

Upute za NoSQL projekt iz predmeta Napredni modeli i baze podataka

akademska godina 2016/2017



Upute su podijeljene u dva dijela:

- 1. Upoznavanje s osnovnim MongoDB funkcionalnostima**
- 2. Zadaci koje je potrebno samostalno riješiti**

Sadržaj

Upoznavanje s osnovnim MongoDB funkcionalnostima.....	2
Pokretanje testnog sustava	2
Unos, izmjena i brisanje vrijednosti	2
Unos zapisa.....	2
Izmjena zapisa	3
Dohvat zapisa	5
Operatori usporedbe.....	6
Operacije nad kursorom	8
Map/Reduce	8
Indeksi	10
Binarne datoteke	11
Replikacija.....	11
Fragmentacija (eng. <i>sharding</i>).....	13
Zadaci koje je potrebno samostalno riješiti	15
NoSQL1: Minimalni portal temeljen na MongoDB (10 bodova)	15
NoSQL2: MapReduce upit (3 + 7 = 10 bodova)	16

Upoznavanje s osnovnim MongoDB funkcionalnostima

Pokretanje testnog sustava

Pokrenite virtualno računalo, spojite se putem terminala i pokrenite mongo:

```
mongod --journal
```

Taj terminal ostavite kakav jest (u njemu ćete kasnije vidjeti neke ispisane vrijednosti s print naredbom), te pokrenite još jedan terminal u kojem ćete obavljati ostatak vježbe.

Unos, izmjena i brisanje vrijednosti

(Otvorite drugi terminal) i pokrenite mongo *shell*:

```
mongo
```

Pokrenite naredbu koja ispisuje pomoć za rad u *shellu*:

```
help
```

Primijetite da za neke kategorije možete dobiti detaljniji opis. Iz *shella* možete izaći s CTRL+C.

Preporučuje se koristiti shell iz terminala. Ipak, ako nekome to predstavlja problem, možete koristiti i alternativne GUI alate koji podržavaju shell naredbe (kako bi mogli pratiti upute), npr. besplatni <http://robomongo.org/>. Shell je moguće pokretati i s vanjskog računala, pogledajte opasku u poglavlju binarne datoteke.

Pogledajte koje su raspoložive baze podataka:

```
show dbs
```

Baza podataka „local“ je interna tj. systemska mongo baza podataka u koju mongo sprema (meta)podatke i ne treba ju dirati. Pogledajte na koju bazu podataka ste spojeni:

```
db
```

Mongo, po defaultu, spaja korisnika na bazu „test“ (zašto se nije vidjela s show dbs?).

Prebacite se na bazu podataka „nmbp“:

```
use nmbp
db
```

Osim mongo naredbi, u mongo ljustici možete obavljati i javascript kod jer je mongo shell i javascript interpreter. Isprobajte neke js izraze, npr.:

```
var d = new Date()
d
d.getYear() <pritisnite TAB nakon Y> // dobiti ćete getYear(), ne pitati zašto vraća 115 ☹
var obj = { ime: 'Ana', voli: ['Milovana', 'Ivana']}
obj
```

Mongo shell se može pokrenuti i u tzv. „blind mode“, tj. može joj se zadati argument (izraz ili npr. js datoteka) koju ona onda obavlja; zgodno za npr. obavljanje zakazanih (*scheduled*) skriptiranih zadataka.

Unos zapisa

Zapisi su grupirani u kolekcije (collections) koje su donekle analogne tablicama u relacijskim bazama podataka. Za razliku od BP, kolekcije se ne moraju eksplicitno stvarati, već same nastaju kada se

unesu prvi zapis u kolekciju.

Što se tiče samih zapisa odnosno elemenata kolekcije, jedino pravilo koje se tiče sheme podataka jest da svaki zapis mora imati „_id“ atribut. Ako se taj atribut ne navede kod unosa zapisa, mongo će ga sam generirati.

U bazi nmbp probajte naredbu kojom se ispisuju postojeće kolekcije:

```
show collections
```

Očekivano, nema niti jedne kolekcije. Unesite jedan zapis u kolekciju student:

```
db.student.save({ime:"Ana", prezime:"Kralj"})
```

Probajte ponovo:

```
show collections
```

Osim kolekcije student, nastala je i nova kolekcija system.indexes u kojoj se pohranjuju indeksi.

