

DS 4300

# Document Databases & MongoDB

Mark Fontenot, PhD  
Northeastern University

# Document Database

A **Document Database** is a non-relational database that stores data as structured documents, usually in JSON.

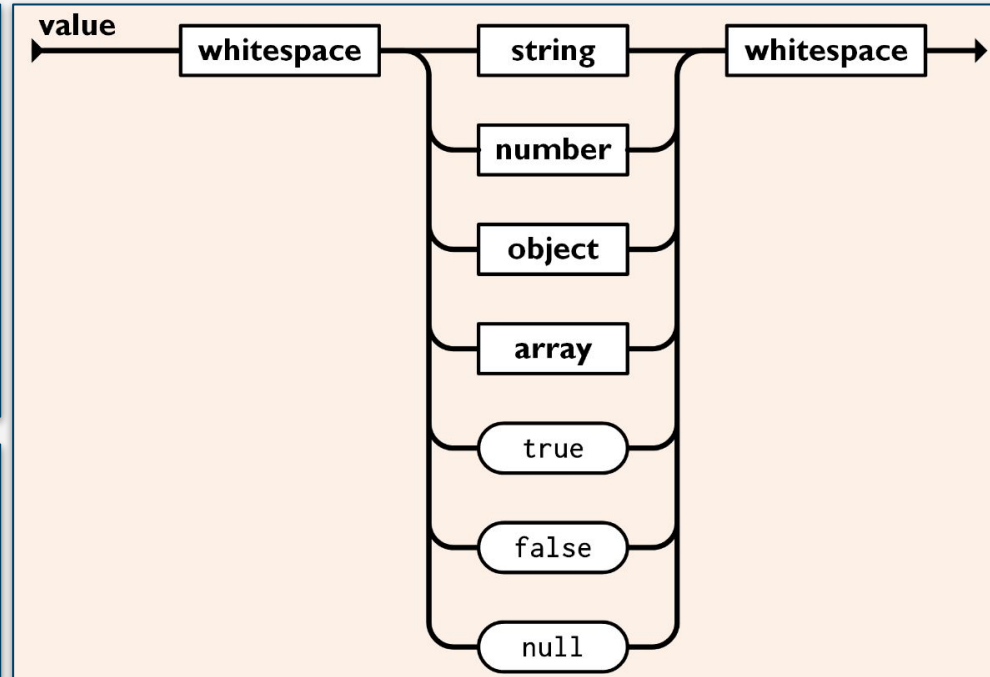
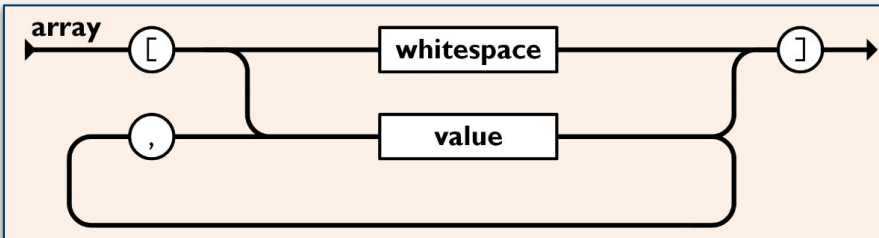
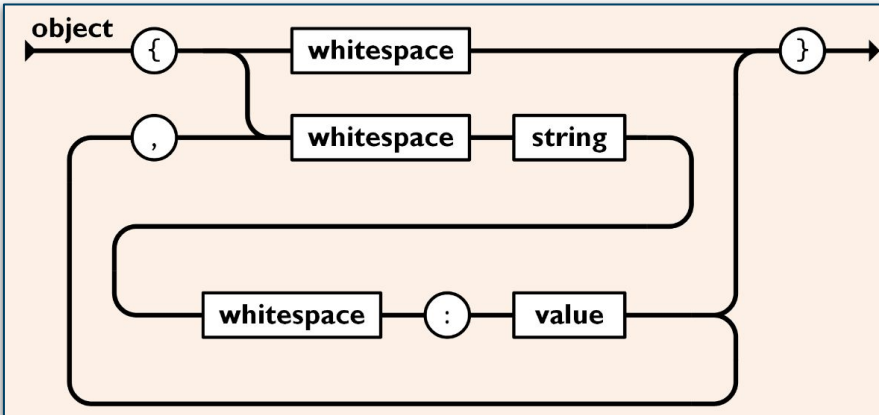
They are designed to be *simple, flexible, and scalable*.

```
{
  "orders": [
    {
      "orderno": "748745375",
      "date": "June 30, 2088 1:54:23 AM",
      "trackingno": "TN0039291",
      "custid": "11045",
      "customer": [
        {
          "custid": "11045",
          "fname": "Sue",
          "lname": "Hatfield",
          "address": "1409 Silver Street",
          "city": "Ashland",
          "state": "NE",
          "zip": "68003"
        }
      ]
    }
  ]
}
```

