

# Document Stores – Hands On

23D020: Big Data Management for Data Science  
Barcelona School of Economics

# Session goals

- Get familiar with MongoDB
  - Set up a MongoDB server
  - MongoDB Shell
  - MongoDB Python API
  - Aggregation Framework
  - Geospatial queries
- Understand the implication of different models
  - Normalization vs denormalization

# MongoDB Tools

# MongoDB tools

- **MongoDB Community Server**

- The Community Edition of the MongoDB distributed document database

- **MongoDB Shell**

- A command line interface that allows to work (i.e., query and update) with your database

- **MongoDB Compass**

- A graphical user interface for MongoDB

- **MongoDB Tools**

- Command line utilities for working with MongoDB

# Setting up MongoDB (1/3)

## MongoDB Community Server

- <https://www.mongodb.com/try/download/community>
- Run the server
  - `./mongod -dbpath folder`
  - `folder`: to store the data, you can create your own folder and relate to it, or use the default one `/data/db`, but you need to make sure the latter exists with the right read/write permissions.

# Setting up MongoDB (1/4)

## MongoDB Community Server

```
bin — mongod --dbpath data/db — 142x49
besim@omega-93-235 bin % ./mongod --dbpath data/db
{"t":{"$date":"2024-02-16T08:09:08.803+01:00"},"s":"I", "c":"CONTROL", "id":23285, "ctx":"thread1","msg":"Automatically disabling TLS 1.0,
to force-enable TLS 1.0 specify --sslDisabledProtocols 'none'"}
{"t":{"$date":"2024-02-16T08:09:08.804+01:00"},"s":"I", "c":"NETWORK", "id":4915701, "ctx":"thread1","msg":"Initialized wire specification",
"attr":{"spec":{"incomingExternalClient":{"minWireVersion":0,"maxWireVersion":21},"incomingInternalClient":{"minWireVersion":0,"maxWireVersion":
21},"outgoing":{"minWireVersion":6,"maxWireVersion":21},"isInternalClient":true}}}}
{"t":{"$date":"2024-02-16T08:09:08.805+01:00"},"s":"I", "c":"NETWORK", "id":4648602, "ctx":"thread1","msg":"Implicit TCP FastOpen in use."}
{"t":{"$date":"2024-02-16T08:09:08.806+01:00"},"s":"I", "c":"REPL", "id":5123008, "ctx":"thread1","msg":"Successfully registered PrimaryO
nlyService", "attr":{"service":"TenantMigrationDonorService","namespace":"config.tenantMigrationDonors"}}
{"t":{"$date":"2024-02-16T08:09:08.806+01:00"},"s":"I", "c":"REPL", "id":5123008, "ctx":"thread1","msg":"Successfully registered PrimaryO
nlyService", "attr":{"service":"TenantMigrationRecipientService","namespace":"config.tenantMigrationRecipients"}}
{"t":{"$date":"2024-02-16T08:09:08.806+01:00"},"s":"I", "c":"CONTROL", "id":5945603, "ctx":"thread1","msg":"Multi threading initialized"}
{"t":{"$date":"2024-02-16T08:09:08.806+01:00"},"s":"I", "c":"TENANT_M", "id":7091600, "ctx":"thread1","msg":"Starting TenantMigrationAccessBl
ockerRegistry"}
{"t":{"$date":"2024-02-16T08:09:08.806+01:00"},"s":"I", "c":"CONTROL", "id":4615611, "ctx":"initandlisten","msg":"MongoDB starting","attr":{"
pid":3590,"port":27017,"dbPath":"data/db","architecture":"64-bit","host":"omega-93-235.lsi.upc.edu"}}
{"t":{"$date":"2024-02-16T08:09:08.806+01:00"},"s":"I", "c":"CONTROL", "id":23403, "ctx":"initandlisten","msg":"Build Info","attr":{"build
Info":{"version":"7.0.5","gitVersion":"7809d71e84e314b497f282ea8aa06d7ded3eb205","modules":[],"allocator":"system","environment":{"distarch":"
aarch64","target_arch":"aarch64"}}}}
{"t":{"$date":"2024-02-16T08:09:08.806+01:00"},"s":"I", "c":"CONTROL", "id":51765, "ctx":"initandlisten","msg":"Operating System","attr":{"
os":{"name":"Mac OS X","version":"23.2.0"}}}
{"t":{"$date":"2024-02-16T08:09:08.806+01:00"},"s":"I", "c":"CONTROL", "id":21951, "ctx":"initandlisten","msg":"Options set by command lin
e","attr":{"options":{"storage":{"dbPath":"data/db"}}}}
{"t":{"$date":"2024-02-16T08:09:08.806+01:00"},"s":"I", "c":"NETWORK", "id":5693100, "ctx":"initandlisten","msg":"Asio socket.set_option fai
led with std::system_error","attr":{"note":"acceptor TCP fast open","option":{"level":6,"name":261,"data":"00 04 00 00"},"error":{"what":"set_
option: Invalid argument","message":"Invalid argument","category":"asio.system","value":22}}}
{"t":{"$date":"2024-02-16T08:09:08.806+01:00"},"s":"I", "c":"STORAGE", "id":22315, "ctx":"initandlisten","msg":"Opening WiredTiger","attr"
:{"config":"create,cache_size=7680M,session_max=33000,eviction=(threads_min=4,threads_max=4),config_base=false,statistics=(fast),log=(enabled=
true,remove=true,path=journal,compressor=snappy),builtin_extension_config=(zstd=(compression_level=6)),file_manager=(close_idle_time=600,close
_scan_interval=10,close_handle_minimum=2000),statistics_log=(wait=0),json_output=(error,message),verbose=[recovery_progress:1,checkpoint_progr
ess:1,compact_progress:1,backup:0,checkpoint:0,compact:0,evict:0,history_store:0,recovery:0,rts:0,salvage:0,tiered:0,timestamp:0,transaction:0
,verify:0,log:0],"}
{"t":{"$date":"2024-02-16T08:09:08.954+01:00"},"s":"I", "c":"STORAGE", "id":4795906, "ctx":"initandlisten","msg":"WiredTiger opened","attr":
{"durationMillis":148}}
{"t":{"$date":"2024-02-16T08:09:08.954+01:00"},"s":"I", "c":"RECOVERY", "id":23987, "ctx":"initandlisten","msg":"WiredTiger recoveryTimesta
mp","attr":{"recoveryTimestamp":{"timestamp":{"t":0,"i":0}}}}
{"t":{"$date":"2024-02-16T08:09:08.991+01:00"},"s":"W", "c":"CONTROL", "id":22120, "ctx":"initandlisten","msg":"Access control is not enab
led for the database. Read and write access to data and configuration is unrestricted","tags":["startupWarnings"]}
{"t":{"$date":"2024-02-16T08:09:08.991+01:00"},"s":"W", "c":"CONTROL", "id":22140, "ctx":"initandlisten","msg":"This server is bound to lo
calhost. Remote systems will be unable to connect to this server. Start the server with --bind_ip <address> to specify which IP addresses it s
hould serve responses from, or with --bind_ip_all to bind to all interfaces. If this behavior is desired, start the server with --bind_ip 127.
0.0.1 to disable this warning","tags":["startupWarnings"]}
{"t":{"$date":"2024-02-16T08:09:08.991+01:00"},"s":"W", "c":"CONTROL", "id":22184, "ctx":"initandlisten","msg":"Soft rlimits for open file
descriptors too low","attr":{"currentValue":256,"recommendedMinimum":64000},"tags":["startupWarnings"]}
{"t":{"$date":"2024-02-16T08:09:08.991+01:00"},"s":"I", "c":"STORAGE", "id":20320, "ctx":"initandlisten","msg":"createCollection","attr":{"
namespace":"admin.system.version","uuidDisposition":"provided","uuid":{"uuid":{"$uuid":"06639d85-b0d8-42d3-8584-ddfbb02d81d0"},"options":{"u
uid":{"$uuid":"06639d85-b0d8-42d3-8584-ddfbb02d81d0"}}}}}}
{"t":{"$date":"2024-02-16T08:09:09.011+01:00"},"s":"I", "c":"INDEX", "id":20345, "ctx":"initandlisten","msg":"Index build: done building
","attr":{"buildUUID":null,"collectionUUID":{"uuid":{"$uuid":"06639d85-b0d8-42d3-8584-ddfbb02d81d0"},"namespace":"admin.system.version","inde
```

