

# MongoDB Extras

---

Olivier Liechti & Simon Oulevay  
COMEM Web Services 2016

heig-vd

Haute Ecole d'Ingénierie et de Gestion  
du Canton de Vaud

# One-to-one relationships

2 documents (requires 2 queries  
to get all of the person data)

```
{
  _id: "joe",
  name: "Joe Bookreader"
}

{
  patron_id: "joe",
  street: "123 Fake Street",
  city: "Faketon",
  state: "MA",
  zip: "12345"
}
```

**Normalized data model**  
(references)

```
{
  _id: "joe",
  name: "Joe Bookreader",
  address: {
    street: "123 Fake Street",
    city: "Faketon",
    state: "MA",
    zip: "12345"
  }
}
```

1 single, aggregate document  
(in this case, it is a better choice)

**Embedded data model**  
(sub-documents)

<https://docs.mongodb.org/manual/tutorial/model-embedded-one-to-one-relationships-between-documents/>

# One-to-many relationships

```
{
  _id: "joe",
  name: "Joe Bookreader"
}

{
  patron_id: "joe",
  street: "123 Fake Street",
  city: "Faketon",
  state: "MA",
  zip: "12345"
}

{
  patron_id: "joe",
  street: "1 Some Other Street",
  city: "Boston",
  state: "MA",
  zip: "12345"
}
```

```
{
  _id: "joe",
  name: "Joe Bookreader",
  addresses: [
    {
      street: "123 Fake Street",
      city: "Faketon",
      state: "MA",
      zip: "12345"
    },
    {
      street: "1 Some Other Street",
      city: "Boston",
      state: "MA",
      zip: "12345"
    }
  ]
}
```



MongoDB document can have  
an arbitrary structure, including  
arrays

<https://docs.mongodb.org/manual/tutorial/model-embedded-one-to-many-relationships-between-documents/>

# One-to-many relationships

heig-vd

Haute Ecole d'Ingénierie et de Gestion  
du Canton de Vaud

ok if if have few books per publisher

```
{
  title: "MongoDB: The Definitive Guide",
  author: [ "Kristina Chodorow", "Mike Dirolf" ],
  published_date: ISODate("2010-09-24"),
  pages: 216,
  language: "English",
  publisher: {
    name: "O'Reilly Media",
    founded: 1980,
    location: "CA"
  }
}

{
  title: "50 Tips and Tricks for MongoDB Developer",
  author: "Kristina Chodorow",
  published_date: ISODate("2011-05-06"),
  pages: 68,
  language: "English",
  publisher: {
    name: "O'Reilly Media",
    founded: 1980,
    location: "CA"
  }
}
```

duplication

```
{
  name: "O'Reilly Media",
  founded: 1980,
  location: "CA",
  books: [123456789, 234567890, ...]
}

{
  _id: 123456789,
  title: "MongoDB: The Definitive Guide",
  author: [ "Kristina Chodorow", "Mike Dirolf" ],
  published_date: ISODate("2010-09-24"),
  pages: 216,
  language: "English"
}

{
  _id: 234567890,
  title: "50 Tips and Tricks for MongoDB Developer",
  author: "Kristina Chodorow",
  published_date: ISODate("2011-05-06"),
  pages: 68,
  language: "English"
}
```

```
{
  _id: "oreilly",
  name: "O'Reilly Media",
  founded: 1980,
  location: "CA"
}

{
  _id: 123456789,
  title: "MongoDB: The Definitive Guide",
  author: [ "Kristina Chodorow", "Mike Dirolf" ],
  published_date: ISODate("2010-09-24"),
  pages: 216,
  language: "English",
  publisher_id: "oreilly"
}

{
  _id: 234567890,
  title: "50 Tips and Tricks for MongoDB Developer",
  author: "Kristina Chodorow",
  published_date: ISODate("2011-05-06"),
  pages: 68,
  language: "English",
  publisher_id: "oreilly"
}
```

better if you have many books per publisher

<https://docs.mongodb.org/manual/tutorial/model-embedded-one-to-many-relationships-between-documents/>

