MongoDB -2

Jánosi-Rancz Katalin Tünde

Embedded documents –Beágyazott dokumentumok

1. subdocument

```
{
        "_id" : ObjectId("..."),
        "name" : {
                "first" : "Péter",
                "middle" : "István",
                "last" : "Kovács”
                  },
                // ...
}
```

Embedded documents –Beágyazott dokumentumok

2. array of embedded documents

```
{
        "_id" : ObjectId("..."),
        "title” : ”Halálos iramban”,
        ”release_date” : 2009,
        ”actors" : [
                {”name" : ”Vin Diesel”},
                {”name" : ”Paul Walker”},
                {”name" : ”Michelle Rodriguez”},
                ],
                // ...
}
```

Cursor

- iterátornak tekintendő, a visszaadott dokumentumok bejárására szolgál
- pl.1:

```
var tweets = db.tweets.find().limit(10), tweet;

while( tweets.hasNext() ){
        tweet = tweets.next();
        print("tweetek: " + tweet.text);
}
```

```
tweetek: Nobody don't know what we want to eat 😭
tweetek: @suput_w http://t.co/1yySnLnsM
tweetek: عطوني رابط للحفل!!!
tweetek: Noooooo so close Kelsey
…
```

Cursor 2

- Pl2.:

```
db.tweets.find().limit(10).forEach( function( doc ) { print("tweetek: " + doc.text); } );
```

tweetek: Nobody don't know what we want to eat 😂
tweetek: @suput_w http://t.co/1yySnLnsM
tweetek: !!!عطوني رابط للحفل
tweetek: Noooooo so close Kelsey
…

COUNT

- db.<collection>.**count**(***<criteria>***)

- A megszámlálás függvénye.
  - a ***<criteria>***-nak megfelelő doc-ok számát adja vissza
- pl.:

```
db.users.count({"age" : 21}); // => 3
```

Update operátorok

- **$inc**: megnöveli vagy csökkenti az adott mezőt , { $inc : { "points" : -5 } }
- **$set**: adott mező értékét módosítja, { $set : { "name" : "Julcsi" } }
- **$unset**: adott mezőt vagy mezőket törli a dokumentumból,

  { $unset : { " name" : "", "gender" : "" }
- **$pull**: a megadott mezőt vagy mezőket törli a tömbből, { $pull : { "votes" : 7 } }
- **$push**: a megadott értéket a tömbbe helyezi, { $push : { "scores" : 89 } }
- **$**: a kritériának megfelelő doc-ok embedded array első elemét módosítja,

  { $set : { "grades.**$**" : 82 } }

- Adatmodellezés

Adatmodellezés

- flexibilis séma -> dokumentum szerkezeteket nem írjuk le
- kulcs koncepciók:
  - az alkalmazásnak mire van szüksége
  - adatbázis motor teljesítmény karakterisztikája
  - adatelérési minták
  - alkalmazás adathasználata (queries, updates)
- dokumentum struktúra: az alkalmazás hogyan reprezentálja az adatok közti kapcsolatokat
- adatok közti kapcsolatok reprezentálásának 2 fajtája:
  - referenciák (references)
  - beágyazott dokumentumok (embedded documents)

Embedded data

- a kapcsolatok egyetlen egy dokumentumban tárolhatóak
- denormalizált adatmodell: egyetlen egy adatbázis műveletként lehetővé teszi az összefüggő adatok lekérdezését és manipulációját
- lehetővé teszi, hogy az összetartozó információ darabkák ugyanazon adatbázis „rekordban” legyenek tárolva
- a dokumentumok írása (módosítása) egyetlen egy atomi művelet
- dokumentumok növekedése: ha a lemezen tárolt dokumentum módosításkor a számára lefoglalt tárterületet

túllépi, akkor a háttértáron a MongoDB új helyet foglal a számára, s az előző pozícióját felszabadítja

- adathasználat és teljesítmény: gyakori adatbeszúrás (insert) esetén ajánlott a *capped collection (*Capped collections are fixed-size collections that support high-throughput operations that insert and retrieve documents based on insertion order. *)*
- a beágyazás olvasás esetén jobb teljesítményt nyújt

