ДЕРЕККҚОР, КОЛЛЕКЦИЯ ЖӘНЕ ҚҰЖАТ ҰҒЫМДАРЫ

Дерекқор дегеніміз не?

Дерекқор – белгілі бір ақпараттық жүйеге немесе қолданбаға қатысты барлық мәліметтерді сақтауға арналған логикалық орта.

MongoDB жүйесінде дерекқор ең жоғарғы деңгейдегі құрылым болып саналады және ол файлдық жүйеде жеке каталог ретінде сақталады. Әдетте бір дерекқор бір қолданбамен байланыстырылады, себебі барлық қажетті деректер сол қолданбаның ішінде өңделеді. Бір серверде бірнеше дерекқор болуы мүмкін, бірақ олардың әрқайсысы бір-бірінен тәуелсіз жұмыс істейді.

Дерекқор деректерді қауіпсіз сақтауға, оларды басқаруға және құрылымдауға мүмкіндік береді.

Коллекция дегеніміз не?

MongoDB-дегі коллекция – бір немесе бірнеше құжаттан (document) тұратын деректер жиыны, ол функциялық жағынан реляциялық дерекқордағы кестеге (table) ұқсас. SQL-кестелерден айырмашылығы, коллекциялар «схемасыз» (schema-less) болып келеді, яғни бір коллекцияның ішіндегі құжаттардың құрылымы әртүрлі болуы мүмкін: өрістер (field), олардың атауы және дерек типтері бірдей болуға міндетті емес.

Коллекциялардың негізгі ерекшеліктері:

- Икемділік: құжаттардың барлығы бірдей құрылымға бағынбайды, қажет өрістер ғана сақтала береді.
- Біріктіру қағидасы: коллекциядағы құжаттар белгілі бір ортақ белгі/тақырып бойынша топтастырылады.
- Индекстер: сұраныстарды жылдамдату үшін индекстер құруға болады.
- Өлшем шектеуі: бір құжаттың ең үлкен көлемі – 16 МБ; одан үлкен файлдармен жұмыс істеу үшін GridFS қолданылады.

Құжат дегеніміз не?

Құжат – MongoDB дерекқорындағы ең кіші және негізгі ақпараттық бірлік. Құжаттар JSON немесе BSON форматында сақталады және кілт–мән принципі бойынша құрылады. Әрбір құжатта міндетті түрде бірегей идентификатор – _id өрісі болады. Құжат ішінде қарапайым мәндермен қатар, массивтер мен ішкі объектілер де болуы мүмкін. Бұл деректерді күрделі құрылымда сақтауға мүмкіндік береді. Құжаттардың икемділігі MongoDB-ні веб-қосымшалар мен үлкен деректерді өңдеуге өте қолайлы етеді.

```
> db.student.find().pretty()
{
        "_id" : ObjectId("601ef9405f8d812f4c95ec39"),
        "name" : "Nikhil",
        "age" : 21
}
{
        "_id" : ObjectId("601ef9405f8d812f4c95ec3a"),
        "name" : "Vishal",
        "age" : 17
}
{
        "_id" : ObjectId("601ef9405f8d812f4c95ec3b"),
        "name" : "Sagar",
        "age" : 17
}
{
        "_id" : ObjectId("601ef9405f8d812f4c95ec3c"),
        "name" : "Sahil",
        "age" : 19
}
```

MongoDB құрылымы иерархиялық түрде ұйымдасқан. Ең жоғары деңгейде дерекқор орналасады. Оның ішінде бірнеше коллекция болады. Ал әр коллекция көптеген құжаттардан тұрады.

Яғни: дерекқор → коллекция → құжат

Осы құрылым деректерді логикалық түрде сақтауға, өңдеуге және кеңейтуге мүмкіндік береді. Бұл тәсіл үлкен көлемдегі ақпаратпен жұмыс істеуде өте тиімді.

Мысал: MongoDB жүйесіндегі дерекқор, коллекция және құжат

1

Дерекқор: LibraryDB

Бұл біздің кітапханамыздың барлық деректерін қамтитын негізгі контейнер.

2

Коллекция: books

LibraryDB ішіндегі "кітаптар" коллекциясы барлық кітаптар туралы ақпаратты топтастырады.

3

Құжат

{"title": "Database Basics", "author": "A. Smith", "year" : 2025} – бұл бір кітап туралы толық ақпаратты қамтитын жеке құжат.

Қорытындылай келе, MongoDB дерекқоры үш негізгі элементтен тұрады: дерекқор, коллекция және құжат. Дерекқор барлық мәліметтерді сақтайтын орта болса, коллекция – сол мәліметтердің логикалық тобы, ал құжат – нақты ақпаратты сақтайтын ең кіші бірлік.
MongoDB-нің басты артықшылығы – оның икемділігі, жоғары өнімділігі және Big Data жобаларға бейімділігі. Сондықтан ол қазіргі таңда веб-қосымшаларда, аналитикалық жүйелерде және үлкен деректерді өңдеуде кеңінен қолданылады.

MONGODB-ДЕ **CRUD** ОПЕРАЦИЯЛАРЫ

КІРІСПЕ

MongoDB NoSQL (Not Only SQL) тұжырымдамасына негізделген. NoSQL тәсілі үлкен деректермен жұмыс істеу кезінде масштабталу мәселелерін шешуге бағытталған. Бұл келесі қағидалар арқылы жүзеге асады:

- Атомарлық – барлық операциялар біртұтас әрекет ретінде орындалады;
- Деректердің келісімділігі – деректердің тұтастығы бұзылмайды.

- MongoDB-де қолданылатын негізгі операциялар **CRUD операциялары** деп аталады. Олар келесі әрекеттерден тұрады:

CREATE ОПЕРАЦИЯЛАРЫ

- CRUD жүйесіндегі *Create* операциясы MongoDB-де **insert** деп аталады. *Insert* – коллекцияға жаңа деректерді қосу. MongoDB-де бұл принцип өзгермейді, бірақ егер коллекция әлі жоқ болса, **insert операциясы алдымен коллекцияны құрып**, содан кейін деректерді енгізеді.

MONGODB-ДЕ CREATE ОПЕРАЦИЯЛАРЫН ОРЫНДАУ ҮШІН КЕЛЕСІ ӘДІСТЕР ПАЙДАЛАНЫЛАДЫ:

Әдіс	**Сипаттамасы**
db.collection.insertOne()	Коллекцияға бір құжатты енгізу үшін қолданылады
db.collection.insertMany()	Коллекцияға бірнеше құжатты бірден енгізу үшін қолданылады
db.createCollection()	Бос коллекция құру үшін қолданылады

CREATE ОПЕРАЦИЯСЫНА МЫСАЛ

Енді MongoDB-дегі CRUD операцияларының **Create** бөлігіне қатысты бірнеше мысалды қарастырайық.