# Setting up MongoDB (2/4)

## MongoDB Shell

- The database shell (interaction with the server)
- If not installed, download it from here
  - <https://www.mongodb.com/try/download/shell>
- With the server running, run the shell on a separate terminal
  - `./mongosh --host --port`
  - `--host`: defaults to `127.0.0.1`
  - `--port`: defaults to `27017`

# Setting up MongoDB (2/4)

## MongoDB Shell

```
bin — mongosh mongodb://127.0.0.1:27017/?directConnection=true&serverSelectionTimeoutMS=2000 — mongosh m — 12...
besim@omega-93-235 bin % ./mongosh
Current Mongosh Log ID: 65cf0ec63332097bf963872f
Connecting to:      mongodb://127.0.0.1:27017/?directConnection=true&serverSelectionTimeoutMS=2000&appName=mongosh+2.1.4
Using MongoDB:      7.0.5
Using Mongosh:      2.1.4

For mongosh info see: https://docs.mongodb.com/mongosh-shell/

To help improve our products, anonymous usage data is collected and sent to MongoDB periodically (https://www.mongodb.com/legal/privacy-policy).
You can opt-out by running the disableTelemetry() command.

-----
  The server generated these startup warnings when booting
  2024-02-16T08:09:08.991+01:00: Access control is not enabled for the database. Read and write access to data and configuration is unrestricted
  2024-02-16T08:09:08.991+01:00: This server is bound to localhost. Remote systems will be unable to connect to this server. Start the server with --bind_ip <address> to specify which IP addresses it should serve responses from, or with --bind_ip_all to bind to all interfaces. If this behavior is desired, start the server with --bind_ip 127.0.0.1 to disable this warning
  2024-02-16T08:09:08.991+01:00: Soft rlimits for open file descriptors too low
  -----

test> show dbs
admin    40.00 KiB
config  12.00 KiB
local   40.00 KiB
test> 
```



