B4M36DS2, BE4M36DS2: **Database Systems 2**

https://cw.fel.cvut.cz/b231/courses/b4m36ds2/


Practical Class 8

# MongoDB

**Yuliia Prokop**
prokoyul@fel.cvut.cz

13. 11. 2023


Authors: Martin Svoboda
(martin.svoboda@matfyz.cuni.cz),
Yuliia Prokop

**Czech Technical University in Prague**, Faculty of Electrical Engineering

# Data Model

Database system structure

Instance → **databases** → **collections** → **documents**

- Database
- Collection
  - Collection of documents, usually of a similar structure
- Document
  - MongoDB **document** = **one JSON object**
    - I.e. even a complex JSON object with other recursively nested objects, arrays or values
  - **Unique immutable identifier** `_id`
  - **Field name restrictions**: `_id`, `$`, `.`

# CRUD Operations

Overview

- **Create**
  - `db.collection.insertOne()`, `insertMany()`
  - Insert a new document (or documents) into a collection
- **Update**
  - `db.collection.replaceOne()`
  - `db.collection.updateOne()`, `updateMany()`
  - Modifiy an existing document / documents or insert a new one
- **Delete**
  - `db.collection.deleteOne()`, `deleteMany()`
  - Delete an existing document / documents
- **Read**
  - `db.collection.find()`

# Mongo Shell

**Connect to our NoSQL server**

- SSH / PuTTY and SFTP / WinSCP
- nosql.felk.cvut.cz

**Start mongo shell**

**mongosh** --port 42222 -u login -p password database

database=login, password – from the first email

**Try several basic commands**

- **help**
  - Displays a brief description of database commands
- **exit**
  quit()
  - Closes the current client connection

# Databases

**Switch to your database**

- **use** login
  db = db.getSiblingDB('login')
  - Use your login name as a name for your database

**List all the existing databases**

- **show databases**
  **show dbs**
  db.adminCommand('listDatabases')
  - Your database will be created later on implicitly

```
test> use f221_student
switched to db f221_student
f221_student> db.getSiblingDB('f221_student')
f221_student
```

# Collections

**Create a new collection** for actors

- db.**createCollection**("actors")
  - Suitable when creating collections with specific options  since collections can also be created implicitly

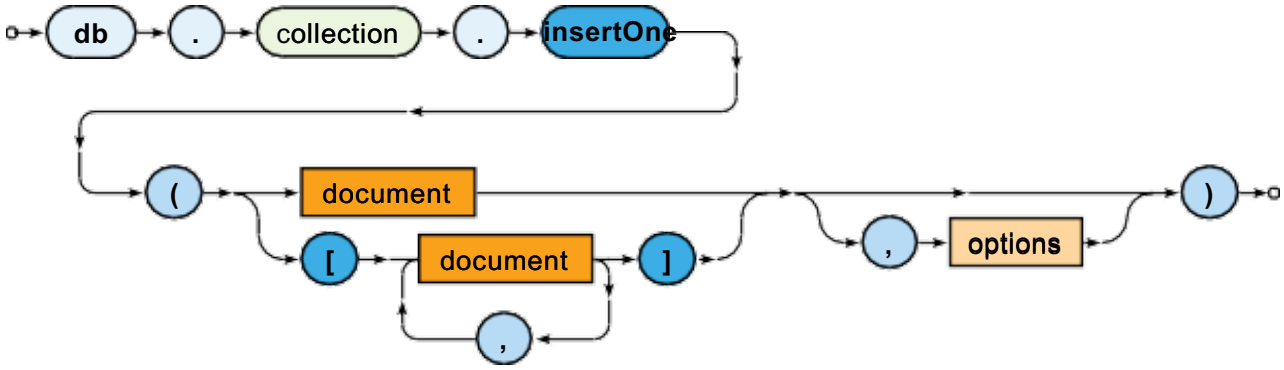
**List all collections** in your database

- **show collections**
  db.getCollectionNames()

```
f221_student> db.createCollection("actors")
{ ok: 1 }
f221_student> db.getCollectionNames()
[ 'actors' ]
```

# Insert Operation: insertOne(), insertMany()

**Inserts a new document** / documents into a given collection




- Parameters
  - **Document**: one or more documents to be inserted
  - **Options**

