

# **SODOBNE NERELACIJSKE PODATKOVNE BAZE (NOSQL)**

**UNIVERZITETNI ŠTUDIJ RI IN UI, 3. LETNIK  
MAGISTRSKI ŠTUDIJ RI**

Matjaž Kukar, 2021/22

# Vsebina

- problemi relacijskih podatkovnih baz (teoretični in praktični)
- nerelacijske podatkovne baze, NoSQL
- MongoDB (dokumentni SUPB)
  - namestitev
  - delo z dokumenti
  - iskanje
  - posodabljanje
  - vgnezdeni dokumenti
  - kurzorji
  - indeksiranje
- Neo4j (grafni SUPB)
  - namestitev
  - delo z grafi
  - iskanje

# Uvod

*"Facebook now has more than 1.8 billion monthly active users. Its data storage is more than 300 petabytes. Every 60 seconds on Facebook:*

- *510 comments are posted*
- *293,000 statuses are updated*
- *136,000 photos are uploaded"*



*"Twitter now has more than 320 million active users, sending > 500 million tweets every day"*



"Webscale" aplikacije

- veliko podatkov
- veliko istočasnih uporabnikov
- veliko zahtevanih obdelav
- časovno pogojene obremenitve (Facebook torek in četrtek +20%)

So relacijske baze primerne za takšna bremena?

# Problemi z resursi - možne rešitve

- **vertikalno skaliranje (scaling up):** nadgradnja virov z zmogljivejšimi
  - kje je meja? cena?
- **horizontalno skaliranje (scaling out):** porazdelitev podatkov po več strežnikih
  - master-slave: pisanje se izvaja na glavnem strežniku, branja se lahko izvajajo iz sekundarnih baz (problem - konsistentnost)
  - deljenje podatkov (partitioning, sharding): razdelitev podatkov v podmnožice, boljše skaliranje, vendar izguba možnosti izvedbe stikov med podatki in referenčne integritete
- **druge možne rešitve**
  - replikacija cele baze v več "master" baz,
  - izvedba brez stikov – denormalizacija (stiki so eno od ozkih grl),
  - hranjenje majhnih baz v primarnem spominu,
  - nerelacijske rešitve?

# NoSql

Not  
Only SQL

- **relacijske baze:** *one size fits all*
- **ozka grla**
  - Stične operacije
  - Konsistentnost podatkov v PB pri sočasni uporabi - ACID
- **nerelacijske baze:** *prilagojene zbirke podatkov za specifične aplikacije*
  - večinoma žrtvujejo shemo ACID
  - shema BASE (basically available, soft state, eventually consistent)
  - prednosti: cena, zmogljivost
  - slabosti: pomanjkanje standardov, potrebna specifična priučitev, omejena prenosljivost, nezrelost tehnologije
  - ne uporabljajo fiksne podatkovne sheme in stikov!



# BASE vs ACID

Not  
Only SQL

## *Basic Availability*

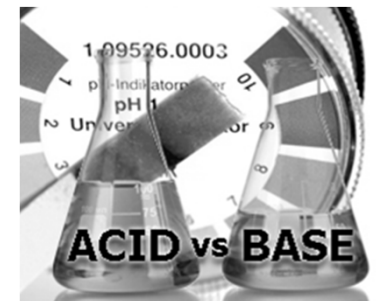
- Podatki v PB so dostopni „večino časa“

## *Soft-state*

- Konsistentnost pisanja ni zagotovljena, ravno tako ne konsistentnost med replikami podatkov (v vsakem trenutku)

## *Eventual consistency*

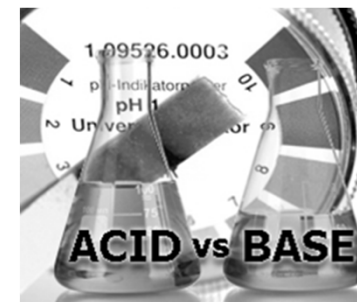
- Sčasoma se podatek „stabilizira“ v konsistentno stanje



# CAP – zaželene lastnosti PB

Not  
Only SQL

- Consistency - konsistentnost: (ACID)  
Vsako branje prebere zadnjo veljavno vrednost podatka (ali pa branje ni dovoljeno)
- Availability – razpoložljivost:  
Vsaka bralna zahteva dobi vrnjeno vrednost (v principu brez garancije, da gre za zadnjo verzijo podatka).
- Partition tolerance – zmožnost delitve podatkov:  
Sistem deluje kljub težavam (zakasnitve ali izgube podatkov) na poljubno velikem delu (<100%) povezav med vozlišči.

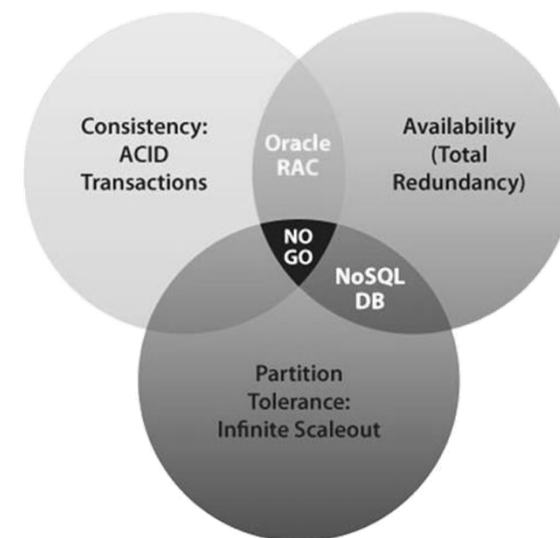


# Teorem CAP

- Eric Brewer, 2000; dokaz na MIT 2002
- Vsak sistem za delo s podatki ima tri lastnosti:
  - konsistentnost (Consistency),
  - razpoložljivost (Availability) in
  - zmožnost delitve podatkov (Partitions)
- Teorem pravi: pri deljenju podatkov lahko zagotovimo **največ dve** od teh treh lastnosti

Primer: podatke razdelimo na več računalnikov. Kasnejša posodobitev podatka zahteva posodobitev vseh kopij. Scenarija:

- zaklenemo vse kopije, da zagotovimo konsistentnost (zmanjšana razpoložljivost), ali
- žrtvujemo konsistentnost na račun večje razpoložljivosti (sčasoma podatki postanejo konsistentni)





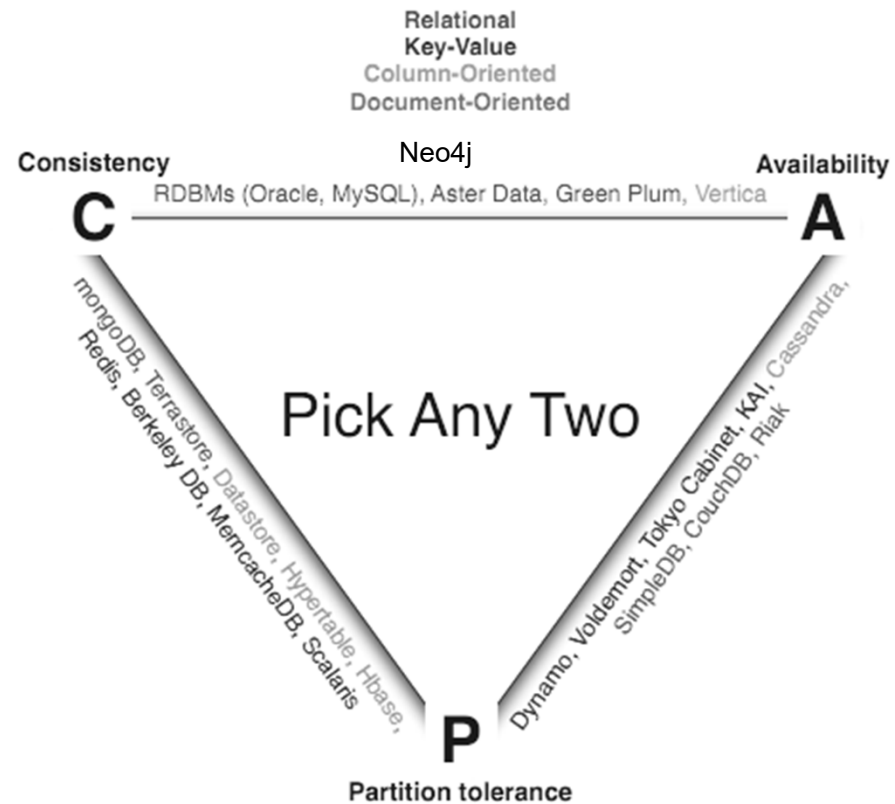
# Teorem CAP

Torej: žrtvovati je potrebno enega od kriterijev: C, A ali P

- **konsistentni in razpoložljivi (CA)** sistemi imajo težavo z deljenjem podatkov, težave običajno obvladujejo z replikacijo,
- **konsistentni, porazdeljeni sistemi (CP)** imajo težavo z razpoložljivostjo podatkov ob naporu zagotavljanja njihove konsistentnosti,
- **razpoložljivi, porazdeljeni sistemi (AP)** dosegajo sčasno konsistentnost (eventual consistency) z replikacijo podatkov in občasnim preverjanjem konsistentnosti v podatkovnih vozliščih.



