uc3m | Universidad Carlos III de Madrid

OpenCourseWare

Database

3.3. Introduction to MongoDB

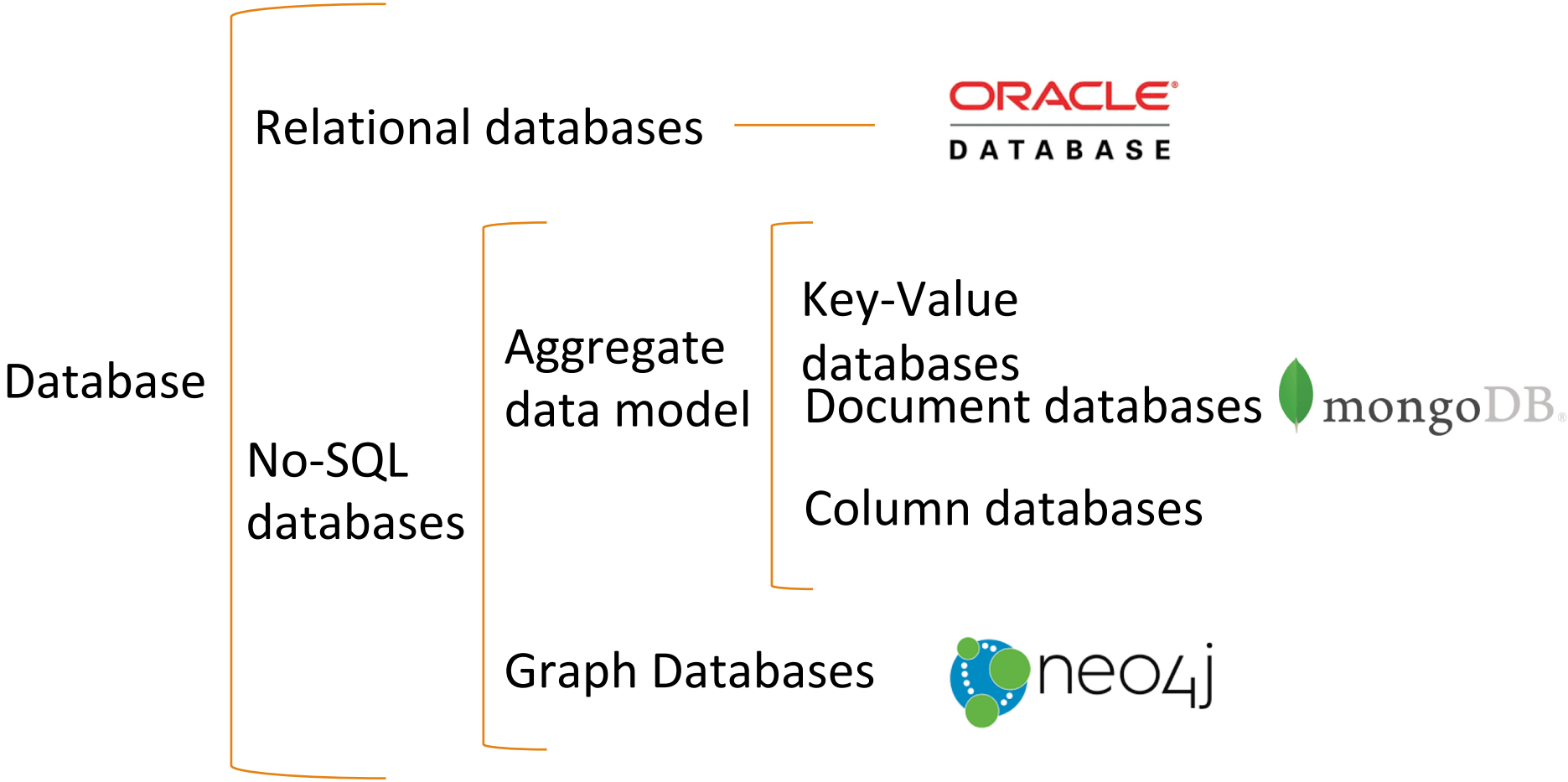
Lourdes Moreno López
Paloma Martínez Fernández
José Luis Martínez Fernández
Rodrigo Alarcón García

