3) Agregácia, indexy

a) Agregačný framework

Agregačné zreťazenie (pipeline)

- **Etapové operátory**
- **Akumulátory** (agregačné funkcie)
- **Výrazové operátory**

Map-reduce

b) Indexy

- Základné koncepty
  - Jednoduchý kľúčový index
  - Kompozitný kľúč
  - Multy-key index
- Vytvorenie indexu

a) Agregačný framework

Agregačné reťazenie (pipeline)

Komplexné agregačné dopyty môžeme uskutočniť pomocou **agregačného zreťazenia** (**pipeline**), čo je framework pre agregáciu dát, modelovanú na základe zreťazeného spracovania dát.

MongoDB rozlišuje agregačné metódy a agregačné príkazy

Agregačné metódy	**Agregačné príkazy**
- db.kolekcia.aggregate()	aggregate
- db.kolekcia.group()	group
- db.kolekcia.mapReduce()	mapReduce
	count
	distinct

Nižšie sa budeme venovať agregačným metódam. Agregačné príkazy budeme ilustrovať príkladom na konci tejto časti.

Agregačná metóda `db.kolekcia.aggregate` poskytuje prístup k zreťazenému (aspoň 2) spracovaniu a vykoná agregáciu dát pomocou etáp (stages). Vstupným argumentom metódy je pole etáp, ktoré určujú dopytovacie kroky, vykonajú sa postupne a transformujú dokumenty do agregovaných výsledkov

```
db.kolekcia.aggregate( [ etapa1, etapa2, ... ] )
```

Našim cieľom je ilustrovať agregačné pojmy.

Pozrime sa predovšetkým na tri základné pojmy:

- etapové operátory
  - akumulátory (agregačné funkcie)
  - výrazové operátory

```
db.studenti2.aggregate( [
    ... ,
{ $match: { vaha: { $gte : 73 } } },
{ $group: { _id : "$vaha", "SumaVysok": { $sum : "$vyska" } } },
    ... ] )
```

Najbežnejšie **etapové operátory pre metódu** `db.kolekcia.aggregate(...)` sú

MongoDB		SQL
$project	pri vrátení dokumentu zmení jeho kľúče	SELECT
$match	štandardná MongoDB filtrácia	WHERE
$limit		
$skip		
$unwind	rozbitie poľa	
$group	na báze identifikujúceho a akumulovaného výrazu	GROUP BY
$sample	výber náhodného počtu dokumentov	
$sort		ORDER BY
$lookup	left outer join	JOIN
$out	ako posledné štádium, zapíše výsledok do kolekcie	

Etapy `$group` a `$project` môžu mať **akumulátory** (agregačné funkcie). Nižšie uvádzeme akumulátory pre etapu `$group`

$min	$sum	$push
$max	$avg	$addToSet
$first	$stdDevPop	
$last	$stdDevSamp	

Uvažujme kolekciu (funkcia randomAB bola definovaná na druhej prednáške) Run on: https://www.mongodb.com/docs/v4.0/tutorial/query-array-of-documents/

```
use dbmaz;
function randomAB(A, B) {
    return Math.floor( A+(Math.random( )*(B-A+1)) );
};
db.studenti.drop();
for (i=1; i<=29; i++){db.studenti.insert( {
   "i":i,
   "meno": "student"+randomAB(10,15),
   vaha:70+randomAB(0,5),
   vyska:170+randomAB(2,10) } ) }
```

```
db.studenti.count();   // 29
db.studenti.find({},{_id:0}).limit(10)
```

Výpočet priemeru bez grupovania

```
db.studenti.aggregate(
 [ { $group: { _id : null, "PriemerVah": { $avg: "$vaha"  } } } ]);
```

```
{ "_id" : null, "PriemerVah" : 72.48275862068965 }
```