# Teorem CAP

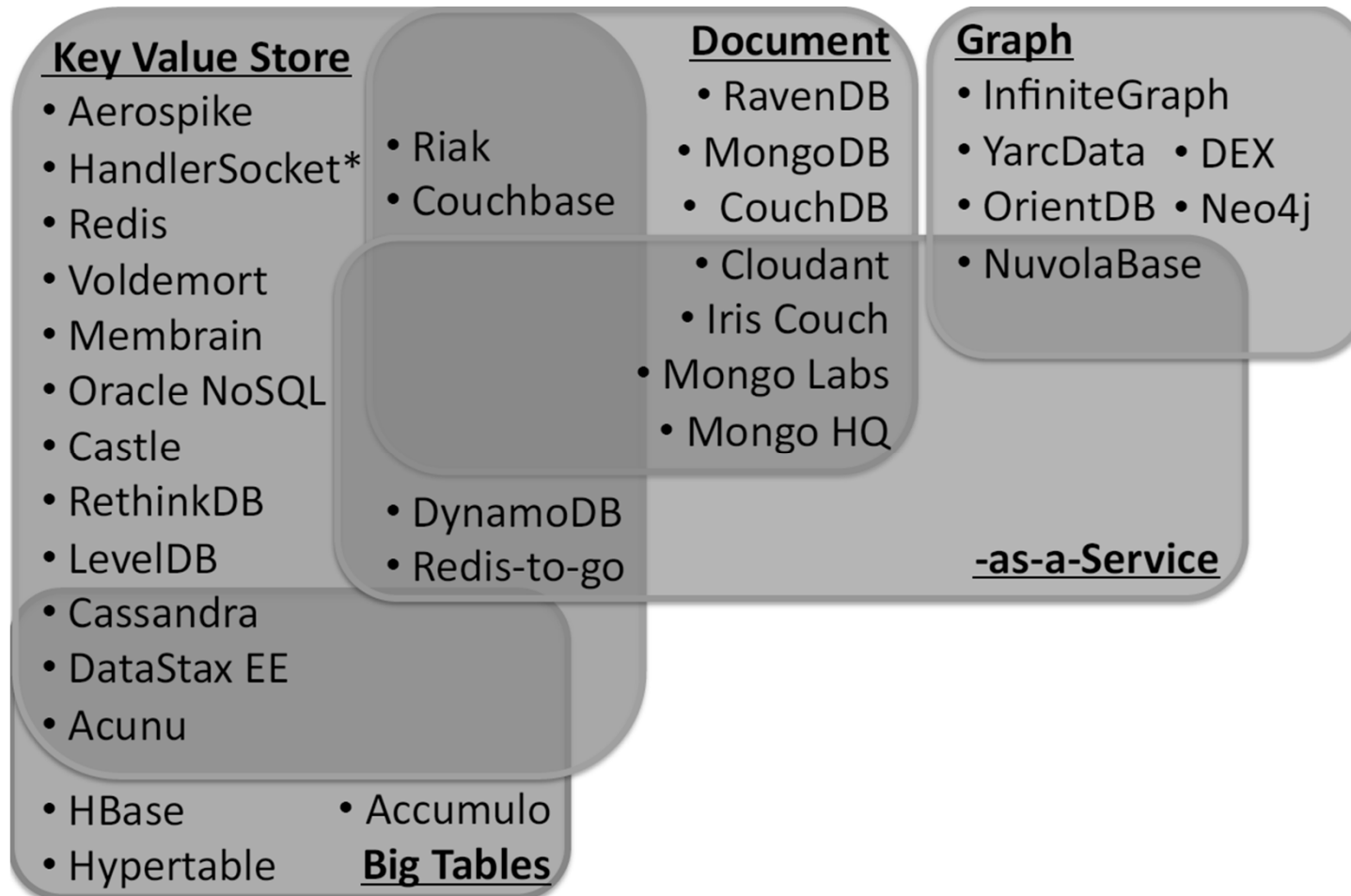


# Izvedbe NoSQL

- Prilagojene specifičnim načinom uporabe in vrstam aplikacij
  - **shrambe s ključi (key-value pairs):** BerkleyDB, Keyspace, Dymomite, Voldemort, MemcachedDB in Tokyo Cabinet.
  - **dokumentne zbirke:** MongoDB (Foursquare, SourceForge, Fotopedia, Joomla Ads), CouchDB (CERN, BBC), Redis
  - **zbirke grafov:** Neo4j, AllegroGraph, InfoGrid, Sones, graphDB in FlockDB
  - **stolpično usmerjene zbirke (column-oriented databases):** Hypertable in Cassandra (Digg, Facebook, Twitter, Rackspace), dodatki rel. bazam
  - **objektne shrambe:** Oracle Coherence, db4o, ObjectStore, GemStone, Polar



# NoSQL SUPB



# Težave NoSQL

- Ni standardnega povpraševalnega jezika
  - Sami delamo, kar sicer dela SQL prevajalnik (nizkonivojske operacije)
- Zavržemo 40 let izkušenj relacijskih SUPB
  - Težko smo boljši od prevajalnika
  - Visokonivojski jeziki so boljši po več kriterijih (neodvisnost od podatkov, manj kode)
- Ni shranjenih podprogramov
  - samo ena interakcija med aplikacijo in SUPB (namesto ena za vsak zapis, kot pri NoSQL)
  - premaknemo kodo k podatkom in ne obratno
- Če ACID danes ne potrebujemo, ali lahko zagotovimo to za jutri?
  - premiki podatkov, nekomutativne posodobitve, večzapisna stanja
  - če potrebujemo integritetne omejitve
  - eventualna konsistentnost → zmeda v podatkih

# Za kaj oz. kdaj je torej primeren NoSQL

- Netransakcijski sistemi
- Enozapisne, komutativne transakcije
- Neprimerno za sodobne OLTP sisteme
- Ne potrebujemo enovitega povpraševalnega jezika
  - programska koda
  - CQL, UnQL (po zgledu SQL, nestandardno, nekompatibilno)

# SQL + NoSQL = NewSQL?

- NoSQL:
  - Nova, moderna zvrst nerelacijskih SUPB
  - Zavračanje pojmov fiksnih shem tabel in stičnih operacij
  - Načrtovan z namenom omogočanja zahtev po masivnem horizontalnem skaliranju
  - Omogoča upravljanje podatkov brez definirane sheme
  - Bye, bye, SQL
- NewSQL
  - Zadnji trend na področju relacijskih SUPB
  - Ohranja SQL in ACID
  - Načrtovan z namenom omogočanja zahtev po masivnem horizontalnem skaliranju ali
  - Tako velik performančni napredek, da horizontalno skaliranje ni več potrebno (VoltDB)

# NewSQL SUPB

## -as-a-Service

- StormDB
- Xeround
- Tokutek

## Storage engines

• Datomic

• Akiban

• GenieDB

• ScaleDB

• MySQL Cluster

• Zimory Scale

• MemSQL

• Drizzle

• VoltDB

• JustOneDB

• ParElastic

• Continuent

• Galera

## New databases

• NuoDB

• SQLFire

• Translattice

• Clustrix

• SchoonerSQL

• ScaleBase

• ScaleArc

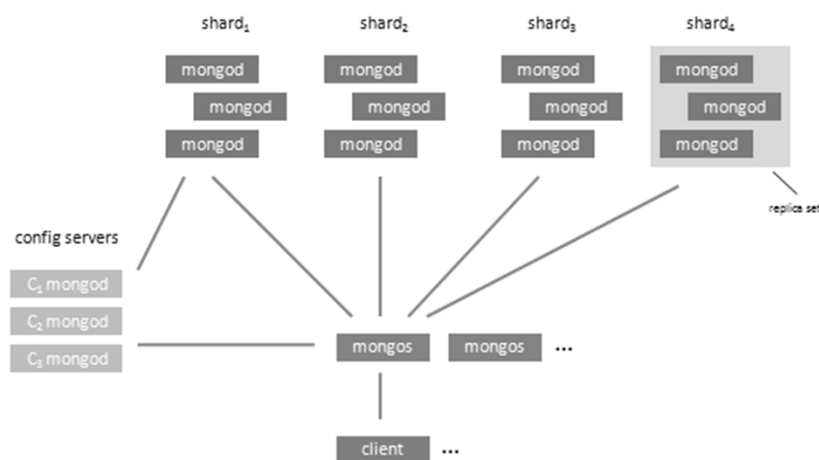
• CodeFutures

## Clustering/sharding



# MongoDB – dokumentni nerelacijski SUPB

- dokumentno-usmerjen SUPB (JSON/BSON)
- lasten komunikacijski protokol, ki uporablja TCP/IP
- konzola uporablja JavaScript sintakso
- podatkovna baza -> zbirke dokumentov (Collections) -> dokumenti (Documents)
- replikacija v obliki master-slave, delitev podatkov (sharding)
- napisan v C++
- sistem ne uporablja stikov podatkov (vendar možni v uporabniškem programu)
- podatkovni tipi: null, boolean, 32-bit integer, 64-bit integer, 64-bit floating point, string



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

# Namestitev

- <http://www.mongodb.org/>, prenos 32/64-bitne različice
- 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)
- mongod - strežniški program
- mongo - priložen odjemalec (ukazna vrstica)
- pogosta uporaba JSON in JavaScript sintakse (specifikacija dokumentov, poizvedb, skripte, ...)