# Insert Operation

**Insert a few new documents** into the collection of actors

```
db.actors.insert({ _id: "trojan", name: "Ivan Trojan" })
```

```
f221_student> db.actors.insert({ _id: "trojan", name: "Ivan Trojan" })
DeprecationWarning: Collection.insert() is deprecated. Use insertOne, insertMany
, or bulkWrite.
{ acknowledged: true, insertedIds: { '0': 'trojan' } }
```

# Insert Operation

**Insert a few new documents** into the collection of actors

```
db.actors.insertOne({ _id: 2, name: "Jiri Machacek" })
```

```
db.actors.insertMany([{ _id: ObjectId(), name: "Jitka Schneiderova" },
{ name: "Zdenek Sverak" }])
```

```
f221_student> db.actors.insertOne({ _id: 2, name: "Jiri Machacek" })
{ acknowledged: true, insertedId: 2 }
f221_student> db.actors.insertMany([{ _id: ObjectId(), name: "Jitka
Schneiderova" }, { name: "Zdenek Sverak" }])
{
  acknowledged: true,
  insertedIds: {
    '0': ObjectId("637161c839d39ce6f6a0eac3"),
    '1': ObjectId("637161c839d39ce6f6a0eac4")
  }
}
```

# Insert Operation

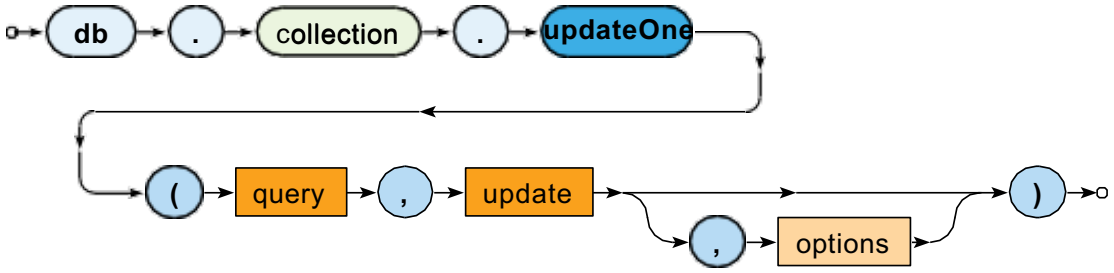
**Retrieve all documents** from the collection of actors

```
db.actors.find()
```

```
f221_student> db.actors.find()
[
  { _id: 'trojan', name: 'Ivan Trojan'},
  { _id: 2, name: 'Jiri Machacek' },
  {
    _id: ObjectId("637161c839d39ce6f6a0eac3"),
    name: 'Jitka Schneiderova'
  },
  { _id: ObjectId("637161c839d39ce6f6a0eac4"), name: 'Zdenek Sverak' }
]
```

# Update Operation: replaceOne, updateOne

**Modifies / replaces an existing document** / documents




- Parameters
  - **Query**: description of documents to be updated
  - **Update**: modification actions to be applied
  - **Options**

Update operators

- $set, $unset, $rename, $inc, $mul, $currentDate, $push, $addToSet, $pop, $pull, ...

# Update Operation

**Update the document** of actor *Ivan Trojan*

```
db.actors.replaceOne(
  { _id: "trojan" },
  { name: "Ivan Trojan", year: 1964 }
)
```

```
f221_student> db.actors.replaceOne( { _id: "trojan" }, { name: "Ivan Trojan", ye
ar: 1964 })
{
  acknowledged: true,
  insertedId: null,
  matchedCount: 1,
  modifiedCount: 1,
  upsertedCount: 0
}
```

```
db.actors.update(
  { name: "Ivan Trojan", year: { $lt: 2000 } },
  { name: "Ivan Trojan", year: 1964 }
)
```

# Update Operation

**Update the document** of actor *Ivan Trojan*

- At most one document is updated
- Its content is replaced with a new value

**Check the current content** of the document

```
db.actors.find({ _id: "trojan" })
```

```
f221_student> db.actors.find()
[
        { _id: 'trojan', name: 'Ivan Trojan', year: 1964 },
        {
                _id: ObjectId("6371503039d39ce6f6a0eac0"),
                name: 'Jitka Schneiderova'
        },
        { _id: 2, name: 'Jiri Machacek' },
        { _id: ObjectId("637161c839d39ce6f6a0eac4"), name: 'Zdenek Sverak' }
]
```