Pogledajte uneseni zapis:

```
db.student.find()
```

i primijetite generirano _id polje. Svojstva id polja su detaljnije opisana ovdje

<http://docs.mongodb.org/manual/reference/object-id/>, npr. iz generiranog id polja možete doznati vrijeme nastanka zapisa što često zna biti korisno:

```
ObjectId().getTimestamp("<ovdje kopirajte svoj _id>");
```

Probajte sad unijeti još dva zapisa s eksplicitno zadanim, istim id-om:

```
db.student.save( { _id: 1, ime:"Eva", prezime:"Kralj" } )
db.student.find()
db.student.save( { _id: 1, ime:"Mirta", prezime:"Car" } )
db.student.find()
```

Što se dogodilo?

Probajte sad insert naredbu za unos podataka:

```
db.student.insert( { _id: 2, ime:"Maksim", prezime:"Beg" } )
db.student.find()
db.student.insert( { _id: 2, ime:"Maks", prezime:"Beg" } )
db.student.find()
```

U čemu je razlika?

Ima li smisla koristiti insert bez zadanog _id polja?

Izmjena zapisa

Probajmo izmijeniti postojeći zapis. Pretpostavimo da imamo kolekciju u kojoj nešto brojimo po danima u tjednu. Unesimo zapis za ponedjeljak:

```
db.counter.insert({_id: 'mon', cnt: 0})
db.counter.find()
```

Zatim želimo povećati brojač za jedan:

```
var mon = db.counter.findOne({_id: 'mon'});
mon
mon.cnt += 1;
db.counter.save(mon);
db.counter.find();
```

Koji je problem s ovakvim pristupom?

Mongo ima update naredbu koja ima svojstvo **atomarnosti** na razini dokumenta:

```
Db.collection.update(query, update, options)
```

Povećajte vrijednost brojača putem update naredbe. Koristi se mongova funkcija za inkrementiranje \$inc (popis funkcija možete pronaći ovdje <https://docs.mongodb.org/manual/reference/operator/update-field/>).

```
db.counter.update( { _id: 'mon' }, { $inc: { cnt: 1 } })
```

Zašto je ovo bolje od prethodnog pristupa?

Dodajmo dodatni atribut u naš dokument, npr. vremensku oznaku zadnje izmjene – koristi se \$set funkcija:

```
db.counter.update( { _id: 'mon' }, { $set: { dateModified: null } });  
db.counter.find();
```

koji ćemo postavljati kod svake operacije (npr. uvećanje za 1)

```
db.counter.update( { _id: 'mon' },  
                  { $inc: { cnt: 1 }, $set: { dateModified: new Date() } })
```

Atribut se može ukloniti s \$unset operatorom, odnosno preimenovati s \$rename operatorom. Isprobajte.

Pretpostavimo da smo se predomislili i da želimo pohranjivati trenutke svakog uvećanja. Pretvorit ćemo dateModified u polje i kod svakog uvećanja dodavati element u polje (mogli smo i ukloniti dateModified pomoću \$unset, druga naredba bi stvorila polje):

```
db.counter.update( { _id: 'mon' }, { $set: { dateModified: [] } });  
db.counter.update( { _id: 'mon' },  
                  { $inc: { cnt: 1 }, $push: { dateModified: new Date() } })
```

Ponovite drugu naredbu više puta, te pogledajte sadržaj.

Napravite neku drugu kolekciju i u polje dodajte duple elemente. \$push naredba dodaje elemente u polje bez provjere duplikata, ako želite imati samo jedinstvene elemente u polju odnosno ako želite imati skup a ne polje, možete umjesto \$push koristiti naredbu \$addToSet.

Isprobajte.

Isprobajte \$pull operator kojim se element uklanja iz skupa. Kako se ponaša kada postoji više istih elemenata?

Isprobajte \$pop operator kojim se uklanja element iz polja (niza), primijetite da mu se mogu zadati i negativni argumenti (npr., 1 i -1). Npr. ako hoćemo poništiti zadnje uvećanje:

```
db.counter.update( { _id: 'mon' },  
                  { $inc: { cnt: -1 }, $pop: { dateModified : 1 } } )
```

Unesimo još nekoliko zapisa:

```
db.counter.insert({ _id: 'tue', cnt: 1 })  
db.counter.insert({ _id: 'wed', cnt: 2 })  
db.counter.insert({ _id: 'thu', cnt: 3 })  
db.counter.insert({ _id: 'fri', cnt: 4 })  
db.counter.insert({ _id: 'sat', cnt: 5 })  
db.counter.insert({ _id: 'sun', cnt: 6 })  
db.counter.find()
```

i probajmo ih sve postaviti na početak (reset). Prazan uvjet dohvata {} će odabrati **sve** dokumente:

```
db.counter.update( { },
                  { $set: {cnt: 0, dateModified: []}})
```

