

2) CRUD, Kurzory

2.1) Sada replík a CRUD

I. Atomicita, transakcia, konzistencia a monotonicita

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

II. CRUD príkazy (kurzor: `forEach`, `toArray`, `next`)

- insert, update, delete

2.2) Dopytovanie

I. Dopytovanie bez polí a vnorených dokumentov

II. Dopytovanie polí a vnorených dokumentov

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 - `JSON.stringify`
- 2b) Kurzor `to array`

3) Generovanie kolekcie

4) Kurzory a JavaScript

- 4a) Pole – kur. `next()`

2.1) Sada replík a CRUD

Úvod do MongoDB CRUD

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

Mongod (Mongo Daemon) je základným procesom pre **správu** celého MongoDB servera (prijímanie požiadaviek, reagovanie na nich, riadenie požiadaviek na pamäť), ktorý beží v pozadí.

Mongo(sh) je JavaScript **rozhranie**, ktoré poskytuje interaktívnu komunikáciu s MongoDB, prijíma **užívateľské príkazy**, dopyty, spája sa s konkrétnou inštanciou `mongod` a potom príkazy spúšťa.

Sada replík a CRUD

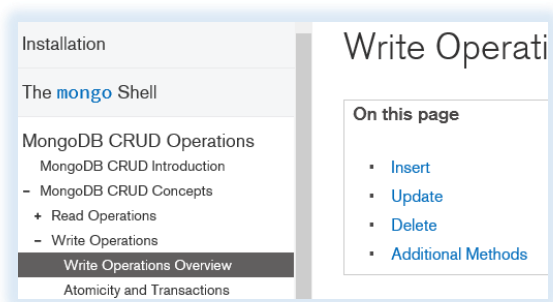
Sada replík (replica set) je skupina **inštancií** (členov) `mongod`, ktoré sa starajú o **rovnakú** sadu údajov. Ak v danom momente **primárny** člen sa funkčne zlyhá, zo **sekundárnych** členov sa zvolí nový primárny, pozri 4. prednášku. V sade replík sekundárni členovia môžu spracovať **čítanie**, ale **zápis iba primárny** člen.

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

- záznam u **primárneho** člena
- potom záznam u primárneho člena do **oplogu** (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 určiť poradie výsledku dopytu
 - Dopytovať v MongoDB okrem **find** môžeme aj pomocou **kurzora** (forEach) a **agregácie**
- Query príkazy
- Projekcie (atrib.)

I. Atomicita, transakcia, konzistencia a monotonicita

a) Atomicita – nevykonané čítanie

b) Dvojfázové vykonanie a sémantika podobná transakcie

c) Trvanlivosť a eventuálna konzistencia

d) Monotonicita zápisu a čítania

a) Atomicita – zápis a nevykonané čítanie

- **Atomickosť/atomicita** zápisu **jedného** dokumentu znamená, že ak **zápis** v dokumente aktualizuje **viac** kľúčov (atribútov), **častočné aktualizácie sa nesmú načítať**, čitateľ ich nemôže uvidieť. Aj keď čitateľ nemôže vidieť **častočne** aktualizovaný jediný dokument, konkurenční čitateľa za isté okolnosti predsa môžu **uvidieť aktualizovaný** dokument predtým, ako zmeny sú **trvanlivé** (copy). To sa nazýva **nevykonané čítanie** (read uncommitted – čítanie *nevykonaného* zápisu).
- Keď jediná operácia zápisu ovplyvňuje **viac** dokumentov, úprava **jednotlivých** dokumentov je atomická, ale operácia ako **celok nie je atomická** - ďalšie operácie môžu zasahovať, prelínať. Jedinú operáciu zápisu, ktorá ovplyvňuje viac dokumentov, je možné **izolovať** pomocou operátora \$isolated, avšak izolovaný zápis neposkytuje "všetko alebo nič" atomicitu.

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

b) Dvojfázové vykonanie a sémantika podobná transakcie – **oneskorená konzistencia**

Pretože jeden dokument často obsahuje **niekoľko vložených** dokumentov, **atomicita jedného** dokumentu je dostatočná pre veľa praktických prípadov. Pre **viac zápisov** naraz, 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). Preto **konzistencia** dát je zaručená **oneskorene**, lebo môže sa stať, že aplikácia 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**), ak bude pretrvávajúť 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 (pozri 4. prednášku).
 - pre skupinu replík, 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** - služobné slovo) **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 väčšinové (**majority**) alebo linearizable.

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 eventuálnu konzistenciu.

Eventuálna konzistencia (**eventual consistency**) je vlastnosť **distribúovaného** 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.

Poznamenáme, že 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) Monotonicitu čítania a zápisu.

Predpokladajme, že aplikácia vykoná postupnosť operácií, ktorá sa skladá

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

Potom 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.