# What is JSON?

- **JSON** (JavaScript Object Notation)
  - a lightweight data-interchange format
  - It is easy for humans to read and write.
  - It is easy for machines to parse and generate.
- JSON is built on two structures:
  - A **collection of name/value pairs**. In various languages, this is operationalized as an object, record, struct, dictionary, hash table, keyed list, or associative array.
  - An **ordered list of values**. In most languages, this is operationalized as an array, vector, list, or sequence.
- These are two universal data structures supported by virtually all modern programming languages
  - Thus, JSON makes a great data interchange format.

# JSON Syntax



# Binary JSON? BSON

- BSON → Binary JSON
  - binary-encoded serialization of a JSON-like document structure
  - supports extended types not part of basic JSON (e.g. Date, BinaryData, etc)
  - **Lightweight** - keep space overhead to a minimum
  - **Traversable** - designed to be easily traversed, which is vitally important to a document DB
  - **Efficient** - encoding and decoding *must* be efficient
  - Supported by many modern programming languages

```
{"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 E00 ('end of object')
```

# XML (eXtensible Markup Language)

- Precursor to JSON as data exchange format
- XML + CSS → web pages that separated content and formatting
- Structurally similar to HTML, but tag set is extensible

```
<CATALOG>
  <CD>
    <TITLE>Empire Burlesque</TITLE>
    <ARTIST>Bob Dylan</ARTIST>
    <COUNTRY>USA</COUNTRY>
    <COMPANY>Columbia</COMPANY>
    <PRICE>10.90</PRICE>
    <YEAR>1985</YEAR>
  </CD>
  <CD>
    <TITLE>Hide your heart</TITLE>
    <ARTIST>Bonnie Tyler</ARTIST>
    <COUNTRY>UK</COUNTRY>
    <COMPANY>CBS Records</COMPANY>
    <PRICE>9.90</PRICE>
    <YEAR>1988</YEAR>
  </CD>
</CATALOG>
```

# XML-Related Tools/Technologies

- **Xpath** - a syntax for retrieving specific elements from an XML doc
- **Xquery** - a query language for *interrogating* XML documents; the *SQL* of *XML*
- **DTD** - Document Type Definition - a language for describing the allowed structure of an XML document
- **XSLT** - eXtensible Stylesheet Language Transformation - tool to transform XML into other formats, including non-XML formats such as HTML.

# Why Document Databases?

- Document databases address the *impedance mismatch* problem between object persistence in OO systems and how relational DBs structure data.
  - OO Programming → Inheritance and Composition of types.
  - How do we save a complex object to a relational database?  
*We basically have to deconstruct it.*
- The structure of a document is *self-describing*.
- They are well-aligned with apps that use JSON/XML as a transport layer



# MongoDB

# MongoDB

- Started in 2007 after Doubleclick was acquired by Google, and 3 of its veterans realized the limitations of relational databases for serving > 400,000 ads per second
- MongoDB was short for *Humongous Database*
- MongoDB Atlas released in 2016 → documentdb as a service

