



# Bazy danych - NoSQL

Autor:

Prawa do korzystania z materiałów posiada Software Development Academy



# Zasady na dzisiaj



Co ~50 minut krótka przerwa. Jeżeli chcecie wcześniej lub później - dajcie znać.



Punktualność, wracamy na czas! Nie czekamy na spóźnialskich.



Zwracamy się do siebie po imieniu. Gorąca prośba o podpisanie kartek z imionami.

# Cześć!



# Paweł Warczyński

<https://www.linkedin.com/in/pawel-warczynski-38b8a34/>



software  
development  
academy

&

allegro

Autor: Paweł Warczyński

Prawa do korzystania z materiałów posiada Software Development Academy

# Baza danych i teoria CAP



- ▶ Baza danych i teoria CAP
- ▶ Bazy relacyjne vs NoSQL
- ▶ MongoDB
- ▶ Instalacja
- ▶ JSON
- ▶ Bazy i kolekcje
- ▶ Dodawanie nowych dokumentów
- ▶ Aktualizacja całego dokumentu
- ▶ Szukanie z wykorzystywaniem kryteriów
- ▶ Modyfikacja dokumentów
- ▶ Usuwanie dokumentów
- ▶ Indeksy
- ▶ Integracja z JAVA
- ▶ MongoDB University
- ▶ Podsumowanie



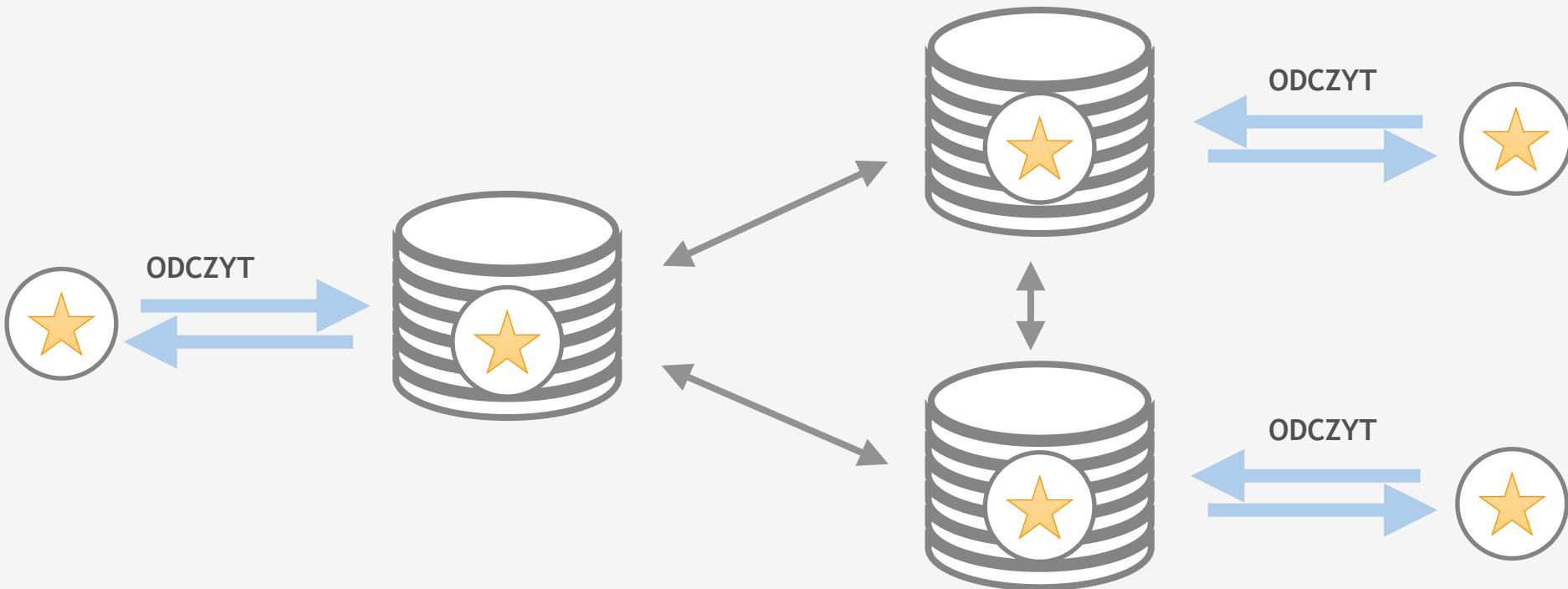
# Baza danych

**Baza danych - to uporządkowany zbiór danych z pewnej dziedziny tematycznej, zorganizowany w sposób umożliwiający ich wyszukiwanie według zadanych kryteriów.**





# Teoria CAP





## Consistency

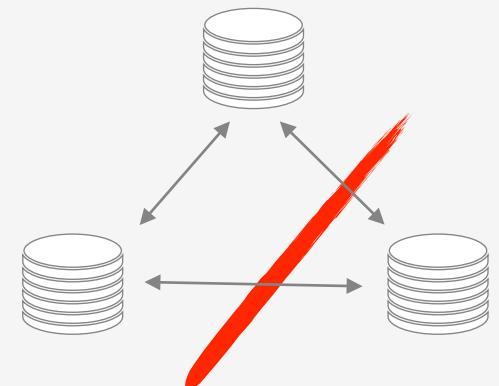
Every read receives the most recent write or an error

## Availability

Every request (write, read) receives a (non-error) response - without guarantee that it contains the most recent write

## Partition Tolerance

The system continues to operate despite an arbitrary number of messages being dropped (or delayed) by the network between nodes





# Teoria CAP

## Partition Tolerance

W systemach rozproszonych działanie sieci nigdy nie jest gwarantowane.  
Z tego powodu zazwyczaj chcemy być odporni na jej awarie.

