2) CRUD, Kurzory

2.1) CRUD

- Atomicita, transakcia a konzistencia

a) **Atomicita – nevykonané čítanie**
b) **Dvojfázové vykonanie a sémantika podobná transakcie**
c) **Trvanlivosť a eventuálna konzistencia**
d) **Monotonicita zápisu a čítania**

- Insert, update, delete

2.2) Dopytovanie

2.3) Príklady a kurzory

1) Kolekcia s poľom hodnôt

1a) Vytvorenie (insert) kolekcie maz1 s poľom/array hodnôt A
1b) Ktoré kľúče treba vrátiť
1c) Dopytovanie poľa
1d) Kurzor a forEach

2) Kolekcia s poľom vnorených dokumentov

2a) Kurzor
2b) Kurzor to array

3) Generovanie kolekcie
4) Kurzory a JavaScript

2.1) CRUD

Úvod do MongoDB CRUD

MongoDB na čítanie a zapisovanie používa **zámky**, ktoré umožňujú súbežným čitateľom zdieľaný (shared) prístup k zdroju, ale ktoré zabezpečujú **exkluzívny** prístup zápisu jedného dokumentu.

Sada replík a CRUD

Ak v danom momente **primarny** člen sa funkčne zlýha, zo **sekundárnyh** členov sa zvolí nový primárny.

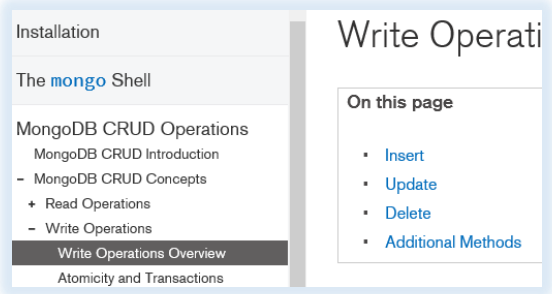
V sade replík sekundárni členovia môžu spracovať čítanie, ale zápis iba primárny člen.

MongoDB insert a update operácie prebiehajú **v poradí**:

- záznam u primárneho člena
- potom záznam u primárneho člena do **oplog**u (operation log)
- vytvorenie kópií u sekundárnych členov asynchrónne.

Write operácie

- Insert dokumenty
- Update
- Delete
- Ďalšie metódy



Read operácie

- Query Interface
- Query pravidlá
    - Dopyt v MongoDB sa vždy vzťahuje iba na jednu kolekciu
    - Za dopytom môžeme nastaviť limit, skip a sort
    - Okrem sortu nie je možné určit poradie výsledku dopytu
    - Dopytovať v MongoDB okrem find môžeme aj pomocou kurzora (forEach) a agregácie
- Query príkazy
- Projekcie

Atomicita, transakcia a konzistencia

a) **Atomicita – nevykonané čítanie**
b) **Dvojfázové vykonanie a sémantika podobná transakcie**
c) **Trvanlivosť a eventuálna konzistencia**
d) **Monotonicita zápisu a čítania**

a) Atomicita – nevykonané čítanie

Atomickosť/atomicita zápisu vzhľadom na jeden dokument znamená, že ak zápis v dokumente aktualizuje viac kľúčov, čiastočné aktualizácie sa nesmú načítať, ich čitateľ nemôže uvidieť.

Aj keď čitateľ nemôže vidieť čiastočne aktualizovaný jediný dokument, konkurenční čitatelia za isté okolnosti predsa môžu uvidieť aktualizovaný dokument predtým, ako zmeny sú trvanlivé. To sa nazýva nevykonané čítanie (**read uncommitted**).

Keď jediná operácia zápisu ovplyvňuje viac dokumentov, úprava každého dokumentu je atómická, ale operácia ako **celok nie je atimická** - ďalšie operácie môžu zasahovať, prelínať. Jedinú operáciu **zápisu**, ktorá ovplyvňuje viac dokumentov, je možné **izolovať** pomocou operátora **$izolated**, avšak izolovaný zápis neposkytuje "všetko alebo nič" atomicitu.