HULAT
Human Language & Accessibility Technologies

Departamento de Informática

Review

DBMS to study in Course "Database"

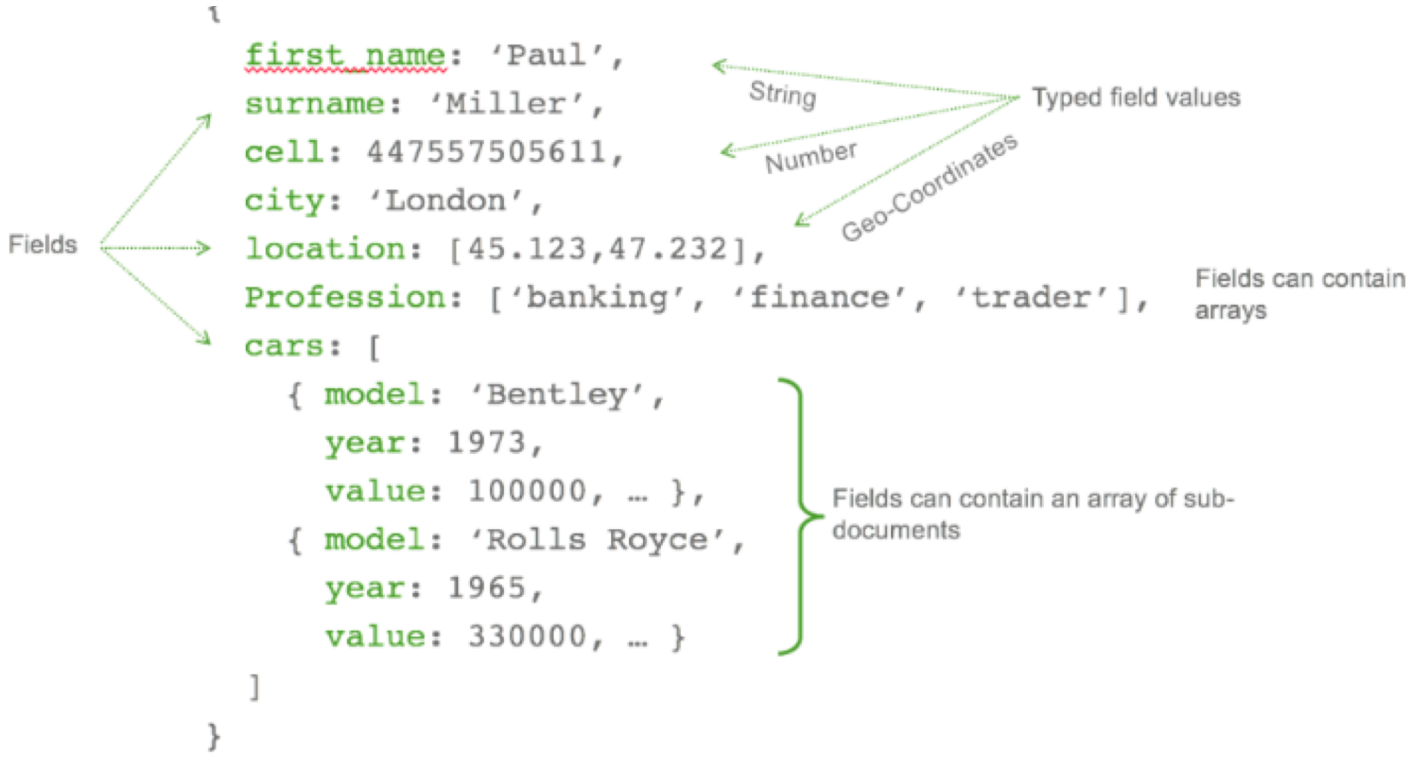


uc3m

- NoSQL database (Document-oriented database model) => Flexible data scheme
- JSON-like documents (http://bsonspec.org/).
  - JSON stored in binary format with some extensions (so it takes up less memory space
- General purpose database
- Open source database management system (DBMS)
- Multiplatform: available for Unix, Linux, Windows, and Mac
- Drivers for multiple programming languages
- Mongodb website: https://www.mongoDB.org/

JSON
JavaScript Object Notation

Document databases such as MongoDB use JSON documents in order to store records, just as tables and rows store records in a relational database



MongoDB Modeling Document Structure

- The key decision in designing data models for MongoDB applications revolves around the structure of documents and how the application represents relationships between data.
  - MongoDB allows related data to be **embedded** within a single document.
  - MongoDB can use **references** to store the relationships between data by including links or references from one document to another.