# Relationships with Mongoose

```
var PublisherSchema = new Schema({  
  name: String  
});
```

Manual reference

```
var BookSchema = new Schema({  
  title: String,  
  publisherName: String  
});
```

Reference through ObjectId

```
var BookSchema = new Schema({  
  title: String,  
  publisherId: Schema.Types.ObjectId  
});
```

Embedding

```
var BlogArticleSchema = new Schema({  
  title: String,  
  comments: [  
    {  
      body: String,  
      date: Date  
    }  
  ]  
});
```

# Insert data in MongoDB

- To insert data in MongoDB, you simply have to provide a **JSON document** (with an **arbitrary structure**).
- The documents in the collection do not have to all have the same structure (this is why we talk about a **schemaless** database).

A document is always inserted  
in a specific collection.

MongoDB will assign a unique  
ID in the **\_id** field for the new  
document.

Collection  
↓  
`db.users.insert(`

Document  
↓  
{  
  name: "sue",  
  age: 26,  
  status: "A",  
  groups: [ "news", "sports" ]  
}

)

Document

```
{  
  name: "sue",  
  age: 26,  
  status: "A",  
  groups: [ "news", "sports" ]  
}
```

insert

Collection

{ name: "al", age: 18, ... }
{ name: "lee", age: 28, ... }
{ name: "jan", age: 21, ... }
{ name: "kai", age: 38, ... }
{ name: "sam", age: 18, ... }
{ name: "mel", age: 38, ... }
{ name: "ryan", age: 31, ... }
{ name: "sue", age: 26, ... }

users

# Insert data in MongoDB

- This is one example. You can also insert multiple documents at the same time, either by passing an array of documents or by performing a bulk operation. If you are dealing with many documents, this is important for performance reasons.

```
db.inventory.insert(  
  {  
    item: "ABC1",  
    details: {  
      model: "14Q3",  
      manufacturer: "XYZ Company"  
    },  
    stock: [ { size: "S", qty: 25 }, { size: "M", qty: 50 } ],  
    category: "clothing"  
  }  
)
```

```
db.inventory.find()
```

```
{ "_id" : ObjectId("53d98f133bb604791249ca99"), "item" : "AB
```

# Update and delete data in MongoDB

- When you update or delete documents, you specify which documents are concerned by the operation.
- You do that by specifying update or remove criteria. Specifying "{}" means that you want to apply the operation on all documents of the collection.

```
db.users.update(  
  { age: { $gt: 18 } },  
  { $set: { status: "A" } },  
  { multi: true }  
)
```

← collection  
← update criteria  
← update action  
← update option

```
db.users.remove(  
  { status: "D" }  
)
```

← collection  
← remove criteria



# Update data in MongoDB

- By default, update will modify only the first document that matches the selection criteria. You can specify the "multi" options if you want to update all documents that match the criteria.

```
db.inventory.update(  
  { item: "MNO2" },  
  {  
    $set: {  
      category: "apparel",  
      details: { model: "14Q3", manufacturer: "XYZ Company" }  
    },  
    $currentDate: { lastModified: true }  
  }  
)
```

selection criteria

update operators

lastModified will be set to the  
current date

```
db.inventory.update(  
  { category: "clothing" },  
  {  
    $set: { category: "apparel" },  
    $currentDate: { lastModified: true }  
  },  
  { multi: true }  
)
```

# Update data in MongoDB

heig-vd

Haute Ecole d'Ingénierie et de Gestion  
du Canton de Vaud

## Update Operators

### Fields

Name	Description
<code>\$inc</code>	Increments the value of the field by the specified amount.
<code>\$mul</code>	Multiplies the value of the field by the specified amount.
<code>\$rename</code>	Renames a field.
<code>\$setOnInsert</code>	Sets the value of a field if an update results in an insert of a document. Has no effect on update operations that modify existing documents.
<code>\$set</code>	Sets the value of a field in a document.
<code>\$unset</code>	Removes the specified field from a document.
<code>\$min</code>	Only updates the field if the specified value is less than the existing field value.
<code>\$max</code>	Only updates the field if the specified value is greater than the existing field value.
<code>\$currentDate</code>	Sets the value of a field to current date, either as a Date or a Timestamp.