Pre jedinú mongod inštanciu operácie čítania a zápisu do jediného dokumentu sú **serializovatelné**. V prípade skupiny **replík** iba v neprítomnosti **rollback**.

b) Dvojfázové vykonanie a sémantika podobná transakcie

Pretože jeden dokument často obsahuje niekoľko vložených dokumentov, **atomicita jedného** dokumentu je dostatočná pre veľa praktických prípadov. Pre viac zápisov, ktoré by sa mali chovať ako

jedna transakcia, je možné vyžiadať **dvojfázové vykonanie** (commit), ktoré zaručuje konzistenciu dát a v prípade chyby stav pred transakciou je obnoviteľný. V priebehu vykonania sa však dokumenty a dáta môžu byť v stave čakania (pending). Teda **konzistencia** dát je zaručená *oneskorene*, lebo môže sa stať, že aplikácie počas dvojfázového commitu alebo rollbacku vráti prechodné dáta - to sa nazýva sémantika podobná transakcie (**transaction-like semantics**).

c) Trvanlivosť a eventuálna konzistencia

V MongoDB, klienti môžu vidieť, čítať **výsledky zápisov** predtým, ako zápisy sú trvanlivé. Operácia zápisu je **trvanlivá** (durable), kedy bude pretrvávať po vypnutí (alebo havárii) a reštarte jedného alebo viacerých serverových procesov.

- U jedného MongoDB servera, operácia zápisu je považovaná za trvanlivú, keď bola zapísaná do **súboru denníka** (journal file) servera.
- Pre skupinu replík (replica set), operácia zápisu je *značne* trvanlivá, akonáhle operácia zápisu je odolná na väčšine **hlasovacích uzlov** (voting nodes) v skupine replík; tzn. zapísané do väčšiny súborov denníka hlasovacích uzlov.

MongoDB pri lokálnom (*local*) **čítaní** vráti najnovšie údaje, ktoré sú k dispozícii v okamihu dotazu, a to bez garancie, že dáta boli trvanlivo zapísané na väčšine členov skupiny replík s možnosťou vrátenia stavu späť.

Čítanie môže byť označené aj ako *majority*.

Eventuálna konzistencia (eventual consistency) je vlastnosť distribuovaného systému, ktorá umožňuje, aby zmeny v systéme (viditeľne) nastali postupne. V databázovom systéme, to znamená, že čitateľní členovia nemusia (jednotne) odrážať najnovšie zápisy za všetkých okolností.

Uvažujme skupinu replík s jedným *primárnym* členom. Ak zápis je

- *local*, čítania z primárneho odrážajú najnovšie zápisy (za predpokladu, že nedošlo k zlyhaniu);
- *majority*, operácie čítania z primárnych alebo sekundárnych členov majú eventuálna konzistenciu.

V MongoDB **kurzor** môže vrátiť ten istý dokument **viackrát** za isté okolnosti (napr. pri zmene indexovaného kľúča), lebo počas vrátenia dokumentov kurzorom, s dotazom sa môžu prelínať iné operácie.

d) Monotonicita zápisu a čítania

Kým MongoDB zaručuje

- **monotónny zápis** pre samostatné inštancie mongod, skupiny replík a sharded klastre
- **monotónne čítanie** iba pre samostatné inštancie mongod.

Predpokladajme, že aplikácia vykoná postupnosť operácií, ktorá sa skladá

- z operácie čítania R1
- za ktorou nasleduje ďalšia operácia čítania R2.

V prípade, že aplikácia vykonáva postupnosť operácií na samostatnej inštancii mongod, neskoršie čítanie R2 nikdy nevracia výsledky, ktoré odrážajú skorší stav, ako je vrátené z R1; tzn. R2 vracia dáta, ktoré rastú monotónne (**monotonically increasing**) na aktuálnosti od R1.

CRUD príkazy

- insert vloží jeden alebo viac dokumentov do kolekcie
- update aktualizuje jeden alebo viac dokumentov
- delete maze jeden alebo viac dokumentov
- find vyhľadáva dokumenty v kolekcii
- getLastError vracia chybu poslednej operácie

Kolekcia a dokumenty

- find, findOne, distinct
- insert, insertOne, insertMany
- validate
- update, updateOne, updateMany, replaceOne
- remove, deleteOne, deleteMany

- drop, dropIndex
- aggregate, count, group
- mapReduce – filter + summary

Kurzor a dokumenty

- forEach, next, hasNext,
- limit, skip, size,
- count, min, max,
- map, sort, toArray

Dokumenty

- db.kol1.insert()
- db.kol1.insertOne()
- db.kol1.insertMany()

