title" : "Star Trek: First Contact",
    "year" : 1996,
    "type" : "movie"
        }
    ],
    {
        "ordered": false
    }
);
```

```
Uncaught:
MongoBulkWriteError: E11000 duplicate key error coll
ection: video.moviesScratch index: _id_ dup key: { _
id: "tt0084726" }
Result: BulkWriteResult {
  insertedCount: 2,
  matchedCount: 0,
  modifiedCount: 0,
  deletedCount: 0,
  upsertedCount: 0,
  upsertedIds: {},
  insertedIds: {
    '0': 'tt0084726',
    '1': 'tt0796366',
    '2': 'tt0084726',
    '3': 'tt1408101',
    '4': 'tt0117731'
  }
}
Write Errors: [
  WriteError {
    err: {
      index: 0,
      code: 11000,
      errmsg: 'E11000 duplicate key error collection
: video.moviesScratch index: _id_ dup key: { _id: "t
t0084726" }',
      errInfo: undefined,
      op: {
        _id: 'tt0084726',
        title: 'Star Trek II: The Wrath of Khan',
        year: 1982,
        type: 'movie'
      }
    }
  },
  WriteError {
    err: {
      index: 1,
      code: 11000,
```

Inspect Added Documents

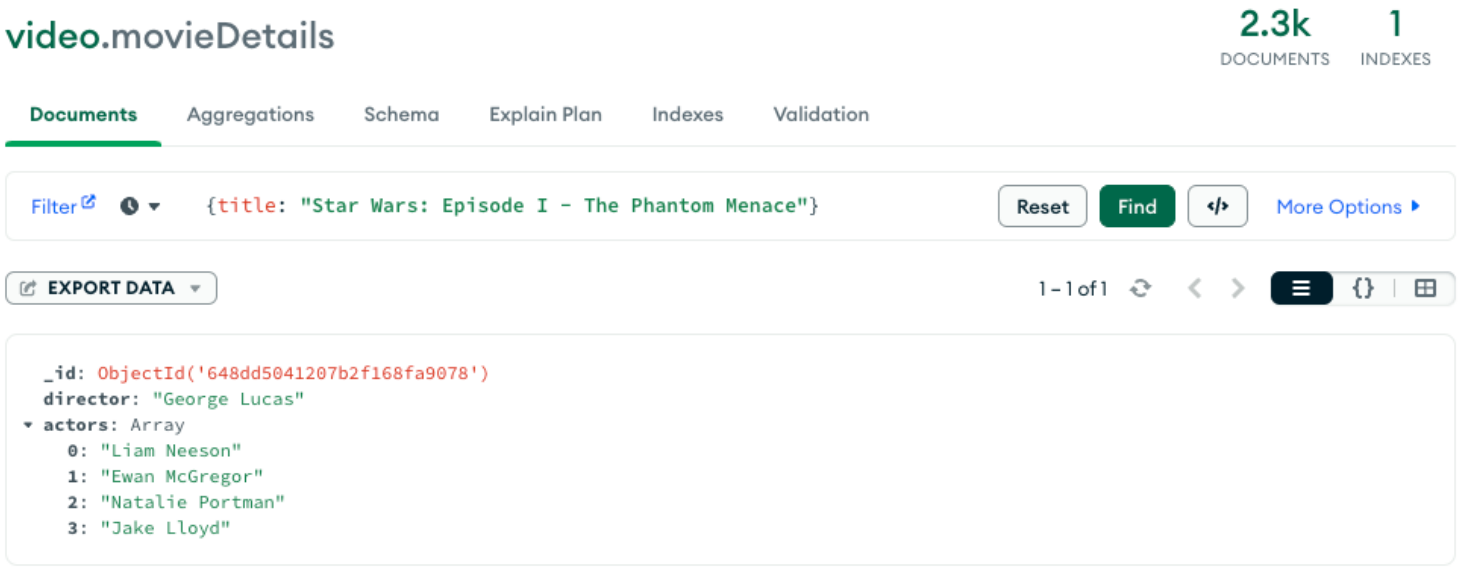
- Notice that even after an error, the insertMany() continued to add new documents
- Two additional documents have been added after initial error
  - Star Trek Into Darkness
  - Star Trek: First Contact

MongoDB

Read

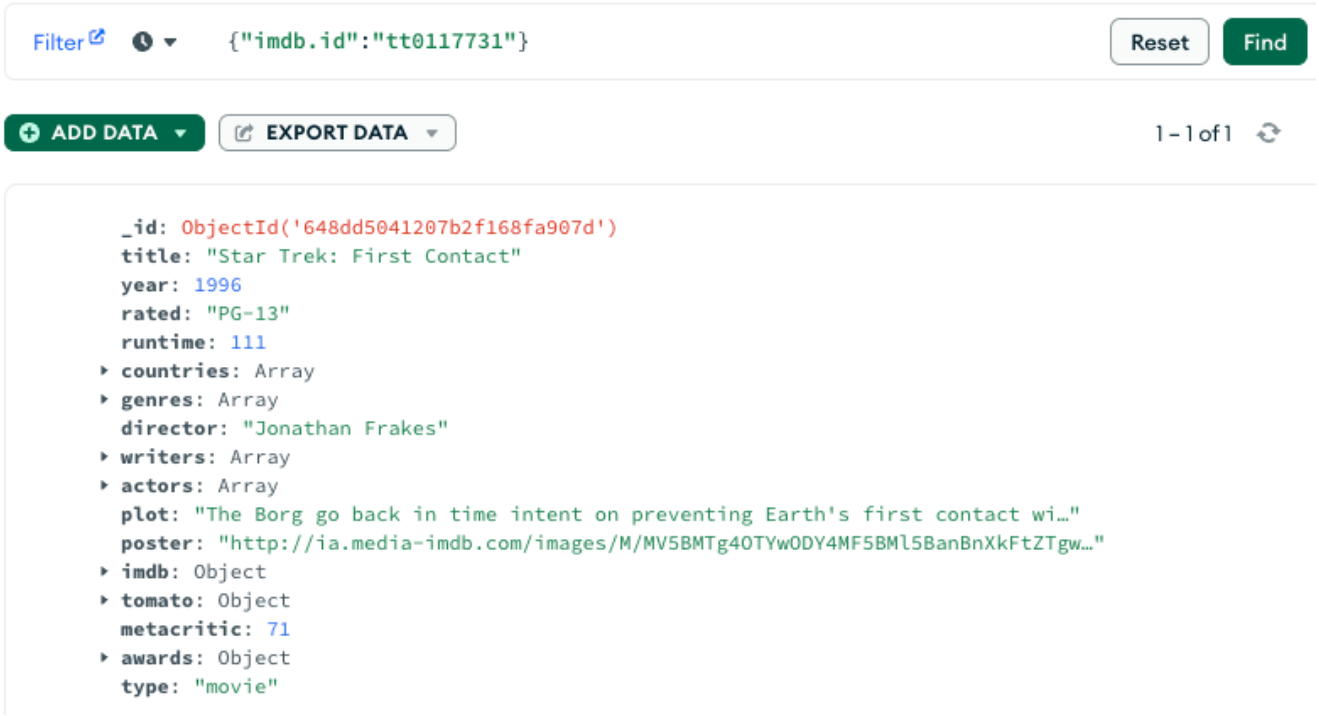
Basic Query Using find()

- Review Query Methods
