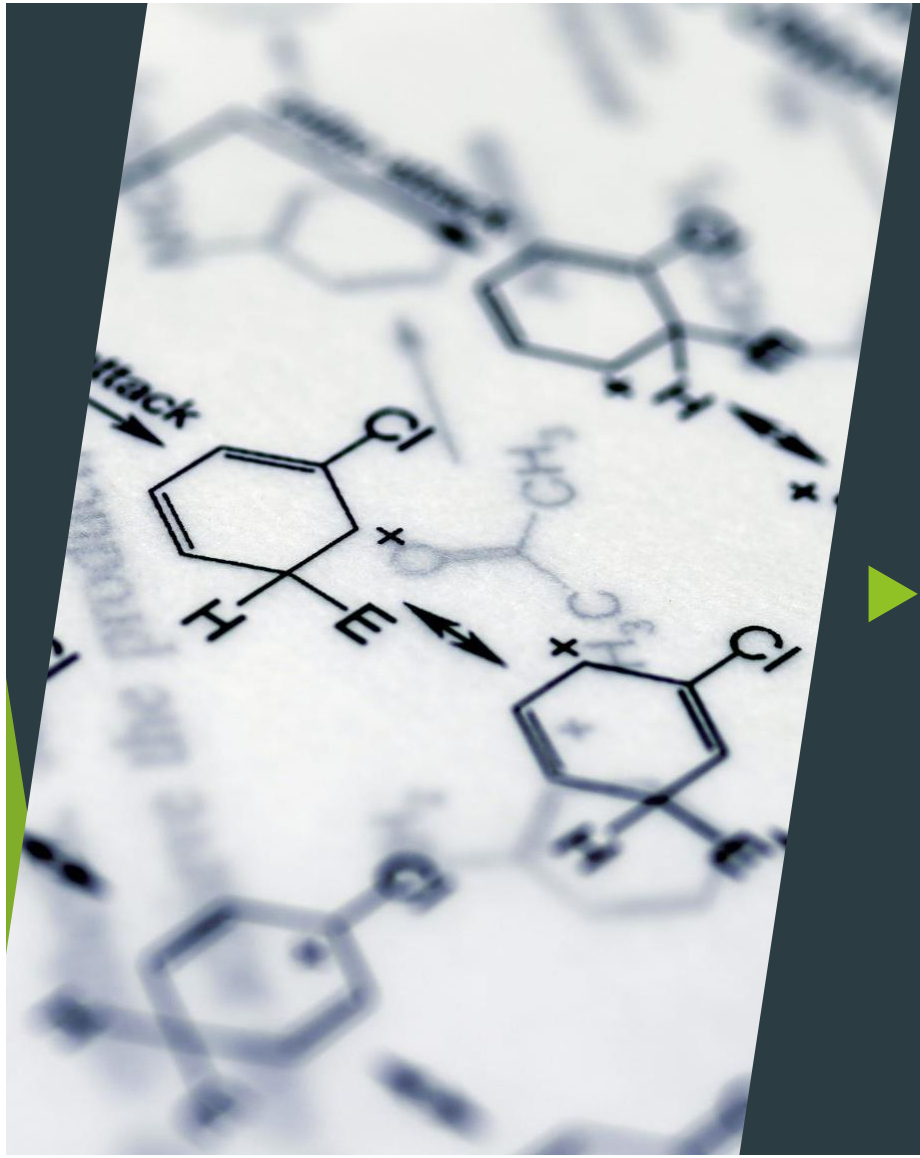


# MongoDB – Basic Operations

1

# Module Objectives

## What you will learn

At the end of this module, you will learn:

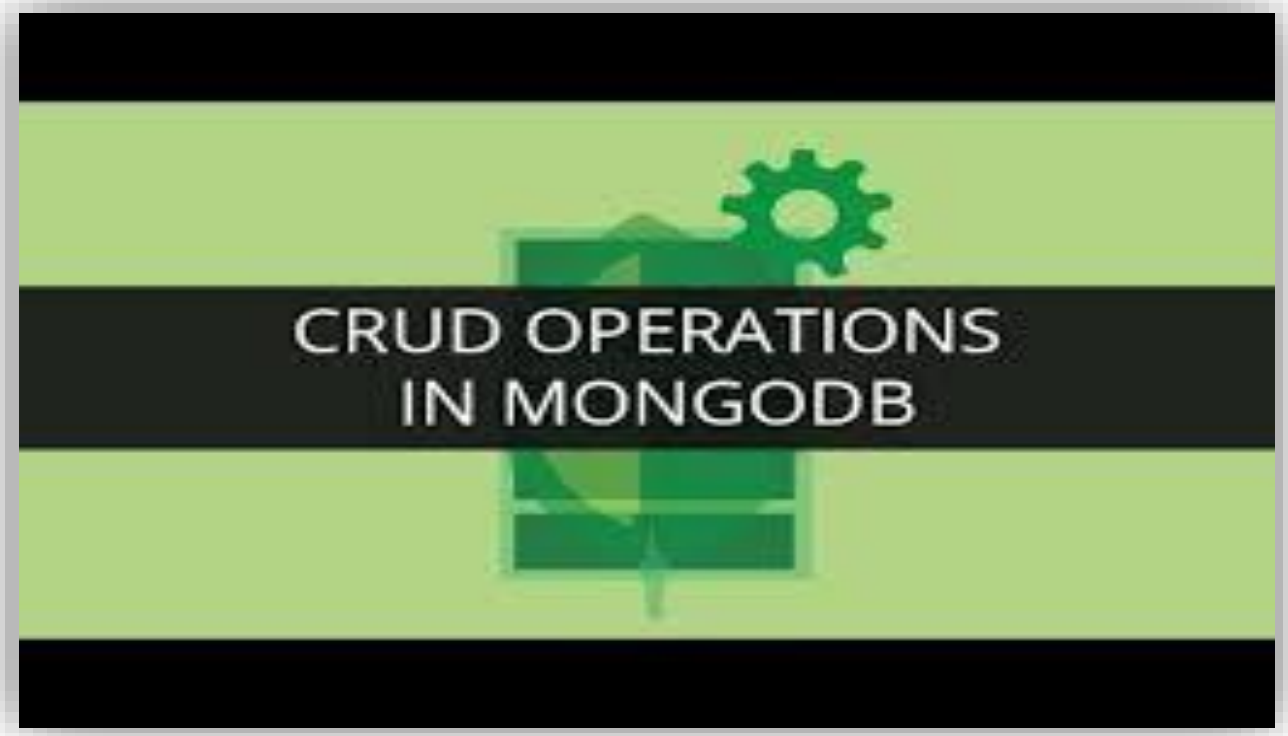
- The Basic Operations of MongoDB

## What you will be able to do

At the end of this module, you be able to:

- Understand the Basic Operations of MongoDB
- Describe the Data Model
- State the features of JSON and BSON
- List the BSON Types
- Explain the features of Document

# Crud Operations






3

# Basic Operations with Mongo Shell

- Create Database
- Drop Database
- Create Collection
- Drop Collection
- JSON, BSON Document
- Datatypes



4

# MongoDB - Commands

To check your currently selected database use the command **db**.

- **> db**
- mydb

If you want to check your databases list, then use the command.

- **> show dbs**

To display the currently created database , you need to insert one document into it.

- db.movie.insert({"name":"tutorials point"})

- show dbs
- local 0.78125GB
- mydb 0.23012GB

# MongoDB - Create Database

MongoDB **use DATABASE_NAME** is used to create database on the fly at the time you use it.

The command will create a new database, if it doesn't exist otherwise it will return the existing database.

Basic syntax of **use DATABASE** statement is as follows:

use DATABASE_NAME

- Example:
  - > use mydb
  - switched to db mydb

# MongoDB - dropDatabase()

MongoDB **db.dropDatabase()** command is used to drop a existing database.

Basic syntax of **dropDatabase()** command is as follows:

- db.dropDatabase()

To delete new database **<mydb>**, then **dropDatabase()** command would be as follows:

- >use mydb
  - >switched to db mydb >db.dropDatabase()
  - >{ "dropped" : "mydb", "ok" : 1 }

# MongoDB – createCollection() method

MongoDB **db.createCollection(name, options)** is used to create collection. Basic syntax of **createCollection()** command is as follows:

- db.createCollection(name, options)

**name** is name of collection to be created. **Options** is a document and used to specify configuration of collection.

db.createCollection("mycollection").

The syntax of **createCollection()** method with few important options:

- db.createCollection("mycol", { capped : true, autoIndexID : true, size : 6142800, max : 10000 } )

# MongoDB - The drop() method

Basic syntax of drop() command is:

- db.COLLECTION_NAME.drop()

drop() method will return true, if the selected collection is dropped successfully otherwise it will return false.

# Data Model

Document-Based (max 16 MB)

Documents are in BSON format, consisting of field-value pairs. Each document stored in a collection.