Consistency

albo

Availability

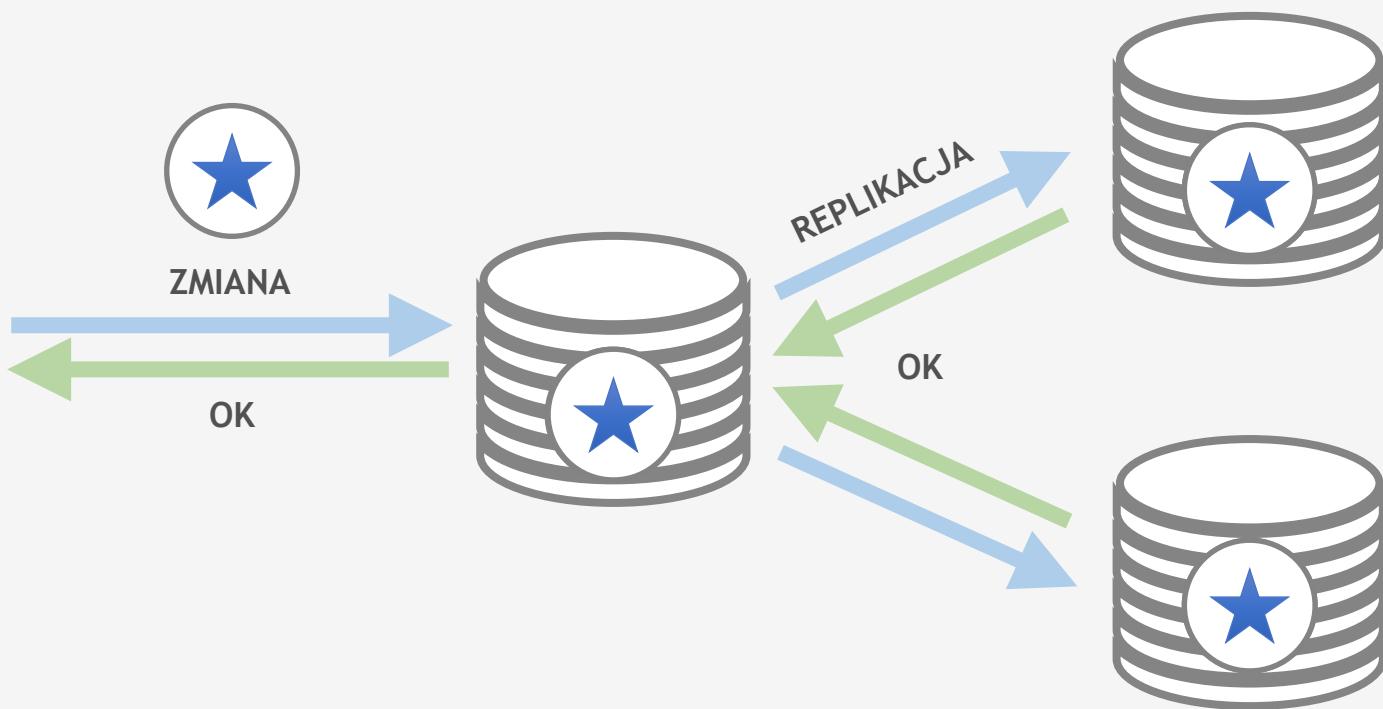


# Teoria CAP (CP)

Consistency

Availability

Partition Tolerance



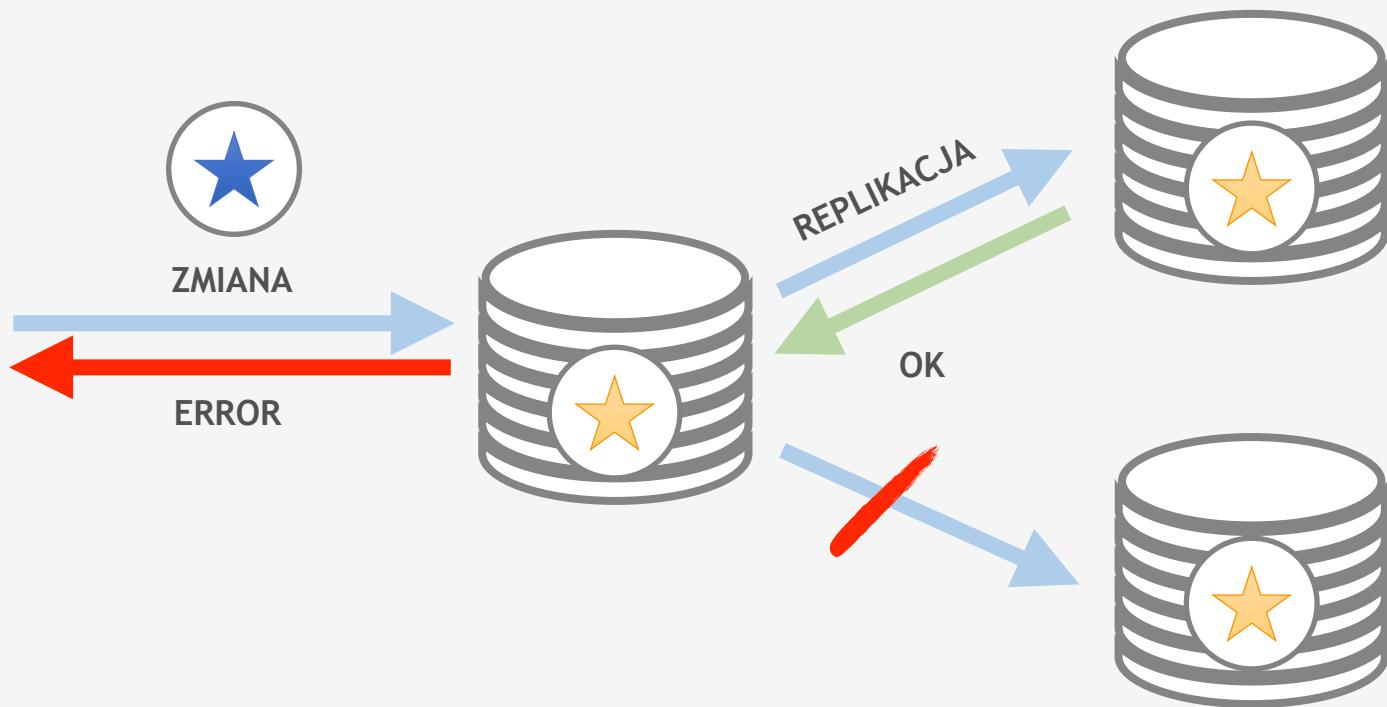


# Teoria CAP (CP)

Consistency

Availability

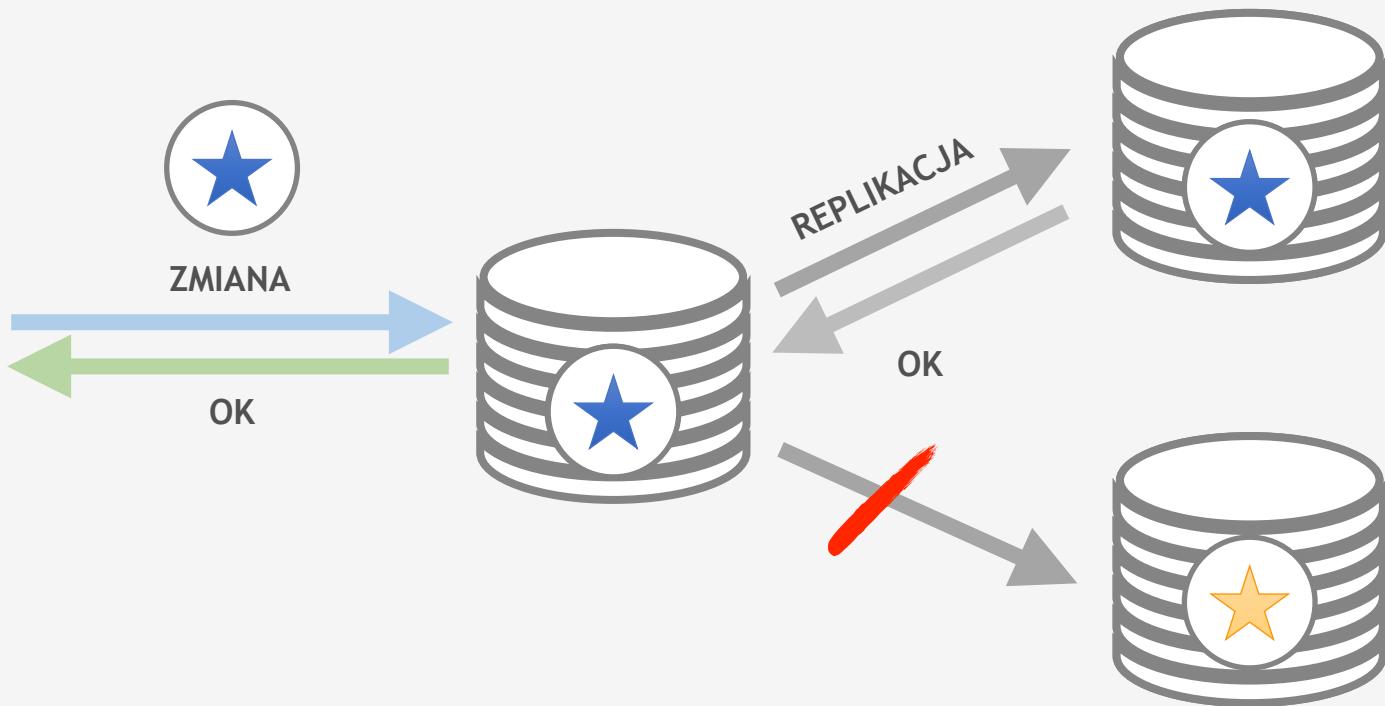
Partition Tolerance





# Teoria CAP (AP)

Consistency      Availability      Partition Tolerance



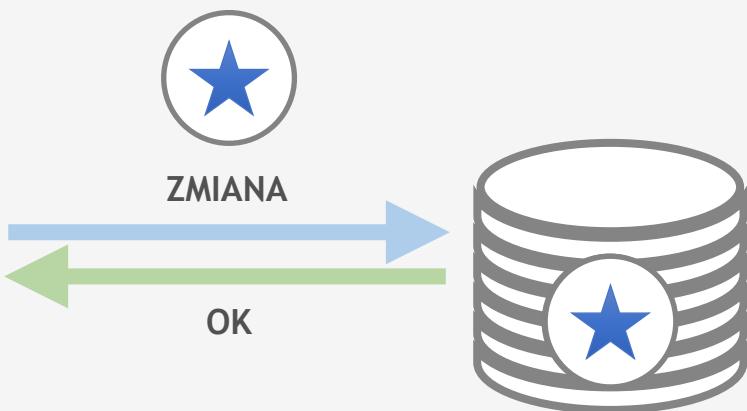


# Teoria CAP (CA)

Consistency

Availability

Partition Tolerance



Najbardziej trywialnym przykładem bazy CA jest pojedyńcza instancja.



# Teoria CAP

*“In any distributed data store, there exists some non-total failure mode that means it must either sacrifice sequential consistency or 100% availability”*

*“W rozproszonej bazie danych, w celu obsłużenia błędów, musimy poświęcić albo spójność albo dostępność.”*

<https://www.quora.com/Whats-the-difference-between-CA-and-CP-systems-in-the-context-of-CAP-Consistency-Availability-and-Partition-Tolerance>

# Bazy relacyjne vs NoSQL



- Baza danych i teoria CAP
- Bazy relacyjne vs NoSQL
- MongoDB
- Instalacja
- JSON
- Bazy i kolekcje
- Dodawanie nowych dokumentów
- Aktualizacja całego dokumentu
- Szukanie z wykorzystywaniem kryteriów
- Modyfikacja dokumentów
- Usuwanie dokumentów
- Indeksy
- Integracja z JAVA
- MongoDB University
- Podsumowanie



# Relacyjna baza danych

Relacyjną bazą danych nazywamy bazę danych w postaci tabel **połączonych relacjami**.

