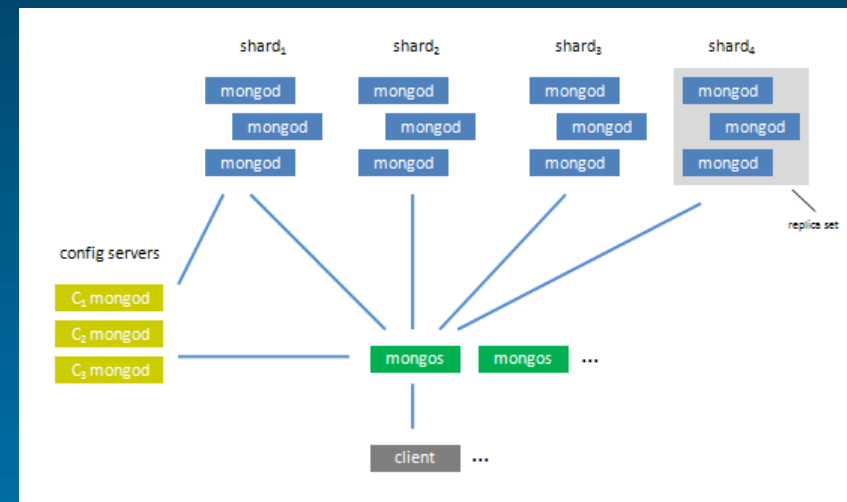


MongoDB – dokumentni nerelacijski SUPB

- dokumentno-usmerjen SUPB (JSON/BSON)
- lasten komunikacijski protokol
- konzola uporablja JavaScript sintakso
- podatkovna baza -> zbirke dokumentov (Collections) -> dokumenti (Documents)
- Porazdelitev in replikacija podatkov v oblikah master-slave ali delitev (sharding)
- sistem ne uporablja stikov podatkov (vendar so možni v uporabniškem programu)
- Osnovni (BSON) podatkovni tipi: null, boolean, 32/64-bit integer, double (64-bit FP), **decimal128** (128-bit BCD izpeljanke), string, datum/ura (timestamp)



- polna postavitve sistema:
 - mongod - strežnik podatkov
 - mongos - usmerjevalnik k podmnožici podatkov
 - strežniki so lahko replicirani (redundantnost)
 - ločena konfiguracija od podatkov (konfiguracijski strežniki)

JSON in BSON dokumentna formata

▪ **JSON (JavaScript Object Notation)**

- Semantično je dober približek za JSON Pythonov slovar (dict)
- Tekstoven format, berljiv za ljudi, a potraten
- Bolj ali manj standarden format za izmenjavo podatkov na spletu.
- Podpira omejen nabor podatkovnih tipov (string, število, boolean, null, polje)

▪ **BSON (Binary JSON)**

- Binaren format, optimiziran za shranjevanje in obdelavo.
- Ni berljiv za ljudi.
- Podpira več podatkovnih tipov kot JSON (npr. datum, binarni podatki).
- Običajno mnogo hitrejši za obdelavo kot JSON.
- Čeprav MongoDB interno uporablja BSON, še vedno lahko komunicira z uporabo JSON (samodejno pretvarja JSON↔BSON).

Namestitev

- <http://www.mongodb.org/>, prenos "community" različice (oktober 2026: 9.0.2 ali 8.3.11); platforma ali Docker vsebnik(i)
- mongod - strežniški program
 - pot do podatkov: argument --dbpath ali uporaba konfiguracijske datoteke in --config
 - strežnik posluša na vratih 27017
 - spletni administrativni strežnik posluša na vratih 28017
- mongosh - odjemalec (ukazna vrstica); compass – GUI okolje
- uporaba JSON in JavaScript sintakse (specifikacija dokumentov, poizvedb, skripte, ...)
- enostaven dostop od drugod, npr. Python - pymongo

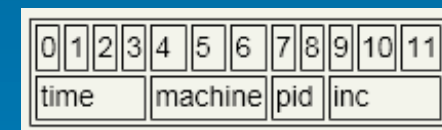
```
> mongod --config mongod.conf
Mon Nov 05 16:20:46 [initandlisten] MongoDB starting : pid=8384 port=27017 dbpath=c:\mongodb\data 64-bit host=quaddrix
Mon Nov 05 16:20:46 [initandlisten] db version v2.2.0, pdfile version 4.5
Mon Nov 05 16:20:46 [initandlisten] git version: f5e83eae9cfbec7fb7a071321928f00d1b0c5207
Mon Nov 05 16:20:46 [initandlisten] build info: windows sys.getwindowsversion(major=6, minor=1, build=7601, platform=2,
service_pack='Service Pack 1') BOOST_LIB_VERSION=1_49
Mon Nov 05 16:20:46 [initandlisten] options: { config: "mongod.conf", dbpath: "c:\mongodb\data" }
Mon Nov 05 16:20:46 [initandlisten] journal dir=c:/mongodb/data/journal
Mon Nov 05 16:20:46 [initandlisten] recover : no journal files present, no recovery needed
Mon Nov 05 16:20:46 [initandlisten] waiting for connections on port 27017
Mon Nov 05 16:20:47 [websvr] admin web console waiting for connections on port 28017
```

Primerjava pojmov

SQL	MongoDB
database (podatkovna baza)	database (podatkovna baza)
table (tabela)	collection (zbirka)
row (vrstica)	dokument JSON
column (stolpec)	JSON field (polje v dokumentu JSON)
primary key (primarni ključ)	polje _id v dokumentu JSON
indeks	indeks
group by	agregacija

Primarni ključ dokumenta: `_id`

- Striktneje: ekvivalent primarnega ključa
 - Imeti mora enolično določeno, nespremenljivo vrednost
 - Lahko je poljubnega tipa (razen array)
 - Lahko ga podamo pri kreiranju dokumenta
 - Če ga ne podamo, ga MongoDB kreira avtomatsko v obliki ObjectId
- ObjectId:
 - 12-bytni BSON (binarni JSON) zapis



