Document Databases

Fundamentals

- Basic concept of data is: Document
- Documents are self-describing piece of information
    - hierarchical data structure
        - nested arrays, nested objects
    - contains related information
    - XML, JSON, BSON
- Documents in a collection are similar; all XML or JSON
    - their schema can differ
- Document often contains values of key-value pair, like in JSON
    - indexes can be applied on various fields / keys

Representatives

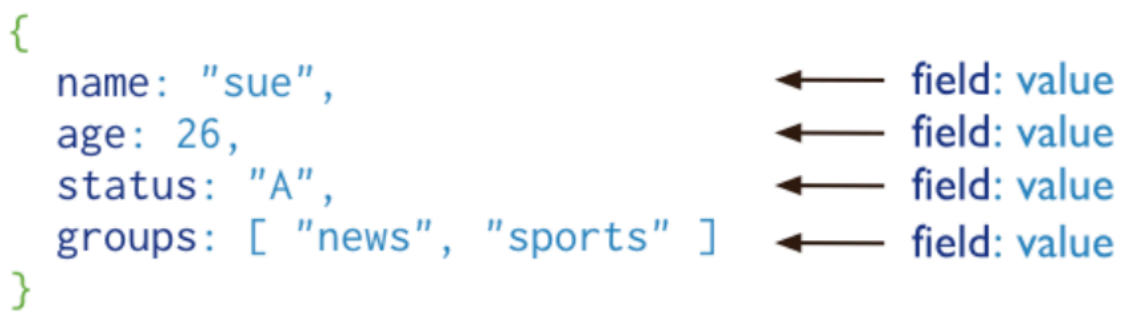
mongoDB
Apache
CouchDB
relax
OrientDB
RAVENDB
terrastore
amazon
DynamoDB
MS Azure
DocumentDB
PostgreSQL

Ranked list: http://db-engines.com/en/ranking/document+store

MongoDB: Basics, Features, installation, Queries

MongoDB

- JSON documents database https://www.mongodb.com/
- Initial release in 2009
- written in C++, C, JS
- open source
- cross platform
  - works on linux, Mac OS x, windows, …



```
{
   name: "sue",
   age: 26,
   status: "A",
   groups: [ "news", "sports" ]
}
```

Basics features

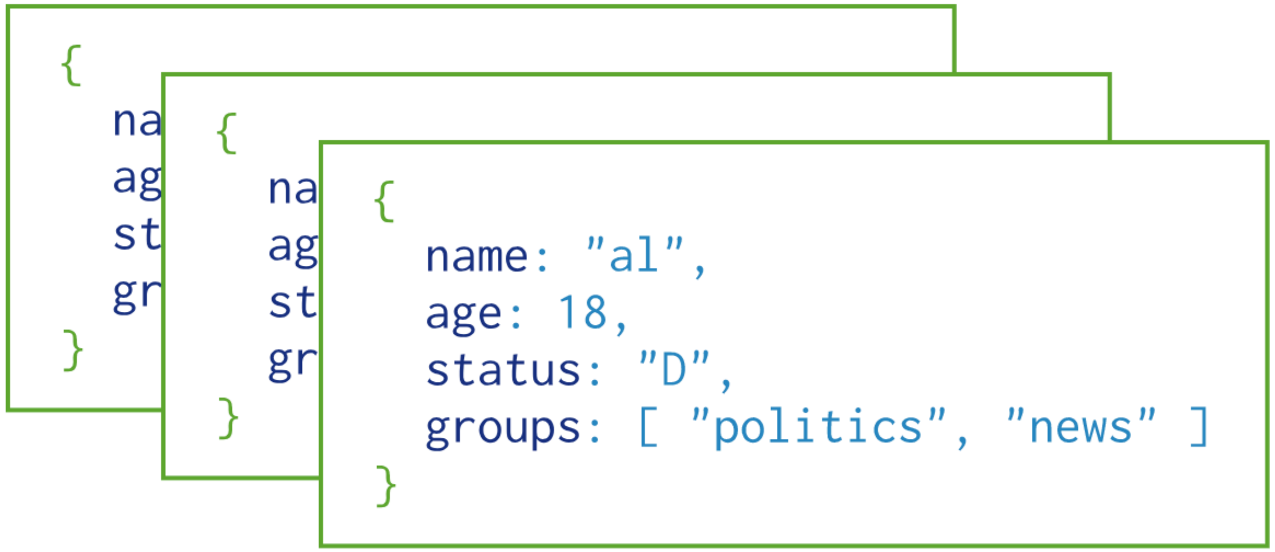
- High performance
    - shards, secondary indexes, data sorted using B Tree
- Automatic scaling
    - automatic sharding across the cluster
- High availability
    - master-slave replication, eventual consistency
- MapReduce support