# Update Operation

Use **replaceOne** method to **insert a new actor**

- Inserts a new document when upsert behavior was enabled and no document could be updated

```
db.actors.replaceOne(
  { _id: "geislerova" },
  { name: "Anna Geislerova" },
  { upsert: true }
)
```

# Update Operation

**Try to modify the document identifier** of an existing document

- Your request will be rejected since **document identifiers are immutable**

```
db.actors.replaceOne(
  { _id: "trojan" },
  { _id: 1, name: "Ivan Trojan", year: 1964 }
)
```

```
f221_student> db.actors.replaceOne(
... { _id: "trojan" },
... { _id: 1, name: "Ivan Trojan", year: 1964 }
... )
MongoServerError: After applying the update, the (immutable) field '_id' was fou
nd to have been altered to _id: 1
```

# Update Operation

**Update the document** of actor *Ivan Trojan*

```
db.actors.updateOne(
  { _id: "trojan" },
  {
    $set: { year: 1964, age: 52 },
    $inc: { rating: 1 },
    $push: { movies: { $each: [ "samotari", "medvidek" ] } }
  }
)
```

```
f221_student> db.actors.find()
[
  {
    _id: 'trojan',
    name: 'Ivan Trojan',
    year: 1964,
    age: 52,
    movies: [ 'samotari', 'medvidek' ],
    rating: 1
  },
  {
    _id: ObjectId("6371503039d39ce6f6a0eac0"),
    name: 'Jitka Schneiderova'
  },
  { _id: 2, name: 'Jiri Machacek' },
  { _id: ObjectId("637161c839d39ce6f6a0eac4"), name: 'Zdenek Sverak' },
  { _id: 'geislerova', name: 'Anna Geislerova' }
]
```

# Update Operation

**Update multiple documents** at once

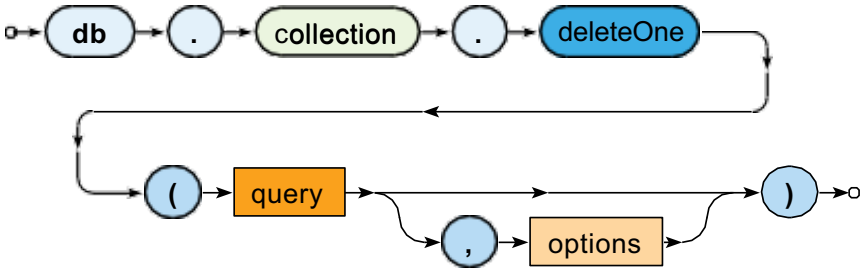
f221_student> db.actors.updateOne( { _id: 2 }, { $set: { year: 1980}, $inc: { rating: 1 } } )

```
db.actors.updateMany(
  { year: { $lt: 2000 } },
  { $set: { rating: 3 } }
)
```

```
f221_student> db.actors.find()
[
  {
    _id: 'trojan',
    name: 'Ivan Trojan',
    year: 1964,
    age: 52,
    movies: [ 'samotari', 'medvidek' ],
    rating: 3
  },
  {
    _id: ObjectId("6371503039d39ce6f6a0eac0"),
    name: 'Jitka Schneiderova'
  },
  { _id: 2, name: 'Jiri Machacek', rating: 3, year: 1980 },
  { _id: ObjectId("637161c839d39ce6f6a0eac4"), name: 'Zdenek Sverak' },
  { _id: 'geislerova', name: 'Anna Geislerova' }
]
```

# Remove Operation

**Removes** a document / documents from a given collection




- Parameters
  - **Query**: description of documents to be removed
  - **Options**

# Remove Operation

**Remove selected documents** from the collection of actors

f221_student> db.actors.insertMany([{year: 1990},{year:1980}])

```
db.actors.deleteOne({ _id: "geislerova" })
```

```
db.actors.deleteOne(
  { year: { $lt: 2000 } })
db.actors.deleteMany(
  { year: { $lt: 2000 } })
```