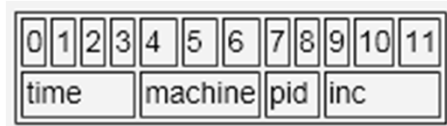
```
> mongod --config mongodb.config
Mon Nov 05 16:20:46 [initandlisten] MongoDB starting : pid=8384 port=27017 dbpath=c:\mongodb\data 64-bit host=quaddrix
Mon Nov 05 16:20:46 [initandlisten] db version v2.2.0, pdfile version 4.5
Mon Nov 05 16:20:46 [initandlisten] git version: f5e83eae9cfbec7fb7a071321928f00d1b0c5207
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: "mongodb.config", 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 vrednost
- Lahko je poljubnega tipa
- Lahko ga podamo pri kreiranju dokumenta
- Če ga ne podamo, ga MongoDB kreira avtomatsko v obliki `ObjectId`
- `ObjectId`:
  - 12-bytni BSON (binarni JSON) zapis

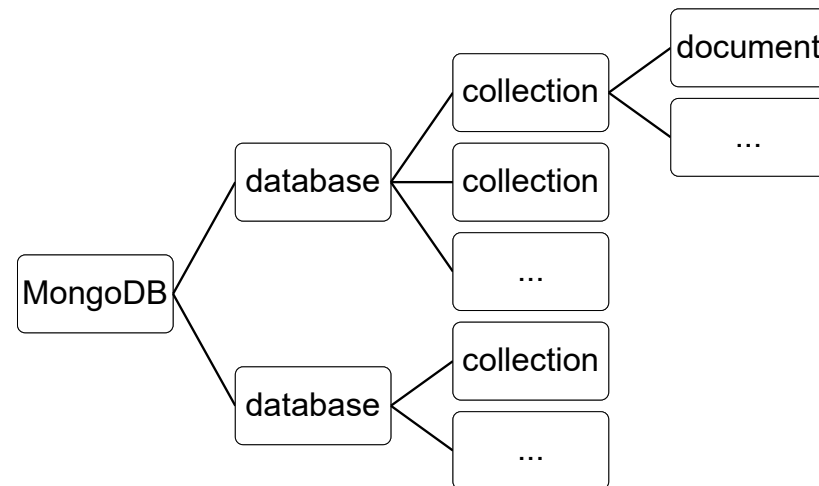


# Prvi koraki

```
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
```

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



# Delo z dokumenti

```
// največja velikost dokumenta 16 MB (2018)
db.studenti.insert({ime: "janko", priimek: "novak"})

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

db.studenti.find()

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

// zamenja vse dokumente, ki ustrezajo kriteriju
db.studenti.update(kriterij, novi_dokument)

// To je Javascript sintaksa, pymongo sintaksa je malenkost drugačna
```

# Iskanje - modifikatorji



```
db.x.find({visina: {$gte: 180}}) // višina večja od 180
// podobno $lt, $lte, $gte
db.x.find({prijatelji: {$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



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

db.x.update({ime: "Eva"}, {$inc : {starost : 5}}, true) // če ne obstaja, dodaj
db.x.update({ime: "Eva"}, {$inc : {starost : 5}}, true, true) // posodobi vse zadetke
```



# Gnezdeni dokumenti

- možno gnezdenje dokumentov in iskanje po gnezdenih poljih (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
```

```
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  
<http://docs.mongodb.org/manual>
- kratki (referenčni) povzetki:  
<http://www.10gen.com/reference>
- Peter Membrey, MongoDB Basics, Apress, 2014  
<https://www.apress.com/gp/book/9781484208960>
- 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 <a href="http://bsonspec.org">bsonspec.org</a> 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

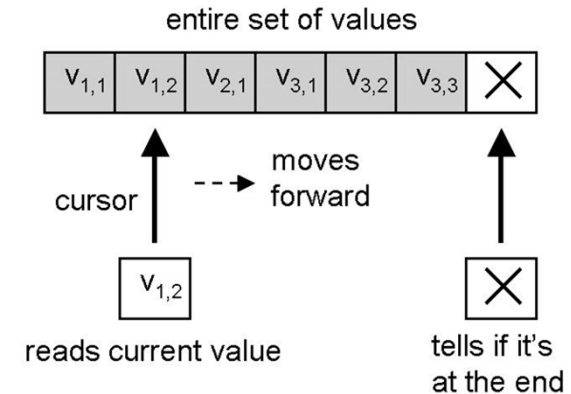
- kurzor - oblika iteratorja po podatkih v zbirki

```
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)

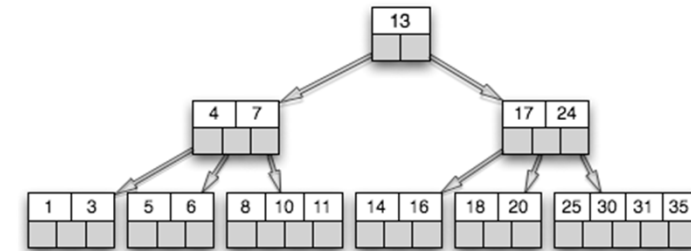
```
db.evidenca.find().limit(3) // omeji na prve 3 zapise
db.evidenca.find().skip(3) // preskoči 3 zapise
db.evidenca.find().sort({starost : 1}) // sortiranje naraščajoče
db.evidenca.find().sort({visina : -1}) // sortiranje padajoče
```

```
db.evidenca.count() // štetje zadetkov
db.evidenca.count({visina: {$gt : 170}}) // štetje s pogojem
db.evidenca.distinct("visina") // seznam različnih
```



# 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.ensureIndex({visina: 1}) // izdelava indeksa
db.x.ensureIndex({visina: 1}, {name : "indy"}) // poimenovanje indeksa
db.x.dropIndex({visina: 1}) // brisanje indeksa
db.x.ensureIndex({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; pozor: Pythonovi (pred 3.6) slovarji niso urejeni (uporabimo `bson.SON()`)
- 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.ensureIndex({"gps" : "2d"})`



# Geoprostorsko indeksiranje

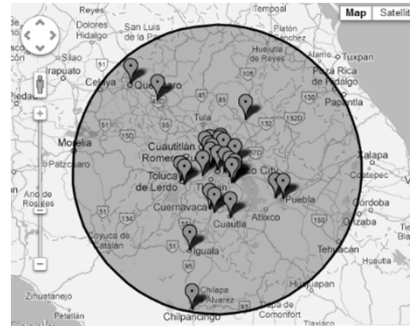
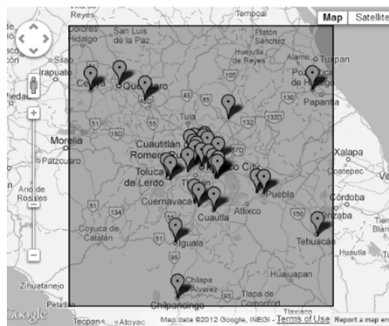
- iskanje:

```
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>
- Zdaj lahko iz Pythona dostopate tako do MariaDB, kot MongoDB (peta – opcijska - domača naloga)

# 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.test # baza test
db.collection_names() # seznam vseh zbirk

db.obvestila.insert(dokument) # dodajanje dokumenta

posts.find_one({})

for post in db.izpiti.find({}): # iteriranje po zadetkih
    print(post)

for post in db.izpiti.find({}): # varnejše iteriranje
    print(post.get("nagrada", "ni nagrade"))
```

# Iskanje, posodabljanje

```
# štetje zapisov - count
db.izpiti.find({"letnik": 1}).count()

# omejitev števila zadetkov - limit
for x in db.izpiti.find({"letnik": 1}).limit(3): x

# 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
from pymongo import GEO2D                          # sinonim za "2dsphere"
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": {"$near": [3, 6]}}).limit(3): doc
for doc in db.prostor.find({"loc": {"$within": {"$box": [[0,0],[3,4]]}}}): doc
```

# Primer (jadralci)

```
# Jadralec Darko
db.jadralec.insert( {"jid": 22, "ime" : "Darko", "rating": 10, "starost":45} )

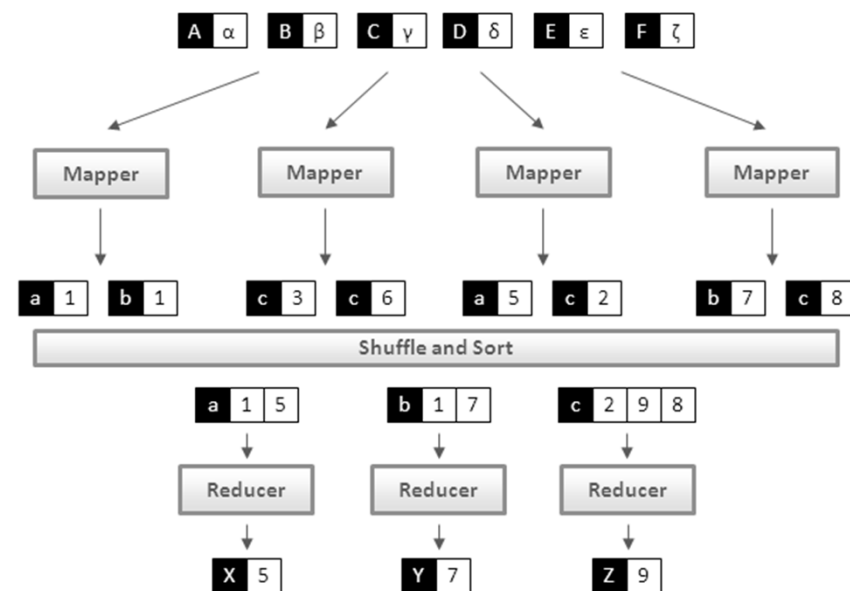
# Darkove rezervacije
db.rezervacija.insert( {"jid": 22, "cid" : 101, "dan": "2006-10-10" } )
db.rezervacija.insert( {"jid": 22, "cid" : 102, "dan": "2006-10-10" } )
db.rezervacija.insert( {"jid": 22, "cid" : 103, "dan": "2006-10-8" } )
db.rezervacija.insert( {"jid": 22, "cid" : 104, "dan": "2006-10-7" } )

# Indeksi
db.rezervacija.ensure_index("jid")
db.rezervacija.ensure_index("cid")
db.rezervacija.ensure_index( [ ("jid", pymongo.DESCENDING),
                                ("cid", pymongo.ASCENDING) ] )

# 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)
```

