- What is MongoDB?
- Key features of MongoDB
- Getting setup with MongoDB
- Ways to interact with data

MongoDB is an open-source NoSQL database
- A document is a BSON object ("binary JSON")
- A document contains fieldname/value pairs
- Values can be simple types, arrays, or nested documents

Here's an example of a MongoDB document:

```
{
    name: "Jeremy",
    age: 61,
    skills: [ "Java", "C++", "JavaScript" ],
    additionalInfo: {
        nationality: 'UK',
        companyCar: {
            make: 'Bugatti',
            model: 'Divo'
        }
    }
}
```

Key Features

High performance
- Via indexes

Rich query language for CRUD operations
- Data aggregation
- Text search and queries

High availability
- Via automatic failover and data redundancy

Horizontal scalability across a cluster
- Via sharding

How to interact?

Installing locally

Long-term, this is probably the best option. There are instructions on the website for each platform.

If you have permissions, feel free to install and use.

Using MongoDB Atlas

To speed up getting setup and learning, we can use MongoDB Atlas cloud offering.

This is built on top of AWS free tier. We'll go through getting set up.

- Command line
- GUI
- ORM

MongoDB Shell

Compass

Atlas

PyMongo

- MongoDB documents
- Field types
- Accessing fields in a document
- MongoDB collections
- Creating a collection

MongoDB provides a simple-to-use API that allows you to perform CRUD operations on NoSQL data
- Create (insert) documents into a collection
- Read (find) documents in a collection
- Update existing documents in a collection
- Delete existing documents in a collection

The MongoDB API is available in several languages, including:
- JavaScript (via the MongoDB Shell - see this chapter)
- Python (via PyMongo)
- C# (via MongoDB NuGet packages)
- Java (e.g. via Spring Boot repositories)

MongoDB Documents

A MongoDB document is a BSON object
- BSON is effectively binary JSON - see http://bsonspec.org/
- Max document size is 16MB

MongoDB documents contain fieldname/value pairs

Miscellaneous notes:
- Field names are strings
- Each document has a special field named _id (primary key)
- MongoDB preserves the ordering of fields (_id is always first)

```
{
    field1: value1,
    field2: value2,
    …
    fieldN: valueN
}
```

Field Types

A field can be:
- Any BSON type
- An array, document, or array of documents

Example:

```
{
    _id:    ObjectId("21aa914e8405a59ce38a94a2"),      //
        Unique ID for this object.
    name:   { first: "Ola", last: "Nordmann" },        //
        Embedded document.
    dob:    new Date('Jul 2, 1997'),                   //
        Date object.
    langs:  [ "Norwegian", "Swedish", "English" ],     //
        Array of strings.
    views:  NumberLong(1250000)                        //
        64-bit long integer.
}
```

Note:
- BSON has many more standard data types than JSON
- See https://docs.mongodb.com/manual/reference/bson-types/

Accessing Fields in a Document

To access a field in a document:
- Use dot notation

To access an element in an array:
- Use [] notation and specify a zero-based index

Examples:

```
emp1.name          // { "first" : "Ola", "last" : "Nordmann" }

emp1.name.first    // Ola

emp1.langs         // [ "Norwegian", "Swedish", "English" ]

emp1.langs[0]      // Norwegian
```

MongoDB Collections

MongoDB stores documents in collections
- MongoDB collections are analogous to tables in a RDBMS

By default, documents in a collection don't have to have the same schema
- This is one of the attractions of NoSQL databases
- You can specify document validation rules if you like (v3.2+)

Creating a Collection

You can explicitly create a collection
- Via db.createCollection()
- Useful if you want to specify creational options

```
db.createCollection("log", {
    capped: true,
    size: 20000,
    max: 500
})
```

If you don't want to set any options for a collection, you don't need to create the collection explicitly
- Just start inserting documents into the collection
- MongoDB creates the collection if it doesn't already exist

- Creating documents
- Reading documents
- Updating documents
- Deleting documents
- Additional useful collection operations

Creating Documents

To create documents in a collection, call:
- insertOne() - insert a single document into a collection
- insertMany() - insert an array of documents into a collection

```
db.log.insertOne(
    { name: "Kevin", age: 39, gender: "M" }
)
```

```
db.log.insertMany([
    { name: "John", age: 20, gender: "M" },
    { name: "Jane",  age: 20, gender: "F", favTeam: "Swans" }
])
```

Notes:
- MongoDB creates the collection if it doesn't already exist
- MongoDB generates unique _id fields if not specified
- Documents don't have to have the same schema

Reading Selective Documents

You can pass a query filter document into find()
- Specify the conditions that determine which documents to select
- Analogous to WHERE in SQL

```
{
    field1: value1,
    field2: { operator: value },
    …
}
```

Here are some of the query operators you can use:
- $eq, $ne, $gt, $gte, $lt, $lte, $in, $nin
- $and, $or, $nor, $not
- $exists, $type
- $mod, $regex, $text, $where

For full details about these query operators and more, see https://docs.mongodb.com/manual/reference/operator/query/

```
db.log.find({
    name: 'Jayne'
})
```

```
db.log.find({
    age: { $gte: 20 }
})
```

```
db.log.find({
    age: { $gte: 20 },
    age: { $lte: 30 }
})
```

```
db.log.find({
    $or: [
        { age: { $lt: 20 } },
        { age: { $gt: 30 } }
    ]
})
```

```
db.log.find({
    name: /^J/,
    gender: 'F',
    $or: [
        { age: { $lt: 20 } },
        { age: { $gt: 30 } }
    ]
})
```