Collections

- Have index set in common.
- Like tables of relational db's.
- Documents do not have to have uniform structure.



10

# JSON

"JavaScript Object Notation"

Easy for humans to write / read, easy for computers to parse / generate.

Objects can be nested.

Built on:

- Name / value pairs
- Ordered list of values

# BSON

"Binary JSON"

Binary-encoded serialization of JSON-like docs.

Also allows "referencing".

Embedded structure reduces need for joins.

Goals

- Lightweight
- Traversable
- Efficient (decoding and encoding)



12

# BSON Example

```
{
"_id" :          "37010"
"city" :         "ADAMS",
"pop" :          2660,
"state" :        "TN",
“councilman” : {
                    name: “John Smith”
                    address: “13 Scenic Way”
                  }
}
```



13

# MongoDB - Datatypes

| | |
|---|---|
| **String** | • This is most commonly used datatype to store the data. String in mongodb must be UTF-8 valid. |
| **Integer** | • This type is used to store a numerical value. Integer can be 32 bit or 64 bit depending upon your server. |
| **Boolean** | • This type is used to store a boolean (true / false) value. |
| **Double** | • This type is used to store floating point values. |
| **Min / Max Keys** | • This type is used to compare a value against the lowest and highest BSON elements. |



14

# MongoDB - Datatypes (contd.)

| | |
|---|---|
| **Arrays** | • This type is used to store arrays or list or multiple values into one key. |
| **Timestamp** | • **ctimestamp**. This can be handy for recording when a document has been modified or added. |
| **Object** | • This datatype is used for embedded documents. |
| **Null** | • This type is used to store a Null value. |
| **Symbol** | • This datatype is used identically to a string however, it's generally reserved for languages that use a specific symbol type. |



15

# MongoDB - Datatypes (contd.)

| | |
|---|---|
| **Object _id** | • This datatype is used to store the document's ID. |
| **Binary Data** | • This datatype is used to store binary data. |
| **Code** | • This datatype is used to store javascript code into document. |
| **Regular Expression** | • This datatype is used to store regular expression. |
| **Date** | • This datatype is used to store the current date or time in UNIX time format. You can specify your own date time by creating object of Date and passing day, month, year into it. |



16

# BSON Types

| Type | Number |
|---|---|
| Double | 1 |
| String | |
| Object | |
| Array | |
| Binary data | 5 |
| Object id | 7 |
| Boolean | 8 |
| Date | 9 |
| Null | 10 |
| Regular Expression | 11 |
| JavaScript | 13 |
| Symbol | 14 |
| JavaScript (with scope) | 15 |
| 32-bit integer | 16 |
| Timestamp | 17 |
| 64-bit integer | 18 |
| Min key | 255 |

The number can be used with the $type operator to query by type!



17

# The _id Field

- By default, each document contains an _id field. This field has a number of special characteristics:

  - Value serves as primary key for collection.
  - Value is unique, immutable, and may be any non-array type.
  - Default data type is ObjectId, which is “small, likely unique, fast to generate, and ordered.” Sorting on an ObjectId value is roughly equivalent to sorting on creation time.



18

# Example: Mongo Collection

```
{ "_id": ObjectId("4efa8d2b7d284dad101e4bc9"),
    "Last Name": "DUMONT",
    "First Name": "Jean",
    "Date of Birth": "01-22-1963" },
    { "_id": ObjectId("4efa8d2b7d284dad101e4bc7"),
      "Last Name": "PELLERIN",
      "First Name": "Franck",
      "Date of Birth": "09-19-1983",
  "Address": "1 chemin des Loges", "City": "VERSAILLES" }
```

# Example: Mongo Document

```
user = {
        name: “Z",
        occupation: "A scientist",
        location: “New York"
                }
```



20

# Document

**Simple Document**

A document is roughly equivalent to a rowin a relational database, which contain one or multiple key-value pairs.