MongoDB: Data Model

- Structure:
    - instance —> databases —> collections —> documents
- collection
    - consists of documents, usually of similar structure
- document
    - one MongoDB document = JSON object

MongoDB: Document

- Each JSON document
    - belong to a collection
    - has a unique identifier (_id) field, which must be unique
- Internally stored as BSON (Binary JSON)
- Maximal allowed size: 16MB (BSON)
    - use GridFS tool to divide large files into fragments



```
{
   name: "al",
   age: 18,
   status: "D",
   groups: [ "politics", "news" ]
}
```

Collection

MongoDB: Fields

- _id is reserved for the primary key
- Field names
    - cannot start with **$**
        - reserved for query operators
    - cannot contain **.**
        - used for accessing nested fields

MongoDB: Primary Key

- is the document identifier
- Features:
  - unique within a collection
  - Immutable (cannot be changed once assigned)
  - can be of any type except array

MongoDB: Identifier Design

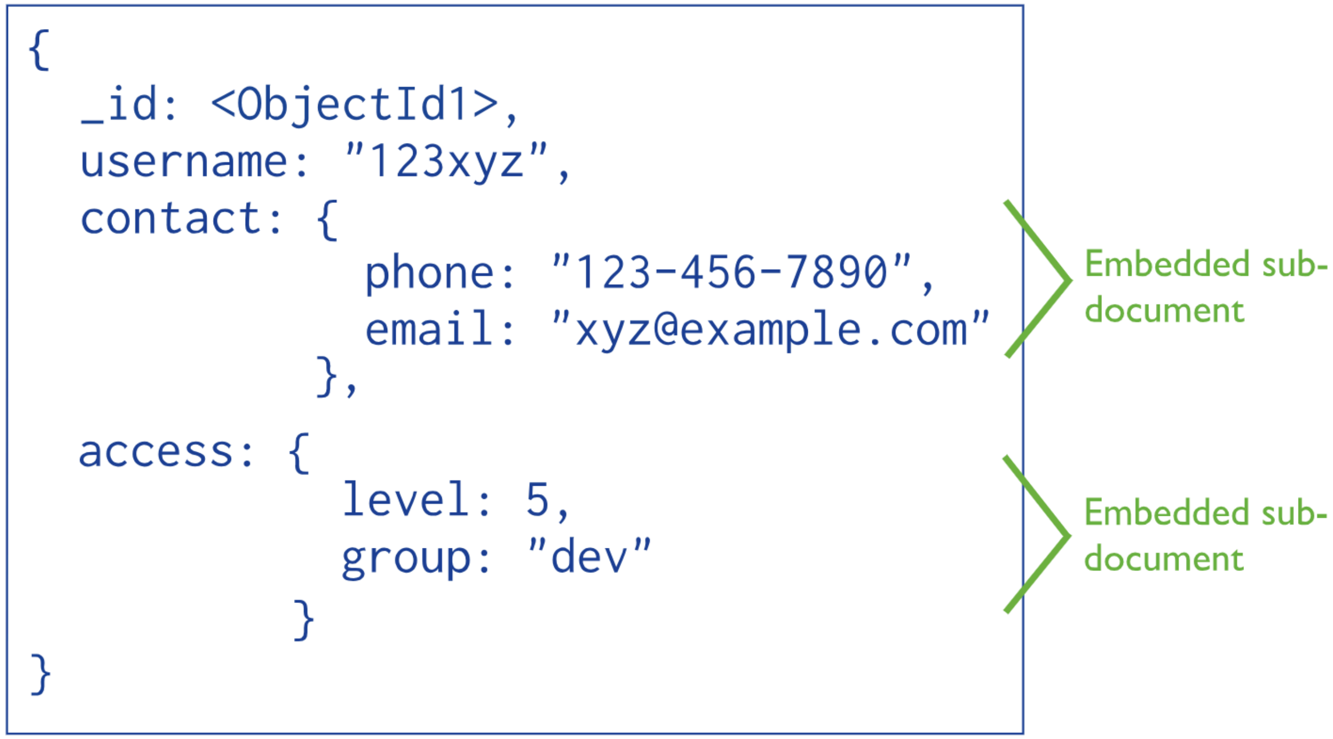
- Design
  - Natural identifier
    - each document comes with a uniq identifier
  - Auto incrementing number - not recommended
    - can be slow, one counter to make sure that the number is unique
  - Universally Unique Identifier (UUID)
    - 128 bit, longer compared to the ObjectId below
    - standard libraries can be used for that
  - ObjectId (default)
    - 12 bytes (96 bits) length
    - 4 bytes representing the timestamp in seconds, 3 bytes machine identifier (usually derived from MAC address), 2 bytes (process id), 3 bytes (counter)

