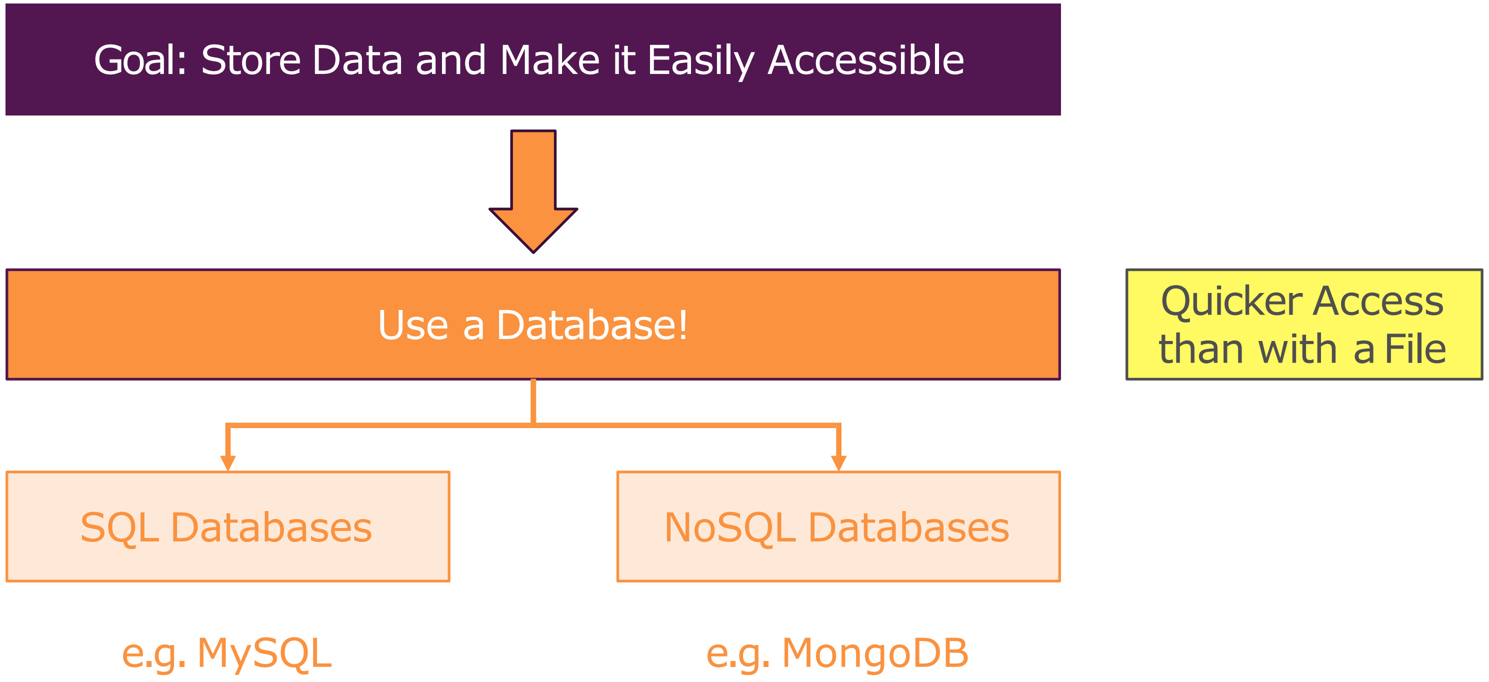
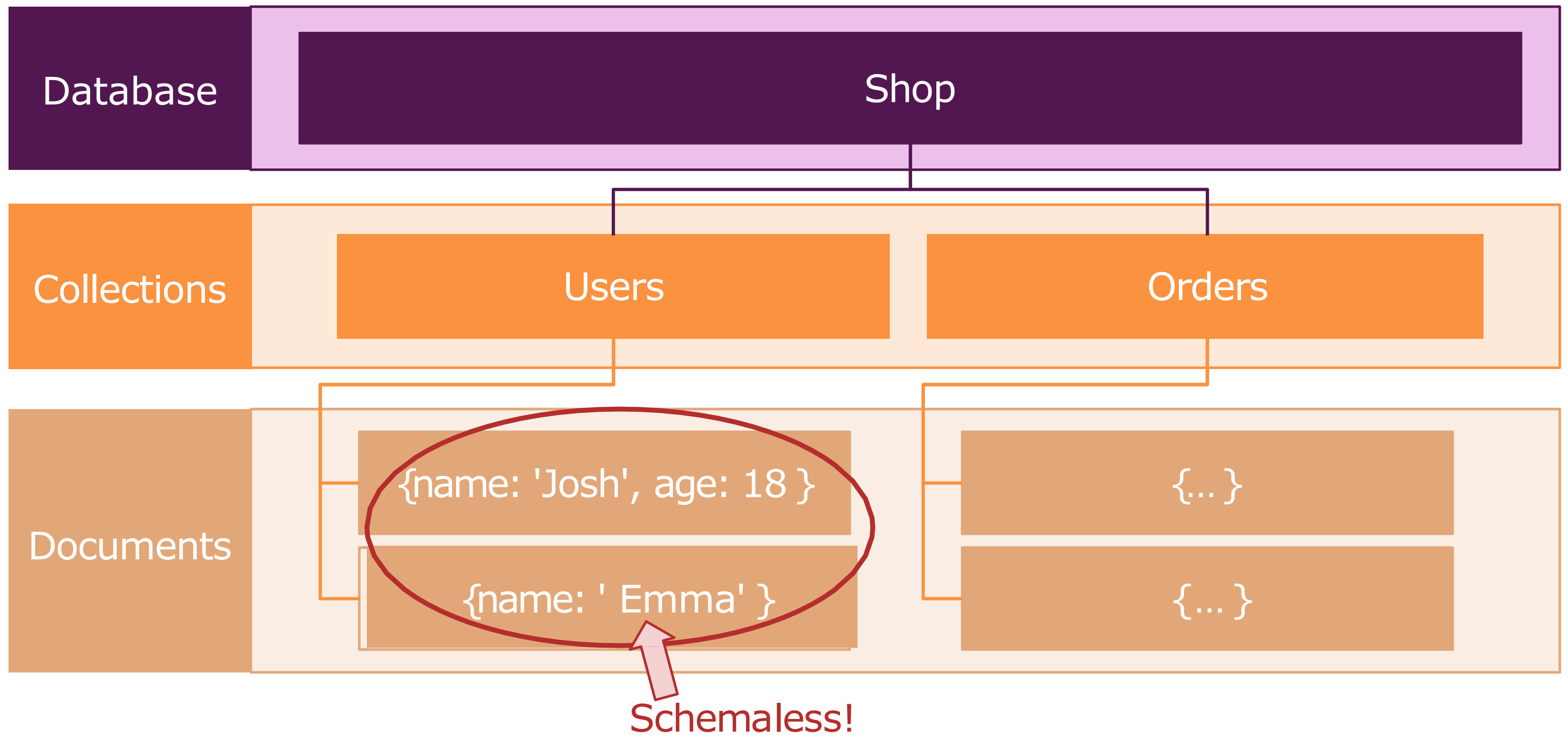
MongoDB – Intro & CRUD

