Dokumentowe bazy danych MongoDB

Bazy danych

Agenda

- NoSQL
- Dokumentowe bazy danych
- MongoDB – podstawowe operacje
- Java API do MongoDB
- MongoDB – zaawansowane koncepcje ??

Literatura

- P. J. Sadalage, M. Fowler, NoSQL Distilled: A Brief Guide to the Emerging World of Polyglot Persistence, …, 2013
  - NoSQL Kompedium wiedzy – Helion 2014
- K.Banker, MongoDB in Action, Manning 2012
- E.Redmond, J.Wilson, Seven Databases in Seven Weeks, The Pragmatic Bookshelf 2012
- K.Chodorow, MongoDB: The Definitive Guide, O'Reilly 2013
- R.Copeland, MongoDB: Applied Design Patterns, O'Reilly 2013
- K.Seguin, The Little MongoDB Book (available online: http://openmymind.net/mongodb.pdf)
- K.Chodorow, 50 Tips & Tricks for MongoDB Developers, O'Reilly 2011
- K.Chodorow, Scaling MongoDB, O'Reilly 2011

MongoDB

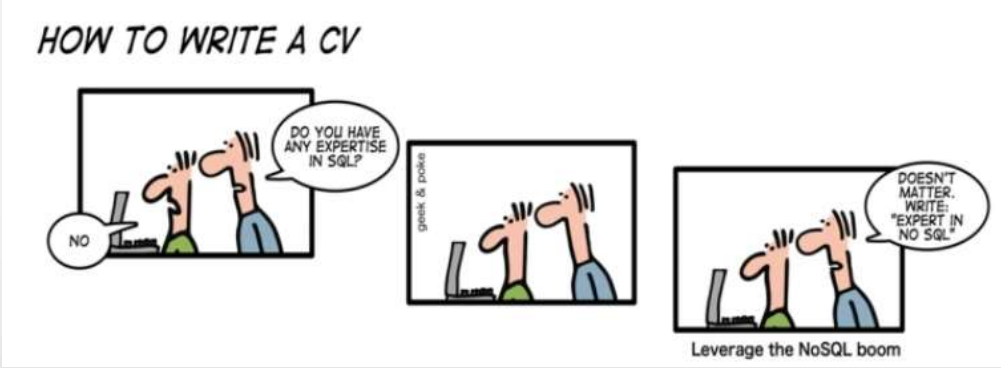
- dokumentowa baza danych (NoSQL)
- bardzo wydajna (implementacja w C++)
- ma wsparcie dla wielu języków programowania (m.in. C, C++, C#, Java, Python, Ruby, Scala…)

https://db-engines.com/

MongoDB – raport Gartner 2015

NoSQL

- NoSQL = Not Only SQL
- Klasa systemów DB przechowujących dane w nierelacyjny sposób
- Zwykle nie wymagają definiowania schematu DB
- Właściwości ACID zwykle są mniej restrykcyjnie przestrzegane



http://www.bigdatafarm.com/wp-content/uploads/2013/03/NoSql_joke11.png

Skalowalność

- wertykalna (scale up)
  - zwiększenie zasobów na danym węźle np. zwiększenie pamięci
- horyzontalna (scale out)
  - dodanie kolejnych węzłów do systemu (system rozproszony)

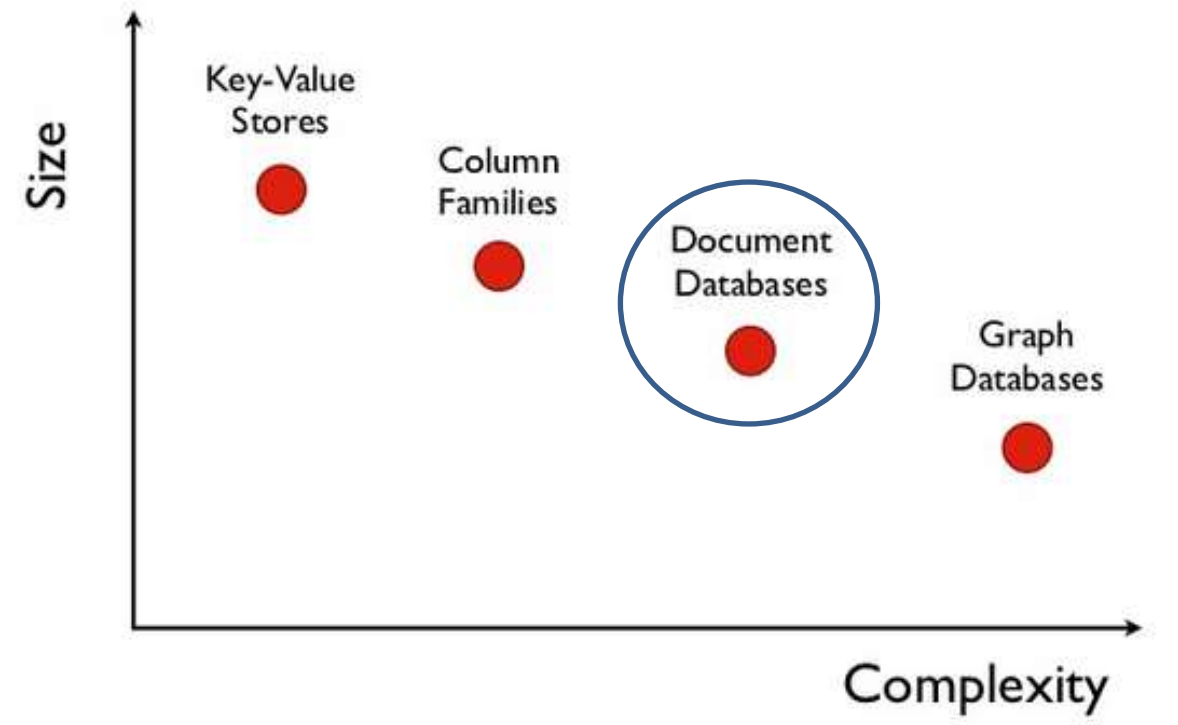
Dlaczego NoSQL?

- nowe zastosowania
- np. eksplozja sieci społecznościowych (Facebook, Twitter) -> potrzeba operowania na b. dużej ilości danych
- rozwój technologii cloud-based (cloud computing, cloud storage -> Amazon S3)
- wraz z większym wykorzystaniem języków dynamicznych (Ruby, Python, Groovy) zaczęto częściej używać dynamicznie typowanych danych z częstymi zmianami schematu

Bazy NoSQL

- Zalety
  - nie wymagają schematu
  - łatwiej się skalują
  - wiele implementacji open source
- Wady
  - nowe i często jeszcze niedopracowane narzędzia
  - dane często duplikowane, podatność na niespójności
  - zwykle brak standardu do tworzenia zapytań
  - często trudno zamodelować złożone struktury

Rodzaje baz NoSQL



http://www.slideshare.net/emileifrem/nosql-east-a-nosql-overview-and-the-benefits-of-graph-databases

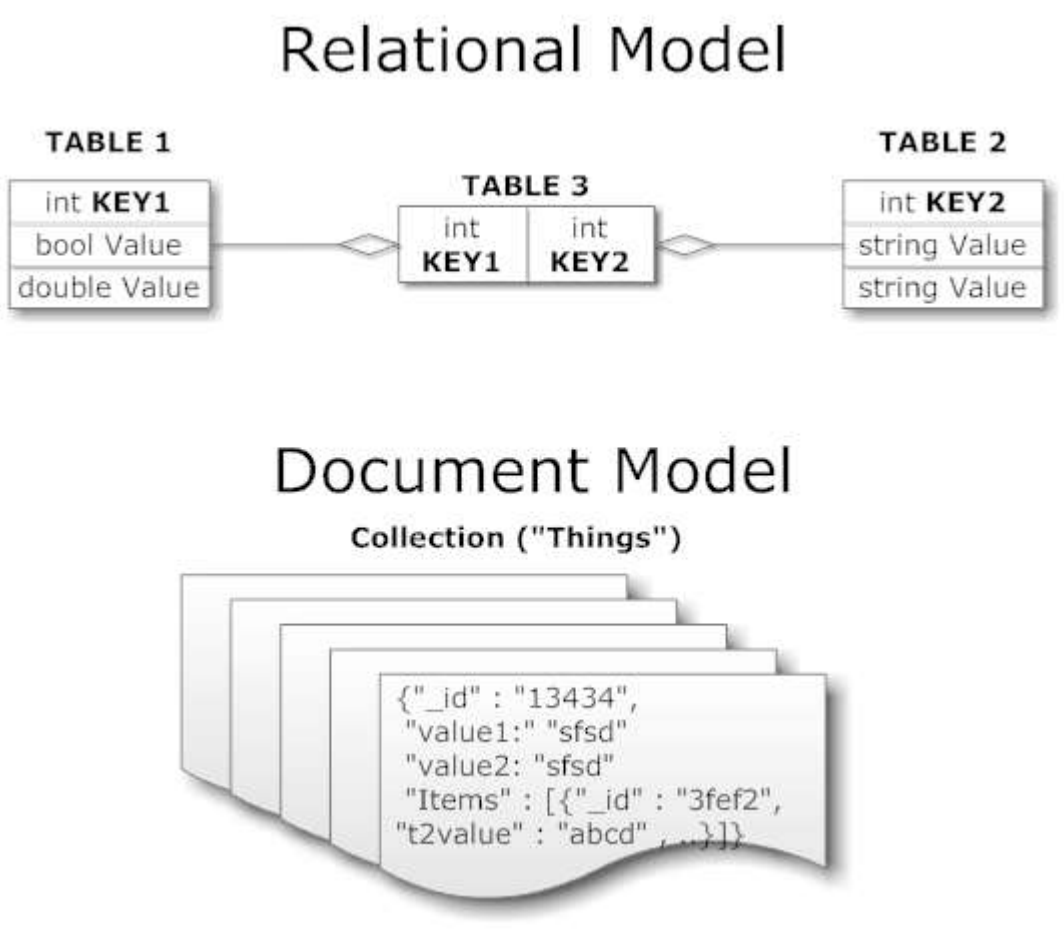
Bazy dokumentowe

- key-document store
- schema-less
- wsparcie dla map reduce
- przykłady
  - MongoDB
  - CouchDB
  - PostgreSQL ??