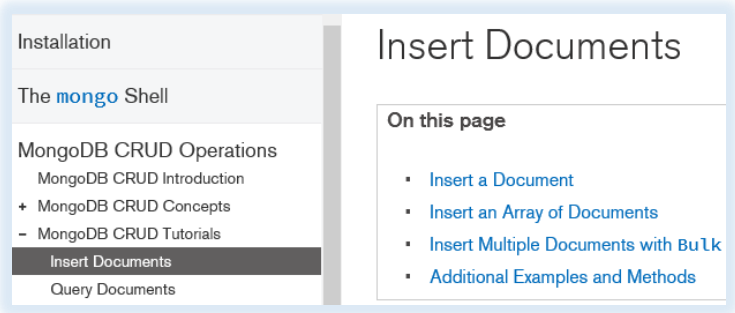
- db. kol1.remove()
- db. kol1.deleteOne()
- db. kol1.deleteMany()

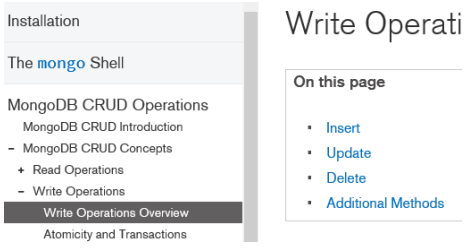
- db.kol1.update()
- db. kol1.updateOne()
- db. kol1.updateMany()
- db. kol1.replaceOne()

update operátory

- kľúča
- poľa

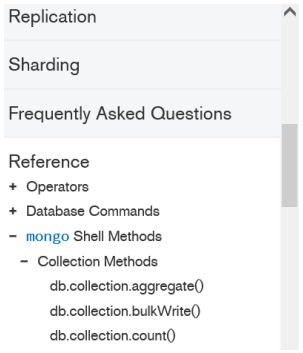
• Insert dokumenty

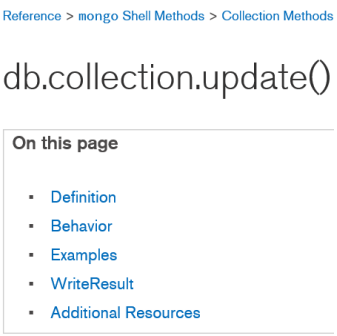




update, remove

```
db.uuu.drop();
db.uuu.insert({ "_id" : 1, "vaha" : 70, vyska : 174 });
var ii = { "_id" : 2, "vaha" : 82, vyska : 175 };
db.uuu.insert(ii);
db.uuu.find();
```





upsert – ak neexistuje, potom insertuj

```
try {
  db.uuu.update(
    { vaha : 75 },
    { $set: { vyska : 175 } }
   ,{ upsert: true }
  );
} catch (e) {
  print(e);
}

db.uuu.find();
```

```
db.uuu.find();
try {
  db.uuu.remove( { "_id" : 1} );
} catch (e) {
  print(e);
}

db.uuu.find();
```

justOne : false – delete viac

```
db.uuu.find();
try {
  db.uuu.remove( { vyska : 175}, {justOne: false} );
} catch (e) {
  print(e);
}

db.uuu.find();
```

2.2) Dopytovanie

db.kol1.find({Horiz.filt}, {Vert.filt.});

Poznámka: RDB pojmy *horizontálna* a *vertikálna filtrácia* sa v MongoDB nepoužívajú. Namiesto nich sa hovorí o dopytovaní, načítaní dokumentov a vrátení vybraných kľúčov.

{Vert.filt.} ⇔ {klucJ : 01, …, klucK : 01},

kde

- 01 sa rovná 1, potom hodnoty zodpovedajúceho kľúča sa vrátia

- 01 sa rovná 0, potom hodnoty zodpovedajúceho kľúča sa nevrátia

```
db.kol1.find( {}, {_id : 0, x : 1,  "A.$": 1 } ); // vrat iba A[1]
```

Najprv sa heslovite pozreme na {Horiz.filt}

I. Dopytovanie **bez** polí a vnorených dokumentov a polí
   A. Dopytovanie **bez vonkajšieho** modifikátora
      a) **Jednoduché** dopytovanie {kluc1 : hodnota1, …, klucM : hodnotaM}
      b) **$** dopytovanie hodnotaJ ⇔{$oper1: hod1, …, $operN: hodN }
   B. Dopytovanie **s vonkajším** modifikátorom

II. Dopytovanie
   A. **polí**
   B. **vnorených** dokumentov

kde $oper:hodnota môže byť napr. { $gt : 1, $lt < 5, $in : [2, 3, 4] }, pozri nižšie.

