NDBI040: **Big Data Management and NoSQL Databases**

http://www.ksi.mff.cuni.cz/~svoboda/courses/2016-1-NDBI040/

Practical Class 4

MongoDB Document Database

Martin Svoboda
svoboda@ksi.mff.cuni.cz

15. 11. 2016

Charles University in Prague, Faculty of Mathematics and Physics
Czech Technical University in Prague, Faculty of Electrical Engineering

Data Model

Database system structure

Instance $\rightarrow$ **databases** $\rightarrow$ **collections** $\rightarrow$ **documents**

- Database
- Collection
  - Collection of documents, usually of a similar structure
- Document
  - MongoDB **document** = **one JSON object**
  - Each document…
    - belongs to right one collection
    - has a **unique immutable identifier** `_id`
  - Field name restrictions apply
    - `_id`, `$`, `.`

CRUD Operations

Overview

- `db.collection.insert()`
  - Inserts a new document into a collection
- `db.collection.update()`
  - Modifies an existing document / documents or inserts a new one
- `db.collection.remove()`
  - Deletes an existing document / documents
- `db.collection.find()`
  - Finds documents based on filtering conditions
  - Projection and / or sorting may be applied too

Tutorial: MongoDB

First Steps

Remotely connect to our NoSQL server

- SSH and SFTP access
- PuTTY and WinSCP on Windows
- **nosql.ms.mff.cuni.cz:42222**

Start mongo shell

- `mongo`

Basic useful commands

- `help`
  - Displays a brief description of basic 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

Collections

Create a new collection for actors

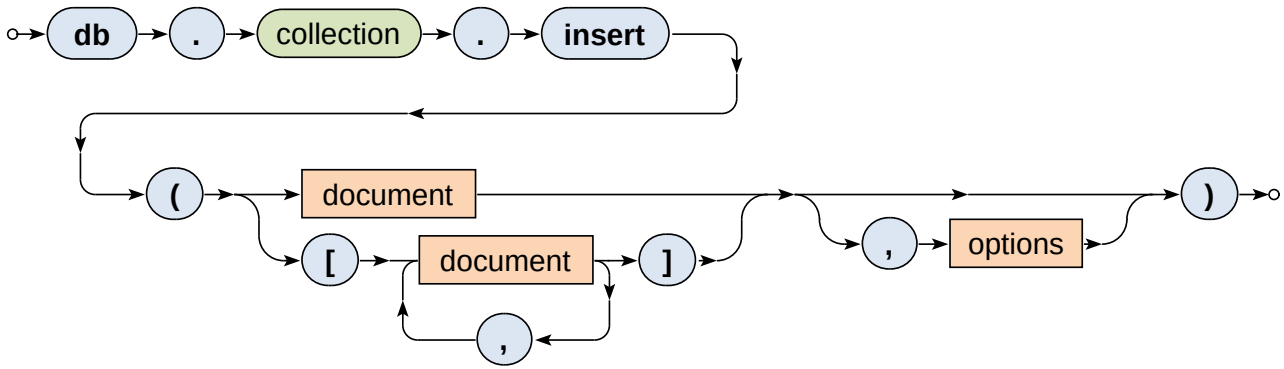
- `db.createCollection("actors")`
  - Suitable when creating collections with specific options since collections can also be created implicitly

List all the collections within your databases

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

Insert Operation

Inserts a new document / documents into a given collection



Insertions

Insert a few new documents into the collection of actors

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

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

```
db.actors.insert({ _id: ObjectId(), name: "Jitka Schneiderova" })
```

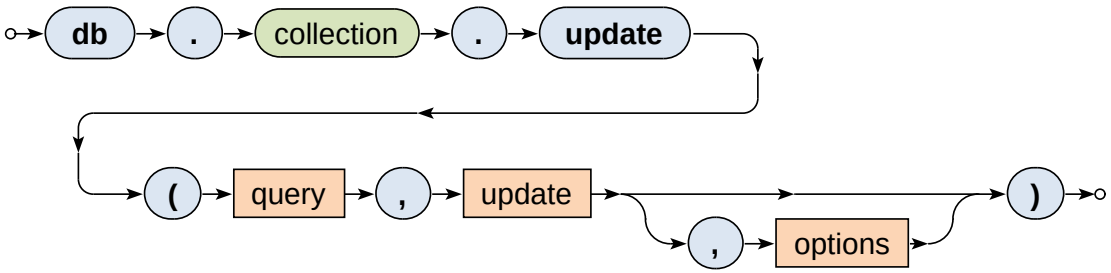
```
db.actors.insert({ name: "Zdenek Sverak" })
```