Мысал 1:

Бұл мысалда біз **student** коллекциясына **бір студенттің мәліметтерін** құжат (document) түрінде енгіземіз. Ол үшін MongoDB-де бір құжатты қосуға арналған **`db.collection.insertOne() method`** әдісі қолданылады.

```
> use GeeksforGeeks
switched to db GeeksforGeeks
> db.student.insertOne({
... name : "Sumit",
... age : 20,
... branch : "CSE",
... course : "C++ STL",
... mode : "online",
... paid : true,
... amount : 1499
... })
{
        "acknowledged" : true,
        "insertedId" : ObjectId("5e540cdc92e6dfa3fc48ddae")
}
>
```

- Бұл мысалда біз **student** коллекциясына **бірнеше студенттің мәліметтерін** құжаттар (documents) түрінде енгіземіз. Ол үшін MongoDB жүйесінде бірнеше құжатты бір уақытта қосуға мүмкіндік беретін **db.collection.insertMany()** әдісі қолданылады.

```
> use GeeksforGeeks
switched to db GeeksforGeeks
> db.student.insertMany([
... {
... name : "Sumit",
... age : 20,
... branch : "CSE",
... course : "C++ STL",
... mode : "online",
... paid : true,
... amount : 1499
... },
...
... {
... name : "Rohit",
... age : 21,
... branch : "CSE",
... course : "C++ STL",
... mode : "online",
... paid : true,
... amount : 1499
... }
...
... ])
{
        "acknowledged" : true,
        "insertedIds" : [
                ObjectId("5e540d3192e6dfa3fc48ddaf"),
                ObjectId("5e540d3192e6dfa3fc48ddb0")
        ]
}
```

READ ОПЕРАЦИЯЛАРЫ (ОҚУ ОПЕРАЦИЯЛАРЫ)

CRUD жүйесіндегі Read операциясы MongoDB-де find әдісі арқылы орындалады. Серверде сақталған деректерді оқу үшін find() қолданылады. Біз filter (сүзгі) арқылы белгілі бір шартқа сай құжаттарды ғана алып оқи аламыз.Егер барлық құжаттарды оқу керек болса, find({}) қолданылады.

Әдістер:

- **db.collection.find()** – коллекциядан барлық сәйкес келетін құжаттарды алу үшін қолданылады.
- **db.collection.findOne()** – сұраныс шарттарына сәйкес келетін бір ғана құжатты қайтарады.

READ ОПЕРАЦИЯСЫНА МЫСАЛ

- Бұл мысалда **student** коллекциясынан студенттердің мәліметтерін **db.collection.find()** әдісін қолдану арқылы аламыз. Бұл әдіс сұраныс шарттарына сәйкес келетін барлық құжаттарды қайтарады және деректерді көруге мүмкіндік береді.

```
> use GeeksforGeeks
switched to db GeeksforGeeks
> db.student.find().pretty()
{
        "_id" : ObjectId("5e540cdc92e6dfa3fc48ddae"),
        "name" : "Sumit",
        "age" : 20,
        "branch" : "CSE",
        "course" : "C++ STL",
        "mode" : "online",
        "paid" : true,
        "amount" : 1499
}
{
        "_id" : ObjectId("5e540d3192e6dfa3fc48ddaf"),
        "name" : "Sumit",
        "age" : 20,
        "branch" : "CSE",
        "course" : "C++ STL",
        "mode" : "online",
        "paid" : true,
        "amount" : 1499
}
{
        "_id" : ObjectId("5e540d3192e6dfa3fc48ddb0"),
        "name" : "Rohit",
        "age" : 21,
        "branch" : "CSE",
        "course" : "C++ STL",
        "mode" : "online",
        "paid" : true,
        "amount" : 1499
}
```

MONGODB-ДЕ **CRUD** ОПЕРАЦИЯЛАРЫ (UPDATE, DELETE)

UPDATE ОПЕРАЦИЯЛАРЫ (ЖАҢАРТУ ОПЕРАЦИЯЛАРЫ)

Update операциялары коллекциядағы бар құжаттарды жаңарту немесе өзгерту үшін қолданылады. Белгілі бір сұранысқа сәйкес келетін бір құжатты да, бірнеше құжатты да жаңартуға болады. MongoDB жүйесінде update операциялары келесі әдістер арқылы орындалады:

Әдістер:

- **db.collection.updateOne()** – берілген шартқа сәйкес келетін бір ғана құжатты жаңарту үшін қолданылады.
- **db.collection.updateMany()** – берілген шартқа сәйкес келетін бірнеше құжатты жаңарту үшін қолданылады.
- **db.collection.replaceOne()** – берілген шартқа сәйкес келетін бір құжатты толықтай басқа құжатпен алмастырады.

UPDATE ОПЕРАЦИЯСЫНА МЫСАЛДАР

1-мысал:

Бұл мысалда **student** коллекциясындағы Sumit атты студенттің жасын

`db.collection.updateOne()`

әдісі арқылы жаңартамыз.

```
> use GeeksforGeeks
switched to db GeeksforGeeks
> db.student.updateOne({name: "Sumit"},{$set:{age: 24 }})
{ "acknowledged" : true, "matchedCount" : 1, "modifiedCount" : 0 }
> db.student.find().pretty()
{
        "_id" : ObjectId("5e540cdc92e6dfa3fc48ddae"),
        "name" : "Sumit",
        "age" : 24,
        "branch" : "CSE",
        "course" : "C++ STL",
        "mode" : "online",
        "paid" : true,
        "amount" : 1499
}
{
        "_id" : ObjectId("5e540d3192e6dfa3fc48ddaf"),
        "name" : "Sumit",
        "age" : 20,
        "branch" : "CSE",
        "course" : "C++ STL",
        "mode" : "online",
        "paid" : true,
        "amount" : 1499
}
{
        "_id" : ObjectId("5e540d3192e6dfa3fc48ddb0"),
        "name" : "Rohit",
        "age" : 21,
        "branch" : "CSE",
        "course" : "C++ STL",
        "mode" : "online",
        "paid" : true,
        "amount" : 1499
}
```