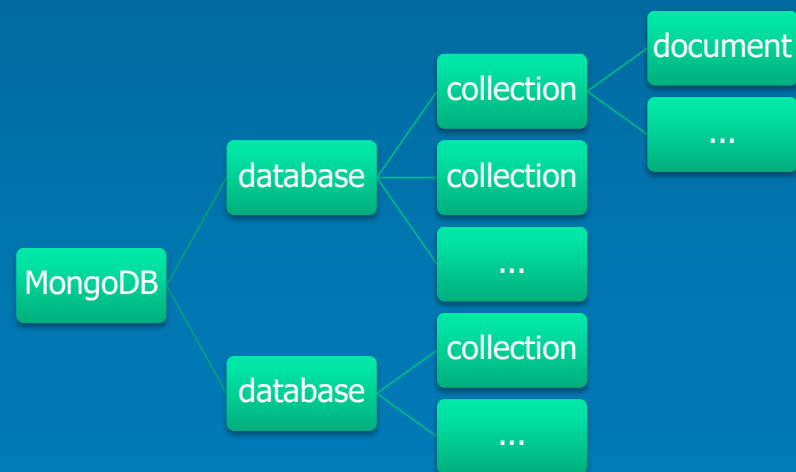
5-bytna psevdonaključna vrednost

Prvi koraki

- `show dbs` // seznam baz podatkov
- `show collections` // seznam zbirk
- `use <podatkovna_baza>` // uporabi podano bazo
- `db` // referenca do aktivne baze

- `db.help()` // pomoč
- `db.stats()` // statistike
- `db.version()` // različica

- lina izvedba: **ustvarjanje baz in zbirk** je **implicitno** ob ustvarjanju dokumentov



Delo z dokumenti

```
// ustvari ali uporabi bazo tup
use tup

// največja velikost dokumenta 16 MB (od 2018 dalje)
db.studenti.insertOne({ime: "janko", priimek: "novak"})
db.studenti.insertOne({ime: "peter", priimek: "klepec"})

db.studenti.deleteOne(kriterij)      // ali deleteMany
db.studenti.deleteOne({priimek: "klepec"})

db.studenti.drop()                  // brisanje cele zbirke (hitreje!)

db.studenti.findOne(kriterij)        // vrne prvi zadetek
db.studenti.find(kriterij)           // vrne kurzor na zadetke

// posodobi enega ali vse dokumente, ki ustrezajo kriteriju
db.studenti.updateOne(kriterij, novi_dokument) // ali updateMany
db.studenti.updateOne({priimek: "klepec"}, {$set: {ime: "pavel", starost: 25}})

// To je JavaScript sintaksa, pymongo sintaksa je malenkost drugačna.
```

Iskanje – primerjalni operatorji in modifikatorji



- `$eq/$ne, $in/$nin, $gt/$lt, $gte/$lte, $and/$or/$not/$nor`
- `db.x.find({visina: {$gt: 180}})` // višina večja od 180
- `db.x.find({prijatelji: {$ne: "Bojan"}})` // prijatelj ni enak "Bojan"
- `db.x.find({teza: {$in: [70, 75]}})` // teža je v podani množici
- `db.x.find({teza: {$nin: [70, 75]}})` // teža ni v podani množici
- `db.x.find({$or: [{teza: {$gt: 80}},
 {visina: {$gt: 180}}]})` // disjunkcija
- `db.x.find({teza: {$not : {$mod: [5,0]}}})` // teža ni deljiva s 5
- `db.x.find({prijatelji: {$all: ["Bojan", "Teo"]}})` // vsebovanost podmnožice
- `db.x.find({prijatelji: {$size: 3}})` // velikost polja je 3
- `db.x.find({}, {prijatelji: {$slice: [0,2]}})` // izberi rezino (del) polja
- `db.x.find({polje : {$exists : 1}})` // izberi, če polje obstaja

Posodabljanje (updateOne/updateMany)



```
db.x.updateOne({ime: "Eva"}, {nov dokument}) // zamenjaj cel dokument
db.x.updateOne({ime: "Eva"}, {$set : {mama : "Ana"}}) // nastavi element
db.x.updateOne({ime: "Eva"}, {$unset : {mama : 1}}) // odstrani element
db.x.updateOne({ime: "Eva"}, {$inc : {starost : 5}}) // inkrement elementa
db.x.updateOne({ime: "Eva"}, {$push : {prijatelji : "Janez"}}) // dodaj v polje
db.x.updateOne({ime: "Eva"}, {$addToSet : {loto : 42}}) // dodaj brez ponav.
db.x.updateOne({ime: "Eva"}, {$pop : {loto : 1}}) // odstrani s konca polja
db.x.updateOne({ime: "Eva"}, {$pop : {loto : -1}}) // odstrani z začetka polja
db.x.updateOne({ime: "Eva"}, {$pull : {loto : 15}}) // odstrani iz polja

db.x.updateOne( {ime: "Eva"}, {$inc : {starost : 5}}, {upsert:true}) // upsert ...
db.x.updateMany({ime: "Eva"}, {$inc : {starost : 5}}, {upsert:true}) // ... za vse zadetke
```

Gnezdeni dokumenti

- možno gnezdenje dokumentov in iskanje po gnezdenih seznamih (npr. knjige.avtorjev)

```
doc = { ime : "Janko",  
        priimek : "Novak",  
        knjige : [  
            { avtorjev : 3, strani : 100},  
            { avtorjev : 5, strani : 50},  
            { avtorjev : 8, strani : 400}  
        ]  
    }
```

- // iskanje z identično vsebino
`db.primer.find({ime: {priimek: "Novak", prvo: "Janko"}})`
- // iskanje po poljih
`db.primer.find({"ime.prvo": "Janko", "ime.priimek": "Novak"})`
- // pogoji vezani na vsebino vsakega elementa
`db.primer.findOne({knjige: {$elemMatch: {avtorjev : 3, strani : 50 }}})`
- // pozicijsko iskanje po prvi knjigi v seznamu
`db.primer.findOne({"knjige.1.strani": 50})`
- // \$ nadomesti pozicijo najdenega dokumenta
`db.primer.findOne({"knjige.avtorjev" : 3, "knjige.strani": 50 }, {"knjige.$":1})`

Literatura

- Osnovna literatura
<https://docs.mongodb.org/manual>
- kratki (referenčni) povzetki:
<https://www.10gen.com/reference>
- MongoDB Fundamentals, Amit Phaltankar (2020)
<https://www.packtpub.com/en-us/product/mongodb-fundamentals-9781839210648>
- Python data persistence: With SQL and NOSQL Databases
<https://bpbonline.com/products/python-data-persistence-sql-and-nosql-programming-book-ebook>

Literatura

Queries and What They Match

<code>{a: 10}</code>	Docs where a is 10, or an array containing the value 10.
<code>{a: 10, b: "hello"}</code>	Docs where a is 10 and b is "hello."
<code>{a: {\$gt: 10}}</code>	Docs where a is greater than 10. Also <code>\$lt</code> (<), <code>\$gte</code> (>=), <code>\$lte</code> (<=), and <code>\$ne</code> (!=).
<code>{a: {\$in: [10, "hello"]}}</code>	Docs where a is either 10 or "hello."
<code>{a: {\$all: [10, "hello"]}}</code>	Docs where a is an array containing both 10 and "hello".
<code>{"a.b": 10}</code>	Docs where a is an embedded document with b equal to 10.
<code>{a: {\$elemMatch: {b: 1, c: 2}}}</code>	Docs where a is an array containing a single item with both b equal to 1 and c equal to 2.
<code>{\$or: [{a: 1}, {b: 2}]}</code>	Docs where a is 1 or b is 2.
<code>db.foo.find({a: /^m/})</code>	Docs where a begins with the letter "m".