Retrieve all the actor documents

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

Update Operation

Modifies / replaces an existing document / documents



Updates

Update the document of actor *Ivan Trojan*

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

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

- 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" })
```

Insertions

Use `update` method to **insert a new actor**

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

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

Use `save` method to **insert new actors**

- Document identifier must not be specified in the query or must not exist in the collection

```
db.actors.save({ name: "Tatiana Vilhelmova" })
```

```
db.actors.save({ _id: 6, name: "Sasa Rasilov" })
```

Updates

Use `save` method to **update actor *Ivan Trojan***

- Document identifier must be specified in the query and must exist in the collection

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

Try to modify the document identifier of an existing document

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

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

Updates

Update the document of actor *Ivan Trojan*

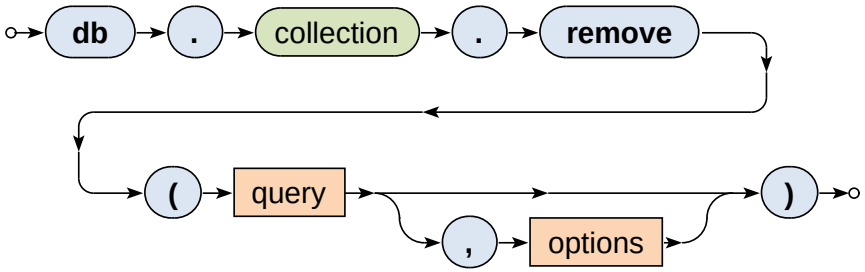
```
db.actors.update(
  { _id: "trojan" },
  {
    $set: { year: 1964, age: 52 },
    $inc: { rating: 1 },
    $push: { movies: { $each: [ "samotari", "medvidek" ] } }
  }
)
```

Update multiple documents at once

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

Remove Operation

Removes a document / documents from a given collection



Removals

Remove selected documents from the collection of actors

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