Što se dogodilo? Kada je dohvatio više dokumenata, mongo po defaultu promijeni samo prvoga. To se može izmijeniti zastavicom multi:

```
db.counter.update( { },
                  { $set: {cnt: 0, dateModified: []}},
                  {multi: true})
```

Ako se hoće promijeniti točno jedan dokument, što je čest slučaj, onda je bolje koristiti naredbu findAndModify koju je upravo za to namijenjena.

Pogledajte dokumentaciju i isprobajte <https://docs.mongodb.org/manual/reference/command/findAndModify/>.

Dohvat zapisa

Dohvat se obavlja pomoću find funkcije koja je oblika:

```
db.collection.find(query, projection)
```

pri čemu su oba argumenta opcionalna. Prvim argumentom se zadaje objekt kojim se odabiru zapisi iz baze, a drugim argumentom je moguće definirati projekciju kako se ne bi vraćali cijeli dokumenti. Funkcija vraća listu zapisa (cursor) po kojoj je zatim moguće iterirati. Mongo shell automatski iterira po cursoru i ispisuje prvih 20 elemenata, a iteriranje se može nastaviti naredbom it.

Prije primjera za dohvat ćemo učitati jednu nešto veću kolekciju test dokumenata u mongo. U direktoriju usr/mongo se nalazi datoteka AllCards.json koja je preuzeta s adrese <http://mtgjson.com/>. Format jedne igrace karte je opisan na <http://mtgjson.com/#exampleCard>.

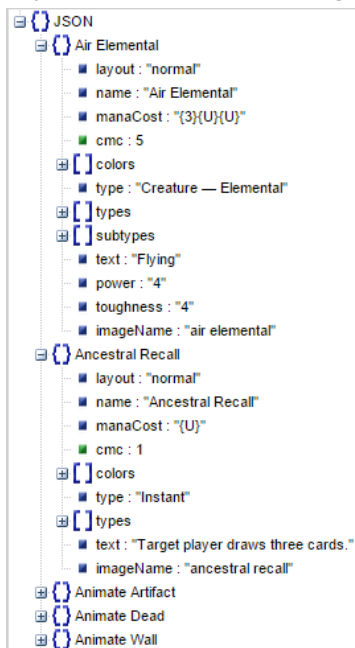
Izađite iz shella i iz spomenutog direktorija pomoću mongoimport naredbe učitajte AllCards.json:

```
mongoimport --db nmbp --collection tmp --file AllCards.json
```

Ovo će učitati sve karte u jedan dokument (u kolekciju tmp). Uvjerite se, pokrenite shell i naredbe:

```
use nmbp
db.tmp.count()
```

To se dogodilo zato jer mongoimport očekuje posebno priređen format ulazne datoteke kojemu ova datoteka ne odgovara, naime njen format je prikazan na slici.



Umjesto da mijenjamo datoteku prije importa, možemo jednostavno iskoristiti mongo engine da razdijelimo taj dokument na N dokumenata, odnosno na N karata. Sljedeća skripta iterira po svim svojstvima našeg (jedinog) dokumenta, postavlja vlastiti identifikator i potom sprema taj dokument u novu kolekciju cards.

```
var all = db.tmp.findOne();
for (var key in all) {
  var obj = all[key];
  obj["_id"] = key;
  db.cards.insert(obj);
}
```

Pogledajte koliko ima elemenata novonastala kolekcija:

```
db.cards.count()
```

Možete obrisati privremenu kolekciju s jednim dokumentom:

```
db.tmp.drop();
```

Dohvat svih elemenata:

```
db.cards.find();
it
```

Upotrijebimo projekciju kako bi pogledali samo name i type polja:

```
db.cards.find({}, {name: 1, type: 1});
```

Očigledno, mongo vraća i _id polje osim ako ga se eksplicitno ne isključi:

```
db.cards.find({}, {_id:0, name: 1, type: 1});
```

Id polje je specijalno polje i jedino kod njega je moguće miješati „include“ i „exclude“ pristup kod projekcije, npr. u gornjem upitu probamo „isključiti“ još i type:

```
db.cards.find({}, {_id:0, name: 1, type: 0});
```

dobiti ćemo pogrešku *BadValue Projection cannot have a mix of inclusion and exclusion*. Moguće je (ako izuzmemo _id) ili specificirati attribute koje treba uključiti ili attribute koje ne treba uključiti u rezultat, npr. ako „isključimo“ još i name, upit postaje ispravan:

Operatori usporedbe

Za definiranja uvjeta dohvata koristite se operatori usporedbe opisani na stranici:

<http://docs.mongodb.org/manual/reference/operator/query-comparison/>

Na primjer, dohvatimo sve kartice koje imaju power veći od 4.

```
db.cards.find({power: { $gt: 4 } });
```

Upit ne vraća niti jedan dokument – zašto? Uvjerimo se da polje power postoji:

```
db.cards.find({power: { $exists: true } }).count();
```

U čemu je problem?

Polje *power* nije cjelobrojnog tipa, pa usporedba ne radi dobro.

Pretvorimo *power* u cjelobrojni tip:

```
db.cards.find({power: {$exists: true}}).forEach(function(obj) {  
  obj.power = parseInt(obj.power);  
  db.cards.save(obj);  
});
```

Probajte dohvatiti 99 s funkcijom za „lijepi ispis“:

```
db.cards.find({power: {$eq: 99}}).pretty()
```

Isprobajmo sada za cjelobrojno polje *cmc* npr. upit koji vraća broj dokumenata koji imaju *cmc* > 10:

```
db.cards.find({cmc: {$gt: 10}}).count();
```

Ili, suprotno (primjer upotrebe negacije):

```
db.cards.find({cmc: {$not: {$gt: 10}}}).count();
```

Dohvatiti karte koje imaju *cmc* jedan ili deset – upotrijebit ćemo *\$in* operator (probajte i *\$nin*):

```
db.cards.find({cmc: {$in: [1, 10]}}).count();
```

Jednakost i *\$in*/*\$nin* operator rade i nad poljem. Isprobajte (može i s *\$eq*):

```
db.cards.find({subtypes: "Human"}).pretty()
```

Dohvatite zapise koji imaju ujedno i *Human* i *Knight* subtypes:

```
db.cards.find({subtypes:{$all: ["Human", "Knight"]}}).count()
```

Postavimo sada svim zapisima koji imaju subtype „Human“ dodatni ugniježđeni objekt „abilities“ s nekoliko svojstava:

```
db.cards.update(  
  { subtypes: "Human" },  
  { $set: {  
    abilites: {  
      canFly : "no",  
      canWalk: "yes",  
      canTalk : "yes"  
    }  
  }  
}, {multi: true}  
);
```

Atributi ugniježđenih objekata se referenciraju dot-notacijom, npr.:

```
db.cards.find( {"abilites.canFly": "no", subtypes : "Wizard"} ).count()
```

Također primijetite da se ovdje koristi AND operator – zarez.

S ugniježđenim dokumentima treba biti oprezan – koliko rezultata vraća naizgled isti upit:

```
db.cards.find( {"abilites" : {canFly: "no"}, subtypes : "Wizard"} ).count()
```

a koliko sljedeća dva:

```
db.cards.find( {"abilites" : {canFly: "no", canWalk: "yes", canTalk : "yes"},  
subtypes : "Wizard"} ).count();  
  
db.cards.find( {"abilites" : {canTalk: "no", canWalk: "yes", canFly: "yes"},  
subtypes : "Wizard"} ).count();
```

Zašto (sjetite se da Mongo pohranjuje podatke u BSON formatu)?

Operacije nad kursorom

Mongo omogućuje niz operacija nad kursorom <http://docs.mongodb.org/manual/reference/method/js-cursor/>, a ovdje ćemo komentirati samo sort, skip i limit koje se često koriste (npr. kod prikaza rezultata upita pomoću više stranica, tzv. *paging*).

Podatke možete sortirati (sortirati se može i pomoću \$orderby query modifera) na sljedeći način (silazno po cmc, uzlazno po _id polju):

```
db.cards.find({subtypes:"Wizard"}, {cmc: 1}).sort( {cmc: -1, _id: 1});
```

te zatim prikazati npr. drugu stranicu veličine 50:

```
db.cards.find({subtypes:"Wizard"}, {cmc: 1}  
).sort( {cmc: -1, _id: 1}  
).skip(50  
).limit(50);
```

Map/Reduce

Mongo ima vrlo kvalitetne mogućnosti agregacije podataka (<http://docs.mongodb.org/master/core/aggregation-pipeline/>) koja u načelu radi brže od M/R operacija i koje su limitirane jedino raspoloživom paletom operatora/izraza. Naime, neki problemi se možda ne mogu izraziti u toj okolini, dok je M/R pristup „neograničenih“ mogućnosti jer počiva na korisničkim javascript funkcijama u kojima je moguće obaviti „bilo što“. U realnim uvjetima, prilikom rješavanja konkretnog problema, aggregation pipeline bi trebao biti prvi izbor. Budući da se u okviru predmeta proučava načelno M/R algoritam (a Mongo je uzet samo kao jedna od implementacija), ovdje se neće obrađivati Mongov aggregation pipeline, već samo M/R.