MongoDB: Schema

- Documents have flexible schema
  - schema is not required or enforced
- Key decision for data modeling
  - references vs. embedded documents
- It is important because it controls
  - the aggregate content
  - the data structure
  - relationship between data

Schema: embedded docs

- contact & access can be considered sub-documents
- related data in one document
    - the aggregate will contain all related data



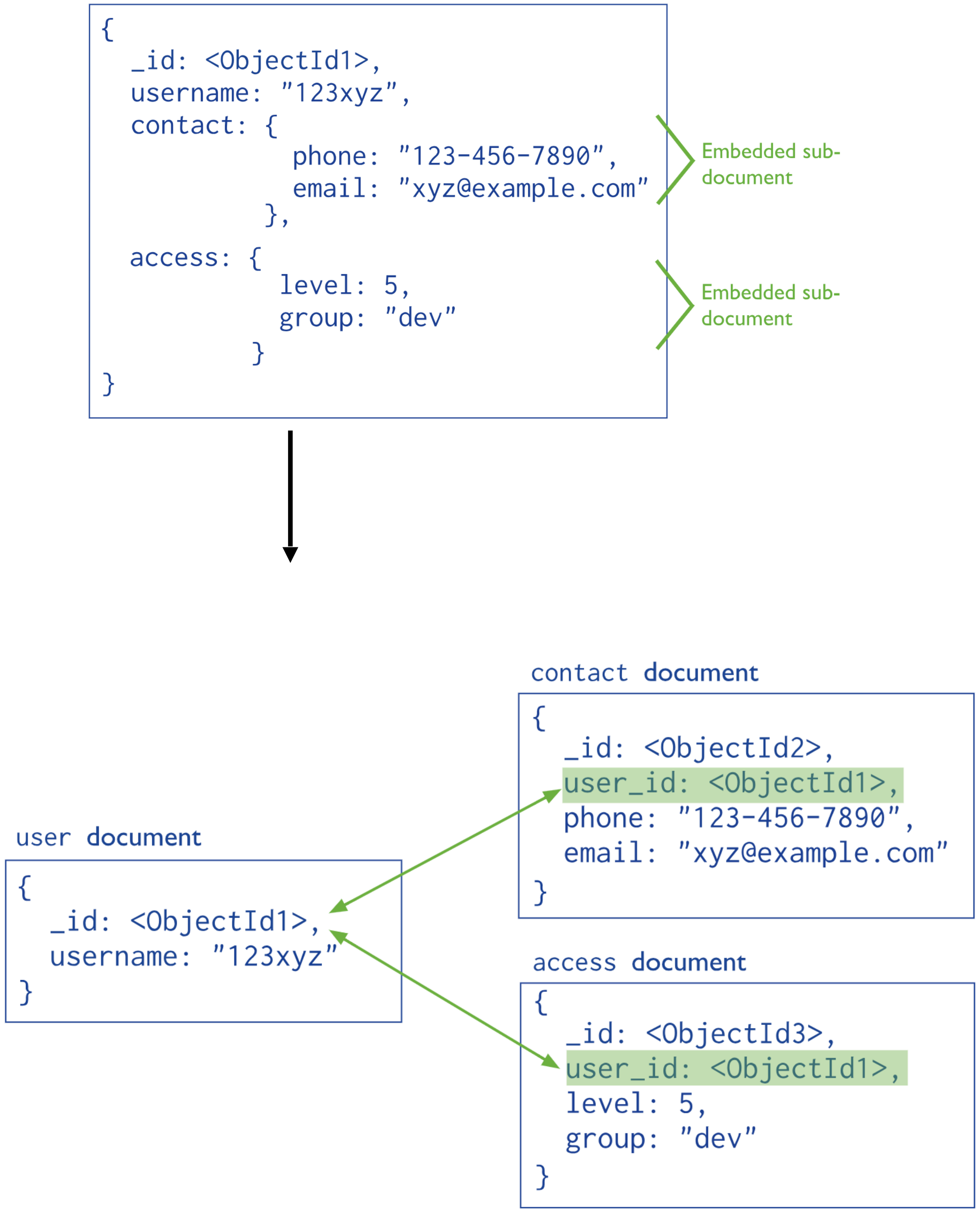
```
{
  _id: <ObjectId1>,
  username: "123xyz",
  contact: {
              phone: "123-456-7890",
              email: "xyz@example.com"
            },
  access: {
             level: 5,
             group: "dev"
           }
}
```