  - https://www.mongodb.com/docs/manual/tutorial/query-documents/
- For the most simple queries, provide the key:value to search
  - Look for the movie by title
    - "Star Wars: Episode I – The Phantom Menace"
- From Compass, add a filter condition and click [Find]



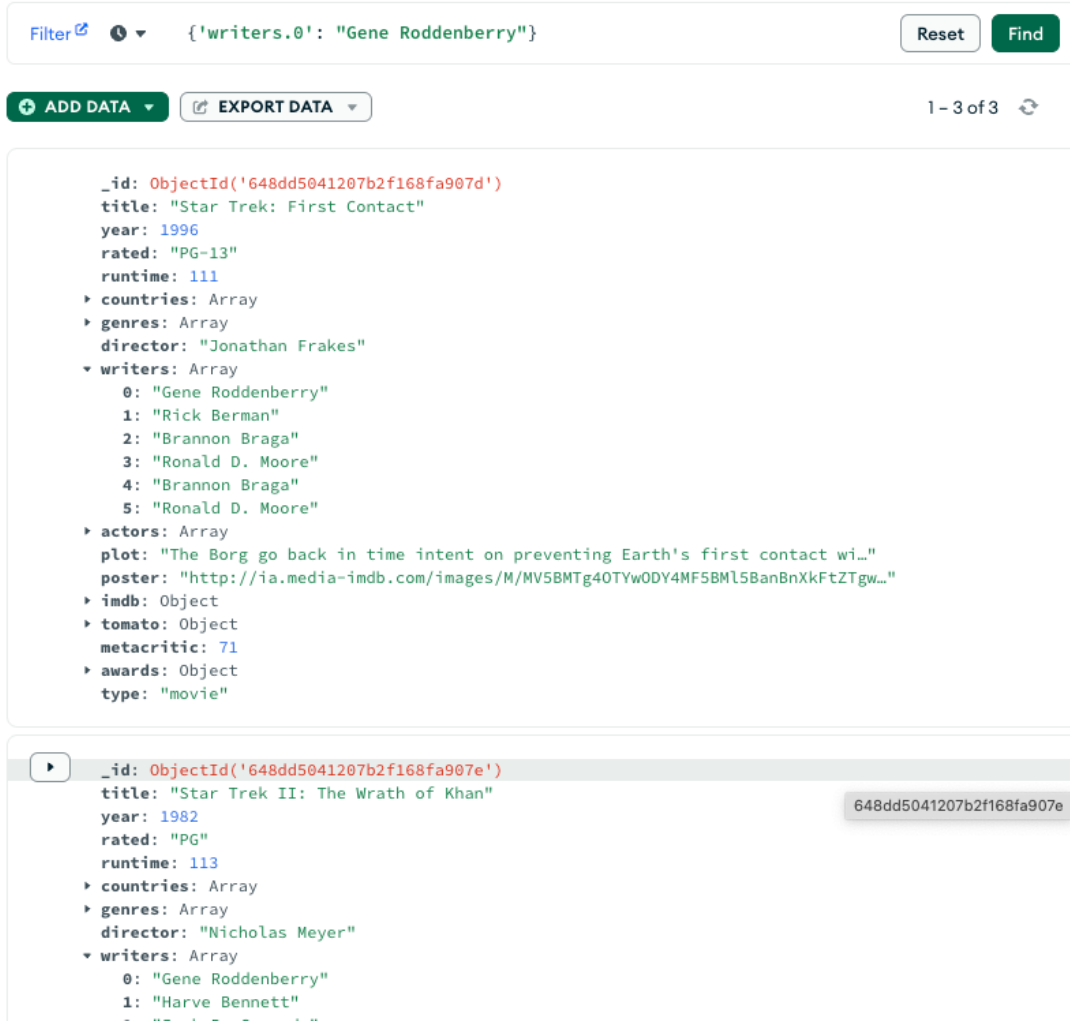
Query Nested Fields

- Documents can contain nested JSON fields
  - To access a nested field, use DOT notation
- IMDB – a movie information site uses imdb.id to track movies
  - Search for imdb.id = tt0117731



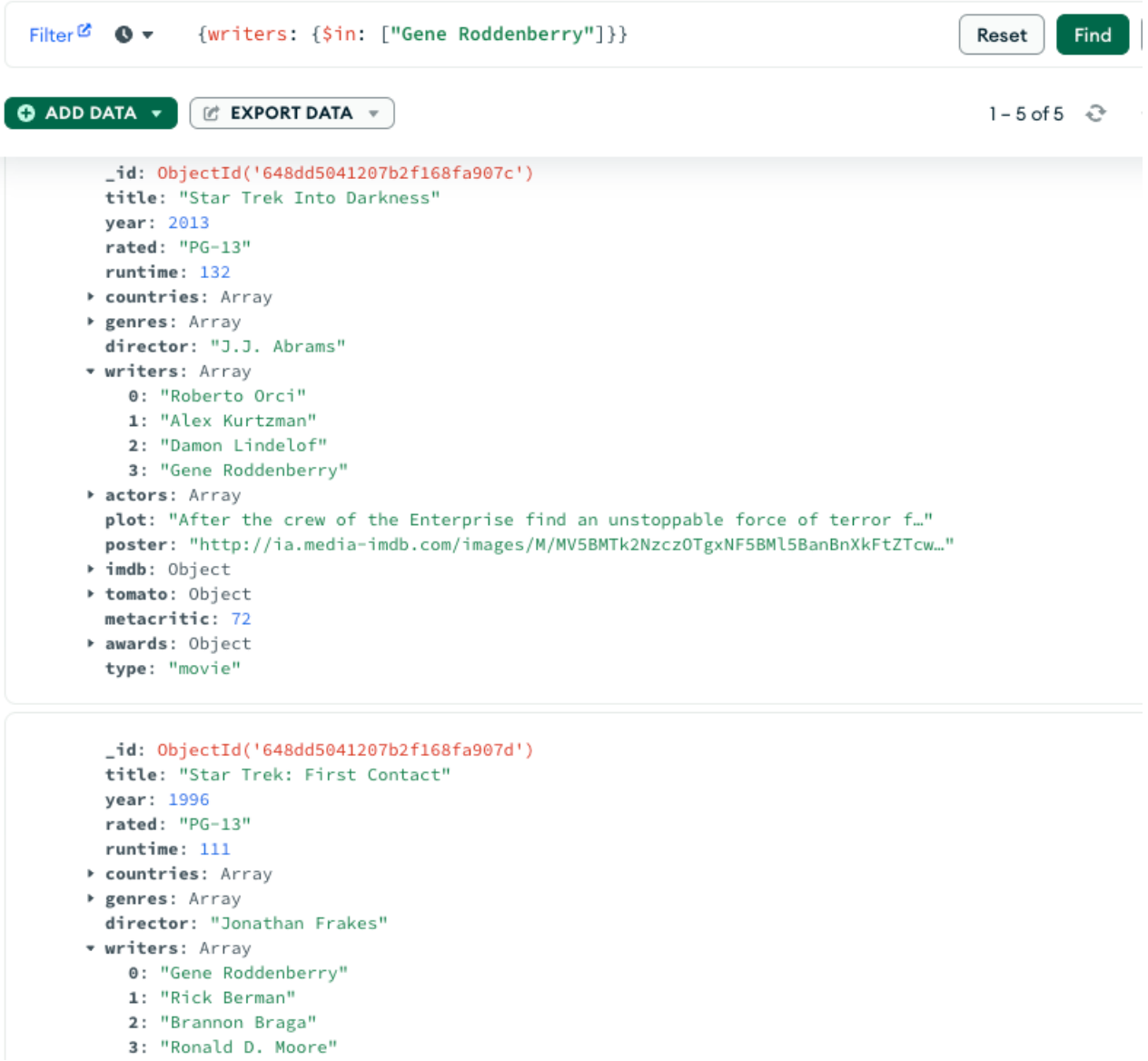
Query Array Element Fields

- Documents can contain Array fields
  - To access an element, use field[index] notation
  - Index starts at 0
  - In the writers field, the first writer is the main writer
  - Search for movies where "Gene Roddenberry" was the main writer



Query Membership of Array Fields

- In some of the movies, Gene Roddenberry was not main writer, but he was a member of the writing staff
- Use the $in operator to find this



Using MongoDB Shell for CRUD

- Using Compass is a good way to practice getting familiar with MongoDB's query syntax
- Easy to transfer the Filter for your query to Mongo Shell
  - Almost equally easy using mongoose or mongodb client
  - Recall the loadMovieDetailsDataset.js script
- Start using the "video" database
  - In the shell, we can achieve this with on of the two options below