**Remove all the documents** from the collection of actors

```
db.actors.deleteMany({ })
```

# Sample Data

**Insert the following actors** into your emptied collection using different ways

```
{ _id: "trojan",
  name: "Ivan Trojan", year: 1964,
  movies: [ "samotari", "medvidek" ] }
```

```
{ _id: "machacek",
  name: "Jiri Machacek", year: 1966,
  movies: [ "medvidek", "vratnelahve", "samotari" ] }
```

```
{ _id: "schneiderova",
  name: "Jitka Schneiderova", year: 1973,
  movies: [ "samotari" ] }
```

```
{ _id: "sverak",
  name: "Zdenek Sverak", year: 1936,
  movies: [ "vratnelahve" ] }
```

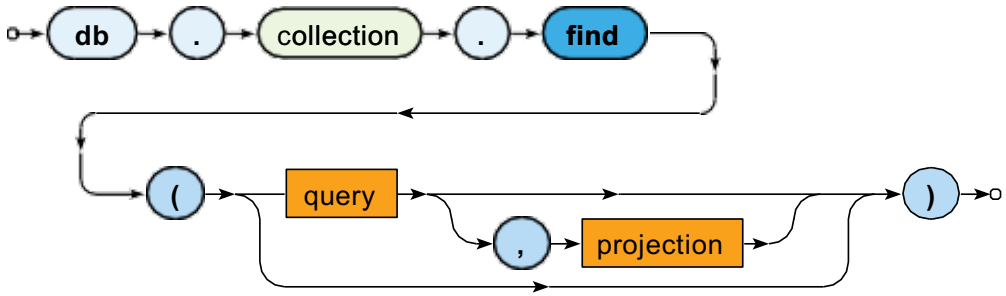
```
{ _id: "geislerova",
  name: "Anna Geislerova", year: 1976 }
```

# Sample Data

```
[
  {
    _id: 'trojan',
    name: 'Ivan Trojan',
    year: 1964,
    movies: [ 'samotari', 'medvidek' ]
  },
  {
    _id: 'machacek',
    name: 'Jiri Machacek',
    year: 1966,
    movies: [ 'medvidek', 'vratnelahve', 'samotari' ]
  },
  {
    _id: 'schneiderova',
    name: 'Jitka Schneiderova',
    year: 1973,
    movies: [ 'samotari' ]
  },
  {
    _id: 'sverak',
    name: 'Zdenek Sverak',
    year: 1936,
    movies: [ 'vratnelahve' ]
  },
  { _id: 'geislerova', name: 'Anna Geislerova', year: 1976 }
]
```

# Find Operation

**Selects** documents from a given collection




- Parameters
  - **Query**: description of documents to be selected
  - **Projection**: fields to be included / excluded in the result

# Querying

**Execute** and explain the meaning of **the following queries**

| |
|---|
| `db.actors.find()` |
| `db.actors.find({ })` |
| `db.actors.find({ _id: "trojan" })` |
| `db.actors.find({ name: "Ivan Trojan", year: 1964 })` |

```
[
  {
    _id: 'trojan',
    name: 'Ivan Trojan',
    year: 1964,
    movies: [ 'samotari', 'medvidek' ]
  }
]
```

# Querying

**Execute** and explain the meaning of **the following queries**

```
db.actors.find({ year: { $gte: 1960, $lte: 1980 } })
```

```
[
  {
    _id: 'trojan',
    name: 'Ivan Trojan',
    year: 1964,
    movies: [ 'samotari', 'medvidek' ]
  },
  {
    _id: 'machacek',
    name: 'Jiri Machacek',
    year: 1966,
    movies: [ 'medvidek', 'vratnelahve', 'samotari' ]
  },
  {
    _id: 'schneiderova',
    name: 'Jitka Schneiderova',
    year: 1973,
    movies: [ 'samotari' ]
  },
  { _id: 'geislerova', name: 'Anna Geislerova', year: 1976 }
]
```