I. Dopytovanie bez vnorených dokumentov, a polí

A. Dopytovanie bez vonkajšieho modifikátora
B. Dopytovanie s vonkajším modifikátorom

A. Dopytovanie bez vonkajšieho modifikátora

a) Jednoduché dopytovanie {kluc1 : hodnota1, …, klucM : hodnotaM}
b) $ dopytovanie hodnotaJ ⇔{$oper1: hod1, …, $operN: hodN }

a) Jednoduché dopytovanie

{Horiz.filt} ⇔ {kluc1 : hodnota1, …, klucM : hodnotaM}

na celú hodnotu kľúča, kde hodnotaJ môže byť

- skalárna veličina, ako číslo, reťazec alebo datum: 123, "Fero", "2016.4.4"

```
db.kol1.find( { x : 12, y : "Fero", d : "2014.4.4"} );
```

- pole skalárnych veličín: [1, 12.1, 5], ["Fero", "Jano"}], ale aj [1, "Fero"]

Dopyt na hodnotu celého vektor-kľúča z

```
db.kol1.find( { x : 12,  z : [1, 12.1, 5] } );
```

b) $ dopytovanie

{Horiz.filt} ⇔ {$oper1: dok1, …, $operN: dokN }

```
db.kol1.find( { x : {$gt : "2014.4.4", $lt : "2016.4.4" }}, {y : {$in : ["Fero", "Jano"] }} );
```

B. Dopytovanie s vonkajším modifikátorom

Kým posledný dopyt je automaticky AND dopyt, OR dopyt sa určuje explicitne ako vonkajší modifikátor:

```
db.kol1.find({ $or : [ { x : {$gt : "2014.4.4",$lt : "2016.4.4" }}, {y : {$in : ["Fero", "Jano"] }} ] } );
```

II. A, B) Dopytovanie polí a vnorených dokum

```
use dbMaz;
db.maz1.drop();
db.maz1.insert( { x: "ab", A: [ 2, 3, 4 ] } );
db.maz1.insert(
   [
         { x: "aa", A: [ 2, 4 ] },
         { x: "aa", A: [ 3, 4, 2 ] }
   ]
);
```

```
db.maz2.drop();
db.maz2.insert(
[
 { x: "ab", A: [ {je:2, ke:3}, {je:2, ke:4} ] },
 { x: "aa", A: [ {je:2, ke:4}, {je:3, ke:4} ] },
 { x: "aa", A: [ {je:3, ke:4}, {je:4, ke:5} ] }
]
);
```

Prvý prvok / nultý index **poľa** A sa rovna 2

```
> db.maz1.find( { 'A.0': 2 } , {_id:0});   // !!! 'A.0'
{ "x" : "ab", "A" : [ 2, 3, 4 ] }
{ "x" : "aa", "A" : [ 2, 4 ] }
```

```
> db.maz2.findOne({}, {_id:0});
```

Dopyt

```
> db.maz2.find();
```

vráti pochopiteľne výsledok

```
{ "x" : "ab", "A" : [ { "je" : 2, "ke" : 3 }, { "je" : 2, "ke" : 4 } ] }
{ "x" : "aa", "A" : [ { "je" : 2, "ke" : 4 }, { "je" : 3, "ke" : 4 } ] }
{ "x" : "aa", "A" : [ { "je" : 3, "ke" : 4 }, { "je" : 4, "ke" : 5 } ] }
```

kde kľúč "A" obsahuje pole **vnorených** dokumentov, preto

```
> db.maz2.find( { 'A.ke': 4, "A.je": 4 }, {_id : 0} );
```

vráti

```
{ "x" : "aa", "A" : [ { "je" : 3, "ke" : 4 }, { "je" : 4, "ke" : 5 } ] }
```

$ Operátory dopytovania a projektovania

- porovnávacie operátory
  - $lt, $lte, $gt, $gte s hodnotami typu číslo alebo dátum
  - $eq, $ne s hodnotami ľubovoľného typu
  - $in, $nin
- $not
- vonkajšie modifikátory $or, $nor, $and
- $exists, $type
- pre pole
  - $elemMatch
  - $all
  - $size
  - $slice
  - $ (projection)
- $where klauzula

