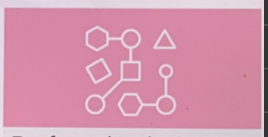
CRUD Operations

What you will learn

Use Mongo shell to connect a database



Perform basic CRUD operations

Mongo shell

A command line tool that helps you interact with MongoDB

```
> mongo "mongodb://USER:PASSWORD@uri/test"

Enterprise my-replicaset [primary] test> show dbs
admin                0.000GB
campusManagementDB   0.000GB
local                1.218GB
```

Mongo shell

Set database context

```
> use campusManagementDB
switched to db campusManagementDB
> show collections
staff
students
```

Create

```
> db.students.insertOne({
                        "firstName": "John",
                        "lastName": "Doe",
                        "email": "john.doe@email.com",
                        "studentId": 20217484
                    })
{
        "acknowledged" : true,
        "insertedId" : ObjectId("60424dad68ad40a3490e0db4")
}
```

Create Many

```
> students_list = [
{ "firstName": "Sarah", "lastName": "Jane", "email": "sarah@mail.com", "studentId": 20215124 },
{ "firstName": "Peter", "lastName": "Parker", "email": "peter@web.com", "studentId": 20218873 }]

> db.students.insertMany(students_list)
{
        "acknowledged" : true,
        "insertedIds" : [
                ObjectId("60424ee468ad40a3490e0db5"),
                ObjectId("60424ee468ad40a3490e0db6")
                ]
}
```

Read

```
> db.students.findOne()
{
        "_id" : ObjectId("60421b5c26fe73f38fdd83db"),
        "firstName" : "John",
        "lastName" : "Doe",
        "email" : "johnd@campus.edu",
        "studentId" : 20217484
}
```

Read

```
> db.students.findOne({ "email": "sarah@mail.com" })
{
        "_id" : ObjectId("604228bee45df18c77dc2dd7"),
        "firstName" : "Sarah",
        "lastName" : "Jane",
        "email" : "sarah@mail.com",
        "studentId" : 20215124
}
```

Read

```
> students.find({ "lastName": "Doe" })
{ "_id" : ObjectId("60421b5c26fe73f38fdd83db"), "firstName" : "John", "lastName" : "Doe",
"email" : "john.doe@email.com", "studentId" : 20217484 }
{ "_id" : ObjectId("60421b5c26fe73f38fdd84ab"), "firstName" : "Mark", "lastName" : "Doe",
"email" : "mark@mail.com", "studentId" : 20215001 }
```

Read

```
> students.count({ "lastName": "Doe" })
2
```

Replace

```
> student = db.students.findOne({ "lastName": "Doe" })
> db.student["onlineOnly"] = true
> db.student["email"] = "johnd@campus.edu"
> db.students.replaceOne({ "lastName": "Doe" }, student)
{ "acknowledged" : true, "matchedCount" : 1, "modifiedCount" : 1 }
```

Update

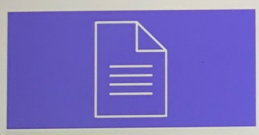
```
> changes = { "$set" : { "onlineOnly" : true,
                         "email" : "johnd@campus.edu" } }

> db.students.updateOne({ "lastName": "Doe" }, changes)
      { "acknowledged" : true, "matchedCount" : 1, "modifiedCount" : 1 }

> db.students.updateMany({}, { "$set": { "onlineOnly": true } })
{ "acknowledged" : true, "matchedCount" : 4, "modifiedCount" : 3 }
```

Delete

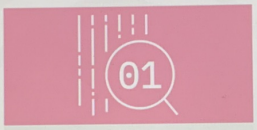
```
> db.students.deleteOne({ "studentId": 20218873 })
{ "acknowledged" : true, "deletedCount" : 1 }
> db.students.deleteMany({ "graduatedYear": 2019 })
```

Indexes

What you will learn



Explain why we need indexes



Perform efficient searches and sorts

Describe how MongoDB stores indexes

Why we need indexes

Indexes help quickly locate data without looking for it everywhere.