Customers

<b>id</b>	<b>name</b>
3	Kowalski
4	Nowak
5	Taczanowski
6	Warczyński

Transactions

<b>id</b>	<b>user_id</b>	<b>item</b>	<b>amount</b>
343	3	Stół	230
344	3	Krzesło	99
345	4	Kubek	29
346	5	Ręcznik	39



# Relacyjna baza danych

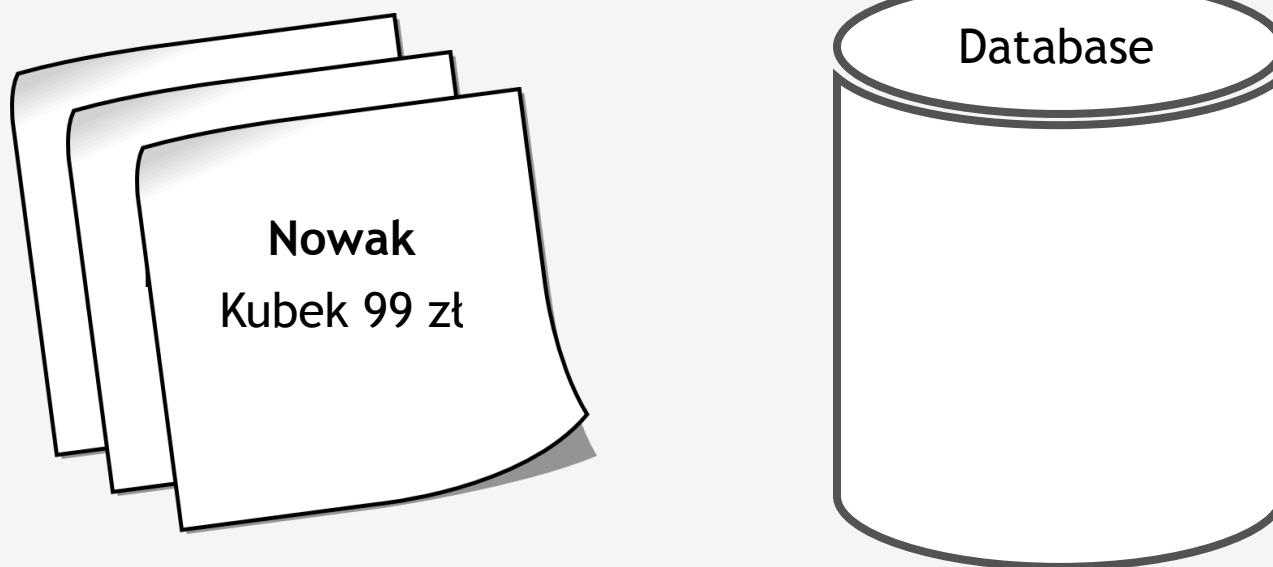
Relacyjną bazą danych nazywamy bazę danych w postaci tabel **połączonych relacjami**.



# NoSQL (non SQL / non relational)



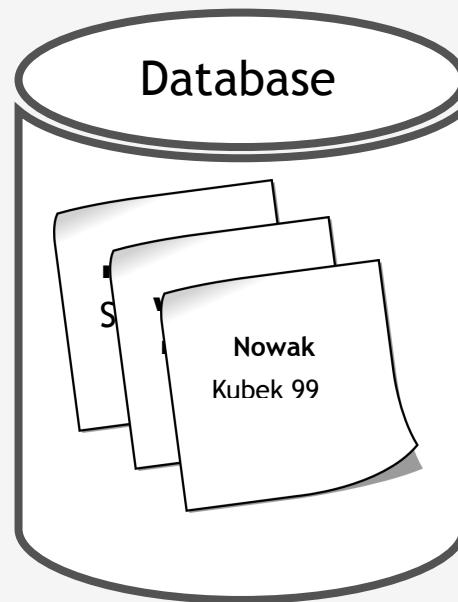
NoSQL - baza danych zapewniająca mechanizm do przechowywania i wyszukiwania danych modelowanych w **innym sposobem niż relacje tabelaryczne** używane w relacjach baz danych SQL.



# NoSQL (non SQL / non relational)



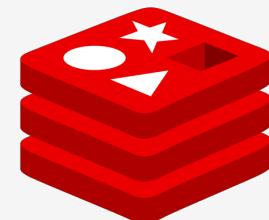
NoSQL - baza danych zapewniająca mechanizm do przechowywania i wyszukiwania danych modelowanych w **innym sposobem niż relacje tabelaryczne** używane w relacjach baz danych SQL.





# NoSQL (non SQL / non relational)

NoSQL - baza danych zapewniająca mechanizm do przechowywania i wyszukiwania danych modelowanych w **innym sposobem niż relacje tabelaryczne** używane w relacjach baz danych SQL.



**redis**

# NoSQL (non SQL / non relational)



W 2009 roku Johan Oskarsson szukał dla spotkania Hadoop w San Francisco nazwy, która stanowiłaby jednocześnie dobry hashtag Twittera; krótkiej, łatwej do zapamiętania i takiej, dla której Google nie wyświetlałby zbyt wielu wyników, tak aby wyszukiwanie po nazwie pozwalało łatwo znaleźć spotkanie.

Poprosił o sugestie na kanale IRC #cassandra i spośród propozycji wybrał nazwę NoSQL, którą zgłosił Eric Evans z RackSpace. Spotkanie dotyczyło otwartych, rozproszonych, nierelacyjnych systemów baz danych. Nazwa NoSQL była negatywna i nie do końca pasowała do opisywanych systemów jednak termin ten rozpowszechnił się na ogólnoświatową sieć i stał się de facto nazwa trendu w IT.

Niektórzy zwolennicy systemów NoSQL twierdzą, nazwa NoSQL nie oznacza „nie” dla SQL a raczej oznacza „Not Only SQL” (z ang. nie tylko SQL), podkreślając możliwość obsługi języka zapytań SQL

<https://pl.wikipedia.org/wiki/NoSQL>

# NoSQL (non SQL / non relational)

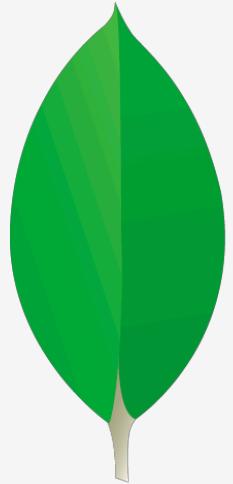


# MongoDB



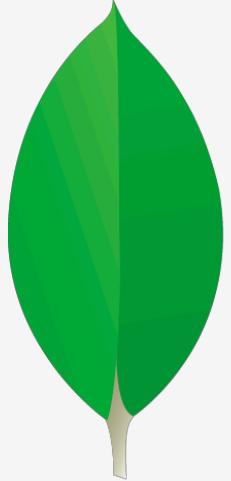
- Baza danych i teoria CAP
- Bazy relacyjne vs NoSQL
- MongoDB
- Instalacja
- JSON
- Bazy i kolekcje
- Dodawanie nowych dokumentów
- Aktualizacja całego dokumentu
- Szukanie z wykorzystywaniem kryteriów
- Modyfikacja dokumentów
- Usuwanie dokumentów
- Indeksy
- Integracja z JAVA
- MongoDB University
- Podsumowanie

# MongoDB



mongoDB®

# MongoDB



# mongoDB®

Availability

vs

Consistency

NoSQL (non relational)

vs

Relational Database

# Instalacja



- Baza danych i teoria CAP
- Bazy relacyjne vs NoSQL
- MongoDB
- Instalacja
- JSON
- Bazy i kolekcje
- Dodawanie nowych dokumentów
- Aktualizacja całego dokumentu
- Szukanie z wykorzystywaniem kryteriów
- Modyfikacja dokumentów
- Usuwanie dokumentów
- Indeksy
- Integracja z JAVA
- MongoDB University
- Podsumowanie

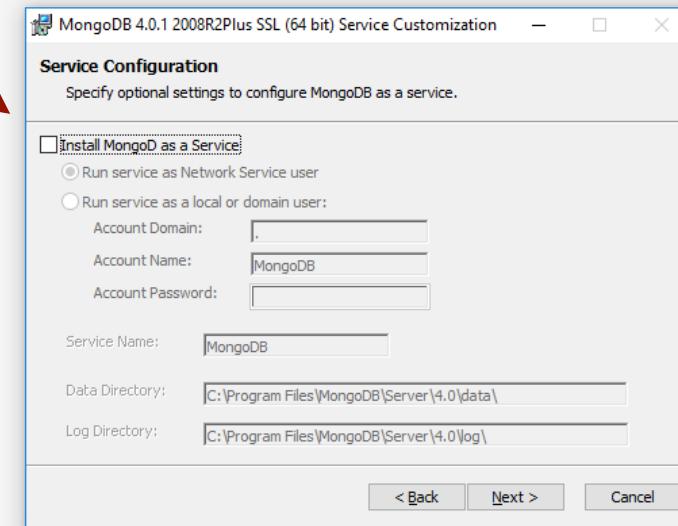
# Instalacja MongoDB (standard)



1. Pobieramy bazę danych z: <https://www.mongodb.com/download-center> zakładka “Community Server”

2. Instalujemy

- opcja **Complete**
- podczas kroku **Service Configuration** **odznaczamy** **Install MongoDB as a Service**
- instalator zaproponuje zainstalowanie **MongoDB Compass**. Narzędzie to nie będzie potrzebne podczas warsztatów



5. Tworzymy katalog c:\data\db

6. Uruchamiamy

```
c:\>"c:\Program Files\MongoDB\Server\4.0\bin\mongod.exe"
```

# Pierwsze połaczenie do bazy danych (standard)



```
c:\>"c:\Program Files\MongoDB\Server\4.0\bin\mongo.exe"      localhost:27017  
                                                               default
```

C:\WINDOWS\system32\cmd.exe - "c:\Program Files\MongoDB\Server\4.0\bin\mongo.exe"

```
c:\>  
c:\>"c:\Program Files\MongoDB\Server\4.0\bin\mongo.exe"  
MongoDB shell version v4.0.1  
connecting to: mongodb://127.0.0.1:27017  
MongoDB server version: 4.0.1  
Welcome to the MongoDB shell.
```

```
> show dbs  
> db.stats()
```

MongoDB shell

WybierzC:\WINDOWS\system32\cmd.exe - "c:\Program Files\MongoDB\Server\4.0\bin\mongo.exe"

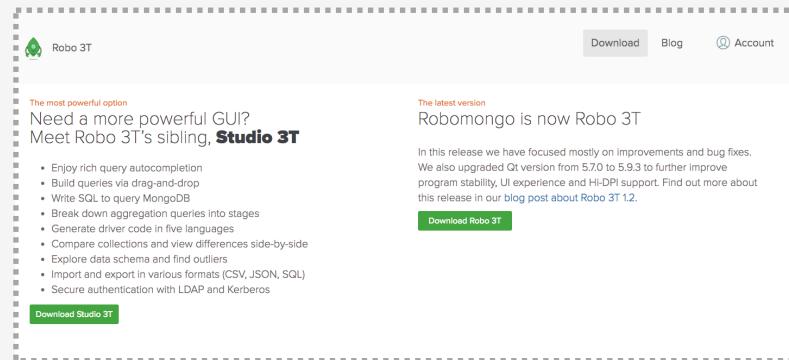
```
> show dbs  
admin    0.000GB  
config   0.000GB  
local    0.000GB  
>
```

