

# MongoDB

CMSC 206 – Database Management Systems



# Introduction

In this topic, we will introduce a different type of DBMS, MongoDB. This will mark a break from relational databases, and our first touch with NoSQL.

We will first introduce the document-oriented data model, on which MongoDB is based on and then survey the MongoDB equivalents of the SQL commands we previously discussed.



# Document-oriented Databases



# Document databases - Introduction

- Document-oriented databases represent data as documents each given a unique identifier. Related documents are grouped into collections. A big difference with the relational model is that no schema is specified for the data. The documents are seen as self-describing data.
- You can think of documents a bit like objects in object-oriented programming, with different attributes that have values. As there is no schema though, two objects in the same collection might not have the same attributes. Documents can be stored in different ways, as XML, YAML, or JSON objects for example. MongoDB uses JSON.
- Another difference is that an attribute's value can be a document itself, or an array. Attributes are no longer simple values as in the classic relational model we studied before.



# Document databases - Example

## Relational

ID	first_name	last_name	cell	city	year_of_birth	location_x	location_y
1	'Mary'	'Jones'	'516-555-2048'	'Long Island'	1986	'-73.9876'	'40.7574'

ID	user_id	profession
10	1	'Developer'
11	1	'Engineer'

ID	user_id	name	version
20	1	'MyApp'	1.0.4
21	1	'DocFinder'	2.5.7

ID	user_id	make	year
30	1	'Bentley'	1973
31	1	'Rolls Royce'	1965

## MongoDB

```
{
  first_name: "Mary",
  last_name: "Jones",
  cell: "516-555-2048",
  city: "Long Island",
  year_of_birth: 1986,
  location: {
    type: "Point",
    coordinates: [-73.9876, 40.7574]
  },
  profession: ["Developer", "Engineer"],
  apps: [
    { name: "MyApp",
      version: 1.0.4 },
    { name: "DocFinder",
      version: 2.5.7 }
  ],
  cars: [
    { make: "Bentley",
      year: 1973 },
    { make: "Rolls Royce",
      year: 1965 }
  ]
}
```

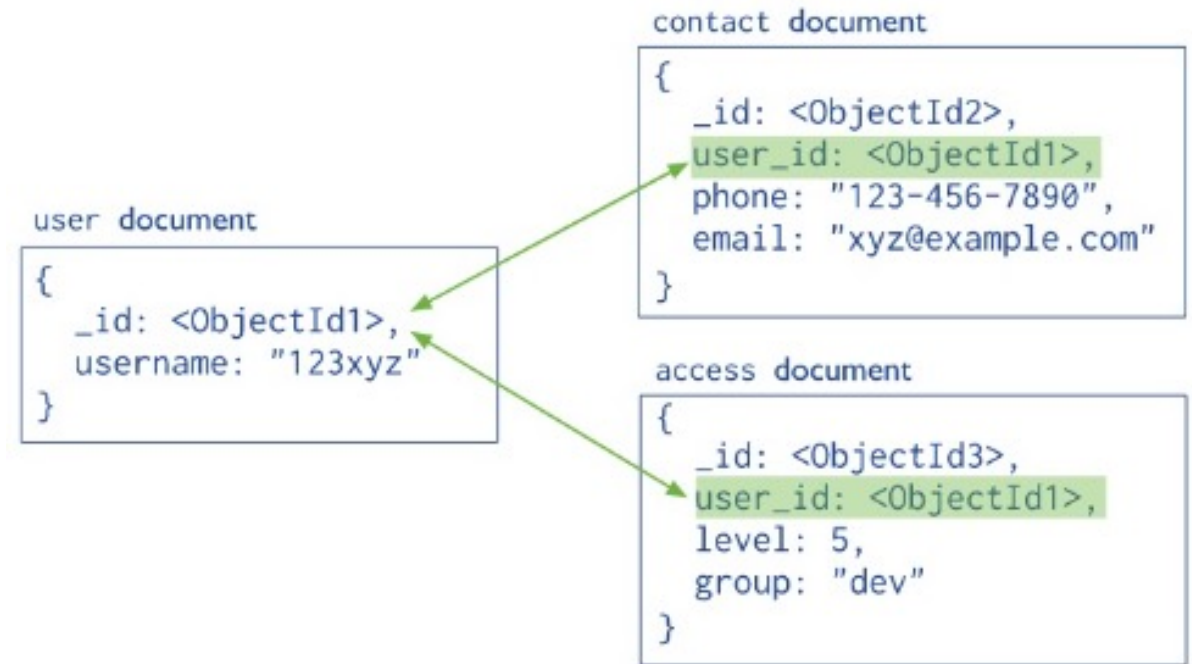
**Figure 1.** Example of relational model to document model in MongoDB from <https://www.mongodb.com/document-databases>. This shows a document that could be part of People collection in a database for example.