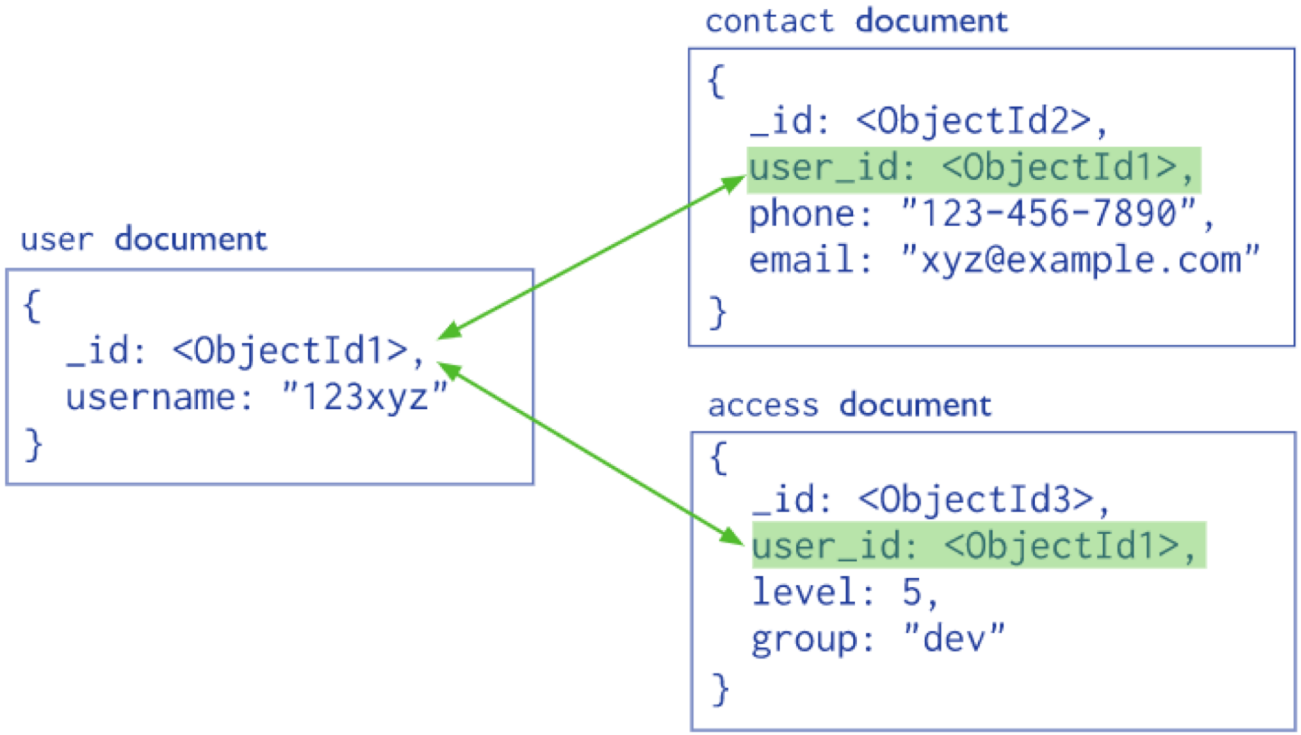
MongoDB Modeling Embedded Data¶

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

Embedded sub-document

Embedded sub-document

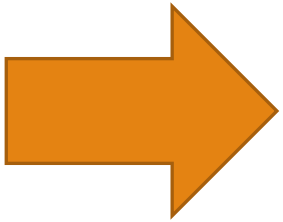
MongoDB Modeling References



Data Modeling Introduction

Relational model vs MongoDB model

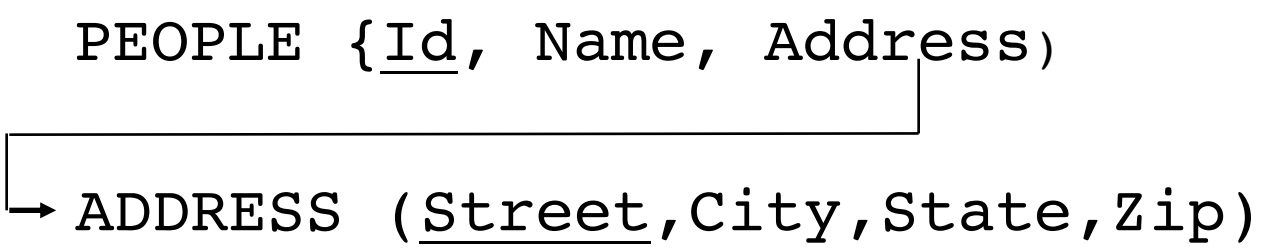
Relational Model
Database
Table, view
Row
Column
Primary key
Foreign Key
Tables join



MongoDB
Database
Collection
Document
Column
_id
Reference
Embedded document

Model One-to-One Relationship Relational => mongoDB

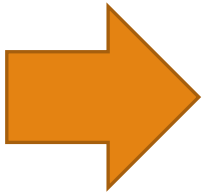