Model danych

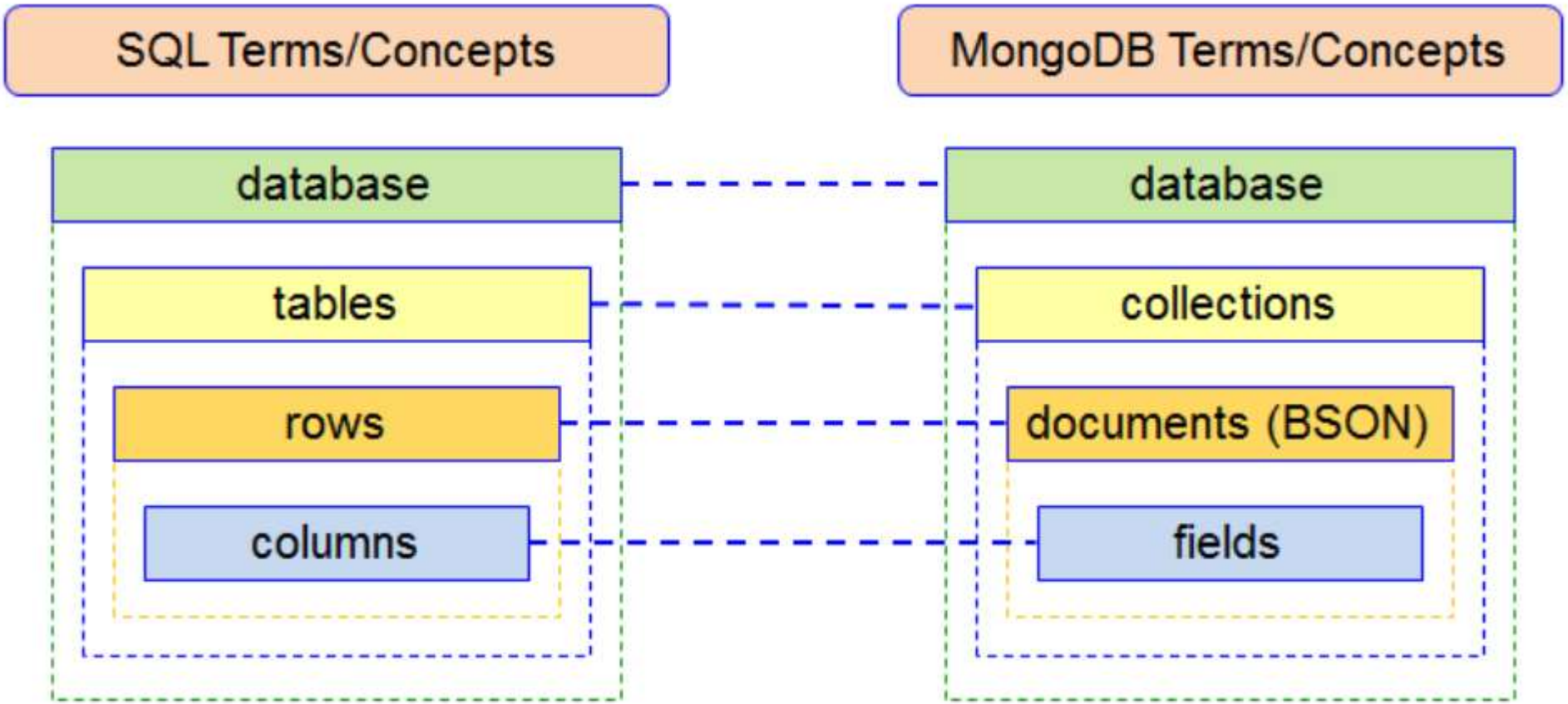
- Pojedyncza instalacja MongoDB może zawierać wiele baz danych
- baza danych zawiera zestaw kolekcji
- kolekcja składa się z dokumentów
- dokument jest zbiorem par klucz-wartość

Model dokumentowy vs model relacyjny



http://www.aaronstannard.com/image.axd?picture=mongo%20vs%20sql%20differences_thumb.png

RDBMS vs MongoDB 1



http://domaintutor.info/invitaciones/sql-vs-mongo-query/

RDBMS vs MongoDB 2

RDBMS	MongoDB
Table	Collection
Row	JSON Document
Index	Index
Join	Embedding & Linking
Partition	Shard

http://sqldev.files.wordpress.com/2011/08/mongo.png

Dokument

- dokumenty w formie JSON (JavaScript Object Notation)
- obiekt – nieuporządkowany zbiór par klucz-wartość

```
{
        name: „Jan Kowalski”
}
```

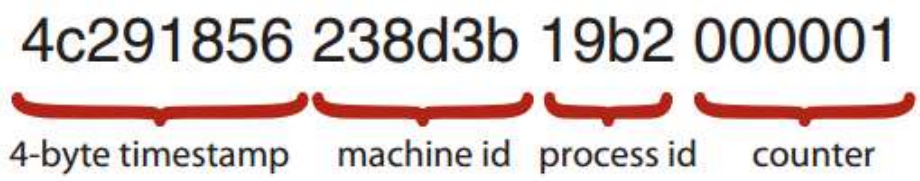
- obiekt może być użyty jako złożona wartość w parze klucz-wartość

```
user1 = {
        name: „Jan Kowalski”,
        age: 35,
        address: {
                street: „Mickiewicza 9”,
                city: „Krakow”
        }
}
```

Id dokumentu

- każdy dokument posiada atrybut _id
- można podać wartość do tego pola
- jeśli nie podana wartość przy wstawianiu dokumentu, zostaje automatycznie ustawiona unikalna wartość typu ObjectId
- musi być unikalne
- format domyślnego id

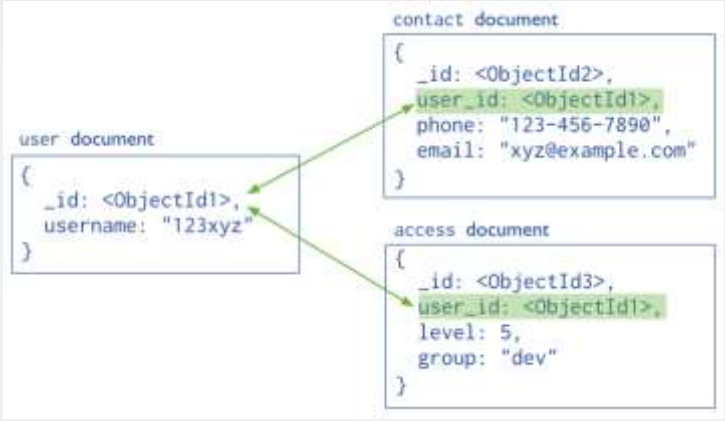
4-byte value representing the seconds since the Unix epoch,
3-byte machine identifier,
2-byte process id
3-byte counter, starting with a random value.



rys z MongoDB in Action

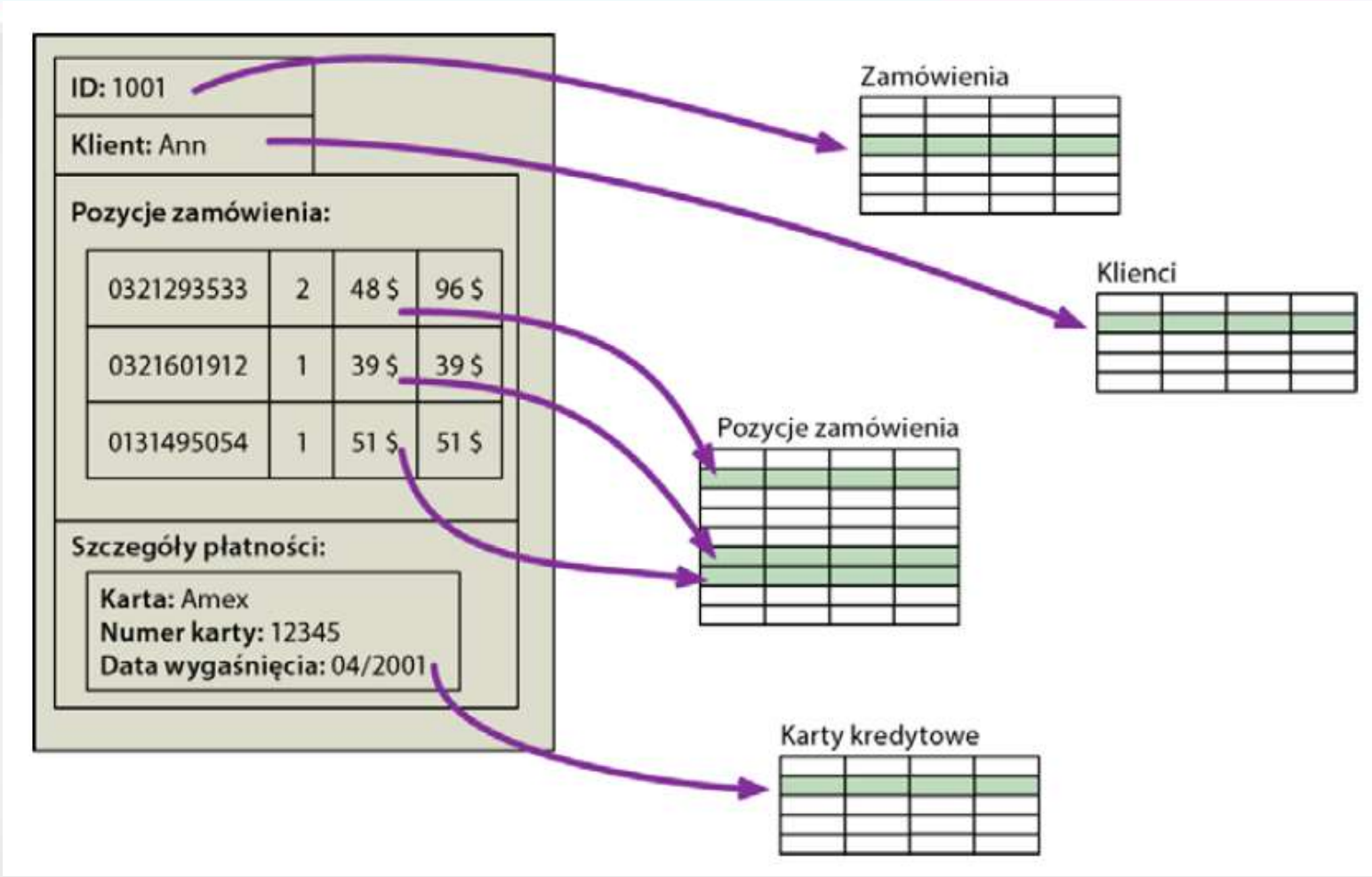
Modelowanie danych 1

References vs Embedded documents



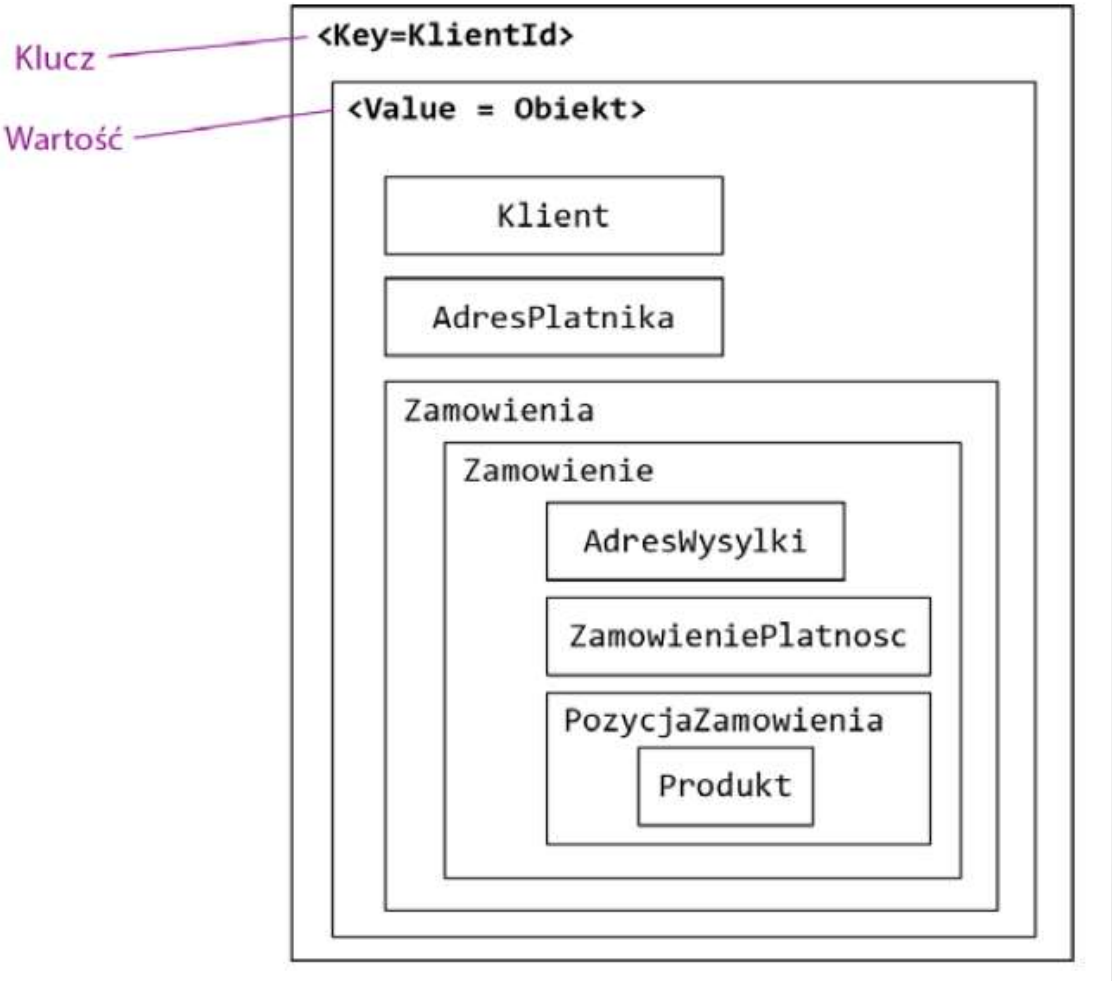
- do rozważenia:
  - atomowość operacji zapisu (poziom pojedynczego dokumentu)
  - rozrost dokumentu (może prowadzić do relokacji na dysku i fragmentacji)
  - typowe przypadki użycia
  - względy wydajnościowe
  - aspekty normalizacji i redundancji danych