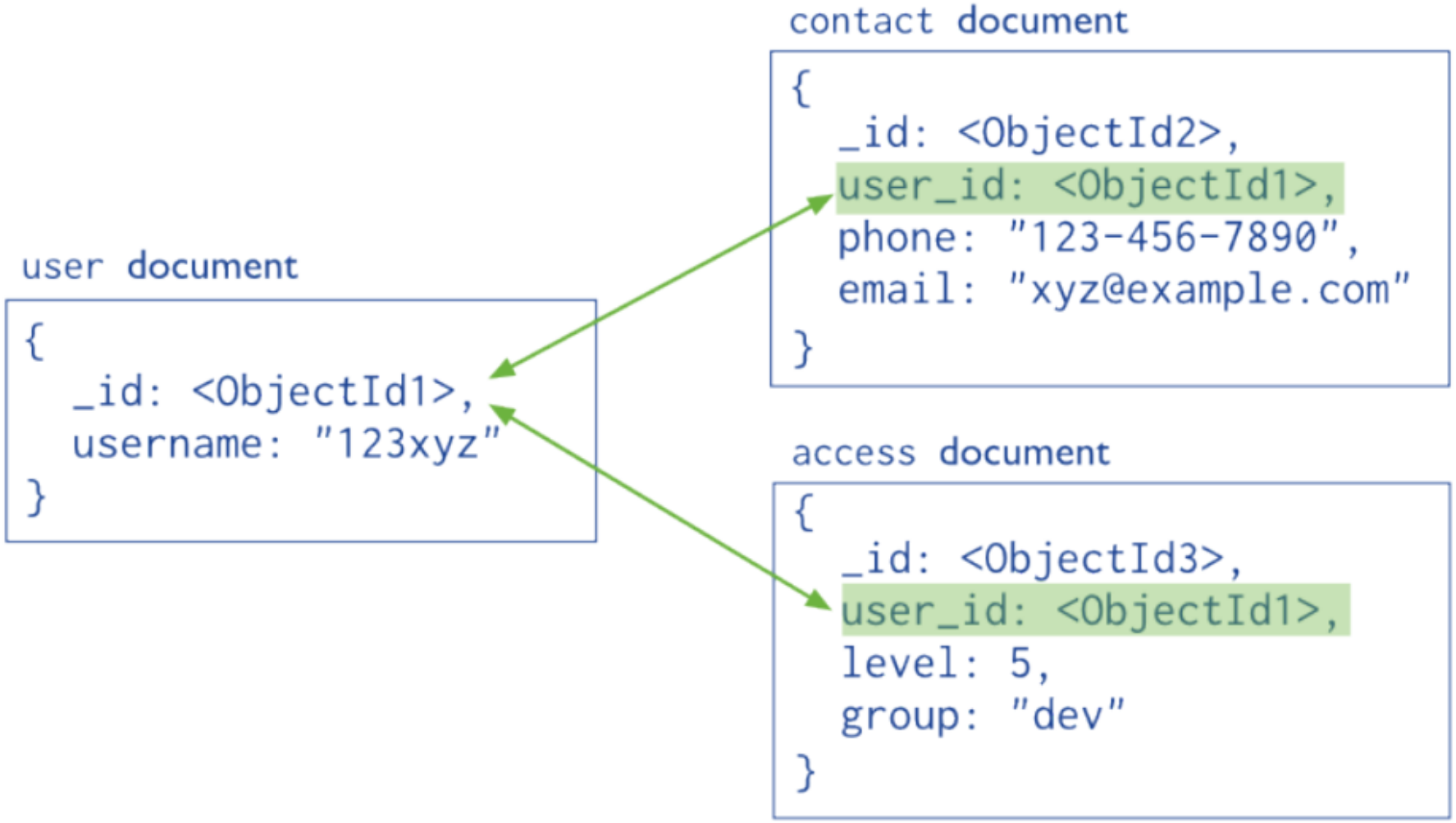
Adatmodellezés beágyazással

- denormalizált adatmodell

References

- a normalizált adatmodellek referenciák segítségével írják le a dokumentumok közötti kapcsolatokat
- akkor használatos, ha:
  - a beágyazás adat-duplikációt eredményezne, így nem szolgál az olvasás művelet előnyére
  - komplex sok-sok kapcsolat reprezentálása
  - hatalmas hierarchikus adathalmazok elkerülése érdekében
- nagyobb flexibilitást nyújtanak a beágyazásokhoz képest
- hátránya, hogy a lekérdezések és manipulációk több adatbázis műveletet igényelnek

Adatmodellezés referenciával



- normalizált adatmodell

One-to-One relationships

- Beágyazással:

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

One-to-Many relationships I.