Poznamenáme, že _id tu nie je štanandardný názov kľúča dokumentu, ale služobné slovo, určujúce kľúč, podľa ktorého sa grupuje.

```
db.studenti.aggregate(
 [ { $group: { _id : "$vaha", "PriemerVah": { $avg: "$vaha"  } } } ]);
```

```
[
  { _id: 74, PriemerVah: 74 },
  { _id: 71, PriemerVah: 71 },
  { _id: 75, PriemerVah: 75 },
  { _id: 70, PriemerVah: 70 },
  { _id: 73, PriemerVah: 73 },
  { _id: 72, PriemerVah: 72 }
]
```

Grupovanie s počtom pomocou $sum: 1 (za každý dokument 1 ⇔ počet dokumentov krát 1)

```
db.studenti.aggregate(
   [
     { $group: { _id : "$vaha", "Pocet": { $sum: 1 } } }
   ]
);
```

⇔ (v jednom riadku)

```
db.studenti.aggregate([{$group:{_id : "$vaha", "Pocet": { $sum: 1}}}]);
```

```
[
  { _id: 73, Pocet: 6 },
  { _id: 70, Pocet: 5 },
  { _id: 71, Pocet: 6 },
  { _id: 75, Pocet: 4 },
  { _id: 74, Pocet: 4 },
  { _id: 72, Pocet: 4 }
]
```

Zreťazenie s dvomi etapami - grupované počty pomocou dvoch kľúčov s usporiadaním podľa počtu

```
db.studenti.aggregate(   [
  { $group: { _id : "$vaha", "Pocet": {
$sum: 1 } } },
 {$sort:{Pocet:1}}  ]);
```

```
[
  { _id: 75, Pocet: 4 },
  { _id: 74, Pocet: 4 },
  { _id: 72, Pocet: 4 },
  { _id: 70, Pocet: 5 },
  { _id: 73, Pocet: 6 },
  { _id: 71, Pocet: 6 }
]
```

```
db.studenti.aggregate(   [
  { $group: { _id : "$vyska", "Pocet": {
$sum: 1 } } },
 {$sort:{Pocet:1}}  ]);
```

```
[
  { _id: 178, Pocet: 1 },
  { _id: 175, Pocet: 1 },
  { _id: 176, Pocet: 2 },
  { _id: 174, Pocet: 2 },
  { _id: 180, Pocet: 4 },
  { _id: 173, Pocet: 4 },
  { _id: 172, Pocet: 4 },
  { _id: 177, Pocet: 5 },
  { _id: 179, Pocet: 6 }
]
```

Grupované súčty s usporiadaním podľa sumy výšok

```
db.studenti.aggregate(   [
{ $group: { _id : "$vaha", "SumaVysok": { $sum: "$vyska" } } } ,
{ $sort: { "SumaVysok" : 1 } }  ]);
```

```
[
  { _id: 72, SumaVysok: 699 },
  { _id: 74, SumaVysok: 704 },
  { _id: 75, SumaVysok: 709 },
  { _id: 70, SumaVysok: 894 },
  { _id: 71, SumaVysok: 1049 },
  { _id: 73, SumaVysok: 1057 }
]
```

[Ďalšie príklady]

Výrazové operátory

Typ	Operátor		Typ	Operátor

Množinový	$setEquals $setIntersection $setUnion $setDifference $setIsSubset $anyElementTrue $allElementsTrue		**Reťazcový**	$concat $substr $toLower $toUpper $strcasecmp
Porovnávací	$cmp $eq, $ne $gt $gte $lt $lte		**Pole**	$arrayElemAt $concatArrays $filter $isArray $size $slice
Boolean	$and, $or, $not		**Podmienený**	$cond, $ifNull
Arritmetický	$abs, $add, $ceil, $divide $exp, $floor, $ln, $log, $log10 $mod, $multiply, $pow, $sqrt $subtract, $trunc		**Dátumový**	$dayOfYear $dayOfMonth $dayOfWeek $year, $month, $week $hour, $minute, $second $millisecond $dateToString