# Querying

**Execute** and explain the meaning of **the following queries**

```
db.actors.find({ movies: { $exists: true } })
```

```
[
  {
    _id: 'trojan',
    name: 'Ivan Trojan',
    year: 1964,
    movies: [ 'samotari', 'medvidek' ]
  },
  {
    _id: 'machacek',
    name: 'Jiri Machacek',
    year: 1966,
    movies: [ 'medvidek', 'vratnelahve', 'samotari' ]
  },
  {
    _id: 'schneiderova',
    name: 'Jitka Schneiderova',
    year: 1973,
    movies: [ 'samotari' ]
  },
  {
    _id: 'sverak',
    name: 'Zdenek Sverak',
    year: 1936,
    movies: [ 'vratnelahve' ]
  }
]
```

# Querying

**Execute** and explain the meaning of **the following queries**

```
db.actors.find({ movies: "medvidek" })
```

```
db.actors.find({ movies: { $in: [ "medvidek", "pelisky" ] } })
```

```
[
  {
    _id: 'trojan',
    name: 'Ivan Trojan',
    year: 1964,
    movies: [ 'samotari', 'medvidek' ]
  },
  {
    _id: 'machacek',
    name: 'Jiri Machacek',
    year: 1966,
    movies: [ 'medvidek', 'vratnelahve', 'samotari' ]
  }
]
```

```
db.actors.find({ movies: { $all: [ "medvidek", "pelisky" ] } })
```

# Querying

**Execute** and explain the meaning of **the following queries**

```
db.actors.find({ $or: [ { year: 1964 }, { rating: { $gte: 3 } } ] })
```

```
[
  {
    _id: 'trojan',
    name: 'Ivan Trojan',
    year: 1964,
    movies: [ 'samotari', 'medvidek' ]
  }
]
```

# Querying

**Execute** and explain the meaning of **the following queries**

```
db.actors.find({ rating: { $not: { $gte: 3 } } })
```

```
[
  {
    _id: 'trojan',
    name: 'Ivan Trojan',
    year: 1964,
    movies: [ 'samotari', 'medvidek' ]
  },
  {
    _id: 'machacek',
    name: 'Jiri Machacek',
    year: 1966,
    movies: [ 'medvidek', 'vratnelahve', 'samotari' ]
  },
  {
    _id: 'schneiderova',
    name: 'Jitka Schneiderova',
    year: 1973,
    movies: [ 'samotari' ]
  },
  {
    _id: 'sverak',
    name: 'Zdenek Sverak',
    year: 1936,
    movies: [ 'vratnelahve' ]
  },
  { _id: 'geislerova', name: 'Anna Geislerova', year: 1976 }
]
```

# Querying

**Execute** and explain the meaning of **the following queries**

```
db.actors.find({ }, { name: 1, year: 1 })
```

```
db.actors.find({ }, { movies: 0, _id: 0 })
```

```
f221_student> db.actors.find({ }, { name: 1, year: 1 })
[
  { _id: 'trojan', name: 'Ivan Trojan', year: 1964 },
  { _id: 'machacek', name: 'Jiri Machacek', year: 1966 },
  { _id: 'schneiderova', name: 'Jitka Schneiderova', year: 1973 },
  { _id: 'sverak', name: 'Zdenek Sverak', year: 1936 },
  { _id: 'geislerova', name: 'Anna Geislerova', year: 1976 }
]
f221_student> db.actors.find({ }, { movies: 0, _id: 0 })
[
  { name: 'Ivan Trojan', year: 1964 },
  { name: 'Jiri Machacek', year: 1966 },
  { name: 'Jitka Schneiderova', year: 1973 },
  { name: 'Zdenek Sverak', year: 1936 },
  { name: 'Anna Geislerova', year: 1976 }
]
```

# Querying

**Execute** and explain the meaning of **the following queries**

```
db.actors.find({ }, { name: 1, movies: { $slice: 2 }, _id: 0 })
```

```
f221_student> db.actors.find({ }, { name: 1, movies: { $slice: 2 }, _id: 0 })
[
  { name: 'Ivan Trojan', movies: [ 'samotari', 'medvidek' ] },
  { name: 'Jiri Machacek', movies: [ 'medvidek', 'vratnelahve' ] },
  { name: 'Jitka Schneiderova', movies: [ 'samotari' ] },
  { name: 'Zdenek Sverak', movies: [ 'vratnelahve' ] },
  { name: 'Anna Geislerova' }
]
```