- **2-мысал:**

Бұл мысалда **student** коллекциясындағы барлық құжаттар үшін оқу жылы (course year) өрісін `db.collection.updateMany()` әдісі арқылы жаңартамыз.

```
> use GeeksforGeeks
switched to db GeeksforGeeks
> db.student.updateMany({}, {$set: {year: 2020}})
{ "acknowledged" : true, "matchedCount" : 3, "modifiedCount" : 3 }
> db.student.find().pretty()
{
        "_id" : ObjectId("5e540cdc92e6dfa3fc48ddae"),
        "name" : "Sumit",
        "age" : 24,
        "branch" : "CSE",
        "course" : "C++ STL",
        "mode" : "online",
        "paid" : true,
        "amount" : 1499,
        "year" : 2020
}
{
        "_id" : ObjectId("5e540d3192e6dfa3fc48ddaf"),
        "name" : "Sumit",
        "age" : 20,
        "branch" : "CSE",
        "course" : "C++ STL",
        "mode" : "online",
        "paid" : true,
        "amount" : 1499,
        "year" : 2020
}
{
        "_id" : ObjectId("5e540d3192e6dfa3fc48ddb0"),
        "name" : "Rohit",
        "age" : 21,
        "branch" : "CSE",
        "course" : "C++ STL",
        "mode" : "online",
        "paid" : true,
        "amount" : 1499,
        "year" : 2020
}
```

DELETE ОПЕРАЦИЯЛАРЫ (ЖОЮ ОПЕРАЦИЯЛАРЫ)

Delete операциялары коллекциядан құжаттарды жою үшін қолданылады. Құжаттарды белгілі бір шарт бойынша немесе барлық құжаттарды бірден жоюға болады.

Әдістері:

- **db.collection.deleteOne()** – берілген шартқа сәйкес келетін бір құжатты жояды.
- **db.collection.deleteMany()** – берілген шартқа сәйкес келетін бірнеше құжатты немесе барлық құжаттарды жояды.

DELETE ОПЕРАЦИЯСЫНА МЫСАЛДАР

- **1-мысал:** Бұл мысалда **student** коллекциясынан бір құжатты **db.collection.deleteOne()** әдісі арқылы жоямыз.

```
> db.student.deleteOne({name: "Sumit"})
{ "acknowledged" : true, "deletedCount" : 1 }
> db.student.find().pretty()
{
        "_id" : ObjectId("5e540d3192e6dfa3fc48ddaf"),
        "name" : "Sumit",
        "age" : 20,
        "branch" : "CSE",
        "course" : "C++ STL",
        "mode" : "online",
        "paid" : true,
        "amount" : 1499,
        "year" : 2020
}
{
        "_id" : ObjectId("5e54103592e6dfa3fc48ddb1"),
        "name" : "Rohit",
        "age" : 21,
        "branch" : "CSE",
        "course" : "C++ STL",
        "mode" : "online",
        "paid" : true,
        "amount" : 1499
}
```

2-мысал:

Бұл мысалда **student** коллекциясындағы барлық құжаттарды

db.collection.deleteMany() әдісі арқылы жоямыз.

```
> use GeeksforGeeks
switched to db GeeksforGeeks
> db.student.find().pretty()
{
        "_id" : ObjectId("5e540d3192e6dfa3fc48ddaf"),
        "name" : "Sumit",
        "age" : 20,
        "branch" : "CSE",
        "course" : "C++ STL",
        "mode" : "online",
        "paid" : true,
        "amount" : 1499,
        "year" : 2020
}
{
        "_id" : ObjectId("5e54103592e6dfa3fc48ddb1"),
        "name" : "Rohit",
        "age" : 21,
        "branch" : "CSE",
        "course" : "C++ STL",
        "mode" : "online",
        "paid" : true,
        "amount" : 1499
}
> db.student.deleteMany({})
{ "acknowledged" : true, "deletedCount" : 2 }
```

ҚОРЫТЫНДЫ

Осылайша, **CRUD-операцияларын** қолдаудың арқасында MongoDB әзірлеушілерге **Big Data объектілерімен** жұмыс істеу үшін өте ыңғайлы әрі интуитивті түсінікті интерфейс ұсынады. Бұл MongoDB-ді үлкен деректермен жұмыс істеуге қолайлы құралға айналдырады.