A person has only one address



Model One-to-One Relationship Relational => mongoDB

Use embedded documents to describe a one-to-one relationship¶

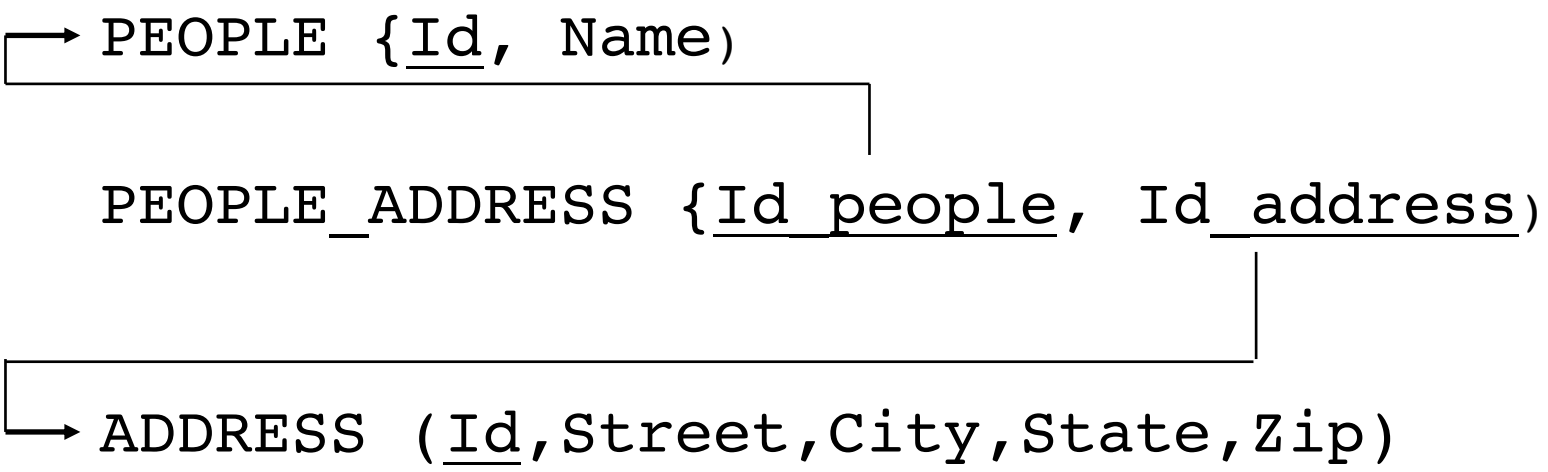
```
{
    _id: "joe",
    name: "Joe Bookreader"
}

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



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

Model One-to-Many Relationships Relational => mongoDB

A person has more than one address



- Two options: embedded or references

Model One-to-Many Relationships Relational => mongoDB

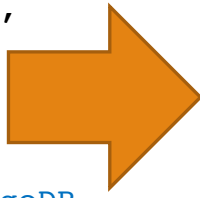
```
{
    _id: "joe",
    name: "Joe Bookreader"
}