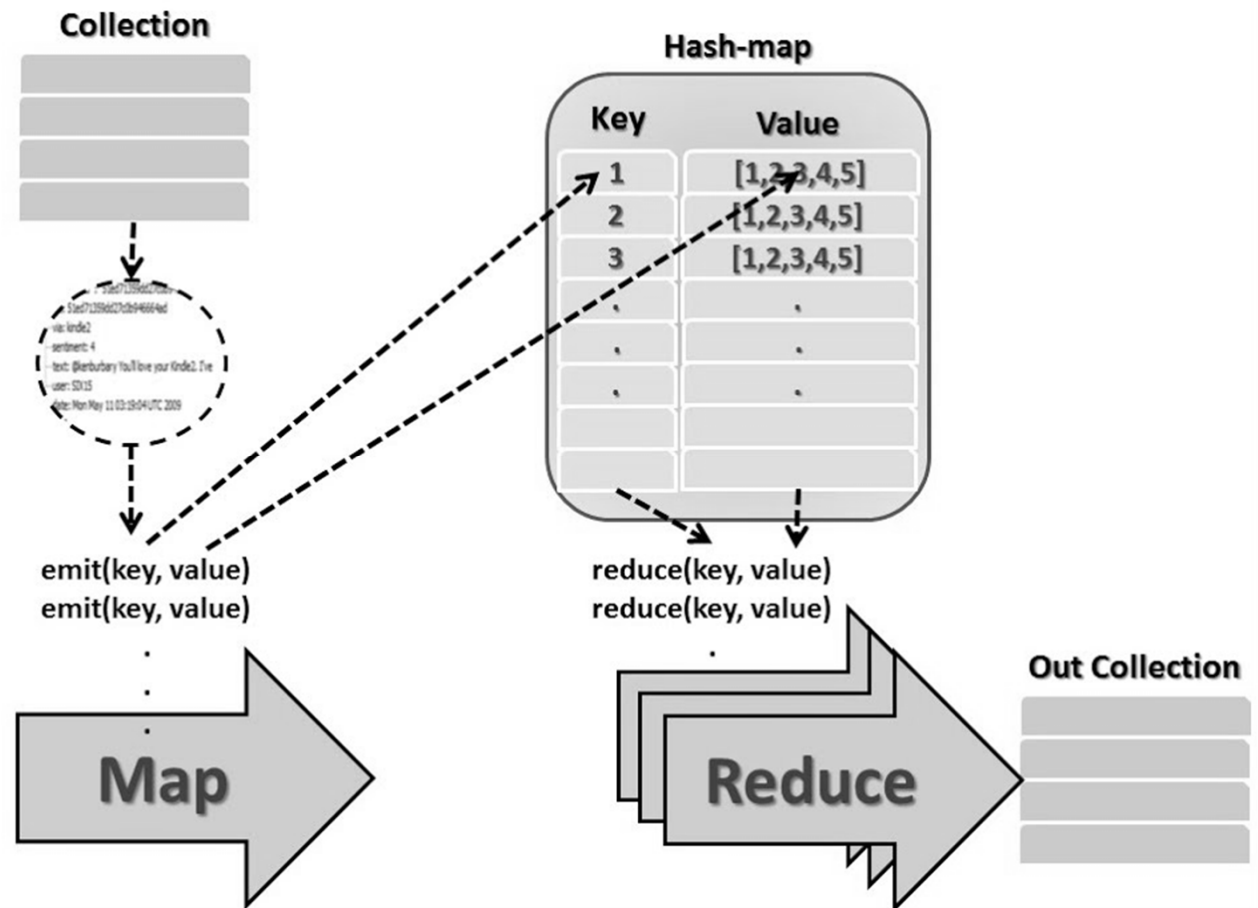
# MapReduce

- 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:

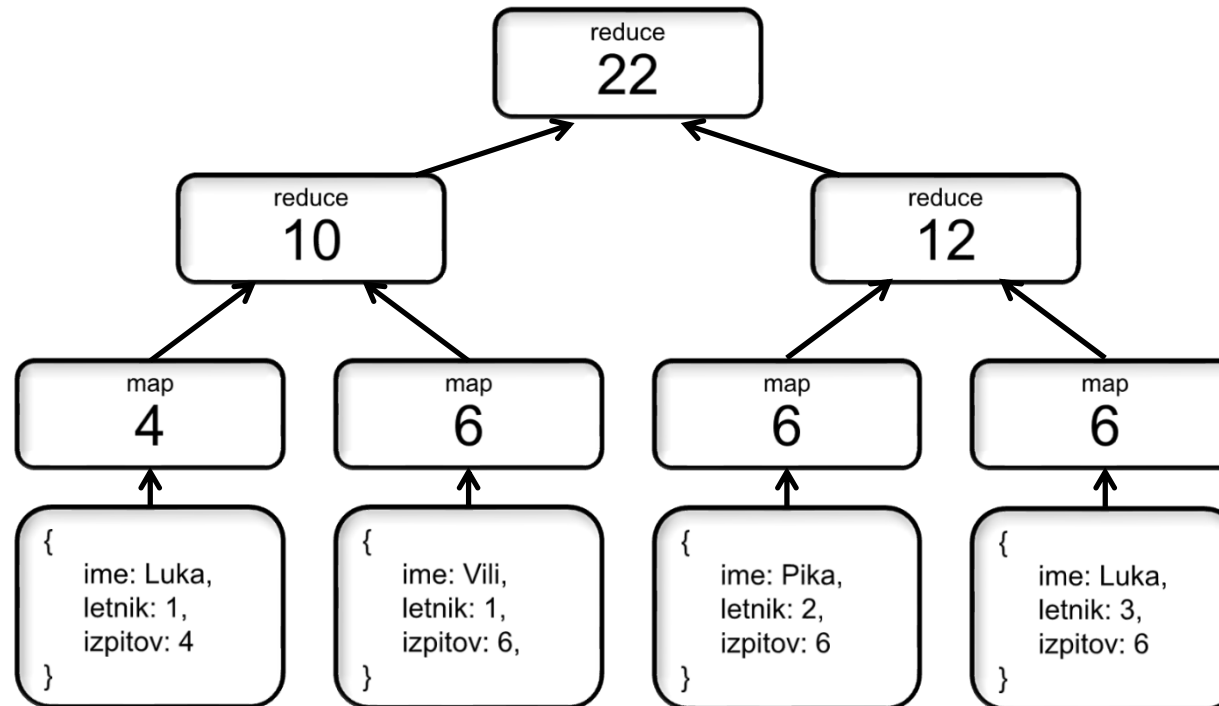
```
> db.izpiti.find({}, {_id:0}) # Projekcija, izpis brez _id
{ "ime" : "Luka", "letnik" : 1, "izpitov" : 4 }
{ "ime" : "Vili", "letnik" : 1, "izpitov" : 6 }
{ "ime" : "Jaka", "letnik" : 1, "izpitov" : 2 }
{ "ime" : "Taja", "letnik" : 2, "izpitov" : 2 }
{ "ime" : "Dino", "letnik" : 2, "izpitov" : 4 }
{ "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:

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

# MapReduce: primer 1

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,  
}
```

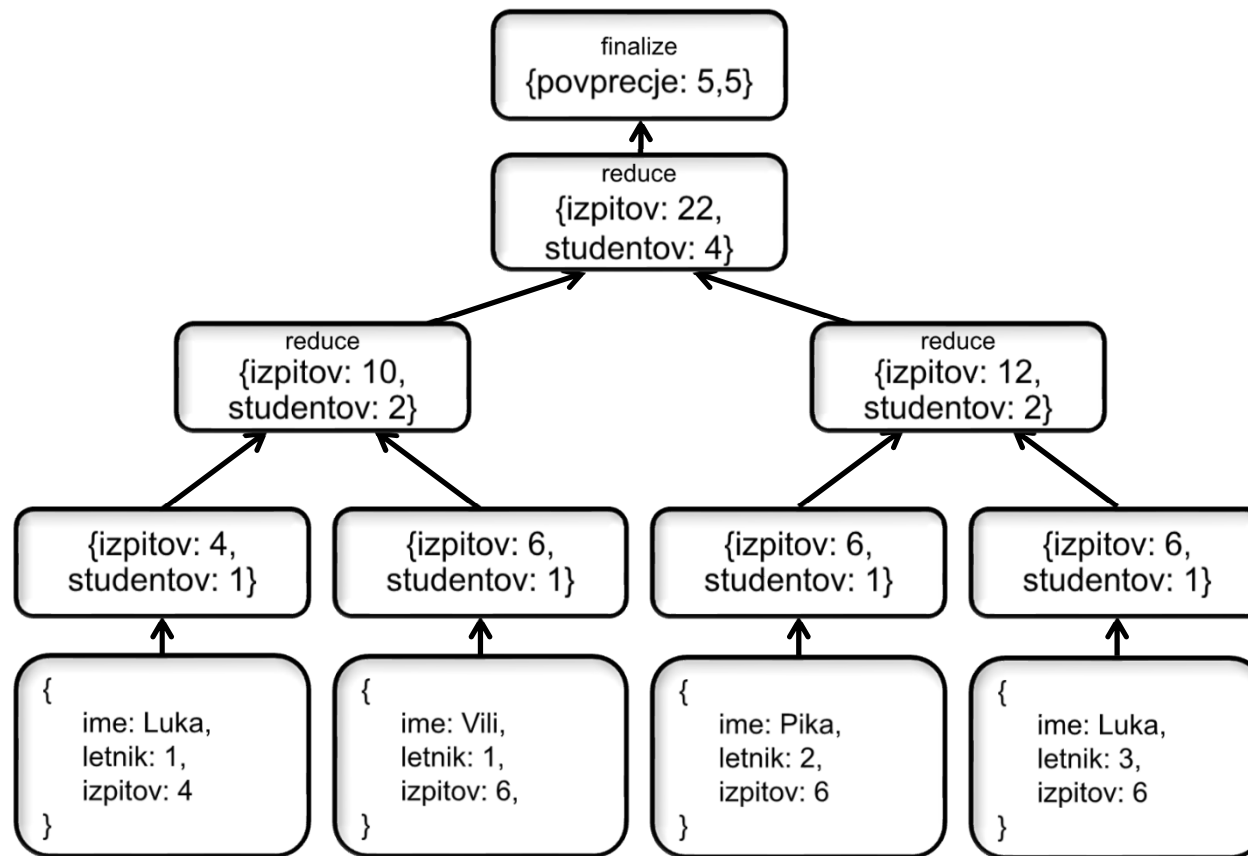
```
> db.rezultati.find()  
{ "_id" : null, "value" : 39 }  
> db.rezultati.findOne().value  
39
```