Karte (ali ne sve!) ima polje subtypes, npr.:

```
"subtypes" : ["Human", "Wizard"], ...
```

Probajmo izračunati koliko ima različitih subtype-ova, odnosno koliko karata ima koje imaju subtype Human, koliko Wizard, itd.

Puni oblik map reduce naredbe možete pogledati ovdje: <http://docs.mongodb.org/master/reference/command/mapReduce/#dbcmd.mapReduce>

Definirajmo prvo map funkciju, koja će za svaku kartu iterirati po polju subtypes (ako postoji subtypes):

```
var map = function() {  
  if (this.subtypes !== undefined)  
    this.subtypes.forEach( function(subtype) {  
      emit( subtype, 1 );  
    }  
  );  
};
```



```
};
```

Dakle, ovo će za primjer polja gore, emitirati dva zapisa:

```
Human, 1  
Wizard, 1
```

U reduce funkciju pristižu svi zapisi grupirani po ključu (to je kod nas subtype – Human, Wizard, ...), te ih je potrebno jednostavno prebrojiti:

```
var reduce = function(key, values) {  
  var rv = {  
    subtype: key,  
    count:0  
  };  
  values.forEach( function(value) {  
    rv.count += value;  
  });  
  return rv;  
};
```

Pokrenimo sad M/R, za zadane dvije funkcije i spremimo rezultat u kolekciju mr_cards (ako kolekcija ne postoji Mongo će ju napraviti):

```
db.cards.mapReduce(  
  map,  
  reduce,  
  { out: "mr_cards" }  
)
```

Te pogledajmo rezultat:

```
db.mr_cards.find()
```

Rezultat nije dobar, zašto, što se dogodilo? Doima se da je umjesto zbrajanja, u nekim slučajevima došlo do konkatencije?

Probajmo doznati o čemu se radi – smanjiti ćemo ulazni skup, tako da je lakše raditi, dodajemo upit kojim definiramo ulazne podatke za M/R – samo one koji imaju (jedan od) subtypes Antelope:

```
db.cards.mapReduce(  
  map,  
  reduce,  
  { out: "mr_cards",  
    query: { subtypes: "Antelope" } }  
);  
db.mr_cards.find();
```

Sad su rezultati dobri!? Doima se da se greška ne događa na manjem skupu?

Dodati ćemo funkcije za ispis u reduce funkciju da vidimo što se zbiva, ali ćemo ispisivati samo za jedan subtype „Whale“, kako bi ispis bio pregledan:

```
var reduce = function(key, values) {  
  var rv = {  
    subtype: key,  
    count:0  
  };  
  values.forEach( function(value) {  
    if (key === 'Whale') print (key + " counting = " + rv.count + " + " + tojson(value));  
    rv.count += value;  
  });  
  if (key === 'Whale') print ("reduce for " + key + " returning " + rv.count);  
  return rv;  
};
```

```
};
```

Ponovo pokrenite M/R za sve zapise i gledajte ispis (vidi se u terminalu u kojem je pokrenut mongo). Sada se vidi da za Whale jedno vrijeme radi dobro, a onda umjesto broja 1 u polju se pojavljuje objekt? Zaključite o čemu se radi, pogledajte u predavanjima temu *combinable reducer* i pogledajte dokumentaciju monga za reduce funkciju: <http://docs.mongodb.org/master/reference/command/mapReduce/#mapreduce-reduce-cmd>

Prepravimo map i reduce funkcije:

```
var map = function() {
  if (this.subtypes !== undefined)
    this.subtypes.forEach( function(subtype) {
      emit( subtype, {count : 1} );
    }
  );
};

var reduce = function(key, values) {
  var rv = {
    subtype: key,
    count:0
  };
  values.forEach( function(value) {
    rv.count += value.count;
  });
  return rv;
};
```

Te ponovo pokrenite M/R. Koliko vremena je trebalo za izračun?