Rujuan Xing

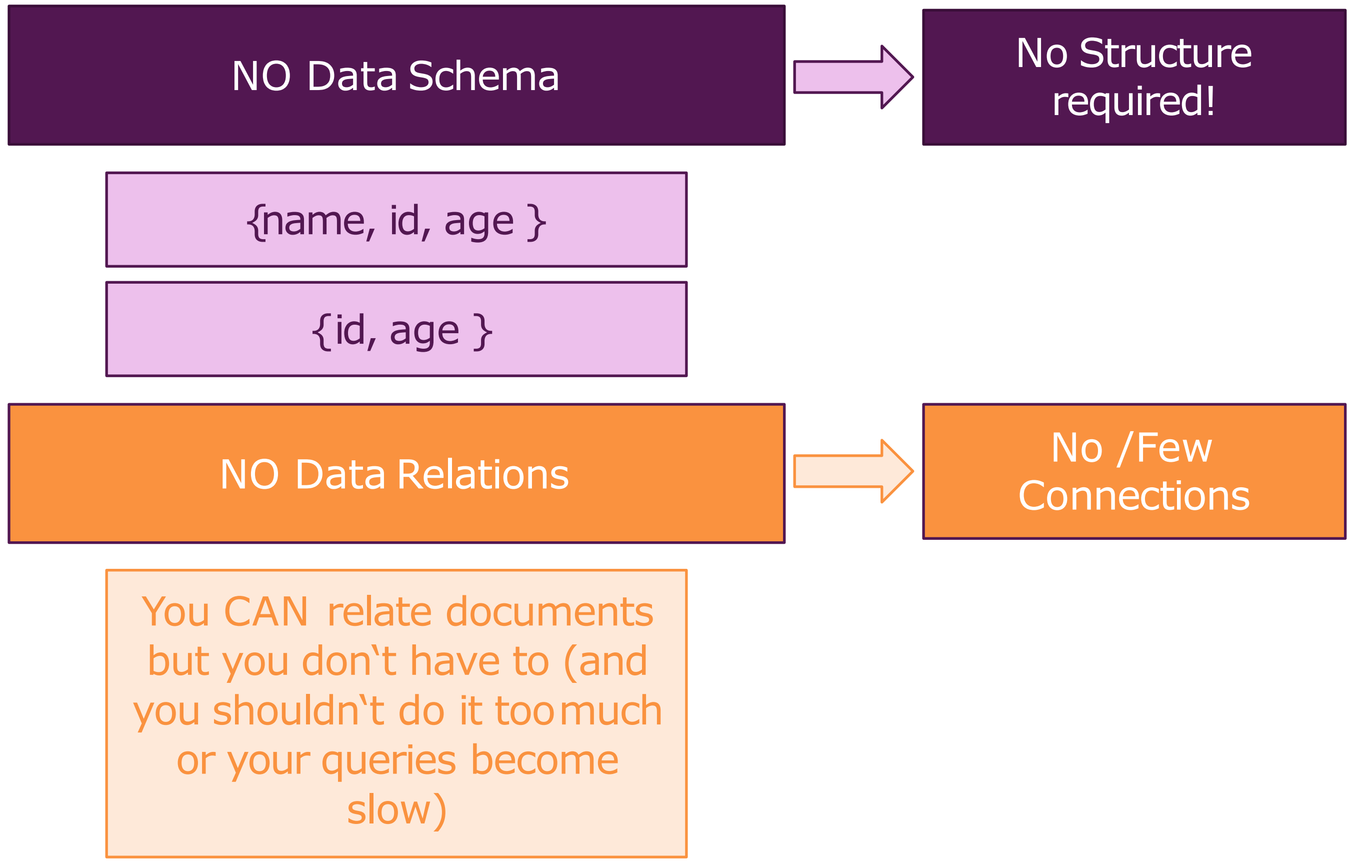
SQL vs NoSQL


Goal: Store Data and Make it Easily Accessible
Use a Database!
Quicker Access than with a File
SQL Databases
NoSQL Databases
e.g. MySQL
e.g. MongoDB

NoSQL


Database
Shop
Collections
Users
Orders
Documents
{name: 'Josh', age: 18 }
{name: ' Emma' }
{... }
{... }
Schemaless!

NoSQL Characteristics



NO Data Schema → No Structure required!

{name, id, age }

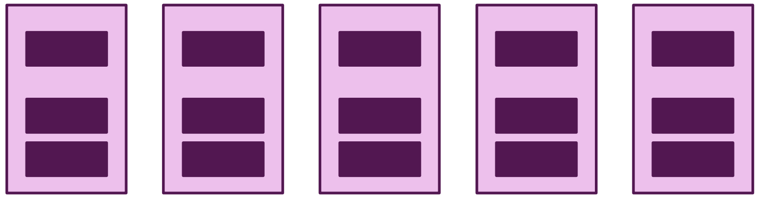
{id, age }

NO Data Relations → No /Few Connections

You CAN relate documents but you don't have to (and you shouldn't do it too much or your queries become slow)

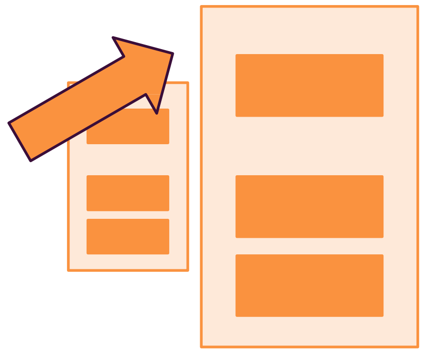
Horizontal vs Vertical Scaling

Horizontal Scaling



Add More Servers (and merge Data into one Database)

Vertical Scaling



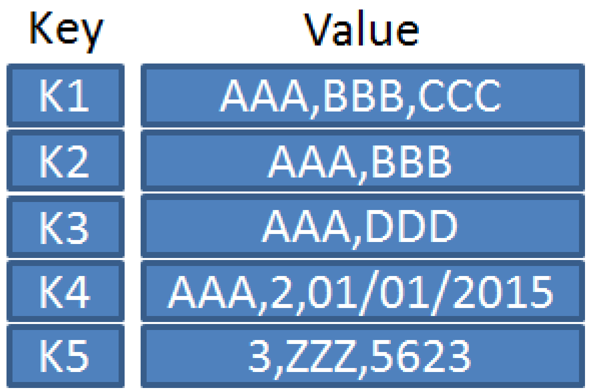
Improve Server Capacity / Hardware

NoSQL Revolution

- NoSQL (originally referring to "non SQL" or "non relational") databases were created for "Big Data" and Real-Time Web Applications, it provides new data architectures that can handle the ever-growing velocity and volume of data.