- {"greeting" : "Hello, world!"}

Most documents will be more complex than this simple one and often will contain multiple key/value pairs:

- {"greeting" : "Hello, world!", "foo" : 3}

Key/value pairs in documents are ordered—the earlier document is distinct from the following document:

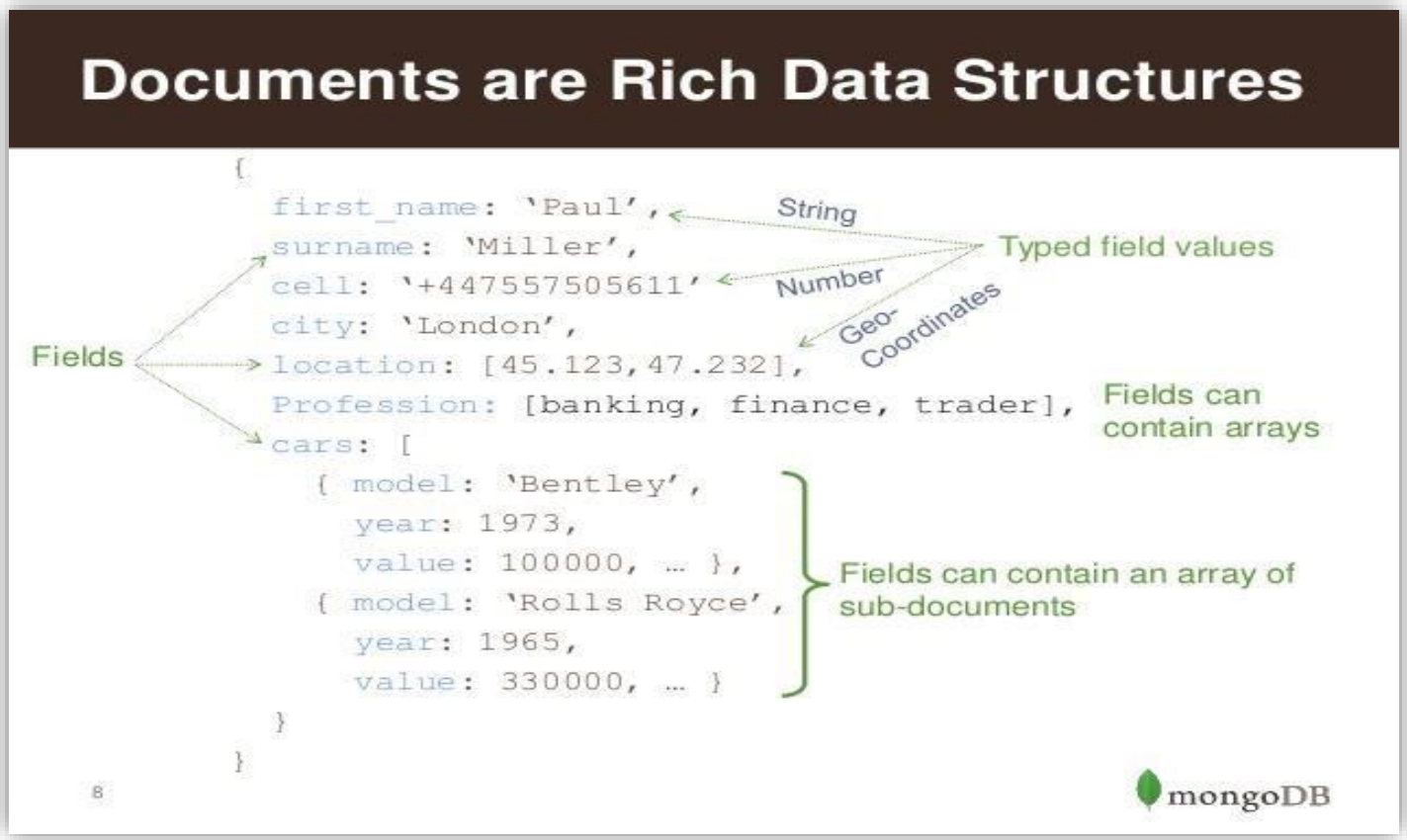
- {"foo" : 3, "greeting" : "Hello, world!"}

Values in documents are not just “blobs.” They can be one of several different data types.