```
use video;
```

```
db = db.getSiblingDB("video")
```

- Issue command with the following syntax:

```
db.<collection_name>.<operation_api>(parameters)
```

Lab

Try this Query On Your Own – Shell and Compass

- Explore the movieDetails collection and then issue a query to answer the following question.
  - How many movies in the movieDetails collection have exactly 2 award wins and 2 award nominations?
  - How about rated PG movies with 10 award nominations?
    - You will find the count() method useful in answering this question using the mongo shell
- Try using Compass and Mongo Shell to do this

Lab

Try this Query On Your Own – Shell and Compass

- Explore the movieDetails collection and then issue a query to answer the following question.
  - How many movies in the movieDetails collection have exactly 2 award wins and 2 award nominations?
  - How about rated PG movies with 10 award nominations?
    - You will find the count() method useful in answering this question using the mongo shell

```
use video
db.movieDetails.find({'awards.nominations': 2,'awards.wins': 2}).count()
db.movieDetails.find({rated: "PG", "awards.nominations": 10}).count()
```

Querying Array Fields – Details and Shortcuts

- Try the following query from movieDetails collection

```
use video
db.movieDetails.find({actors: ["Jeff Bridges", "Gwyneth Paltrow"]})
```

- Notice that this selects a document with an exact match
  - Even the order of the actors must match
  - Even the actual movie does not match because there are extra casts

```
db.movieDetails.find({title: "Iron Man"})
```

- What if I just want to choose a movie where Jeff Bridges is a cast – notice the shortcut (we didn't use $in)

```
db.movieDetails.find({actors: "Jeff Bridges"})
```

- Or perhaps in a movie where he is the star

```
db.movieDetails.find({'actors.0': "Jeff Bridges"})
```

Lab

Try this Query On Your Own – Shell and Compass

- Explore the movieDetails and then issue a query to answer the following question.
  - How many documents list just two writers: "Ethan Coen" and "Joel Coen", in that order?
  - How many movies are Family oriented movies?
  - How many movies are listed as Western as the second entry?
    - You will find the count() method useful in answering this question using the mongo shell.

Lab

Try this Query On Your Own – Shell and Compass

- Explore the movieDetails collection that you loaded into your Atlas sandbox cluster and then issue a query to answer the following question.
  - How many documents list just two writers: "Ethan Coen" and "Joel Coen", in that order?
  - How many movies are Family oriented movies?
  - How many movies are listed as Western as the second entry?
    - You will find the count() method useful in answering this question using the mongo shell.

```
use video
db.movieDetails.find({writers: ["Ethan Coen", "Joel Coen"]}).count()
```

```
db.movieDetails.find({genres: 'Family'}).count()
```

```
db.movieDetails.find({'genres.1': 'Western'}).count()
```

Introduction to Cursors

- The `find()` operator returns a cursor
  - In the shell, 20 documents are printed automatically
  - From the shell, we can type `it` to get the next 20
- From the shell, return all documents and use `it` to continue

```
db.movieDetails.find()
```

```
        "actors" : [
                "Peter Cushing",
                "Bernard Cribbins",
                "Ray Brooks",
                "Andrew Keir"
        ],
        "plot" : "The Daleks' fiendish plot in 2150 against Earth and its people is foiled when Dr. Who and friends arrive from the 20th century and
figure it out.",
        "poster" : "http://ia.media-imdb.com/images/M/MV5BMTg0OTc0MDc0M15BMl5BanBnXkFtZTgwMTg5ODYxMTE@._V1_SX300.jpg",
        "imdb" : {
                "id" : "tt0060278",
                "rating" : 6,
                "votes" : 2236
        },
        "awards" : {
                "wins" : 0,
                "nominations" : 0,
                "text" : ""
        },
        "type" : "movie"
}
Type "it" for more
MongoDB Enterprise >
```

Projections

- Use Projections to select only certain fields to be returned from a find() query
- Second argument to the find() operator
  - Use 1 if you want to explicitly select that field
  - Use 0 if you want to explicitly remove that field
- Try the following command from the shell

```
db.movieDetails.find(
  { genres:
    { $in: ["Action", "Adventure"]}
  },
  // projection to select only title field
  { title: 1}
)
```

Lab

Projections

- Notice that the primary field _id is included automatically
  - How might we explicitly exclude this field?

Lab

Projections

- Notice that the primary field _id is included automatically
  - How might we explicitly exclude this field?