{
    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"
}
```

Use embedded documents to describe one-to-many relationships

```
{
_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"
        }
]
}
```

Model One-to-Many Relationships Relational => mongoDB

```
{ 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"
            }
}
```



```
{
   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"
}
```

use references between documents to describe one-to-many relationships

MongoDB Shell Commands

mongo shell
Command helpers

help	Show help.
db.help()	Show help for database methods.
db.<collection>.help()	Show help on collection methods. The <collection> can be the name of an existing collection or a non-existing collection

mongo shell
Command print

show dbs	Print a list of all databases on the server.
show collections	Print a list of all collections for the current database.

mongo shell
Command create, switch

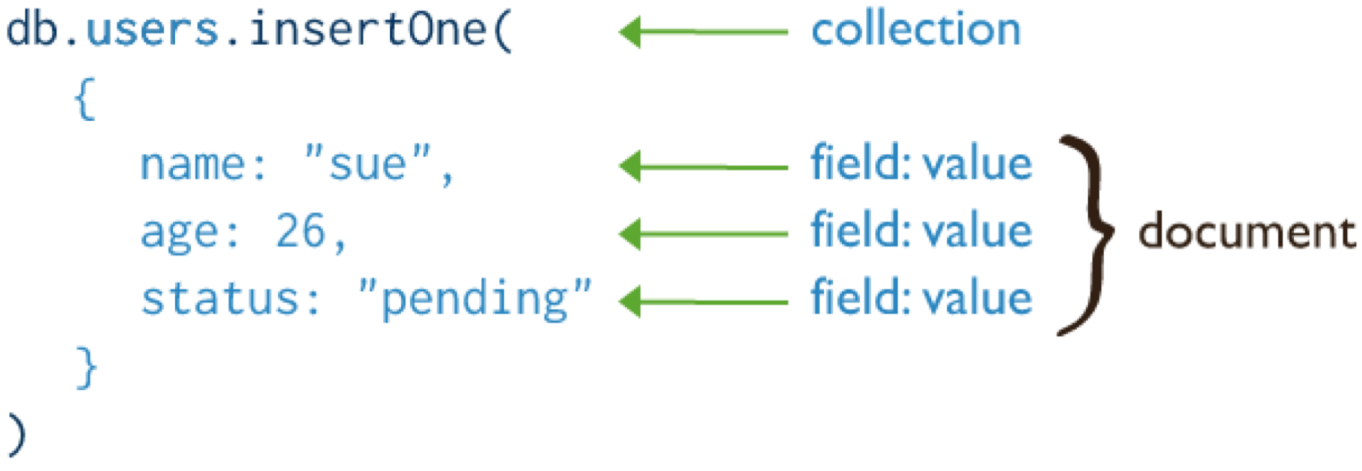
use <db>	Switch current database to <db>. The mongo shell variable db is set to the current database.

mongo shell
CRUD methods

db.<collection>.<method>(<filter>, <options>)

mongo shell
CRUD methods. Create Operations

db.collection.insertOne()	Inserts a document into a collection.
db.collection.insertMany()	Inserts multiple documents into a collection.



mongo shell
CRUD methods. Read Operations

db.collection.find()	Selects documents in a collection based on the filter and returns a cursor to the selected documents

```
db.users.find(
    { age: { $gt: 18 } },
    { name: 1, address: 1 }
).limit(5)
```

← collection
← query criteria
← projection
← cursor modifier

mongo shell CRUD methods. Update Operations

uc3m

db.collection.updateOne()	Updates a single document within the collection based on the filter.
db.collection.updateMany()	Updates all documents within the collection that match the filter.
db.collection.replaceOne()	Replaces a single document within the collection based on the filter.