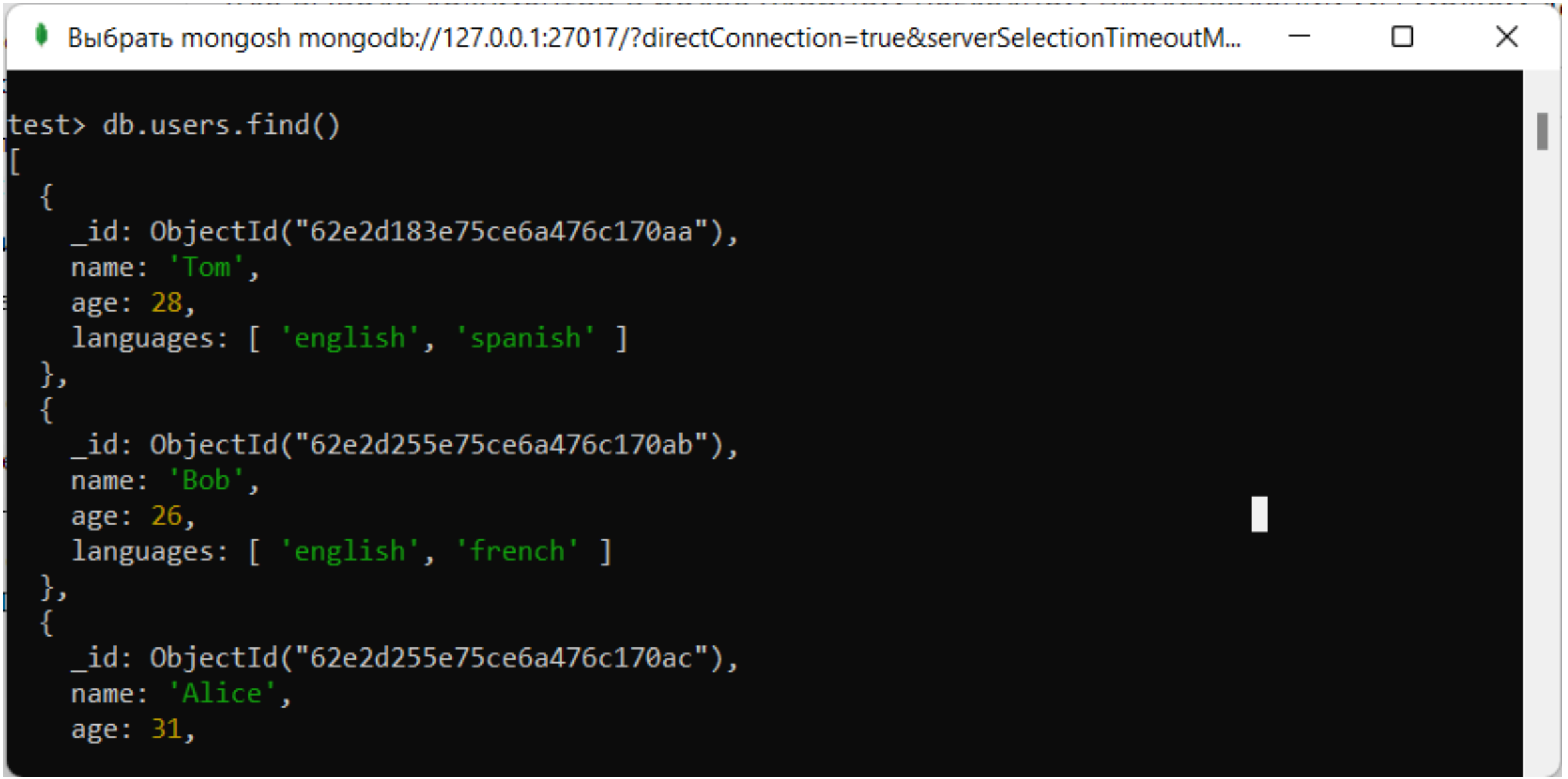
MONGODB-ДЕГІ FIND() ФУНКЦИЯСЫ

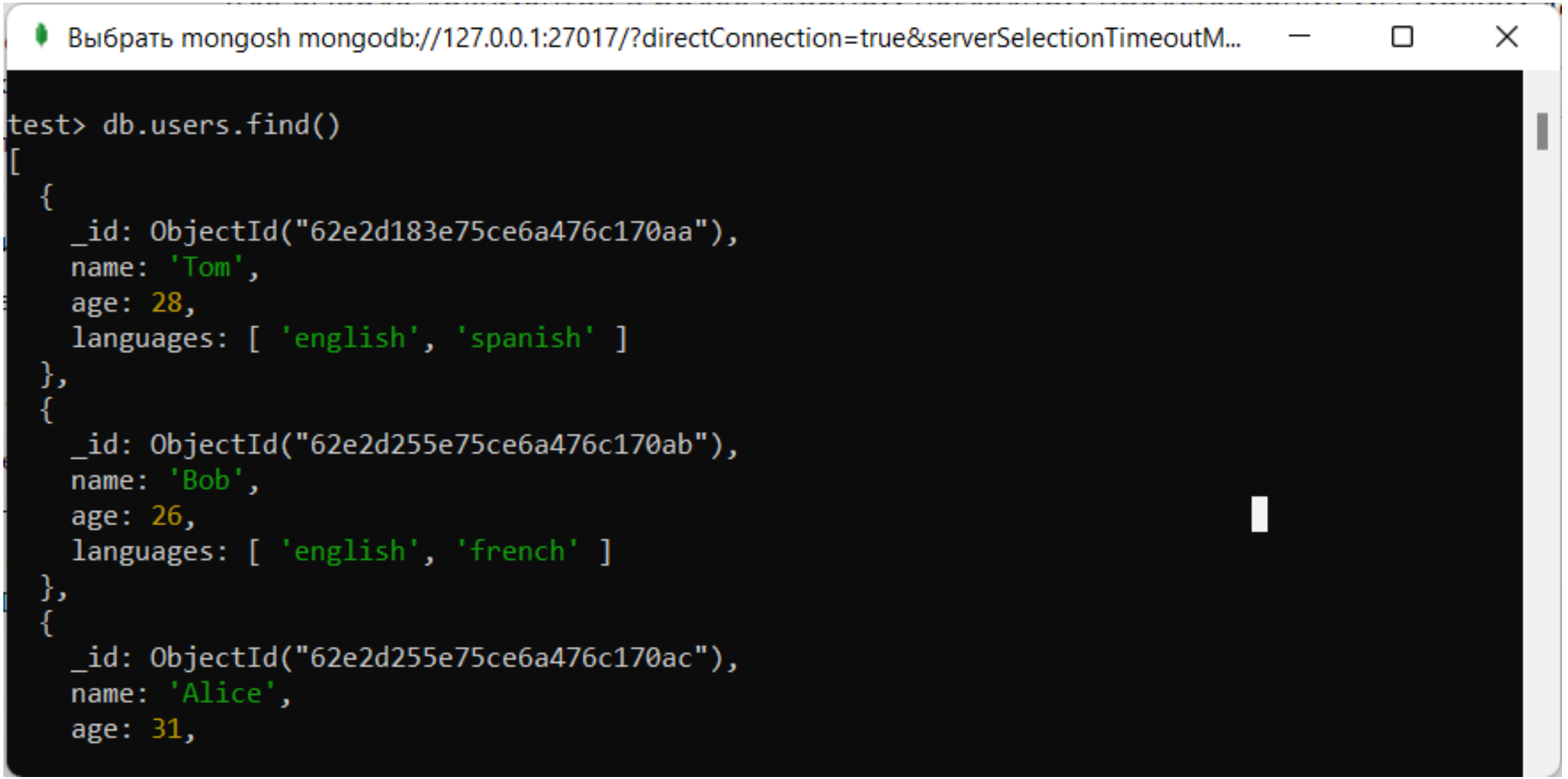
MONGODB-ДЕГІ FIND() ФУНКЦИЯСЫ

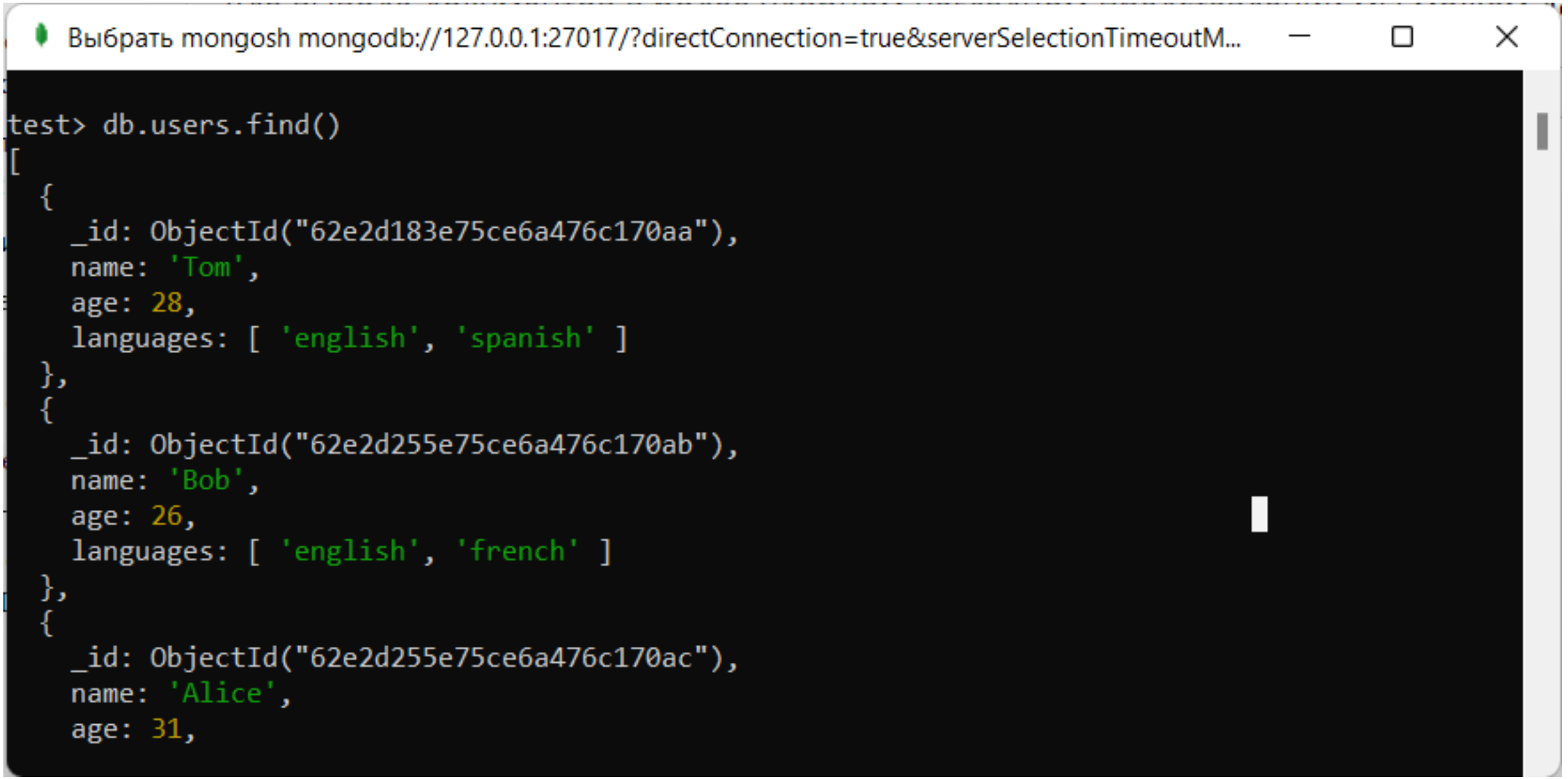
FIND() ФУНКЦИЯСЫ

- Құжаттарды коллекциядан таңдаудың (шығарудың) ең қарапайым тәсілі – **find()** функциясын қолдану. Бұл функцияның жұмысы SQL тіліндегі **SELECT * FROM Table** сұранысына ұқсайды, яғни ол кестедегі барлық жолдарды шығарады.
- Мысалы, **users** коллекциясынан барлық құжаттарды алу үшін келесі команданы пайдаланамыз:

```
db.users.find()
```


Выбрать mongosh mongodb://127.0.0.1:27017/?directConnection=true&serverSelectionTimeoutM...
test> db.users.find()
[
{
_id: ObjectId("62e2d183e75ce6a476c170aa"),
name: 'Tom',
age: 28,
languages: ['english', 'spanish']
},
{
_id: ObjectId("62e2d255e75ce6a476c170ab"),
name: 'Bob',
age: 26,
languages: ['english', 'french']
},
{
_id: ObjectId("62e2d255e75ce6a476c170ac"),
name: 'Alice',
age: 31,


Выбрать mongosh mongodb://127.0.0.1:27017/?directConnection=true&serverSelectionTimeoutM...
test> db.users.find()
[
{
_id: ObjectId("62e2d183e75ce6a476c170aa"),
name: 'Tom',
age: 28,
languages: ['english', 'spanish']
},
{
_id: ObjectId("62e2d255e75ce6a476c170ab"),
name: 'Bob',
age: 26,
languages: ['english', 'french']
},
{
_id: ObjectId("62e2d255e75ce6a476c170ac"),
name: 'Alice',
age: 31,


Выбрать mongosh mongodb://127.0.0.1:27017/?directConnection=true&serverSelectionTimeoutM...
test> db.users.find()
[
{
_id: ObjectId("62e2d183e75ce6a476c170aa"),
name: 'Tom',
age: 28,
languages: ['english', 'spanish']
},
{
_id: ObjectId("62e2d255e75ce6a476c170ab"),
