# Modern Database Concepts

Practicals: Document stores

Doc. RNDr. Irena Holubova, Ph.D.
holubova@ksi.mff.cuni.cz

# mongoDB


- Initial release: 2009
- Written in C++
  - Open-source
- Cross-platform
- JSON documents
  - Dynamic schemas
- Features:
  - High performance – indices
  - High availability – replication + eventual consistency + automatic failover
  - Automatic scaling – automatic sharding across the cluster
  - MapReduce support

http://www.mongoDB.org/

# Mongo Shell

`mongo`

- Run the Mongo shell
- By default it connects to localhost on default port
  - The database should run on our testing server

`help`

- List of basic commands

`exit/quit`

- Terminate the current connection

# Databases

`show databases`

- List the current databases

`use <login>`

- Switch to your database
  - Will be created when you store the first document

`show collections`

- List the collections in the current database

`exit/quit`

- Terminate the current connection

# mongoDB

## Data Insertion

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

- Inserts a document with three fields into collection `inventory`
  - User-specified `_id` field

```
db.inventory.update(
                    { type: "book", item : "journal" },
                    { $set : { qty: 10 } },
                    { upsert : true }
                      )
```

- Creates a new document if no document in the inventory collection contains `{ type: "books", item : "journal" }`
  - mongoDB adds the `_id` field and assigns as its value a unique `ObjectId`
  - The result contains fields `type`, `item`, `qty` with the specified values

# mongoDB

## Data Insertion and Removal

```
db.inventory.save( { type: "book", item:
  "notebook", qty: 40 } )
```

- Creates a new document in collection `inventory` if `_id` is not specified or does not exist in the collection

```
db.inventory.remove( { type : "food" } )
```

- Removes all documents that have `type` equal to `food` from the `inventory` collection

```
db.inventory.remove( { type : "food" }, 1 )
```

- Removes one document that has `type` equal to `food` from the `inventory` collection

# mongoDB

## Data Updates

```
db.inventory.update(
                    { type : "book" },
                    { $inc : { qty : -1 } },
                    { multi: true }
                  )
```

- Finds all documents with `type` equal to `book` and modifies their `qty` field by `-1`

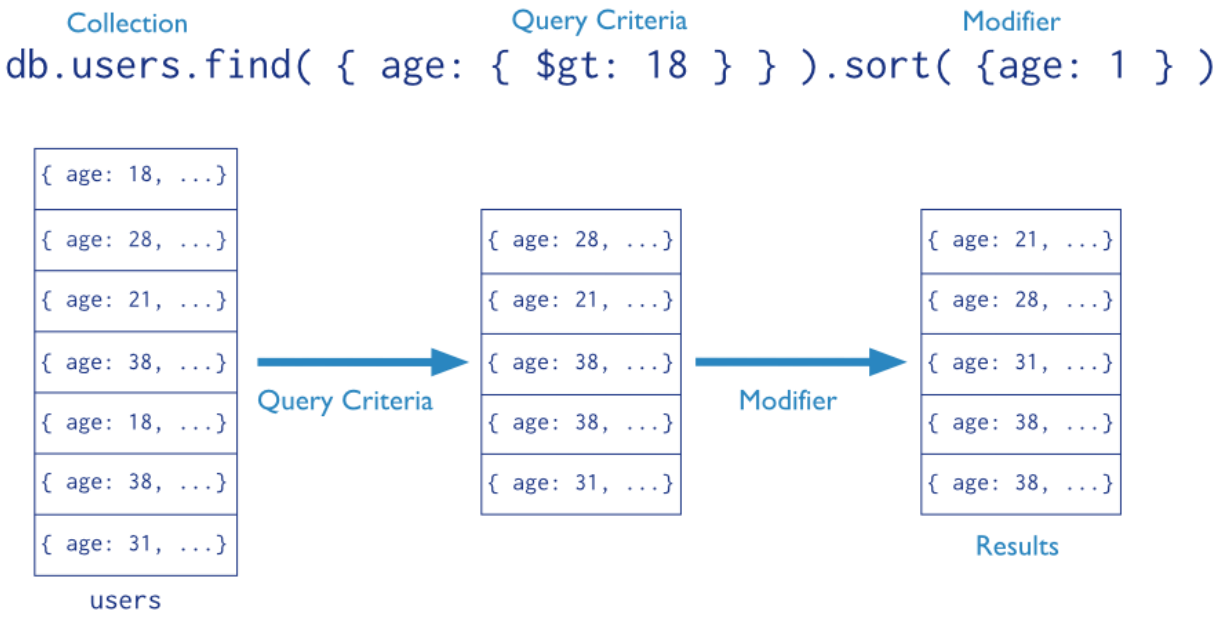
```
db.inventory.save(
    {
      _id: 10,
      type: "misc",
      item: "placard"
    } )
```

- Replaces document with `_id` equal to `10`

# mongoDB

## Query

- Targets a specific collection of documents
- Specifies criteria that identify the returned documents
- May include a projection that specifies the fields from the matching documents to return
- May impose limits, sort orders, …



# mongoDB

## Query – Basic Queries, Logical Operators

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

- All documents in the collection

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

- All documents where the `type` field has the value `snacks`

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

- All documents where value of the `type` field is either `food` or `snacks`

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

- All documents where the `type` field has the value `food` **and** the `value` of the `price` field is less than (`$lt`) `9.95`

# mongoDB

## Query – Logical Operators

```
db.inventory.find(
                    { $or: [
                              { qty: { $gt: 100 } },
                              { price: { $lt: 9.95 } }
                            ] } )
```

- All documents where the field `qty` has a value greater than (`$gt`) `100` **or** the value of the `price` field is less than `9.95`

```
db.inventory.find( { type: 'food', $or: [
                     { qty: { $gt: 100 } },
                     { price: { $lt: 9.95 } } ]
                   } )
```