Za filtráciou s výrazovým operátorom $gte nasleduje zreťazenie - grupovaný súčet a usporiadanie

```
db.studenti.aggregate([
    { $match: { vaha: {$gte:73} } },
    { $group: { _id : "$vaha", "SumaVysok": { $sum: "$vyska" } } },
    { $sort: { "SumaVysok": -1 } }   ])
```

```
[
  { _id: 73, SumaVysok: 1057 },
  { _id: 75, SumaVysok: 709 },
  { _id: 74, SumaVysok: 704 }
]
```

Uvažujme kolekciu studenti2 s dátumom (funkcia plusKdni bola definovaná na druhej prednáške)

```
db.studenti2.drop();
for (i=1; i<=10; i++){db.studenti2.insert( {
    "i":i,
    "meno": "student"+randomAB(10,15),
    "datNarod" : plusKdni(new Date(), -randomAB(1000,10000)),
    vaha:70+randomAB(0,5),
    vyska:170+randomAB(2,10)
} ) };

db.studenti2.findOne();
```

```
{
        "_id" : ObjectId("5729c5c60ca97bfc131af699"),
        "i" : 1,
        "meno" : "student14",
        "datNarod" : ISODate("1998-06-13T09:49:58.754Z"),
        "vaha" : 70,
        "vyska" : 178
}
```

// Prehľadný výpis

```
db.studenti2.find().pretty();
```

Etapový operátor `$project` s výrazovým operátorom `$year`

```
db.studenti2.aggregate([
      { $match: { vaha: {$lte:73} }  },
      { $project:{meno:1, rokNar: { $year: "$datNarod" }, _id:0, vaha:1,
vyska:1}}  ] );
```

```
[
  { meno: 'student11', vaha: 72, vyska: 174, rokNar: 2011 },
  { meno: 'student15', vaha: 70, vyska: 180, rokNar: 1999 },
  { meno: 'student11', vaha: 72, vyska: 177, rokNar: 1997 },
  { meno: 'student10', vaha: 70, vyska: 176, rokNar: 1997 },
  { meno: 'student10', vaha: 71, vyska: 172, rokNar: 2003 },
  { meno: 'student11', vaha: 70, vyska: 177, rokNar: 2005 }
]
```

Etapový operátor `$project` s výrazovým operátorom `$year a usporiadaním`

```
db.studenti2.aggregate([
      { $match: { vaha: {$lte:73} }  },
      { $project:{meno:1, rokNar: { $year: "$datNarod" }, _id:0, vaha:1,
vyska:1}},
      { $sort:  { "rokNar": -1 }  }     ] );
```

```
[
  { meno: 'student11', vaha: 72, vyska: 174, rokNar: 2011 },
  ...
  { meno: 'student10', vaha: 70, vyska: 176, rokNar: 1997 }
]
```

```
db.studenti2.aggregate([
        { $match: { vaha: {$lte:73} } },
        { $group: { _id : "$vaha", "SumaVysok": { $sum: "$vyska" } }  },
        { $sort: { "SumaVysok": -1 } } ]);
```

```
[
  { _id: 70, SumaVysok: 533 },
  { _id: 72, SumaVysok: 351 },
  { _id: 71, SumaVysok: 172 }
]
```

Agregačné príkazy

- `aggregate`
- `group` grupuje a agreguje podľa kľúča
- `mapReduce` agreguje veľké dátasety
- `count` počet dokumentov podľa kľúča
- `distinct` jedinečné dokumenty podľa kľúča

Map-reduce

Pre väčšinu agregačných operácií *Agregačné zreťazenie* poskytuje lepší výkon, avšak je limitované s implementovanými operátormi. *Map-reduce* zabezpečuje väčšiu pružnosť (viac možností) vďaka JavaScriptu a môže pracovať s obrovskými dátasetmi, ale vo všeobecnosti je menej výkonný. Od verzie 5 sa neodporúča.