# Robo 3T (standard)



**Robo 3T** - graficzny interfejs umożliwiający pracę z bazą danych MongoDB

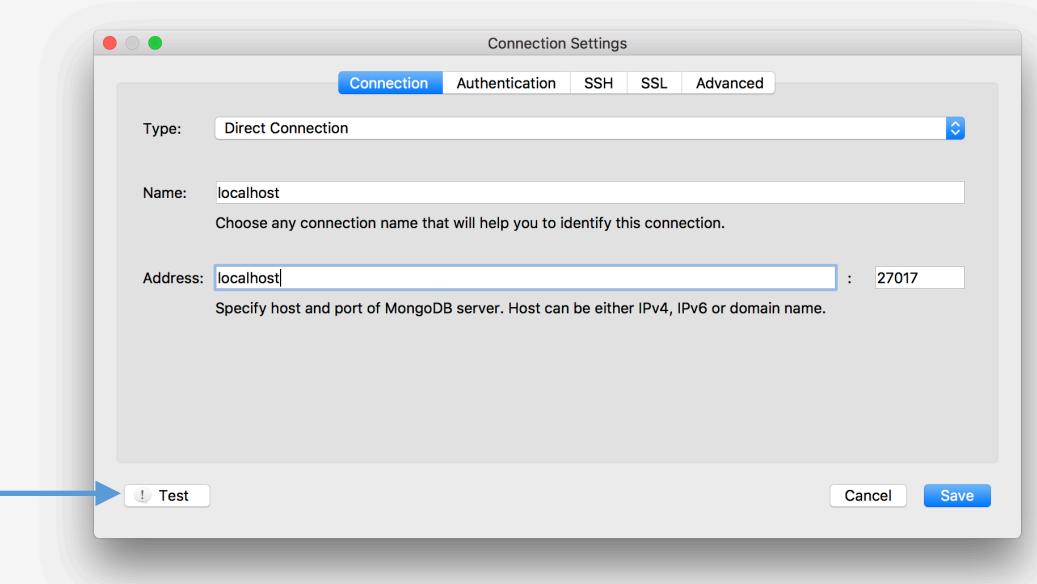
1. Pobieramy narzędzie z:  
<https://robomongo.org/download>



2. Instalujemy

3. Uruchamiamy

4. Konfigurujemy połączenie do  
localhost port 27017



Zanim zapiszecie konfiguracje  
kliknijcie przycisk **Test**



# Robo 3T interfejs

The screenshot shows the Robo 3T application window. On the left is a sidebar with a tree view of databases: 'localhost (4)' (System, config, mongo\_workshop), 'mongo\_workshop' (Collections (1) - drivers), and 'Users'. The main area has two panes. The top pane is a terminal window titled 'Welcome' showing a MongoDB shell session:localhost:27017 mongo\_workshop
var invitation = "Hello World"
print(invitation)
db.getCollection('drivers').find({})Output below the session shows 'Hello World'. The bottom pane is titled 'drivers' and displays the contents of the 'drivers' collection in JSON format:/\* 1 \*/
{
 "\_id" : "3ad3ad8a17",
 "first\_name" : "Anna",
 "last\_name" : "Kowalska",
 "age" : 26.0,
 "vehicles" : [
 {
 "number" : "P03493",
 "color" : "blue",
 "seats" : 2.0
 }
 ]
}A small orange arrow points from the text 'Interaktywna, obsługująca JavaScript, konsola do MongoDB' to the top pane of the application.

Interaktywna, obsługująca  
JavaScript, konsola do MongoDB



# Robo 3T interfejs

```
* var invitation = "Hello World" print(invitation)
var invitation = "Hello World"
print(invitation)

db.getCollection('drivers').find({})

0 sec.

Hello World

/* 1 */
{
  "_id" : "3ad3ad8a17",
  "first_name" : "Anna",
  "last_name" : "Kowalska",
  "age" : 26.0,
  "vehicles" : [
    {
      "number" : "P03493",
      "color" : "blue",
      "seats" : 2.0
    }
  ]
}
```

← Wynik pierwszej operacji

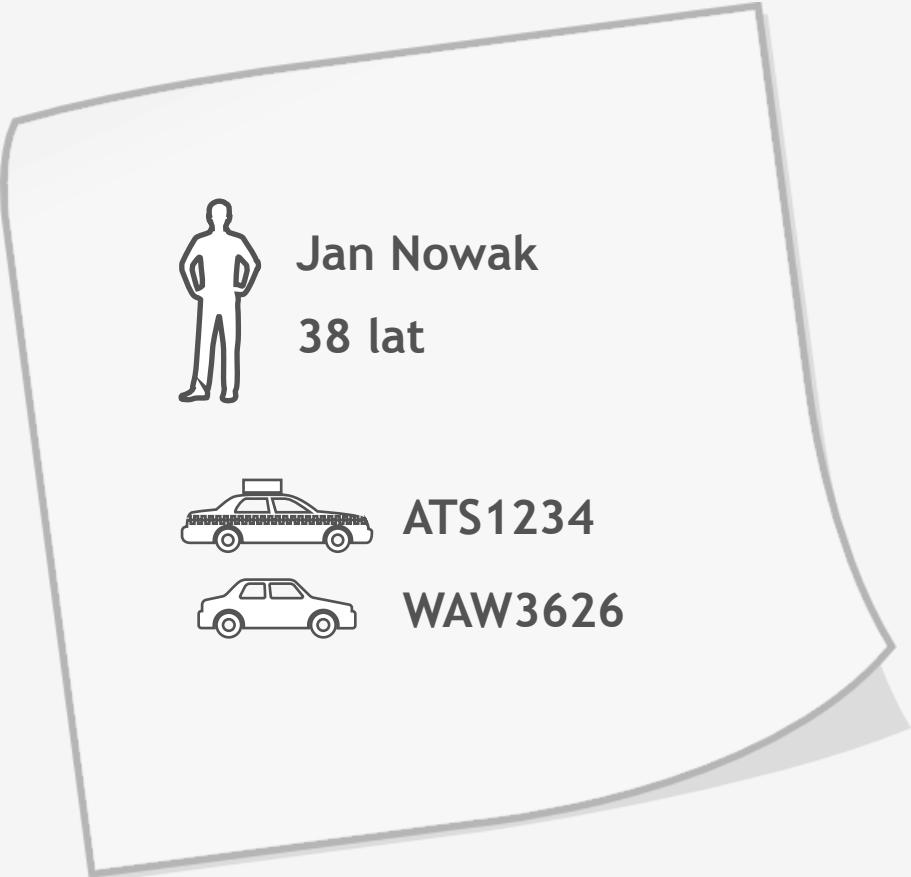
← Wynik drugiej operacji

# JSON



- Baza danych i teoria CAP
- Bazy relacyjne vs NoSQL
- MongoDB
- Instalacja
- JSON
- Bazy i kolekcje
- Dodawanie nowych dokumentów
- Aktualizacja całego dokumentu
- Szukanie z wykorzystywaniem kryteriów
- Modyfikacja dokumentów
- Usuwanie dokumentów
- Indeksy
- Integracja z JAVA
- MongoDB University
- Podsumowanie

# MongoDB - baza dokumentowa



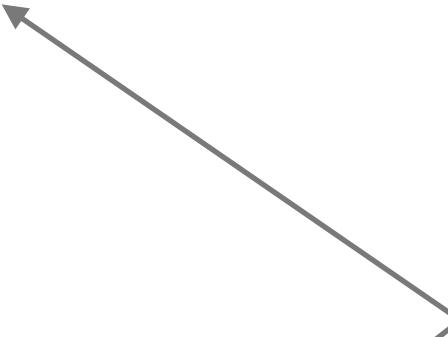
**Format: JSON**

```
{  
  "first_name" : "Jan",  
  "last_name" : "Nowak",  
  "age" : 38,  
  "vehicles" : [  
    "ATS1234",  
    "WAW3626"  
  ]  
}
```

# JSON



{



Początek i koniec obiektu

}

# JSON



```
{  
    "first_name" : "Jan"  
}  
}
```



Znak rozdzielający klucz od wartości

# JSON



```
{  
    "first_name" : "Jan",  
    "last_name" : "Nowak"  
}
```



Przecinek rozdziela kolejne bloki  
klucz : wartość

# JSON



```
{  
    "first_name" : "Jan",  
    "last_name" : "Nowak",  
    "vehicles" : [  
        ]  
    ]  
}
```