```
db.actors.remove(
  { year: { $lt: 2000 } },
  { justOne: true }
)
```

Remove all the documents from the collection of actors

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

Sample Data

Insert the following actors into your emptied collection

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

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

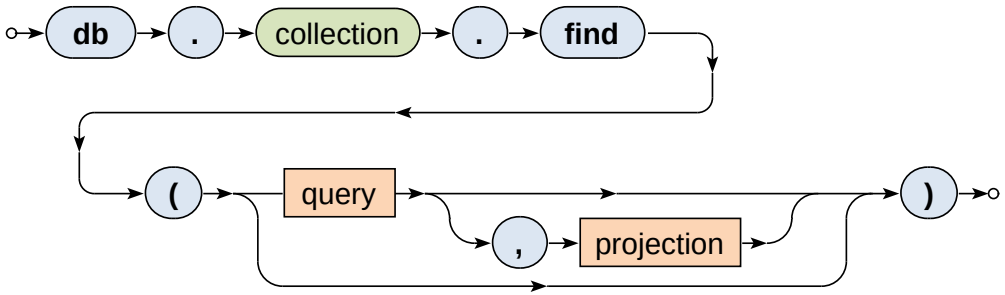
```
{ _id: "schneiderova",
  name: "Jitka Schneiderova", year: 1973,
  movies: [ "samotari" ] }
```

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

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

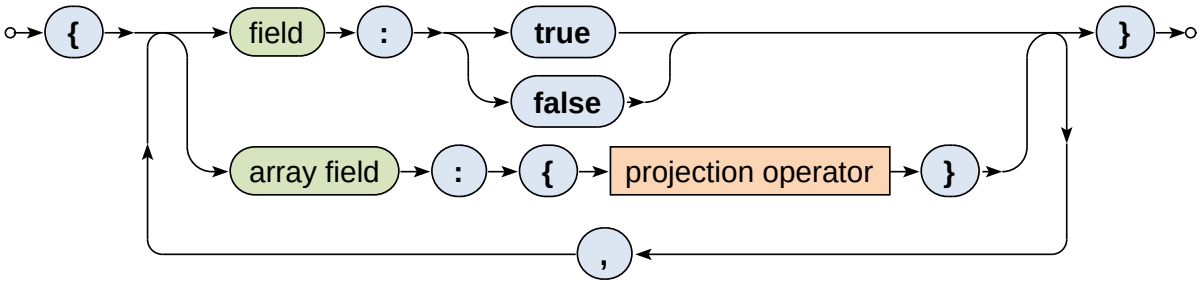
Find Operation

Selects documents from a given collection



Find Operation: Projection

Projection allows us to determine the fields returned in the result

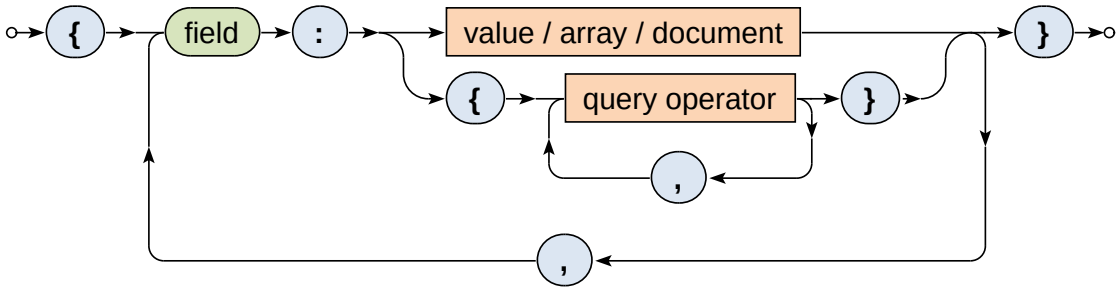


Projection operators

- `$elemMatch`, `$slice`,…

Find Operation: Selection

Query parameter describes the documents we are interested in



Selection operators

- `$eq`, `$neq`, `$lt`, `$lte`, `$gte`, `$gt`
- `$in`, `$nin`
- `$and`, `$or`, `$not`
- `$exists`, `$regex`, `$text`
- …

Querying

Execute the following **document retrievals**

- Explain the meaning of these queries as well

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

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

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

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

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

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

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

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

Querying

Execute the following **document retrievals**

- Explain the meaning of these queries as well

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

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

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

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

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

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

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

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

Indexing

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

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

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

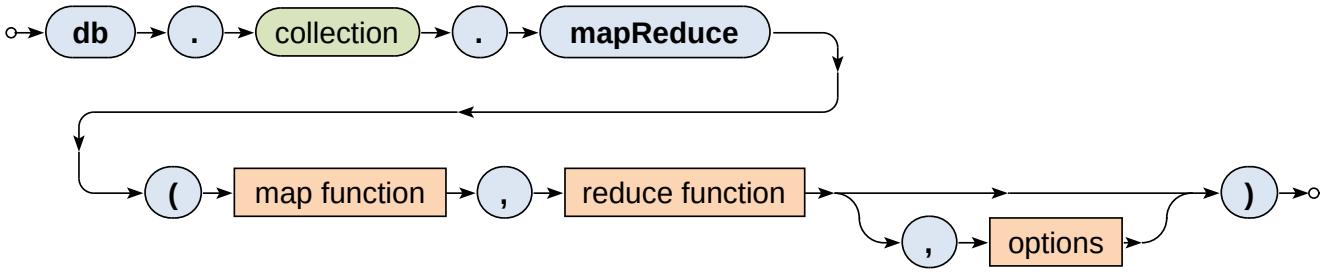
Create multikey index for movies of actors

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

Examine the execution plan once again

MapReduce

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



MapReduce

Execute the following **MapReduce job**

```
db.actors.mapReduce(
  function() {
    emit(this.year, this.name);
  },
  function(key, values) {
    return values.sort();
  },
  {
    query: { year: { $lte: 2000 } },
    sort: { year: -1 },
    out: { inline: 1 }
  }
)
```

MapReduce

Implement and execute the following **MapReduce job**

- Calculate the overall number of actors for each movie
- Find inspiration in…
  - `this.movies.forEach(function(m) { … })`
  - `Array.sum(values)`

References

Documentation

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