Chapter 2.
Creating and Manipulating Database

References

- MongoDB The Definitive Guide: Powerful and Scalable Data Storage 3rd Edition
- https://docs.mongodb.com/
- https://www.mongodb.com/docs/manual/

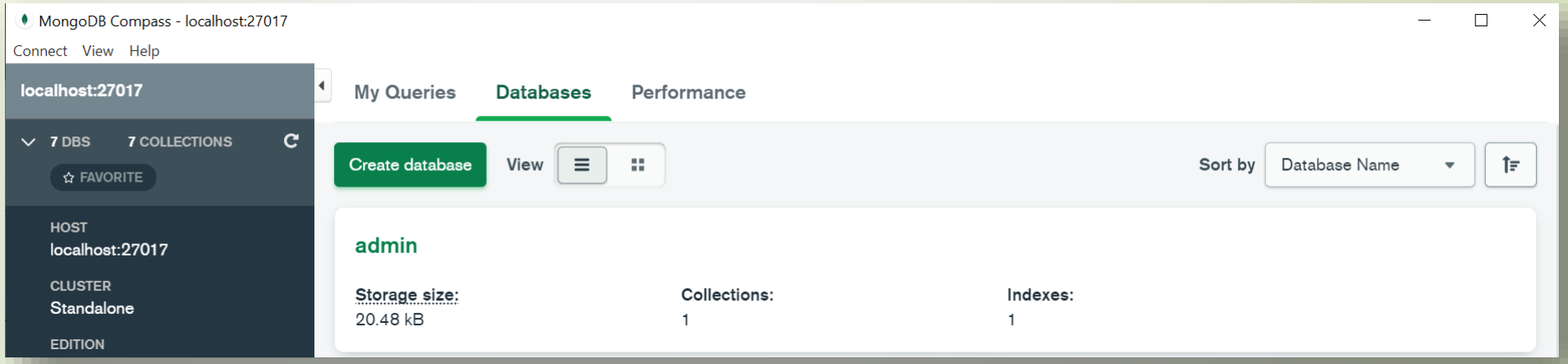
Learning objectives

- Create databases, collections by MongoDB Compass/ MongoDB Shell/ MongoDB Atlas
- Recognize the structure of JSON documents
- Import, export databases

Contents

1. Create Databases Using the MongoDB Compass

- Creating a MongoDB Database with Compass
  - In MongoDB Compass, click “Create Database” button on the Databases tab.



  - Enter the name of the database and its first collection.
  - Click Create Database.

1. Create Databases Using the MongoDB Shell

- Syntax: **use** <DatabaseName>
- Example:

```
---
MongoDB Enterprise > show dbs;
admin    0.000GB
config   0.000GB
local    0.000GB
mydb     0.000GB
MongoDB Enterprise > use QuanlySV
switched to db QuanlySV
```

1. Create Databases Using the MongoDB Atlas UI

- Creating a MongoDB Database with the Atlas UI
  - From your cluster page, click on “Browse Collections”.
  - If there are no databases in this cluster, you will be presented with the option to create your first database by clicking on the “Add My Own Data” button.
  - This will open up a modal, asking you for a database name and collection name. Once these fields are filled, click on “Create” and your database will be created for you.

Contents

2. Create Collections
Explicit creation

- Mainly used in case there is need to create special collections like capped collections or collections with document validation rules.

- Syntax: **db.createCollection(**<collection_name>, <options>**)**
  - *options* is a document, specify options about memory size and indexing
- Example: **db.createCollection**(“customers“)

2. Create Collections
Implicit creation

- You can create a collection implicitly by simply **adding a document to a non-existent collection** and the collection is created if it doesn't already exist.
  - Syntax: **db.**<collection_name>**.insert(**<document>**)**
  - Example: **db**.customers.**insert**({ name: "Honey", age: 25})
- The collection is created automatically when data is imported from the application (see the next content).

Contents

1. Create databases
2. Create collections
3. **Document validation**
4. Import – Export databases

3. Document validation

- Document validation provides significant flexibility to customize which parts of the documents are and are not validated for any collection. For any key it might be appropriate to check:
  - That a key exists
  - If a key does exist, is it of the correct type
  - That the value is in a particular format (regular expressions can be used to check if the contents of the string matches a particular pattern)
  - That the value falls within a given range

3. Document validation

- To specify validation rules when creating a new collection, use db.createCollection() with the *validatior* option.

- To add document validation to an existing collection, use *collMod* command with the *validator* option

3. Document validation

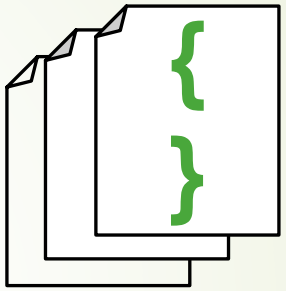
- Example: the following snippet adds validations to the contacts collection that validates:
  - The year of birth is no later than 1994
  - The document contains a phone number and/or an email address
  - When present, the phone number and email address are strings

```
db.runCommand({
   collMod: "contacts",
   validator: {
      $and: [
        {yearOfBirth: {$lte: 1994}},
        {$or: [
                  {"contact.phone": { $type: "string"}},
                  {"email": { $type: "string"}}
              ]}]
    }})
```

Contents

4. Import – Export databases
How does MongoDB store data?

BSON
MongoDB stores data in BSON, internally and over the network



JSON
Can be natively stored and retrieved in MongoDB

4. Import – Export databases
JSON – BSON

	JSON - JavaScript Object Notation	BSON - Binary JSON
Encoding	UTF-8 String	Binary
Data Support	String, Boolean, Number, Array, Object, null	String, Boolean, Number (Integer, Float, Long, Decimal128...), Array, null, Date, BinData
Readability	Human and Machine	Machine Only