Schema: embedded docs

- Called denormalized schema
  - document is not flat, contains nested sub-docs
- Benefits:
  - manipulate related data in one operation
    - better performance, less queries
- when to use this
  - one-to-one, one-to-many relationship
- Drawback:
  - document size might exceed max. allowed doc. size

Schema: references

- links / references from one document to another
- normalized schema
  - flat document



```
{
  _id: <ObjectId1>,
  username: "123xyz",
  contact: {
              phone: "123-456-7890",
              email: "xyz@example.com"
            },
  access: {
             level: 5,
             group: "dev"
          }
}
```

user document

```
{
  _id: <ObjectId1>,
  username: "123xyz"
}
```

contact document

```
{
  _id: <ObjectId2>,
  user_id: <ObjectId1>,
  phone: "123-456-7890",
  email: "xyz@example.com"
}
```

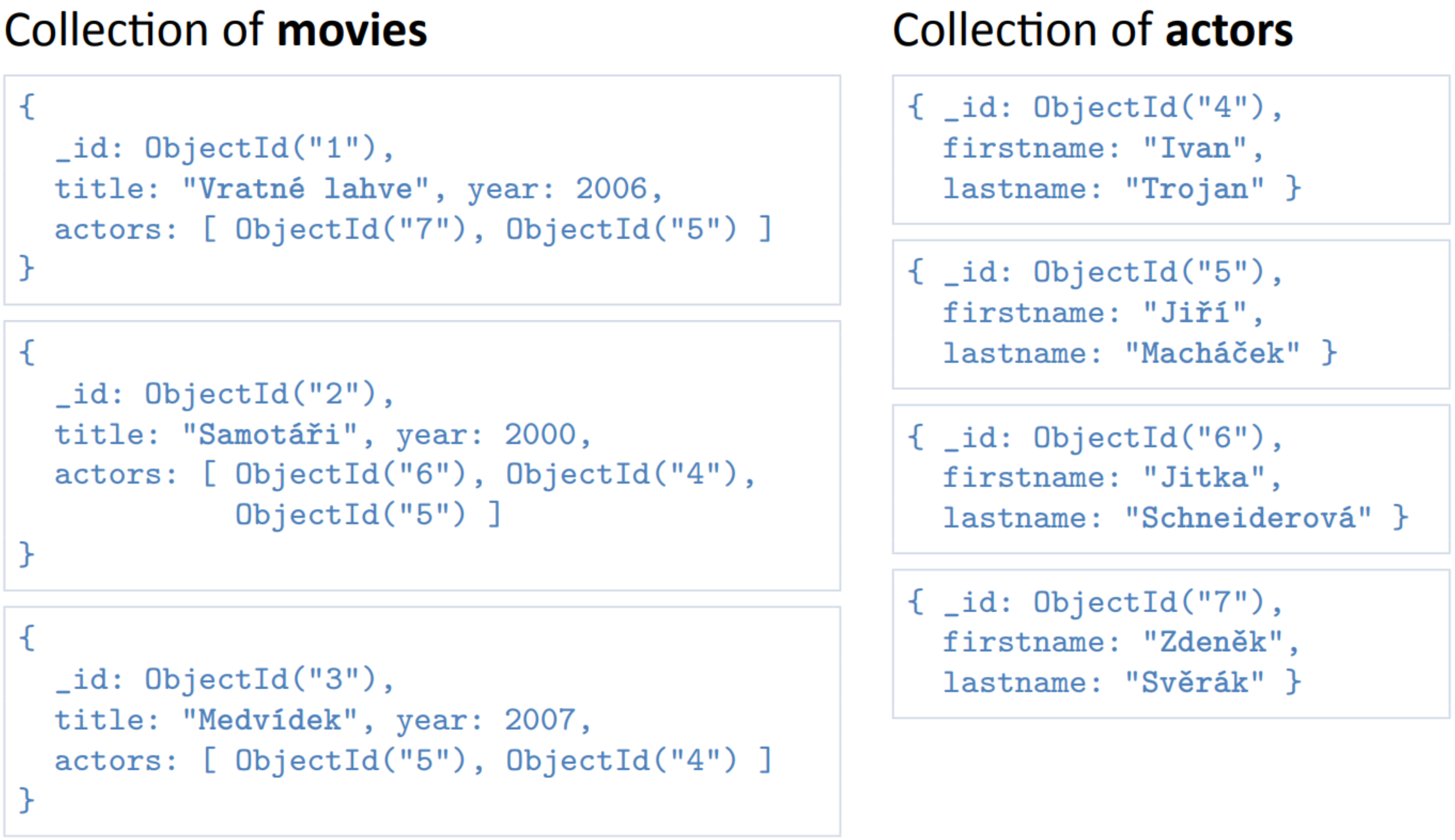
access document

```
{
  _id: <ObjectId3>,
  user_id: <ObjectId1>,
  level: 5,
  group: "dev"
}
```