```
db.movieDetails.find(
  { genres:
    { $in: ["Action", "Adventure"]}
  },
  // projection to select only title field
  // explicitly remove _id field
  { title: 1, _id: 0}
)
```

- Challenge: Include the genres so we know why the movie was included in the find() filter

MongoDB

Update

Dynamic and Flexible Schema

- One of the core advantages of MongoDB is that the schema is dynamic and flexible
  - Not all documents are required to have the same fields
  - This means that we do not need to have fields with NULL values
- Let's look at movieDetails – notice that many of the documents are missing fields such as awards and posters
- From movieDetails, locate the movie title "The Martian"

```
use video
db.movieDetails.find({title: "The Martian"})
```

- Notice that this document does not have a link to a poster picture

Updating a document with updateOne()

- We will use the updateOne() operator to add a poster field to the movie – The Martian
- The update operator allows us to specify how fields should be modified in matching documents
- The first parameter specifies the filter
- The second parameter specifies the type of update
- Here we will use the $set operator
  - Think of it as write or replace
  - If the field does not exist, if will write
  - If it exists, $set will overwrite with the new data

```
db.movieDetails.updateOne(
  {title: "The Martian"},
  {$set:
    {poster: "https://www.imdb.com/title/tt3659388/mediaviewer/rm1391324160?ref_=tt_ov_i"}
  }
)
```

MongoDB Update Operators - $inc

- Please review the update operators
  - https://docs.mongodb.com/manual/reference/operator/update/
- Make note of current value for tomato.reviews and tomato.userReviews
- Let's try one - $inc
  - While we can also use $set to just change, it may be prone to errors

```
db.movieDetails.updateOne(
  {title: "The Martian"},
  {$inc:
    {
      "tomato.reviews": 3,
      "tomato.userReviews": 25
    }
  }
)
```

MongoDB Update Operators - $push

- Let's try another one that works on Arrays
  - The join() joins all the elements into a single string

```
let reviewText1 = [
    "The Martian could have been a sad drama film, instead it was a",
    "hilarious film with a little bit of drama added to it.  The Martian is what ",
    "everybody wants from a space adventure.  Ridley Scott can still make great ",
    "movies and this is one of the best"
].join()

db.movieDetails.updateOne(
  {title: "The Martian"},
  {$push:
    {reviews:
      {
        rating: 4.5,
        date: ISODate("2016-01-12T09:00:00Z"),
        reviewer: "Spencer H.",
        text: reviewText1
      }
    }
  }
)
```

Lab

Look for the Review Just Added

- Challenge:

- Look for the Review we just added
  - We can search my title
  - Try searching for the review name
  - Add a projection so that we just view the text of the review

Lab

Look for the Review Just Added

- Challenge:

- Look for the Review we just added
  - We can search my title
  - Try searching for the review name
  - Add a projection so that we just view the text of the review

```
db.movieDetails.find({"reviews.reviewer": "Spencer H."});

db.movieDetails.find(
  {"reviews.reviewer": "Spencer H."},
  {"reviews.text": 1, _id: 0}
);
```

Lab

Updating Multiple Documents with updateMany()

- With updateMany(), you can update multiple documents that match your filter criteria
- Observe that many of the documents contain a null rated field
- Let's remove them

```
db.movieDetails.updateMany(
  // this is the filter
  {rated: null},
  // this is the update
  {$unset:
    {rated: ""}
  }
)
```

- Challenge: Remove all empty poster fields

Lab

Updating Multiple Documents with updateMany()

- You try: Remove all empty poster fields

```
db.movieDetails.updateMany(
  // this is the filter
  {poster: null},
 // this is the update
  {$unset:
    {poster: ""}
  }
)
```

Use findOne() to copy to Memory

- Use findOne() to get the document into memory
  - find() returns a cursor, findOne() returns the first matching document
  - Update the document in memory
- Try "finding" the document again
  - Notice that nothing has been updated in the actual document

```
detailDoc = db.movieDetails.findOne({"imdb.id": "tt4368814"});
detailDoc.poster;
detailDoc.poster =
"https://www.imdb.com/title/tt4368814/mediaviewer/rm2926634240";
detailDoc.genres;
detailDoc.genres.push("Documentary");

db.movieDetails.findOne({"imdb.id": "tt4368814"});
```

Replace Document with replaceOne()

- Use replaceOne() to substitute with my in-memory document