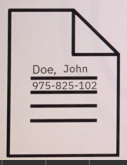
- Example: In the British Library, find *Designing Data-Intensive Applications* by Martin Kleppmann
  - **Without indexes**: You will need to look through all of the books
  - **With indexes**: Go to Computers / Databases / M (for Martin)

Everyday index samples

- Telephone book
- Dictionary
- Index at the end of a book

D

J



When to index

- Most frequent queries

collection name

db.courseEnrollment.find({ "courseId": 1547 })

key value

means store the index in AESC order

db.courseEnrollment.createIndex({ "courseId": 1 })

Use indexes to sort

Indexes can help with sorting

change to be a compound index → means indexing more than one field. So once we find all documents that matched 'courseId' they will all be already sorted by 'studentId' in ASEC order

db.courseEnrollment.find({ "courseId": 1547 }).sort({ "studentId": 1 })

db.courseEnrollment.createIndex({ "courseId": 1, "studentId": 1 })

Indexes in MongoDB

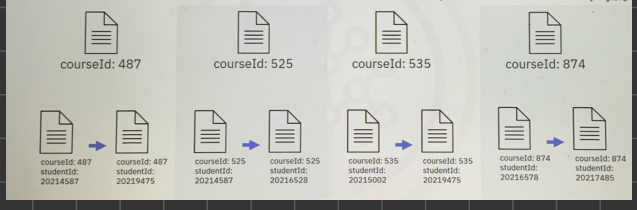
- MongoDB indexes are special data structures
- They store the fields you are indexing
- They also store the location of the document

How MongoDB stores indexes → in a tree form

first field with AESC, this makes finding documents much more effient

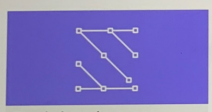
db.courseEnrollment.createIndex({ "courseId": 1, "studentId": 1 })

So if first field is already indexed, MongoDB doesn't need to sort it again

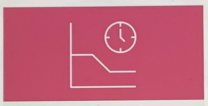


Aggregation Framework

What you will learn



Explain what an aggregation framework is in MongoDB



Describe how the aggregation framework is built and list its most common stages

Describe use cases for aggregation

Aggregation framework

sometime referred to aggregation pipeline

series of operation

What is the average student score in each course for 2020?

```
> db.courseResults.aggregate([
        { $match: { "year": 2020 } },
        { $group: { "_id": "$courseId", "avgScore": {
                    $avg: "$score" } } }
])
```

filter first

avg score

then group

Course results

```
courseResults = [
    { _id: 1, courseId: 1, year: 2019, score: 95 },
    { _id: 2, courseId: 1, year: 2019, score: 84 },
    { _id: 3, courseId: 2, year: 2019, score: 91 },
    { _id: 4, courseId: 1, year: 2020, score: 95 },
    { _id: 5, courseId: 1, year: 2020, score: 100 },
    { _id: 6, courseId: 2, year: 2020, score: 96 },
    { _id: 7, courseId: 2, year: 2020, score: 89 },
    { _id: 8, courseId: 2, year: 2020, score: 73 },
    { _id: 9, courseId: 2, year: 2021, score: 92 },
    { _id: 10, courseId: 1, year: 2021, score: 77 },
]
```

Aggregation stages

```
> db.courseResults.aggregate([
    { $match: { "year": 2020 } },
    { $group: { "_id": "$courseId", "avgScore": { $avg: "$score" } } }
])
```

use the id field to set group key

it

hold avg value

separates documents into groups according to a group key

Common aggregation stages

reshapes documents by including/excluding fields

- $project
- $sort
- $count count in that stage
- $merge move output to collection

Applying $project

```
students = [
    { "_id": 1, "firstName": "Sophie", "lastName": "Foucher", "city": "Quebo"
    },
    { "_id": 2, "firstName": "Stafford", "lastName": "von Hagt", "city":
    "Terezin" },
    { "_id": 3, "firstName": "Olivero", "lastName": "Liggett", "city":
    "Guaranda" }
]
```

Applying the $project stage

```
db.students.aggregate([
    {
    $project: { firstName: 1, lastName: 1 }
    }
])
```