Schema: references

- Useful to model
  - large hierarchal collection
  - many-to-many relationships
  - Drawback
    - various queries to related data might be required
      - related data is stored in multiple documents

Collections Example



Collection of **movies**

```
{
  _id: ObjectId("1"),
  title: "Vratné lahve", year: 2006,
  actors: [ ObjectId("7"), ObjectId("5") ]
}
```

```
{
  _id: ObjectId("2"),
  title: "Samotáři", year: 2000,
  actors: [ ObjectId("6"), ObjectId("4"),
            ObjectId("5") ]
}
```

```
{
  _id: ObjectId("3"),
  title: "Medvídek", year: 2007,
  actors: [ ObjectId("5"), ObjectId("4") ]
}
```

Collection of **actors**

```
{ _id: ObjectId("4"),
  firstname: "Ivan",
  lastname: "Trojan" }
```

```
{ _id: ObjectId("5"),
  firstname: "Jiří",
  lastname: "Macháček" }
```

```
{ _id: ObjectId("6"),
  firstname: "Jitka",
  lastname: "Schneiderová" }
```

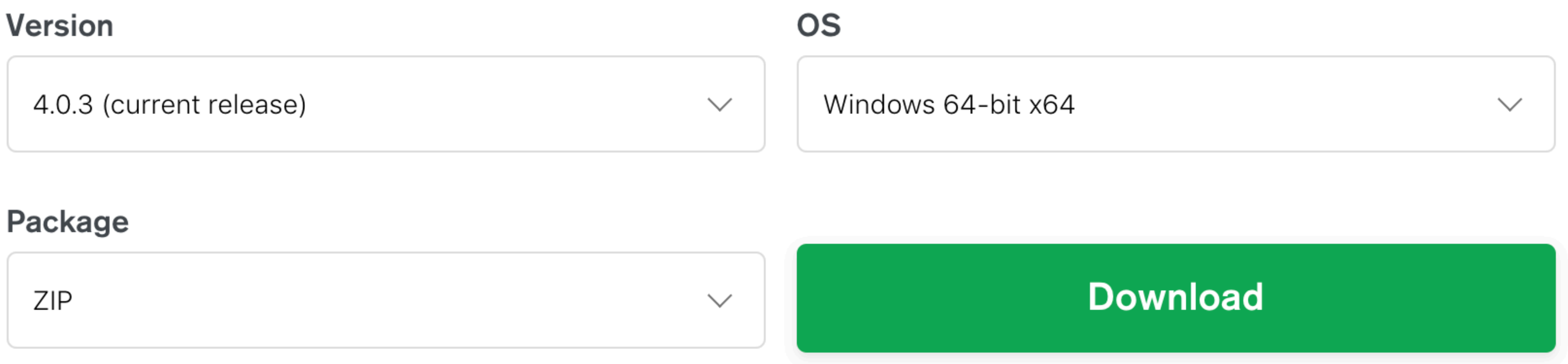
```
{ _id: ObjectId("7"),
  firstname: "Zdeněk",
  lastname: "Svěrák" }
```