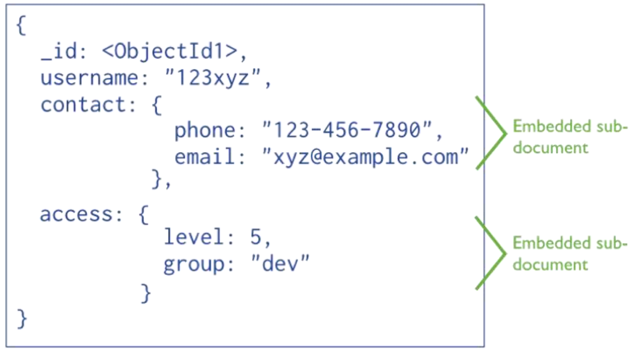
Name	Year	Type	Developer
MongoDB	**2008**	**Document**	**10Gen**
CouchDB	2005	Document	Apache
Cassandra	2008	Column Store	Apache
CouchBase	2011	Document	Couchbase
Riak	2009	Key-Value	Basho Technologies
SimpleDB	2007	Document	Amazon
BigTable	2015	Column Store	Google
Azure Cosmos DB	2017	Multi-Model	Microsoft

NOSQL Database Types



Key	Value
K1	AAA,BBB,CCC
K2	AAA,BBB
K3	AAA,DDD
K4	AAA,2,01/01/2015
K5	3,ZZZ,5623

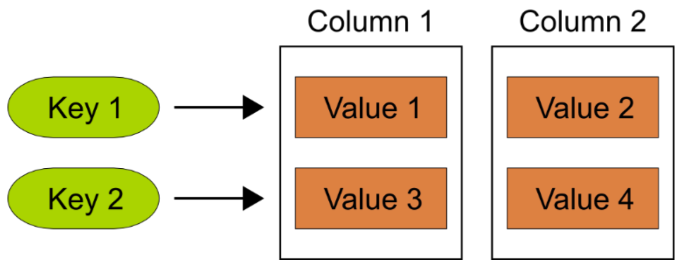
Key-Value Stores



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

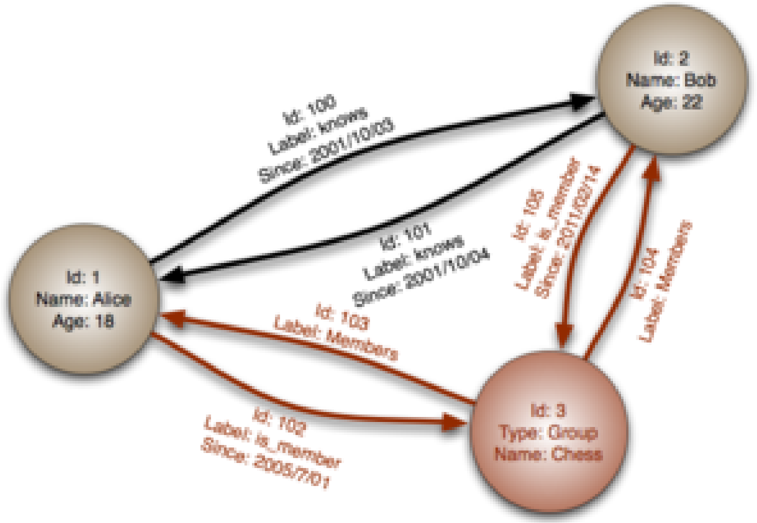
Embedded sub-document
Embedded sub-document

Document Databases



Column Family Stores

Key-Value pairs in hash table, always unique key. Logical group of keys are called: buckets
Document Databases uses Key-Value pairs in a document (JSON, BSON)
Column Stores data is stored in cells that are grouped in columns of data rather than rows (unlimited columns)
Graph Databases, uses flexible graphical representation (edges and nodes) instead of k/v pairs. Index free. Very fast for associative data sets and maps.



Graph Databases

What is MongoDB?

- MongoDB is an open-source document database that provides high performance, high availability, and automatic scaling.
- Non relational DB, stores BSON documents.
- Schemaless: Two documents don't have the same schema.

Document Data Model

- A record in MongoDB is a Document
- Structure of key/value pairs
- Values may contain other documents, arrays and arrays of documents.

```
{
    _id: 1,
    firstname: "Josh",
    lastname: "Edward",
    email: "test@mim.edu",
    phones: ["6414511111", "6414512222"]
}
```

BSON

- BSON, short for Binary JSON, is a binary-encoded serialization of JSON-like documents.
- Both JSON and BSON support Rich Documents (embedding documents and arrays within other documents and arrays).
- BSON also contains extensions that allow representation of data types that are not part of the JSON spec. (For example, BSON has a BinData ObjectId, 64 bits Integers and Date type…etc)

BSON characteristics

- Lightweight
  - Keeping spatial overhead to a minimum is important for any data representation format, especially when used over the network.
- Traversable
  - BSON is designed to be traversed easily. This is a vital property in its role as the primary data representation for MongoDB.
- Efficient
  - Encoding data to BSON and decoding from BSON can be performed very quickly in most languages. For example, integers are stored as 32 (or 64) bit integers and they don't need to be parsed to and from text.