# Querying

**Execute** and explain the meaning of **the following queries**

```
db.actors.find().sort({ year: 1, name: -1 })
```

```
[
  {
    _id: 'sverak',
    name: 'Zdenek Sverak',
    year: 1936,
    movies: [ 'vratnelahve' ]
  },
  {
    _id: 'trojan',
    name: 'Ivan Trojan',
    year: 1964,
    movies: [ 'samotari', 'medvidek' ]
  },
  {
    _id: 'machacek',
    name: 'Jiri Machacek',
    year: 1966,
    movies: [ 'medvidek', 'vratnelahve', 'samotari' ]
  },
  {
    _id: 'schneiderova',
    name: 'Jitka Schneiderova',
    year: 1973,
    movies: [ 'samotari' ]
  },
  { _id: 'geislerova', name: 'Anna Geislerova', year: 1976 }
]
```

# Querying

**Execute** and explain the meaning of **the following queries**

```
db.actors.find().sort({ name: 1 }).skip(1).limit(2)
```

```
db.actors.find().sort({ name: 1 }).limit(2).skip(1)
```

```
[
  {
    _id: 'trojan',
    name: 'Ivan Trojan',
    year: 1964,
    movies: [ 'samotari', 'medvidek' ]
  },
  {
    _id: 'machacek',
    name: 'Jiri Machacek',
    year: 1966,
    movies: [ 'medvidek', 'vratnelahve', 'samotari' ]
  }
]
```

# Index Structures

Motivation

- Full **collection scan** must be conducted when searching for documents **unless an appropriate index exists**

**Primary index**

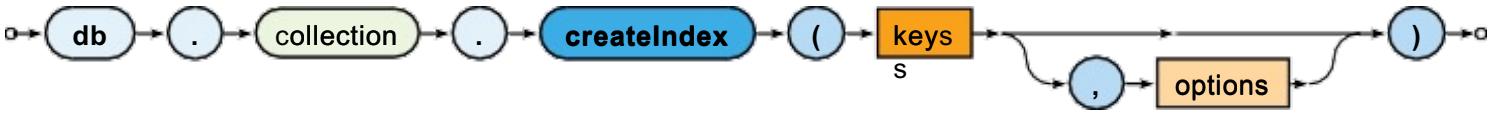
- Unique index on values of the **_id field**
- Created automatically

**Secondary indexes**

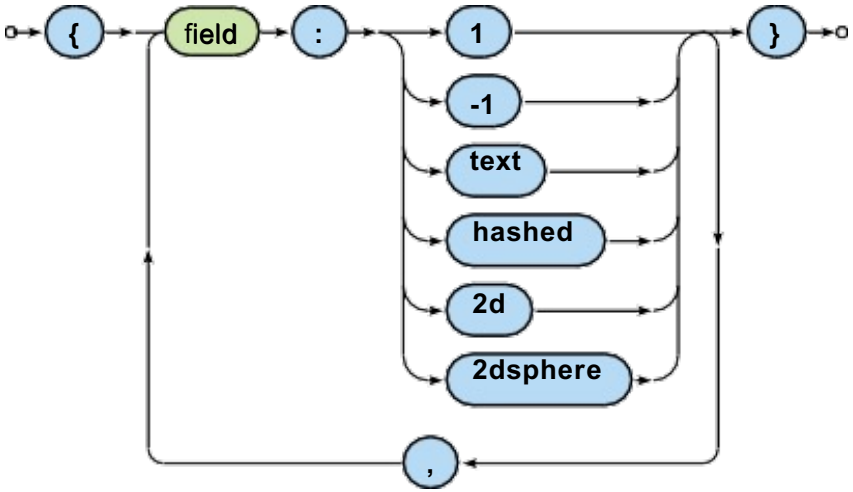
- Created manually for values of a given key field / fields
- Always within just a single collection

# Index Structures

Secondary **index creation**




Definition of keys (fields) to be involved



# Index Structures

**Index types**

- **`1, -1`** – standard ascending / descending value indexes
  - Both scalar values and embedded documents can be indexed
- **`hashed`** – hash values of a single field are indexed
- **`text`** – basic full-text index
- **`2d`** – points in planar geometry
- **`2dsphere`** – points in spherical geometry

