NDBI040: PRACTICAL CLASS 5

# MONGODB

Based on NDBI040 practical class materials created by Martin Svoboda; Tutor: Pavel Čontoš; November 18th 2020

## (RECOMMENDED) REQUIREMENTS

- Database concepts
- Javascript (basics)
- macOS / Linux command line or PuTTy / WinSCP on Windows

## SERVER ACCESS

### CONNECT TO NOSQL SERVER

- `ssh` on macOS / Linux
- `PuTTy` on Windows

- nosql.ms.mff.cuni.cz:42222
- Login and password send by e-mail
- Change your initial password (if not yet changed) by `passwd`

### TRANSFER FILES

- `scp` on macOS / Linux
- `WinSCP` on Windows

## DATA MODEL

- `Instance` → `database` → `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

- `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 document based on filtering conditions
  - Projection and / or sorting may be applied too

## MONGO SHELL

### START MONGO SHELL

- `mongo`

### 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

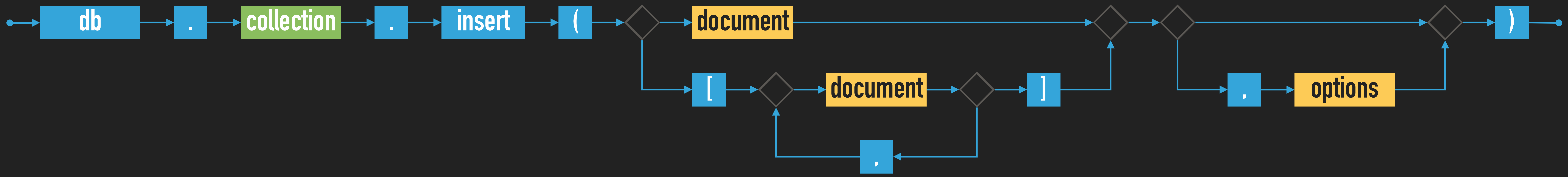
- `db.createCollection("actors")`
    - Suitable when creating collection with specific options since collections can also be created implicitly

### LIST ALL COLLECTIONS IN YOUR DATABASE

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

## INSERT OPERATION

- Inserts a new document / documents into a given collection



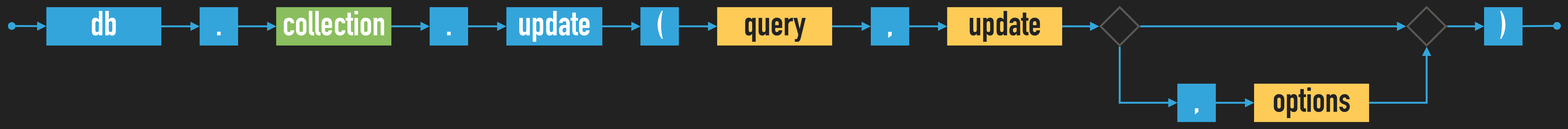
- Parameters
  - `Document`: one or more documents to be inserted
  - `Options`

## EXERCISE 1: INSERT AND RETRIEVE DOCUMENTS (SOLVED)

- 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, ...`

## EXERCISE 2: UPDATE OPERATION (SOLVED)

- 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" } )`

## EXERCISE 3: UPSERT (SOLVED)

- 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 } )
      ```

## EXERCISE 4: IDENTIFIER MODIFICATION (SOLVED)

- 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 } )
    ```

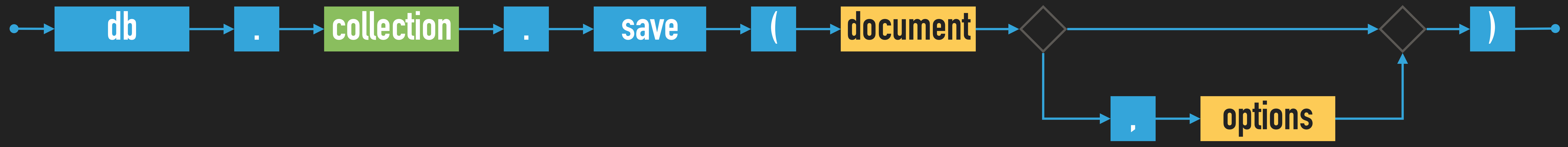
## EXERCISE 5: MULTIPLE DOCUMENTS UPDATE (SOLVED)