Modelowanie danych 2



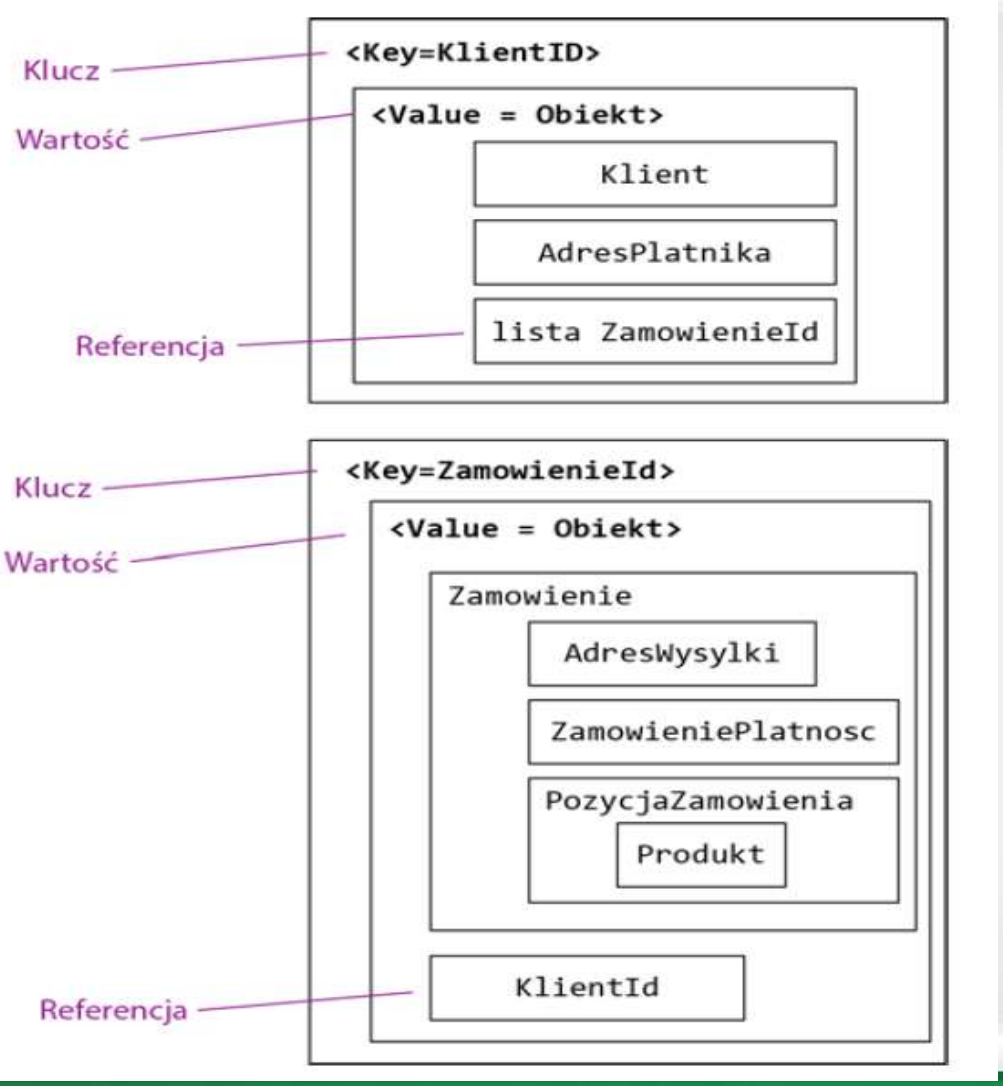
rys z NoSQL Kompedium wiedzy

Modelowanie z myślą o dostępie do danych 1



rys z NoSQL Kompedium wiedzy

Modelowanie z myślą o dostępie do danych 2



rys z NoSQL Kompedium wiedzy

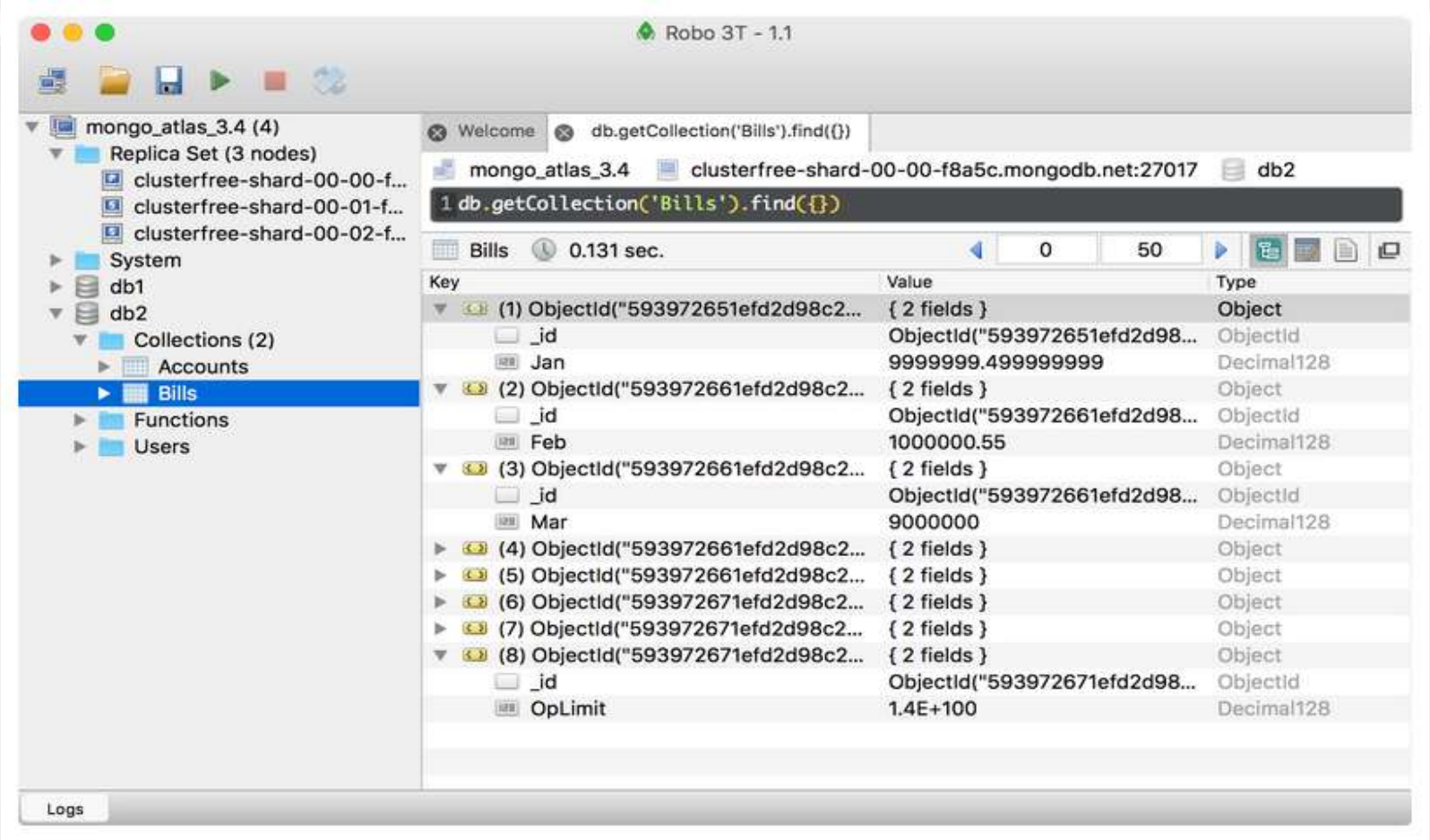
Interakcja z MongoDB

- konsola mongo
- zewn. graficzne narzędzia np. Robo 3T/Robomongo
- z poziomu jęz. programowania

Konsola mongo

```
λ mongo
MongoDB shell version v3.4.2
connecting to: mongodb://127.0.0.1:27017
MongoDB server version: 3.4.2
Server has startup warnings:
2017-03-07T21:43:07.665+0100 I CONTROL  [initandlisten]
2017-03-07T21:43:07.666+0100 I CONTROL  [initandlisten] ** WARNING: Access control is not enabled for the database.
2017-03-07T21:43:07.666+0100 I CONTROL  [initandlisten] **          Read and write access to data and configuration is
 unrestricted.
2017-03-07T21:43:07.667+0100 I CONTROL  [initandlisten]
>
```

Robo 3T



rys z https://robomongo.org/

Uruchomienie demona MongoDB

- mongod
- domyślny port: 27017

- mongo – uruchomienie konsoli (JavaScript shell)

Praca z konsolą

- wybór bazy: use nazwa_bazy
- nie tworzy się bazy exlicit (tak samo nie ma konieczności tworzenia kolekcji explicit, choć można używając polecenia db.createCollection(nazwa)) – przy wstawianiu dokumentów są od razu tworzone
- lista baz: show dbs
- lista kolekcji: show collections

Operacje zapisu

- wstawianie do kolekcji (kolekcji nie tworzy się wcześniej, tylko wstawiając element do niej)

```
db.cars.insert({
    make: "Opel",
    model: "Astra",
    year: 2010,
    price: 25500,
    advert:{
        date: ISODate("2014-10-25"),
        mail: "test@test.com"
    }
})
```

```
db.cars.insert({
    make: "Seat",
    model: "Toledo",
    year: 2012,
    price: 35500,
    features: ["airbags","air conditioning"],
    advert:{
        date: ISODate("2014-10-15"),
        phone: "123456"
    }
})
```

```
db.cars.insert({
    make: "Ford",
    model: "Focus",
    year: 2005,
    price: 15500,
    features: ["airbags","EPS"],
    advert:{
        date: ISODate("2014-10-08"),
        mail: "test2@test.com"
    }
})
```

Walidacja

```
db.createCollection("students", {
   validator: {
      $jsonSchema: {
         bsonType: "object",
         required: [ "name", "year", "major", "gpa" ],
         properties: {
            name: {
               bsonType: "string",
               description: "must be a string and is required"
            },
            gender: {
               bsonType: "string",
               description: "must be a string and is not required"
            },
            year: {
               bsonType: "int",
               minimum: 2017,
               maximum: 3017,
               exclusiveMaximum: false,
               description: "must be an integer in [ 2017, 3017 ] and is required"
            },
            major: {
               enum: [ "Math", "English", "Computer Science", "History", null ],
               description: "can only be one of the enum values and is required"
            },
            gpa: {
               bsonType: [ "double" ],
               minimum: 0,
               description: "must be a double and is required"
            }
         }
      }
   }
})
```