naziv zbirke, v kateri se bo shranil rezultat

izpis rezultata

# MapReduce: primer 2

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 } }
```

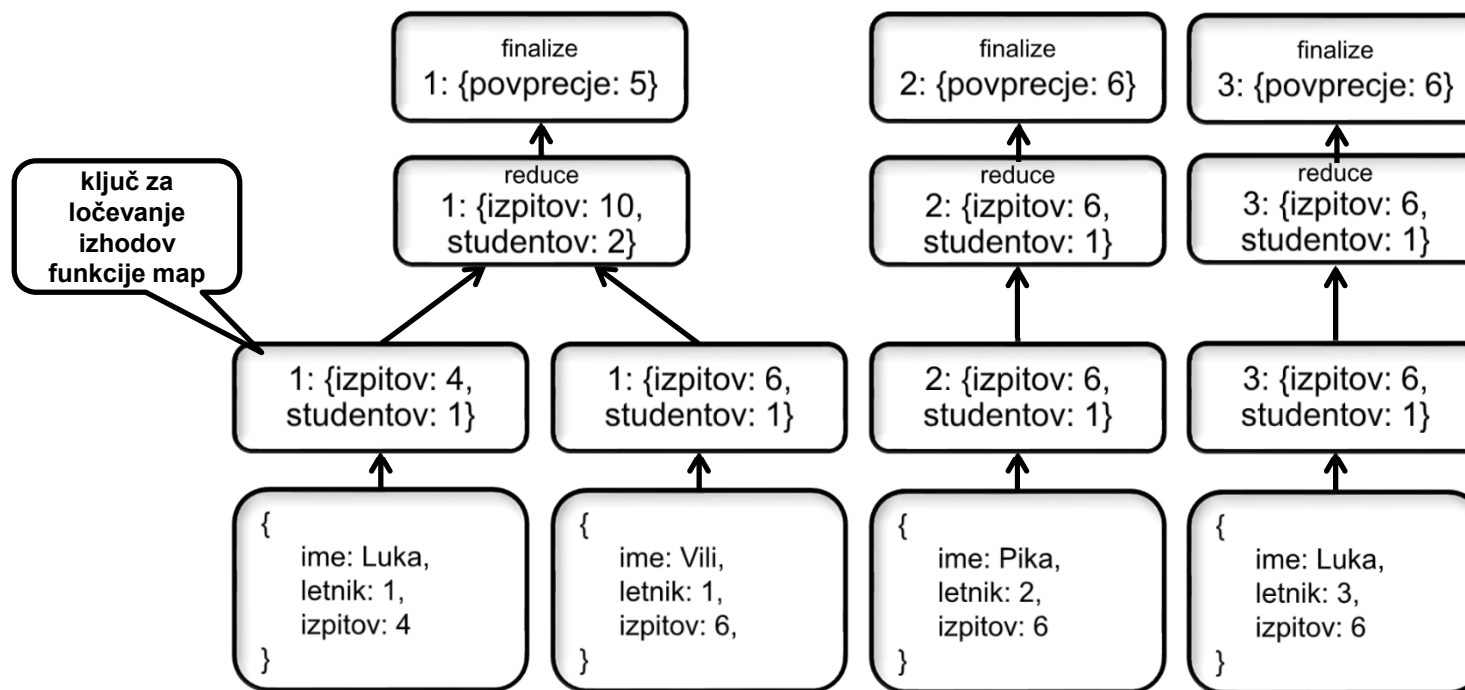
funkcija za  
finalizacijo  
rezultata, ki se  
izvede

dodatni parameter  
za uporabo  
finalizatorja

# MapReduce: primer 3

3.) Kakšno je povprečno število izpitov, ki so jih opravili študenti po posameznih letnikih?

- emit(key, value)



# MapReduce: primer 3

Kakšno je povprečno število izpitov, ki so jih opravili študenti po posameznih letnikih?

```
var mapper = function () {  
  emit(this.letnik, { studentov: 1, izpitov: this.izpitov })  
}  
% reducer in finalizer ostaneta enaka!
```

uporaba ključa

```
> db.izpiti.mapReduce(mapper, reducer,  
  { out: {inline: 1}, finalize: finalizer }).results  
[  
  {  
    "_id" : 1,  
    "value" : {  
      "povprecje" : 4  
    }  
  },  
  {  
    "_id" : 2,  
    "value" : {  
      "povprecje" : 3  
    }  
  },  
  {  
    "_id" : 3,  
    "value" : {  
      "povprecje" : 4.2  
    }  
  }  
]
```

izpis na konzolo

# PyMongo in map\_reduce

```
from bson.code import Code
```

objekt za hranjenje  
programske kode

```
map = Code("function() {"  
    "emit(this.letnik, {studentov: 1, izpitov: this.izpitov})"  
    "})")
```

```
reduce = Code("function(key, values) {"  
    "var sum = {studentov: 0, izpitov: 0};"  
    "values.forEach(function(x) {"  
        "sum.studentov += x.studentov;"  
        "sum.izpitov += x.izpitov;"  
    "});"  
    "return sum;"  
    "})")
```

zapis programske  
kode v več vrsticah

```
result = db.izpiti.map_reduce(map, reduce, "myresults")
```

zbirka, kamor se  
zapiše rezultat