Applying $project: Resulting outcomes

always included unless explicitly excluded

```
[
   { "_id": 1, "firstName": "Sophie", "lastName": "Foucher" },
   { "_id": 2, "firstName": "Stafford", "lastName": "von Hagt" },
   { "_id": 3, "firstName": "Olivero", "lastName": "Liggett"}
]
```

Applying the $sort stage: Sorting order options

$sort

```
db.students.aggregate([
   {
   $project: { firstName: 1, lastName: 1 }
   },
   {
   $sort: { lastName: -1 }
   }
])
```

sorting by last name in DSC

Applying $sort stage: The resulting output

```
[{
   "_id": 2,
   "firstName": "Stafford", "lastName": "von Hagt"
   },
   {
   "_id": 3,
   "firstName": "Olivero", "lastName": "Liggett"
   },
   {
   "_id": 1,
   "firstName": "Sophie", "lastName": "Foucher"
}]
```

Applying the $count stage

$count

```
> db.students.aggregate([{ $count: "totalStudents" }])
{ "totalStudents": 3 }
```

Applying $match

used $merge

```
courseResults = [
{ _id: 1, courseId: 1, year: 2019, score: 95 },
{ _id: 2, courseId: 1, year: 2019, score: 84 },
{ _id: 3, courseId: 2, year: 2019, score: 91 },
{ _id: 4, courseId: 1, year: 2020, score: 95 },
{ _id: 5, courseId: 1, year: 2020, score: 100 },
{ _id: 6, courseId: 2, year: 2020, score: 96 },
{ _id: 7, courseId: 2, year: 2020, score: 89 },
{ _id: 8, courseId: 2, year: 2020, score: 73 },
{ _id: 9, courseId: 2, year: 2021, score: 92 },
{ _id: 10, courseId: 1, year: 2021, score: 77 }]
```

Aggregation use cases

- Reporting: Track student progress
- Analysis: Determine the average product sales by country

Replication and Shading

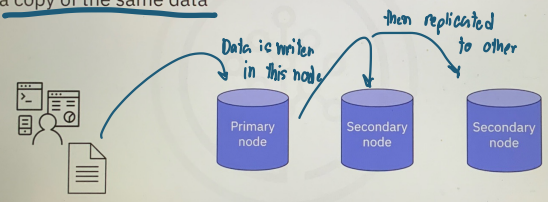
What you will learn

- Define replication
- Explain the benefits of replication
- Describe what sharding is and describe its benefits
- Explain how sharding helps scale a database

Replication

made of three bearing node

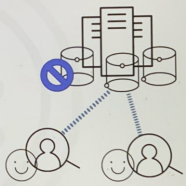
A MongoDB Replica Set is a cluster of data-bearing nodes, each maintaining a copy of the same data



Benefits of replication

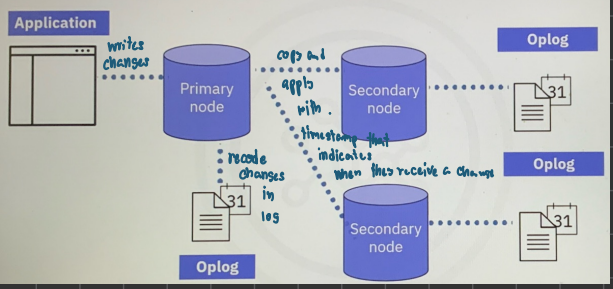
- Redundancy — if one node failed, it still run from other copied data
- High availability
- Fault tolerance

Caution! Be aware that replication does not equal disaster recovery



The replication process

MongoDB's



How an Oplog works

Example: A database of student information:

```
{"_id":1,"firstName":"Sophie","lastName":"Foucher","city":"Quebo"}
{"_id":2,"firstName":"Stafford","lastName":"von Hagt","city":"Terezin"}
{"_id":3,"firstName":"Olivero","lastName":"Liggett","city":"Guaranda"}
```

You can delete the preceding data using the statement:

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

How an Oplog works

The command, db.students.deleteMany({}) displays as an individual operation in the Oplog

```
{ "op" : "d", "ns" : "campusManagement.student", "o" : { "_id" : 1 } }
{ "op" : "d", "ns" : "campusManagement.student", "o" : { "_id" : 2 } }
{ "op" : "d", "ns" : "campusManagement.student", "o" : { "_id" : 3 } }
```

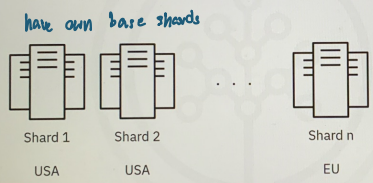
What is Sharding?