```
db.users.updateMany(
   { age: { $lt: 18 } },
   { $set: { status: "reject" } }
)
```

collection

update filter

update action

mongo shell CRUD methods. Delete Operations

db.collection.deleteOne()	Removes a single document from a collection based on the filter.
db.collection.deleteMany()	Removes all documents that match the filter from a collection.

```
db.users.deleteMany(
   { status: "reject" }
)
```

collection

delete filter

mongo shell
query filter parameters

db.inventory.find({ "qty" : { $gt: 10 }})

{a: 10}	Docs where a is 10 or an array containing the value 10.
{a: 10, b: "hello"}	Docs where a is 10 and b is "hello".
{a: {$gt: 10}}	Docs where a is greater than 10. Also available: $lt (<), $gte (>=), $lte (<=), and $ne (!=).
{a: {$in: [10, "hello"]}}	Docs where a is either 10 or "hello".
{a: {$all: [10, "hello"]}}	Docs where a is an array containing both 10 and "hello".

mongo shell
query filter parameters

{"a.b": 10}	Docs where a is an embedded document with b equal to 10.
{a: {$elemMatch: {b: 1, c: 2}}}	Docs where a is an array that contains an element with both b equal to 1 and c equal to 2.
{$or: [{a: 1}, {b: 2}]}	Docs where a is 1 or b is 2.
{a: /^m/}	Docs where a begins with the letter m. One can also use the regex operator: {a: {$regex: "^m"}}
{a: {$mod: [10, 1]}}	Docs where a mod 10 is 1.
{a: {$type: "string"}}	Docs where a is a string.
{$text: {$search: "hello"}}	Docs that contain "hello" on a full text search.

mongo shell
not indexable queries

a: {$nin: [10, "hello"]}}	Docs where a is anything but 10 or "hello".
{a: {$size: 3}}	Docs where a is an array with exactly 3 elements.
{a: {$exists: true}}	Docs containing an a field.
{a: /foo.*bar/}	Docs where a matches the regular expression foo.*bar.
{a: {$not: {$type: 2}}}	Docs where a is not a string. $not negates any of the other query operators.

mongo shell
field update operators

{$inc: {a: 2}}	Increment a by 2.
{$set: {a: 5}}	Set a to the value 5.
{$unset: {a: 1}}	Delete the a key.
{$max: {a: 10}}	Set a to the greater value, either current or 10. If a does not exist, set a to 10.
{$min: {a: -10}}	Set a to the lowest value, either current or -10. If a does not exist, set a to -10.
{$mul: {a: 2}}	Set a to the product of the current value of a and 2. If a does not exist set a to 0.
{$rename: {a: "b"}}	Rename field a to b.
{$setOnInsert: {a: 1}}, {upsert: true}	Set field a to 1 in case of upsert operation.

mongo shell
field update operators
array update operators

{$push: {a: 1}}	Append the value 1 to the array a.
{$push: {a: {$each: [1, 2]}}}	Append both 1 and 2 to the array a.
{$push: {a: {$each: [10, 20, 30], $slice: -5}}}	Append 10, 20, and 30 to the array a, then trim the resulting array to contain only the last 5 elements. $slice can only be used with the $each modifier. Negative values trim to the last <num> elements, while positive values trim to the first <num> elements

mongo shell
field update operators
array update operators

{$push: {a: {$each: [50, 60, 70], $position: 0}}}	Insert 50, 60, and 70 starting at position 0 of the array a. $position can only be used with the $each modifier.
{$addToSet: {a: 1}}	Append the value 1 to the array a (if the value doesn't already exist).
{$addToSet: {a: {$each: [1, 2]}}}	Append both 1 and 2 to the array a (if they don't already exist).
{$pop: {a: 1}}	Remove the last element from the array a.