Beágyazással:

```
{
    _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”
            }
                ]
}
```

One-to-Many relationships II.

- Referenciával:

```
{
    name: "O'Reilly Media", founded: 1980, location: "CA",
    books: [12346789, 234567890, ...] }
{
    _id: 123456789, title: "MongoDB: The Definitive Guide",
    author: [ "Kristina Chodorow", "Mike Dirolf" ] }
{
    _id: 234567890, title: "50 Tips and Tricks for MongoDB Developer",
    author: "Kristina Chodorow” }
```

Many-to-Many relationships

- Kapcsoló collection-nel:
- *felhasznalok collection egy dokumentuma:*

  { **_id : ObjectId(“A”),** nev : “Eleonóra”, taj : “AAABBB111”, … }

- *autok collection egy dokumentuma:*

  { **_id : ObjectId(“123”),** tipus : “M6”, gyarto : “BMW”, szin : “fehér” }

- *felhasznalok__autok kapcsoló collection egy dokumentuma:*

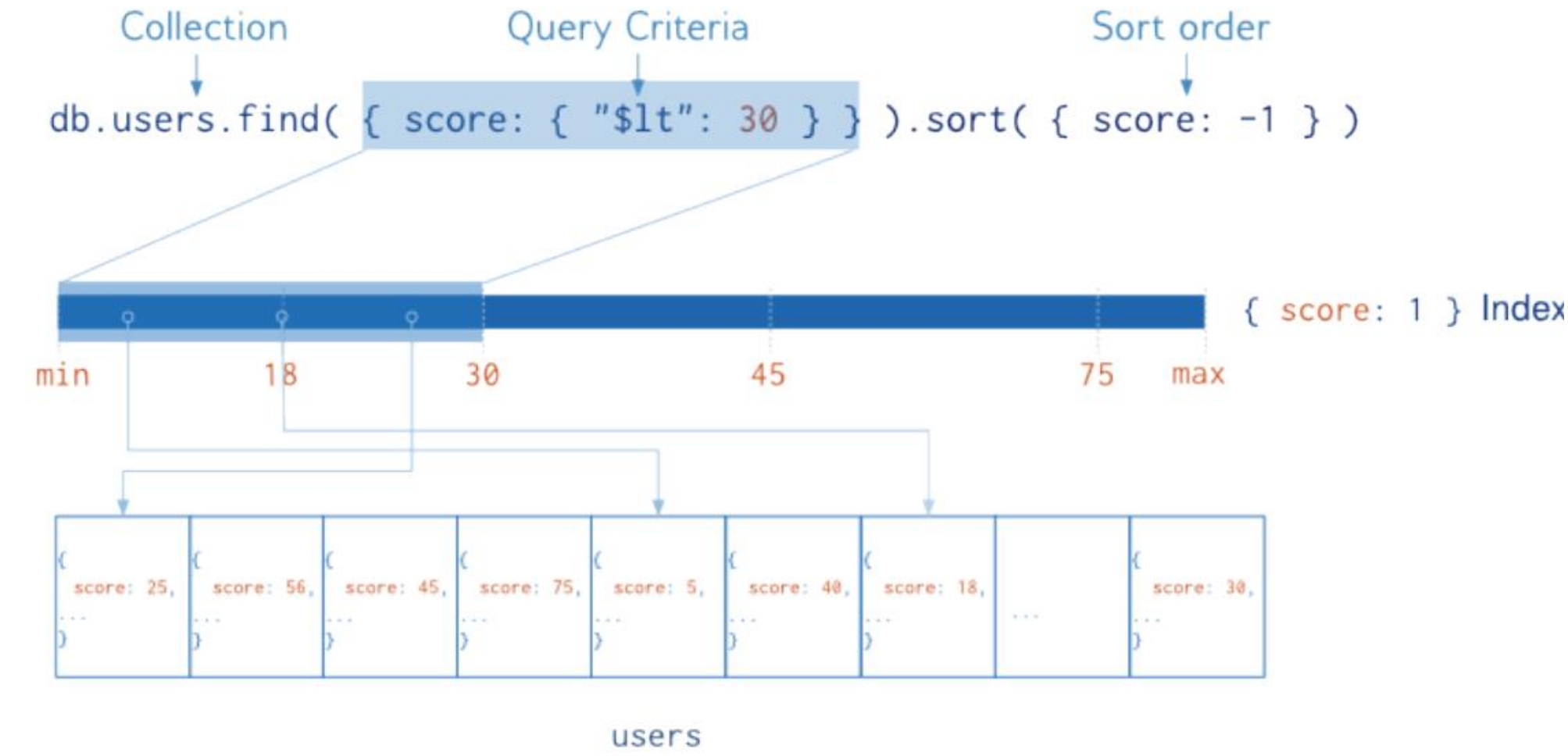
  { _id : ObjectId(“4af9f23d8ead0e1d32000000”),
  _id__felhasznalo : **ObjectId(“A”)**,
  _id__auto : **ObjectId(“123”)**
  }

