MDK: **Modern Database Concepts**

http://www.ksi.mff.cuni.cz/~svoboda/courses/192-MDK/

Practical Class 9

MongoDB

Martin Svoboda
svoboda@ksi.mff.cuni.cz

5. 6. 2020

Charles University, Faculty of Mathematics and Physics
OTH Regensburg, Faculty of Computer Science and Mathematics

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**
    - 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

- `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

Mongo Shell

Connect to our NoSQL server

- SSH / PuTTY and SFTP / WinSCP
- **nosql.ms.mff.cuni.cz:42222**

Start mongo shell

- `mongo`

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

Collections

Create a new collection for actors

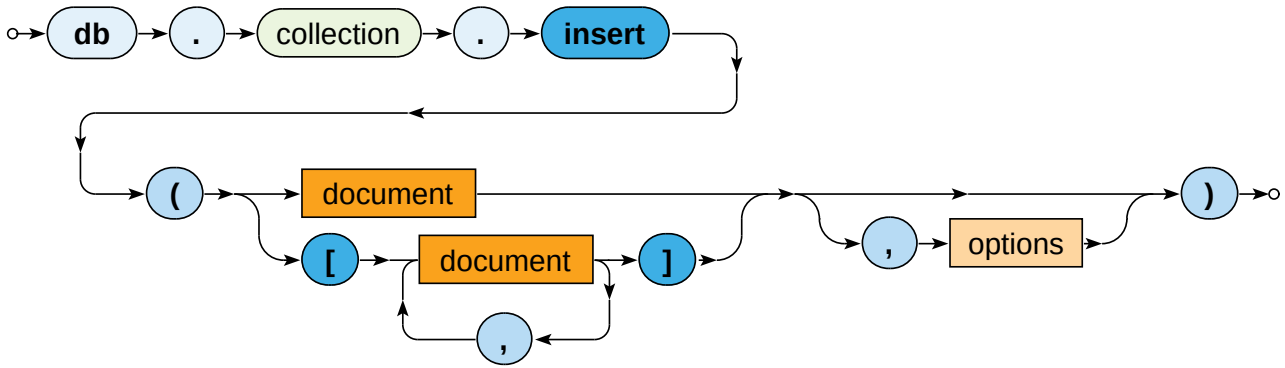
- `db.createCollection("actors")`
  - Suitable when creating collections with specific options since collections can also be created implicitly

List all collections in your database

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

Insert Operation

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" })
```

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

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

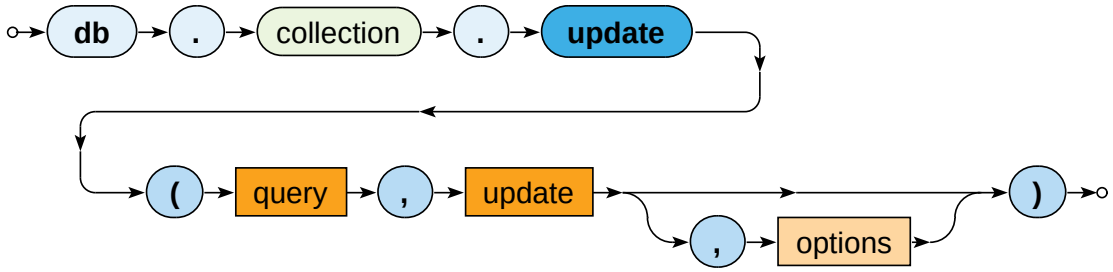
```
db.actors.insert({ name: "Zdenek Sverak" })
```

Retrieve all documents from the collection of actors

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

Update Operation

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.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" })
```

Update Operation

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

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

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

Update Operation

Update the document of actor *Ivan Trojan*

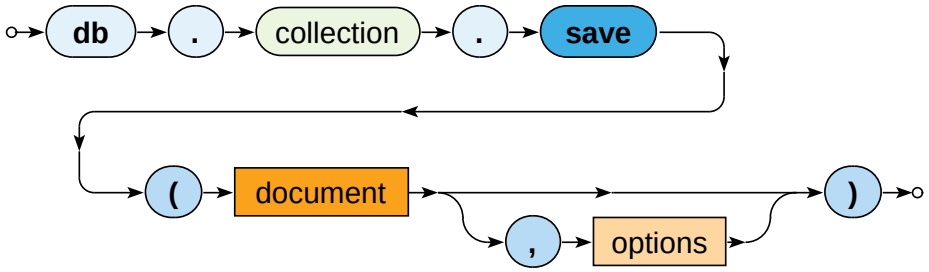
```
db.actors.update(
  { _id: "trojan" },
  {
    $set: { year: 1964, age: 52 },
    $inc: { rating: 1 },
    $push: { movies: { $each: [ "samotari", "medvidek" ] } }
  }
)
```

Update multiple documents at once

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

Save Operation

Replaces an existing / inserts a new document



- Parameters
  - **Document**: document to be modified / inserted
  - **Options**