Primijetite da mongo omogućuje i finalize funkciju kojom je moguće dodatno obraditi rezultate reduce faze, npr. promijenimo malo format rezultata:

```
var fin = function (key, reducedVal) {
  return {count : reducedVal.count};
};

db.cards.mapReduce(
  map,
  reduce,
  { out: "mr_cards",
    finalize : fin
  }
)
```

Ovime su pokazane osnovne M/R funkcionalnosti, kao i osnovni pristup pri uklanjanju pogrešaka.

Više o svemu možete naći na službenoj web stranici: <http://docs.mongodb.org/master/tutorial/map-reduce-examples/>

Uočite da mongo ima i inkrementalni M/R (neće se ispitivati).

Indeksi

Pročitajte uvodno objašnjenje o indeksima na stranici: <http://docs.mongodb.org/master/core/indexes-introduction/>

Odgovorite na sljedeća pitanja:

- Zašto nastankom prve kolekcije u bazi odmah nastaje i system.indexes kolekcija? Pogledajte njen sadržaj.
- Koje vrste indeksa mongo podržava?
- Zašto hashed index ne može odgovoriti na range queries? Koja svojstva može posjedovati indeks?

Pogledajmo i usporedimo planove obavljanja upita za dva različita atributa naše kolekcije:

```
db.cards.find({_id: "Black Knight"}).explain();
db.cards.find({cmc: 3}).explain();
```

Napravimo sad i indeks na cmc atributu, pa ponovo pogledajmo system.indexes i plan obavljanja:

```
db.cards.createIndex({cmc : 1})
db.system.indexes.find()
db.cards.find({cmc: 3}).explain();
```

Za vježbu još napravite i *multikey index* i *text index* na odgovarajućim poljima.

Binarne datoteke

Binarne datoteke (npr. slike) se također mogu pohraniti u mongo (koristeći BSON BinData tip podataka), ali uz ograničenje da ne smiju biti veće od 16MB.

Ako su vaše datoteke potencijalno veće od 16MB, moguće je koristiti GridFs:

<http://docs.mongodb.org/manual/faq/developers/#faq-developers-when-to-use-gridfs>

Naravno, možete ga koristiti i kad nisu veće od toga.

Datoteku s diska možete učitati u GridFs koristeći mongofiles utility, npr.:

```
mongofiles --host 192.168.56.12 --db nmbp put -l "D:\put_do_datoteke\slika.jpg" slika.jpg
```

<http://docs.mongodb.org/manual/reference/program/mongofiles/>

Primijetite da možete na vašem vanjskom/fizičkom računalu (neka je to Windows OS), skinuti MongoDB za Windowse i onda s vanjskog (host) računala pokrenuti mongofiles (ili mongo shell) programe, bez ikakve instalacije (samo raspakirate .zip).

Pritom treba privremeno dodati bin direktorij u PATH ili pokretati baš iz njega.

Napomena: Konfiguracija replikacije i fragmentacije prikazana u nastavku je prikladna samo za potrebe testiranja i razumijevanja koncepata, nikako za produkcijske servere!

Replikacija

Pročitajte stranicu s osnovama replikacije: <http://docs.mongodb.org/master/core/replication-introduction/>

Isprobati ćemo napraviti replica set od tri čvora, operacije čitanja i pisanja, te uzrokovati izbore novog primary čvora.

Napravite tri nova direktorija u kojima će tri instance mongod čuvati svoje podatke:

```
mkdir -p /usr/mongo/rs0-0 /usr/mongo/rs0-1 /usr/mongo/rs0-2
```

pokrenite tri nova terminala (onu standalone instancu iz prethodnih vježbi čak možete ostaviti da radi, side-by-side, s našim novim replica setom), prijavite se i u svakom pokrenite odgovarajuću naredbu:

```
mongod --port 27018 --dbpath /usr/mongo/rs0-0 --replSet rs0 --smallfiles --oplogSize 128
mongod --port 27019 --dbpath /usr/mongo/rs0-1 --replSet rs0 --smallfiles --oplogSize 128
```

```
mongod --port 27020 --dbpath /usr/mongo/rs0-2 --replSet rs0 --smallfiles --oplogSize 128
```

Sada su pokrenute tri instance, svaka s vlastitim direktorijem za pohranu podataka, ali nisu još umreženi odnosno nije još inicijaliziran replica set.