```
for doc in result.find():  
    print (doc)
```

kurzor po  
rezultatih



# MongoDB - literatura

The Little MongoDB Book  
by Karl Seguin



- Eelco Plugge, Tim Hawkins, Peter Membrey: The Definitive Guide to MongoDB: The NoSQL Database for Cloud and Desktop Computing. Apress, 2010.
- Karl Seguin: The Little MongoDB Book. <http://github.com/karlseguin/the-little-mongodb-book>, 2012.
- MongoDB manual: <http://www.mongodb.org/display/DOCS/Manual>
- PyMongo Documentation: <http://api.mongodb.org/python/current/>
- Donovan Hsieh, NoSQL Data Modelling, eBay, 2014  
<http://www.ebaytechblog.com/2014/10/10/nosql-data-modeling>

# Grafni SUPB

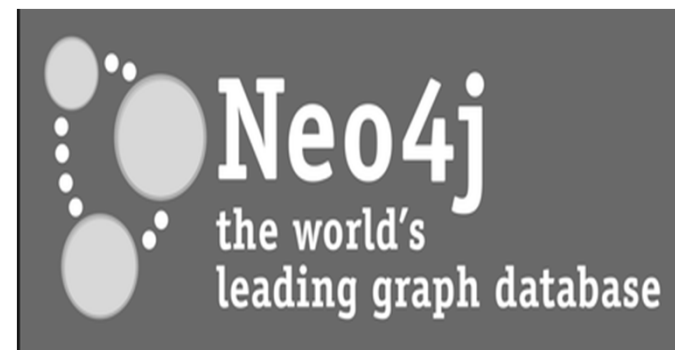
- Podatkovni model je graf z lastnostmi (omrežje, „property graph“)
  - Podatki se hranijo v vozliščih, povezavah in njihovih dodanih lastnostih
- Vsako vozlišče vsebuje kazalce na svoje naslednike
  - Posledično ne potrebujemo nujno (lahko pa) dodatnih indeksov
- Povezave vsebujejo najpomembnejši del informacije, saj povezujejo
  - Vozlišča z drugimi vozlišči
  - Vozlišča z njihovimi lastnostmi

# Neo4j, grafni nerelacijski SUPB

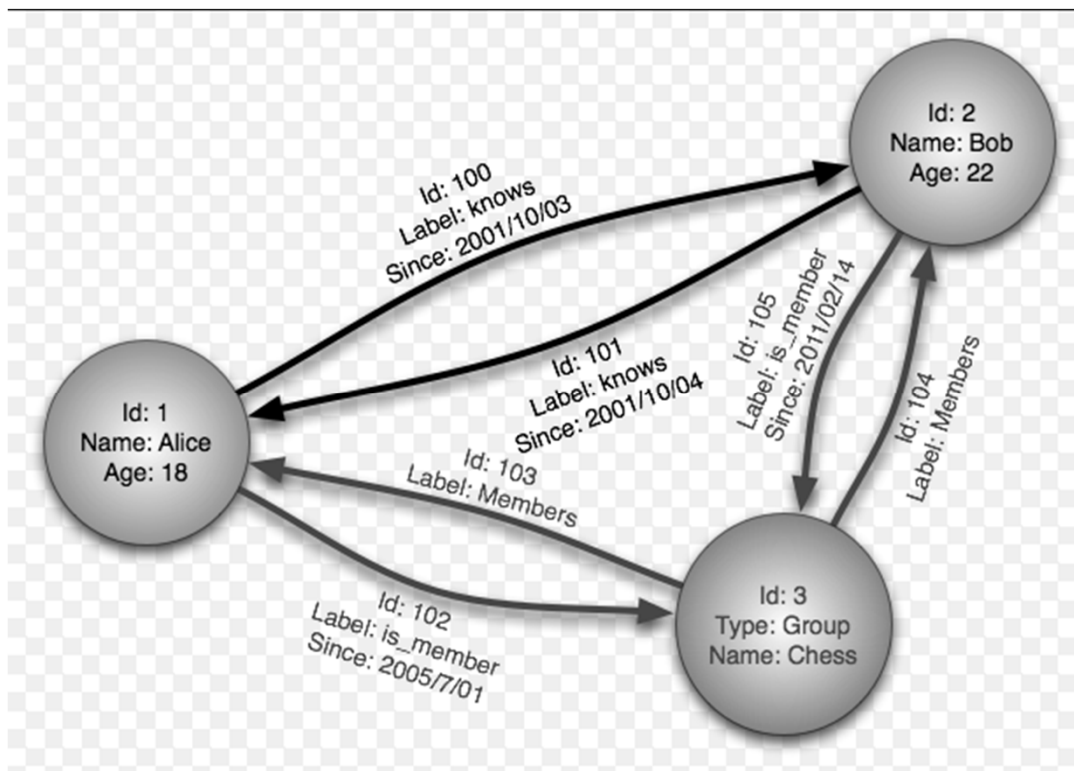
- Kaj je Neo4j?
- Grafni nerelacijski SUPB
- Povpraševalni jezik Cypher
- Povezava z aplikacijami (Python)
- Kdaj ga izbrati?
  
- Namestitev:
  - Prenos (OS in arhitektura sistema)
  - Preprosta namestitev (le specifikacija avtentikacije, kasneje jo lahko izklopite)

# Kaj je Neo4j

- Produkt podjetja Neo Technologies
- Najpopularnejši grafni SUPB
- Implementiran v Javi
- Odprtokoden



# GRAFNI SUPB

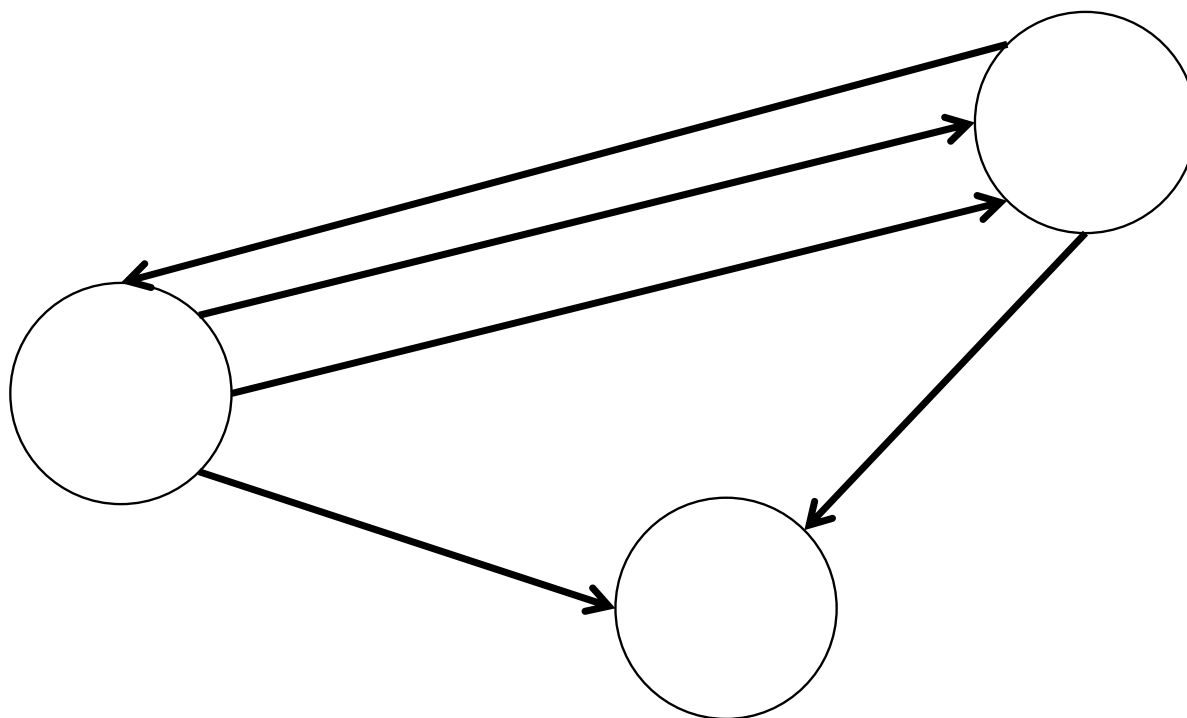


Vozlišča lahko vsebujejo

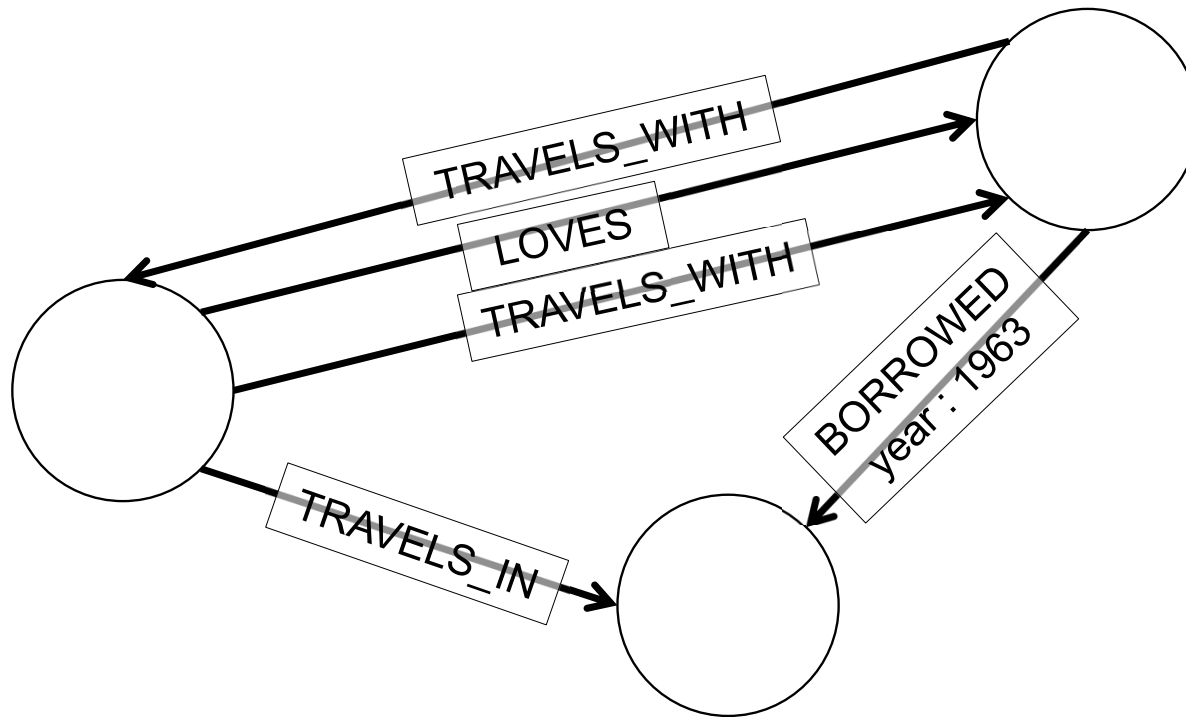
- Identifikator (id, ime)
- Lastnosti (properties): pari (ime lastnosti, vrednost)
- Oznako (label): pripadnost skupini/tipu vozlišč

(Wikipedia)

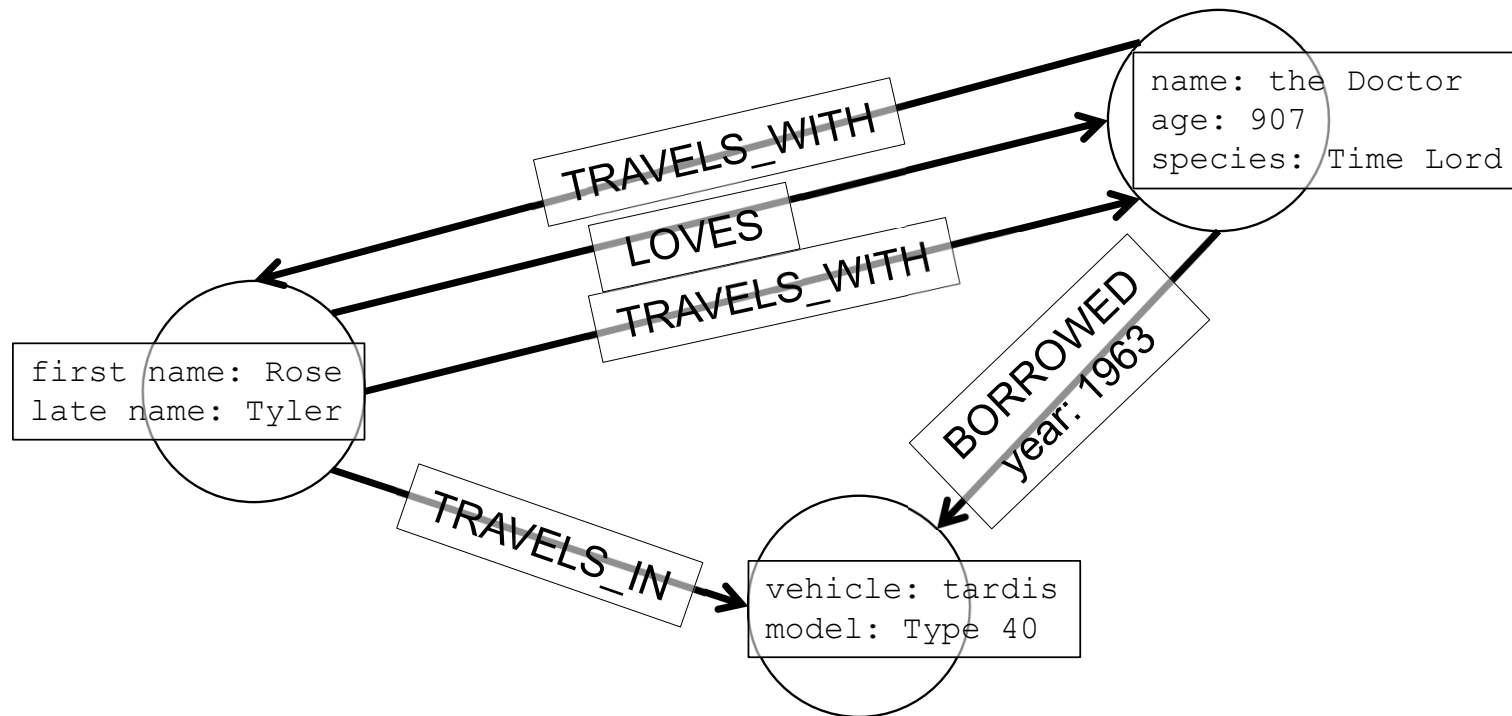
# PODATKOVNI MODEL: GRAF Z LASTNOSTMI (PROPERTY GRAPH)



# PODATKOVNI MODEL: GRAF Z LASTNOSTMI (PROPERTY GRAPH)



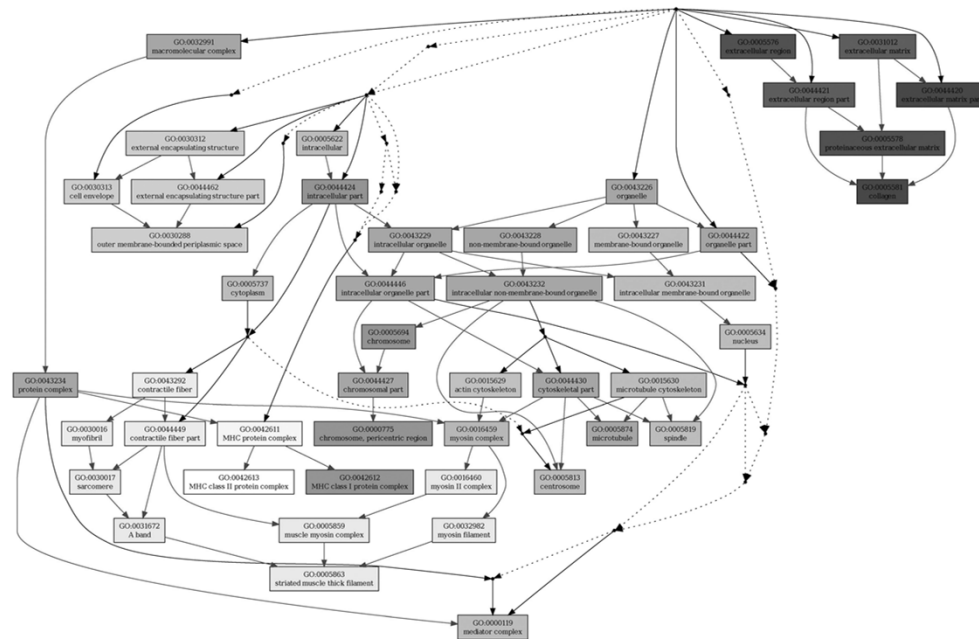
# PODATKOVNI MODEL: GRAF Z LASTNOSTMI (PROPERTY GRAPH)





# Prednosti grafnih SUPB

- Če nas zanimajo povezave (razmerja) med entitetami (vozlišči) so grafni SUPB zelo uporabni zaradi svojega podatkovnega modela
- Grafni SUPB so zelo primerne za asociativne podatkovne množice