MongoDB: Install

- Consideration
  - Use window command interpreter cmd.exe
  - Add mongoDB binaries to the system path
    - this will help in typing mongodb from command line with no need to put the full path

MongoDB: Install

- Download MongoDB community edition
  - select the platform on which you want to install mongoDB and the package format



Version: 4.0.3 (current release)

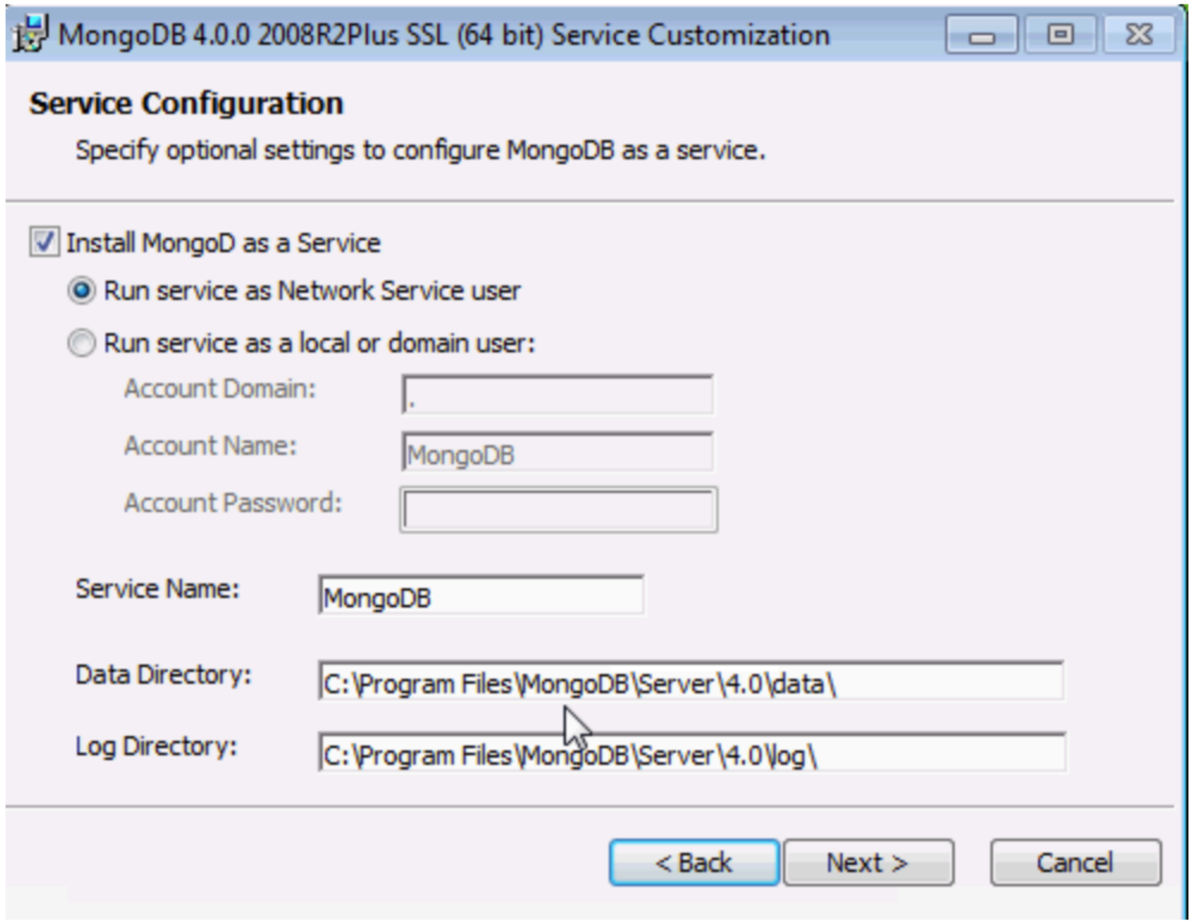
OS: Windows 64-bit x64

Package: ZIP

download page: https://www.mongodb.com/download-center/community?jmp=docs

MongoDB: Install

- Double click on the **.msi** file
- follow the installation wizard
- you can choose custom or complete installation



MongoDB 4.0.0 2008R2Plus SSL (64 bit) Service Customization

Service Configuration

Specify optional settings to configure MongoDB as a service.

☑ Install MongoD as a Service

● Run service as Network Service user

○ Run service as a local or domain user:

Account Domain: .

Account Name: MongoDB

Account Password:

Service Name: MongoDB

Data Directory: C:\Program Files\MongoDB\Server\4.0\data\

Log Directory: C:\Program Files\MongoDB\Server\4.0\log\

MongoDB: Install

- Specify the directory path
  - directory where mongoDB will store collections
- specify the log directory
  - this directory will be used to store the logging

MongoDB: start DB

- from the path where mongoDB is installed
  - run

```
“C:\Program Files\MongoDB\Server\4.0\bin\mongod.exe” --dbpath="c:\data\db"
```

  - - - dbpath points to the DB directory

MongoDB: connect

- Open another command interpreter
- run

```
"C:\Program Files\MongoDB\Server\4.0\bin\mongo.exe"
```