Save Operation

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

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

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

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

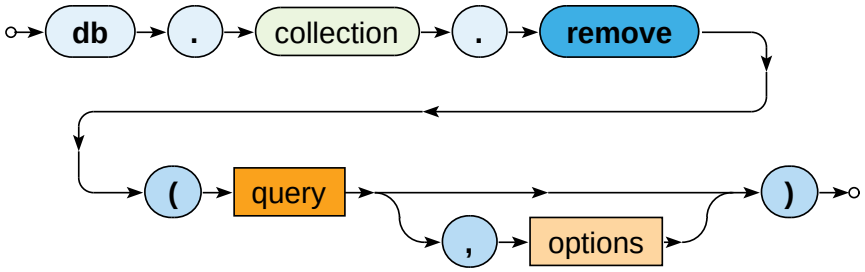
Use `save` method to **update actor *Ivan Trojan***

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

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

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

```
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" ] }
```

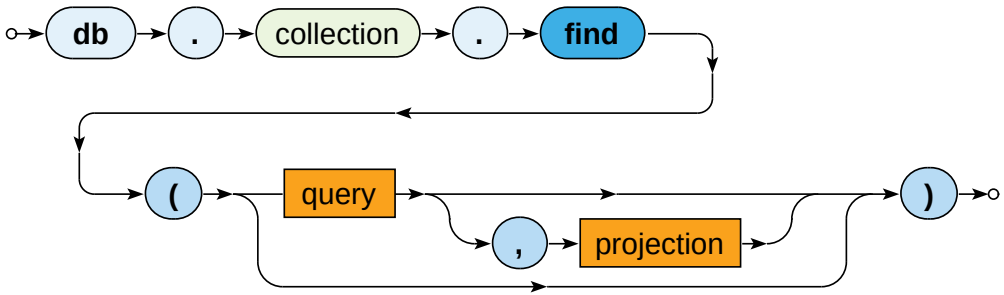
```
{ _id: "schneiderova",
  name: "Jitka Schneiderova", year: 1973,
  movies: [ "samotari" ] }
```

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

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

Find Operation

Selects documents from a given collection



- 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 })
```

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

```
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)
```

Index Structures

Motivation

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

Primary index

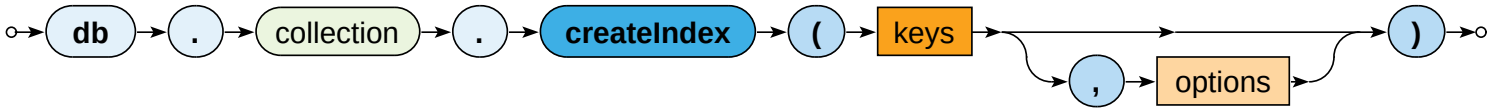
- Unique index on values of the **_id field**
- Created automatically

Secondary indexes

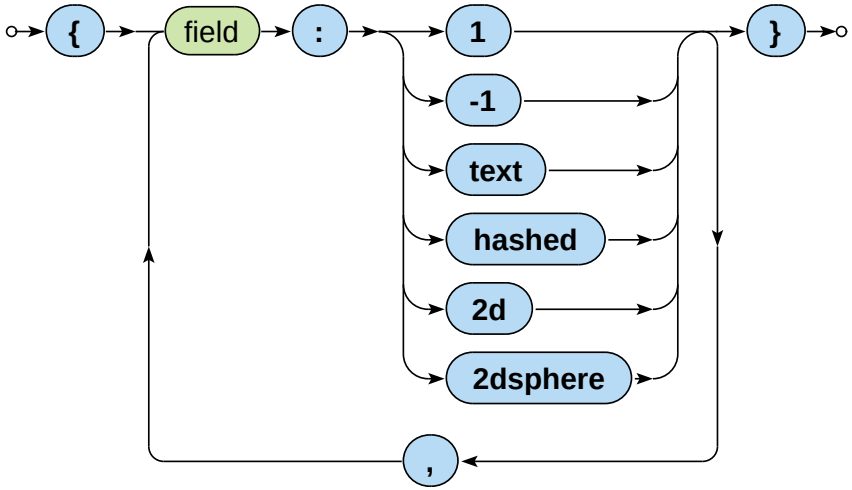
- Created manually for values of a given key field / fields
- Always within just a single collection

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()
```

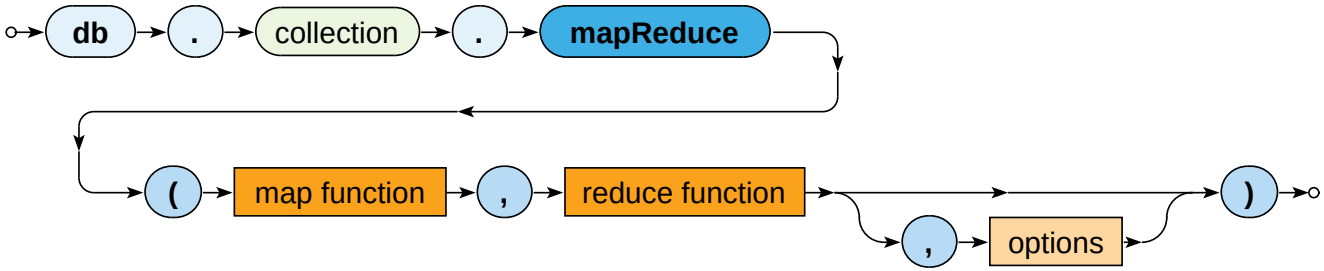
Create a multikey index for movies of actors

```
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/