  - Podobno kot družabna omrežja
- Omogočajo direktno preslikavo objekto usmerjenih struktur in aplikacij
  - Klasifikacija objektov
  - Hierarhije (razmerja starš - potomec)
- **Zelo fleksibilen podatkovni model**



# Pomanjkljivosti grafnih SUPB

- Manj primerno za tabelarične podatke z malo razmerji
- Slaba podpora za OLAP in druge večdimenzionalne strukture
  - Aktivno področje, potrebne nadaljnje raziskave in razvoj
- V določenih primerih problematično horizontalno skaliranje

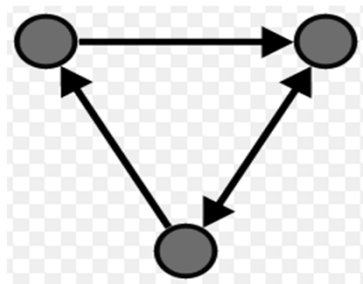
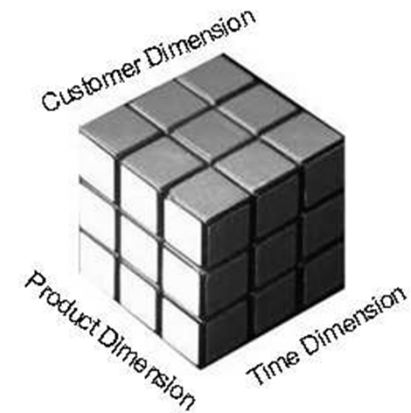


Diagram illustrating a table structure with labels:

- Table name: STUDENTS
- Column name: Rollno, Name, Phone
- Tuple / Row: s1, s2, s3, s4
- Table / Relation: The entire table structure
- Attribute / Column: Individual data points within the table

Rollno	Name	Phone
s1	Louis Figo	454333
s2	Raul	656675
s3	Roberto Carlos	546782
s4	Guti	567345

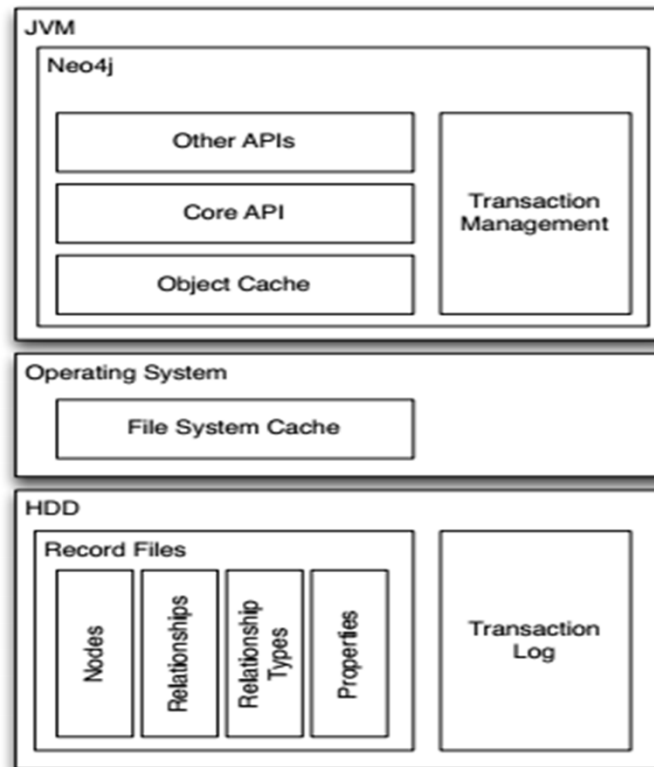


Enostavnost in učinkovitost agregacije

# Primerjava Neo4j in relacijskih SUPB

- Nima vnaprej definirane sheme/konvencije, ki se je mora držati
  - Vsako vozlišče/povezava ima lahko poljuben nabor lastnosti
- Podpora ACID transakcijam (logičnim enotam dela)
- Povpraševalni jezik Cypher (nekoliko podoben SQL)
  - Inicijativa openCypher (virji/materiali za implementacijo jezika)
  - Del iniciative SQL/PGQ (Property Graph Queries in SQL), integracija v SQL ISO standard SQL/PGQ:2020, trenutno v fazi specifikacije zahtev
- Enostaven za učenje in uporabo
- Dobra dokumentacija, velika podpora skupnosti
- Podpora sodelovanju z drugimi jeziki
  - Java, Python, Perl, Scala, ...

# ARHITEKTURA NEO4J



(Bachman, 2013, p.11)

# Povpraševalni jezik Cypher

- Eden od načinov za povpraševanje v Neo4j (ostalo: REST API , Gremlin, Core API + povezave z zunanji jeziki, ...)
- Formulacija povpraševanj, ki temelji na razmerjih med podatki
- Specifikacija z vzorci (patterns)
- Primer vzorca (ustvarjanje povezave med konkretnima vozliščema):

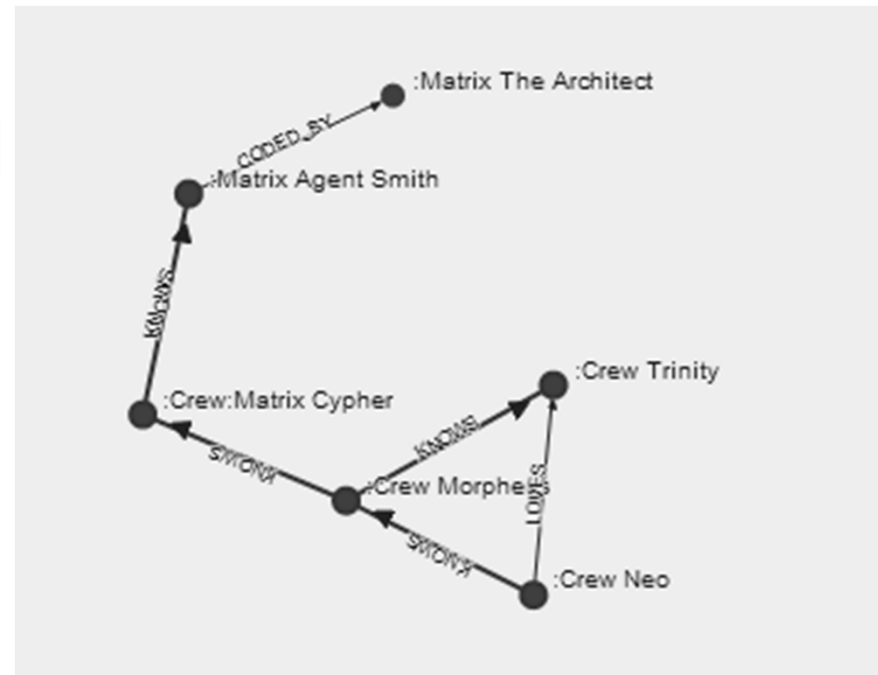
```
MATCH (a:Person { name: 'Ann' }), (b:Person { name: 'Dan' })  
      CREATE (a)-[:KNOWS]->(b)
```

- Mnoge značilnosti izhajajo iz želje po odpravi težav relacijskih SUPB (npr. odprava stičnih tabel)

# CYPHER