Indexelés

Indexelés

- lekérdezések hatékony végrehajtása
- indexelés nélkül: *collection scan* (a collection összes dokumentumának átfésülése)
- az indexek B-fa struktúrát használnak
- a collection adathalmazának kis porcióját tartalmazzák
- valamely mezőnek vagy mezőknek az értékét tárolja, a mező értéke alapján rendezett módon
- *covered query*: ha a kritéria és a projekció csakis az indexelt mező(ke)t tartalmazza, akkor az eredmény az index-ből kerül ki, nem lesz keresés (find)

Indexelés működése



Indexek típusai

- _id
- single field (single field, embedded field, embedded document)
- compound index (multiple fields)
- multikey index (arrays)
- geospatial index (geo koordináták)
- text indexes
  - Case Insensitivity (é, É)
  - Diacritic Insensitivity (e, E, é, É)
  - Tokenization Delimiters
- 2d ([x, y])
- 2dsphere ({ type: "Point", coordinates: [x, y] })
- hashed indexes (hashing function)

Indexek tulajdonságai

- unique (duplikált értékek nem fordulhatnak elő)
- sparse (csak azon doc-ok indexelése, melyek rendelkeznek az indexelt mezővel)
- TTL (doc-ok automatikus törlése a collection a megadott idő letelte után)
- partial (csak adott filter-re [criteria-ra] illeszkedő dokumentumok lesznek indexelve)
- case insensitive (kis- és nagybetűk megkülönböztetése)

Indexek lekérdezése

- db.<collection>.**getIndexes**( )

- az adott collection összes index-ét kilistázza

Index létrehozása

- db.<collection>.**createIndex**(***<attributes>, <options>***)
- **<attributes>:** JSON, melyben az attribútum-rendezés kulcs-értékek vannak felsorolva
- egyszerre akár több attribútum is megadható
- ertékként az 1 (ASC) vagy a -1 (DESC) adható
    - db.cars.createIndex({ “gyarto” : 1 });
    - db.users.createIndex({ “name.first” : 1 });
    - db.users.createIndex({ “taj” : 1, “szuletesi_datum” : -1 });
    - db.users.createIndex({ “email” : 1 }, { unique : true });

Index létrehozása –2

- db.<collection>.**createIndex**(***<attributes>, <options>***)

- options JSON:
  - **{ name : " " }** : az index neve, ha nem lesz megadva, akkor automatikusan generálódik
  - **{ background : true }** : az index létrehozása a háttérben, így nem blokkolja az adatbázis műveleteket
  - **{ sparse : true }** : csak azon dokumentumokat indexeli, amelyek rendelkeznek az <attributes>-ban megadott attribútumokkal
  - **{ expireAfterSeconds : <secs> }** : a megadott másodperc letelte után automatikusan törlődnek a dokumentumok a collection-ből
  - **{ unique : true }** : az <attributes>-ban megadott kulcsok értékei csakis egyediek lehetnek (alapértelmezetten false az értéke)
  - **{ partialFilterExpression : {} }** : csakis a JSON-ban megadott criteria-ra illeszkedő dokumentumok kerülnek indexelésre

Index törlése

- db.<collection>.**dropIndex**(***<attributes>***)

- **<attributes>:** JSON, melyben az indexelt attribútumok vannak felsorolva a sorrendbe rendezésük típusával együtt