Operacje odczytu

- odczytywanie kolekcji

```
db.cars.find()
```

cars 0 sec.

```
/* 0 */
{
    "_id" : ObjectId("5467e0638b784aa6a9e7d8bd"),
    "make" : "Seat",
    "model" : "Toledo",
    "year" : 2012,
    "price" : 35500,
    "features" : [
        "airbags",
        "air conditioning"
    ],
    "advert" : {
        "date" : ISODate("2014-10-15T00:00:00.000Z"),
        "phone" : "123456"
    }
}

/* 1 */
{
    "_id" : ObjectId("5467e0dd8b784aa6a9e7d8be"),
    "make" : "Opel",
    "model" : "Astra",
    "year" : 2010,
    "price" : 25500,
    "advert" : {
        "date" : ISODate("2014-10-25T00:00:00.000Z"),
        "mail" : "test@test.com"
    }
}
```

Operacje odczytu 2

- odczytywanie kolekcji z dodatkowymi warunkami

```
db.cars.find({year:2005})
```

cars 0 sec.

```
/* 0 */
{
    "_id" : ObjectId("5467e3988b784aa6a9e7d8bf"),
    "make" : "Ford",
    "model" : "Focus",
    "year" : 2005,
    "price" : 15500,
    "features" : [
        "airbags",
        "EPS"
    ],
    "advert" : {
        "date" : ISODate("2014-10-08T00:00:00.000Z"),
        "mail" : "test2@test.com"
    }
}

/* 1 */
{
    "_id" : ObjectId("5467e7dc8b784aa6a9e7d8c1"),
    "make" : "Fiat",
    "model" : "Punto",
    "year" : 2005,
    "price" : 10000,
    "advert" : {
        "date" : ISODate("2014-11-05T00:00:00.000Z")
    }
}
```

Operacje odczytu 3 -Projekcje

- odczyt kolekcji z wyborem pól
  - podanie jakie pola mają się znaleźć w wyniku

```
db.cars.find({year:2005},{make:1,model:1})
```

cars 0 sec.

```
/* 0 */
{
    "_id" : ObjectId("5467e3988b784aa6a9e7d8bf"),
    "make" : "Ford",
    "model" : "Focus"
}

/* 1 */
{
    "_id" : ObjectId("5467e7dc8b784aa6a9e7d8c1"),
    "make" : "Fiat",
    "model" : "Punto"
}
```

  - podanie jakie mają zostać wykluczone

```
db.cars.find({year:2005},{features:0,advert:0,year:0})
```

cars 0 sec.

```
/* 0 */
{
    "_id" : ObjectId("5467e3988b784aa6a9e7d8bf"),
    "make" : "Ford",
    "model" : "Focus",
    "price" : 15500
}

/* 1 */
{
    "_id" : ObjectId("5467e7dc8b784aa6a9e7d8c1"),
    "make" : "Fiat",
    "model" : "Punto",
    "price" : 10000
}
```

uwaga: nie można łączyć tych 2 sposobów

Operacje odczytu 4

- dopasowanie dokładne

```
db.cars.find({features:"airbags"},{make:1,model:1})
```

cars 0 sec.

```
/* 0 */
{
    "_id" : ObjectId("5467e0638b784aa6a9e7d8bd"),
    "make" : "Seat",
    "model" : "Toledo"
}

/* 1 */
{
    "_id" : ObjectId("5467e3988b784aa6a9e7d8bf"),
    "make" : "Ford",
    "model" : "Focus"
}
```

- dopasowanie „częściowe”

```
db.cars.find({features:/cond/},{make:1,model:1,features:1})
```

cars 0 sec.

```
/* 0 */
{
    "_id" : ObjectId("5467e0638b784aa6a9e7d8bd"),
    "make" : "Seat",
    "model" : "Toledo",
    "features" : [
        "airbags",
        "air conditioning"
    ]
}
```

Operacje odczytu 5

- jak odwoływać się do zagnieżdżonych pól?

```
db.cars.find({'advert.mail': {$exists: true}},{make:1,model:1,advert:1})
```

cars 0 sec.

```
/* 0 */
{
    "_id" : ObjectId("5467e0dd8b784aa6a9e7d8be"),
    "make" : "Opel",
    "model" : "Astra",
    "advert" : {
        "date" : ISODate("2014-10-25T00:00:00.000Z"),
        "mail" : "test@test.com"
    }
}

/* 1 */
{
    "_id" : ObjectId("5467e3988b784aa6a9e7d8bf"),
    "make" : "Ford",
    "model" : "Focus",
    "advert" : {
        "date" : ISODate("2014-10-08T00:00:00.000Z"),
        "mail" : "test2@test.com"
    }
}
```

Często używane operatory w zapytaniach

Command	Description
$regex	Match by any PCRE-compliant regular expression string (or just use the // delimiters as shown earlier)
$ne	Not equal to
$lt	Less than
$lte	Less than or equal to
$gt	Greater than
$gte	Greater than or equal to
$exists	Check for the existence of a field
$all	Match all elements in an array
$in	Match any elements in an array
$nin	Does not match any elements in an array
$elemMatch	Match all fields in an array of nested documents
$or	or
$nor	Not or
$size	Match array of given size
$mod	Modulus
$type	Match if field is a given datatype
$not	Negate the given operator check

Sortowanie

rosnąco

```
db.cars.find({},{model:1,price:1}).sort({price:1})
```

cars 0 sec.

```
/* 0 */
{
    "_id" : ObjectId("5467e7dc8b784aa6a9e7d8c1"),
    "model" : "Punto",
    "price" : 10000
}

/* 1 */
{
    "_id" : ObjectId("5467e3988b784aa6a9e7d8bf"),
    "model" : "Focus",
    "price" : 15500
}

/* 2 */
{
    "_id" : ObjectId("5467e0dd8b784aa6a9e7d8be"),
    "model" : "Astra",
    "price" : 25500
}
```

malejąco

```
db.cars.find({},{model:1,price:1}).sort({price:-1})
```

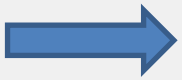
cars 0 sec.

```
/* 0 */
{
    "_id" : ObjectId("5467e0dd8b784aa6a9e7d8be"),
    "model" : "Astra",
    "price" : 25500
}

/* 1 */
{
    "_id" : ObjectId("5467e3988b784aa6a9e7d8bf"),
    "model" : "Focus",
    "price" : 15500
}

/* 2 */
{
    "_id" : ObjectId("5467e7dc8b784aa6a9e7d8c1"),
    "model" : "Punto",
    "price" : 10000
}
```

Aktualizacja danych

- update(criteria,operation)

```
db.cars.update(
{make:"Seat"},
{$set: {price:40000}}
)
```



```
db.cars.find({make:"Seat"})
```

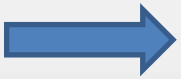
cars 0 sec.

```
/* 0 */
{
    "_id" : ObjectId("5467e0638b784aa6a9e7d8bd"),
    "make" : "Seat",
    "model" : "Toledo",
    "year" : 2012,
    "price" : 40000,
    "features" : [
        "airbags",
        "air conditioning"
    ],
    "advert" : {
        "date" : ISODate("2014-10-15T00:00:00.000Z"),
        "phone" : "123456"
    }
}
```

uwaga na słowo kluczowe $set
- bez tego podmienia dokument

to samo ObjectId

```
db.cars.update(
{_id:ObjectId("5467e0638b784aa6a9e7d8bd")},
{price:40000}
)
```



```
db.cars.find({_id:ObjectId("5467e0638b784aa6a9e7d8bd")})
```

cars 0 sec.

```
/* 0 */
{
    "_id" : ObjectId("5467e0638b784aa6a9e7d8bd"),
    "price" : 40000
}
```

Często używane operatory przy aktualizacji

Command	Description
$set	Sets the given field with the given value
$unset	Removes the field
$inc	Adds the given field by the given number
$pop	Removes the last (or first) element from an array
$push	Adds the value to an array
$pushAll	Adds all values to an array
$addToSet	Similar to push, but won't duplicate values
$pull	Removes matching value from an array
$pullAll	Removes all matching values from an array

przykład dla $inc

```
db.cars.find({_id:ObjectId("5467e0638b784aa6a9e7d8bd")})
```

cars 0 sec.

```
/* 0 */
{
    "_id" : ObjectId("5467e0638b784aa6a9e7d8bd"),
    "price" : 40000
}
```

```
db.cars.update(
{_id:ObjectId("5467e0638b784aa6a9e7d8bd")},
{$inc: {price:5000}}
)
```

```
db.cars.find({_id:ObjectId("5467e0638b784aa6a9e7d8bd")})
```

cars 0 sec.

```
/* 0 */
{
    "_id" : ObjectId("5467e0638b784aa6a9e7d8bd"),
    "price" : 45000
}
```

Usuwanie dokumentów i kolekcji

- funkcja *remove* na kolekcji

```
db.cars.find({_id:ObjectId("5467e0638b784aa6a9e7d8bd")})
```

cars 0 sec.

```
/* 0 */
{
    "_id" : ObjectId("5467e0638b784aa6a9e7d8bd"),
    "price" : 45000
}
```

```
db.cars.count()
```

0 sec.

```
4
```

```
db.cars.remove({_id:ObjectId("5467e0638b784aa6a9e7d8bd")})
```

```
db.cars.find({_id:ObjectId("5467e0638b784aa6a9e7d8bd")})
```

Script executed successfully, but there is no results to show.

```
db.cars.count()
```

0 sec.

```
3
```

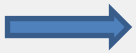
- funkcja *drop* na kolekcji

```
db.test.drop()
```

Referencje do innych dokumentów

- MongoDB nie wspiera joinów, ale umożliwia przechowywanie w dokumencie referencji do innego dokumentu (złączenia muszą być wykonywane po stronie aplikacji przez pobieranie odpowiednich dokumentów)
- konstrukcja {$ref:”nazwa_kolekcji”, $id:”id_referencji”}

```
db.countries.insert({
    _id: "us",
    name: "USA"
})

db.cars.update(
{make:"Ford"},
{$set: {country: {$ref: "countries", $id: "us"}}}
)
```



```
{
    "_id" : ObjectId("5467e3988b784aa6a9e7d8bf"),
    "make" : "Ford",
    "model" : "Focus",
    "year" : 2005,
    "price" : 15500,
    "features" : [
        "airbags",
        "EPS"
    ],
    "advert" : {
        "date" : ISODate("2014-10-08T00:00:00.000Z"),
        "mail" : "test2@test.com"
    },
    "country" : {
        "$ref" : "countries",
        "$id" : "us"
    }
}
```

Referencje do innych dokumentów 2

- pobieranie dokumentów z użyciem referencji

```
var ford = db.cars.findOne({make:"Ford"})
db.countries.findOne({_id: ford.country.$id})
```

0 sec.

```
/* 0 */
{
    "_id" : "us",
    "name" : "USA"
}
```

Lista kolekcji

- wyświetlenie listy kolekcji

```
show collections
```

0 sec.

```
cars
system.indexes
```

Funkcje 1

- Definiowanie funkcji

```
function insertCar(
 make, model, year, price, date){
     db.cars.insert({
         make:make,
         model:model,
         year:year,
         price:price,
         advert:{
             date:ISODate(date)
         }
     });
}

insertCar("Fiat","Punto",2005,10000,"2014-11-05")
```