Non-Relational

- Scalability and Performance *(embedded data models reduces I/O activity on database system)*
- Depth of Functionality *(Aggregation framework, Text Search, Geospatial Queries)*
- To retains scalability
  - MongoDB **does not support Joins** between two collections *($loopup)*
  - **No relational algebra:** tables/columns/rows *(SQL)*
  - **No Transactions** across multiple collections *(Do it programmatically, documents can be accessed atomically)*

Schema

- By default, a collection does not require its documents to have the same schema, the documents in a single collection do not need to have the same set of fields and the data type for a field can differ across documents within a collection.
- Starting of MongoDB 3.2, you can enforce document validation rules for a collection during update and insert operations

Document Structure

- The value of a field can be any of the BSON data types, including other documents, arrays, and arrays of documents.

```
const doc = {
    _id: new ObjectID('5e44ab7638d4f738f05c57a8'),
    name: { first: "Josh", last: "Edward" },
    birth: new Date('Oct 31, 1979'),
    email: "test@mim.edu",
    phones: ["6414511111", "6414512222"]
}
```

Setup

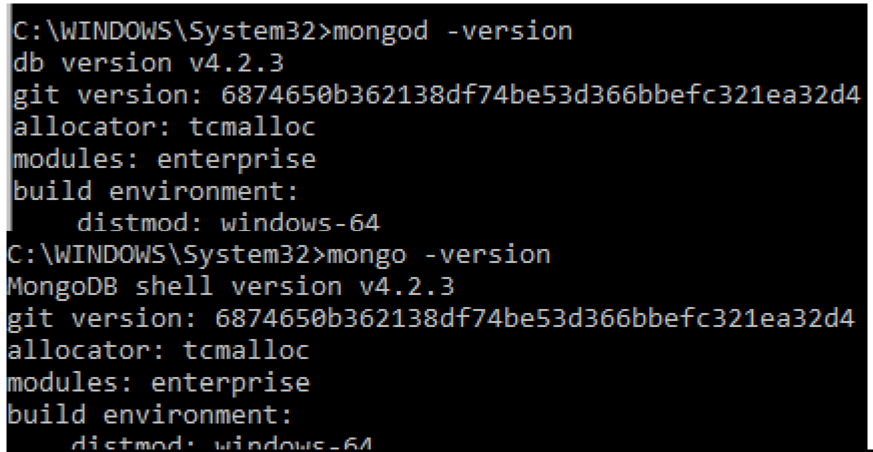
- Download MongoDB Enterprise Version:
  - https://www.mongodb.com/try/download/enterprise
- Install MongoDB Enterprise: Find the right manual based on your OS
  - https://docs.mongodb.com/manual/administration/install-enterprise/

- Two ways to start your MongoDB on Windows. For Mac OS, follow the installation link.
  - as a Windows Service
    1. From the Services console, locate the MongoDB service.
    2. Right-click on the MongoDB service and click **Stop** (or **Pause**).

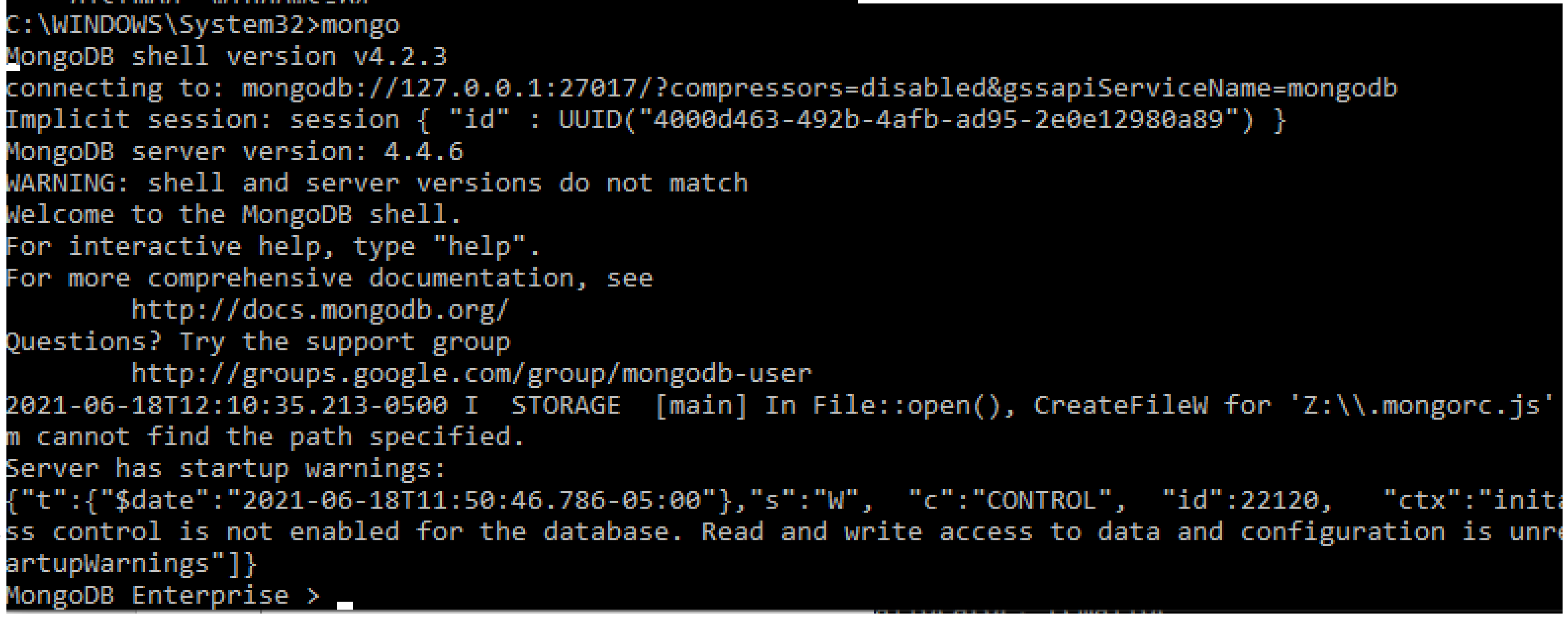
  - from the Command Interpreter
    1. Create database directory - C:/data/db
    2. Start your MongoDB database.
       - "C:\Program Files\MongoDB\Server\4.4\bin\mongod.exe" --dbpath="c:\data\db"
    3. Connect to MongoDB
       - "C:\Program Files\MongoDB\Server\4.4\bin\mongo.exe"

Setup Testing

- Test if MongoDB installed and started successfully
  - mongod -version
  - mongo -version
  - mongo