Aggregation Framework

Aggregation Framework

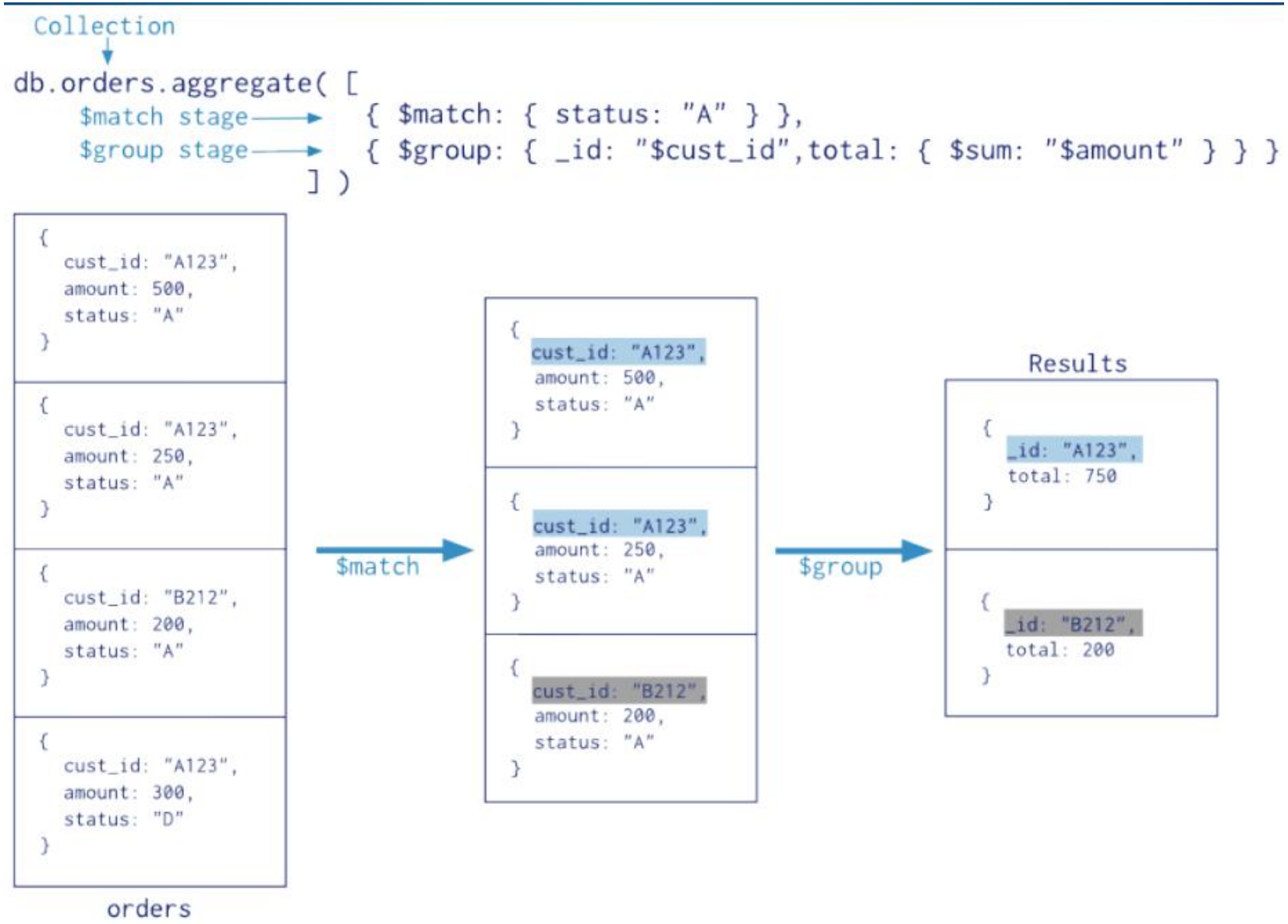
- aggregációk: olyan műveletek, amelyek az adatokon valamilyen folyamatot hajtanak végre, majd visszaadják a számított eredményt.