Kým MongoDB zaručuje

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

II. CRUD príkazy (kurzor: **forEach**, **toArray**, **next**)

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

Dokumenty a kolekcie

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

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

Kurzor a dokumenty

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

```
use dbpred1;
db.tab1.drop();
db.kol1.drop(); //skontroluj: db.kol1.find();
db.dropDatabase(); // maze DB dbpred1 - //skontroluj: show dbs;

db.tab1.insertOne( { meno : "Fero", vaha : 82 } );
db.tab1.insertMany([ { meno : "Jano", vaha : 88 }, { meno : "Stevo", vaha : 88 } ] );
//db.tab1.copyTo("kol9"); // copy tab1 kolekciu to kol9

db.tab1.find().forEach( function(x) { db.kol1.insertOne(x); } ); // vytvori kol1
// ⇔ rychlejsie:
db.tab1.aggregate([ { $match: {} }, { $out: "kol2" } ])
db.kol1.find();
```

<https://www.mongodbmanager.com/clone-mongodb-collection/copyTo>

Dokumenty

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

Delete

- db.kol1.deleteOne()
- db.kol1.**deleteMany**() - **nemaž, kol1 sa používa**

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

update operátory

- kľúča
- poľa

• Dokumenty

Installation

The **mongo** Shell

MongoDB CRUD Operations

- MongoDB CRUD Introduction
- + MongoDB CRUD Concepts
- MongoDB CRUD Tutorials
- Insert Documents**
- Query Documents

Insert Documents

On this page

- [Insert a Document](#)
- [Insert an Array of Documents](#)
- [Insert Multiple Documents with Bulk](#)
- [Additional Examples and Methods](#)

Installation

The **mongo** Shell

MongoDB CRUD Operations

- MongoDB CRUD Introduction
- MongoDB CRUD Concepts
- + Read Operations
- Write Operations
- Write Operations Overview**
- Atomicity and Transactions

Write Operations

On this page

- [Insert](#)
- [Update](#)
- [Delete](#)
- [Additional Methods](#)

update, delete

```
db.uuu.drop();// odstrani celu kolekciu uuu, tu este nic
db.uuu.insert({ "_id" : 1, "vaha" : 70, vyska : 174 });
var ii = { "_id" : 2, "vaha" : 82, vyska : 175 };
db.uuu.insert(ii);
db.uuu.find();
```

update

```
db.uuu.updateOne( { vaha : 82 }, { $set: { vyska : 177 } } );
db.uuu.deleteOne({_id: 1}); db.uuu.find();
```

upsert – ak neexistuje pre update, potom insertuj

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

```
try {
  db.uuu.deleteMany ( { "vaha" : 74 } ); // odstrani dokumenty, splnujuce kriterium
} catch (e) {
  print(e);
}; db.uuu.find();
```

justOne : false – delete viac

```
db.uuu.insert({ "_id" : 4, "vaha" : 77, vyska : 177 });
try {
  db.uuu.deleteMany ( { vyska : 177 }, {justOne: false} );
} catch (e) {
  print(e);
}; db.uuu.find();
//db.uuu.find().toArray()
```

```
{ "_id" : 1, "vaha" : 70, "vyska" : 174 }
```

2.2) Dopytovanie (poradie!)

```
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 (atrib.).

```
{Vert.filt.} ⇔ {_id : 01, ..., klucJ : 1, klucK : 1},
```

Kde ak

- 01 sa rovná 1, potom sa hodnoty zodpovedajúceho kľúča vrátia
- 01 sa rovná 0, potom sa hodnoty zodpovedajúceho kľúča nevrátia

```
db.kol1.find( {}, {_id : 0, meno : 1, "vaha": 1 } ); // vrat iba kluce meno a vaha
```

The screenshot shows the MongoDB documentation page for `db.collection.update()`. The page title is `db.collection.update()`. The breadcrumb navigation is `Reference > mongo Shell Methods > Collection Methods`. The page content includes a table of contents with sections like `Reference`, `Operators`, `Database Commands`, `mongo Shell Methods`, and `Collection Methods`. The `Collection Methods` section lists `db.collection.aggregate()`, `db.collection.bulkWrite()`, and `db.collection.count()`. There is also a section titled `On this page` with links to `Definition`, `Behavior`, `Examples`, `WriteResult`, and `Additional Resources`.

Najprv sa **heslovite** pozremo na `{Horiz.filt}`

I. Dopytovanie **bez** 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 (`$or`)

II. Dopytovanie

C. **polí**

D. **vnorených** dokumentov

kde operátor `$oper`: hodnota môže byť napr.

`{ $gt :1, $lt :5, $in : [2, 3, 4] },`
Výrazový op.

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

`{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.koll.find( {vaha: 88, meno:"Jano"}, {_id : 0, meno : 1, "vaha": 1 } );  
{"meno" : "Jano", "vaha" : 88 }
```

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

```
db.koll.deleteOne({_id:4});  
db.koll.deleteOne({_id:5});  
db.koll.insert({_id:4, vaha:[1,2,3], dtm:"2016.9.9", meno:["Fero", "Jano"]});  
db.koll.insert({_id : 5, vaha : [1,2,3], dtm:"2017.9.9" });
```