```
db.movieDetails.replaceOne(
  {"imdb.id": detailDoc.imdb.id},
  detailDoc
);

db.movieDetails.findOne({"imdb.id": "tt4368814"});
```

- Notice that it is updated now

```
video> db.movieDetails.findOne({"imdb.id": "tt4368814"});
{
  _id: ObjectId("648dd5041207b2f168fa98bc"),
  title: 'Louis C.K.: Live at the Comedy Store',
  year: 2015,
  rated: null,
  runtime: 66,
  countries: [ 'USA' ],
  genres: [ 'Comedy', 'Documentary' ],
  director: 'Louis C.K.',
  writers: [ 'Louis C.K.', 'Jay London' ],
  actors: [ 'Louis C.K.', 'Jay London' ],
  plot: 'Comedian Louis C.K. performs live at the Comedy store in LA.',
  poster: 'https://www.imdb.com/title/tt4368814/mediaviewer/rm2926634240',
  imdb: { id: 'tt4368814', rating: 7.8, votes: 1402 },
  awards: {
    wins: 0,
    nominations: 1,
    text: 'Won 1 Primetime Emmy. Another 1 nomination.'
  },
  type: 'movie'
}
```

MongoDB

Delete

Lab

Load a Scratch Dataset

- Challenge:
  - Review `loadReviewsDataset.js`
  - This will create a review collection in the video database
  - `Verify that we have the new collection`

- `Review Delete Methods`
  - https://www.mongodb.com/docs/manual/reference/delete-methods/

Lab

Load a Scratch Dataset

- Challenge:
  - Review `loadReviewsDataset.js`
  - This will create a review collection in the video database
  - `Verify that we have the new collection`

```
video> load("loadReviewsDataset.js")
true
video> show collections;
movieDetails
moviesScratch
reviews
```

Lab

MongoDB deleteOne() operation

- The deleteOne() method allow us to delete a document
- The argument to deleteOne() is a filter
- Use the Compass filter or the find() method to review the reviews collection
  - Make note of one of the ObjectId's
  - Use the ObjectId as the filter

```
db.reviews.deleteOne({_id: ObjectId("<your_object_id>")})
```

```
video> db.reviews.deleteOne({_id: ObjectId("648ef8ea1207b2f168fa996f")})
{ acknowledged: true, deletedCount: 1 }
```

Lab

MongoDB deleteMany() operation

- Look for documents for reviews from
  - {reviewer_id: 759723314}
- First try the deleteOne() to delete all his/her reviews
- Use find() to see if we have deleted them
  - Notice that only 1 of the reviews have been deleted
  - The first one that is found is deleted
- This time, use deleteMany()
  - Check that all have been deleted this time

```
db.reviews.deleteOne({_id: ObjectId("5c754f064ad41406ab794dfa")})
db.reviews.deleteMany({reviewer_id: 759723314})
```

Lab

MongoDB deleteMany() operation

- Look for documents for reviews from
  - {reviewer_id: 759723314}
- First try the deleteOne() to delete all his/her reviews
- Use find() to see if we have deleted them
  - Notice that only 1 of the reviews have been deleted
  - The first one that is found is deleted
- This time, use deleteMany()
  - Check that all have been deleted this time

```
db.reviews.deleteOne({_id: ObjectId("5c754f064ad41406ab794dfa")})
db.reviews.deleteMany({reviewer_id: 759723314})
```

Lab

MongoDB deleteMany() operation

```
video> db.reviews.find({reviewer_id: 759723314}).count()
3
video> db.reviews.deleteOne({reviewer_id: 759723314})
{ acknowledged: true, deletedCount: 1 }
video> db.reviews.find({reviewer_id: 759723314}).count()
2
video> db.reviews.deleteMany({reviewer_id: 759723314})
{ acknowledged: true, deletedCount: 2 }
video> db.reviews.find({reviewer_id: 759723314}).count()
0
```

MongoDB

Query Operators

Lab

Query Operators

- All of the C R U D operations use a filter as the first parameter
- Query operators allow us to build complex queries that can be used for these filters
- Please review the many query operators available here:
  - https://docs.mongodb.com/manual/reference/operator/query/