Funkcje 2

- wyświetlenie treści funkcji

```
db.cars.insert
```

0 sec.

```
function (obj, options, _allow_dot) {
    if (!obj) {
        throw "no object passed to insert!";
    }
    if (!_allow_dot) {
        this._validateForStorage(obj);
    }
    if (typeof options == "undefined") {
        options = 0;
    }
    if (typeof obj._id == "undefined" && !Array.isArray(obj)) {
        var tmp = obj;
        obj = {_id:new ObjectId};
        for (var key in tmp) {
            obj[key] = tmp[key];
        }
    }
    this._db._initExtraInfo();
    this._mongo.insert(this._fullName, obj, options);
    this._lastID = obj._id;
    this._db._getExtraInfo("Inserted");
}
```

Zapytania

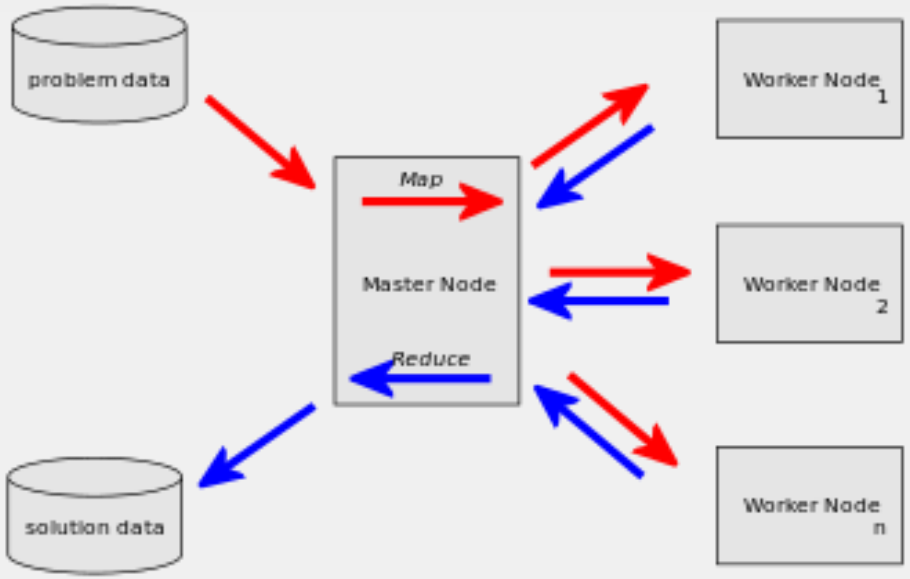
SQL	MongoDB Query Language
`SELECT a,b FROM users`	`db.users.find({}, {a:1,b:1})`
`SELECT * FROM users`	`db.users.find()`
`SELECT * FROM users WHERE age=33`	`db.users.find({age:33})`
`SELECT a,b FROM users WHERE age=33`	`db.users.find({age:33}, {a:1,b:1})`
`SELECT * FROM users WHERE age=33 ORDER BY name`	`db.users.find({age:33}).sort({name:1})`
`SELECT * FROM users WHERE age>33`	`db.users.find({age:{$gt:33}})`
`SELECT * FROM users WHERE age!=33`	`db.users.find({age:{$ne:33}})`

Map reduce

- MapReduce - platforma do przetwarzania równoległego dużych zbiorów danych stworzona przez Google
- **krok "map"**
  - węzeł nadzorczy (master node) pobiera dane z wejścia i dzieli je na mniejsze podproblemy, po czym przesyła je do węzłów roboczych (worker nodes).
  - każdy z węzłów roboczych może albo dokonać kolejnego podziału na podproblemy, albo przetworzyć problem i zwrócić odpowiedź do głównego programu.
- **krok "reduce"**
  - główny program bierze odpowiedzi na wszystkie podproblemy i łączy je w jeden wynik - odpowiedź na główny problem.

https://pl.wikipedia.org/wiki/MapReduce

Map reduce



http://en.wikipedia.org/wiki/MapReduce

Przykład map reduce 1

- do wyliczenia:

  średnia cena samochodów na poszczególne lata produkcji

- map

```
var mapF = function(){
    emit(this.year, this.price);
};
```

- reduce

```
var reduceF = function(year, valuesPrices){
    return Array.avg(valuesPrices);
};
```

Przykład map reduce 1 cd

- wywołanie

```
db.cars.mapReduce(
    mapF,
    reduceF,
    {out: "avg_price_per_year"}
)
```

wynik map reduce zapisany jest w kolekcji podanej po nazwie *out*

- wynik

```
db.avg_price_per_year.find()
```

avg_price_per_year 0 sec.

```
/* 0 */
{
    "_id" : 2005,
    "value" : 12750
}

/* 1 */
{
    "_id" : 2010,
    "value" : 25500
}
```

Single purpose aggregation operations

- wbudowane idiomy wyliczania pewnych operacji
- count

```
db.cars.count()
```

0 sec.

```
3
```

- count z filtrem – count(query)

```
db.cars.count({year:2005})
```

0 sec.

```
2
```

Single purpose aggregation operations 2

- distinct(field, query)

```
db.cars.distinct("year")
```

0 sec.

```
/* 0 */
{
    "0" : 2010,
    "1" : 2005
}
```

uwaga:
Robomongo (po lewej) zwraca dokument JSON, natomiast w konsoli Mongo wynik jest w postaci tablicy (po prawej)

```
> db.cars.distinct("year")
[ 2010, 2005 ]
```

- query można użyć do dodatkowego przefiltrowania

```
> db.cars.distinct("year",{price:{$gt:20000}})
[ 2010 ]
```

Single purpose aggregation operations 3

- group

przygotowanie danych

```
db.cars.insert({make:"Opel",model:"Corsa",year:2013, price:40000})
```

```
db.cars.find({},{make:1,model:1,price:1,year:1})
```

cars 0 sec.

	_id	make	model	year	price
1	ObjectId("5...	Opel	Astra	2010.0...	25500.0...
2	ObjectId("5...	Ford	Focus	2005.0...	15500.0...
3	ObjectId("5...	Fiat	Punto	2005.0...	10000.0...
4	ObjectId("5...	Opel	Corsa	2013.0...	40000.0...

ile samochodów z każdego roku ma cenę poniżej 40000?

```
db.cars.group({
    key: {year:1},
    cond: {price: {$lt: 40000}},
    initial: {count: 0},
    reduce: function(current, result){result.count += 1}
})
```

0.004 sec.

```
/* 0 */
{
    "0" : {
        "year" : 2010,
        "count" : 1
    },
    "1" : {
        "year" : 2005,
        "count" : 2
    }
}
```

Single purpose aggregation operations 4

- group cd

średnia cena samochodu w każdym roku spośród tych, co kosztują poniżej 40000

```
db.cars.group({
    key: {year:1},
    cond: {price: {$lt: 40000}},
    initial: {count: 0, sum: 0},
    reduce: function(current, result){
        result.count += 1
        result.sum += current.price
     },
    finalize: function(result){result.average_price = result.sum/result.count}
})
```

0.004 sec.

```
/* 0 */
{
    "0" : {
        "year" : 2010,
        "count" : 1,
        "sum" : 25500,
        "average_price" : 25500
    },
    "1" : {
        "year" : 2005,
        "count" : 2,
        "sum" : 25500,
        "average_price" : 12750
    }
}
```

Indeksy

- podobne do indeksów w RDBMS
- MongoDB wspiera indeksy na dowolne pole w dokumencie lub zagnieżdżonym dokumencie
- indeksy są zdefiniowane na poziomie kolekcji
- indeksy korzystają z B-drzew,
  - są także indeksy do danych geograficznych
- tworzenie indeksów: ensureIndex(fields, options)

Przykład działania indeksów

na podstawie MongoDB in Action

```
for(i=0; i < 2000000; i++){
    db.numbers.save({num: i})
}
```

```
db.numbers.ensureIndex({num: 1})
```

```
db.numbers.find({num: {$gt: 15000, $lt: 23000}}).explain()
```

0.808 sec.

```
/* 0 */
{
    "cursor" : "BasicCursor",
    "isMultiKey" : false,
    "n" : 7999,
    "nscannedObjects" : 2000000,
    "nscanned" : 2000000,
    "nscannedObjectsAllPlans" : 2000000,
    "nscannedAllPlans" : 2000000,
    "scanAndOrder" : false,
    "indexOnly" : false,
    "nYields" : 15625,
    "nChunkSkips" : 0,
    "millis" : 807,
```

```
db.numbers.find({num: {$gt: 15000, $lt: 23000}}).explain()
```

0.006 sec.

```
/* 0 */
{
    "cursor" : "BtreeCursor num_1",
    "isMultiKey" : false,
    "n" : 7999,
    "nscannedObjects" : 7999,
    "nscanned" : 7999,
    "nscannedObjectsAllPlans" : 7999,
    "nscannedAllPlans" : 7999,
    "scanAndOrder" : false,
    "indexOnly" : false,
    "nYields" : 62,
    "nChunkSkips" : 0,
    "millis" : 6,
    "indexBounds" : {
        "num" : [
            [
                15000,
                23000
            ]
        ]
    },
```

Rodzaje indeksów

- single field – indeks na pojedyncze pole dokumentu (także pole zagnieżdżonego dokumentu)
- compound – indeks na więcej niż jedno pole dokumentu
- multikey – indeks na pole tablicowe (dodaje do indeksu każdą wartość w tablicy)
- geospatial – indeks obsługujący zapytania przestrzenne
- text – obsługuje zapytania na tekstach w dokumencie (language specific – używa stop words, stemming wyrazów)
- hashed – zarządza elementami z hashami wartości

Właściwości indeksów

- unikalny (unique) – domyślnie false

```
db.members.ensureIndex( { "user_id": 1 }, { unique: true } )
```

  - odrzuca dokumenty ze zduplikowanymi wartościami indeksowanego pola
- rzadki (sparse) – domyślnie indeksy są gęste

```
db.addresses.ensureIndex( { "xmpp_id": 1 }, { sparse: true } )
```

  - rzadki oznacza, że indeks w sobie zawiera pozycje dla dokumentów, które zawierają dane pole (rzadki => nie zawiera wszystkich dokumentów z danej kolekcji)
  - gęsty indeks zawiera pozycje dla wszystkich dokumentów w kolekcji, nawet jeśli nie zawierają one danego pola (przechowuje null dla takich dokumentów)

MongoDB Java API

- podstawowa klasa do połączenia z MongoDB to MongoClient (zastępuje klasę Mongo, po której dziedziczy)
  - główna różnica między tymi 2 klasami: MongoClient domyślnie przy wszystkich operacjach zapisu oczekuje na potwierdzenie z serwera
- Dokument jest reprezentowany przez klasę BasicDBObject
- Java MongoDB driver
  - MongoClient sam zarządza wewnętrzną pulą połączeń do bazy => dla każdego żądania do bazy, wątek Javy otrzymuje wątek z puli, wykonuje operację i zwalnia połączenie (czyli przy wykonywaniu różnych operacji różne połączenia mogą być używane)

Typy danych Java vs MongoDB

- większość (int, double, String…) jest automatycznie mapowana
- Object Ids

za http://docs.mongodb.org/ecosystem/drivers/java-types/

```
ObjectId id = new ObjectId();
ObjectId copy = new ObjectId(id);
```

org.bson.types.ObjectId

- wyrażenia regularne

```
Pattern john = Pattern.compile("joh?n", Pattern.CASE_INSENSITIVE);
BasicDBObject query = new BasicDBObject("name", john);

// finds all people with "name" matching /joh?n/i
DBCursor cursor = collection.find(query);
```

java.util.regex.Pattern

Typy danych Java vs MongoDB 2

- data

```
Date now = new Date();
BasicDBObject time = new BasicDBObject("ts", now);

collection.save(time);
```

java.util.Date

- referencje bazodanowe

```
DBRef addressRef = new DBRef(db, "foo.bar", address_id);
DBObject address = addressRef.fetch();

DBObject person = BasicDBObjectBuilder.start()
    .add("name", "Fred")
    .add("address", addressRef)
    .get();
collection.save(person);

DBObject fred = collection.findOne();
DBRef addressObj = (DBRef)fred.get("address");
addressObj.fetch()
```

com.mongodb.DBRef

Typy danych Java vs MongoDB 3

- embedded documents

```
{
    "x" : {
            "y" : 3
    }
}
```

```
BasicDBObject y = new BasicDBObject("y", 3);
BasicDBObject x = new BasicDBObject("x", y);
```