- 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 modifier / inserted
  - `Options`

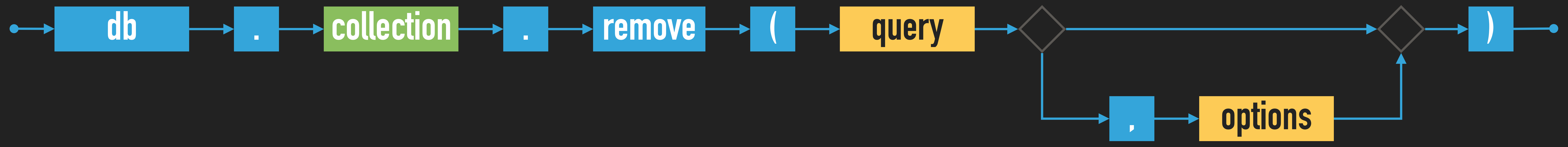
## EXERCISE 6: SAVE OPERATION (SOLVED)

- 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

## EXERCISE 7: REMOVE OPERATION (SOLVED)

- Remove selected documents from the collection of actors
  - `db.actors.remove( { _id: "geislerova" } )`
  - `db.actors.remove( { year: { $lt: 2000 } }, { justOne: true } )`

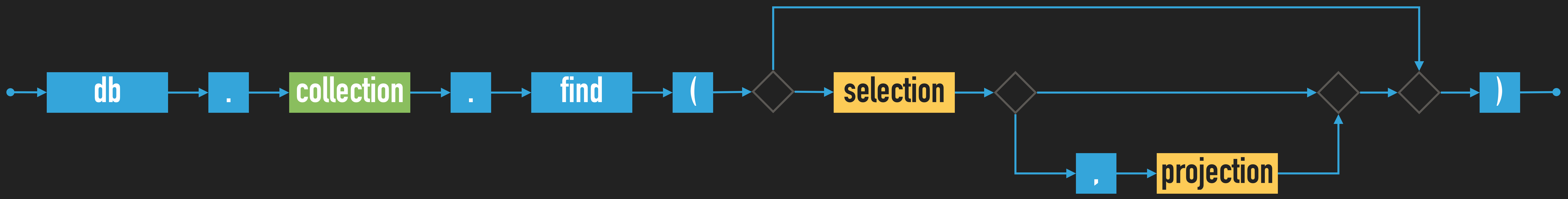
- Remove all the documents from the collection of actors
  - `db.actors.remove( { } )`

## INSERT SAMPLE DATA

- Insert sample data into your emptied database
  - See `/home/NOSQL/mongodb/data.js`
  - Or download `data.js` from practical class website