The following queries cannot use indexes as of MongoDB v2.0. These query forms should normally be accompanied by at least one other query term which *does* use an index:

<code>{a: {\$nin: [10, "hello"]}}</code>	Docs where a is anything but 10 or "hello."
<code>{a: {\$mod: [10, 1]}}</code>	Docs where a mod 10 is 1.
<code>{a: {\$size: 3}}</code>	Docs where a is an array with exactly 3 elements.
<code>{a: {\$exists: true}}</code>	Docs containing an a field.
<code>{a: {\$type: 2}}</code>	Docs where a is a string (see bsonspec.org for more types).
<code>{a: /foo.*bar/}</code>	Docs where a matches the regular expression "foo.*bar".
<code>{a: {\$not: {\$type: 2}}}</code>	Docs where a is not a string. <code>\$not</code> negates any of the other query operators.

Literatura

SQL	MongoDB
SELECT * FROM users WHERE age <= 33	db.users.find({age: {\$lte: 33}})
SELECT * FROM users WHERE name LIKE '%Joe%'	db.users.find({name: /Joe/})
SELECT * FROM users WHERE name LIKE 'Joe%'	db.users.find({name: /^Joe/})
SELECT * FROM users WHERE age > 33 AND age < 40	db.users.find({age: {\$gt: 33, \$lt: 40}})
SELECT * FROM users ORDER BY name DESC	db.users.find().sort({name: -1})
SELECT * FROM users WHERE age = 32 AND name = 'Bob'	db.users.find({age: 32, name: "Bob"})
SELECT * FROM users LIMIT 10 SKIP 20	db.users.find().skip(20).limit(10)
SELECT * FROM users WHERE age = 33 OR name = 'Bob'	db.users.find({\$or:[{age:33}, {name: "Bob"}]})
SELECT * FROM users LIMIT 1	db.users.findOne()
SELECT DISTINCT name FROM users	db.users.distinct("name")
SELECT COUNT(*) FROM users	db.users.count()
SELECT COUNT(*) FROM users WHERE AGE > 30	db.users.find({age: {\$gt: 30}}).count()
SELECT COUNT(AGE) FROM users	db.users.find({age: {\$exists: true}}).count()
CREATE INDEX ON users (name ASC)	db.users.ensureIndex({name: 1})
CREATE INDEX ON users (name ASC, age DESC)	db.users.ensureIndex({name: 1, age: -1})
EXPLAIN SELECT * FROM users WHERE age = 32	db.users.find({age: 32}).explain()
UPDATE users SET age = 33 WHERE name = 'Bob'	db.users.update({name: "Bob"}, {\$set: {age: 33}}, false, true)
UPDATE users SET score = score + 2 WHERE name = 'Bob'	db.users.update({name: "Bob"}, {\$inc: {score: 2}}, false, true)
DELETE FROM users WHERE name = 'Bob'	db.users.remove({name: "Bob"})

Kurzorji

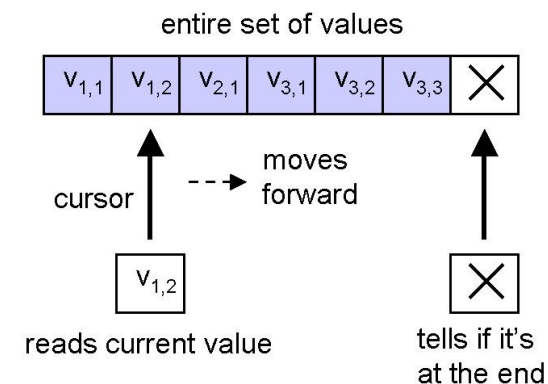
- kurzor - oblika iteratorja po podatkih v zbirki

```
var cur = db.evidenca.find()           // vrni kurzor
cur.forEach(function (x) {printjson(x); }) // iteracija po kurzorju
```

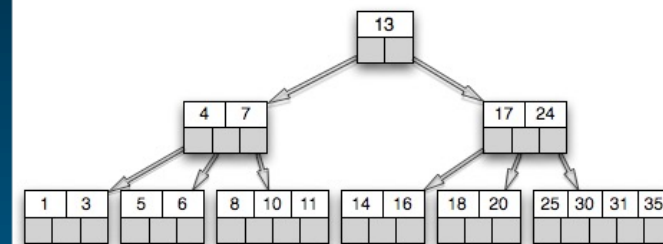
- limit, skip, sort
- limit in skip uporabna za odstranjevanje (pagination) v aplikacijah

```
db.evidenca.find().limit(3)           // omeji na prve 3 zapise
db.evidenca.find().skip(3)            // preskoči 3 zapise
db.evidenca.find().sort({starost : 1}) // sortiranje naraščajoče
db.evidenca.find().sort({visina : -1}) // sortiranje padajoče
```

```
db.evidenca.countDocuments()           // štetje zadetkov
db.evidenca.countDocuments({visina: {$gt : 170}}) // štetje s pogojem
db.evidenca.distinct("visina")         // seznam različnih
db.evidenca.estimatedDocumentCount()   // "približno" štetje dokumentov iz metapodatkov
```



Indeksiranje



- omejitev: največ 64 indeksov na zbirko
- indeks na $\{a : 1, b : 1, c : 1, \dots, z : 1\}$ pohitri delovanje poizvedb $\{a: 1\}, \{a : 1, b: 1\}, \{a : 1, b : 1, c : 1, \dots\}, \dots$
- možno tudi indeksiranje gnezdenih dokumentov
- podatkovna struktura indeksov: B-drevesa (kot pri relacijskih PB)

- `db.x.createIndex({visina: 1})` // izdelava indeksa
- `db.x.createIndex({visina: 1}, {name : "indy"})` // poimenovanje indeksa
- `db.x.dropIndex({visina: 1})` // brisanje indeksa
- `db.x.createIndex({visina: 1}, {unique : true})` // unikatni indeks (za ključe)