```
CREATE (Neo:Crew { name:'Neo' })
```

```
(Neo)-[:KNOWS]->(Morpheus)
```

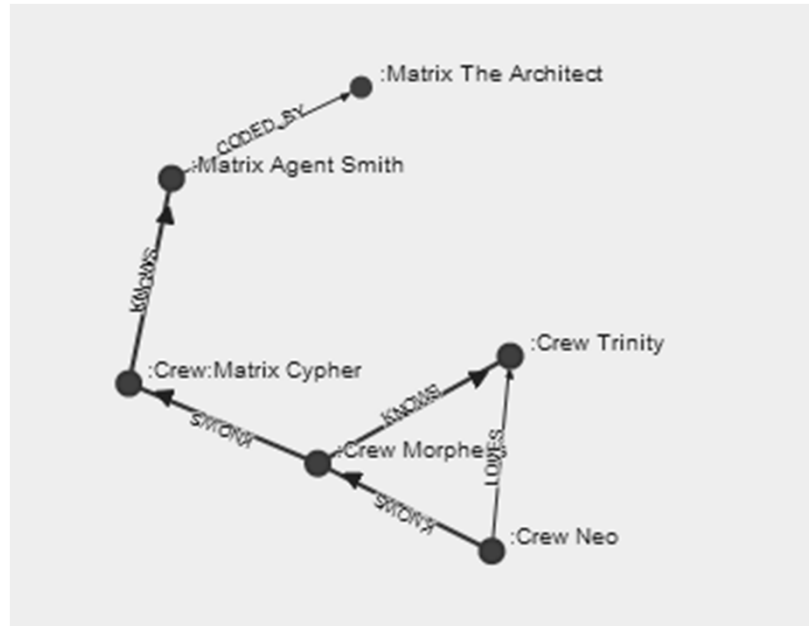


([www.neo4j.org/learn/cypher](http://www.neo4j.org/learn/cypher))

# CYPHER

```













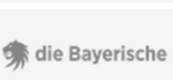











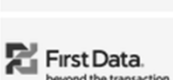


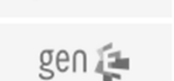


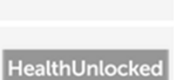


















Query:
MATCH (n:Crew)-[r:KNOWS*]-m
WHERE n.name='Neo'
RETURN n AS Neo,r,m
    
```



Neo	r	m
{name:"Neo"}	[(0)-[0:KNOWS]->(1)]	(1:Crew {name:"Morpheus"})
{name:"Neo"}	[(0)-[0:KNOWS]->(1), (1)-[2:KNOWS]->(2)]	(2:Crew {name:"Trinity"})
{name:"Neo"}	[(0)-[0:KNOWS]->(1), (1)-[3:KNOWS]->(3)]	(3:Crew:Matrix {name:"Cypher"})
{name:"Neo"}	[(0)-[0:KNOWS]->(1), (1)-[3:KNOWS]->(3), (3)-[4:KNOWS]->(4)]	(4:Matrix {name:"Agent Smith"})

([www.neo4j.org/learn/cypher](http://www.neo4j.org/learn/cypher))

# KDO VSE UPORABLJA NEO4J?

Tudi:

- Ebay
- LinkedIn
- Walmart
- ...



# Kdaj uporabiti grafni SUPB?

- Ključna vprašanja
  - Ali bomo obravnavali predvsem razmerja?
  - Kakšna bodo poizvedovanja?
- Grafni SUPB Neo4j je med najpogosteje uporabljanimi nerelacijskimi SUPB, popularnejši so le MongoDB (dokumentni), Cassandra (stolpčni), Redis (ključ-vrednost), Hbase (stolpčni).
- Najpopularnejši grafni SUPB (2020), poudarjeni so čisti grafni SUPB, neo4j ima več kot 50% delež

- |                              |                   |
|------------------------------|-------------------|
| 1. <b>Neo4j</b>              | 7. Amazon Neptune |
| 2. Microsoft Azure Cosmos DB | 8. GraphDB        |
| 3. ArangoDB                  | 9. <b>Dgraph</b>  |
| 4. OrientDB                  | 10. <b>Giraph</b> |
| 5. Virtuoso                  |                   |
| 6. <b>JanusGraph</b>         |                   |

# Neo4j / grafni SUPB in skaliranje

- Horizontalno skaliranje grafov in izvajanje grafnih algoritmov je v splošnem problematično!
- Zakaj: tipično visoka povezanost netrivialnih grafov („six degrees of separation“), zato so potrebne poizvedbe med vozlišči
- Vendar: iskanje vzorcev je pogosto možno dobro paralelizirati (neodvisnost)
- Možne rešitve:
  - Popolna replikacija podatkov (grafov) na vseh vozliščih – ni vedno mogoče. Enostavno, vendar prostorsko potratno!
  - Omejevanje globine lokalnih povezav oz. poti (npr. do 3 lokalno, višje/daljše lahko tudi med vozlišči). Relativno zahteven pristop!
- Neo4j - tipična topologija: porazdeljeno branje, centralizirano zapisovanje, popolna replikacija podatkov

# Neo4j - literatura

- Ian Robinson, Jim Webber, Emil Eifrem: Graph Databases, 2nd Edition, O'Reilly Media, <https://neo4j.com/books/> (free)
- <http://www.neo4j.org>
- <http://www.neo4j.org/learn/cypher>
- Bachman, Michal (2013). GraphAware: Towards Online Analytical Processing in Graph Databases
  - <http://graphaware.com/assets/bachman-msc-thesis.pdf>
- Hunger, Michael (2012). Cypher and Neo4j
  - <http://vimeo.com/83797381>
- Mistry, Deep (2013). Neo4j: A Developer's Perspective
  - <http://osintegrators.com/opensoftwareintegrators%7Cneo4jadevelopersperspective>
- Wikipedia (Neo4j, Graph Database)

# Zaključek: nerelacijski SUPB – da ali ne?

- Ni standardnega povpraševalnega jezika.
- V primerjavi z relacijskimi PB problematično pisanje (nižjenivojski jeziki) in optimizacija poizvedb.
- Podpora transakcijam ACID. Ali jih res ne potrebujemo na dolgi rok?
- BASE zahteva trezen razmislek, sicer lahko vodi v zmedo.
- Kdaj torej uporabiti nerelacijske SUPB?
  - Zahtevana masivna (webscale) paralelizacija
  - Ni potrebe po transakcijah
  - Ne potrebujemo enovitega, standardnega povpraševalnega jezika

# Za kaj oz. kdaj je torej primeren NoSQL

- Netransakcijski sistemi
- Enozapisne, komutativne transakcije
- Neprimerno za sodobne OLTP sisteme
- Ne potrebujemo enovitega povpraševalnega jezika
  - programska koda
  - CQL, UnQL (po zgledu SQL, nestandardno, nekompatibilno)