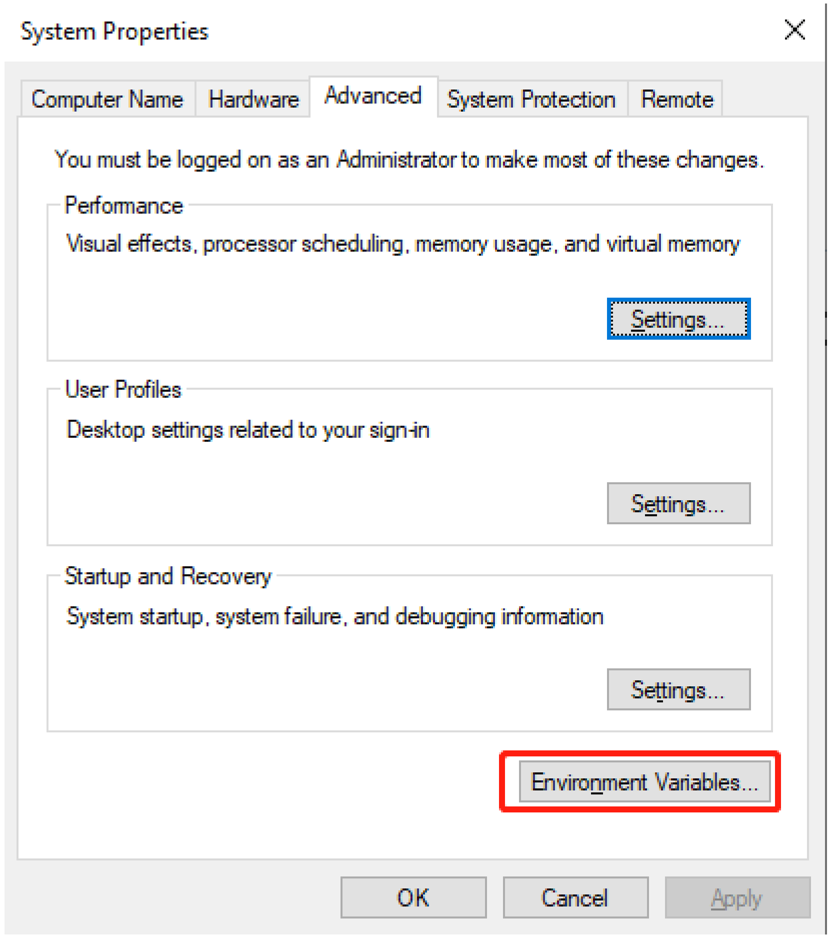


```
C:\WINDOWS\System32>mongod -version
db version v4.2.3
git version: 6874650b362138df74be53d366bbefc321ea32d4
allocator: tcmalloc
modules: enterprise
build environment:
    distmod: windows-64
C:\WINDOWS\System32>mongo -version
MongoDB shell version v4.2.3
git version: 6874650b362138df74be53d366bbefc321ea32d4
allocator: tcmalloc
modules: enterprise
build environment:
    distmod: windows-64
```



```
C:\WINDOWS\System32>mongo
MongoDB shell version v4.2.3
connecting to: mongodb://127.0.0.1:27017/?compressors=disabled&gssapiServiceName=mongodb
Implicit session: session { "id" : UUID("4000d463-492b-4afb-ad95-2e0e12980a89") }
MongoDB server version: 4.4.6
WARNING: shell and server versions do not match
Welcome to the MongoDB shell.
For interactive help, type "help".
For more comprehensive documentation, see
        http://docs.mongodb.org/
Questions? Try the support group
        http://groups.google.com/group/mongodb-user
2021-06-18T12:10:35.213-0500 I  STORAGE  [main] In File::open(), CreateFileW for 'Z:\\.mongorc.js'
m cannot find the path specified.
Server has startup warnings:
{"t":{"$date":"2021-06-18T11:50:46.786-05:00"},"s":"W",  "c":"CONTROL",  "id":22120,   "ctx":"inita
ss control is not enabled for the database. Read and write access to data and configuration is unre
artupWarnings"]}
MongoDB Enterprise >
```

Set up Environment Variables



System Properties

You must be logged on as an Administrator to make most of these changes.

Performance

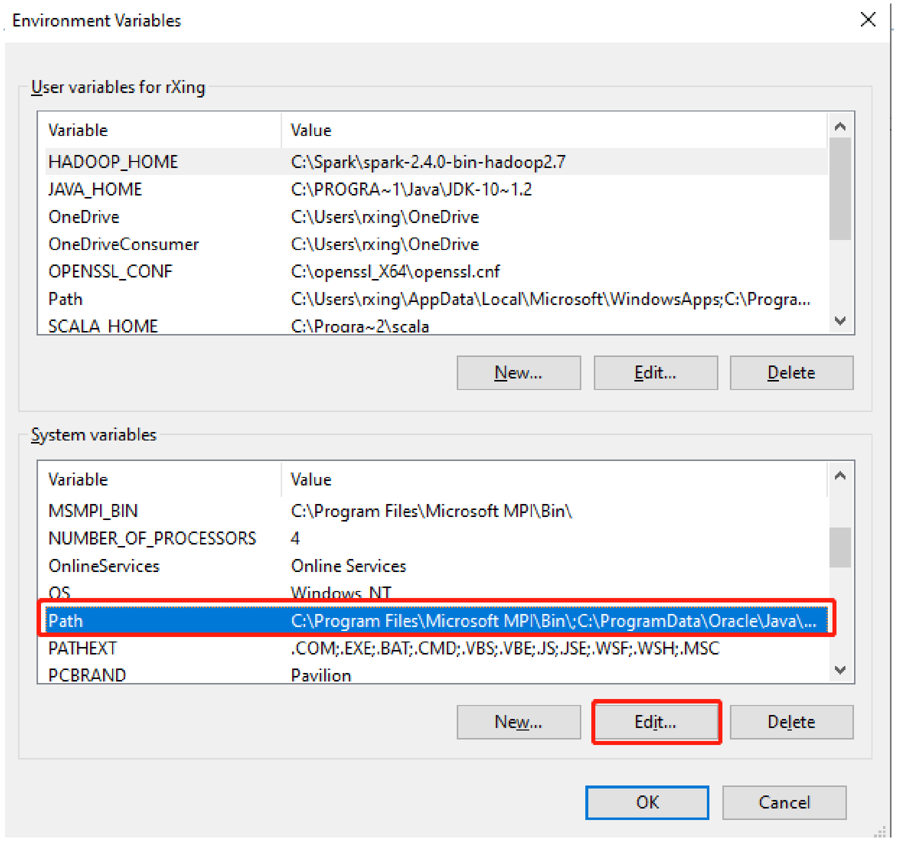
Visual effects, processor scheduling, memory usage, and virtual memory