# Index Structures

**Index forms**

- One key / multiple keys (**composed index**)
- Ordinary fields / array fields (**multi-key index**)

**Index properties**

- **Unique** – duplicate values are rejected (cannot be inserted)
- **Partial** – only certain documents are indexed
- **Sparse** – documents without a given field are ignored
- **TTL** – documents are removed when a timeout elapses

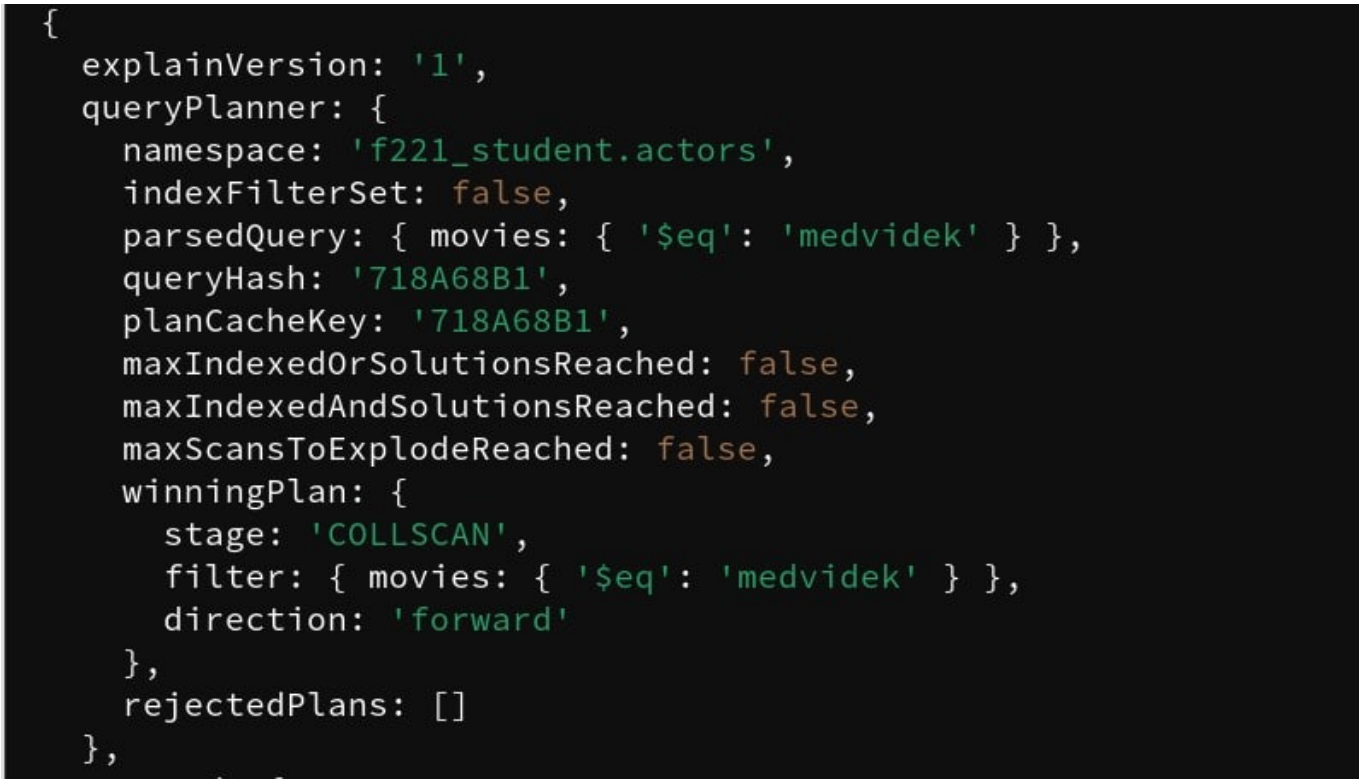
Just some type / form / property combinations can be used!

