SZERVER OLDALI JAVASCRIPT

8. hét

MongoDB, séma tervezés, performancia kérdések

ADATBÁZISOK

Model - View - Controller (vagy MW vagy bármi)

TRADICIONÁLIS SQL

- Adatot tárol perzisztens módon (CRUD)
- Előre meghatározott séma, struktúra, jól ellenőrizhető
- Nagyon jól optimalizálható, indexelhető
- Iszonyatosan komplex adattárolás (de egyszerű interface)
- Kiforrott technológia (10-20+ éve létezik)

NOSQL

Not Only SQL

- Egyszerűbb felépítés (kevesebb művelet)
- u yorsaság és skálázhatóság
- Teljesen más tervezési elvek, mint az SQL esetében
- Jobban illeszkedik a "modern" elvárásokhoz (?)

NOSQL

Típusai

- Dokumentum adatbázisok (JSON, YAML, XML) - MongoDB
- Oszlopos (vagy tabulált) adatbázis - o assandra
- u ráf adatbázisok - Neo4j
- Kulcs-érték adatbázisok (vagy Tuple, quad store) - Redis
- Objektum adatbázisok - OrientDB

MONGODB

2007-ben kezdték fejleszteni, 2009-től open source

Dokumentum adatbázisok közül a legnépszerűbb
(Craigslist, eBay, Foursquare, LinkedIn használja itt-ott)

MONGODB

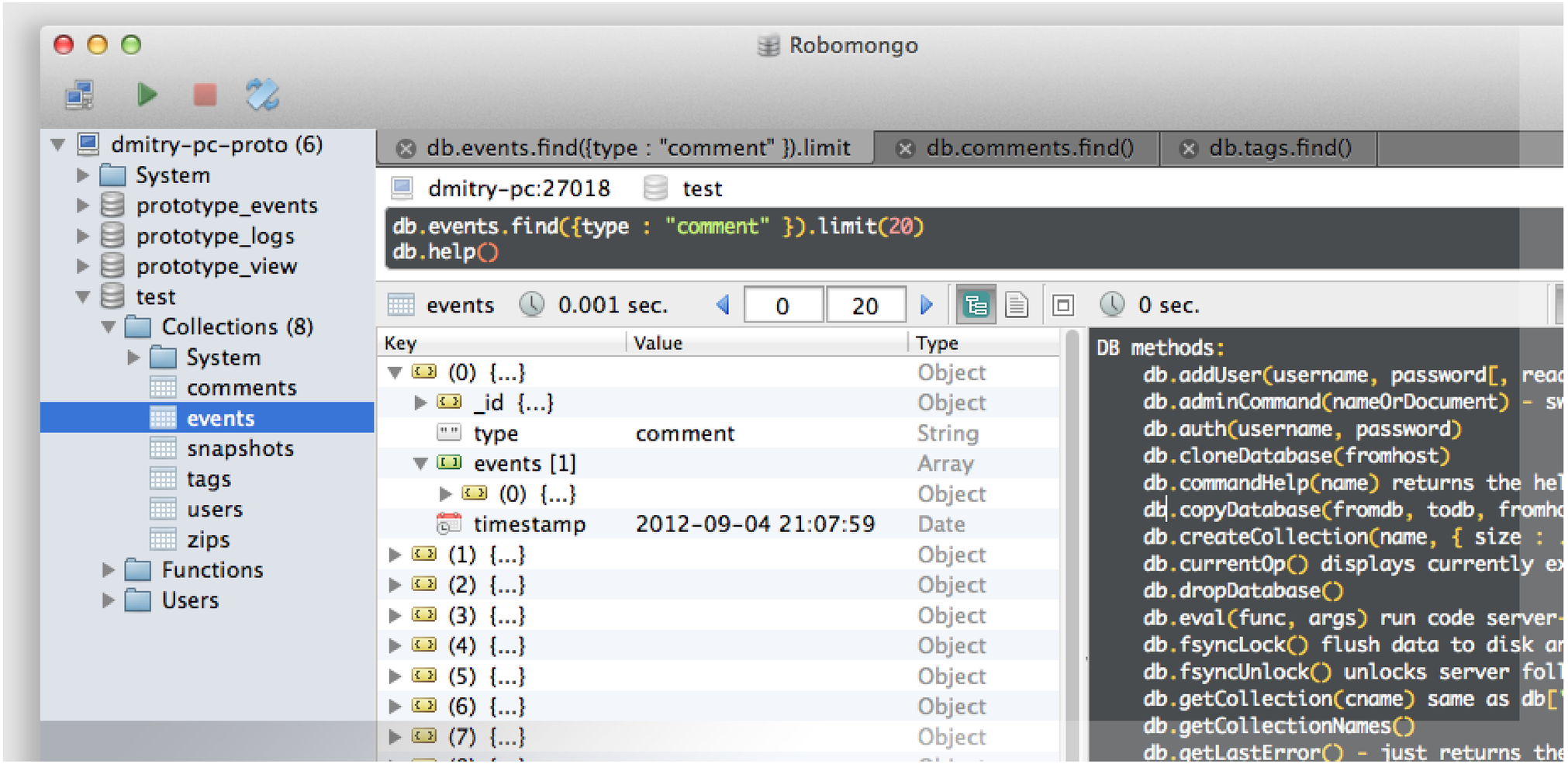
Bár C++ -ban írták, de egy "mongo" shellel érkezik, amin keresztül parancssorból is kezelhetők. A shell javascript kódot fogad el, a használt javascript engine a V8.