- Aggregation Pipeline: a dokumentumok több szakaszú csővezetéken mennek keresztül, amelyben aggregált eredménnyé transzformálódnak

- alapvető szakaszai:
  - *filters*: lekérdezésekként működnek
  - *document transformations*: módosítják a kimeneti dokumentumok formáját

- az adat-aggregáció natív műveletekkel hajtódik végre
- Map-Reduce is a része az Aggregation Framework-nek

Működése


Collection
db.orders.aggregate([
$match stage
{ $match: { status: "A" } },
$group stage
{ $group: { _id: "$cust_id",total: { $sum: "$amount" } } }
])
{ cust_id: "A123", amount: 500, status: "A" }
{ cust_id: "A123", amount: 250, status: "A" }
{ cust_id: "B212", amount: 200, status: "A" }
{ cust_id: "A123", amount: 300, status: "D" }
orders
$match
{ cust_id: "A123", amount: 500, status: "A" }
{ cust_id: "A123", amount: 250, status: "A" }
{ cust_id: "B212", amount: 200, status: "A" }
$group
Results
{ _id: "A123", total: 750 }
{ _id: "B212", total: 200 }

Aggregation Pipeline

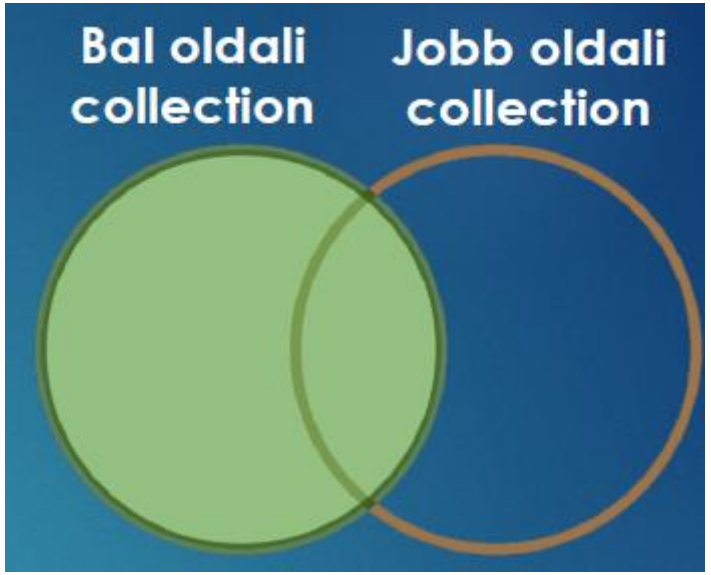
- Map-Reduce alternatívaként szolgál
- az aggregációs feladatokat hivatott megoldani abban az esetben, ha a mapreduce számára túl komplexek lennének
- szakaszokból épül fel -> minden egyes szakasz transzformálja a rajta átmenő dokumentumokat
- a szakaszok mindig csak az aktuális dokumentumhoz férnek hozzá
- optimalizálási szakasszal is rendelkezik, mely megkísérli átalakítani a pipeline-t a jobb teljesítmény érdekében
- eredményül egy szimpla dokumentumot ad, mely hibaleírást és az eredményt tartalmazza (16MB méret korlát!!)

Aggregation Pipeline használata

- db.<collection>.**aggregate**([{ <stage1> }, { <stage2> }, ...])

- szakaszok:
  - **$project**: projekcióként szolgál (1 vagy -1 érték)
  - **$match**: a find()-nál megszokott kritéria, szűrést valósít meg
  - **$redact**: különleges állatfajzat, említésre méltó csak jelen esetben
  - **$limit**, **$skip**: darabszám limitálása, kihagyása
  - **$unwind**: tömbökbe beágyazott dokumentumok külön dokumentumokként való megjelenítése
  - **$group**: csoportosítást valósít meg
  - **$sort**: újrarendezi a dokumentum streamet adott kulcs alapján
  - **$lookup**: left outer join, több collection-ben lévő adatok összekapcsolása
  - **$out**: ezzel a névvel egy új collection-t hoz létre, s ebbe helyezi az eredményt