Dopyt na hodnotu celého vektor-kľúča z

```
db.koll.find({vaha:[1,2,3], dtm: { $gt : "2017.9.8" } });  
{"_id" : 5, "vaha" : [ 1, 2, 3 ], "meno" : "2017.9.9" }
```

b) \$ dopytovanie

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

```
db.koll.find( {dtm: { $gt:"2015.9.9", $lt:"2018.9.9"},  
              meno:{$in: ["Fero", "Jano"] } } );  
{"_id" : 4, "vaha" : [ 1, 2, 3 ], "dtm" : "2016.9.9", "meno" : [ "Fero", "Jano" ] }
```

§ Operátory dopytovania a projektovania

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

B. Dopytovanie s vonkajším modifikátorom

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

```
db.koll1.find({$or:[{dtm: {$gt:"2015.9.9",$lt:"2018.9.9"}},
                    {meno:{$in:["Fero", "Jano"]}} ] } );
{ "_id" : ObjectId("5ad5a545452815f627de2b24"), "meno" : "Fero", "vaha" : 82 }
{ "_id" : ObjectId("5ad5a590452815f627de2b25"), "meno" : "Jano", "vaha" : 88 }
{ "_id" : 4, "vaha" : [ 1, 2, 3 ], "dtm" : "2016.9.9", "meno" : [ "Fero", "Jano" ] }
{ "_id" : 5, "vaha" : [ 1, 2, 3 ], "dtm" : "2017.9.9" }
```

II. Dopytovanie polí a polí vnorených dokumentov

https://www.tutorialsteacher.com/mongodb/documents?utm_content=cmp-true

```
use dbpred2;
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} ] }
  ]
); // Run on: https://www.mongodb.com/docs/v4.0/tutorial/query-array-of-documents/
```

Prvý prvok (nultý index) poľa A sa rovná 2:

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

```
db.maz2.findOne({}, { _id:0 }); // vrat bez _id
```

Dopyt

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

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

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 - **JSON.stringify**

2b) Kurzor **to array**

3) Generovanie kolekcie

4) Kurzory a JavaScript

4a) Pole – kur. **next()**

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  
  ]  
}
```

// ⇔ vyššie

```
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 } );  
{ "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

- Dopytujeme na cele pole

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

- Dopytujeme na hodnotu prvku

```
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 ] }
```

- Dopytujeme pomocou indexu

```
db.maz1.find( { 'A.1': 4 } ); // !!! 'A.1' druhy prvok A sa rovna 4; nutny apostrof  
{ "_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)}); // e je dokument/riadok  
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} ] }  
]  
);
```

- Dopytujeme pomocou indexu a kľúča

```
db.maz2.find( { 'A.1.ke': 4 }, {id:0} );  
[ { "x" : "ab", "A" : [ { "je" : 2, "ke" : 3 }, { "je" : 2, "ke" : 4 } ] },  
  { "x" : "aa", "A" : [ { "je" : 2, "ke" : 4 }, { "je" : 3, "ke" : 4 } ] } ]
```

- Dopytujeme iba pomocou kľúča

```
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 - JSON.stringify

```
var kurzor = db.maz2.find();
```

```
kurzor.forEach(function(e) { print(e.x, e.A); }); // ? uz sa to vyriesilo!
```

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

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

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

```
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 } ]
```

Riešenie: JSON.stringify

```
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, e.A[0]); });
```

```
//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

Kolekciu Autori sme vytvorili na prvej prednáške o MongoDB.

```
use knihy
```

```
var kurz = db.Autori.find(); kurz; // uz ⇔ (zhustene)
```

```
//var kurz = db.Autori.find(); kurz.toArray(); // (skor prehľadne)
```

```
var kurz = db.Autori.find();
```

```
//NO kurz[1];
```

```
kurz.toArray()[1]; // ⇔
```

```
var kurz = db.Autori.find();
```

```
var dcs = kurz.toArray(); dcs[1];
```

```
{
  _id: ObjectId("661e4656c31e80b5c9933a76"),
  meno: 'Imro',
  adresa: 'AL',
  dat_nar: '1990',
  knihy: [
    { nazov: 'RDBS', zaner: 'PC', rok: 2005, cena: 45 },
    { nazov: 'NoSQL', zaner: 'PC', rok: 2010, cena: 45 },
    { nazov: 'C#', zaner: 'PC', rok: 2016, cena: 50 }
  ]
}
```

a[1].A[1].je

```
use dbpred2
// 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, "ke" : 4 }
```

3) Generovanie kolekcie

Math.random() vráti pseudonáhodné číslo z intervalu [0; 1], teda ide o rovnomerné rozdelenie.

```
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<=5; i++) { print(randomAB(10,20)) }; // vypise 5 cel.cisel z intervalu [10;20]
```

Vytvoríme 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( ); // 29
```

```
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);}  
};
```

```
6 student18  
17 student18  
20 student18  
27 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);}  
};
```

4a) Pole – kur.**next**()

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

```
[{"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.koll.explain()