Początek i koniec listy  
obiektów

# JSON

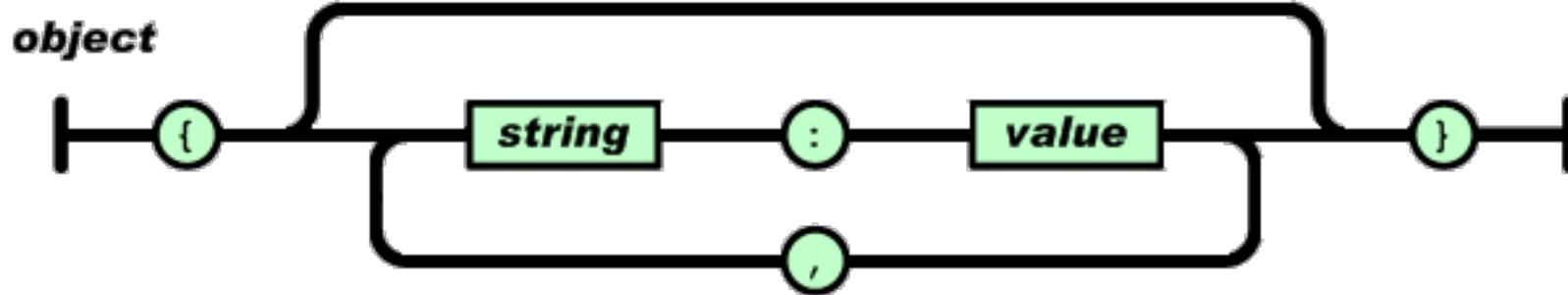


```
{  
    "first_name" : "Jan",  
    "last_name" : "Nowak",  
    "vehicles" : [  
        "ATS1234",  
        "WAW3626"  
    ]  
}
```

# JSON - pusta lista

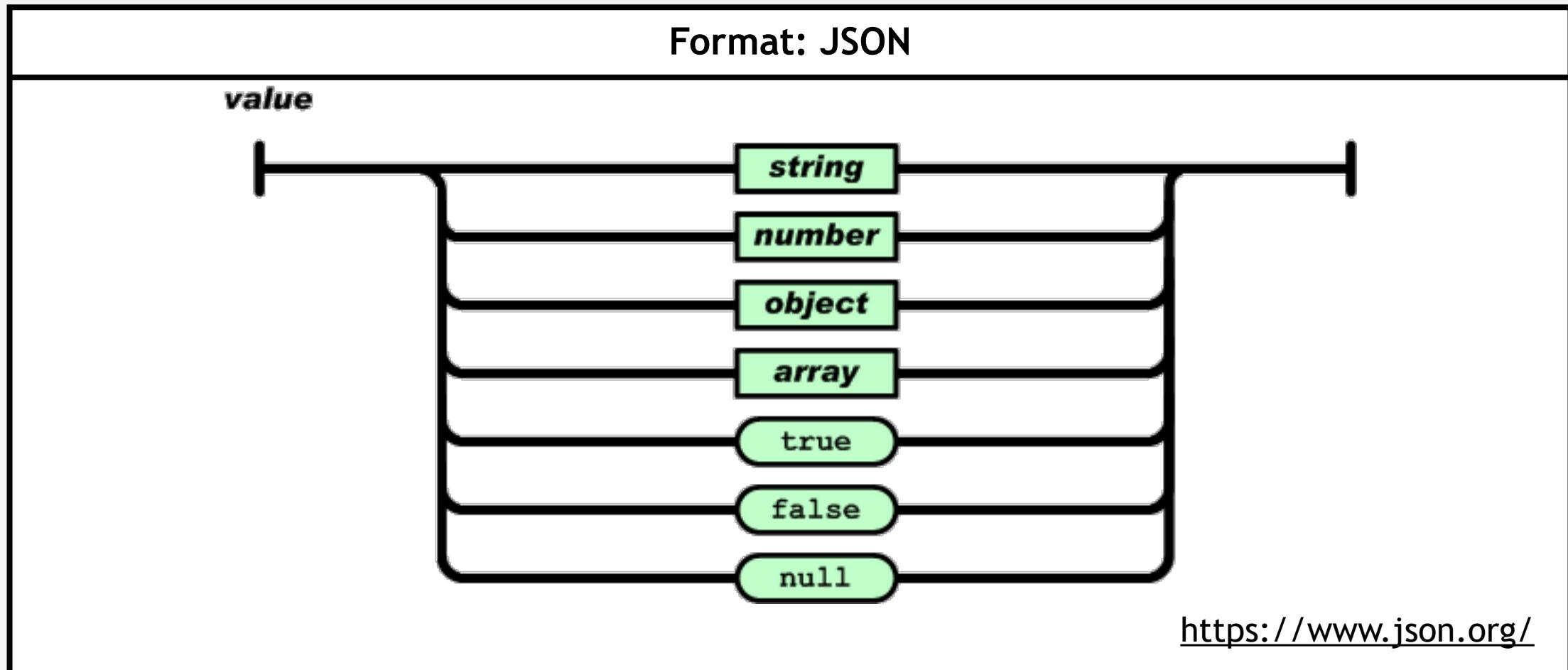


## Format: JSON



<https://www.json.org/>

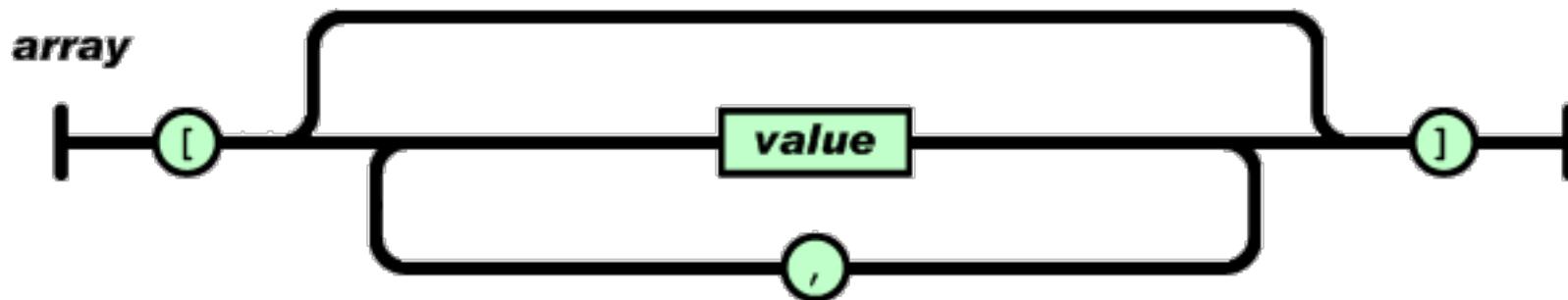
# JSON - pusta lista



# JSON - pusta lista



Format: JSON



<https://www.json.org/>

# JSON - pusta lista



```
{  
    "first_name" : "Jan",  
    "last_name" : "Kowalski",  
    "vehicles" : []  
}
```



# JSON - obiekt w obiekcie

```
{  
    "first_name" : "Adam",  
    "last_name" : "Tomkiewicz",  
    "statistics" : {  
        "years_of_experience": 45,  
        "number_of_accidents": 0  
    }  
}
```

Normalny obiekt  
{ klucz : wartość, ... }



# JSON - lista obiektów

```
{  
    "first_name" : "Jan" ,  
    "last_name" : "Nowak" ,  
    "vehicles" : [ .....  
        { "number": "ATS1234" , "color": "pink"} ,  
        { "number": "WAR3626" , "color": "orange"}  
    ] .....  
}
```

Lista  
obiektów



**Binarna reprezentacja JSON'a**

Zawiera dodatkowe typy danych (np.  
Date, BinData).

**Wykorzystywany w MongoDB ze  
względów wydajnościowych.**

# BSON { }

01010100  
11101011  
10101110  
01010101

BSON [*bee · sahn*], short for Binary [JSON](#), is a binary-encoded serialization of JSON-like documents. Like JSON, BSON supports the embedding of documents and arrays within other documents and arrays. BSON also contains extensions that allow representation of data types that are not part of the JSON spec. For example, BSON has a Date type and a BinData type.

BSON can be compared to binary interchange formats, like [Protocol Buffers](#). BSON is more "schema-less" than Protocol Buffers, which can give it an advantage in flexibility but also a slight disadvantage in space efficiency (BSON has overhead for field names within the serialized data).

BSON was designed to have the following three characteristics:

**1. Lightweight**

Keeping spatial overhead to a minimum is important for any data representation format, especially when used over the network.

**2. Traversable**

BSON is designed to be traversed easily. This is a vital property in its role as the primary data representation for [MongoDB](#).

**3. Efficient**

Encoding data to BSON and decoding from BSON can be performed very quickly in most languages due to the use of C data types.

[specification](#)[implementations](#)[FAQ](#)[discussion](#)



**Ćwiczenie:** Stwórzcie cztery dokumenty w formacie (JSONy)

# Karol Tomkiewicz (32 lata)  
Ma dwa zielone samochody osobowe.

# Anna Kowalska (26 lata)  
Ma jeden mały, dwu-osobowy, niebieski samochód.

# Michał Piwowarski (43 lata)  
Ma jednego czarnego vana na 10 osób.

# Alicja Jarosz (34 lata)  
Ma jeden żółty samochód osobowy.

Walidacja: <https://jsonlint.com>

```
{  
    "first_name" : "Anna",  
    "last_name" : "Kowalska",  
    "age" : 26,  
    "vehicles" : [  
        {  
            "number" : "PO3493",  
            "color" : "blue",  
            "seats" : 2  
        }  
    ]  
}
```