Reading Selective Fields

By default, find() returns all fields in a document
- Analogous to SELECT * in SQL

You can pass a projection document into find()
- Specify the fields to include/exclude in the result documents
- To specify fields to include, set fields to 1
- To specify fields to exclude, set fields to 0

```
{
    fieldToInclude: 1,
    anotherFieldToInclude: 1,
    …
}
```

```
{
    fieldToExclude: 0,
    anotherFieldToExclude: 0,
    …
}
```

Note:
- The _id field is always returned, by default
- Apart from _id, you can't combine 1s and 0s in your projection

```
db.log.find(
    { name: 'Jane' },
    { age: 1, gender: 1 }
)
```

```
db.log.find(
    { name: 'Jane' },
    { age: 1, gender: 1, _id: 0 }
)
```

```
db.log.find(
    { name: 'Jane' },
    { name: 0 }
)
```

```
db.log.find(
    { name: 'Jane' },
    { name: 0, _id: 0 }
)
```

Updating Documents

To update existing documents in a collection, call:
- updateOne() - update a single document in a collection
- updateMany() - update an array of documents in a collection
- replaceOne() - replace a single document in a collection

For updateOne() and updateMany(), pass 3 params:
- Filter, same as for find()
- Update to perform (e.g. $set, $unset, etc.)
- Options object:
  - upsert - If true, will cause an insert if no matching document found
  - writeConcern - Details about how to perform the "write" operation
  - collation - Language-specific rules for string comparison (locale etc.)

replaceOne() is the same, except the 2nd param is the replacement object

```
db.log.updateOne(
    { name: 'Jayne' },
    { $set: { name: 'JAYNE', favTeam: 'Swans' } }
)
```

```
db.log.updateMany(
    {},
    { $inc: { age: 1 } }
)
```

```
db.log.updateMany(
    {},
    { $rename: { favTeam: 'favouriteTeam' } }
)
```

```
db.log.updateMany(
    {},
    { $currentDate: {
        datestamp: { $type: 'date' },
        timestamp: { $type: 'timestamp' }
      }
    }
)
```

```
db.log.replaceOne(
    { name: 'JAYNE' },
    { name: 'Jayne', age: 52, gender: 'F' }
)
```

Deleting Documents

To delete documents in a collection, call:
- deleteOne() - delete a single document in a collection
- deleteMany() - delete an array of documents in a collection

For both these methods, pass 2 params:
- Filter, same as for find()
- Options object:
  - writeConcern - Details about how to perform the "write" operation
  - collation - Language-specific rules for string comparison (locale etc.)

```
db.log.deleteOne(
    { name: 'Wilfried' }
)
```

```
db.log.deleteMany(
    { favouriteTeam: 'Cardiff' }
)
```

```
db.log.deleteMany(
    { overdraft: { $exists: true } }
)
```

Additional Useful Collection Operations

MongoDB has various other useful collection operations available, including:
- aggregate()
- bulkWrite()
- count(), totalSize(), explain(), distinct(),
- createIndex(), dropIndex(), reIndex(),
- findAndModify(), findAndReplace(), findAndDelete()
- mapReduce()
- remove()

For full details, see:
- http://docs.mongodb.com/manual/reference/method/js-collection/

- PyMongo API documentation
- Connecting to a MongoDB instance
- Getting a handle to a database
- Getting a collection
- MongoDB documents in Python

You can use Python to access MongoDB databases
- Via the PyMongo Python package

The PyMongo API is similar to the MongoDB JS API
- Generally you just convert method name capitals to _
- See https://pymongo.readthedocs.io/en/stable/api/index.html

You can connect to the MongoDB instance in Python
- Via the MongoClient class

Ex1

A MongoDB instance can support multiple databases
- E.g. different databases for different sets of data

The MongoClient class allows you to access the databases on a MongoDB instance

Ex2

You can get a collection in the database as follows:

Ex3

In Python, a MongoDB document is represented as a dictionary
- The dictionary contains name/value pairs
- Correspond to fieldnames/values in a MongoDB document

```
# Create and initialize a Python dictionary object.
person1 = {
    "name": "Jayne",
    "age": 52,
    "gender": "F",
    "langs": ["English", "French"]
}

# Create an empty Python dictionary first, and then add fields via [].
person2 = {}
person2["name"]   = "Andy"
person2["age"]    = 52
person2["gender"] = "M"
person2["langs"]  = ["English", "French", "Norwegian"]
```

- Creating documents
- Reading documents
- Updating documents
- Deleting documents

To create documents in a collection, call:
- insert_one()
- insert_many()

Ex4

To read documents in a collection, call:
- find()
- find_one()

Ex5

To update documents in a collection, call:
- update_one()
- update_many()
- replace_one()

These methods are similar to the JavaScript API, except the parameters are slightly different. For details, see:
- https://pymongo.readthedocs.io/en/stable/api/pymongo/collection.html

Ex6

To delete documents in a collection, call:
- delete_one()
- delete_many()

These methods are similar to the JavaScript API, except the parameters are slightly different. For details, see:
- https://pymongo.readthedocs.io/en/stable/api/pymongo/collection.html

Ex7