- arrays

Dowolny obiekt dziedziczący po interfejsie List w Javie będzie odwzorowany na array

```
{
    "x" : [
            1,
            2,
            {"foo" : "bar"},
            4
    ]
}
```

```
ArrayList x = new ArrayList();
x.add(1);
x.add(2);
x.add(new BasicDBObject("foo", "bar"));
x.add(4);

BasicDBObject doc = new BasicDBObject("x", x);
```

Java API

- tworzenie połączenie i uzyskanie dostępu do bazy danych

za http://docs.mongodb.org/ecosystem/tutorial/getting-started-with-java-driver/

```
MongoClient mongoClient = new MongoClient();
// or
MongoClient mongoClient = new MongoClient( "localhost" );
// or
MongoClient mongoClient = new MongoClient( "localhost" , 27017 );

DB db = mongoClient.getDB( "mydb" );
```

- pobieranie kolekcji z bazy

```
Set<String> colls = db.getCollectionNames();

DBCollection coll = db.getCollection("testCollection");
```

Java API 2

- wstawianie dokumentu do bazy

```
{
    "name" : "MongoDB",
    "type" : "database",
    "count" : 1,
    "info" : {
                x : 203,
                y : 102
              }
}
```

```
BasicDBObject doc = new BasicDBObject("name", "MongoDB")
        .append("type", "database")
        .append("count", 1)
        .append("info", new BasicDBObject("x", 203).append("y", 102));
coll.insert(doc);
```

- wstawianie wielu dokumentów

```
for (int i=0; i < 100; i++) {
    coll.insert(new BasicDBObject("i", i));
}
```

- zliczenie dokumentów w kolekcji

```
System.out.println(coll.getCount());
```

Java API 3

- wyszukiwanie dokumentów

db.things.find({i: 71});

```
BasicDBObject query = new BasicDBObject("i", 71);

cursor = coll.find(query);

try {
   while(cursor.hasNext()) {
       System.out.println(cursor.next());
   }
} finally {
   cursor.close();
}
```

```
db.things.find({j: {$ne: 3}, k: {$gt: 10} });
```

```
query = new BasicDBObject("j", new BasicDBObject("$ne", 3))
        .append("k", new BasicDBObject("$gt", 10));

cursor = coll.find(query);

try {
    while(cursor.hasNext()) {
        System.out.println(cursor.next());
    }
} finally {
    cursor.close();
}
```

Java API 4

- aggregation framework

1. First operation

```
$match: { type: "airfare"}
```

2. Piped into:

```
$project: { department: 1, amount: 1 }
```

3. Piped into:

```
$group: { _id: "$department",
          average: { $avg: "$amount" } }
```

4. Piped into:

```
$sort: { "$amount": -1 }
```

```
// create our pipeline operations, first with the $match
DBObject match = new BasicDBObject("$match", new BasicDBObject("type", "airfare"));

// build the $projection operation
DBObject fields = new BasicDBObject("department", 1);
fields.put("amount", 1);
fields.put("_id", 0);
DBObject project = new BasicDBObject("$project", fields );

// Now the $group operation
DBObject groupFields = new BasicDBObject( "_id", "$department");
groupFields.put("average", new BasicDBObject( "$avg", "$amount"));
DBObject group = new BasicDBObject("$group", groupFields);

// Finally the $sort operation
DBObject sort = new BasicDBObject("$sort", new BasicDBObject("amount", -1));

// run aggregation
List<DBObject> pipeline = Arrays.asList(match, project, group, sort);
AggregationOutput output = coll.aggregate(pipeline);
```

```
db.coll.aggregate([
   {$match: {type: ”airfare”}},
   {$project: {department:1, amount: 1}},
   {$group: {_id:”$department”},
                  average: {$avg: ”$amount”}},
   {$sort: {”$amount”:-1}}
])
```

```
for (DBObject result : output.results()) {
    System.out.println(result);
}
```

Java API 5

- map reduce

przykład za http://www.javacodegeeks.com/2012/06/mapreduce-with-mongodb.html

```
DBCollection books = db.getCollection("books");

String map = "function() { "+
             "var category; " +
             "if ( this.pages >= 250 ) "+
             "category = 'Big Books'; " +
             "else " +
             "category = 'Small Books'; "+
             "emit(category, {name: this.name});}";

String reduce = "function(key, values) { " +
                "var sum = 0; " +
                "values.forEach(function(doc) { " +
                "sum += 1; "+
                "}); " +
                "return {books: sum};} ";

MapReduceCommand cmd = new MapReduceCommand(books, map, reduce,
  null, MapReduceCommand.OutputType.INLINE, null);

MapReduceOutput out = books.mapReduce(cmd);

for (DBObject o : out.results()) {
 System.out.println(o.toString());
}
```

wyrażenia JavaScript

Odwzorowanie klas Javy na kolekcje

- dziedziczenie klasy po BasicDBObject

```
public class Address extends BasicDBObject {

   public Address() {
   }

  public Address(String streetName, String postalCode, String place, String countr
y) {
    put("street", streetName);
    put("postalcode", postalCode);
    put("place", place);
    put("country", country);
   }

  public String place(){
    return (String) get("place");
   }
}
```

http://blog.knuthaugen.no/2010/07/comparing-mongodb-java-frameworks.html

```
Mongo m = new Mongo( "127.0.0.1" , 27017 );

testDb = m.getDB( "test" );
persons = testDb.getCollection("persons");
persons.setObjectClass(Person.class);
```

```
Person test = new Person("Knut Haugen",
                         new Address("Josefines gate", "0401",
                                     "oslo", "Norge"));
persons.insert(test);

Person test2 = (Person) persons.findOne();
```

Odwzorowanie klas Javy na kolekcje 2

- implementacja przez klasę interfejsu DBObject

```
public Object put(String s, Object o) { }
public void putAll(DBObject dbObject) { }
public void putAll(Map map) { }
public Object get(String s) { }
public Map toMap() { }
public Object removeField(String s) { }
public boolean containsKey(String s) { }
public boolean containsField(String s) { }
public Set&lt;String&gt; keySet() { }
public void markAsPartialObject() { }
public boolean isPartialObject() {}
```

rys. http://blog.knuthaugen.no/2010/07/comparing-mongodb-java-frameworks.html

przykład za http://docs.mongodb.org/ecosystem/tutorial/use-java-dbobject-to-perform-saves/

```
public class Tweet implements DBObject {
    /* ... */
}
```

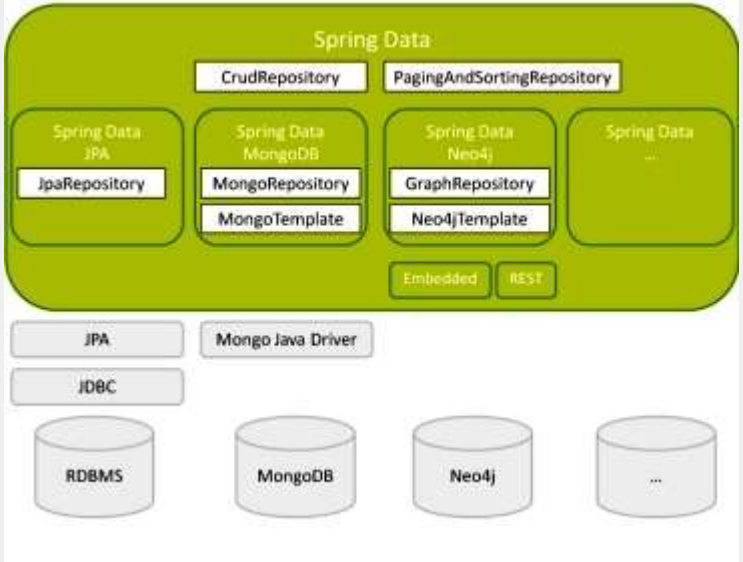
```
Tweet myTweet = new Tweet();
myTweet.put("user", userId);
myTweet.put("message", msg);
myTweet.put("date", new Date());

collection.insert(myTweet);
```

```
collection.setObjectClass(Tweet.class);

Tweet myTweet = (Tweet)collection.findOne();
```

```
Tweet myTweet = (Tweet)collection.findOne();
myTweet.put("message", newMsg);

collection.save(myTweet);
```

Odwzorowanie klas Javy na kolekcje 3

- Spring Data MongoDB



http://projects.spring.io/spring-data-mongodb/

- Hibernate OGM (Object/Grid Mapper)

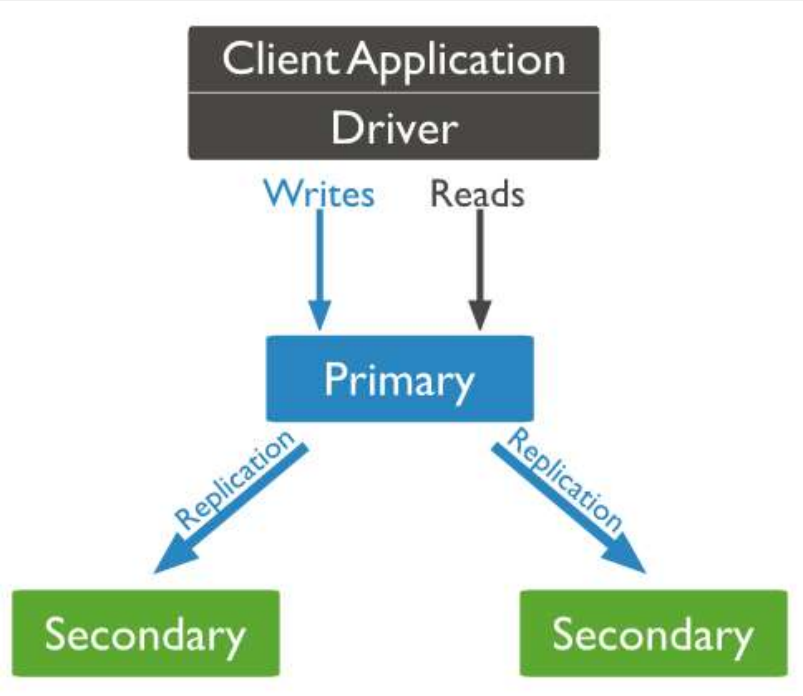
http://hibernate.org/ogm/

Replikacja

- proces synchronizacji danych na wielu serwerach
- cel
  - redundancja danych
  - większa dostępność danych (umożliwia zmniejszenie czasu na przywrócenie działania usługi po awarii)
  - zwiększa przepustowość odczytu danych (klienci mogą równolegle odczytywać dane z różnych replik)