# Index Structures

Execute the following query and study its **execution plan**

```
db.actors.find({ movies: "medvidek" })
```

```
db.actors.find({ movies: "medvidek" }).explain()
```



```
{
  explainVersion: '1',
  queryPlanner: {
    namespace: 'f221_student.actors',
    indexFilterSet: false,
    parsedQuery: { movies: { '$eq': 'medvidek' } },
    queryHash: '718A68B1',
    planCacheKey: '718A68B1',
    maxIndexedOrSolutionsReached: false,
    maxIndexedAndSolutionsReached: false,
    maxScansToExplodeReached: false,
    winningPlan: {
      stage: 'COLLSCAN',
      filter: { movies: { '$eq': 'medvidek' } },
      direction: 'forward'
    },
    rejectedPlans: []
  },
```

# Index Structures

```
  command: {
    find: 'actors',
    filter: { movies: 'medvidek' },
    '$db': 'f221_student'
  },
  serverInfo: {
    host: 'database',
    port: 42222,
    version: '6.0.1',
    gitVersion: '32f0f9c88dc44a2c8073a5bd47cf779d4bfdee6b'
  },
  serverParameters: {
    internalQueryFacetBufferSizeBytes: 104857600,
    internalQueryFacetMaxOutputDocSizeBytes: 104857600,
    internalLookupStageIntermediateDocumentMaxSizeBytes: 104857600,
    internalDocumentSourceGroupMaxMemoryBytes: 104857600,
    internalQueryMaxBlockingSortMemoryUsageBytes: 104857600,
    internalQueryProhibitBlockingMergeOnMongoS: 0,
    internalQueryMaxAddToSetBytes: 104857600,
    internalDocumentSourceSetWindowFieldsMaxMemoryBytes: 104857600
  },
  ok: 1
}
```

# Index Structures

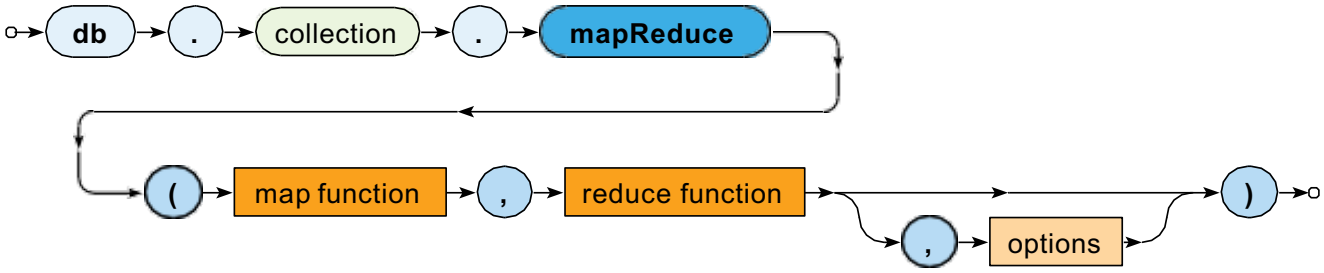
**Create a multikey index** for movies of actors

```
db.actors.createIndex({ movies: 1 })
```

Examine the execution plan once again

# MapReduce

Executes a **MapReduce** job on a selected collection




- Parameters
  - **Map**: JavaScript implementation of the Map function
  - **Reduce**: JavaScript implementation of the Reduce function
  - **Options**

# MapReduce

**Map function**

- Current document is accessible via `this`
- `emit(key, value)` is used for emissions

**Reduce function**

- Intermediate key and values are provided as arguments
- Reduced value is published via `return`

**Options**

- `query`: only matching documents are considered
- `sort`: they are processed in a specific order
- `limit`: at most a given number of them is processed
- `out`: output is stored into a given collection

# MapReduce: Example

Count the number of movies filmed in each year, starting in *2005*

```
db.movies.mapReduce(
  function() {
    emit(this.year, 1);
  },
  function(key, values) {
    return Array.sum(values);
  },
  {
    query: { year: { $gte: 2005 } },
    sort: { year: 1 },
    out: "statistics"
  }
)
```

# MapReduce

Implement and execute the following MapReduce jobs

- **Find a list of actors (their names sorted alphabetically) for each year (they were born)**
  - Only consider actors born in year 2000 or before
  - `values.sort()`
  - Use `out: { inline: 1 }` option
- **Calculate the overall number of actors for each movie**
  - `this.movies.forEach(function(m) { ..})`
  - `Array.sum(values)`
  - Use `out: { inline: 1 }` option once again

# References

Documentation

- https://docs.mongodb.com/v3.2/