- All documents where the value of the `type` field is `food` **and** either the `qty` has a value greater than (`$gt`) `100` **or** the value of the `price` field is less than `9.95`

# mongoDB

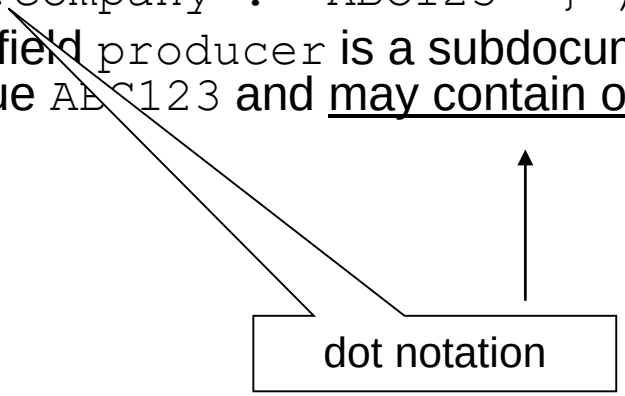
## Query – Subdocuments

```
db.inventory.find( {
                     producer: {
                                  company: 'ABC123',
                                  address: '123 Street'
                                }
                   } )
```

- All documents where the value of the field `producer` is a subdocument that contains only the field `company` with the value `ABC123` and the field `address` with the value `123 Street`, in the exact order

```
db.inventory.find( { 'producer.company': 'ABC123' } )
```

- All documents where the value of the field `producer` is a subdocument that contains a field `company` with the value `ABC123` and may contain other fields



# mongoDB

## Query – Arrays

exact match

```
db.inventory.find( { tags: [ 'fruit', 'food',
   'citrus' ] } )
```

- All documents where the value of the field `tags` is an array that holds exactly three elements, `fruit`, `food`, and `citrus`, in this order

```
db.inventory.find( { tags: 'fruit' } )
```

- All documents where  value of the field `tags` is an array that contains `fruit` as one of its elements

```
db.inventory.find( { 'tags.0' : 'fruit' } )
```

- All documents where the value of the `tags` field is an array whose first element equals `fruit`

# mongoDB

## Query – Arrays of Subdocuments

```
db.inventory.find( { 'memos.0.by': 'shipping' } )
```

- All documents where the `memos` field contains an array whose first element is a subdocument with the field `by` with the value `shipping`

```
db.inventory.find( { 'memos.by': 'shipping' } )
```

- All documents where the `memos` field contains an array that contains at least one subdocument with the field `by` with the value `shipping`

```
db.inventory.find({
                    'memos.memo': 'on time',
                    'memos.by': 'shipping'
                  })
```

- All documents where the value of the `memos` field is an array that has at least one subdocument that contains the field `memo` equal to `on time` and the field `by` equal to `shipping`

# mongoDB

## Query – Limit Fields of the Result

or true

```
db.inventory.find( { type: 'food' }, { item: 1, qty: 1 } )
```

- Only the `item` and `qty` fields (and by default the `_id` field) return in the matching documents

```
db.inventory.find( { type: 'food' }, { item: 1, qty: 1, _id: 0 } )
```

- Only the `item` and `qty` fields return in the matching documents

```
db.inventory.find( { type: 'food' }, { type : 0 } )
```

- The `type` field does not return in the matching documents

or false

- Note: With the exception of the `_id` field we cannot combine inclusion and exclusion statements in projection documents.

# mongoDB

## Query – Sorting

```
db.collection.find().sort( { age: -1 } )
```

- Returns all documents in `collection` sorted by the `age` field in descending order

```
db.bios.find().sort( { 'name.last': 1,
  'name.first': 1 } )
```

- Specifies the sort order using the fields from a sub-document `name`
- Sorts first by the `last` field and then by the `first` field in ascending order

# JSON Sample Data Generator

- http://jsongen.pykaso.net/

```
{
        "_id" : "%index%",
        "surname" : "%surname%",
        "fullname" : "%fullname%",
        "email" : "%email%",
        "holiday" : "%bool%",
        "salary": "%randFloat(10,20)%",
        "address" : {
                "city" : "%name%",
                "street" : "%name%",
                "number" : "%randInt(0,50)%",
        },
}
```

# Data Import

- `mongoimport --db <login> --collection people --jsonArray --stopOnError < /home/NOSQL/MFF-NDBI040/document/simpleData.txt`
  - database: `<login>`
  - collection: `people`
  - file type: JSON
    - assuming that the file contains a list of documents
  - file path: `simpleData.txt`

  - Note: First exit `mongo` or open a new shell window - `mongoimport` is another tool. Then run `mongo` again to be able to query the data.

# Simple Example

## Indexing

- Import data from `simpleData.txt`
  - `db.people.insert([ {…}, {…}, …])`

```
db.people.find({ "salary" : { $lt  : 10.1 }})

db.people.find({ "salary" : { $lt  : 10.1
   }}).explain()

db.people.ensureIndex( {"salary" : 1 } )

db.people.find({ "salary" : { $lt  : 10.1
   }}).explain()
```

  - Add an index and test the amount again

# Assignment

- Chose your unique problem domain
  - E.g., the results of football matches of various teams
- For your selected problem domain, think about an application that uses MongoDB collections (create/generate data, store them in MongoDB, use indexes, create meaningful queries)
- Submit a script with respective commands for MongoDB + explanatory comments

# References

- MongoDB:
  - Shell methods: http://docs.mongodb.org/manual/reference/method/
  - Indexes: http://docs.mongodb.org/manual/indexes/
  - mongoimport: http://docs.mongodb.org/v2.2/reference/mongoimport/