Repliki w MongoDB

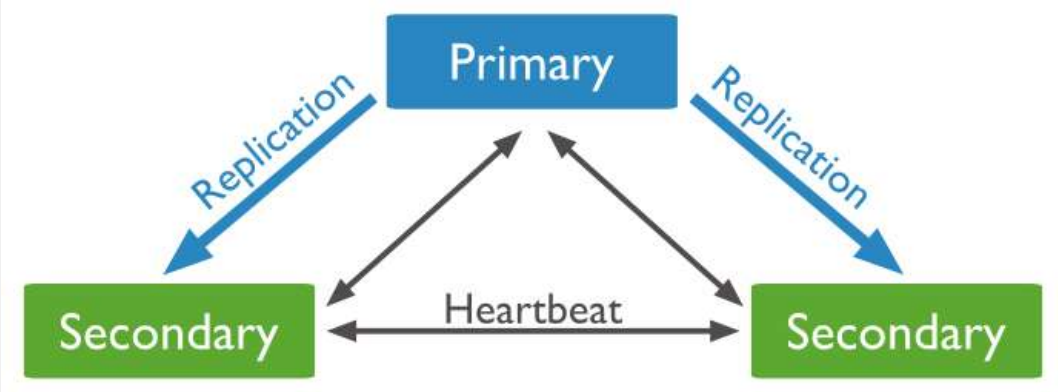
- zbiór replik jest grupą instancji uruchomionych demonów mongod, które hostują ten sam zbiór danych
- podział na primary (jeden) i secondaries (reszta)



http://docs.mongodb.org/manual/core/replication-introduction/

Repliki w MongoDB 2

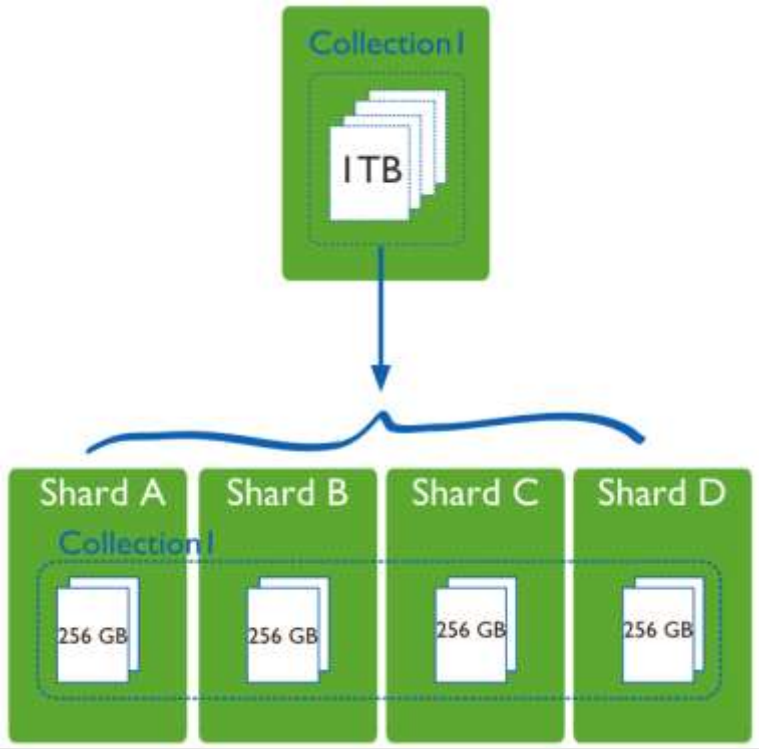
- repliki nasłuchują na komunikaty heartbeat, a gdy primary ma awarię jest wybierany nowy primary z pozostałych przez głosowanie



http://docs.mongodb.org/manual/core/replication-introduction/

Sharding

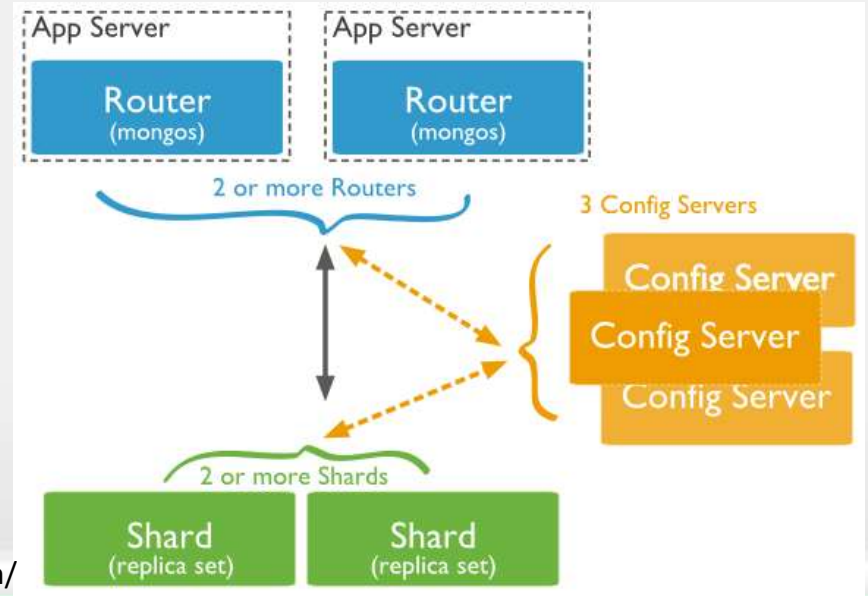
- horizontal scaling, podział przechowywanych danych na wiele maszyn



http://docs.mongodb.org/manual/core/sharding-introduction/

Sharding w MongoDB

- Komponenty sharded cluster:
  - shards – przechowują dane (ze względów wysokiej dostępności i spójności danych shardy często są zbiorami replik)
  - query routers (mongos instances) – są interfejsem dla aplikacji klienta, kierują zapytania do odpowiednich shardów i zwracają wynik zapytań do klienta (klient wysyła zapytanie do jednego query routera, ale zwykle te routery są zwielokrotnione)
  - config servers – przechowują metadane dotyczące klastru (query routers używają tych metadanych do wyboru odpowiednich shardów)



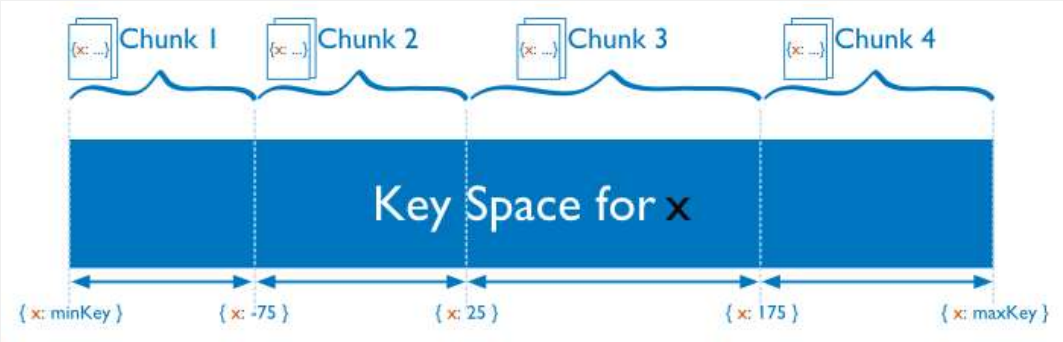
http://docs.mongodb.org/manual/core/sharding-introduction/

Sharding w MongoDB 2

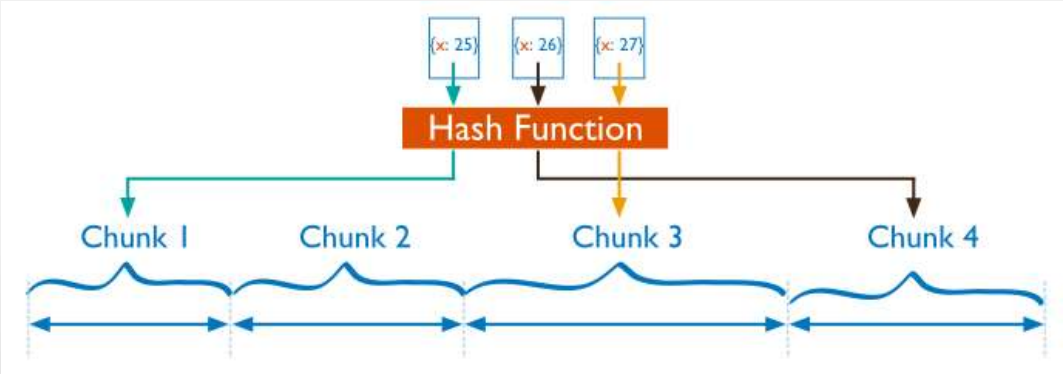
- Partycjonowanie danych odbywa się na poziomie kolekcji z wykorzystaniem shard key
- shard key jest wybierany ręcznie (indeksowane pole lub indeksowany compound – istniejące w każdym dokumencie w kolekcji)
- shard key jest dzielony na części (chunks) i rozdystrybuowany na shardach

Sharding w MongoDB 3

- Sposoby partycjonowania
  - range based sharding



  - hash based sharding



http://docs.mongodb.org/manual/core/sharding-introduction/

Transakcyjność

- MongoDB zapewnia atomowość operacji na poziomie pojedynczego dokumentu (który może być bardzo złożony)
- transakcje nie są obsługiwane
- w razie potrzeby twórcy MongoDB sugerują używanie „pseudotransakcji”

Pseudotransakcje w MongoDB

przykład z http://cookbook.mongodb.org/patterns/perform-two-phase-commits/

- stworzenie obiektu transakcji w stanie initial i pola pendingTransactions we właściwych obiektach

```
foo:PRIMARY> db.accounts.save({name: "A", balance: 1000, pendingTransactions: []})
foo:PRIMARY> db.accounts.save({name: "B", balance: 1000, pendingTransactions: []})
foo:PRIMARY> db.accounts.find()
{ "_id" : ObjectId("4d7bc66cb8a04f512696151f"), "name" : "A", "balance" : 1000, "pendingTransactions" : [ ] }
{ "_id" : ObjectId("4d7bc67bb8a04f5126961520"), "name" : "B", "balance" : 1000, "pendingTransactions" : [ ] }
{code}And we need one collection representing transactions:
{code}foo:PRIMARY> db.transactions.save({source: "A", destination: "B", value: 100, state: "initial"})
foo:PRIMARY> db.transactions.find()
{ "_id" : ObjectId("4d7bc7a8b8a04f5126961522"), "source" : "A", "destination" : "B", "value" : 100, "state" : "initial" }
```

- zmiana stanu obiektu transakcji na pending

```
foo:PRIMARY> t = db.transactions.findOne({state: "initial"})
{
    "_id" : ObjectId("4d7bc7a8b8a04f5126961522"),
    "source" : "A",
    "destination" : "B",
    "value" : 100,
    "state" : "initial"
}
foo:PRIMARY> db.transactions.update({_id: t._id}, {$set: {state: "pending"}})
foo:PRIMARY> db.transactions.find()
{ "_id" : ObjectId("4d7bc7a8b8a04f5126961522"), "source" : "A", "destination" : "B", "value" : 100, "state" : "pending" }
```

Pseudotransakcje w MongoDB 2

- wykonanie operacji skojarzonych z transakcją, gdy stan transakcji jest pending

```
foo:PRIMARY> db.accounts.update({name: t.source, pendingTransactions: {$ne: t._id}}, {$inc: {balance: -t.value}, $push: {pendingTransactions: t._id}})
foo:PRIMARY> db.accounts.update({name: t.destination, pendingTransactions: {$ne: t._id}}, {$inc: {balance: t.value}, $push: {pendingTransactions: t._id}})
foo:PRIMARY> db.accounts.find()
{ "_id" : ObjectId("4d7bc97fb8a04f5126961523"), "balance" : 900, "name" : "A", "pendingTransactions" : [ ObjectId("4d7bc7a8b8a04f5126961522") ] }
{ "_id" : ObjectId("4d7bc984b8a04f5126961524"), "balance" : 1100, "name" : "B", "pendingTransactions" : [ ObjectId("4d7bc7a8b8a04f5126961522") ] }
```

- zmiana stanu transakcji na committed

```
foo:PRIMARY> db.transactions.update({_id: t._id}, {$set: {state: "committed"}})
foo:PRIMARY> db.transactions.find()
{ "_id" : ObjectId("4d7bc7a8b8a04f5126961522"), "destination" : "B", "source" : "A", "state" : "committed", "value" : 100 }
```