Fázy *map-reduce*

Skrátená syntax agregačného príkazu mapReduce:

```
db.collection.mapReduce(
```

```
        function() {emit(key,value);},                  //map funkcia
        function(key,values) {return reduceFunction;},  //reduce funkcia
        {       out: kolekcia,
                query: dokument,
                sort: dokument,
                limit: pocet
        }
)
```

Funkcia *map* priradí *kľúč-hodnotu* pár každému dokumentu, ktorý vyhovuje podmienke dopytu. Pre hodnoty kľúča (atribútu), MongoDB aplikuje *reduce* fázu, ktorá zbiera a zhusťuje súhrnné údaje do polí.

Funkcia `emit(atrib1, atrib2)` vráti toľko *polí* s hodnotami atrib2, koľko rôznych hodnôt má atribut1, teda sa grupuje podla atrib1 a vratia sa *polia* hodnôt atrib2.

Map-reduce prebieha v **štyroch krokoch**, fázach. 3. krok spojí 1., 2. a 4.:

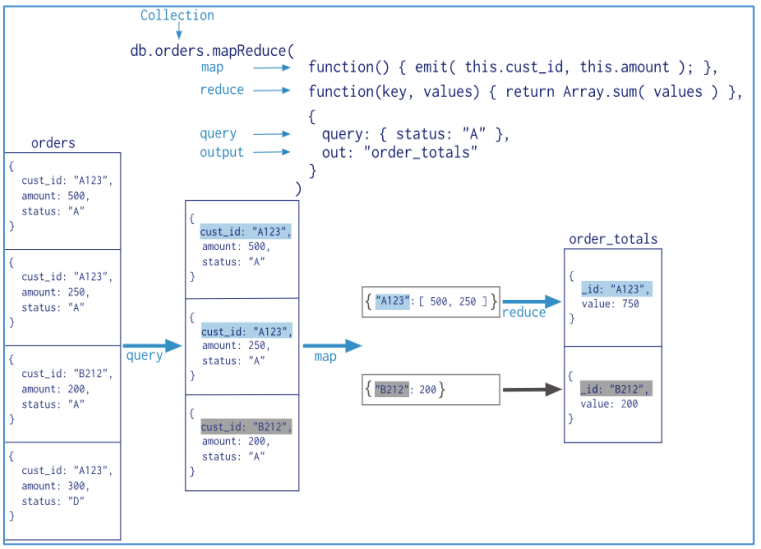
```
db.studenti2.mapReduce(map1, reduce1, {"out" : "haha"});
```

Pr.1 Zistime, že jednotlivé kľúče sa koľkokrát nachádzajú v kolekcii, teda v koľkých dokumentoch.

Definujme map a reduce funkcie (**1. a 2. krok**):

```
map1 = function() {
        for (var kluc1 in this) {
         emit(kluc1, {pocet : 1});
        }
};
```

```
red1 = function(kluc1, aa) {
        total = 0;
        for (var i in aa) {
             total += aa[i].pocet;
        }
        return {"pocet" : total};
};
```



Ilustračný príklad https://docs.mongodb.com/manual/core/map-reduce/

Možné testovanie metódy `red1` s polom dokumentov:

```
r1 = red1("k", [{pocet : 1, hoci : 1}, {pocet : 1, hoci : 2}]); r1
{ "pocet" : 2 }
```

```
r2 = red1("k", [{pocet : 3, hoci : 3}]); r2
{ "pocet" : 3 }
```

```
red1("k", [r1, r2])
{ "pocet" : 5 }
```

3.krok

```
db.studenti2.mapReduce(map1, red1, {"out" : "haha"});
{ result: 'haha', ok: 1 }
```

Starting in MongoDB 5.0, map-reduce is deprecated: https://docs.mongodb.com/manual/core/map-reduce