name: 'Bob',
age: 26,
languages: ['english', 'french']
},
{
_id: ObjectId("62e2d255e75ce6a476c170ac"),
name: 'Alice',
age: 31,

- Алайда, егер бізге барлық құжаттар емес, тек **белгілі бір шартқа сай келетін** құжаттар ғана керек болса ше? Мысалы, бұған дейін деректер қорына келесі құжаттарды қосқан болатынбыз:

```
1 db.users.insertOne({"name": "Tom", "age": 28, languages: ["english", "spanish"]})
2 db.users.insertOne({"name": "Bill", "age": 32, languages: ["english", "french"]})
3 db.users.insertOne({"name": "Tom", "age": 32, languages: ["english", "german"]})
```

- **name = "Tom" болатын барлық құжаттарды шығару**
- Төмендегі сұраныс **name өрісінің мәні "Tom"** болатын барлық құжаттарды көрсетеді:

```
test> db.users.find({name: "Tom"})
[
  {
    _id: ObjectId("62e2d6a5e75ce6a476c170b3"),
    name: 'Tom',
    age: 28,
    languages: [ 'english', 'spanish' ]
  },
  { _id: ObjectId("62e2d348e75ce6a476c170ae"),
    _id: ObjectId("62e2d6a5e75ce6a476c170b5"),
    name: 'Tom',
    age: 32,s: [ 'english', 'spanish' ]
    languages: [ 'english', 'german' ]
  }
]   _id: ObjectId("62e2d3d5e75ce6a476c170af"),
test>
```

- Алайда, егер бізге барлық құжаттар емес, тек **белгілі бір шартқа сай келетін** құжаттар ғана керек болса ше? Мысалы, бұған дейін деректер қорына келесі құжаттарды қосқан болатынбыз:

```
1 db.users.insertOne({"name": "Tom", "age": 28, languages: ["english", "spanish"]})
2 db.users.insertOne({"name": "Bill", "age": 32, languages: ["english", "french"]})
3 db.users.insertOne({"name": "Tom", "age": 32, languages: ["english", "german"]})
```