{$pop: {a: -1}}	Remove the first element from the array a.
{$pull: {a: ($gt: 5}}}	Remove all values greater than 5 from the array a.
{$pullAll: {a: [5, 6]}}	Remove multiple occurrences of 5 or 6 from the array a.

MAPPING SQL TO MONGODB

SQL TERM	MONGODB TERM
database (schema)	database
table	collection
index	index
row	document
column	field

MAPPING SQL TO MONGODB examples

CREATE TABLE people (id MEDIUMINT NOT NULL AUTO_INCREMENT, user_id Varchar(30), age Number, status char(1), PRIMARY KEY (id))	db.people.insertOne({ user_id: "abc123", age: 55, status: "A" })
ALTER TABLE people ADD join_date DATETIME	db.people.updateMany({ }, { $set: { join_date: new Date() } })
ALTER TABLE people DROP COLUMN join_date	db.people.updateMany({ }, { $unset: { "join_date": "" } })
CREATE INDEX idx_user_id_asc ON people(user_id)	db.people.createIndex({ user_id: 1 })
CREATE INDEX idx_user_id_asc_age_desc ON people(user_id, age DESC)	db.people.createIndex({ user_id: 1, age: -1 })
DROP TABLE people	db.people.drop()
INSERT INTO people(user_id, age, status) VALUES ("bcd001", 45, "A")	db.people.insertOne({ user_id: "bcd001", age: 45, status: "A" })
SELECT * FROM people	db.people.find()

MAPPING SQL TO MONGODB examples

SELECT id, user_id, status FROM people	db.people.find({ }, { user_id: 1, status: 1 })
SELECT user_id, status FROM people	db.people.find({ }, { user_id: 1, status: 1, _id: 0 })
SELECT * FROM people WHERE status = "A"	db.people.find({ status: "A" })
SELECT user_id, status FROM people WHERE status = "A"	db.people.find({ status: "A" }, { user_id: 1, status: 1, _id: 0 })
SELECT * FROM people WHERE status != "A"'	db.people.find({ status: { $ne: "A" } })
SELECT * FROM people WHERE status = "A" AND age = 50	db.people.find({ status: "A", age: 50 })
SELECT * FROM people WHERE status = "A" OR age = 50	db.people.find({ $or: [{ status: "A" } , { age: 50 }] })
SELECT * FROM people WHERE age > 25	db.people.find({ age: { $gt: 25 } })
SELECT * FROM people WHERE age < 25	db.people.find({ age: { $lt: 25 } })
SELECT * FROM people WHERE age > 25 AND age <= 50	db.people.find({ age: { $gt: 25, $lte: 50 } })
SELECT * FROM people WHERE user_id like "%bc%"	db.people.find({ user_id: /bc/ })

MAPPING SQL TO MONGODB examples

SELECT * FROM people WHERE user_id like "bc%"	db.people.find({ user_id: { $regex: /^bc/ } })
SELECT * FROM people WHERE status = "A" ORDER BY user_id ASC	db.people.find({ status: "A" }).sort({ user_id: 1 })
SELECT * FROM people WHERE status = "A" ORDER BY user_id DESC	db.people.find({ status: "A" }).sort({ user_id: -1 })
SELECT COUNT(*) FROM people	db.people.count()
SELECT COUNT(user_id) FROM people	db.people.count({ user_id: { $exists: true } })
SELECT COUNT(*) FROM people WHERE age > 30	db.people.count({ age: { $gt: 30 } })
SELECT DISTINCT(status) FROM people	db.people.distinct("status")
SELECT * FROM people LIMIT 1	db.people.findOne()
SELECT * FROM people LIMIT 5 SKIP 10	db.people.find().limit(5).skip(10)
EXPLAIN SELECT * FROM people WHERE status = "A"	db.people.find({ status: "A" }).explain()
UPDATE people SET status = "C" WHERE age > 25	db.people.updateMany({ age: { $gt: 25 } }, { $set: { status: "C" } })

REFERENCES

- MongoDB: The Definitive Guide, Kristina Chodorow & Michael Dirolf
- The Definitive Guide to MongoDB: A complete guide to dealing with Big Data using MongoDB (Definitive Guide Apress) , David Hows, 2013
- mondoDB, Data Models¶: https://docs.mongodb.com/manual/data-modeling/