2.3) Príklady a kurzory

1) Kolekcia s poľom hodnôt

1a) Vytvorenie (insert) kolekcie maz1 s poľom/array hodnôt A

1b) Ktoré kľúče treba vrátiť

1c) Dopytovanie poľa

1d) Kurzor a forEach

2) Kolekcia s poľom vnorených dokumentov

2a) Kurzor

2b) Kurzor to array

3) Generovanie kolekcie

4) Kurzory a JavaScript

1) Kolekcia s poľom hodnôt

1a) Vytvorenie (insert**) kolekcie maz1 s poľom/array hodnôt** A

A: [2, 3, 4]

Vkladanie jedného dokumentu alebo viac dokumentov naraz pomocou poľa [ ]

```
db.maz1.findOne();
{
        "_id" : ObjectId("5707ed009fed16950670b068"),
        "x" : "ab",
        "A" : [
                2,
                3,
                4
        ]
}
```

```
use dbMaz;
db.maz1.drop();
db.maz1.insert( { x: "ab", A: [ 2, 3, 4 ] } );
db.maz1.insert(
  [
        { x: "aa", A: [ 2, 4 ] },
        { x: "aa", A: [ 3, 4, 2 ] }
  ]
);
```

```
db.maz1.find()
{ "_id" : ObjectId("5707ed009fed16950670b068"), "x" : "ab", "A" : [ 2, 3, 4 ] }
{ "_id" : ObjectId("5707ed009fed16950670b069"), "x" : "aa", "A" : [ 2, 4 ] }
{ "_id" : ObjectId("5707ed009fed16950670b06a"), "x" : "aa", "A" : [ 3, 4, 2 ] }
```

1b) Ktoré kľúče treba vrátiť

```
db.maz1.findOne( { }, {_id: 1} );
"_id" : ObjectId("5707ed009fed16950670b068")
```

```
db.maz1.findOne( { }, {_id: 0} );
"_id" : ObjectId("5707ed009fed16950670b068")
{ "x" : "ab", "A" : [ 2, 3, 4 ] }
```

```
db.maz1.findOne( { }, {"A":1, _id:0} );
{ "A" : [ 2, 3, 4 ] }
db.maz1.find( { }, {"A":1, _id:0} );
{ "A" : [ 2, 3, 4 ] }
{ "A" : [ 2, 4 ] }
{ "A" : [ 3, 4, 2 ] }
```

1c) Dopytovanie poľa

```
db.maz1.find( { A: [ 2, 4 ] } );
"_id" : ObjectId("5707ed009fed16950670b069"), "x" : "aa", "A" : [ 2, 4 ] }
```

```
db.maz1.find( { A: 3 } );  // ⇔
db.maz1.find( { A: { $elemMatch: { $eq: 3 } } } );
{ "_id" : ObjectId("5707ed009fed16950670b068"), "x" : "ab", "A" : [ 2, 3, 4 ] }
{ "_id" : ObjectId("5707ed009fed16950670b06a"), "x" : "aa", "A" : [ 3, 4, 2 ] }
```

```
db.maz1.find( { 'A.1': 4 } );  // !!! 'A.1' druhy prvok A sa rovna 4
{ "_id" : ObjectId("5707ed009fed16950670b069"), "x" : "aa", "A" : [ 2, 4 ] }
{ "_id" : ObjectId("5707ed009fed16950670b06a"), "x" : "aa", "A" : [ 3, 4, 2 ] }
```

```
db.maz1.find( { A: { $elemMatch: { $gt: 2, $lt: 4 } } } );
{ "_id" : ObjectId("5707ed009fed16950670b068"), "x" : "ab", "A" : [ 2, 3, 4 ] }
{ "_id" : ObjectId("5707ed009fed16950670b06a"), "x" : "aa", "A" : [ 3, 4, 2 ] }
```

1d) Kurzor a forEach

```
var kurzor = db.maz1.find();
kurzor.forEach(function(e) {print(e.x, e.A);});
ab 2,3,4
aa 2,4
aa 3,4,2
```

```
var kurzor = db.maz1.find().limit(2);
kurzor.forEach(function(e) {print(e.x, e.A[1]);});  // !!!
ab 3
aa 4
```