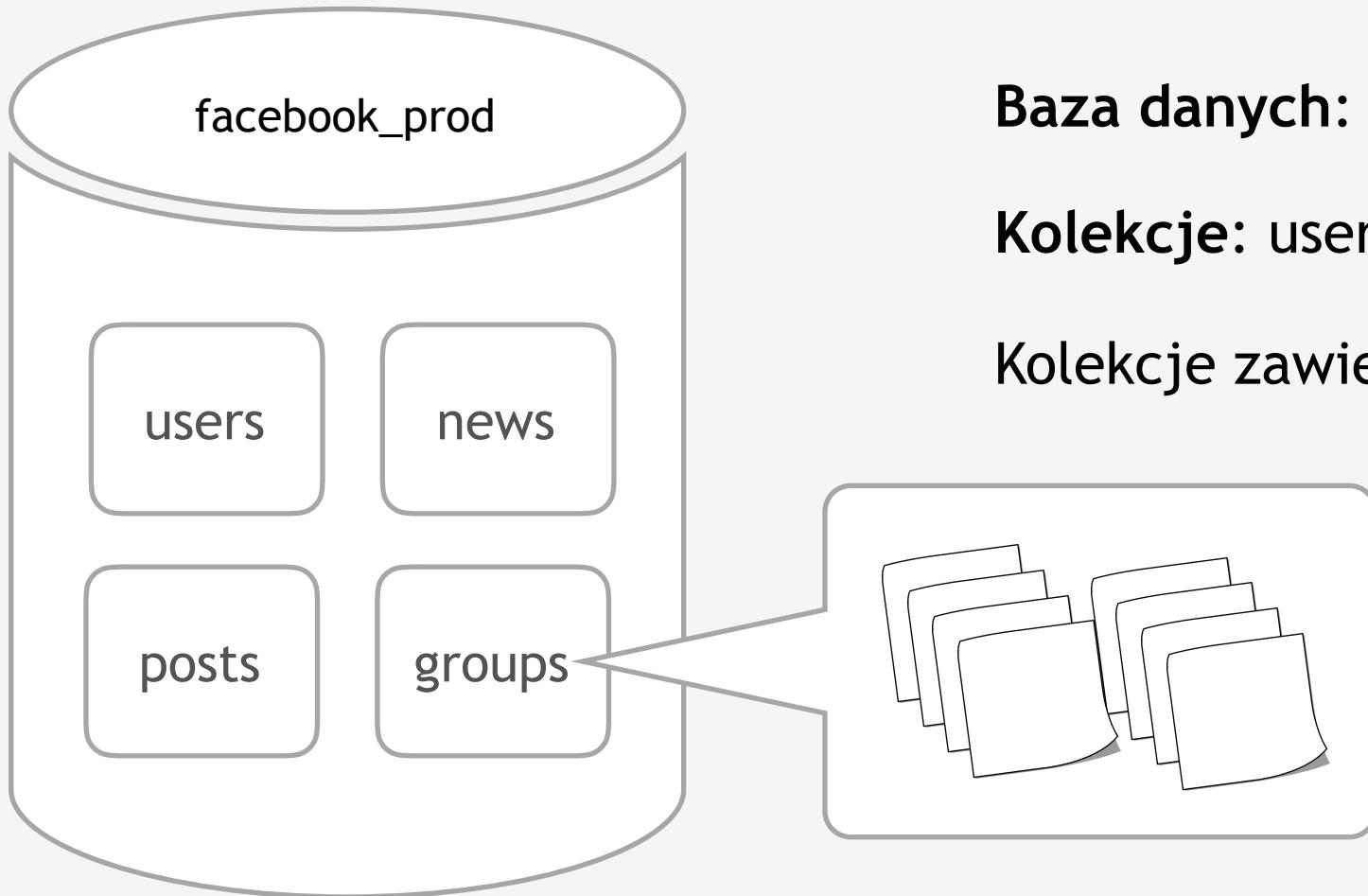
# Bazy i kolekcje



- Baza danych i teoria CAP
- Bazy relacyjne vs NoSQL
- MongoDB
- Instalacja
- JSON
- Bazy i kolekcje
- Dodawanie nowych dokumentów
- Aktualizacja całego dokumentu
- Szukanie z wykorzystywaniem kryteriów
- Modyfikacja dokumentów
- Usuwanie dokumentów
- Indeksy
- Integracja z JAVA
- MongoDB University
- Podsumowanie



# MongoDB - bazy i kolekcje



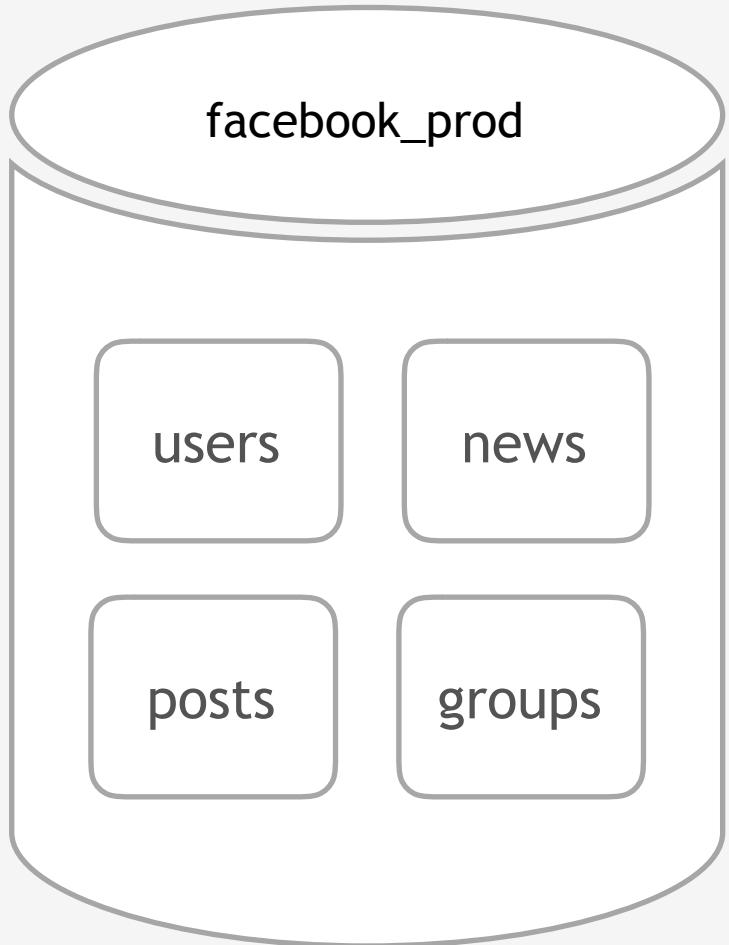
**Baza danych:** facebook\_prod

**Kolekcje:** users, news, posts, groups

**Kolekcje zawierają dokumenty**



# MongoDB - bazy i kolekcje



Lista dostępnych baz danych

```
> show dbs  
facebook_prod
```

Wybranie bazy danych

```
> use facebook_prod
```

Lista kolekcji w ramach wybranej bazy danych

```
> show collections  
users  
news  
posts  
groups
```

# Dodanie nowych dokumentów



- Baza danych i teoria CAP
- Bazy relacyjne vs NoSQL
- MongoDB
- Instalacja
- JSON
- Bazy i kolekcje
- Dodawanie nowych dokumentów
- Aktualizacja całego dokumentu
- Szukanie z wykorzystywaniem kryteriów
- Modyfikacja dokumentów
- Usuwanie dokumentów
- Indeksy
- Integracja z JAVA
- MongoDB University
- Podsumowanie

# CRUD

Create · Read · Update · Delete



- Baza danych i teoria CAP
- Bazy relacyjne vs NoSQL
- MongoDB
- Instalacja
- JSON
- Bazy i kolekcje
- Dodawanie nowych dokumentów
- Aktualizacja całego dokumentu
- Szukanie z wykorzystywaniem kryteriów
- Modyfikacja dokumentów
- Usuwanie dokumentów
- Indeksy
- Integracja z JAVA
- MongoDB University
- Podsumowanie



# Operacja insert vs save

```
> db.nazwakolekcji.insert(DOKUMENT)
```

Dodaje nowy **DOKUMENT** do bazy danych.

Jeżeli **DOKUMENT** nie zawiera pola **\_id** zostanie ono automatycznie wygenerowane.

Jeżeli **DOKUMENT** zawiera pole **\_id** zostanie ono użyte jako identyfikator.

Jeżeli dokument o podanym **\_id** już istnieje dokument **nie zostanie dodany**

```
> db.nazwakolekcji.save(DOKUMENT)
```

Dodaje lub aktualizuje **DOKUMENT** w bazie danych

Jeżeli **DOKUMENT** nie zawiera pola **\_id** zostanie ono automatycznie wygenerowane.

Jeżeli **DOKUMENT** zawiera pole **\_id** zostanie ono użyte jako identyfikator.

Jeżeli dokument o podanym **\_id** już istnieje dokument zostanie **zaktualizowany**.