- `db.x.find({starost : 33}).explain()` // poda statistiko poizvedbe

Geoprostorsko indeksiranje

- iskanje točk, ki so po lokaciji (koordinatah) sorodne izvorni točki
- vsak dokument vsebuje en par podatkov, ki predstavljajo lokacijo;
- koordinate so običajno na intervalu od -180 do 180 (ustreza zemljepisni dolžini/širini)

```
{ "gps" : [ 0, 100 ] }  
{ "gps" : { "x" : -30, "y" : 30 } }  
{ "gps" : { "latitude" : -180, "longitude" : 180 } }
```
- izgradnja indeksa

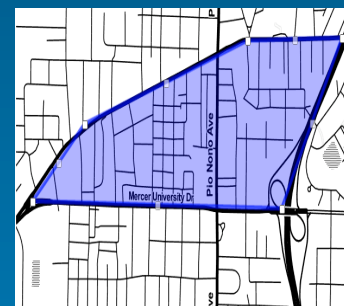
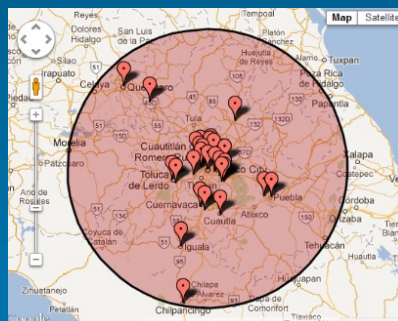
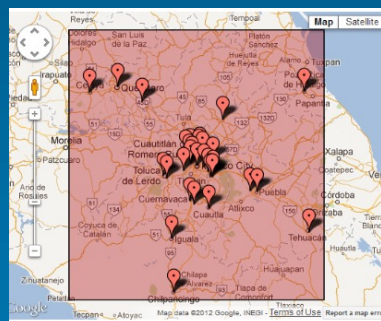
```
db.map.createIndex({ "gps" : "2d" })
```



Geoprostorsko indeksiranje

- iskanje:

- `db.map.find({"gps" : { $near : [40, -73] } })`
- `db.map.find({"gps" : { $within : { $box : [[10, 20], [15, 30]] } } })`
- `db.map.find({"gps" : { $within : { $center : [[12, 25], 5] } } })`
 - kot oblika možni tudi `$center`, `$circle` ali `$poly`



- možna kombinacija indeksov za optimizacijo iskanja po različnih poljih
 - `db.places.ensureIndex( { location : "2d" , category : 1 } );`
 - `db.places.find( { location : { $near : [50,50] }, category : 'coffee' } );`
- tipi indeksov: 2d – ravninska geometrija, 2dsphere – sferična geometrija (WGS84, World Geodetic System, 1984; uporablja GPS)

Kompleksne poizvedbe

- operator \$where
- pozor: počasna izvedba (pretvorba iz BSON v JavaScript, dela brez uporabe indeksov)

```
> db.primer.insert({jabolko: 1, banana : 6, breskev : 3})
> db.primer.insert({jabolko: 8, ananas : 4, lubenica : 4})

> db.primer.findOne({$where: function () {
...   for (var prvi in this) {
...     for (var drugi in this) {
...       if (prvi != drugi && this[prvi] == this[drugi]) return true;
...     }
...   }
...   return false;
... }})

{
  "_id" : ObjectId("5098b267b6979a0c0c662391"),
  "jabolko" : 8,
  "ananas" : 4,
  "lubenica" : 4
}
```

PyMongo – MongoDB odjemalec

- Instalacija v Python:
 - conda install pymongo (distribucija Anaconda)
 - pip install pymongo
- Navodila: <http://api.mongodb.org/python/current/tutorial.html>

Uporaba iz Pythona (pymongo)

- Bistvene razlike do JS konzole
 - zapis funkcij s podčrtaji (find_one namesto findOne ipd.)
 - slovarji ali BSON namesto JSON

```
from pymongo import MongoClient

client = MongoClient('localhost', 27017) # privzeti parametri
db = client.sample_mflix                # baza sample_mflix
db.list_collection_names()              # seznam vseh zbirk

db.movies.insert({'title': 'The Terminator Test'}) # dodajanje dokumenta

term = db.movies.find({'title': {'$regex' : '.*terminator.*', '$options': 'i'} })

for movie in
db.movies.find({'title':{'$regex':'.*terminator.*','$options':'i'}},{'year':1,'title':1}):
    print(movie['title'], movie['year']) # ... ali ...
    print(movie['title'], movie.get('year', '???'))
```

Iskanje, posodabljanje

```
query = {'title': {'$regex': '.*terminator.*', '$options': 'i'}}
```

```
# štetje zapisov - count
```

```
len(list(db.movies.find(query))) # 6
```

```
db.movies.count_documents(query) # 6
```

```
# omejitev števila zadetkov - limit
```

```
len(list(db.movies.find(query).limit(3))) # 3
```

```
# uporaba modifikatorjev, sortiranje
```

```
▪ for x in db.izpiti.find({"letnik": {"$gt": 1}}).sort("ime"): x
```

```
▪ # posodabljanje zapisov
```

```
▪ db.izpiti.update({"letnik": 1}, {"$inc": {"letnik": 1}}) # samo prvi
```

```
▪ db.izpiti.update({"letnik": 1}, {"$inc": {"letnik": 1}}, multi=True) # vseh zadetkov
```

```
▪ db.izpiti.update({"letnik": 1}, {"$inc": {"letnik": 1}}, upsert=True) # ustvari, če ne obstaja
```

Indeksi: navadni in geoprostorski

- # indeksiranje

```
db.izpiti.create_index([("letnik", ASCENDING), ("author", ASCENDING)])  
db.izpiti.drop_index([("letnik", ASCENDING), ("author", ASCENDING)])
```

```
db.izpiti.find({...}).explain()["cursor"]           # statistike iskanja  
db.izpiti.find({...}).explain()["nscanned"]
```

- # geoprostorsko indeksiranje

```
import bson  
loc = bson.SON()           # ohranja vrstni red elementov  
loc["x"]=3  
loc["y"]=4  
dok = {"ime": "Metka", "loc": loc}  
db.prostor.insert(dok)
```

```
db.prostor.create_index([("loc", GEO2D)])  
for doc in db.prostor.find({"loc": {"$near": [3, 6]}}).limit(3):  
    print(doc)  
for doc in db.prostor.find({"loc": {"$within": {"$box": [[0,0],[3,4]]}}}):  
    print(doc)
```

Primer (jadralci)