User Profiles

Desktop settings related to your sign-in

Startup and Recovery

System startup, system failure, and debugging information



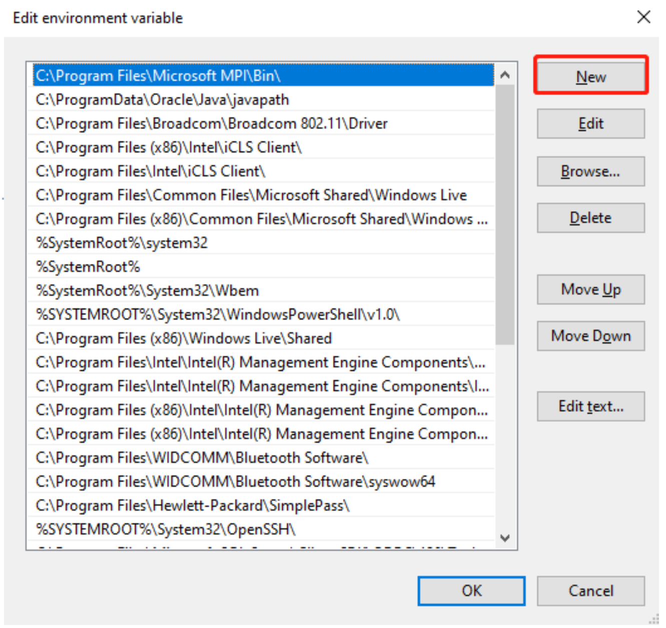
Environment Variables

User variables for rXing

Variable	Value
HADOOP_HOME	C:\Spark\spark-2.4.0-bin-hadoop2.7
JAVA_HOME	C:\PROGRA~1\Java\JDK-10~1.2
OneDrive	C:\Users\rxing\OneDrive
OneDriveConsumer	C:\Users\rxing\OneDrive
OPENSSL_CONF	C:\openssl_X64\openssl.cnf
Path	C:\Users\rxing\AppData\Local\Microsoft\WindowsApps;C:\Progra...
SCALA_HOME	C:\Progra~2\scala

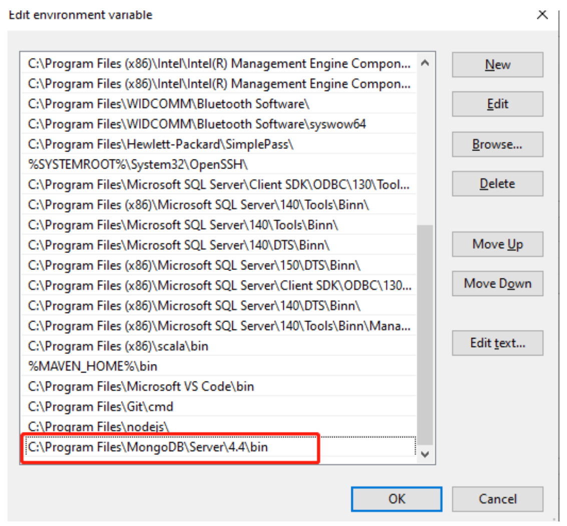
System variables

Variable	Value
MSMPI_BIN	C:\Program Files\Microsoft MPI\Bin\
NUMBER_OF_PROCESSORS	4
OnlineServices	Online Services
OS	Windows_NT
Path	C:\Program Files\Microsoft MPI\Bin\;C:\ProgramData\Oracle\Java\...
PATHEXT	.COM;.EXE;.BAT;.CMD;.VBS;.VBE;.JS;.JSE;.WSF;.WSH;.MSC
PCBRAND	Pavilion



Edit environment variable

C:\Program Files\Microsoft MPI\Bin\
C:\ProgramData\Oracle\Java\javapath
C:\Program Files\Broadcom\Broadcom 802.11\Driver
C:\Program Files (x86)\Intel\iCLS Client\
C:\Program Files\Intel\iCLS Client\
C:\Program Files\Common Files\Microsoft Shared\Windows Live
C:\Program Files (x86)\Common Files\Microsoft Shared\Windows ...
%SystemRoot%\system32
%SystemRoot%
%SystemRoot%\System32\Wbem
%SYSTEMROOT%\System32\WindowsPowerShell\v1.0\
C:\Program Files (x86)\Windows Live\Shared
C:\Program Files\Intel\Intel(R) Management Engine Components\...
C:\Program Files\Intel\Intel(R) Management Engine Components\I...
C:\Program Files (x86)\Intel\Intel(R) Management Engine Compon...
C:\Program Files (x86)\Intel\Intel(R) Management Engine Compon...
C:\Program Files\WIDCOMM\Bluetooth Software\
C:\Program Files\WIDCOMM\Bluetooth Software\syswow64
C:\Program Files\Hewlett-Packard\SimplePass\
%SYSTEMROOT%\System32\OpenSSH\



Edit environment variable

C:\Program Files (x86)\Intel\Intel(R) Management Engine Compon...
C:\Program Files (x86)\Intel\Intel(R) Management Engine Compon...
C:\Program Files\WIDCOMM\Bluetooth Software\
C:\Program Files\WIDCOMM\Bluetooth Software\syswow64
C:\Program Files\Hewlett-Packard\SimplePass\
%SYSTEMROOT%\System32\OpenSSH\
C:\Program Files\Microsoft SQL Server\Client SDK\ODBC\130\Tool...
C:\Program Files (x86)\Microsoft SQL Server\140\Tools\Binn\
C:\Program Files\Microsoft SQL Server\140\Tools\Binn\
C:\Program Files\Microsoft SQL Server\140\DTS\Binn\
C:\Program Files (x86)\Microsoft SQL Server\150\DTS\Binn\
C:\Program Files (x86)\Microsoft SQL Server\Client SDK\ODBC\130...
C:\Program Files (x86)\Microsoft SQL Server\140\DTS\Binn\
C:\Program Files (x86)\Microsoft SQL Server\140\Tools\Binn\Mana...
C:\Program Files (x86)\scala\bin
%MAVEN_HOME%\bin
C:\Program Files\Microsoft VS Code\bin
C:\Program Files\Git\cmd
C:\Program Files\nodejs\
C:\Program Files\MongoDB\Server\4.4\bin

Collections

- MongoDB stores documents in collections. (Collections are similar to tables in relational databases)

```
use myDB
```

- If a database/collection does not exist, MongoDB creates the db/collection when you first store data for that collection

```
use myNewDB
db.myNewCollection.insert( { x: 1 } )
```

- *The insert() operation creates both the database myNewDB and the collection myNewCollection if they do not already exist.*