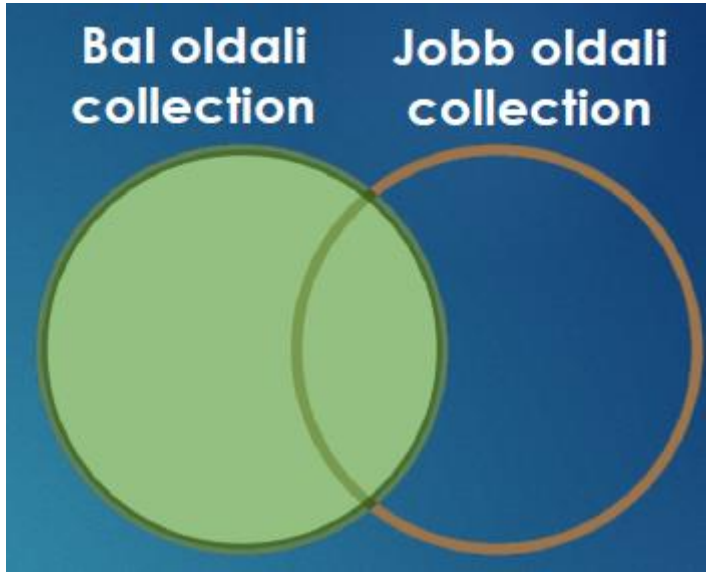
LEFT OUTER JOIN



- **$lookup** aggregációs művelet

- 3.2-es MongoDB verzió óta érhető el
- az Aggregation Framework része: a **$lookup** kulcsszó valósítja meg
- lehetővé teszi 2 collection-ben lévő dokumentumok összekapcsolását külső/idegen kulcs alapján
- fontos, hogy az idegen kulcs egy szimpla érték (pl. Number, String, ObjectId()) legyen, ne tömb vagy JSON
- **LEFT OUTER JOIN**: a bal oldali collection összes dokumentuma megjelenik, míg a jobb oldali collection dokumentumai közül csak azok, amelyek kapcsolódnak a bal oldali collection dokumentumaihoz külső/idegen kulcs alapján

LEFT OUTER JOIN



- **$lookup** használata:

```
db.<bal_oldali_collection_neve>.aggregate( [ {
        "$lookup" : {
                "localField" : "<bal_oldali_collection_hivatkozó_kulcsa>",
                "from" : "<jobb_oldali_collection_neve>",
                "foreignField" : "<jobb_oldali_collection_hivatkozott_kulcsa>",
                "as" : "<joinolt_adatok_ezen_kulcsnéven_jelennek_meg>"
                }
} ] );
```

LEFT OUTER JOIN - példa

- **Autók felvétele:**

```
db.autok.insertMany( [
    { "_id" : 1, "szin" : "fekete", "gyarto" : "Volkswagen", "tipus" : "Passat" },
    { "_id" : 2, "szin" : "fekete", "gyarto" : "BMW", "tipus" : "X5" },
    { "_id" : 3, "szin" : "fehér", "gyarto" : "Suzuki", "tipus" : "SX4" },
    { "_id" : 4, "szin" : "piros", "gyarto" : "Lada", "tipus" : "Samara" },
    { "_id" : 5, "szin" : "ezüst", "gyarto" : "Peugeot", "tipus" : "208" }
] );
```

LEFT OUTER JOIN - példa

- **Tulajdonosok felvétele:**

```
db.tulajdonosok.insertMany( [
        {
        "nev" : "Nagy László", "kor" : 34, "_id__autok" : [ 2, 5 ]
        },
        {
        "nev" : "Kiss József", "kor" : 41, "_id__autok" : [ 4 ]
        }
] );
```

LEFT OUTER JOIN - példa

- **JOIN művelet futtatása $lookup használatával:**

```
db.tulajdonosok.aggregate( [
    { "$unwind" : "$_id__autok” }
    , {
    "$lookup" : {
            "from" : "autok",
            "localField" : "_id__autok",
            "foreignField” : "_id",
            "as" : "joinoltAdat”
    }
}
] );
```

LEFT OUTER JOIN - példa

- **A $lookup művelet eredménye:**

[{ "_id" : ObjectId("57eb01ab59b732e510cfea5c"),
"nev" : "Nagy László", "kor" : 34, "_id__autok" : 2,
"joinoltAdat" : [{ "_id" : 2, "szin" : "fekete", "gyarto" : "BMW", "tipus" : "X5” }]
}
, { "_id" : ObjectId("57eb01ab59b732e510cfea5c"),
"nev" : "Nagy László", "kor" : 34, "_id__autok" : 5,
"joinoltAdat" : [{ "_id" : 5, "szin" : "ezüst", "gyarto" : "Peugeot", "tipus" : "208” }]
}
, { "_id" : ObjectId("57eb01ab59b732e510cfea5d"),
"nev" : "Kiss József", "kor" : 41, "_id__autok" : 4,
"joinoltAdat" : [{ "_id" : 4,"szin" : "piros", "gyarto" : "Lada", "tipus" : "Samara” }]
}]

Felhasznált irodalom:

- *Hugyák Tamás:* MongoDB FOR GIANT IDEAS