- # Jadralci
db.jadralec.insert_one({"jid": 22, "ime" : "Darko", "rating": 10, "starost":45})
Čolni
db.coln.insert(...) # ← ???
Rezervacije
db.rezervacija.insert_one({"jid": 22, "cid" : 101, "dan": "2006-10-10" })
db.rezervacija.insert_one({"jid": 22, "cid" : 102, "dan": "2006-10-10" })
db.rezervacija.insert_one({"jid": 22, "cid" : 103, "dan": "2006-10-8" })
db.rezervacija.insert_one({"jid": 22, "cid" : 104, "dan": "2006-10-7" })

Indeksi
db.jadralec.create_index("jid")
db.coln.create_index("cid")
db.rezervacija.create_index([("jid", pymongo.DESCENDING), ("cid", pymongo.ASCENDING), "dan"])
- # STIK: Poišči vse Darkove rezervacije
darko = db.jadralec.find_one({"ime" : "Darko"})
rez = db.rezervacija.find({"jid" : darko["jid"]})
for r in rez:
 print(r)

Jadralci v duhu MongoDB

Jadralec - (1,1)----(0,n)- Rezervacija

```
db.jadralec.update_one( {"jid": 22}, {"$set" : {"rezervacija" : []} })
db.jadralec.update_one( {"jid": 22}, {"$push" : {"rezervacija" : {"cid" : 101, "dan": "2006-10-10"} }} )
db.jadralec.update_one( {"jid": 22}, {"$push" : {"rezervacija" : {"cid" : 101, "dan": "2006-10-10"} }} )
db.jadralec.update_one( {"jid": 22}, {"$push" : {"rezervacija" : {"cid" : 103, "dan": "2006-10-8" } }} )
db.jadralec.update_one( {"jid": 22}, {"$push" : {"rezervacija" : {"cid" : 104, "dan": "2006-10-7" } }} )
db.jadralec.find_one({'jid':22})
```

```
{ '_id': ObjectId('670bcd3e863257cfab8c6530'),
  'jid': 22,
  'ime': 'Darko',
  'rating': 10,
  'starost': 45,
  'rezervacija': [{ 'cid': 101, 'dan': '2006-10-10'}, { 'cid': 101, 'dan': '2006-10-10'},
                  { 'cid': 103, 'dan': '2006-10-8'}, { 'cid': 104, 'dan': '2006-10-7'}]
}
```

Jadralci v duhu MongoDB

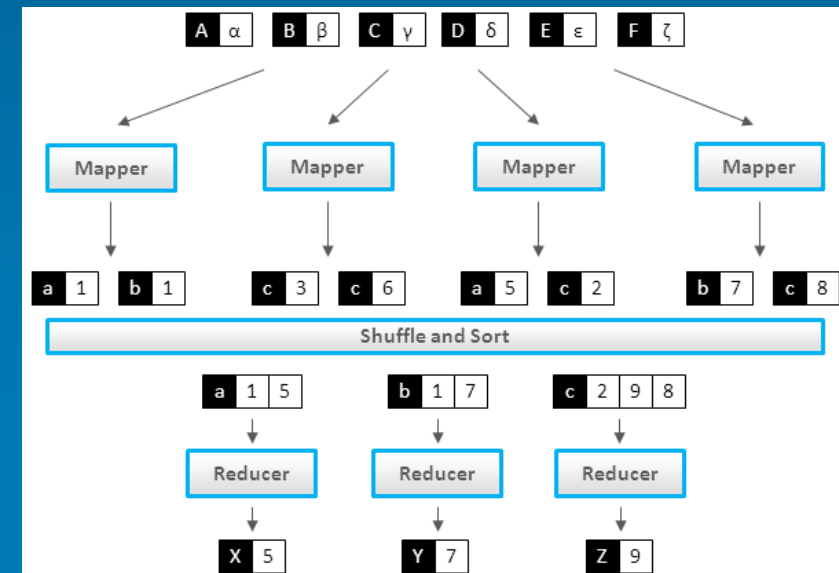
Jadralec - (1,1)----(0,n) - Rezervacija

```
db.jadralec.update_one( {"jid": 22}, {"$set" : {"rezervacija" : []} })
db.jadralec.update_one( {"jid": 22}, {"$push" : {"rezervacija" : {"cid" : 101, "dan": "2006-10-10"} }} )
db.jadralec.update_one( {"jid": 22}, {"$push" : {"rezervacija" : {"cid" : 101, "dan": "2006-10-10"} }} )
db.jadralec.update_one( {"jid": 22}, {"$push" : {"rezervacija" : {"cid" : 103, "dan": "2006-10-8" } }} )
db.jadralec.update_one( {"jid": 22}, {"$push" : {"rezervacija" : {"cid" : 104, "dan": "2006-10-7" } }} )
db.jadralec.find_one({'jid':22})
```

```
{ '_id': ObjectId('670bcd3e863257cfab8c6530'),
  'jid': 22,
  'ime': 'Darko',
  'rating': 10,
  'starost': 45,
  'rezervacija': [{ 'cid': 101, 'dan': '2006-10-10'}, { 'cid': 101, 'dan': '2006-10-10'},
                  { 'cid': 103, 'dan': '2006-10-8'}, { 'cid': 104, 'dan': '2006-10-7'}]
}
```