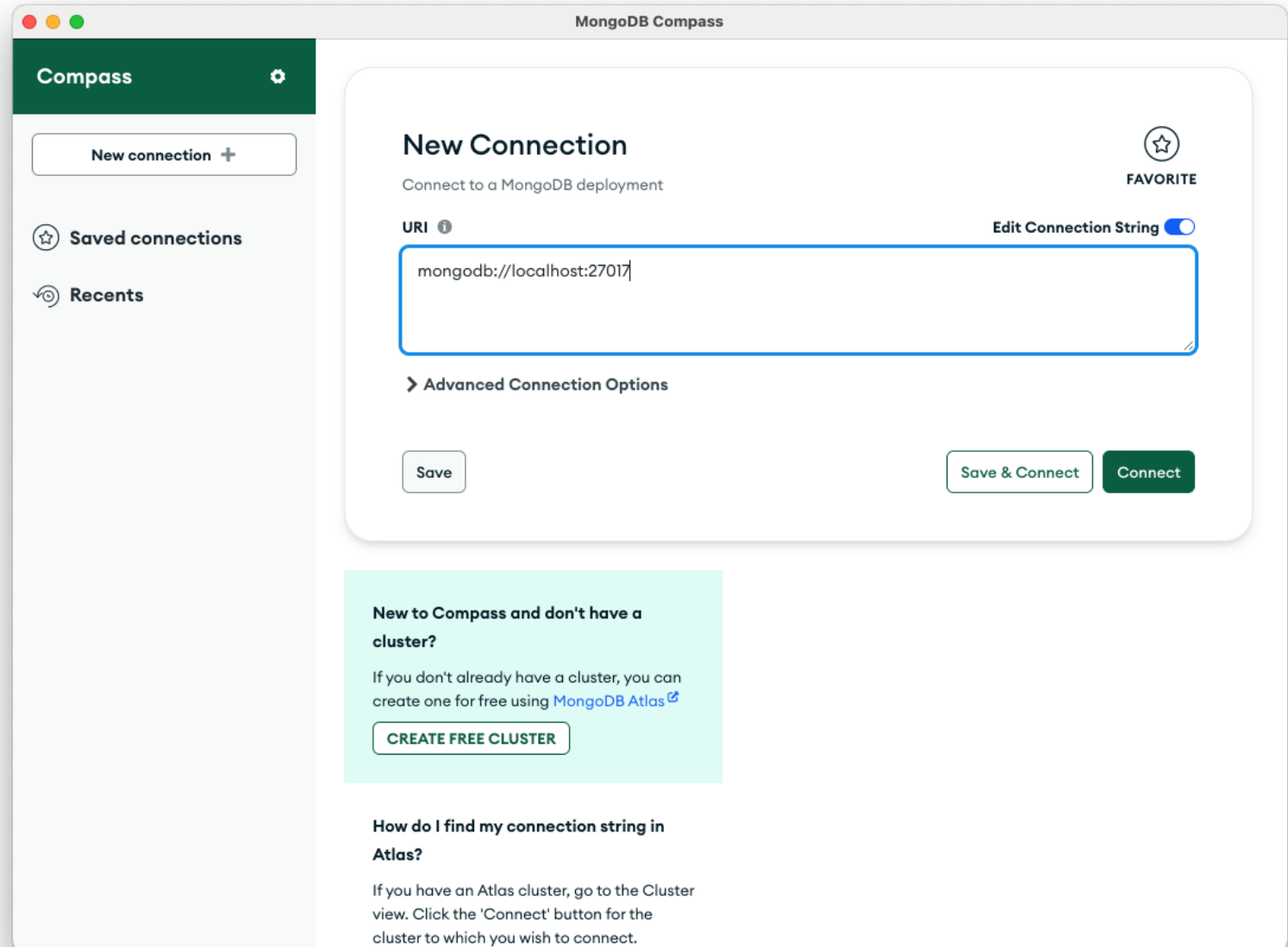
# Setting up MongoDB (3/4)

## MongoDB Compass

- Interactive tool for querying, optimizing, and analyzing MongoDB data
- Link to download
  - <https://www.mongodb.com/try/download/compass>
- Execute it as a usual application in the system