# Zapisanie danych do bazy

ĆWICZENIE



1

```
> db.drivers.insert({  
    "first_name" : "Anna",  
    "last_name" : "Kowalska",  
    "age" : 26,  
    "vehicles" : [  
        {  
            "number" : "PO3493",  
            "color" : "blue",  
            "seats" : 2  
        }  
    ]  
})
```

3

```
> db.drivers.insert({ . . . })
```

4

```
> db.drivers.insert({ . . . })
```

5

```
> db.drivers.insert({ . . . })
```

Po dodaniu dokumentów, za pomocą komendy **find** można podejrzeć je w bazie danych

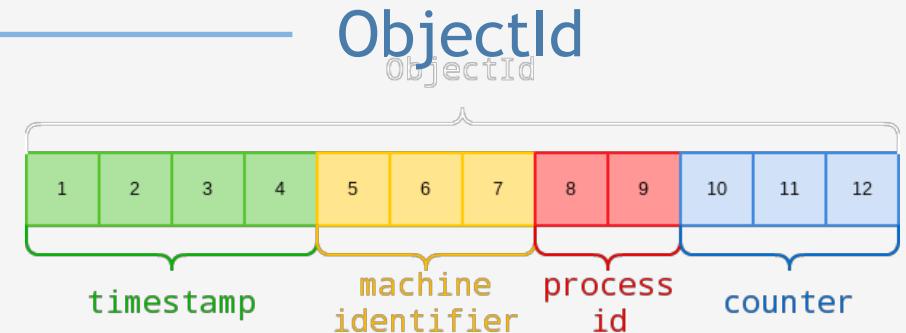
6

```
> db.drivers.find()
```



# ObjectId

```
> db.drivers.find()  
{  
    "_id" : ObjectId("5b5e2f89529592926f70f022") ,  
    "first_name" : "Anna" ,  
    "last_name" : "Kowalska" ,  
    "age" : 26 ,  
    "vehicles" : [  
        {  
            "number" : "PO3493" ,  
            "color" : "blue" ,  
            "seats" : 2  
        }  
    ]  
} ,  
... +3 dokumenty
```



4 pierwsze bajty zawierają czas stworzenia dokumentu

3 kolejne to identyfikator maszyny

2 kolejne to identyfikator procesu

3 ostatnie bajty to licznik zaczynający się od losowej liczby

# ObjectId - data utworzenia

ĆWICZENIE



**Ćwiczenie:** za pomocą komendy `getTimestamp()` wywołanej na obiekcie `ObjectId` pobierzcie datę utworzenia identyfikatora.

```
> ObjectId("5b5e2f89529592926f70f022").getTimestamp()  
ISODate("2018-07-30T12:34:03Z")
```

# Aktualizacja całego dokumentu



- Baza danych i teoria CAP
- Bazy relacyjne vs NoSQL
- MongoDB
- Instalacja
- JSON
- Bazy i kolekcje
- Dodawanie nowych dokumentów
- Aktualizacja całego dokumentu
- Szukanie z wykorzystywaniem kryteriów
- Modyfikacja dokumentów
- Usuwanie dokumentów
- Indeksy
- Integracja z JAVA
- MongoDB University
- Podsumowanie



**Ćwiczenie:** Ania kupiła nowy samochód. Zaktualizujmy informacje w bazie danych.

```
> db.drivers.save ( {  
    "_id" : ObjectId("5b5e2f89529592926f70f022") ,  
    "first_name" : "Anna" ,  
    "last_name" : "Kowalska" ,  
    "age" : 26 ,  
    "vehicles" : [  
        { "number" : "PO3493" , "color" : "blue" , "seats" : 2 } ,  
        { "number" : "PO9831" , "color" : "black" , "seats" : 5 }  
    ]  
} )
```



**Ćwiczenie:** Ania wyszła za mąż - zaktualizujmy nazwisko na "Nowak"

```
> db.drivers.save ( {  
    "_id" : ObjectId("5b5e2f89529592926f70f022") ,  
    "first_name" : "Anna" ,  
    "last_name" : "Nowak" ,  
    "age" : 26 ,  
    "vehicles" : [  
        { "number" : "PO3493" , "color" : "blue" , "seats" : 2 } ,  
        { "number" : "PO9831" , "color" : "black" , "seats" : 5 }  
    ]  
} )
```



**Ćwiczenie:** Michał został złapany przez policję za nadmierną prędkość. Dodaj pole **points** z wartością **5**.

```
> db.drivers.save ({  
    "_id" : ObjectId("5b688099bfbc096efb08da6a") ,  
    "first_name" : "Michał" ,  
    "last_name" : "Piwowarski" ,  
    "age" : 43 ,  
    "points" : 5 ,  
    "vehicles" : [  
        { "number" : "PO3400" , "color" : "black" , "seats" : 15 }  
    ]  
})
```

# Szukanie z wykorzystaniem kryteriów



- Baza danych i teoria CAP
- Bazy relacyjne vs NoSQL
- MongoDB
- Instalacja
- JSON
- Bazy i kolekcje
- Dodawanie nowych dokumentów
- Aktualizacja całego dokumentu
- Szukanie z wykorzystywaniem kryteriów
- Modyfikacja dokumentów
- Usuwanie dokumentów
- Indeksy
- Integracja z JAVA
- MongoDB University
- Podsumowanie



# Import danych

Zimportujcie do bazy danych informacje o kierowcach.

Dane: [drivers.json](#)

```
c:\>"c:\Program Files\MongoDB\Server\4.0\bin\mongoimport.exe"  
--db=mongo_workshop  
--collection=drivers  
--drop  
--file=c:\xxxxxx\drivers.json  
# baza danych: mongo_workshop  
# kolekcja: drivers  
# jeżeli kolekcja drivers istnieje to zostanie wcześniej wyczyszczona  
# ścieżka do pliku z danymi
```

The screenshot shows a Windows Command Prompt window titled 'C:\WINDOWS\system32\cmd.exe'. The command entered is:

```
C:\Users\Warek>"c:\Program Files\MongoDB\Server\4.0\bin\mongoimport.exe" --db=mongo_workshop --collection=drivers  
--drop --file="c:\Users\Warek\Downloads\drivers.json"
```

The output of the command is displayed below the command line:

```
2018-08-18T21:19:30.435+0200      connected to: localhost  
2018-08-18T21:19:30.443+0200      dropping: mongo_workshop.drivers  
2018-08-18T21:19:31.515+0200      imported 1013 documents
```

The command prompt then returns to the user's directory:

```
C:\Users\Warek>
```



# Kryteria

```
{ "nazwa pola" : "wyrażenie" }
```

```
{
    "nazwa pola" : "wyrażenie",
    "inne pole" : "wyrażenie",
    (...)
```



AND



# Kryteria

Kryteria są używane w wielu miejscach

```
> db.drivers.find({ "points": 5 })
```

```
> db.drivers.find({ "points": { $gt: 20 } })
```

```
> db.drivers.count({ "last_name": "Kowalski" })
```

```
> db.drivers.count({ "age": 18, "points": { $gt: 0 } })
```



# Kryteria

Kryteria są używane w wielu miejscach

```
> db.drivers.remove( { "_id": ObjectId("5b5e2f89529592926f70f022") } )
```

```
> db.drivers.remove( { "points": { $gt: 20 } } )
```

```
> db.drivers.update(  
    { "_id": ObjectId("5b5e2f89529592926f70f022") } ,  
    { $set: { "points": 17 } }  
)
```



# \$gt, \$lt, \$gte, \$lte, \$ne

```
> db.drivers.find( { "points": { $gte: 20 } } )
```

**\$gt** większy niż (greater than)

**\$gte** większy niż lub równy (greater than or equal)

**\$lt** mniejszy niż (lower than)

**\$lte** mniejszy niż lub równy (lower than or equal)

**\$ne** nie równy niż (not equal)

# \$gt, \$lt, \$gte, \$lte, \$ne

ĆWICZENIE



```
> db.drivers.find({ "points": { $gte: 20 } })
```

```
> db.drivers.count({ "points": 0 })
```

**\$gt      \$gte      \$ne      \$lt      \$lte**

**Ćwiczenie 1:** Ilu kierowców ma na imię "Marta"? (32) ← liczba znalezionych dokumentów

**Ćwiczenie 2:** Ilu kierowców ma 60 lat lub więcej? (490)

**Ćwiczenie 3:** Ilu kierowców **nie** ma na imię "Jan" (985)

**Ćwiczenie 4:** Wyświetl wszystkich kierowców o imieniu "Jan" i nazwisku "Nowak" (2)

↑  
**zadanie dodatkowe o wyższym poziomie trudności**



# sort, skip, limit

```
> db.drivers.find({ "points": { $gt: 20 } }) # znajdź kierowców, którzy mają ponad 20 punktów
      .sort({ "points": -1 })                 # na początku ustaw tych o największej liczbie punktów
      .skip(2)                                # pomiń pierwszych dwóch
      .limit(10)                             # i wyświetl kolejnych 10-ciu
```

**sort (nazwa-pola: kierunek)** → 1 rosnąco (ASC)  
→ -1 malejąco (DESC)

**skip (liczba-dokumentów)**

**limit (liczba-dokumentów)**



```
> db.drivers.find({ "points": { $gt: 20 } })
    .sort({ "points": -1 })
    .skip(2)
    .limit(10)
```

**Ćwiczenie 1:** Wyświetl dziesięciu kierowców, którzy mają najwięcej punktów (60, 60, ..., 59)

**Ćwiczenie 2:** Wyświetl drugą dziesiątkę kierowców, którzy mają najwięcej punktów (59, 58, ..., 56)

**Ćwiczenie 3:** Ile lat ma najstarszy Jan wśród kierowców? (J.K ma 100 lat)

**Ćwiczenie 4:** Wśród najstarszych kierowców kto jest rekordzistą pod względem punktów? (S.W.)

podpowiedź: `sort({pierwsze-pole : kierunek, drugie-pole : kierunek})`



```
> db.drivers.find({ "first_name": { $in: ["Emi", "Adam", "Ania"] } })
```

**\$in** pole ma wartość jedną z podanych w tablicy

**\$nin** pole **nie** ma wartości jednej z podanych (not in)



```
> db.drivers.find( { "first_name": { $in: ["Emi", "Adam", "Ania"] } } )
```

**Ćwiczenie 1:** Wyświetl kierowców, których nazwisko to Kaczyński lub Kukiz lub Schetyna (3)

**Ćwiczenie 2:** Ile kierowców ma 25 lub 50 lub 75 lub 100 lat? (54)

**Ćwiczenie 3:** Wyświetl kierowców o imieniu Paweł którzy **nie mają** na nazwisko Kowalski, Nowak, Koza, Kukiz. (61)

**Ćwiczenie 4:** Wyświetl kierowców którzy mają od 20 do 50 lat z pominięciem tych co mają dokładnie 30, 35, 40 i 45 lat (348)