Application Interface

- Mongo shell
    - interactive JavaScript interface to mongoDB

- Drivers for various languages
    - Java, Python, Scala, Ruby, PHP, C, C++, C#

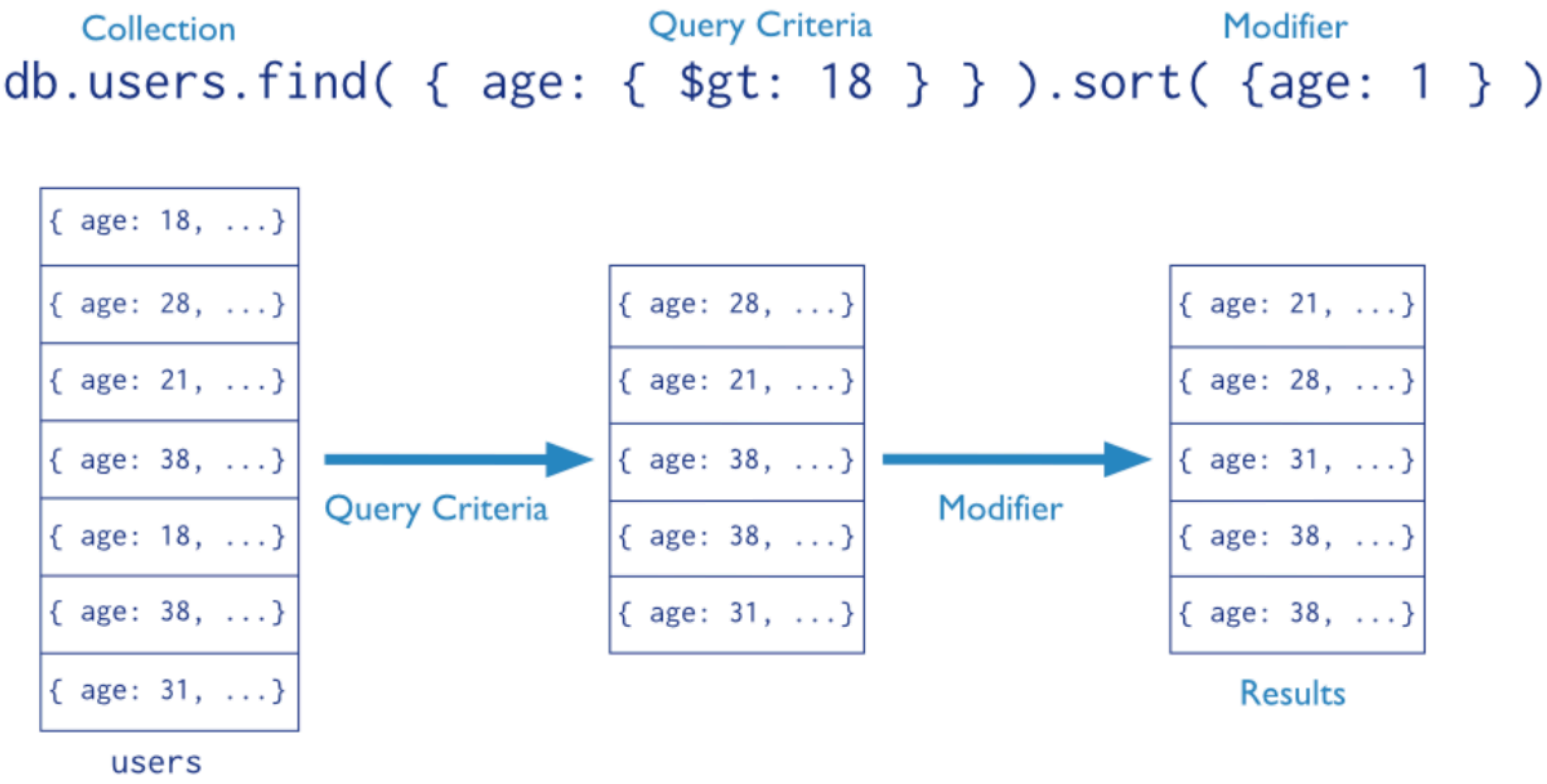
Mongo Query Language

- A mongoDB query
  - targets a specific collection of documents
  - specifies criteria/ condition that identify returned document (selection)
  - May select fields (projection)
  - May impose limit, sort on returned result
- Query syntax: db.collectionName.query
- return all documents
  - db.users.find(), db.users.find({})

CRUD Operations

- CRUD refers to:
  - Create, Read, Update, Delete
- Operations:
  - insert new document
    - db.collection.**insert()**
  - delete an existing document
    - db.collection.**remove()**
  - update an existing document
    - db.collection.**update()**
  - find document(s)
    - db.collection.**find()**

Querying: Example



```
db.users.find( { age: { $gt: 18 } } ).sort( {age: 1 } )
```

users

{ age: 18, ...}
{ age: 28, ...}
{ age: 21, ...}
{ age: 38, ...}
{ age: 18, ...}
{ age: 38, ...}
{ age: 31, ...}

{ age: 28, ...}
{ age: 21, ...}
{ age: 38, ...}
{ age: 38, ...}
{ age: 31, ...}

Results

{ age: 21, ...}
{ age: 28, ...}
{ age: 31, ...}
{ age: 38, ...}
{ age: 38, ...}

Selection

```
db.inventory.find({ type: "snacks" })
```

- All documents from a collection inventory, where type field has the value snacks

```
db.inventory.find({type:{$in:['food','snacks']}})
```

- All documents from a collection inventory, where type field is either snacks or food

```
db.inventory.find({type:’snacks',price:{$lt:9.95}})
```

- All documents from a collection inventory, where type field is snacks and price is <9.95

Inserts

- insert document with three fields
  - the _id is user specified

```
db.inventory.insert({
_id: 10,
type: "misc",
item: "card",
qty: 15})
```

- insert document, the _id is not provided

```
db.inventory.insert({type: "book", item: "journal"})
```

  - it will be generated by the database

```
{ "_id": ObjectId("58e209ecb3e168f1d3915300"),
type: "book", item: "journal" }
```

Update

- Find all documents matching the query

```
{type: "book", item : "journal"}
```

- Sets the field qty to 10

```
{ qty: 10 }
```

- upsert is true
  - then in case of no match
  - create new document
    - contains: _id, type, item, qty

```
db.inventory.update(

 { type: "book", item :

"journal" },

 { $set: { qty: 10 } },

 { upsert: true }  )
```