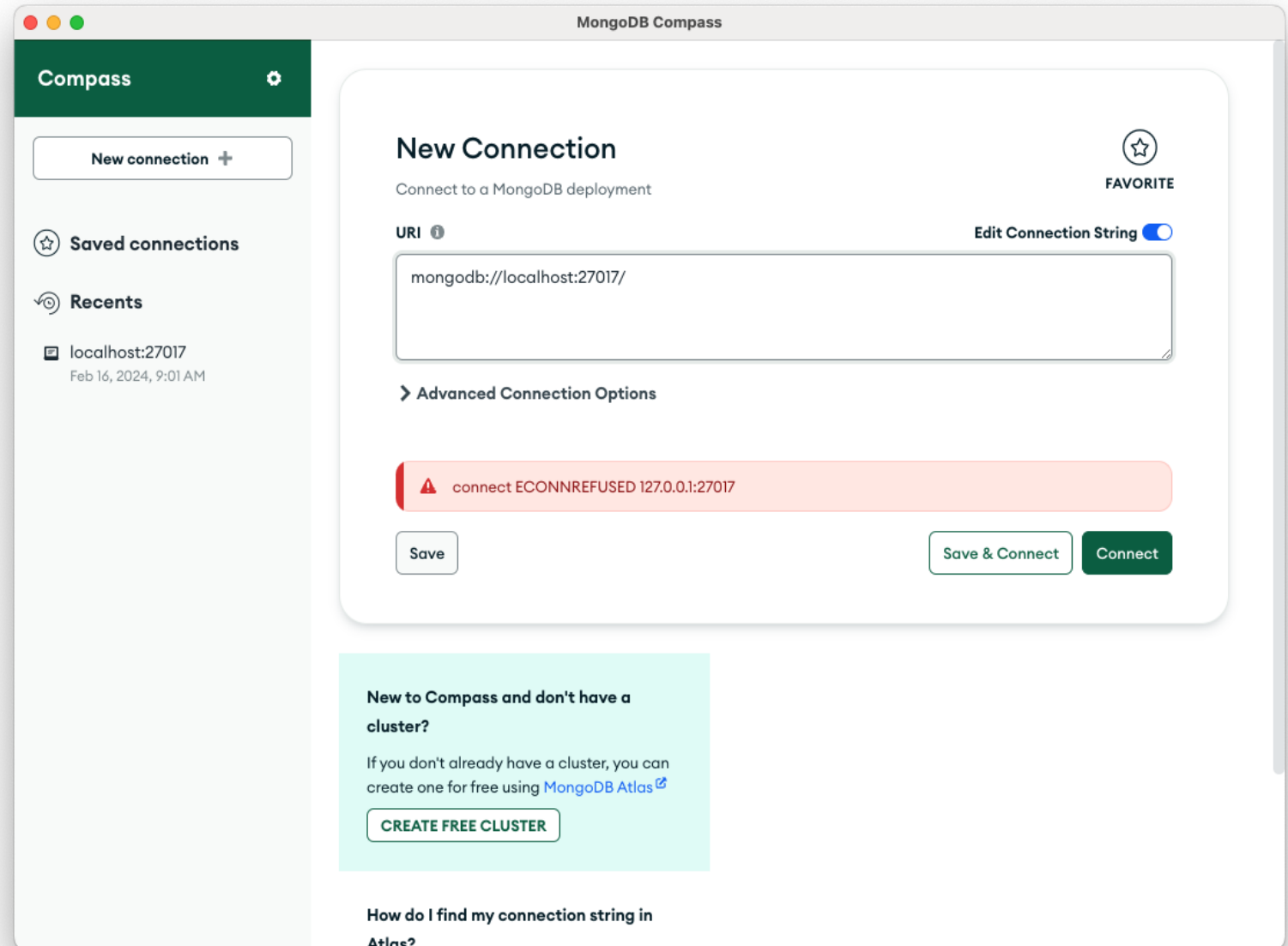
# Setting up MongoDB (3/4)

## MongoDB Compass



# Setting up MongoDB (3/4)

## MongoDB Compass



# Setting up MongoDB (4/4)

## MongoDB Database Tools

- Collection of command-line utilities for working with a MongoDB deployment
- Download link
  - <https://www.mongodb.com/try/download/database-tools>
- Tools provided
  - `./mongoimport`
  - `./mongoexport`
  - `./mongodump`
  - `./mongostat`
  - ...

# Querying in MongoDB

# Documents and collections

- Documents are ordered set of keys with associated values
  - Most languages have a data structure that is a natural fit:
    - Map, Hash, Dictionary, Object → JSON (BSON)  
`{"greeting" : "Hello, world!", "foo" : 3}`
- Keys in a document must be Strings
  - No duplicate keys
- Documents (rows) are grouped into Collections (tables)
  - Collections are Schemaless

# Documents and collections

- Basic Data Types
  - null, Boolean, Integer, String, Binary, ...
- Complex Data Types
  - Array, Embedded Documents, Date, ...
- Special Data Types
  - ObjectIds : "\_id"
    - Unique value representing the document key
    - 12 bytes

0	1	2	3	4	5	6	7	8	9	10	11
Timestamp				Machine			PID		Increment		

## Example: Student document

```
{
  "_id": ObjectId
    ("507f1f77bcf86cd799439011")
  "name": "Marc"
  "courses": [
    "DB", "BDM", "DEBD"
  ]
  "address": {
    "street": "Carrer de Sants"
    "number": 19
    "city": "Barcelona"
  }
}
```

# MongoDB syntax

Global variable  
↓  
db.