Pokrenite četvrti ☺ terminal, te se spojite iz shella na jedan od novih čvorova:

```
mongo --port 27018
```

te konfigurirajte replica set:

```
rsconf = {
  _id: "rs0",
  members: [
    {
      _id: 0,
      host: "192.168.56.12:27018"
    }
  ]
}
rs.initiate( rsconf )
rs.conf()
```

Konačno, dodajte preostale dvije instance u replica set. Dodavanjem se izabire primary čvor, primijetite kako vam se mijenja prompt u shellu:

```
rs.add("192.168.56.12:27019")
rs.add("192.168.56.12:27020")
```

Probajte:

```
rs.conf();
rs.status();
```

Spojite se na PRIMARY (ako već niste) i snimite jedan zapis u defaultnoj test bazi podataka:

```
db.test.save({opis: "prvi"});
```

Izađite iz shella i spojite se na jedan od SECONDARY čvorova i probajte:

```
mongo --port 27019
db.test.save({opis: "drugi"})
show collections
show dbs
```

Ništa ne radi. Postavite da je za tekuću konekciju dozvoljeno čitati s SECONDARY čvora:

```
rs.slaveOk()
```

i probajte ponovo sve te naredbe (kako biste to napravili iz npr. web aplikacije?).

Izađite iz shella i spojite se na PRIMARY. Naredite mu da odstupi:

```
rs.stepDown()
```

i gledajte što se događa u terminalima ostala dva SECONDARY čvora – oni organiziraju izbore! Prethodni PRIMARY je i dalje aktivan u replica setu, samo je sada SECONDARY.

Doznajte koji je novi PRIMARY i spojite se na njega. Ugasite ga:

```
use admin
db.shutdownServer()
```

Spojite se na neki od preostalih čvorova i pogledajte stanje:

```
rs.status();
```

Na isti način ugase preostali PRIMARY i pogledajte što se dogodilo.

Fragmentacija (eng. *sharding*)

Pročitajte uvodni tekst na: <http://docs.mongodb.org/master/core/sharding-introduction/>

Nastavak je opcionalan, ne morate ga znati reproducirati u okviru predmeta. Slijedi niz naredbi kojima će se uspostaviti testni *sharding cluster* s :

- jednim *config* serverom,
- jednim *routing* serverom i
- dva *shard* servera.

Ugasite sve servere iz prethodnih vježbi. Otvorite novi, prvi terminal.

Napravimo direktorij za *config* server

```
mkdir -p /usr/mongo/configdb
```

i pokrenimo ga (na portu 27019):

```
mongod --configsvr --dbpath /usr/mongo/configdb --port 27019
```

Otvorite drugi terminal te pokrenite *router (mongos)*. Kao argument predajmo adresu *config* servera:

```
mongos --configdb 127.0.0.1:27019
```

Mongos ne treba podatkovni direktorij i sluša na defaultnom portu 27017.

Otvorite nova dva terminala. Napravimo nove podatkovne direktorije za dva *shard* servera:

```
mkdir -p /usr/mongo/sh1 /usr/mongo/sh2
```

i pokrenimo ih:

```
mongod --port 27020 --dbpath /usr/mongo/sh1 --smallfiles --oplogSize 128
mongod --port 27021 --dbpath /usr/mongo/sh2 --smallfiles --oplogSize 128
```

Pokrenite peti terminal ☺, te se spojite sa *shellom* na *routing server*:

```
mongo --host 127.0.0.1 --port 27017
```

Konačno, dodajmo naša dva *sharding servera*:

```
sh.addShard( "127.0.0.1:27020")
sh.addShard( "127.0.0.1:27021")
```

provjerimo status:

```
sh.status();
```

Trenutno se još uvijek ništa ne fragmentira. Omogućimo fragmentaciju na bazi *shtest*:

```
use shtest;
sh.enableSharding("shtest");
```

te omogućimo i definirajmo fragmentaciju na kolekciji *cards* (zasad nepostojećoj) na temelju atributa *_id*:

```
sh.shardCollection("shtest.cards", { "_id": "hashed" } )
```

Ponovite postupak kojim se karte iz datoteke AllCards.json učitavaju u bazu, te pogledajte kako su distribuirani podatci:

```
db.cards.getShardDistribution()  
sh.status();
```

Zadaci koje je potrebno samostalno riješiti

Potrebno je napraviti dva zadatka opisana u nastavku.

NoSQL1: Minimalni portal temeljen na MongoDB (10 bodova)

Napraviti web portal koji ispisuje najnovijih N (npr. N=10) vijesti.