21

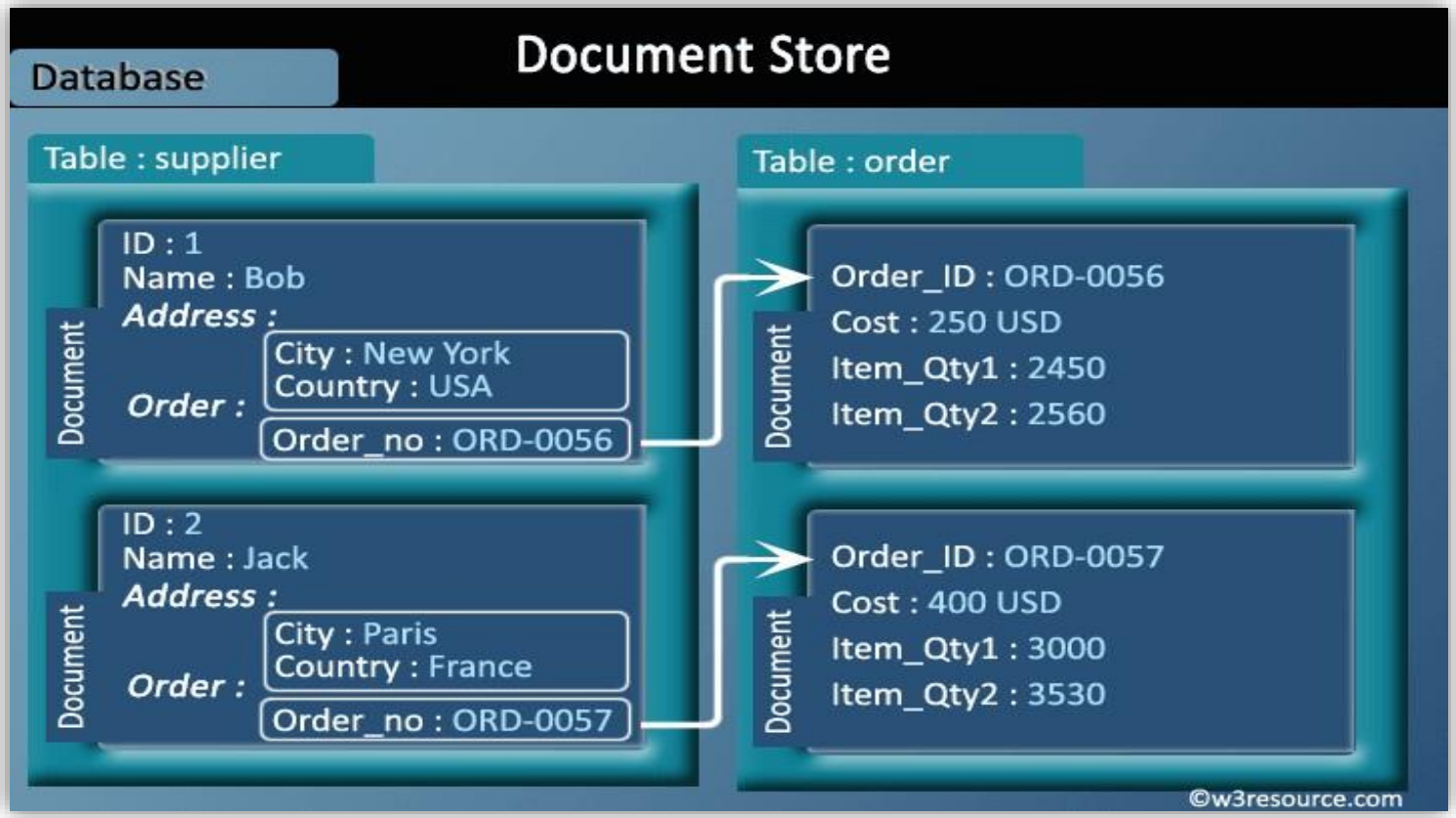
# Document (contd.)






22

# Document Store






23

# Document Store (contd.)

**Scenario**

- A blog post has an author, some text, and many comments.
- The comments are unique per post, but one author has many posts.
- How would you design this in SQL?



24

# Example: A Blog: Bad Design

Collections for posts, authors, and comments.

References by manually created ID.



25

# Example: A Blog: Bad Design (contd.)

```
post = {
  id: 150,
 author: 100,
 text: 'This is a pretty awesome post.',
comments: [100, 105, 112]
  }
 author = {
 id: 100,
name: 'Michael Arrington'
posts: [150]
}
comment = {
id: 105,
text: 'Whatever this sux.‘
}
```



26

# Example: Blog - A Better Design

Collection for Posts

```
post = {
      author: 'Michael Arrington',
          text: 'This is a pretty awesome post.',
           comments: [
           'Whatever this post .',
             'I agree, lame!'
            ]
           }
```

Why is this one better?



27

# Benefits

Embedded Objects  brought back in the same query as the parent  Object.

Only 1 trip to the DB server required.

Objects in the same collection are generally stored contiguously  on disk.

Spatial locality = faster

If the document model matches your domain well ,it can be much easier comprehend the nasty joins.



28

# Summary

| | | |
|---|---|---|
| Create Database | Drop Database | Create Collection |
| Drop Collection | CRUD Operations | JSON, BSON Document |
| | Data Types | |



29