4.krok

```
db.haha.find();
[
  { _id: 'vaha', value: { pocet: 10 } },
  { _id: 'meno', value: { pocet: 10 } },
  { _id: '_id', value: { pocet: 10 } },
  { _id: 'i', value: { pocet: 10 } },
```

```
  { _id: 'vyska', value: { pocet: 10 } },
  { _id: 'datNarod', value: { pocet: 10 } }
]
```

⇔

```
db.runCommand({"mapreduce" : "studenti2", "map" : map1, "reduce" : red1,
"out" : "haha"});
db.haha.find();
```

Možná zmena kolekcie `studenti2` pre mapReduce (2 --> 0):

```
db.studenti2.count({vaha:74}) ;
db.studenti2.remove( { vaha : 74}, {justOne: false} );
db.studenti2.count({vaha:74});
```

Pozri príklady v https://docs.mongodb.com/manual/tutorial/map-reduce-examples/

b) Indexy

- Základné koncepty
  - Jednoduchý kľúčový index
  - Kompozitný kľúč
  - Multy-key index - pre pole
- Vytvorenie indexu

Bez indexov databáza skenuje, prezrie každý dokument kolekcie, aby zistila ktoré dokumenty vyhovejú dopytovacím podmienkam.

Vytvor kolekciu s dvadsaťtisíc dokumentom

```
use dbindexy;
db.dropDatabase();
//db.testIndex2.dropIndex({"meno":1});
//db.testIndex2.drop();
// ~35 sec.
for(i=0;i<20000;i++){db.testIndex2.insert( {"meno":"osoba"+i} )};
db.testIndex2.count();
```

Skenovaná filtrácia bez indexu trvá zhruba 14-20 msec (viacnásobné vykonanie)

```
db.testIndex2.find({"meno":"osoba101"}).explain("allPlansExecution").executio
nStats.executionTimeMillis;
```

15

```
db.testIndex2.find({"meno":"osoba101"}).explain("executionStats").executionSt
ats.executionTimeMillis;
```

20

```
// db.testIndex.find({"username" : "user101"}).explain();
```

Pre kľúč _id sa automaticky vytvorí index, ale ten nepomôže pri filtrácii kľúča `meno`.

```
db.testIndex2.getIndexes();
```

Po vytvorení indexu pre meno sa čas filtrácie redukuje na nulu (viacnásobné vykonanie).

```
db.testIndex2.ensureIndex({"meno":1});
db.testIndex2.find({"meno":"osoba101"}).explain("executionStats").executionStats.executionTimeMillis;
```

0

```
db.testIndex2.getIndexes();
```

```
//db.testIndex2.dropIndexes();
[
  { v: 2, key: { _id: 1 }, name: '_id_' },
  { v: 2, key: { meno: 1 }, name: 'meno_1' }
]
```

Old:

```
[
	{
		"v" : 1,
		"key" : {
			"_id" : 1
		},
		"name" : "_id_",
		"ns" : "dbMaz.testIndex2"
	},
	{
		"v" : 1,
		"key" : {
			"Name" : 1
		},
		"name" : "Name_1",
		"ns" : "dbMaz.testIndex2"
	}
]
```

Nasledujúci kód vráti indexy pre každú **kolekciu** danej databázy.

```
for(i=0; i<1000; i++) {db.testIndex1.insert( {"meno":"osoba"+i} )};
db.testIndex1.count();

db.getCollectionNames().forEach(function(x) {
  i = db[x].getIndexes();
  print("Indexes for " + x + ":");
  printjson(i);
});
```

```
Indexes for testIndex1:
[ { v: 2, key: { _id: 1 }, name: '_id_' } ]
Indexes for testIndex2:
[
  { v: 2, key: { _id: 1 }, name: '_id_' },
  { v: 2, key: { meno: 1 }, name: 'meno_1' }
]
```

Každý typ indexu je realizovaný ako B-strom. Každá update operácia využíva iba jeden index, o výbere ktorého sa rozhoduje optimalizátor.