Shell Demo

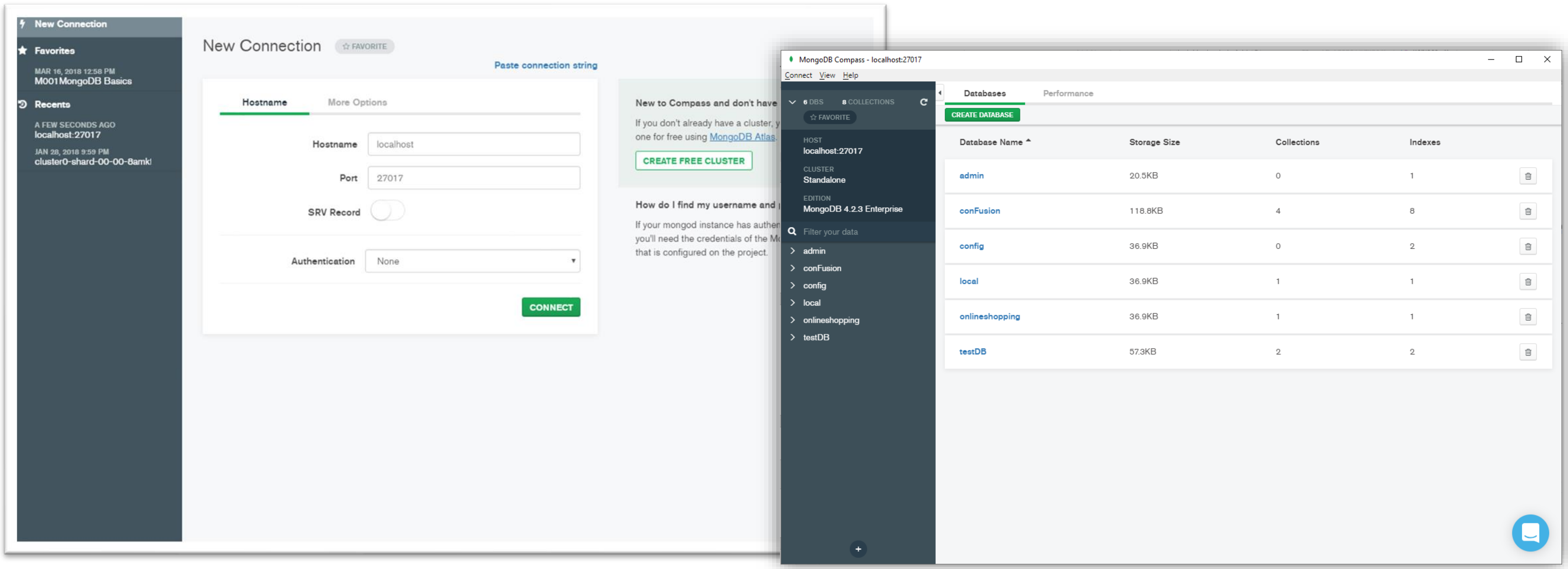
```
show dbs
use testDB // switch or create
show collections
db.testCol.insert({"name": "Josh"})// db var refers to the current database
db.testCol.find() // notice _id
// passing a parameter to find a document that has a property "name" and value "Josh"
db.testCol.find({"name":"Josh"})
// save() = upsert if _id provided
db.testCol.save({"name":"Mike"})
// insert 10 documents – Shell is C++ app that uses V8
for (var i=0; i<10; i++){ db.testCol.insert({"x": i}) }
```

Shell Demo

```
db.testCol.save({a:1, b:2})
db.testCol.save({a:3, b:4, fruit: ["apple", "orange"] })
db.testCol.save({name: "Josh", address: {city: "Fairfield",
                                         zip: 52557,
                                         street: "1000 N 4th street"}
})
// show documents in a nice way, it will only work when you
have nested or larger documents:
db.testCol.find().pretty()
```

MongoDB Compass

- As the GUI for MongoDB, MongoDB Compass allows you to make smarter decisions about document structure, querying, indexing, document validation, and more. Commercial subscriptions include technical support for MongoDB Compass.



MAR 16, 2018 12:58 PM
M001 MongoDB Basics

A FEW SECONDS AGO
localhost:27017

JAN 28, 2018 9:59 PM
cluster0-shard-00-00-8amk

Hostname: localhost
Port: 27017
SRV Record
Authentication: None

New to Compass and don't have
If you don't already have a cluster, y
one for free using MongoDB Atlas.

How do I find my username and
If your mongod instance has authen
you'll need the credentials of the M
that is configured on the project.

MongoDB Compass - localhost:27017

6 DBS 8 COLLECTIONS

HOST
localhost:27017
CLUSTER
Standalone
EDITION
MongoDB 4.2.3 Enterprise

- admin
- conFusion
- config
- local
- onlineshopping
- testDB

Database Name	Storage Size	Collections	Indexes
admin	20.5KB	0	1
conFusion	118.8KB	4	8
config	36.9KB	0	2
local	36.9KB	1	1
onlineshopping	36.9KB	1	1
testDB	57.3KB	2	2

General Rules

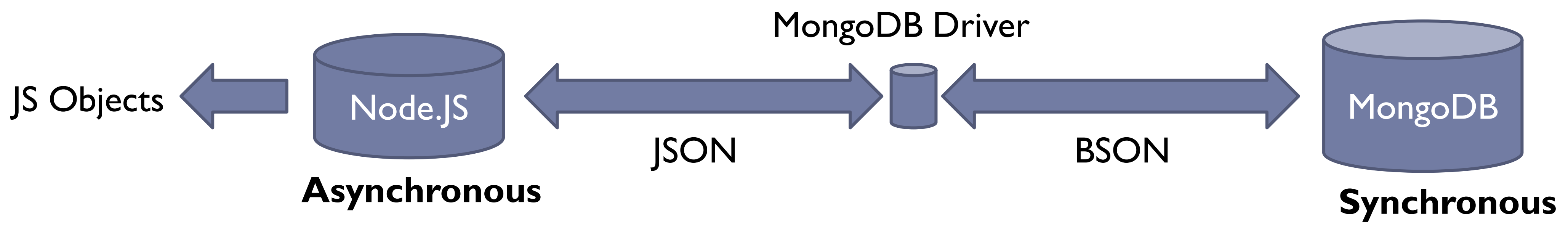
- Field names are strings.
- The field name `_id` is reserved for use as a primary key. It is immutable and always the first field in the document. It may contain values of any BSON data type, other than an array.
- The field names cannot start with the dollar sign (`$`) character and cannot contain the dot (`.`) character or `null`. Field names cannot be duplicated.
- The maximum BSON document size is 16 megabytes. *(To store documents larger than the maximum size, MongoDB provides the GridFS API)*

MongoDB Driver

- A library written in JS to handle the communication, open sockets, handle errors and talk with MongoDB Server.

```
npm install mongodb
```

- Note that Mongo Shell is **Synchronous** while Node.JS is **Asynchronous**.