## Array

### Operators

Name	Description
<code>\$</code>	Acts as a placeholder to update the first element that matches the query condition in an update.
<code>\$addToSet</code>	Adds elements to an array only if they do not already exist in the set.
<code>\$pop</code>	Removes the first or last item of an array.
<code>\$pullAll</code>	Removes all matching values from an array.
<code>\$pull</code>	Removes all array elements that match a specified query.
<code>\$pushAll</code>	<i>Deprecated.</i> Adds several items to an array.
<code>\$push</code>	Adds an item to an array.

<https://docs.mongodb.org/manual/reference/operator/update/>

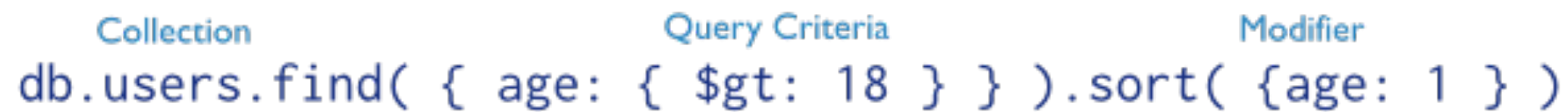
# Query MongoDB

- MongoDB is one of the most popular **NoSQL** databases (and one of the first to have been categorized as such).

Collection  
`db.users.find(`

Query Criteria  
`{ age: { $gt: 18 } } ).sort(`

Modifier  
`{age: 1 } )`



{ age: 18, ... }
{ age: 28, ... }
{ age: 21, ... }
{ age: 38, ... }
{ age: 18, ... }
{ age: 38, ... }
{ age: 31, ... }

users

Query Criteria

{ age: 28, ... }
{ age: 21, ... }
{ age: 38, ... }
{ age: 38, ... }
{ age: 31, ... }

Modifier

{ age: 21, ... }
{ age: 28, ... }
{ age: 31, ... }
{ age: 38, ... }
{ age: 38, ... }

Results

We look for documents within one collection, within one database

We define the criteria for which documents should be considered, and which of their fields should be considered (projection)

We optionally sort the documents, limit the number of documents returned, etc.

<https://docs.mongodb.org/manual/core/crud-introduction/>

# Query MongoDB

```
db.inventory.find()
```

all documents

```
db.inventory.find( { type: "snacks" } )
```

documents, which have a field "type"  
with a value of "snacks"

```
db.inventory.find( { type: { $in: [ 'food', 'snacks' ] } } )
```

documents, which have a field "type" with have a value  
of "food" or "snacks"

```
db.inventory.find( { type: 'food', price: { $lt: 9.95 } } )
```

documents with a type field equal to "food" AND a price  
field with a value less than 9.95

```
db.inventory.find(
  {
    $or: [ { qty: { $gt: 100 } }, { price: { $lt: 9.95 } } ]
  }
)
```

documents where the quantity is  
more than 100 OR the price is  
less than 9.95

# Query MongoDB: arrays

```
{ _id: 5, type: "food", item: "aaa", ratings: [ 5, 8, 9 ] }  
{ _id: 6, type: "food", item: "bbb", ratings: [ 5, 9 ] }  
{ _id: 7, type: "food", item: "ccc", ratings: [ 9, 5, 8 ] }
```

```
db.inventory.find( { ratings: [ 5, 8, 9 ] } )
```

```
{ "_id" : 5, "type" : "food", "item" : "aaa", "ratings" : [ 5, 8,
```

Exact match on the entire array

```
db.inventory.find( { ratings: 5 } )
```

Return documents if the array  
contains a specific value

```
{ "_id" : 5, "type" : "food", "item" : "aaa", "ratings" : [ 5, 8, 9 ] }  
{ "_id" : 6, "type" : "food", "item" : "bbb", "ratings" : [ 5, 9 ] }  
{ "_id" : 7, "type" : "food", "item" : "ccc", "ratings" : [ 9, 5, 8 ] }
```

# Query MongoDB: arrays

```
{ _id: 5, type: "food", item: "aaa", ratings: [ 5, 8, 9 ] }  
{ _id: 6, type: "food", item: "bbb", ratings: [ 5, 9 ] }  
{ _id: 7, type: "food", item: "ccc", ratings: [ 9, 5, 8 ] }
```

```
db.inventory.find( { ratings: { $elemMatch: { $gt: 5, $lt: 9 } } } )
```