2) Kolekcia maz2 s poľom vnorených dokumentov A

A : [{je : 2, ne : 3}, {je : 2, ne : 4}]

```
db.maz2.drop();
db.maz2.insert(
[
  { x: "ab", A: [ {je:2, ke:3}, {je:2, ke:4} ] },
  { x: "aa", A: [ {je:2, ke:4}, {je:3, ke:4} ] },
  { x: "aa", A: [ {je:3, ke:4}, {je:4, ke:5} ] }
]
);
```

```
db.maz2.find( { 'A.1.ke': 4 } );
{ "_id" : ObjectId("...5e"), "x" : "ab", "A" : [ { "je" : 2,"ke" : 3 }, { "je" : 2, "ke" : 4 } ] }
{ "_id" : ObjectId("...5f"), "x" : "aa", "A" : [ { "je" : 2,"ke" : 4 }, { "je" : 3, "ke" : 4 } ] }
```

```
db.maz2.find( { 'A.ke': 4 } );
{ "_id" : ObjectId("...5e"), "x" : "ab", "A" : [ { "je" : 2,"ke" : 3 }, { "je" : 2, "ke" : 4 } ] }
{ "_id" : ObjectId("...5f"), "x" : "aa", "A" : [ { "je" : 2,"ke" : 4 }, { "je" : 3, "ke" : 4 } ] }
{ "_id" : ObjectId("...60"), "x" : "aa", "A" : [ { "je" : 3,"ke" : 4 }, { "je" : 4, "ke" : 5 } ] }
```

```
db.maz2.find( { 'A.ke': 4, "A.je": 4 } );
{ "_id" : ObjectId("...60"), "x" : "aa", "A" : [ { "je" : 3, "ke" : 4 }, { "je" : 4, "ke" : 5 } ] }
```

```
db.maz2.find( { A: { $elemMatch: { "je": 2, "ke": 4 } } } );
{ "_id" : ObjectId("...05e"), "x" : "ab", "A" : [ { "je" : 2, "ke" : 3 }, { "je" : 2, "ke" : 4 } ] }
{ "_id" : ObjectId("...05f"), "x" : "aa", "A" : [ { "je" : 2, "ke" : 4 }, { "je" : 3, "ke" : 4 } ] }
```

2a) Kurzor

```
var kurzor = db.maz2.find();
kurzor.forEach(function(e) {print(e.x, e.A);});  // ??
```

```
ab [object Object],[object Object]
aa [object Object],[object Object]
aa [object Object],[object Object]
```

JSON.stringify

```
var kurzor = db.maz2.find();
kurzor.forEach(function(e) {print(e.x, JSON.stringify(e.A));});
```

```
ab [{"je":2,"ke":3},{"je":2,"ke":4}]
aa [{"je":2,"ke":4},{"je":3,"ke":4}]
aa [{"je":3,"ke":4},{"je":4,"ke":5}]
```

```
var kurzor = db.maz2.find();
kurzor.forEach(function(e) {print(e.x, JSON.stringify(e.A[0]));});
```

```
ab {"je":2,"ke":3}
aa {"je":2,"ke":4}
aa {"je":3,"ke":4}
```

2b) Kurzor to array

```
var kurz = db.Autori.find(); kurz;                // zhustene
var kurz = db.Autori.find(); kurz.toArray();  // prehladne

var kurz = db.Autori.find();
kurz[1] ;  // <=>
kurz.toArray()[1] ;  // <=>
var dcs = kurz.toArray(); dcs[1];
```


```
{
        "_id" : ObjectId("5720bf3c0ca97bfc131af52b"),
        "meno" : "Imro",
        "adresa" : "AL",
        "dat_nar" : "2000",
        "knihy" : [
                {
                        "nazov" : "RDBS",
                        "zaner" : "PC",
                        "rok" : 2010,
                        "cena" : 45
                },
                {
                        "nazov" : "NoSQL",
                        "zaner" : "PC",
                        "rok" : 2011,
                        "cena" : 40
                }
        ]
}
```


a[1].A[1].je

```
//    db.maz2.find().toArray().je;   // nie
a = db.maz2.find().toArray(); a[1].A[1].je;
```

```
3
```

s využitím pomocnej kolekcie MAZ

```
a = db.maz2.find().toArray();   db.MAZ.insert(a[1].A[1]);   db.MAZ.find( {je:3} );
```


```
{ "_id" : ObjectId("570a22197a0780fe760ea78b"), "je" : 3, "ne" : 4 }
```