## FIND OPERATION

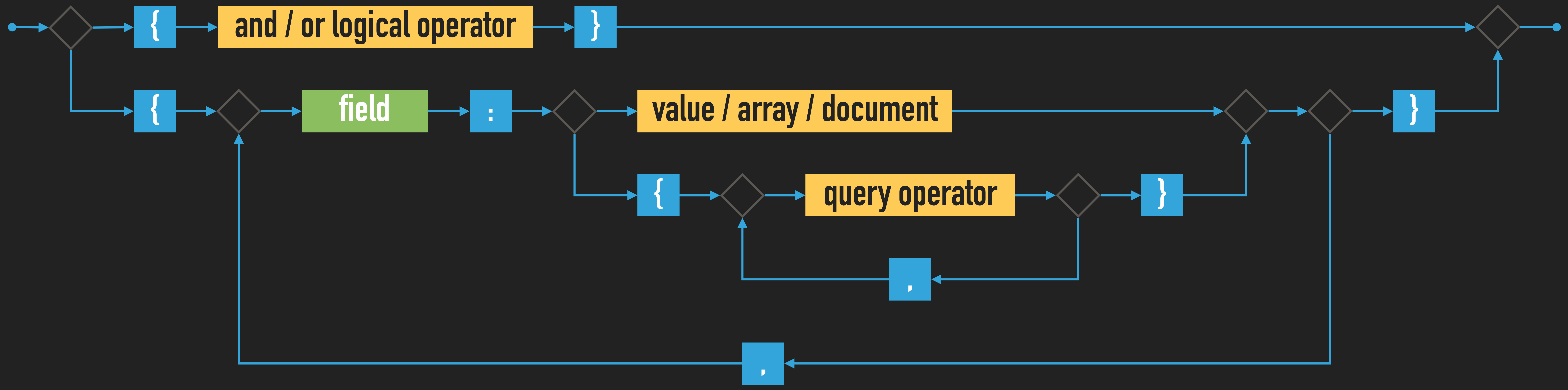
- Selects documents from a given collection



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

## SELECTION

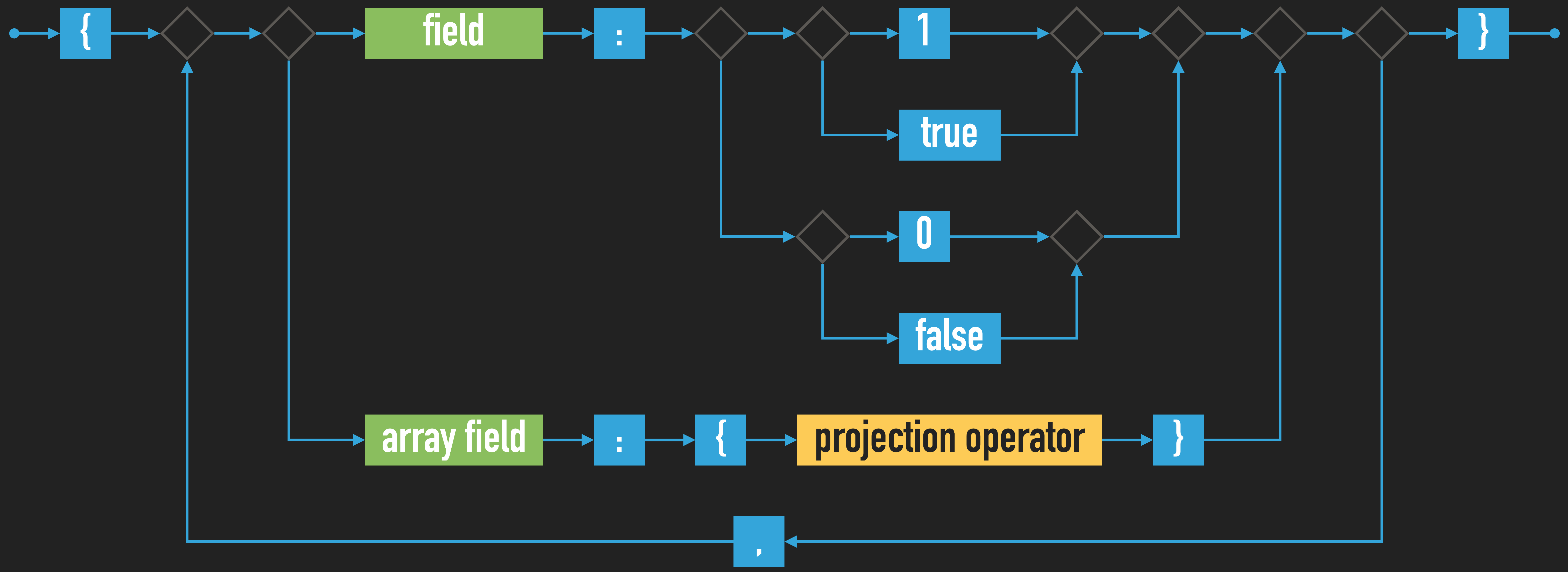
- Describes the documents we are interested in