- **name = "Tom" болатын барлық құжаттарды шығару**
- Төмендегі сұраныс **name өрісінің мәні "Tom"** болатын барлық құжаттарды көрсетеді:

```
test> db.users.find({name: "Tom"})
[
  {
    _id: ObjectId("62e2d6a5e75ce6a476c170b3"),
    name: 'Tom',
    age: 28,
    languages: [ 'english', 'spanish' ]
  },
  { _id: ObjectId("62e2d348e75ce6a476c170ae"),
    _id: ObjectId("62e2d6a5e75ce6a476c170b5"),
    name: 'Tom',
    age: 32,s: [ 'english', 'spanish' ]
    languages: [ 'english', 'german' ]
  }
]   _id: ObjectId("62e2d3d5e75ce6a476c170af"),
test>
```

- Алайда, егер бізге барлық құжаттар емес, тек **белгілі бір шартқа сай келетін** құжаттар ғана керек болса ше? Мысалы, бұған дейін деректер қорына келесі құжаттарды қосқан болатынбыз:

```
1 db.users.insertOne({"name": "Tom", "age": 28, languages: ["english", "spanish"]})
2 db.users.insertOne({"name": "Bill", "age": 32, languages: ["english", "french"]})
3 db.users.insertOne({"name": "Tom", "age": 32, languages: ["english", "german"]})
```

- **name = "Tom" болатын барлық құжаттарды шығару**
- Төмендегі сұраныс **name өрісінің мәні "Tom"** болатын барлық құжаттарды көрсетеді:

```
test> db.users.find({name: "Tom"})
[
  {
    _id: ObjectId("62e2d6a5e75ce6a476c170b3"),
    name: 'Tom',
    age: 28,
    languages: [ 'english', 'spanish' ]
  },
  { _id: ObjectId("62e2d348e75ce6a476c170ae"),
    _id: ObjectId("62e2d6a5e75ce6a476c170b5"),
    name: 'Tom',
    age: 32,s: [ 'english', 'spanish' ]
    languages: [ 'english', 'german' ]
  }
]   _id: ObjectId("62e2d3d5e75ce6a476c170af"),
test>
```

КҮРДЕЛІ СҰРАНЫСТАР

- Енді күрделірек сұранысты қарастырайық: бізге name = "Tom" және бір уақытта age = 32 болатын құжаттарды шығару керек. SQL тілінде бұл былай жазылар еді:SELECT * FROM Table WHERE Name='Tom' AND Age=32.MongoDB-де осы шартқа сәйкес құжатты алу үшін келесі сұранысты жазамыз:

```
1 db.users.find({name: "Tom", age: 32})
```

ДЕРЕКТЕРДІ ЖОҚ ӨРІСТЕР БОЙЫНША СҮЗГІЛЕУ

- Кейбір құжаттарда белгілі бір өріс болады, ал басқаларында ол өріс мүлде болмауы мүмкін. Егер біз **белгілі бір өрісі жоқ** құжаттарды тапқымыз келсе, онда бұл өріске **null** мәнін береміз.
- Мысалы, **languages** өрісі жоқ барлық құжаттарды табайық:

```
db.users.find({ languages: null })
```

Немесе **name="Tom"**, бірақ **languages** өрісі анықталмаған құжаттарды табу:

```
db.users.find({ name: "Tom", languages: null })
```

Ескерту: кей жағдайда null – өріс жоқты да, өріс бар бірақ мәні null екенін де қамтуы мүмкін.

МАССИВ ЭЛЕМЕНТТЕРІ БОЙЫНША СҮЗГІЛЕУ

- Массивтің ішіндегі белгілі бір элементті табу да оңай. Мысалы, languages массивінде "english" бар барлық құжаттарды шығару:
  - `db.users.find({ languages: "english" })`
- Енді сұранысты күрделендірейік: languages массивінде бір уақытта екі тіл – "english" және "german" болсын:
  - `db.users.find({ languages: ["english", "german"] })`
- Бұл жерде маңыздысы: тәртібі дәл осылай болуы керек, яғни "english" – бірінші, "german" – екінші болуы қажет.

МАССИВТЕГІ ЭЛЕМЕНТТІҢ ОРНЫН ТЕКСЕРУ

Енді "english" тілі **languages массивінің бірінші элементі** болатын құжаттарды шығарайық:

```
db.users.find({ "languages.0": "english" })
```

Мұнда "languages.0" – күрделі (құрама) өріс болғандықтан, тырнақшаға алынады.

Егер "english" екінші орында болса (мысалы ["german", "english"]), онда "languages.1" деп жазамыз:

```
db.users.find({ "languages.1": "english" })
```

МАССИВТЕГІ ЭЛЕМЕНТТІҢ ОРНЫН ТЕКСЕРУ

Енді "english" тілі **languages массивінің бірінші элементі** болатын құжаттарды шығарайық:

```
db.users.find({ "languages.0": "english" })
```

Мұнда "languages.0" – күрделі (құрама) өріс болғандықтан, тырнақшаға алынады.

Егер "english" екінші орында болса (мысалы ["german", "english"]), онда "languages.1" деп жазамыз:

```
db.users.find({ "languages.1": "english" })
```

MONGODB-ДЕ СҰРАНЫС НӘТИЖЕЛЕРІН **СҰРЫПТАУ ЖӘНЕ ШЕКТЕУ**

КІРІСПЕ

MongoDB үлкен әрі күрделі деректер жиынтығымен жұмыс істеуге арналған қуатты сұраныс мүмкіндіктерін ұсынады. Соның ішінде ең жиі қолданылатындарының бірі – **нәтижені сұрыптау (sorting)** және **қайтарылатын құжаттар санын шектеу (limiting)**.

- **Сұрыптау** – алынған құжаттарды белгілі бір ретпен (өсу/кему) орналастыру.
- **Шектеу** – сұраныс қанша құжат қайтаратынын басқару.

Бұл бөлімде sort() және limit() әдістерін негізгі және кеңейтілген қолдану мысалдарымен, өнімділік мәселелерімен және тиімді жұмыс істеу тәсілдерімен қарастырамыз.

MONGODB-ДЕ СҰРЫПТАУ (SORTING) ДЕГЕНІМІЗ НЕ?

Sorting – сұраныс нәтижелерін бір немесе бірнеше өріс бойынша реттеу. MongoDB-де бұған sort() әдісі жауап береді. Ол құжаттарды:

- уақыт бойынша реттеу (мысалы, оқиғаларды күнмен сұрыптау);
- әріп бойынша реттеу (A–Z немесе Z–A);
- сандық мән бойынша реттеу (баға, жас, рейтинг т.б.)үшін өте пайдалы.