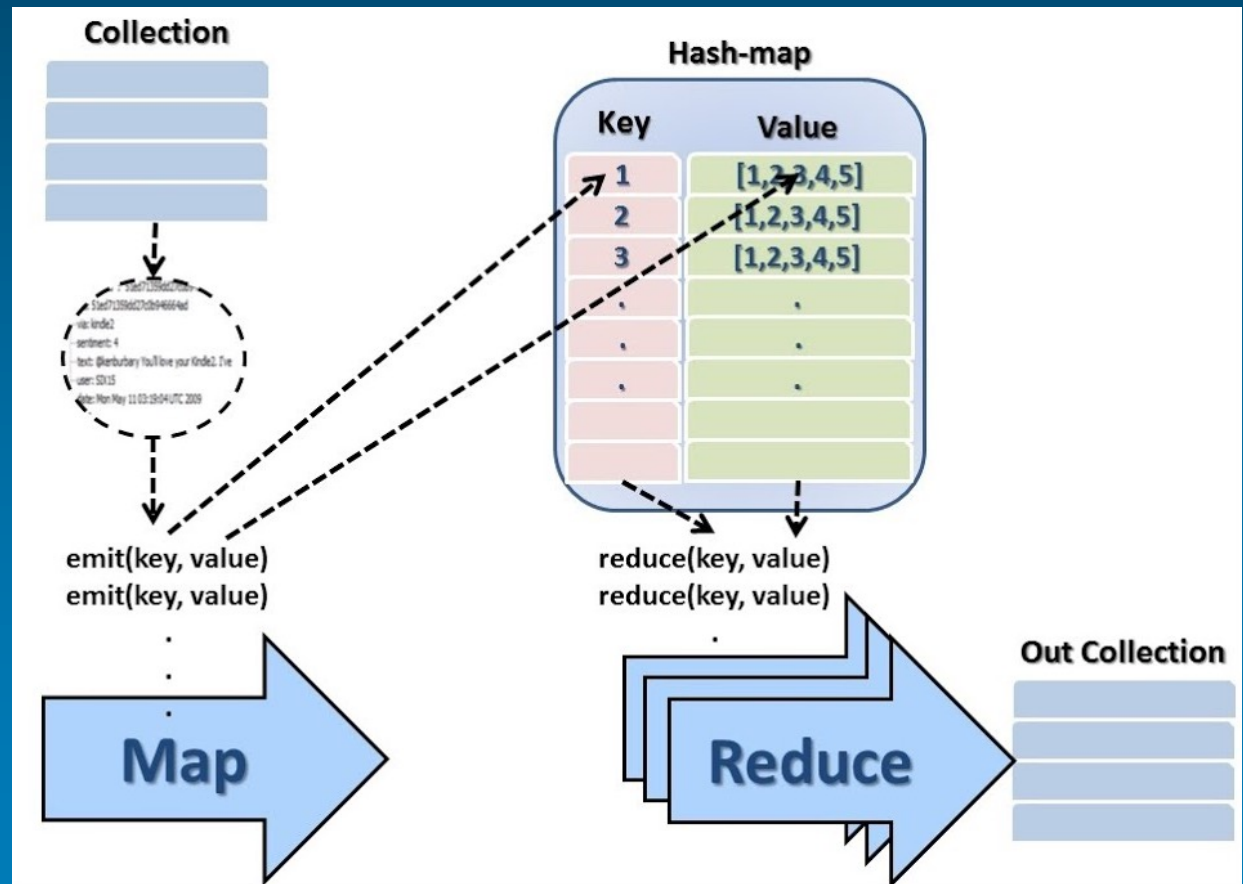
MapReduce

- postopek omogoča preprosto paralelizacijo programov (poizvedb) v velikih podatkovnih bazah preko velikega števila računalnikov/jeder/procesorjev
- splošen koncept, popularizirala Google in Apache Hadoop
- podobna ideja kot pri funkcijskem programiranju (map-fold)
- enostavna in učinkovita paralelizacija asociativnih *idempotentnih* funkcij
- velik **pretok** podatkov vendar previsoka **latenca** za izvajanje v realnem času



MongoDB MapReduce

- **map**: preslika vhodne dokumente v drugačno predstavitev (in/ali ključ) in jo/ju posreduje naprej s stavkom **emit**
- **reduce**: za vsak ključ prejme polje vrednosti od funkcije map in izračuna končni rezultat



MapReduce

- Podana je študentska evidenca opravljenih izpitov:

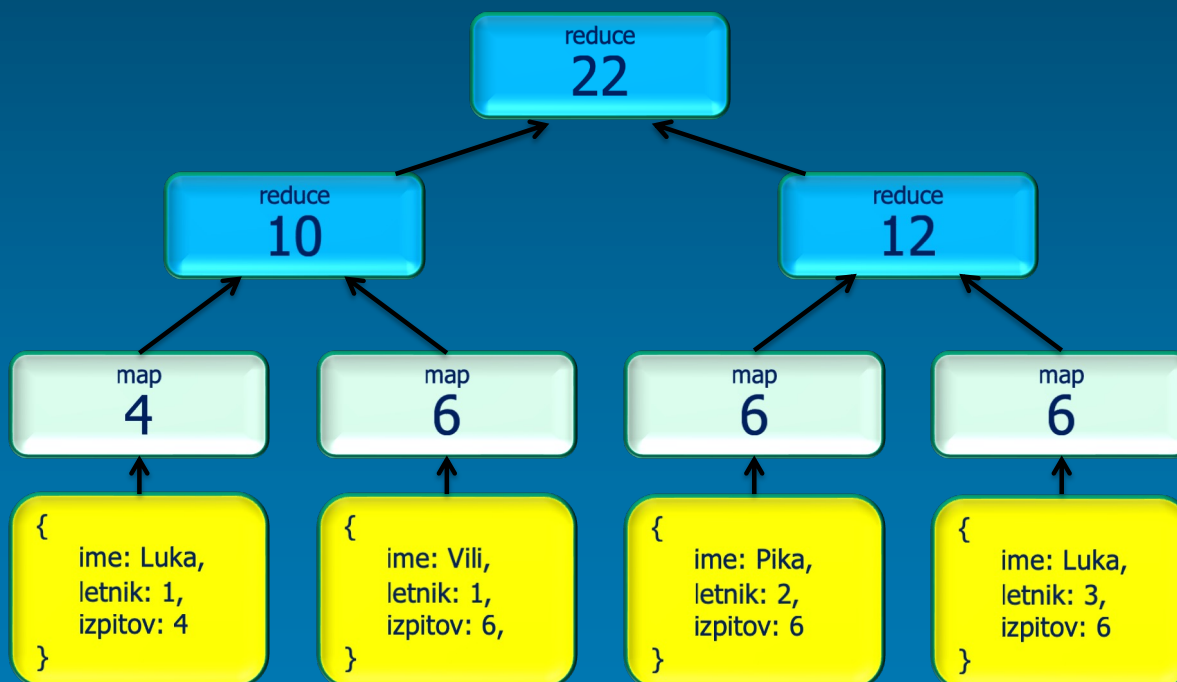
- `> db.izpiti.find({}, {_id:0})` # Projekcija, izpis brez `_id`

```
{ "ime" : "Luka", "letnik" : 1, "izpitov" : 4 }  
{ "ime" : "Vili", "letnik" : 1, "izpitov" : 6 }  
{ "ime" : "Jaka", "letnik" : 1, "izpitov" : 2 }  
{ "ime" : "Taja", "letnik" : 2, "izpitov" : 2 }  
{ "ime" : "Dino", "letnik" : 2, "izpitov" : 4 }  
{ "ime" : "Igor", "letnik" : 3, "izpitov" : 2 }  
{ "ime" : "Nina", "letnik" : 3, "izpitov" : 3 }  
{ "ime" : "Dani", "letnik" : 3, "izpitov" : 6 }  
{ "ime" : "Pika", "letnik" : 3, "izpitov" : 5 }  
{ "ime" : "Dasa", "letnik" : 3, "izpitov" : 5 }
```

- Zanima nas:
 1. Kakšno je skupno število izpitov, ki so jih opravili vsi študenti v letnikih?
 2. Kakšno je povprečno število izpitov, ki so jih opravili vsi študenti v letnikih?