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

## PROJECTION

- Allows us to determine fields returned in the result



- Projection operators
  - $elemMatch, $slice, …

## EXERCISE 8: 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.first": "Ivan", 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", "vratnelahve" ] } } )`
  - `db.actors.find( { movies: { $all: [ "medvidek", "samotari" ] } } )`

## EXERCISE 8: 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)`

## EXERCISE 9

- Find actors born in 1966 with first name Jiri

## EXERCISE 10

- Find movies directed by Jan Hrebejk
  - Note that the order of fields for first and last names can be arbitrary

## EXERCISE 11

- Find actors with first name Jiri who played in Medvidek movie
- Return names of these actors only

## EXERCISE 12

- Find movies filmed between years 2000 and 2005 such that they have a director specified
- Return movie identifier only
- Order the result by ratings in descending order and then by years in ascending order

## EXERCISE 13

- Find actors who stared in Samotari or Medvidek movies
- Return actor identifier only
- Propose two different approaches

## EXERCISE 14

- Find actors who played in both Samotari and Medvidek
- Return actor identifier only
- Propose two different approaches

## EXERCISE 15

- Find movies with Czech title equal to Vratne lahve
- Return movie title only
- Note that there are two means how movie titles are defined

## EXERCISE 16

- Find movies that have a Czech Lion award from 2005
- Return movie identifier and all awards

## EXERCISE 17

- Find movies that are comedies and dramas at the same time or have a rating 80 or more
- Return movie identifier and at most 2 countries

## INDEX STRUCTURES

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

### PRIMARY INDEX

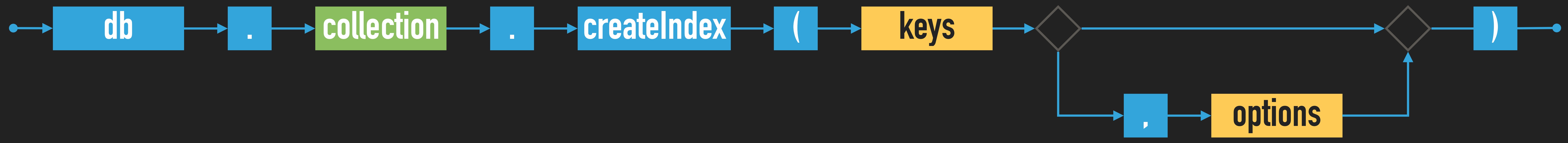
- Unique index on values of the `_id` field
- Created automatically