Query-by-example  
(Depending on the method:  
document, array of documents, etc.)  
↓  
[query]

```
db.[collection-name].[method]([query],[options])
```

- **Collection methods:** insert, update, remove, find, ...

```
db.restaurants.find({"name": "x"})
```

- **Cursor methods:** forEach, hasNext, count, sort, skip, size, ...

```
db.restaurants.find({"name": "x"}).count()
```

- **Database methods:** createCollection, copyDatabase, ...

```
db.createCollection("collection-name")
```

- ...



# Syntax - operators

- Query and Projection operators
  - Comparison ops: \$eq, \$gt, \$gte, \$in, \$nin, \$lt, \$lte  
`{field: {$eq: <value>}}`
  - Logical ops: \$and, \$not, \$nor, \$or  
`{$and: [{<expression1>},...,{<expressionN>}]}`
  - Array ops: \$push, \$pull, \$addToSet, \$all, \$elemMatch  
`{<field>:{$elemMatch:{<query1>,<query2>,...}}}`
- Update operators: \$set, \$inc, \$min, \$max, ...
- ...

# Insert Documents

- Insert one or more documents to the people collection  
`db.users.insertOne({"name":"Sergi"})` ← document  
`db.users.insertMany([{"name":"Sergi"}, {"name":...}])` ← array
- Insert will add an "\_id" key to the document (if you do not provide one)
  - You can use a custom id, by adding a field "\_id" to inserted documents
- MongoDB checks that the document does not exceed 16MB

# Delete Documents

- Removing by example

a filter document  
is the first parameter



```
db.users.deleteOne({"name": "Sergi"})
```

← deletes the first document  
that matches the filter

```
db.users.deleteMany({"name": "Sergi"})
```

← deletes all docs  
that match the filter

- Remove all documents. Collection and indexes remain intact!

```
db.users.deleteMany({})
```

- Drop a collection; it is faster. You need to recreate indexes!

```
db.users.drop()
```

# Modify/Update Document Schema

mongo shell

```
{
  "_id" : ObjectId("4b2b9f67a1f631733d917a7a"),
  "name" : "joe",
  "friends" : 32,
  "enemies" : 2
}
```



```
{
  "_id" : ObjectId("4b2b9f67a1f631733d917a7a"),
  "username" : "joe",
  "relationships" :
    {
      "friends" : 32,
      "enemies" : 2
    }
}
```

Get document to update

```
> var joe = db.users.findOne({"name":"joe"});
```

Create "username" / Delete "name"

```
> joe.username = joe.name;
```

```
> delete joe.name;
```

Create relationships / Delete "friends" and "enemies"

```
> joe.relationships={"friends":joe.friends,"enemies":joe.enemies};
```

```
> delete joe.friends;
```

```
> delete joe.enemies;
```

Store updated document

```
> db.users.replaceOne({"name":"joe"},joe)
```

make sure that your update always specifies a unique document, e.g., use `_id`

# Update Document Values

- Looks for docs that match the query and updates spec. fields

```
db.users.updateMany({"name":"Sergi"}),
```

Update operator

→ `{ $set: { "address": "..."} }` ← Update the field or create it

- Update only the first doc that matches the query

```
db.collection.updateOne({filterDoc},{updateDoc})
```

- “Upsert option” `updateOne({...},{...},{ "upsert" : true })`
  - If no document matches the query, it creates a new document
  - Should be used with caution

# Update Arrays

- “\$push” operator adds elements to the end of an array if the array exists and creates a new array if it does not

```
db.users.updateOne({"name":"Sergi"},  
  {"$push":{"emails":{"email":"sn@php.net"}}})
```

“modifier”, push multiple elements in one operation

```
...{"$push" : {"emails" : {"$each" : ["a","b","c"]}}})
```

- “addToSet” operator in combination with the “\$each” modifier adds multiple **unique** values

```
db.users.updateOne({"name":"Sergi"},  
  {"$addToSet" : {"emails" : {"$each" : ["sn@php.net", "sn@example.com",  
  "sne@python.org"]}}
```

# Query Documents

- Find all documents in a collection  
`db.users.find()`
- Query by example (build a document that looks similar to the one you want)

```
db.users.find({"username":"oscar"})
```

- Attribute projection, specify which keys to return

```
> db.users.find({}, {"username" : 1, "email" : 1})
{
  "_id" : ObjectId("4ba0f0dfd22aa494fd523620"),
  "username" : "joe",
  "email" : "joe@example.com"
}
```

To prevent "\_id" from being returned: ...{..., "\_id":0}

# Query Documents (II)

Operators: comparison, logical,  
array, geospatial, etc

- Conditionals

- Find users with age between 18 and 30

`db.users.find({"age": {"$gte":18, "$lte":30}})`

- Also available `$lt`, `$lte` and `$gt`

- OR query (`$or/$in/$nin`)