# MongoDB Structure

## Database

### Collection A

Document 1

Document 2

Document 3

### Collection B

Document 1

Document 2

Document 3

### Collection C

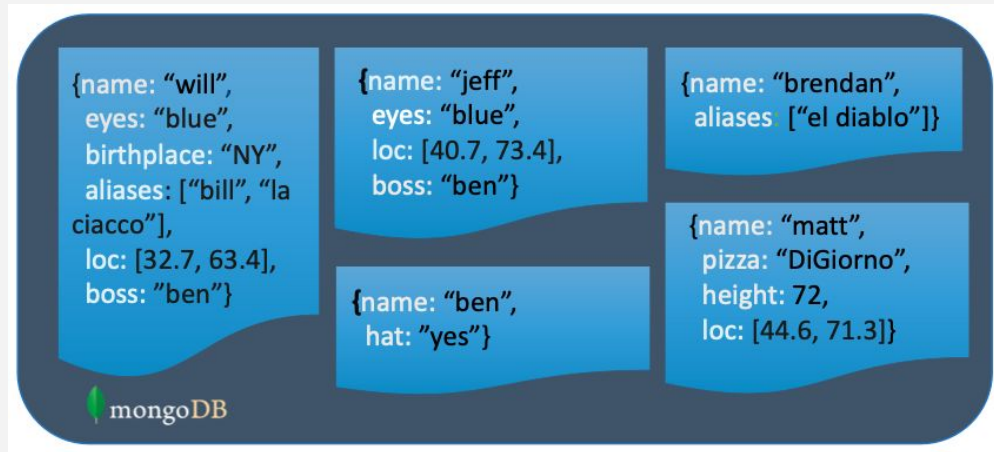
Document 1

Document 2

Document 3

# MongoDB Documents

- No predefined schema for documents is needed
- Every document in a collection could have different data/schema



# Relational vs Mongo/Document DB

RDBMS	MongoDB
Database	Database
Table/View	Collection
Row	Document
Column	Field
Index	Index
Join	Embedded Document
Foreign Key	Reference

# MongoDB Features

- Rich Query Support - robust support for all CRUD ops
- Indexing - supports primary and secondary indices on document fields
- Replication - supports replica sets with automatic failover
- Load balancing built in

# MongoDB Versions

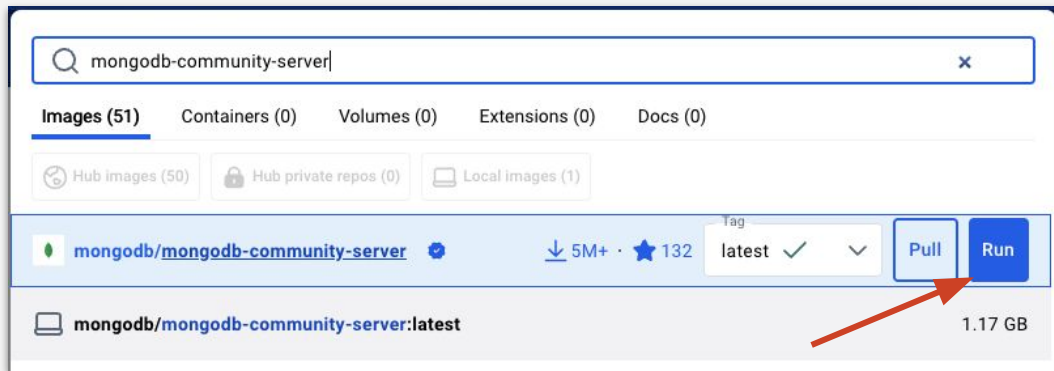
- MongoDB Atlas
  - Fully managed MongoDB service in the cloud (DBaaS)
- MongoDB Enterprise
  - Subscription-based, self-managed version of MongoDB
- MongoDB Community
  - source-available, free-to-use, self-managed