### SECONDARY INDEXES

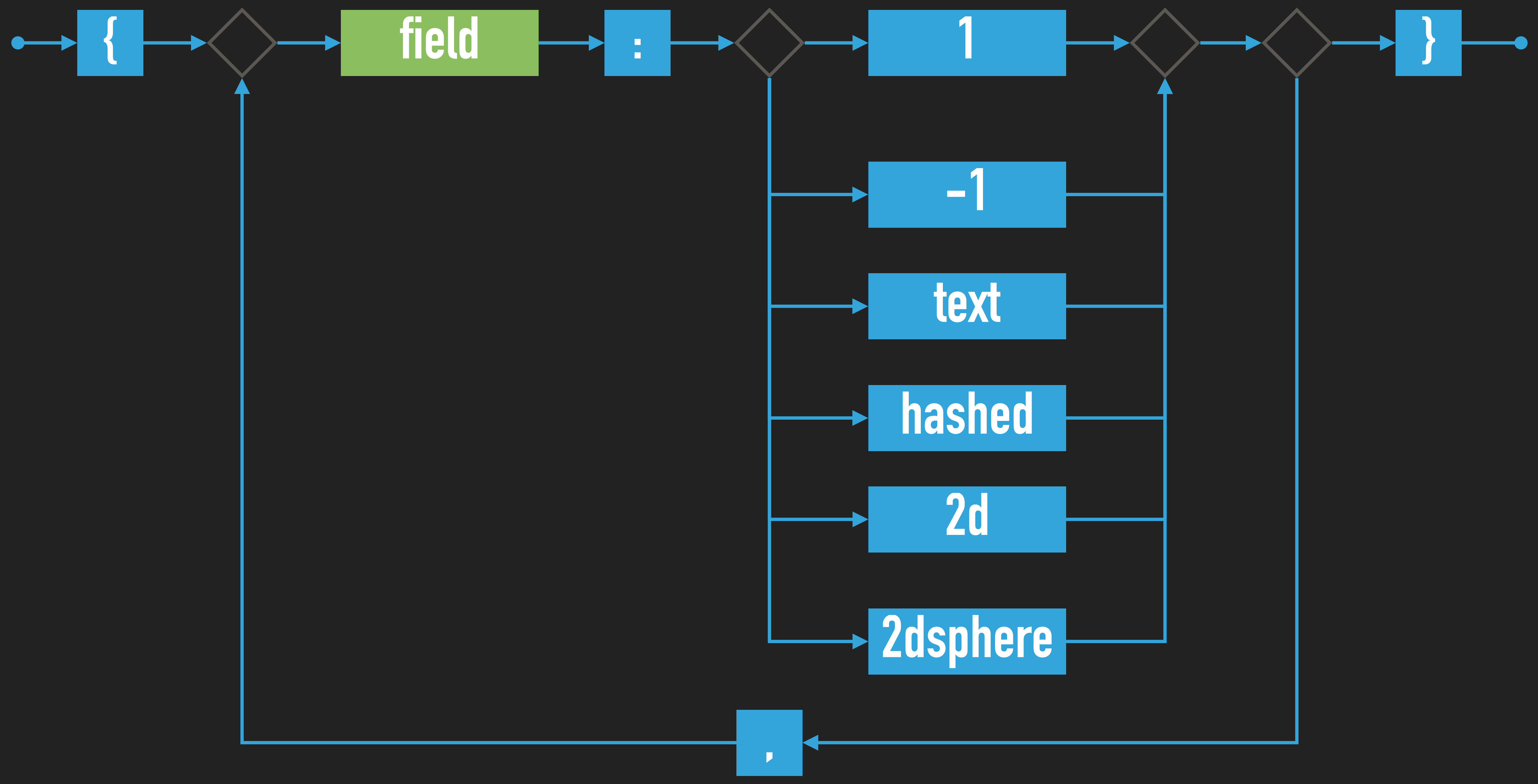
- Created manually for values of a given key field / fields
- Always within just a single collection

## 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

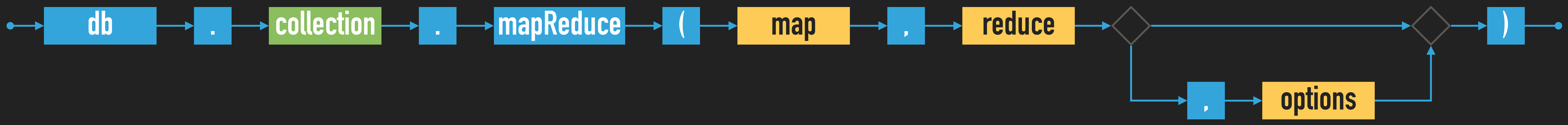
- Just some type / form / property combinations can be used

## EXERCISE 18: INDEX STRUCTURES (SOLVED)

- 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

## EXERCISE 19: MAPREDUCE (SOLVED)

- 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"
  }
)
```

## EXERCISE 20

- 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/
  - https://docs.mongodb.com/manual/ (latest version)