Connect to MongoDB – 3.0+

```
const MongoClient = require('mongodb').MongoClient;

MongoClient.connect('mongodb://localhost:27017', { useUnifiedTopology: true })
    .then(client => {
        console.log('Connected......');
        const db = client.db('testDB');
        db.collection('testCol').find().each(function(err, doc) {
            if (err) throw err;
            // Print the result.
            // Will print a null if there are no documents in the db.
            console.log(doc);
            // Close the DB
            client.close();
        });
    })
    .catch(err => console.log('Error: ', err));
```

Example - Using findOne()

```
const mongodb = require('mongodb');
const MongoClient = mongodb.MongoClient;

MongoClient.connect('mongodb://localhost:27017', { useUnifiedTopology: true })
    .then(client => {
        console.log('Connected......');

        const db = client.db('testDB');

        db.collection('testCol').findOne({ 'name': 'John' }, function(err, doc) {
            if (err) throw err;
            // Print the result.
            // Will print a null if there are no documents in the db.
            console.log(doc);
            // Close the DB
            client.close();
        });
    })
    .catch(err => console.log(err));
```

`console.dir` vs `console.log`

`console.log()` only prints out a string, whereas `console.dir()` prints out a navigable object tree

Example - Using find() with query & projection

```
const MongoClient = require('mongodb').MongoClient;
const { ObjectID } = require('mongodb');

MongoClient.connect('mongodb://localhost:27017', { useUnifiedTopology: true }, function(err, c
lient) {
    if (err) throw err;

    const db = client.db('testDB');

    const query = { _id: new ObjectID('60ce143d3c9d6e6678b1f6a7') };
    const projection = { firstname: 1, lastname: 1, _id: 0 };

    db.collection('testCol').find(query).project(projection).toArray(function(err, docArr) {
        if (err) throw err;
        docArr.forEach(function(doc) {
            console.log(doc);
        });
        client.close();
    });
});
```

Note: Projection is a good practice to save bandwidth and retrieve only the data we need.

Example - Skip, Limit and Sort

```
const MongoClient = require('mongodb').MongoClient;

MongoClient.connect('mongodb://localhost:27017', { useUnifiedTopology: true }, function(err
, client) {
    if (err) throw err;

    const db = client.db('testDB');

    const query = {};
    const projection = { firstname: 1, lastname: 1, _id: 0 };

    db.collection('testCol').find(query).project(projection)
        .skip(1).limit(3).sort('age', 1).toArray(function(err, docArr) {
            if (err) throw err;
            docArr.forEach(function(doc) {
                console.log(doc);
            });

            client.close();
        });
});
```

Note: These will be implemented in the DB in a very specific order: **1. sort, 2. skip, 3. limit** no matter how we put them in the code *(remember cursor object is being built and sent to the server only , to run call* `.each()`*)*

Example - Using insertOne()

```
const MongoClient = require('mongodb').MongoClient;

MongoClient.connect('mongodb://localhost:27017', { useUnifiedTopology: true }, fu
nction(err, client) {
    if (err) throw err;
    const db = client.db('testDB');
    var doc = { 'firstname': 'Bella', 'lastname': 'Xing', 'age': 10 };

    db.collection('testCol').insertOne(doc, (err, docInserted) => {
        if (err) throw err;

        console.log(`Success: ${JSON.stringify(docInserted)}!`);
        return client.close();
    });
});
```

Example - Using updateOne()

```
const MongoClient = require('mongodb').MongoClient;
const { ObjectID } = require('mongodb');

MongoClient.connect('mongodb://localhost:27017', { useUnifiedTopology: true }, functio
n(err, client) {
    if (err) throw err;
    const db = client.db('testDB');

    db.collection('testCol').updateOne({ '_id': new ObjectID('60ce12cb3c9d6e6678b1f6a5
') }, { $set: { 'firstname': 'Ivy', 'lastname': 'Friday' } }, function(err, result) {
        if (err) throw err;
        console.log(result.result);
        return client.close();
    });

});
```

Example - Using deleteOne()

```
const MongoClient = require('mongodb').MongoClient;

MongoClient.connect('mongodb://localhost:27017', { useUnifiedTopology: true }
    , function (err, client) {
        if (err) throw err;

        const db = client.db('testDB');
        var query = { 'student': 'Susie' };
        // remove all documents that have 'student' value is 'Susie'
        db.collection('testCol').deleteOne(query, function (err, result) {
            console.log("Result: " + JSON.stringify(result));
            return client.close();
        });
    });
```

In Real Application... Like this?

utils/database.js

```
const mongodb = require('mongodb');
const MongoClient = mongodb.MongoClient;

const mongoConnect = (callback) => {
    MongoClient.connect('mongodb://localhost:27017')
        .then(client => {
            console.log('Connected......');
            callback(client);
        })
        .catch(err => console.log(err));
}

module.exports = mongoConnect;
```

models/product.js

```
const mongoConnect = require('../util/database');

class Product {
…
 save() {
   mongoConnect((client) => {
       client.db('onlineshopping').collection('products')
                .insertOne(this)
                .then(result => console.log(result))
                .catch(err => console.log(err));
        });
    }}
```

In Real Application…

utils/database.js

```
const mongodb = require('mongodb');
const MongoClient = mongodb.MongoClient;
let _db; //indicate private variable
const mongoConnect = (callback) => {
    MongoClient.connect('mongodb://localhost:27017',
        { useUnifiedTopology: true })
        .then(client => {
            console.log('Connected......');
            _db = client.db('testCol');
            callback();
        })
        .catch(err => console.log(err));
}
const getDb = () => {
    if (_db) {
        return _db;
    }
    throw new Error('No Database Found!');
}
exports.mongoConnect = mongoConnect;
exports.getDb = getDb;
```

app.js

```
const mongoConnect = require('./util/database').m
ongoConnect;
mongoConnect(() => {
    app.listen(3000);
});
```

models/product.js

```
const { ObjectID } = require('mongodb');
const getDb = require('../utils/database').getDb;

module.exports = class Product {

    save() {
        return getDb().collection('products').ins
ertOne(this);
    }
```

Resources

- SQL vs NoSQL: https://academind.com/learn/web-dev/sql-vs-nosql/
- Mongo Shell: https://docs.mongodb.com/manual/mongo/
- MongoDB CRUD Operations: https://docs.mongodb.com/manual/crud/
- Node.js MongoDB Driver API: https://mongodb.github.io/node-mongodb-native/4.0/