Негізгі сұрыптау (Basic Sorting)sort() әдісі объект қабылдайды: әр кілт – өріс атауы, мәні – сұрыптау бағыты.

- 1 → өсу реті (кіші → үлкен, A → Z)
- -1 → кему реті (үлкен → кіші, Z → A)

Синтаксис:

```
db.collection.find().sort({ field1: 1, field2: -1 });
```

Мысал 1: Бір өріс бойынша сұрыптауusers коллекциясында age өрісі бар делік. Біз қолданушыларды жасы бойынша өсу ретімен шығарамыз (ең жас → ең үлкен).

Сұраныс:

```
db.users.find().sort({ age: 1 });
```

Мысалы:

Alice (25) → Bob (30) → Charlie (35).

БІРНЕШЕ ӨРІС БОЙЫНША СҰРЫПТАУ

MongoDB бірнеше өріс бойынша сұрыптай алады. Бұл әсіресе бірінші өрісте мәндері бірдей құжаттар болған кезде пайдалы (екінші критерий қосылады).

Мысалы: алдымен age бойынша өсу, ал жасы бірдей болса name бойынша кему (Z → A).

Сұраныс:

```
db.users.find().sort({ age: 1, name: -1 });
```

НӘТИЖЕНІ ШЕКТЕУ (LIMITING) ДЕГЕНІМІЗ НЕ?

limit() – қайтарылатын құжаттар санын шектейді. Бұл :

- ✓ парақтау (pagination) жасау;
- ✓ артық деректі тасымалдауды азайту;
- ✓ аналитика үшін тек бір бөлігін алуда өте пайдалы

Синтаксис:

```
db.collection.find().limit(n);
```

Мұндағы n – қайтарылатын құжаттар саны.

- Тек бір құжат алу

```
db.users.find().limit(1);
```

- Белгілі бір санға дейін шектеу

```
db.users.find().limit(2);
```

СҰРЫПТАУ МЕН ШЕКТЕУДІ БІРІКТІРУ (SORT + LIMIT)

Көбіне “ең үлкен”, “ең соңғы”, “ең қымбат” сияқты top-N нәтижелер керек болады.

Мысал 5: Ең үлкен жастағы 2 қолданушы

```
db.users.find().sort({ age: -1 }).limit(2);
```

Түсіндірме: age: -1 → жас бойынша кему реті (үлкен → кіші) limit(2) → тек 2 құжат қайтарады. Нәтижеде ең жасы үлкен 2 адам шығады.

ПАРАҚТАУ (PAGINATION): SKIP() + LIMIT()

Pagination жасау үшін көбіне skip() пен limit() бірге қолданылады.

- skip(n) → алғашқы n құжатты өткізіп жібереді
- limit(n) → кейінгі n құжатты қайтарады

Мысал 6: 2 құжаттан тұратын 2-парақ

```
db.users.find().sort({ age: 1 }).skip(2).limit(2);
```

Ескерту: үлкен коллекцияда skip() мәні тым үлкен болса, өнімділік төмендейді.

ҚОРЫТЫНДЫ

Қорытындылай келе, MongoDB-де sort() және limit() әдістері сұраныс нәтижелерін басқарудың ең маңызды мүмкіндіктерінің бірі болып табылады. sort() арқылы деректерді белгілі бір логика бойынша (жас, баға, дата, атау т.б.) реттеп, нәтижені түсінікті әрі ыңғайлы түрде ұсынуға болады. Ал limit() көмегімен қайтарылатын құжаттар санын бақылап, жүйенің жүктемесін азайтып, сұраныстың жылдам орындалуына ықпал етеміз.

Нәтижені беттеу кезінде skip() және limit() кең қолданылғанымен, үлкен коллекцияларда skip() тиімсіз болуы мүмкін, сондықтан мұндай жағдайда диапазондық фильтрлерге негізделген pagination анағұрлым тиімді шешім болып саналады.

Сонымен қатар сұрыптау және фильтрация қолданылатын өрістерге индекс құру – өнімділікті арттырудың негізгі шарты.Демек, дұрыс сұрыптау, нәтижені шектеу, тиімді беттеу және индекстерді орынды қолдану – MongoDB-де жылдам әрі тұрақты жұмыс істейтін қосымшалар жасаудың маңызды негізі.

MONGODB-ДЕГІ АГРЕГАЦИЯ

MONGODB-ДЕГІ AGGREGATION

MongoDB-дегі aggregation – NoSQL әлеміндегі ең пайдалы құралдардың бірі. Көп жағдайда бізге “деректерді қалай тез есептеймін, қалай топтаймын, қалай аналитика жасаймын?” деген сұрақтар кездеседі. Егер мұндай өңдеуді қолданба деңгейіне (application layer) шығарып жіберсек, онда:

- серверге түсетін жүктеме өседі;
- желі арқылы көп дерек тасымалданады;
- сұраныс баяулайды;
- ал жүйелік әкімші “серверді асқабаққа айналдырмайық” деп аландайды.

Aggregation framework осы мәселелерді шешеді: деректерді тікелей MongoDB ішінде сүзу, түрлендіру, топтау, есептеу, біріктіру (join) сияқты операцияларды орындауға мүмкіндік береді.

Үлкен көлемдегі деректермен жұмыс істейтін жобаларда MongoDB-ні орналастырғанда ресурсы жеткілікті орта (VPS немесе dedicated) таңдаудың маңызы артады, себебі агрегация ауыр есептеулерді сервер ішінде орындайды.

MONGODB-ДЕ АГРЕГАЦИЯ ҚАЛАЙ ЖҰМЫС ІСТЕЙДІ

MongoDB-дегі агрегация конвейер (pipeline) қағидасы бойынша жұмыс істейді. Мұны Unix pipes сияқты елестетіңіз, тек құжаттар (documents) үшін. Pipeline-дің әр кезеңі алдыңғы кезеңнен құжаттарды қабылдап, оларды өңдейді де, нәтижені келесі кезеңге өткізеді. Осылайша қарапайым операциялардан күрделі сұраныстарды құрастыруға болады.