- usunięcie informacji o pendingTransaction we właściwych obiektach

```
foo:PRIMARY> db.accounts.update({name: t.source}, {$pull: {pendingTransactions: ObjectId("4d7bc7a8b8a04f5126961522")}})
foo:PRIMARY> db.accounts.update({name: t.destination}, {$pull: {pendingTransactions: ObjectId("4d7bc7a8b8a04f5126961522")}})
foo:PRIMARY> db.accounts.find()
{ "_id" : ObjectId("4d7bc97fb8a04f5126961523"), "balance" : 900, "name" : "A", "pendingTransactions" : [ ] }
{ "_id" : ObjectId("4d7bc984b8a04f5126961524"), "balance" : 1100, "name" : "B", "pendingTransactions" : [ ] }
```

- zmiana stanu transakcji na done

```
foo:PRIMARY> db.transactions.update({_id: t._id}, {$set: {state: "done"}})
foo:PRIMARY> db.transactions.find()
{ "_id" : ObjectId("4d7bc7a8b8a04f5126961522"), "destination" : "B", "source" : "A", "state" : "done", "value" : 100 }
```

Kolekcje z danymi o określonym czasie trwania

- dla kolekcji można ustawić TTL (Time To Live), a po określonym czasie mongod automatycznie usuwa przeterminowane dane

```
db.log.events.ensureIndex( { "status": 1 }, { expireAfterSeconds: 3600 } )
```

Pole *status* musi zawierać datę/timestamp. MongoDB sprawdza to pole by określić czy dana jest przeterminowana czy nie

- kiedy przydatne?
  - logi, informacje sesyjne
- dostępne od wersji MongoDB 2.2

Server-side commands

funkcja z dużą interakcją z bazą danych

```
update_area = function() {
  db.phones.find().forEach(
    function(phone) {
      phone.components.area++;
      phone.display = "+"+
        phone.components.country+" "+
        phone.components.area+"-"+
        phone.components.number;
      db.phone.update({ _id : phone._id }, phone, false);
    }
  )
}
```

wywołanie funkcji JS po stronie serwera

```
> db.eval(update_area)
```

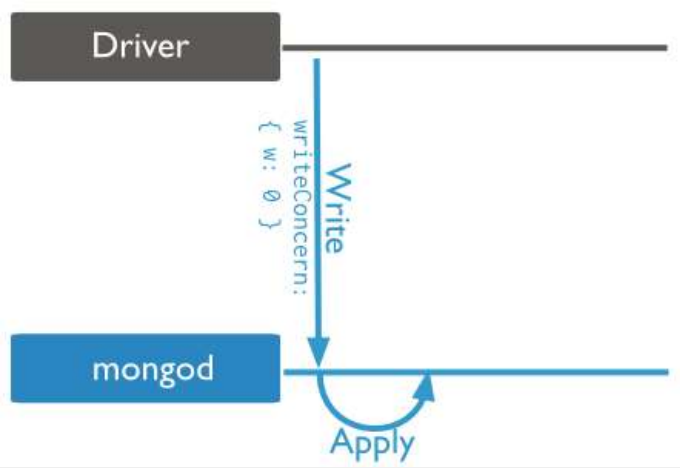
przykład za Seven Databases in Seven Weeks

- zalety
  - wydajność
- wady
  - blokowanie demona mongod podczas wykonywania kodu JS (!!!) => używać tylko do administracyjnych zadań, gdy nikt nie korzysta z bazy

Write concern

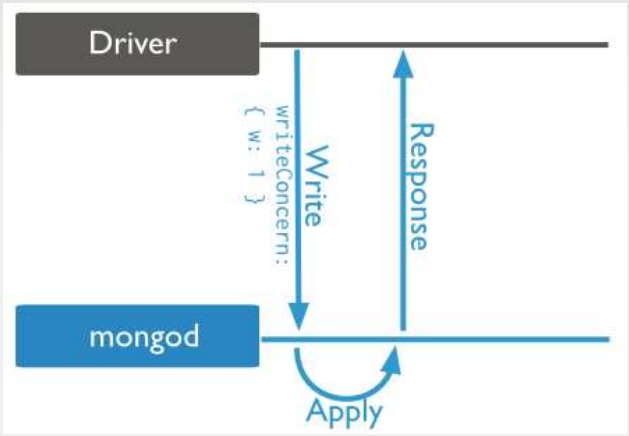
- określa gwarancje dot. raportowania sukcesu na operacjach zapisu
- typy: unacknowledged, acknowledged, journaled, replica acknowledged
- unacknowledged

za http://docs.mongodb.org/manual/core/write-concern/

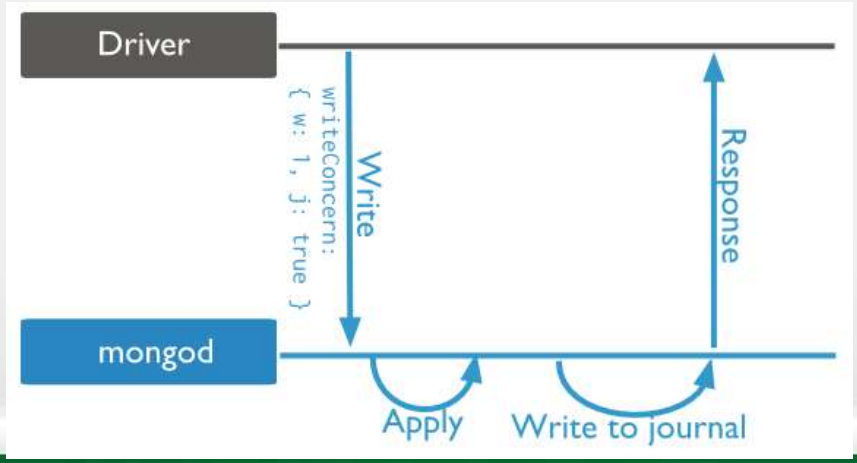


Write concern 2

- acknowledged
  - domyślny tryb w nowszych driverach

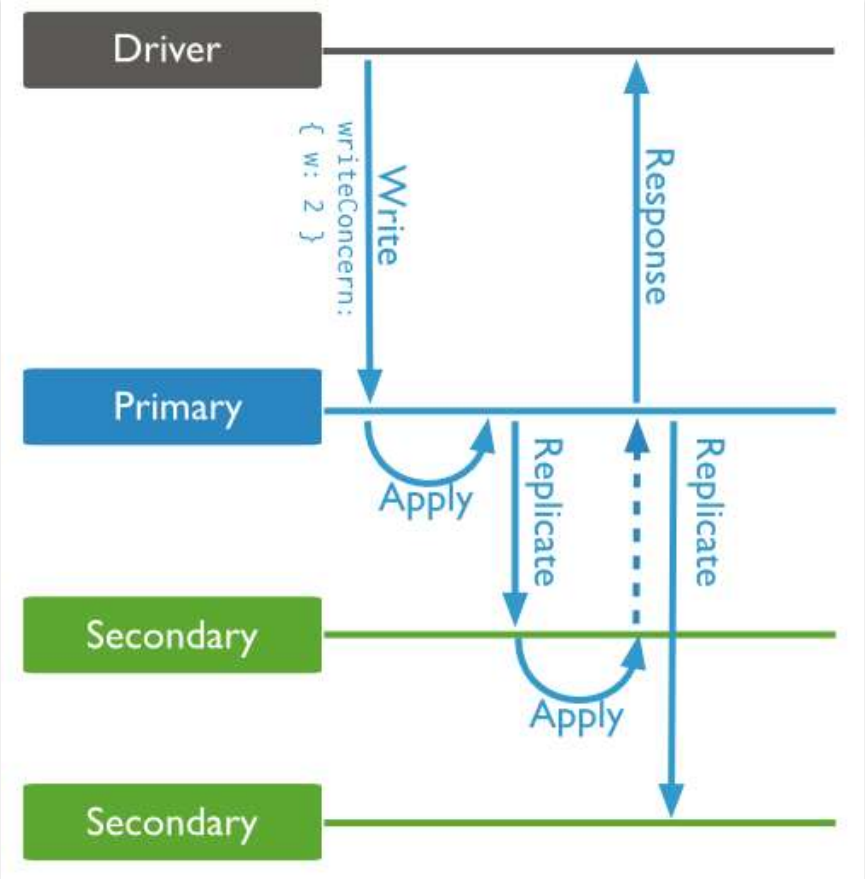


- journaled



Write concern 3

- replica acknowledged



- jak zmienić?

```
mongoClient.setWriteConcern(WriteConcern.JOURNALED);
```

GridFS

- specyfikacja dedykowana do zapisywania i odczytywania plików przekraczających dopuszczalny rozmiar dokumentów BSON 16MB, zwłaszcza do zapisywania plików binarnych
- można używać do dowolnych dokumentów (są one dzielone i przetrzymywane w postaci chunks)

Jak korzystać z GridFS w Javie?

przykład za http://blog.rezajp.info/posts/how-to-write-or-read-images-to-or-from-mongodb-gridfs/

- zapis

```
public String saveImage(byte[] content,String fileName,String contentType) {

// create a table/collection or use the existing one
GridFS gfsPhoto = new GridFS(mongoTemplate.getDb(), "photo");

GridFSInputFile gfsFile = gfsPhoto.createFile(content);

gfsFile.setFilename(fileName);
gfsFile.setContentType(contentType);
gfsFile.save();

return gfsFile.getId().toString();
}
```

- odczyt

```
GridFS gfsPhoto = new GridFS(mongoTemplate.getDb(), "photo");

GridFSDBFile image = gfsPhoto.findOne(new ObjectId(id));

InputStream stream = image.getInputStream();
```

Backup

- mongodump i mongorestore – narzędzia do tworzenia backupu (hot backup) i przywracania z niego bazy
- mongoexport i mongoimport – eksport i import JSON, CSV i TSV (tab separated values)
  - przykład dla importu w json:

```
mongoimport --db <dbname> --collection <collname>
        --type json --file <file>
```

Bazy dokumentowe

- Zalety
  - wydajność
  - schema-less
  - dobrze się skalują
- Wady
  - brak powiązania między polami => dane mogą łatwo stać się niespójne
  - redundantność danych

Kiedy warto używać MongoDb?

- nie ma stałego schematu danych
- high write load
- wymagania dot. dużej dostępności
  - ?
- b.duża ilość danych (skalowanie relacyjnych baz danych jest trudniejsze)
  - ?
- caching

Best practices

- unikaj negacji w zapytaniach przy indeksach rzadkich
  - w takim przypadku MongoDB nie indeksuje braku wartości, więc warunek negacji może wymagać przeskanowania wszystkich rekordów
- używaj polecenia explain()
  - analogicznie do Explain Plan z relacyjnych baz
- korzystaj z indeksów
  - i zapytań, które korzystają z indeksów – sprawdzaj zapytania poleceniem explain()
- zapewnij uniform distribution of shard keys
- range-based sharding wymaga dobrej znajomości zapytań i danych, łatwiej użyć hash-based sharding

Literatura

- P. J. Sadalage, M. Fowler, NoSQL Distilled: A Brief Guide to the Emerging World of Polyglot Persistence, …, 2013
- K.Banker, MongoDB in Action, Manning 2012
- E.Redmond, J.Wilson, Seven Databases in Seven Weeks, The Pragmatic Bookshelf 2012
- K.Chodorow, MongoDB: The Definitive Guide, O'Reilly 2013
- R.Copeland, MongoDB: Applied Design Patterns, O'Reilly 2013
- K.Seguin, The Little MongoDB Book (available online: http://openmymind.net/mongodb.pdf)
- K.Chodorow, 50 Tips & Tricks for MongoDB Developers, O'Reilly 2011
- K.Chodorow, Scaling MongoDB, O'Reilly 2011

Laboratorium

- tworzenie zapytań oraz implementacja kodu w Javie do interakcji z MongoDB