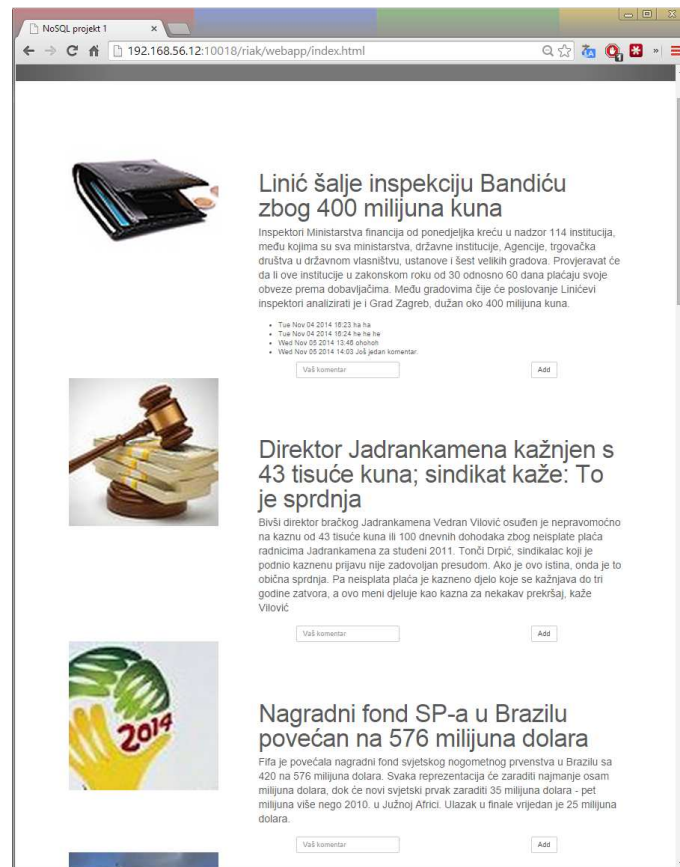
Osmisliti kako, odnosno upotrijebiti prikladnu strukturu podataka za to.
Na primjer, ako pretpostavimo da u bazi imamo 10.000 članaka, **nije dobra** strategija dohvatiti svih 10.000 zapisa na klijenta, sortirati i uzeti prvih deset.

Svaka vijest treba biti (otprilike) sljedećeg oblika:

Naslov	Slika
Tekst	
Autor	

Omogućiti komentiranje vijesti: ispod svake vijesti dodati polje za unos komentara.
Nije potrebno napraviti sučelje za unos vijesti, odnosno možete ih sve unijeti ručno.

Npr.:



NoSQL2: MapReduce upit (3 + 7 = 10 bodova)

Napisati:

(a) (3 boda) MapReduce upit koji vraća listu članaka poredanu silazno po broju komentara.

(b) (7 bodova) MapReduce upit koji za svakog autora vraća prvih 10 najkorištenijih riječi.

Pojam „riječ“ shvatiti u najjednostavnijem mogućem obliku (niz slova odvojen od drugih s razmakom, zarezom ili točkom).

Nije potrebno raditi nikakve leksičke transformacije na riječima (svođene na korijen i sl.).

Nije potrebno ispisati i 11. riječ ako se ima isti broj korištenja kao 10. riječ.

Priznavati će se (sa smanjenim bodovima) i polovična rješenja, npr. sve riječi koje je autor koristio (a ne prvih 10) i sl.

Za oba zadatka:

Možete koristiti proizvoljnu tehnologiju, postoji velik broj drivera za MongoDB:

<http://docs.mongodb.org/ecosystem/drivers/>

Zadatci će se predavati osobno, tj. potrebno je demonstrirati rješenje.

Razmatrati će se i polovična rješenja (npr. samo prvi zadatak, bez komentiranja vijesti).

Rješenja projekta trebaju sadržavati readme.txt dokument u korijenskom direktoriju u kojoj se objašnjava:

- Kako je riješen zadatak (koja tehnologija, kako su uneseni test podatci, itd.)
- Kako pokrenuti rješenje

Studente se kod predavanja može pitati da:

- objasne nešto iz rješenja (main.txt)
- objasne neki koncept iz Uputa (npr. *replica set*)
- pokrenu tj. demonstriraju rješenje (i npr. dodaju vijest na portal)
- naprave neke manje modifikacije (npr. blago modificirati M/R upit)
- itd.