`db.users.find({"user_id": {"$nin": [12345, "joe"]}})`

`db.users.find({"$or": [{"age":18}, {"winner":true}]})`



# Aggregation Framework (Pipeline)

Document: Customer

```
{
  cust_id: "A123",
  amount: 100,
  status: "A"
}
```

*Pipeline stages:* (\$match, \$group, \$addfields, \$sort, \$unwind ...)

db.orders.aggregate([

*The name of the  
computed/accumulator field*

**\$match:** {status: "A"}},

**{ \$group: { \_id: "\$cust\_id", total: { \$sum: "\$amount" } } } ] )**

*Required field:  
to identify the field  
for the group by*

*References  
the field  
(field path)*

*Pipeline operators:  
\$sum, \$max, \$min ...*

# Aggregation Framework: Stages

- `$project` – adds or removes attributes for each document
- `$match` – filters the document stream
- `$limit` – passes the first n documents
- `$skip` – skips the first n documents
- `$group` – applies a groupBy function. Returns the grouper field and accumulator fields
- `$sort` – reorders the data pipe
- `$unwind` – deconstructs an array field and creates a doc for each arr. elem.
- ...

# Geospatial queries

# MongoDB Indexes

- `db.users.createIndex({"username":1})`
- An index can make a dramatic difference in query times
- Indexes are costly: write operations (inserts, update and deletes) that modify an indexed field will take longer
  - In addition to updating the document, MongoDB has to update indexes
- In which field to create an index:
  - What are your frequent queries? Which queries need to be fast?
  - Find a common set of keys from those
- Compound indexes: `db.users.createIndex({"age":1, "username":1})`

# Special Indexes

- Geospatial indexes for 2D and spherical geometries
  - **2dsphere** and **2d** indexes
- They work with spherical geometries that model the surface of the earth based on the WGS84 datum (earth as oblate spheroid)
- Distance calculations using 2dsphere indexes, take the shape of the earth into account and provide a more accurate treatment of distance
  - **2dsphere** allows you to specify geometries for points, lines, polygons in the GeoJSON format.

# Geospatial queries

- Perform geospatial operations on MongoDB documents

```
{
  "address": {
    "building": "1007",
    "coord": [-73.856077, 40.848447 ],
    "street": "Morris Park Ave",
    "zipcode": "10462"
  },
  "borough": "Bronx",
  "cuisine": "Bakery",
  "grades": [
    { "date": { "$date": 1393804800000 }, "grade": "A", "score": 2 },
    { "date": { "$date": 1378857600000 }, "grade": "A", "score": 6 },
    { "date": { "$date": 1358985600000 }, "grade": "A", "score": 10 },
    { "date": { "$date": 1322006400000 }, "grade": "A", "score": 9 },
    { "date": { "$date": 1299715200000 }, "grade": "B", "score": 14 }
  ],
  "name": "Morris Park Bake Shop",
  "restaurant_id": "30075445"
}
```

# GeoJSON documents

- A JSON standard to represent goespatial data
- To specify GeoJSON data, use an embedded document with two fields:
  - **type**: specifies GeoJSON geometry type
  - **coordinates**: specifies the object's coordinates

```
{  
  "name": "Biblioteca Vila de Gracia",  
  "type": "library",  
  "geometry": {  
    "type": "Point",  
    "coordinates": [2.158311862578671, 41.401512395604129 ]  
  }  
}
```

# GeoJSON Types

- Point

```
{ type: "Point", coordinates: [ 40, 5 ] }
```

- LineString

```
{ type: "LineString", coordinates: [ [ 40, 5 ], [ 41, 6 ] ] }
```

- Polygon

```
{  
  type: "Polygon",  
  coordinates: [ [ [ 0, 0 ], [ 3, 6 ], [ 6, 1 ], [ 0, 0 ] ] ]  
}
```



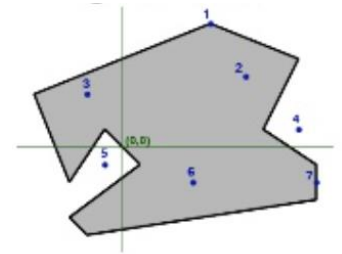
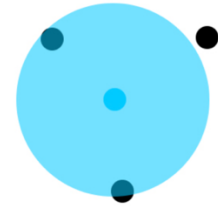
# Geospatial Indexes

- Used to calculate geometries on earth-like spheres
- A collection **must** have a spatial index before executing geospatial queries
- Creating a 2dsphere index:
  - > `db.collection.createIndex({geometry: "2dsphere"})`

Documentation: <https://docs.mongodb.com/manual/core/2dsphere/>

# Supported geospatial queries

- `$near`
  - Returns documents in proximity to a point
  - Optional: `$minDistance` and `$maxDistance`
- `$geoWithin`
  - Returns documents within a specified geometry
- `$geoIntersects`
  - Returns documents that intersect with a specified geometry