3) Generovanie kolekcie

```
function plusKdni(datum, k) {
   var d = new Date(datum);
   d.setDate(d.getDate( ) + k);
   return d;
};
```

```
function plusKrokov(datum, k) {
   var d = new Date(datum);
   d.setYear(d.getFullYear( ) + k);
   return d;
};

function randomAB(A, B) {
   return Math.floor( A+(Math.random( )*(B-A+1)) );
};
```

```
new Date(2016, 4-1, 4+1)
ISODate("2016-04-04T22:00:00Z")

plusKrokov(new Date(), 1);
ISODate("2017-04-18T20:32:55.191Z")

plusKdni(new Date(), 1);
ISODate("2016-04-19T20:30:09.020Z")

for ( i=1; i<=10; i++) { print(randomAB(10,20)) };
```

Vytvorme kolekciu studenti s 29-mi dokumentami

```
db.studenti.drop();
for (i=1; i<=29; i++){db.studenti.insert( { "i":i, "meno": "student" + randomAB(10,20) } ) };

db.studenti.find({meno:"student18"},{i:1, meno:1,_id:0});
{ "i" : 7, "meno" : "student18" }
{ "i" : 99, "meno" : "student18" }
```

Dopyt štandardne vráti iba prvých dvadsať dokumentov. Aby sme uvideli ďalšie dokumenty, systém po vykonaní limit dopytu nás upozorňuje, aby sme zadali it

```
db.studenti.find().limit(30);
```

4) Kurzory a JavaScript

```
var kur = db.studenti.find({},{i:1, meno:1,_id:0});
kur;
kur.count( );

var kur = db.studenti.find({},{i:1, meno:1,_id:0});
kur.limit(50);

var kur = db.studenti.find({},{i:1, meno:1,_id:0}); // kur
while(kur.hasNext()){
    var k = kur.next();
```

```
    //print(k.i,  k.meno);
    if(k.meno=="student18"){print(k.i, k.meno);}
};
```

```
7 student18
14 student18
28 student18
```

⇔

```
var kur = db.studenti.find({},{i:1, meno:1,_id:0}); // kur
var A= kur.toArray();
for( k = 0; k <  kur.count( ); k++){
        //print(A[k].i, A[k].meno);
       if( A[k].meno =="student18"){ print(A[k].i, A[k].meno);}
};
```

Pole - print(JSON.stringify(B));

```
var kur = db.maz2.find({}, {_id:0}); //kur;
var B= kur.toArray();
print( JSON.stringify(B) );
```

```
[{"x":"ab","A":[{"je":2,"ke":3},{"je":2,"ke":4}]},{"x":"aa","A":[{"je":2,"ke":4}
,{"je":3,"ke":4}]},{"x":"aa","A":[{"je":3,"ke":4},{"je":4,"ke":5}]}]
```

```
var kur = db.maz2.find();
while(kur.hasNext()){
    var k = kur.next();
    print(k.x, JSON.stringify(k.A[0]));
};
```

```
ab {"je":2,"ke":3}
aa {"je":2,"ke":4}
aa {"je":3,"ke":4}
```

```
var kur = db.maz2.find();
while(kur.hasNext()){
    var k = kur.next();
    print(k.x, JSON.stringify(k.A[0].je));
};
```

```
ab 2
aa 2
aa 3
```

```
var kur = db.maz2.find();
while(kur.hasNext()){
    var k = kur.next();
    print(k.x, JSON.stringify(k.A));
};
```

```
ab [{"je":2,"ke":3},{"je":2,"ke":4}]
aa [{"je":2,"ke":4},{"je":3,"ke":4}]
aa [{"je":3,"ke":4},{"je":4,"ke":5}]
```

```
var kur = db.maz2.find({}, {_id:0});
while(kur.hasNext()){
    var k = kur.next();
    var x;
    var s = "";
    for (x in k) {
      s += JSON.stringify(k[x])+",";
    }
    print(s);
};
```

```
"ab",[{"je":2,"ke":3},{"je":2,"ke":4}],
"aa",[{"je":2,"ke":4},{"je":3,"ke":4}],
"aa",[{"je":3,"ke":4},{"je":4,"ke":5}],
```

Optimalizácia dopytu

- indexy
- db.kol1.explain()