# Interacting with MongoDB

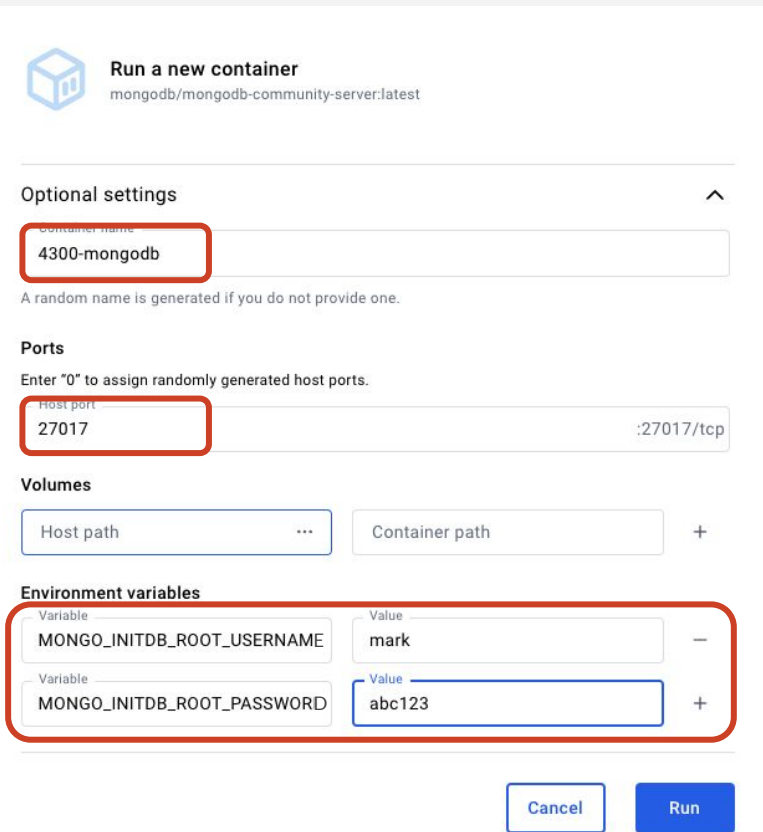
- **mongosh** → MongoDB Shell
  - CLI tool for interacting with a MongoDB instance
- **MongoDB Compass**
  - free, open-source GUI to work with a MongoDB database
- **DataGrip and other 3rd Party Tools**
- **Every major language has a library to interface with MongoDB**
  - PyMongo (Python), Mongoose (JavaScript/node), ...



# Mongodb Community Edition in Docker



- Create a container
- Map host:container port 27017
- Give initial username and password for superuser



# MongoDB Compass

- GUI Tool for interacting with MongoDB instance
- Download and install from > [here](#) <.

## New Connection

Manage your connection settings

URI ⓘ Edit Connection String ☒

mongodb://mark:\*\*\*\*\*@localhost:27017/

Name

Color

☐ **Favorite this connection**  
Favoriting a connection will pin it to the top of your list of connections

▼ Advanced Connection Options

### Advanced Connection Options

General **Authentication** TLS/SSL Proxy/SSH In-Use Encryption Advanced

**Authentication Method**

**Username**

Optional

**Password**

Optional

**Authentication Database ⓘ**

Optional

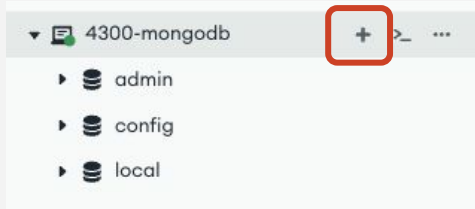
**Authentication Mechanism**

# Load MFlix Sample Data Set

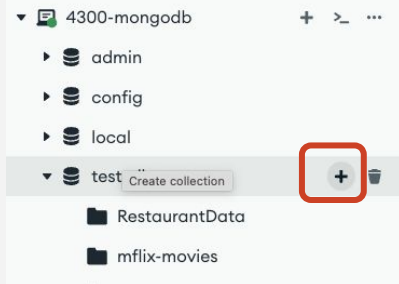
- In Compass, create a new Database named **mflix**
- Download [mflix sample dataset](#) and unzip it
- Import JSON files for users, theaters, movies, and comments into new collections in the mflix database