MapReduce: primer 1

- Kakšno je skupno število izpitov, ki so jih opravili vsi študenti v letnikih?



- reduce mora biti **asociativen**, vrstni red izvedbe je neopredeljen!!!
- reduce mora vrniti podatek istega tipa kot je tip rezultata funkcije emit

MapReduce: primer 1

```
var mapper = function() {
  emit(null, this.izpitov)
}

var reducer = function(key, values) {
  var sum = 0;
  values.forEach(function(x) {
    sum += x;
  })
  return sum;
}

> db.izpiti.mapReduce(mapper, reducer, {out: "rezultati"})
{
  "result" : "rezultati",
  "timeMillis" : 97,
  "counts" : {
    "input" : 10,
    "emit" : 10,
    "reduce" : 1,
    "output" : 1
  },
  "ok" : 1,
}

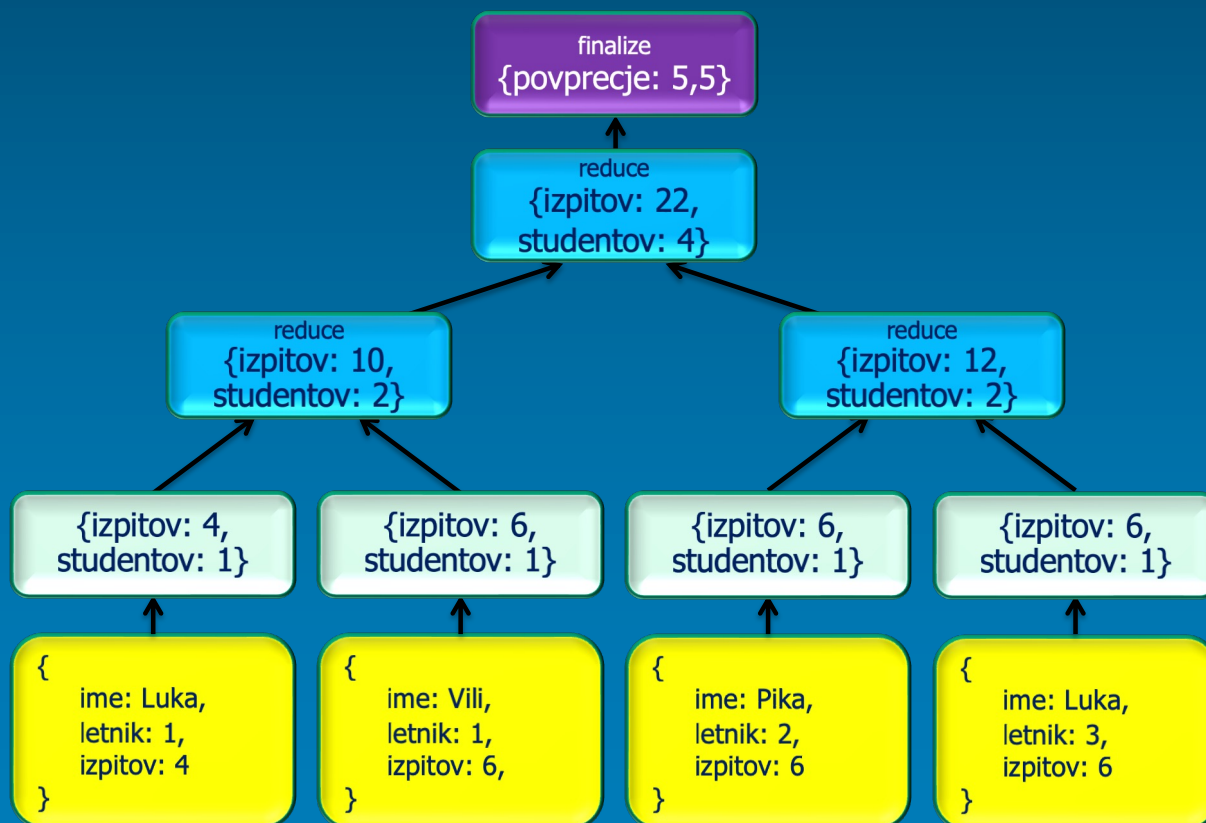
> db.rezultati.find()
{ "_id" : null, "value" : 39 }
> db.rezultati.findOne().value
39
```

naziv zbirke, v
kateri se bo
shranil rezultat

izpis rezultata

MapReduce: primer 2

- Kakšno je povprečno število izpitov, ki so jih opravili vsi študenti v letnikih?



MapReduce: primer 2

Kakšno je povprečno število izpitov, ki so jih opravili vsi študenti v letnikih?

```
var mapper = function() {  
    emit(null, {studentov: 1, izpitov: this.izpitov})  
}  
  
var reducer = function(key, values) {  
    var sum = {studentov: 0, izpitov: 0}  
    values.forEach(function(x) {  
        sum.studentov += x.studentov;  
        sum.izpitov += x.izpitov  
    })  
    return sum;  
}  
  
> db.izpiti.mapReduce(mapper, reducer, {out: "rezultati"})  
> db.rezultati.find()  
{ "_id" : null, "value" : { "studentov" : 10, "izpitov" : 39 } }  
  
> var finalizer = function(key, value) {  
    var povprecje = value.izpitov/value.studentov;  
    delete value.izpitov;  
    delete value.studentov;  
    value.povprecje = povprecje;  
    return value;  
}  
  
> db.izpiti.mapReduce(mapper, reducer,  
    {out: "rezultati", finalize: finalizer})  
> db.rezultati.find()  
{ "_id" : null, "value" : { "povprecje" : 3.9 } }
```