A mutatott kódot a mongo shellben futtathatóak.

TELEPÍTÉS

https://www.mongodb.org/downloads

FELTELEPÜLT, DE HOGYAN ÉREM EL?



Sok GUI van, mi Robomongo-t fogunk használni (mert egyszerű és crossplatform).

ADATSTRUKTURA

MongoDB végtelenül egyszerű (félig strukturált):

- collection: "kb" mint egy tábla, azonos típusú dokumentumokat fogja össze (nem követel meg strukturális azonosságot)
- dokumentum: maga az adat

Természetesen léteznek adatbázisok, mint a hozzáférés és tárolás legmagasabb szintje.

DOKUMENTUM

Lényegében egy JSON (BSON) minden dokumentum (az _id a dokumentum azonosítója):

```
{
   "_id" : ObjectId("54c955492b7c8eb21818bd09"),
   "address" : {
      "street" : "2 Avenue",
      "zipcode" : "10075",
      "building" : "1480",
      "coord" : [ -73.9557413, 40.7720266 ],
   },
   "borough" : "Manhattan",
   "cuisine" : "Italian"
}
```

ALAPMŰVELETEK - BESZÚRÁS

db.collection.insert(document or array of documents,optional_options);

```
db.inventory.insert(
   {
     item: "ABC1",
     details: {
        model: "14Q3",
        manufacturer: "XYZ Company"
     },
     stock: [ { size: "S", qty: 25 }, { size: "M", qty: 50 } ],
     category: "clothing"
   }
)
```

ALAPMŰVELETEK - TÖRLÉS

db.collection.remove(query,justOne)

```
db.inventory.remove( { category : "clothing" } )
```

ALAPMŰVELETEK - MÓDOSÍTÁS

db.collection.update(query,update,optional_options);

```
db.inventory.update({ item: "ABC1" },{category: "Food"})
```

ALAPMŰVELETEK - KERESÉS

db.collection.find(query, projection) vagy
db.collection.findOne(query, projection

```
db.inventory.find(
   {
     type: 'food',
     $or: [ { qty: { $gt: 100 } }, { price: { $lt: 9.95 } } ]
   }
)
```

Jelentősen komplexebb tud lenni, de jól dokumentált a forma:
https://docs.mongodb.org/manual/reference/operator/

COLLECTION ÁLTAL DEFINIÁLT STRUKTÚRA

Nem definiáltunk még dokumentum strukturát, így bármilyen dokumentumok "elférnek" egy collectionben.

COLLECTION ÁLTAL DEFINIÁLT STRUKTÚRA

Miért kellhet egyáltalán struktúra nekünk?

1. Collectionök közötti kapcsolatok
2. Típusok / kötöttségek ellenőrzése
3. Indexek létrehozása a gyors keresés miatt

KAPCSOLATOK COLLECTIONÖK KÖZÖTT

Manuális referencia: **_id** alapján hivatkozás egy másik objektumra

```
original_id = ObjectId()

db.places.insert({
    "_id": original_id,
    "name": "Broadway Center",
    "url": "bc.example.net"
})

db.people.insert({
    "name": "Erin",
    "places_id": original_id,
    "url":  "bc.example.net/Erin"
})
```

A legtöbb driver ezt feloldja mikor lekérdezést végzel.

KAPCSOLATOK COLLECTIONÖK KÖZÖTT

DBRef: direkt explicit kapcsolat két objektum között

```
db.places.insert({
  "_id" : ObjectId("5126bbf64aed4daf9e2ab771"),
  "name": "Broadway Center",
  "url": "bc.example.net"
  "creator" : {
                "$ref" : "creators",
                "$id" : ObjectId("5126bc054aed4daf9e2ab772"),
                "$db" : "users"
              }
})
```

TÍPUSOK

A mongoDB bár nem erőlteti, de ismeri a típusokat.

Double, String, Object, Array, Binary data, Object id, Boolean, Date, Null, Regular Expression, JavaScript, Symbol, JavaScript (with scope), 32-bit integer, Timestamp, 64-bit integer

Mihez kellhet: összehasonlítás, sorrendezés, aggregáció, sharding, stb.