# Example

```
> var barcelona = {  
  "type": "Polygon",  
  "coordinates": [  
    [-73,40],  
    [-73.9,40.72],  
    ...  
  ]}
```

Find all point, lines and polygon containing documents that have a point in Barcelona

```
db.openStreetMaps.find("geometry":{"$geoIntersects": {"$geometry": barcelona}})
```

Location field

\$geometry operator

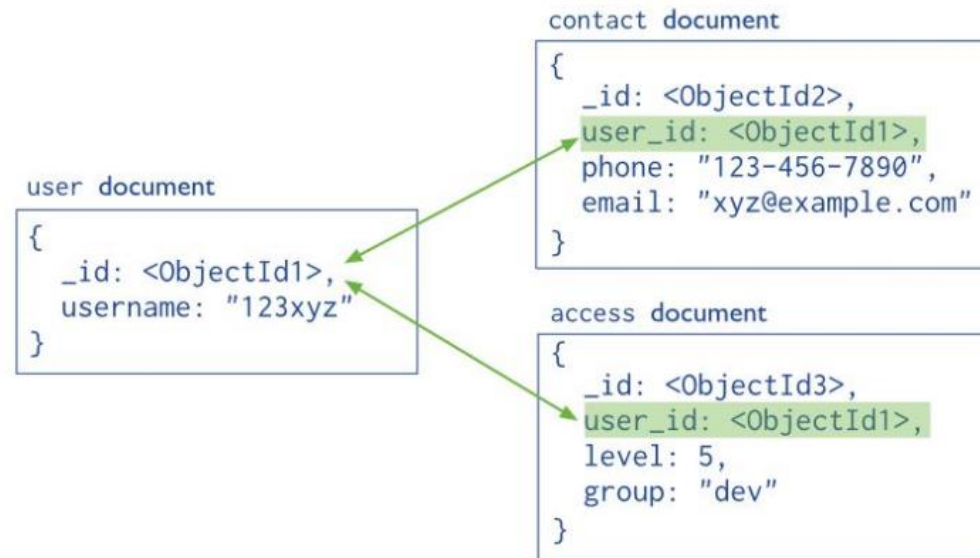
Find all point, lines and polygon containing documents that are completely contained in Barcelona

```
db.openStreetMaps.find("geometry":{"$geoWithin": {"$geometry": barcelona}})
```

# Modeling in MongoDB

# Modeling with MongoDB

- MongoDB allows different degrees of normalization
- Choice depends on information requirements (i.e., queries)
- Normalized model



# Denormalization options

- Embedded documents
  - Simplest way
  - Insert document as sub-document
- Move attrs to the root document
  - All data directly in one document
  - Simpler document structure

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

Embedded sub-document

Embedded sub-document

```
{
  _id: <ObjectId1>,
  username: "123xyz",
  contact_phone: "123-456-7890",
  contact_email: "xyz@example.com",
  access_level: 5
  access_group: "dev"
}
```

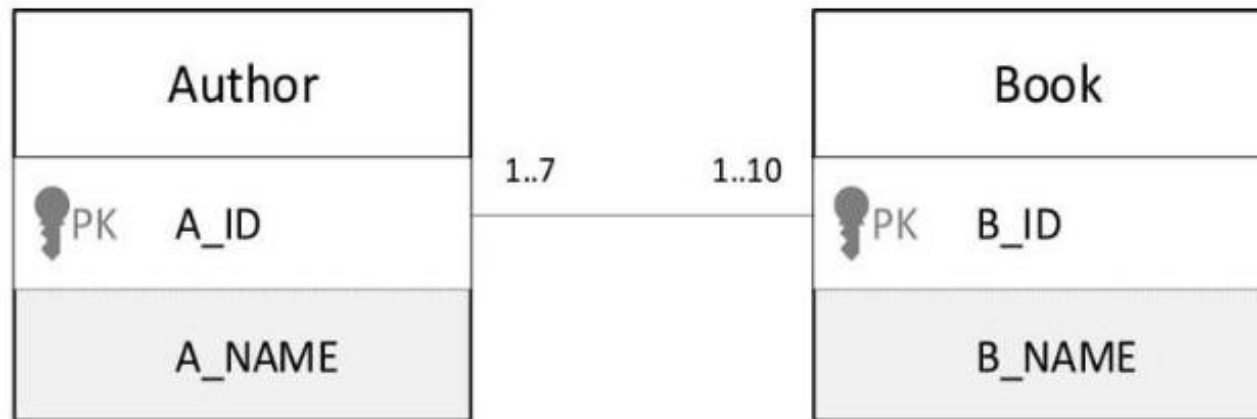
# Denormalization options

- Array of nested documents
  - Many sub documents related to root
  - JSON array of documents

```
{
  _id: <ObjectId>,
  username: "123xyz",
  contacts: [
    { type: "work", phone: "123-456-7890", email: "xyz@example.com" },
    { type: "home", phone: "098-765-4321", email: "xyz@home.com" },
  ]
  access_level: 5
  access_group: "dev"
}
```

# Modeling

- Given the conceptual schema. How many logical designs can you come up with?