# Creating a Database and Collection

To Create a new DB:



To Create a new Collection:



## Create Database

Database Name

mflix

Collection Name

users

☐ Time-Series  
Time-series collections efficiently store sequences of measurements over a period of time. [Learn More](#)

> Additional preferences (e.g. Custom collation, Clustered collections)

Cancel

Create Database

# mongosh - Mongo Shell

- find(...) is like SELECT

```
collection.find({ ____ }, { ____ })
```

filters

projections

## mongosh - find()

- SELECT \* FROM users;

```
use mflix
```

```
db.users.find()
```

- SELECT \*  
FROM years  
WHERE name = "Davos Seaworth";

mongosh - find()

filter



```
db.users.find({"name": "Davos Seaworth"})
```

```
< {  
  _id: ObjectId('59b99dbecfa9a34dcd7885c9'),  
  name: 'Davos Seaworth',  
  email: 'liam_cunningham@gameofthron.es',  
  password: '$2b$12$jbGNoWG97LHNIm4axwXDz.tkFITsmw/aylIY/lZDaJRgnHZjB029e'  
}
```

## mongosh - find()

- SELECT \*  
FROM movies  
WHERE rated in ("PG", "PG-13")

```
db.movies.find({rated: {$in:[ "PG", "PG-13" ]}})
```



## mongosh - find()

- Return movies which were released in Mexico and have an IMDB rating of at least 7

```
db.movies.find( {  
  "countries": "Mexico",  
  "imdb.rating": { $gte: 7 }  
} )
```

## mongosh - find()

- Return movies from the **movies** collection which were released in 2010 and either won at least 5 awards or have a genre of Drama

```
db.movies.find( {  
  "year": 2010,  
  $or: [  
    { "awards.wins": { $gte: 5 } },  
    { "genres": "Drama" }  
  ]  
})
```

# Comparison Operators

Name	Description
<code>\$eq</code>	Matches values that are equal to a specified value.
<code>\$gt</code>	Matches values that are greater than a specified value.
<code>\$gte</code>	Matches values that are greater than or equal to a specified value.
<code>\$in</code>	Matches any of the values specified in an array.
<code>\$lt</code>	Matches values that are less than a specified value.
<code>\$lte</code>	Matches values that are less than or equal to a specified value.
<code>\$ne</code>	Matches all values that are not equal to a specified value.
<code>\$nin</code>	Matches none of the values specified in an array.

## mongosh - countDocuments()

- How many movies from the **movies** collection were released in 2010 and either won at least 5 awards or have a genre of Drama

```
db.movies.countDocuments( {  
  "year": 2010,  
  $or: [  
    { "awards.wins": { $gte: 5 } },  
    { "genres": "Drama" }  
  ]  
})
```

## mongosh - project

- Return the names of all movies from the **movies** collection that were released in 2010 and either won at least 5 awards or have a genre of Drama

```
db.movies.countDocuments( {  
  "year": 2010,  
  $or: [  
    { "awards.wins": { $gte: 5 } },  
    { "genres": "Drama" }  
  ],  
  { "name": 1, "_id": 0 } )
```

 1 = return; 0 = don't return

# PyMongo

- PyMongo is a Python library for interfacing with MongoDB instances

```
from pymongo import MongoClient  
client = MongoClient(  
    'mongodb://user_name:pw@localhost:27017'  
)
```

# Getting a Database and Collection

```
from pymongo import MongoClient
client = MongoClient(
    'mongodb://user_name:pw@localhost:27017'
)

db = client['ds4300']
collection = db['myCollection']
```



# Inserting a Single Document

```
db = client['ds4300']
collection = db['myCollection']

post = {
    "author": "Mark",
    "text": "MongoDB is Cool!",
    "tags": ["mongodb", "python"]
}

post_id = collection.insert_one(post).inserted_id
print(post_id)
```

# Count Documents in Collection

- `SELECT count(*) FROM collection`

```
demodb.collection.count_documents({})
```

??