TÍPUS SZINTŰ LIMITÁCIÓK

Ez a driverek feladata, a mongoDB-t nem érdekli.

INDEXEK LÉTREHOZÁSA

Továbbra is az egyszerűség a jellemző:

```
db.records.createIndex( { userid: 1 } )
db.products.createIndex( { item: 1, category: 1, price: 1 } )
db.accounts.createIndex( { "tax-id": 1 }, { unique: true } )
```

Sharding alapja az indexelés tud lenni, ez alapján osztható szét az adat egy clusterben.

HASZNÁLJUK MINDEZT NODE.JS ALÓL

Ehhez kell egy npm modul: mongoose

```
npm install mongoose --save
```

https://www.npmjs.com/package/mongoose

A MONGOOSE EGYSZERŰ!

```
const mongoose = require('mongoose');
mongoose.connect('mongodb://127.0.0.1:27017/test');

const Cat = mongoose.model('Cat', { name: String });

const kitty = new Cat({ name: 'Zildjian' });
kitty.save().then(() => console.log('meow'));
```

https://mongoosejs.com/docs/migrating_to_7.html#dropped-callbac˝-support

ODB

A mongoose az ODB (majdnem ORM) funkciók miatt megköveteli, hogy sémát definiáljunk, hogy le tudja képezni a lekérdezéseket / válaszokat.

SCHEMA

A *Schema* osztály felelős azért, hogy a validációt illetve bármilyen kiterjesztett működést lehetővé tegyen. Csak akkor használjuk, ha valami "komplex dolog" kell.

```
const schema = kittySchema = new Schema({ name: String }));

schema.method('meow', function () {
  console.log('meeeeeooooooooooooow');
})

const Kitty = mongoose.model('Kitty', schema);

const fizz = new Kitty;
fizz.meow();
```

SCHEMA

A Schema gyakorlatilag minden modern ORM elvárást meg tud oldani:

```
const schema = new Schema({
  name:     String,
  binary:   Buffer,
  living:   Boolean,
  updated: { type: Date, default: Date.now },
  age:      { type: Number, min: 18, max: 65 },
  mixed:    Schema.Types.Mixed,
  _someId: Schema.Types.ObjectId,
  array:       [],
  ofString:    [String],
  ofNumber:    [Number],
  ofDates:     [Date],
  ofBuffer:    [Buffer],
  ofBoolean:   [Boolean],
  ofMixed:     [Schema.Types.Mixed],
  ofObjectId: [Schema.Types.ObjectId],
  nested: {
```

MODEL

A *Schema* vagy egyszerűbb attributum lista alapján létrehozható a tényleges model (ODB szint).

```
const Cat = mongoose.model('Cat',
    {
        name: String
        _owner: {
            type: mongoose.Schema.Types.ObjectId,
            ref: 'Owner'
        },
    });
```

PÉLDÁNY LÉTREHOZÁSA, MÓDOSÍTÁSA

A model alapján már létre tudunk példányt hozni, ami objektumként viselkedik.

```
const fizz = new Kitty();
fizz.name = 'Cica';
fizz.save().then(() => {
    //console.log
}).catch(err=>{
    //console.error
});
```

TÖRLÉS

```
Comment.deleteOne({ title: 'baby born from alien father' })
    .then(() => console.log).catch(console.error);

Comment.deleteMany({ _id: id }).then(console.log).catch(console.error)

Comment.findOne({ _id: id }).then(comment => {
  return comment.deleteOne();
}).then(console.log).catch(console.error);
```

7.x: The remove() method on documents and models has been removed. Use deleteOne() or deleteMany() instead.

KERESÉS

```
Kitten.find({name : 'Cica'})
    .then(cicak =>{ ... }).catch(err=>{ ... });
Kitten.findOne({name : 'Cica'})
    .then(cica =>{ ... }).catch(err=>{ ... });
```

Mongoo shell szintű lekérdezések is elvégezhetőek:

http://mongoosejs.com/docs/queries.html

MIRE ÉRDEMES ODAFIGYELNI

- Séma tervezés változatlan, az ER diagramoknak itt is van haszna
- ObjectId castolása néha nem triviális
- Az adatbázisban lévő adatoka schema módosításával nem változnak (nincs migráció)

PERFORMANCIA

1. Adatstruktura
2. Indexek
3. Cache
4. Skálázhatóság
5. Architektura
6. Ténylegesen használt eszköz

PERFORMANCIA

Ha ezek után is gond lenne:

- db.setProfilingLevel(1)
- query.explain("executionStats")

A segítség:
https://docs.mongodb.org/manual/tutorial/analyze-query-plan/

HÁZI FELADATHOZ

A háziban saját adatbázis nevet használjatok: **NEPTUN KÓD!!!** (localhost, default port), az első collection íráskor ugyanis létrejön az adatbázis (nálatok is és az ellenőrzésnél is)!