<b>#1: Authors nested</b> <pre>Books:{   "B_ID":int,   "B_NAME": varchar,   "Authors" : [{     "A_ID": int,     "A_NAME": varchar,   }]} </pre>	<b>#2: Books nested</b> <pre>Authors:{   "A_ID":int,   "A_NAME": varchar,   "Books" : [{     "B_ID": int,     "B_NAME": varchar,   }]} </pre>	<b>#3: Both nested</b> <pre>Books:{   "B_ID":int,   "B_NAME": varchar,   "Authors" : [{     "A_ID": int,     "A_NAME": varchar,   }]} Authors:{   "A_ID":int,   "A_NAME": varchar,   "Books" : [{     "B_ID": int,     "B_NAME": varchar,   }]} </pre>	<b>#4: Authors referred</b> <pre>Books:{   "B_ID":int,   "B_NAME": varchar,   "Authors" : ["A_ID": int] }, Authors:{   "A_ID":int,   "A_NAME": varchar } </pre>
<b>#5: Books referred</b> <pre>Authors:{   "A_ID":int,   "A_NAME": varchar,   "Books" : ["B_ID": int] }, Books:{   "B_ID":int,   "B_NAME": varchar, } </pre>	<b>#6: Both referred</b> <pre>Authors:{   "A_ID":int,   "A_NAME": varchar,   "Books": ["B_ID": int] }, Books:{   "B_ID":int,   "B_NAME": varchar,   "Authors" : ["A_ID": int] } </pre>	<b>#7: Normalization</b> <pre>Authors:{   "A_ID":int,   "A_NAME": varchar }, Books:{   "B_ID":int,   "B_NAME": varchar }, Author_Book:{   "A_ID":int,   "B_ID": int } </pre>	

Which one is better?

Depends on the workload (queries)!



- Query: Find the book names, given the author identifier.

<b>#1: Authors nested</b> <pre>Books:{   "B_ID":int,   "B_NAME": varchar,   "Authors" : [{     "A_ID": int,     "A_NAME": varchar,   }]}</pre>	<b>#2: Books nested</b> ✓ <pre>Authors:{   "A_ID":int,   "A_NAME": varchar,   "Books" : [{     "B_ID": int,     "B_NAME": varchar,   }]}</pre>	<b>#3: Both nested</b> ✓ <pre>Books:{   "B_ID":int,   "B_NAME": varchar,   "Authors" : [{     "A_ID": int,     "A_NAME": varchar,   }]} Authors:{   "A_ID":int,   "A_NAME": varchar,   "Books" : [{     "B_ID": int,     "B_NAME": varchar,   }]}</pre>	<b>#4: Authors referred</b> <pre>Books:{   "B_ID":int,   "B_NAME": varchar,   "Authors" : ["A_ID": int] }, Authors:{   "A_ID":int,   "A_NAME": varchar }</pre>
<b>#5: Books referred</b> <pre>Authors:{   "A_ID":int,   "A_NAME": varchar,   "Books" : ["B_ID": int] }, Books:{   "B_ID":int,   "B_NAME": varchar, }</pre>	<b>#6: Both referred</b> <pre>Authors:{   "A_ID":int,   "A_NAME": varchar,   "Books": ["B_ID": int] }, Books:{   "B_ID":int,   "B_NAME": varchar,   "Authors" : ["A_ID": int] }</pre>	<b>#7: Normalization</b> <pre>Authors:{   "A_ID":int,   "A_NAME": varchar }, Books:{   "B_ID":int,   "B_NAME": varchar }, Author_Book:{   "A_ID":int,   "B_ID": int }</pre>	<pre> graph LR     Author[Author] -- "1..7" --- Book[Book]     subgraph Author         A_ID((A_ID))         A_NAME[A_NAME]     end     subgraph Book         B_ID((B_ID))         B_NAME[B_NAME]     end     style A_ID fill:none,stroke:none     style B_ID fill:none,stroke:none   </pre>

# MongoDB Python API

- Pymongo – Python driver for MongoDB

```
from pymongo import MongoClient
client = MongoClient('127.0.0.1:27017')
db = client['test']
collection = db.create_collection('example')
```

- Create object

```
d = {"x":1, "y":"foo"}
db.collection.insert_one(d)

for doc in db.collection.find({}):
    print(doc)
```

# Closing

- MongoDB Tools
  - Community Server
  - Mongo Shell
  - MongoDB Compass
  - MongoDB command line tools
- Querying in MongoDB
- Geospatial data
- Modeling data in MongoDB

# References

- K. Chodorow, M. Dirolf. MongoDB: The Definitive Guide, O'Reilly Media 1st edition (2010)
- MongoDB Manual
  - <https://docs.mongodb.org/manual/>
- Getting Started with MongoDB (Java)
  - <https://docs.mongodb.org/getting-started/java/>