4. Import – Export databases JSON

- JSON format:
  - Start and end with { }
  - Separate each field and value with colon :
  - Separate each field : value pair with comma ,
  - Field must be surrounded by quotation mark “ ”

Name

Value

```
{
    "id" : 14,
    "firstName" : "Mario",
    "lastName" : "Rossi",
    "active" : true
}
```

4. Import – Export databases
JSON Values

- Numbers: no quotes
- String: in double quotes
- Boolean: true, false
- Null
- Nested JSON object: It is a collection of field-value pairs and always separated by a comma and enclosed in curly brackets.
- Array: It is an ordered sequence of values separated

4. Import – Export databases JSON Values

- Example

```
{
  "id": 14,
  "firstName": "Mario",
  "lastName": "Rossi",
  "active": true,
  "address" : {
                "street" : "100 Main St",
                "city" : "Philadelphia",
                "state" : "Pennsylvania",
                "zip" : "19103",
                "country" : "USA"
              }
}
```

Nested JSON object

```
{
  "id": 14,
  "firstName": "Mario",
  "lastName": "Rossi",
  "active": true,
  "languages" : ["Java", "C#", "Python", "Javascript"]
}
```

JSON Array

4. Import – Export databases JSON Values

- Example

```
{
  "_id": 1,
  "name" : { "first" : "John", "last" : "Backus" },
  "contribs" : [ "Fortran", "ALGOL", "Backus–Naur Form", "FP" ],
  "awards" : [
    {
      "award" : "W.W. McDowell Award",
      "year" : 1967,
      "by" : "IEEE Computer Society"
    }, {
      "award" : "Draper Prize",
      "year" : 1993,
      "by" : "National Academy of Engineering"
    }
  ]
}
```

4. Import – Export databases
Pros – Cons of JSON

Pros of JSON	Cons of JSON
Friendly	Text–based
Readable	Space-Consuming
Familiar	Limited of data types

4. Import – Export databases
BSON

- BSON is a binary-encoded serialization of JSON documents.
- Example:

```
{"hello": "world"} →
\x16\x00\x00\x00            // total document size
\x02                        // 0x02 = type String
hello\x00                   // field name
\x06\x00\x00\x00world\x00   // field value
\x00                        // 0x00 = type EOO ('end of object')
```

- Optimized for:
  - Speed
  - Space
  - Flexibility to achieve high performance

4. Import – Export databases

Export to a local machine

Import from a different system

Export to a different system

Import from a local machine

Backup cloud data locally

4. Import – Export databases

- Data is stored in BSON but viewed in JSON.

  → Which format we're going to use?

JSON	BSON
mongoexport	mongodump
mongoimport	mongorestore

4. Import – Export databases
Export: BSON

- Syntax: **mongodump** <options>
  - <options>:
    - -d/--database: database to use
    - -c/--collection: collection to use
    - -u/--username: username for authentication
    - -p/--password: password for authentication
    - -o/--out: output directory
    - -h/--host: mongodb host to connect
    - -p/--port: server port (can also use –h hostname:port)
    - …

4. Import – Export databases
Export: BSON

- Example:

1. From localhost:

mongodump -d *dbtest* -c *restaurants* -o *backup*

connect to a local MongoDB instance running on port 27017 and use the default settings to export the content (no parameters with all databases and collections)

2. From Atlas:

mongodump --uri *mongodb+srv://mongobasic:pass@cluster0.msr5i.mongodb.net/sample_restaurants* -c *restaurants* –o *backup*

creates a dump file that contains only the collection named restaurants of sample_restaurants database

4. Import – Export databases
Import: BSON

- Syntax: **mongorestore** <options> <directory or file to restore>
- Example:

1. From localhost:

mongorestore -d *backup* -c *restaurants* *backup/dbtest/restaurants.bson*

restores the collection named restaurants in the database backup from the corresponding files located in the backup/dbtest/restaurants.bson

2. From Atlas:

mongorestore --uri *mongodb+srv://mongobasic:pass@cluster0.msr5i.mongodb.net* -d *backup* -c *restaurants* *backup/dbtest/restaurants.bson*

restore from a backup/dbtest/ directory to MongoDB Atlas Cluster.

4. Import – Export databases
Export: JSON

- Syntax: **mongoexport** <options>
- Example:

1. From localhost:

mongoexport -d *dbtest* -c *restaurants* -o *res.json*

export the restaurants collection of dbtest db to the res.json output file from a local MongoDB instance running on port 27017.

2. From Atlas:

mongoexport --uri *mongodb+srv://mongobasic:pass@cluster0.msr5i.mongodb.net/sample_restaurants* -c *restaurants* -o *res.json*

connect to a MongoDB Atlas Cluster and export the restaurants collection of sample_restaurant db to res.json output file.

4. Import – Export databases
Import: JSON

- Syntax: **mongoimport** <options> <file to import>

1. From localhost:

Mongoimport --drop -d *backup* -c *restaurants* *res.json*

Mongoimport -h *localhost:27017* --drop -d *backup* -c *restaurants* *res.json*

import the json data form the res.json file into the collection restaurants in the backup db to a local mongod instance running on port 27017.

--drop: before restoring the collections from the dumped backup, drops the collections from the target database. does not drop collections that are not in the backup.

2. From Atlas:

mongoimport --uri
mongodb+srv://mongobasic:pass@cluster0.msr5i.mongodb.net/backup
--drop -c *restaurants* res.json
Import from the res.json file to MongoDB Atlas Cluster.

Question?