When data is growing beyond hardware capacity:

- Scale vertically (bigger, faster hardware) → sometime it's not feasible
- Scale horizontally (partition the data) → better

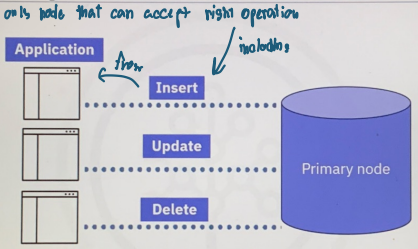
Benefits of Sharding

- Helps with increased throughput and capacity
- Can help adhere to legal requirements for data storage



The Primary node and elections

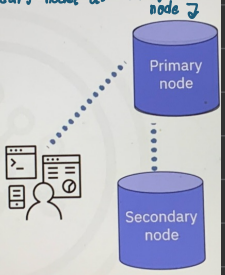
only node that can accept right operation



Why does an election happen?

MongoDB can automatically elect one of the secondary nodes as new primary node

- The current primary node becomes unavailable
- A new replica set is initialized and needs to choose its initial primary
- A database administrator initiates a manual failover for maintenance or upgrades

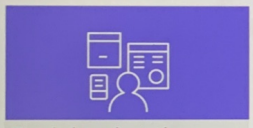


Requirements to become Primary

- ✓MongoDB's voting system ensures only one member becomes a primary node
- ✓Member nodes that are eligible have minimal replication lag and have the most recent data
- ✓The member node receiving the majority of the votes becomes the new Primary.

Accessing MongoDB from Python

What you will learn



Explain what the MongoClient is

Perform basic CRUD operations using Python

MongoClient

A class that helps you interact with MongoDB

```
from pymongo import MongoClient
uri = "mongodb://USER:PASSWORD@uri/test"
client = MongoClient(uri)
campusDB = client.campusManagementDB
students = campusDB.students
```

the address of MongoDB

should do once in our code

object pointing

point collection

Create

```
students.insert_one({"firstName": "John",
                     "lastName": "Doe",
                     "email": "john.doe@email.com",
                     "studentId": 20217484
                     })
```

Create Many

```
students_list = [
{"firstName": "Sarah", "lastName": "Jane",
"email": "sarah@mail.com", "studentId": 20215124},
{"firstName": "Peter", "lastName": "Parker",
"email": "peter@web.com", "studentId": 20218873}]

students.insert_many(students_list)
```

Read

```
students.find_one()
students.find_one({"email": "sarah@mail.com"})
students.find({"lastName": "Doe"})
students.count_documents({"lastName": "Doe"})
```

return first doc.

specify

return all doc that equal to

read only the first one

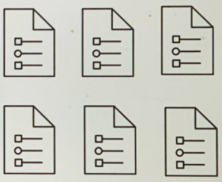
count #

Print read documents

```
from bson.json_util import dumps
cursor = students.find({"lastName": "Doe"})
print(dumps(cursor, indent=4))
```

point

to retrieve doc



Replace for large changes

```
student = students.find_one({"lastName": "Doe"})
student["onlineOnly"] = True
student["email"] = "johnd@campus.edu"
students.replace_one({"lastName": "Doe"}, student)
```

create new field to indicate whose students are

update email

filter

update

Update for small changes

```
changes = {"$set": {"onlineOnly": True,
"email": "johnd@campus.edu"
}}

students.update_one({"lastName": "Doe"}, changes)
students.update_many({}, {"$set": {"onlineOnly": True}})
```

to change all doc

Delete

```
students.delete_one({"studentId": 20218873})

students.delete_many({"graduatedYear": 2019})
```