Documents where one element of ratings is at the same time  $> 5$  AND  $< 9$

```
{ "_id" : 5, "type" : "food", "item" : "aaa", "ratings" : [ 5, 8, 9 ] }  
{ "_id" : 7, "type" : "food", "item" : "ccc", "ratings" : [ 9, 5, 8 ] }
```

```
db.inventory.find( { ratings: { $gt: 5, $lt: 9 } } )
```

Documents where there is one element of ratings  $> 5$  and one element  $< 9$

```
{ "_id" : 5, "type" : "food", "item" : "aaa", "ratings" : [ 5, 8, 9 ] }  
{ "_id" : 6, "type" : "food", "item" : "bbb", "ratings" : [ 5, 9 ] }  
{ "_id" : 7, "type" : "food", "item" : "ccc", "ratings" : [ 9, 5, 8 ] }
```



# SQL vs MongoDB

heig-vd

Haute Ecole d'Ingénierie et de Gestion  
du Canton de Vaud

<pre>SELECT * FROM users WHERE status != "A"</pre>	<pre>db.users.find(   { status: { \$ne: "A" } } )</pre>	<pre>SELECT * FROM users WHERE status = "A" ORDER BY user_id ASC</pre>	<pre>db.users.find( { status: "A" } ).sort( { user_id: 1 } )</pre>
<pre>SELECT * FROM users WHERE status = "A" AND age = 50</pre>	<pre>db.users.find(   { status: "A",     age: 50 } )</pre>	<pre>SELECT * FROM users WHERE status = "A" ORDER BY user_id DESC</pre>	<pre>db.users.find( { status: "A" } ).sort( { user_id: -1 } )</pre>
<pre>SELECT * FROM users WHERE status = "A" OR age = 50</pre>	<pre>db.users.find(   { \$or: [ { status: "A" } ,            { age: 50 } ] } )</pre>	<pre>SELECT COUNT(*) FROM users</pre>	<pre>db.users.count()  or  db.users.find().count()</pre>
<pre>SELECT * FROM users WHERE age &gt; 25</pre>	<pre>db.users.find(   { age: { \$gt: 25 } } )</pre>	<pre>SELECT COUNT(user_id) FROM users</pre>	<pre>db.users.count( { user_id: { \$exists: true } } )  or  db.users.find( { user_id: { \$exists: true } } ).count()</pre>
<pre>SELECT * FROM users WHERE age &lt; 25</pre>	<pre>db.users.find(   { age: { \$lt: 25 } } )</pre>	<pre>SELECT COUNT(*) FROM users WHERE age &gt; 30</pre>	<pre>db.users.count( { age: { \$gt: 30 } } )  or  db.users.find( { age: { \$gt: 30 } } ).count()</pre>
<pre>SELECT * FROM users WHERE age &gt; 25 AND age &lt;= 50</pre>	<pre>db.users.find(   { age: { \$gt: 25, \$lte: 50 } } )</pre>	<pre>SELECT DISTINCT(status) FROM users</pre>	<pre>db.users.distinct( "status" )</pre>
<pre>SELECT * FROM users WHERE user_id like "%bc%"</pre>	<pre>db.users.find( { user_id: /bc/ } )</pre>		

<https://docs.mongodb.org/manual/reference/sql-comparison/>

# Query MongoDB: projections

- When you perform a query, you can specify which fields you are interested in (think about performance)

```
db.inventory.find( { type: 'food' } )
```

Return all fields (no second argument)

```
db.inventory.find( { type: 'food' }, { item: 1, qty: 1 } )
```

We are only interested by the item and qty fields (we will also get \_id)

```
db.inventory.find( { type: 'food' }, { item: 1, qty: 1, _id: 0 } )
```

We are only interested by the item and qty fields (we really don't want to get \_id)

```
db.inventory.find( { type: 'food' }, { type: 0 } )
```

We want all fields except type

<https://docs.mongodb.org/manual/tutorial/project-fields-from-query-results/>