# Document databases - Embedding

- In the previous example, we can see that a way to represent relationships between entities is to embed them into one another, e.g. embedding car documents into the person document in Figure 1. Embedding usually leads to better read speed and lets us easily retrieve related data: when we retrieve the “parent” document, we also get all the related, embedded data.
- Embedding is particularly adapted to one-to-one or one-to-many relationships. When used for many-to-many relationships, it leads to data redundancy: the “child” document must be repeated in each “parent.”
- Embedding can also lead to uncontrolled growth of the “parent” document if we have many embedded documents, which could hinder performance , or even break the DBMS document size limit.

# Data modeling document-oriented database - Referencing

- Even in a document database, we can still use the idea of “foreign keys” to represent relationships.
- A big difference with foreign keys in relational databases is that in document-oriented databases, we have no constraints to ensure the integrity of data. Ensuring the integrity is left to the application logic instead of the DBMS.



**Figure 2.** Example of reference between documents in 3 collections

# Data modeling document-oriented database – Embedding vs. Referencing

- Choosing when to use embedding or linking can be a complex problem, and depends on many factors. We should consider the performance trade off we are willing to make (e.g. can we accept some data redundancy for better read performance with embedding?), and the data we are modeling.
- MongoDB has a lot of information available on the subject, as well as examples and patterns [here](#).





# MongoDB



# Introduction

- MongoDB is a very popular document-oriented, JSON-based database. It is designed to be used as a distributed database with replication and high availability and uses an ad-hoc query language that integrates JavaScript.
- The documentation for MongoDB is available [here](#).
- If you want to use MongoDB, there are several options:
  - install it on your computer
  - use the one hosted [here](#) then create a free cluster and connect your DB
- In the next slides, we will see the basics of MongoDB usage. As usual, the documentation should be your first stop for more complete and complex information.



# DDL

MongoDB does not enforce a schema on the data, so there is no need for a DDL. That said, [schema validation](#) is possible.

## Selecting the database to use

```
use dbName;
```

If the database does not exist, it will be created when data is inserted in it. Similarly, when inserting a document in a collection, the collection will be created if it does not already exist. That said, we can also explicitly create a collection and specify options for it, such as a maximum number of documents in the collection, or schema validation rules.

## Creating a collection

```
db.createCollection(name, options)
```

## Example

```
db.createCollection("myCollection", {"capped": true, "max": 100})
```



# DML - Insert

Inserting data into a database is very straightforward, as we just have to specify the collection we want to insert data in and the document(s) we want to insert.

## Inserting data

```
db.collection.insert(document or array of documents)
```

## Example

```
db.people.insert({
  FName: "Thomas",
  LName: "Laurent",
  Nationality: "French"
})
db.carBrands.insert([
  {name: "Renault"},
  {name: "Volkswagen"}
])
```

# DML - update

To update a collection, we have to specify a *query* (WHERE clause in SQL) and the *update* we want to apply (SET clause in SQL).

We can also specify options on how the update should behave, e.g. the ***multi*** option that decides if the update should update only one record matching the query or all of them, with the default being only updating one.

## Updating data

```
db.collection.update(query, update, options)
db.collection.updateOne(query, update, options)
db.collection.updateMany(query, update, options)
```

# DML – update continued

## Updating data

```
db.collection.update(query, update, options)
```

The *query* is expressed with the same operators we will describe later in the DQL.

The update can be expressed in different ways:

- By specifying a new document that should replace the updated document(s)
- By specifying an *update document* that contains [update operators](#) such as **\$set** to set an attribute in the updated documents or **\$inc** to increase an attribute's value.

# DML – Example with new document

## Example

```
db.people.update(  
  { name: "Andy"},    // Query parameter  
  {                   // Update document  
    name: "Andy",  
    rating: 1,  
    score: 1  
  }  
)
```

This will replace one document where the name is "Andy" with the update document.

# DML – Example with new document

## Example

```
db.books.update(  
  { _id: 1 },  
  {  
    $inc: { stock: 5 },  
    $set: {  
      item: "ABC123",  
      "info.publisher": "2222",  
      tags: [ "software" ],  
      "ratings.1": { by: "xyz", rating: 3 }  
    }  
  }  
)
```

This will set **\$set** with the *item*, *info.publisher*, *tags*, and *ratings.1* attributes to the given values. It would also set the *stock* attribute of **\$inc** with 5, in the document with id number 1.



# DML – delete

## Example

```
db.orders.deleteOne(  
  { "expiryts" :  
    { $lt: ISODate("2015-11-01T12:40:15Z")}  
  }  
);
```

This will delete one document in collection "orders" with an expiryts before (less than, \$lt) the 1<sup>st</sup> of November 2015 at 12:50:15.

# DQL

## Querying data

```
db.collection.find(query)
db.collection.findOne(query)
```

MongDB does not use SQL. It uses its own ad-hoc query language based on operators. The main operators are query and [projection operators](#) and the [lookup operator](#), that lets us "join" collections.

You can check a good, simple, interactive introduction to queries in MongoDB, with comparison to SQL [here](#).

# DCL

- By default, MongoDB has no access control, so no need for DCL. Still, if need be, it can be activated.
- You can check how to manage roles of users and permissions [here](#).

