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 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

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

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

výrazový op.

akumulátor

etapový op.

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 BV
$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 poslednej prednáške)

```
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(30)
```

Výpočet priemeru bez grupovania

```
db.studenti.aggregate(   [ { $group: { _id : null, "PriemerVah": { $avg: "$vaha"  } } } ]);
{ "_id" : null, "PriemerVah" : 72.48275862068965 }
```

Tu ⇔

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

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

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

```
db.studenti.aggregate(
  [
    { $group: { _id : "$vaha", "Pocet": { $sum: 1 } } }
  ]
);
⇔
db.studenti.aggregate(  [ { $group: { _id : "$vaha", "Pocet": { $sum: 1 } } } ]);
{ "_id" : 71, "Pocet" : 5 }
{ "_id" : 75, "Pocet" : 4 }
{ "_id" : 70, "Pocet" : 2 }
{ "_id" : 72, "Pocet" : 9 }
{ "_id" : 74, "Pocet" : 2 }
{ "_id" : 73, "Pocet" : 7 }
```

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" : 70, "Pocet" : 4 }
{ "_id" : 73, "Pocet" : 4 }
{ "_id" : 71, "Pocet" : 5 }
{ "_id" : 72, "Pocet" : 8 }
```

```
db.studenti.aggregate(  [
  { $group: { _id : "$vyska", "Pocet": { $sum: 1 } } },
  { $sort: { Pocet: 1 } } ]);
{ "_id" : 174, "Pocet" : 1 }
{ "_id" : 177, "Pocet" : 2 }
{ "_id" : 172, "Pocet" : 2 }
{ "_id" : 176, "Pocet" : 3 }
```

Grupované súčty s usporiadaním podľa sumy výšok

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

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 grupovaný súčet a usporiadanie

```
db.studenti.aggregate([
      { $match: { vaha: {$gte:73} } },
      { $group: { _id : "$vaha", "SumaVysok": { $sum: "$vyska" } } },
      { $sort: { "SumaVysok": -1 } }    ] )
{ "_id" : 75, "SumaVysok" : 2654 }
{ "_id" : 74, "SumaVysok" : 2291 }
{ "_id" : 73, "SumaVysok" : 1930 }
```

Uvažujme kolekciu studenti2 s dátumom (funkcia plusKdni bola definovaná na poslednej 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" : "student14", "vaha" : 70, "vyska" : 178, "rokNar" : 1998 }
{ "meno" : "student12", "vaha" : 72, "vyska" : 177, "rokNar" : 1998 }
{ "meno" : "student12", "vaha" : 72, "vyska" : 179, "rokNar" : 1991 }
{ "meno" : "student12", "vaha" : 71, "vyska" : 176, "rokNar" : 1994 }
{ "meno" : "student14", "vaha" : 72, "vyska" : 176, "rokNar" : 1994 }
{ "meno" : "student10", "vaha" : 70, "vyska" : 180, "rokNar" : 1996 }
```

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" : "student14", "vaha" : 70, "vyska" : 178, "rokNar" : 1998 }
...
{ "meno" : "student12", "vaha" : 72, "vyska" : 179, "rokNar" : 1991 }
```

```
db.studenti2.aggregate([
      { $match: { vaha: {$lte:73} }  },
      { $group: { _id : "$vaha", "SumaVysok": { $sum: "$vyska" } }  },
      { $sort: { "SumaVysok": -1 }  }    ] );
{ "_id" : 72, "SumaVysok" : 532 }
{ "_id" : 70, "SumaVysok" : 358 }
{ "_id" : 71, "SumaVysok" : 176 }
```

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 vďaka JavaScriptu a môže pracovať s obrovskými dátasetmi, ale vo všeobecnosti je menej výkonný.

Fázy *map* a *reduce*

Funkcia *map* priradí *kľúč-hodnotu* pár každému dokumentu, ktorý vyhovuje podmienke dopytu. Pre tie kľúče, ktoré majú viac hodnôt, MongoDB aplikuje *reduce* fázu, ktorá zbiera a zhusťuje súhrnné údaje.

Skrátená syntax 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
        }
)
```

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

Definujme map a reduce:

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

reduce1 = function(kluc1, aa) {
	total = 0;
	for (var i in aa) {
		total += aa[i].pocet;
	}
	return {"pocet" : total};
};
```

Testujeme reduce1.

```
r1 = reduce1("k", [{pocet : 1, hoci : 1}, {pocet : 1, hoci : 2}])
"pocet" : 2 }

r2 = reduce1("k", [{pocet : 3, hoci : 3}])
{ "pocet" : 1 }

reduce1("k", [r1, r2])
{ "pocet" : 5 }


db. studenti2.count({vaha:70}) ;
db. studenti2.remove( { vaha : 70}, {justOne: false} );
db. studenti2.count({vaha:70});


db.studenti2.mapReduce(map1, reduce1, {"out" : "haha"});
{
	"result" : "haha",
	"timeMillis" : 9,
	"counts" : {
		"input" : 8,
		"emit" : 48,
		"reduce" : 6,
		"output" : 6
	},
	"ok" : 1
}

db.haha.find();
{ "_id" : "_id", "value" : { "pocet" : 8 } }
{ "_id" : "datNarod", "value" : { "pocet" : 8 }
{ "_id" : "i", "value" : { " pocet " : 8 } }
{ "_id" : "meno", "value" : { "pocet" : 8 } }
{ "_id" : "vaha", "value" : { "pocet" : 8 } }
{ "_id" : "vyska", "value" : { "pocet" : 8 } }
⇔
db.runCommand({"mapreduce" : "studenti2", "map" : map1, "reduce" : reduce1, "out" : "haha"});
db.haha.find();
```

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

```
db.testIndex2.dropIndex({"meno":1});
db.testIndex2.drop();
for(i=0;i<20000;i++){db.testIndex2.insert( {"meno":"osoba"+i} )};
db.testIndex2.count();
```

Skenovaná filtrácia bez indexu trvá zhruba 20 msec

```
db.testIndex2.find({"meno":"osoba101"}).explain("allPlansExecution").executionStats.executionTimeMillis;
21
⇔
db.testIndex2.find({"meno":"osoba101"}).explain("executionStats").executionStats.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.

```
db.testIndex2.ensureIndex({"meno":1});
db.testIndex2.find({"meno":"osoba101"}).explain("executionStats").executionStats.executionTimeMillis;
0

db.testIndex2.getIndexes();
//db.testIndex2.dropIndexes();
[
        {
                "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.

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

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