Агрегациялық framework-тің негізгі компоненттері:

- **$match** – құжаттарды сүзу (SQL-дегі WHERE аналогы)
- **$group** – құжаттарды топтау (SQL-дегі GROUP BY аналогы)
- **$sort** – нәтижелерді сұрыптау
- **$project** – өрістерді таңдау және түрлендіру
- **$limit және $skip** – беттеу (pagination)
- **$lookup** – JOIN операциялары
- **$unwind** – массивтерді “жазу” (элементтерге бөлу)

ТӘЖІРИБЕ ҮШІН ТЕСТ ДЕРЕКТЕР

Агрегациямен тәжірибе жасаудан бұрын, тесттік деректер бар екеніне көз жеткізіңіз. Көрсету үшін қарапайым коллекция жасайық:

```
db.orders.insertMany([
  { _id: 1, customer: "Alice", amount: 100, status: "completed", date: new
Date("2023-01-15") },
  { _id: 2, customer: "Bob", amount: 250, status: "pending", date: new Date("2023-
01-16") },
  { _id: 3, customer: "Alice", amount: 175, status: "completed", date: new
Date("2023-01-17") },
  { _id: 4, customer: "Charlie", amount: 300, status: "completed", date: new
Date("2023-01-18") },
  { _id: 5, customer: "Bob", amount: 150, status: "cancelled", date: new
Date("2023-01-19") }
])
```

ҚАРАПАЙЫМ МЫСАЛ: СҮЗУ ЖӘНЕ СҰРЫПТАУ. АЛДЫМЕН АЯҚТАЛҒАН (COMPLETED) ТАПСЫРЫСТАРДЫ ШЫҒАРЫП, СОМАСЫ БОЙЫНША КЕМУ РЕТІМЕН СҰРЫПТАЙМЫЗ:

```
db.orders.aggregate([
  { $match: { status: "completed" } },
  { $sort: { amount: -1 } }
])
```

$match ерте қолданылса – өңделетін құжат азаяды, сұраныс жылдамырақ.

Топтау және есептеулер. Клиент бойынша сатылым статистикасын есептеу:

Мұнда біз бірден бірнеше метрика аламыз: жалпы сома, тапсырыс саны, орташа чек.

```
db.orders.aggregate([
  { $match: { status: "completed" } },
  { $group: {
    _id: "$customer",
    totalAmount: { $sum: "$amount" },
    orderCount: { $sum: 1 },
    avgAmount: { $avg: "$amount" }
  }}
])
```

ПРАКТИКАЛЫҚ МЫСАЛДАР ЖӘНЕ КЕЙСТЕР

Кейс 1: Айлар бойынша сатылым аналитикасы

- Ай сайынғы сатылым бойынша есеп құру керек делік. Мұны былай әдемі жасауға болады:

```
db.orders.aggregate([
  { $match: { status: "completed" } },
  { $project: {
    amount: 1,
    month: { $month: "$date" },
    year: { $year: "$date" }
  }},
  { $group: {
    _id: { year: "$year", month: "$month" },
    totalSales: { $sum: "$amount" },
    orderCount: { $sum: 1 }
  }},
  { $sort: { "_id.year": 1, "_id.month": 1 } }
])
```

$project арқылы уақытты “жыл/ай” формасына келтіріп, кейін $group жасаймыз.

КЕЙС 2: ТОЛЫҚ ДЕТАЛИЗАЦИЯСЫ БАР ТОР-КЛИЕНТТЕР

Егер клиенттер бөлек customers коллекциясында тұрса, тапсырысты клиент мәліметімен біріктіреміз.

```
db.orders.aggregate([
  { $match: { status: "completed" } },
  { $group: {
    _id: "$customer",
    totalSpent: { $sum: "$amount" },
    orderCount: { $sum: 1 },
    avgOrderValue: { $avg: "$amount" },
    lastOrderDate: { $max: "$date" }
  }},
  { $sort: { totalSpent: -1 } },
  { $limit: 10 }
])
```

КЕЙС 3: $LOOKUP ҚОЛДАНЫЛҒАН КҮРДЕЛІ МЫСАЛ

Бізде клиенттер үшін бөлек коллекция бар делік. Деректерді біріктірейік:

Алдымен клиенттер коллекциясын жасаймыз

```
db.customers.insertMany([
  { _id: "Alice", email: "alice@example.com", city: "New York" },
  { _id: "Bob", email: "bob@example.com", city: "Los Angeles" },
  { _id: "Charlie", email: "charlie@example.com", city: "Chicago" }
])

// Теперь JOIN-запрос
db.orders.aggregate([
  { $match: { status: "completed" } },
  { $lookup: {
    from: "customers",
    localField: "customer",
    foreignField: "_id",
    as: "customerInfo"
  }},
  { $unwind: "$customerInfo" },
  { $group: {
    _id: "$customerInfo.city",
    totalRevenue: { $sum: "$amount" },
    customerCount: { $addToSet: "$customer" }
  }},
  { $project: {
    city: "$_id",
    totalRevenue: 1,
    customerCount: { $size: "$customerCount" }
  }}
])
```

Маңызды: $lookup дұрыс қолданылмаса сұраныс қатты баяулауы мүмкін, сондықтан индекстер және сүзуді ерте жасау – міндетті.

ӨНІМДІЛІК ЖӘНЕ ОҢТАЙЛАНДЫРУ

Тәсіл	Артықшылықтары	Кемшіліктері	Қашан қолдану керек
Ерте сүзу ($match-ті басына қою)	Дерек көлемін азайтады, тез орындалады	Кей күрделі логикаға сәйкес келмеуі мүмкін	Мүмкін болған сайын әрдайым
Индекстерді қолдану	Операцияларды айтарлықтай жылдамдатады	Қосымша орын алады	Жиі қолданылатын өрістер үшін
allowDiskUse: true	Үлкен dataset-терді өңдеуге мүмкіндік береді	Баяуырақ жұмыс істейді	Дерек көлемі үлкен болғанда
Артық өрістерді алып тастау үшін $project	Жад пен желіні үнемдейді	Pipeline-ді күрделендіруі мүмкін	Үлкен құжаттармен жұмыс істегенде

ЖИІ ҚАТЕЛЕР

- $group-ті сүзусіз қолдану → бүкіл коллекцияны “сыпырып” өтеді.
- 100MB stage лимитін ұмыту → үлкен дерекке allowDiskUse керек.
- $limit алдында $sort жасамау → кездейсоқ “TOP” шығып кетеді.
- $lookup-ты индекстерсіз қолдану → join баяулайды.

АГРЕГАЦИЯНЫ ҚАШАН ҚОЛДАНУ КЕРЕК?

Aggregation framework-ті қолданамыз:

- есептер мен аналитикаға;
- топтау, суммалау, орташа мәндерге;
- клиентке жіберер алдында деректі “тазалау/түрлендіруге”;
- үлкен дерекпен жұмыс істеуге.