**funkcija za
finalizacijo
rezultata, ki se
izvede**

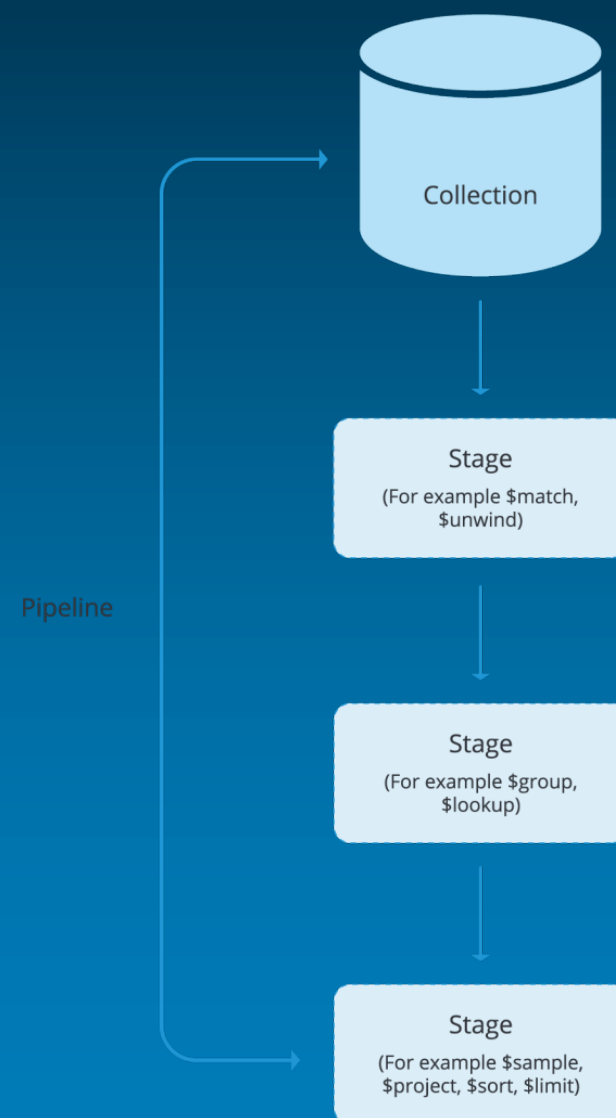
**dodatni
parameter za
uporabo
finalizatorja**

MapReduce in agregacijske funkcije

- MapReduce je v novejših verzijah označen kot zastarel (deprecated)
- Velika večina MapReduce opravil v MongoDB je agregacijskih!
- Spomnimo se:
SELECT MIN/MAX/SUM/AVG/...
...
GROUP BY ...
HAVING ...
- Agregacijske funkcije in cevovodi nadomeščajo in poenostavljajo MapReduce, pa še hitreje delujejo

Agregacijske funkcije in cevovodi

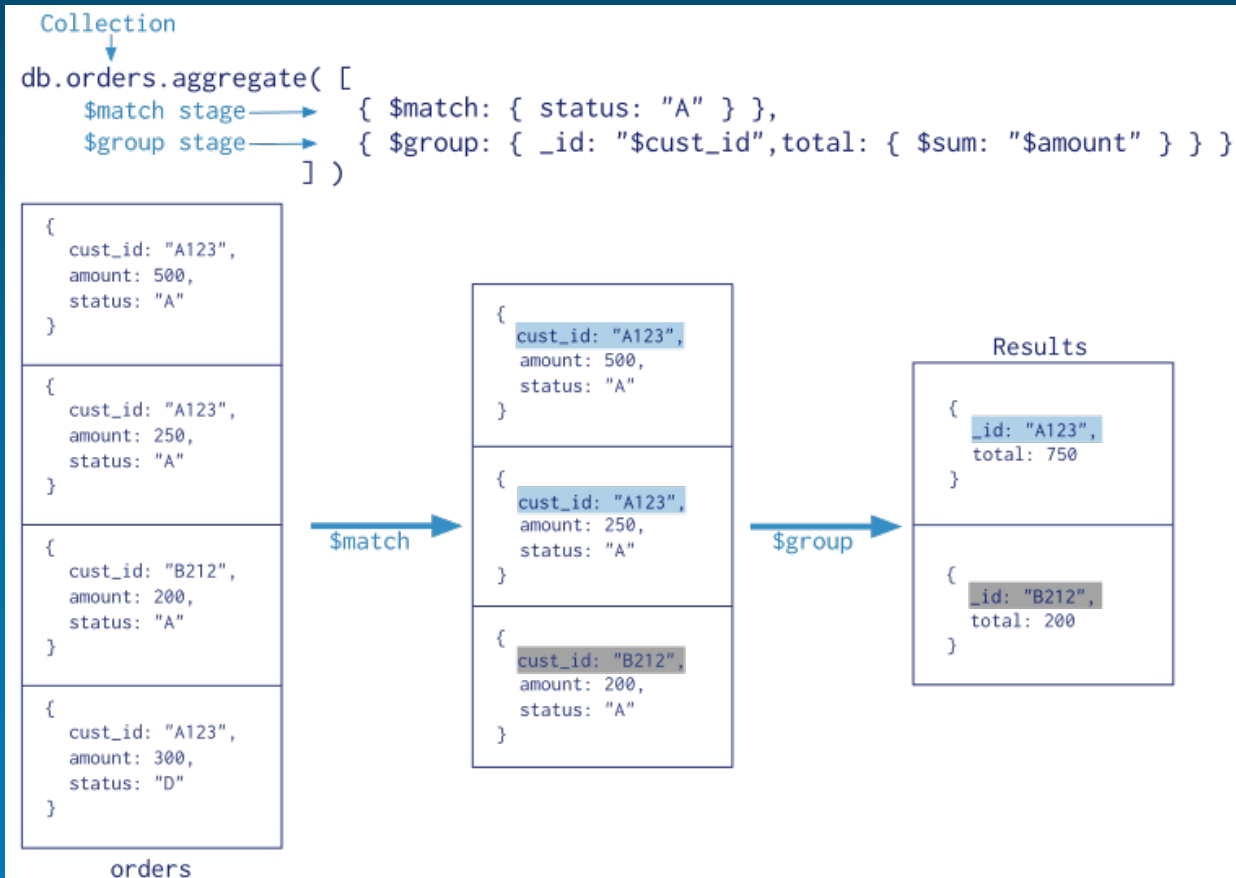
- Preproste: count, distinct, ...
- Agregacije so vsaj tako močne kot v SQL
- Agregacijo je smiselno organizirati v stopnje, ki jih potem poljubno sestavljamo v cevovode
-
- Agregacije so implementirane v C/C++ (mnogo hitrejše kot Java Script) in se avtomatsko horizontalno skalirajo



Agregacijske funkcije in cevovodi

- Primer: ločevanje operacij v \$match in \$group stopnjah

Vsota po strankah (vsaki posebej) s statusom "A"



MongoDB - zaključek

- Enostavno razumljiv podatkovni model
- Bogat nabor operacij
- Preprosto horizontalno skaliranje (ob vzpostavitvi sistema, uporabnik tega pretežno ne vidi)
- Preprosto porazdeljeno procesiranje (agregacije)
- Od verzije 4 dalje podpora ACID transakcijam (na zahtevo, znotraj namenskih sej - session), vendar z omejitvami:
 - Počasnejše procesiranje (zaklepanje virov)
 - Časovna omejitev (privzeto 60 s)
 - Omejitev obsega (maksimalno 16 MB sprememb v transakciji)