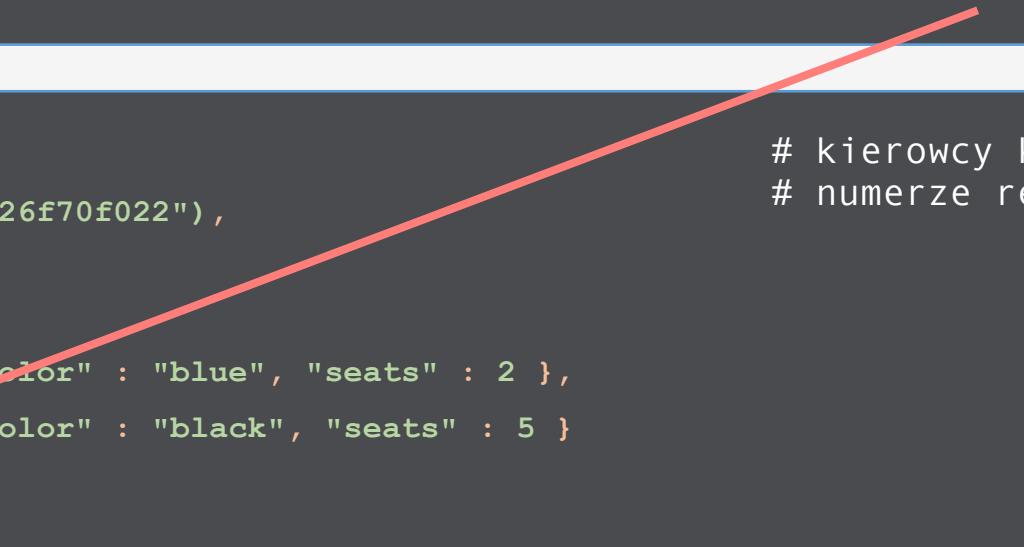


# \$elemMatch

```
> db.drivers.find( { "vehicles": { $elemMatch: { "number": "PO9831" } } } )
```

```
{  
  "_id" : ObjectId("5b5e2f89529592926f70f022") ,  
  (...)  
  "vehicles" : [  
    { "number" : "PO3493" , "color" : "blue" , "seats" : 2 } ,  
    { "number" : "PO9831" , "color" : "black" , "seats" : 5 }  
  ]  
}
```

# kierowcy którzy mają samochód o  
# numerze rejestracyjnym PO9831



**\$elemMatch**  
tylko dla list [...]

znajduję dokumenty, w których wskazana listą zawiera  
przynajmniej jeden element, który spełnia kryterium



```
> db.drivers.find( { "vehicles": { $elemMatch: { "number": "PO9831" } } } )
```

**Ćwiczenie 1:** Wyświetl kierowców którzy mają czarny samochód (338)

podpowiedź: `{"color" : "black"}`

**Ćwiczenie 2:** Wyświetl kierowców którzy mają samochód dwuosobowy (602)

podpowiedź: `{"seats" : 2.0}`

**Ćwiczenie 3:** Wyświetl kierowców którzy mają **tylko jeden** samochód dwuosobowy (67)

podpowiedź: `{ $size : x, $elemMatch: { "seats" : x } }`

**Ćwiczenie 4:** Znajdź wszystkich Grzegorzów, którzy mają dwa samochody i przynajmniej jeden z nich jest koloru czarnego. (2)

# Wypadek na drodze

ĆWICZENIE



Doszło do wypadku drogowego, w którym sprawca uciekł z miejsca zdarzenia. Świadek opisał zdarzenie w następujący sposób.



Sprawca miał od **20 do 35 lat**



Jechał samochodem osobowym (**5 osobowym**) koloru **niebieskiego**

P#####

Z tyłu samochodu była tabliczka "z drogi śledzie P..." jedzie. Końcówka imienia była zamazana ale prawdopodobnie był to **Paweł, Piotr lub Patryk**.



Po sposobie prowadzenia samochodu można było wywnioskować, że lubił przekraczać prędkość i łamać przepisy. Możliwe, że ma już zgromadzonych wiele punktów karnych.

**Wytypuj trzech najbardziej prawdopodobnych kierowców, którzy mogli popełnić to przestępstwo. (P.M., P.A., P.S.)**

# Modyfikacja dokumentów



- Baza danych i teoria CAP
- Bazy relacyjne vs NoSQL
- MongoDB
- Instalacja
- JSON
- Bazy i kolekcje
- Dodawanie nowych dokumentów
- Aktualizacja całego dokumentu
- Szukanie z wykorzystywaniem kryteriów
- Modyfikacja dokumentów
- Usuwanie dokumentów
- Indeksy
- Integracja z JAVA
- MongoDB University
- Podsumowanie



# Nadpisywanie stanem

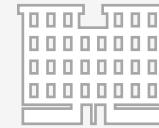


+200 zł

```
> db.accounts.find({ "_id" : ObjectId("5b5..8") })
```

```
{
  "_id" : ObjectId("5b5..8"),
  "account_balance" : 1000
}
```

```
> db.accounts.save({
  "_id" : ObjectId("5b5..8"),
  "account_balance" : 1200
})
```



+1800 zł

```
> db.accounts.find({ "_id" : ObjectId("5b5..8") })
```

```
{
  "_id" : ObjectId("5b5..8"),
  "account_balance" : 1000
}
```

```
> db.accounts.save({
  "_id" : ObjectId("5b5..8"),
  "account_balance" : 2800
})
```

Ile pieniędzy powinno być na koncie?

Ile jest po wykonaniu operacji?



# Delta



+200 zł

```
> db.accounts.update(  
  {"_id" : ObjectId("5b5...8")},  
  {$inc: {"account_balance" : 200}}  
)
```



+1800 zł

```
> db.accounts.update(  
  {"_id" : ObjectId("5b5...8")},  
  {$inc: {"account_balance" : 1800}}  
)
```

Mówimy **co ma się zmienić i o ile (delta)** zamiast jak ma wyglądać **cały obiekt po zmianie (stan)**.



# Update

```
> db.drivers.update( {KRYTERIA} , {ZMIANA} )  
↓  
{ "first_name": { $in: ["Emi", "Jan"] } } { $set: { "last_name": "Kowalska" } }  
{ "points": { $gt: 20 } } { $set: { "points": 12 } }  
{ "supervision": { $exists: true } } { $inc: { "age": 1 } }  
{ "pole": { $warunek: wartość } } { $operacja: { "pole": wartość } }
```

W zależności od tego czy potrzebujemy **kryterium** czy **definicji zmiany** zaczynamy albo od nazwy pola albo od operacji. Istnieją pojedyncze wyjątki, przykładowo: **\$or**.



# Update \$set

```
> db.drivers.update({KRYTERIA} , {ZMIANA})          # Aktualizuje tylko  
# jeden dokument!
```

```
> db.drivers.updateMany({KRYTERIA} , {ZMIANA})        # Aktualizuje wszystkie pasujące  
# do kryteriów dokumenty
```

```
> db.drivers.update(  
    {_id : ObjectId("5b5e2f89529592926f70f022") } ,  
    {$set: {last_name: "Kowalska"} }  
)
```

**\$set** Zmienia wartość wskazanego pola

# Update \$set

ĆWICZENIE



```
> db.drivers.update(  
  { "_id" : ObjectId("5b5e2f89529592926f70f022") } ,  
  { $set: { "last_name": "Kowalska" } }  
)
```

**Ćwiczenie 1:** Zmień nazwisko kierowcy Michał Piwowarski na "Kenobi"

podpowiedź: skorzystaj z update (...)

**Ćwiczenie 2:** Należy wszystkim kierowcą dodać nowe pole insurance z wartością true

podpowiedź: skorzystaj z updateMany (...), jako kryterium pusty obiekt {}

**Ćwiczenie 3:** Kierowcy z 8 punktami powinni mieć zmniejszoną liczbę punktów do 4

podpowiedź: skorzystaj z updateMany (...)



# Update \$inc

```
> db.drivers.update(  
  {"first_name": "Adam"},  
  {$inc: {"age": 1}}  
)  
# Pierwszy znaleziony kierowca  
# o imieniu Adam  
# jest o rok starszy
```

```
> db.drivers.updateMany(  
  {"first_name": "Ania"},  
  {$inc: {"age": -2}}  
)  
# Wszystkie Anie (updateMany)  
# są o dwa lata młodsze
```

**\$inc** Zmienia wartość wskazanego pola o podaną liczbę (dodatnią lub ujemną). Działa na polach numerycznych.



```
> db.drivers.update(  
  {"first_name": "Adam"},  
  {$inc: {"age": 1}}  
)  
# Pierwszy znaleziony kierowca  
# o imieniu Adam  
# jest o rok starszy
```

**Ćwiczenie 1:** Wszyscy kierowcy, którzy mają na imię Paweł, Anna lub Jan powinni być o rok młodsi.

**Ćwiczenie 2:** Co się stanie jak wywołamy operacje **\$inc**

- a) na polu tekstowym np. **first\_name**
- b) polu, które nie istnieje

**Ćwiczenie 3:** Zmniejszamy o **1** liczbę punktów wszystkim kierowcą. Uwaga: liczba punktów nie może spaść poniżej zera.

# Usuwanie dokumentów



- Baza danych i teoria CAP
- Bazy relacyjne vs NoSQL
- MongoDB
- Instalacja
- JSON
- Bazy i kolekcje
- Dodawanie nowych dokumentów
- Aktualizacja całego dokumentu
- Szukanie z wykorzystywaniem kryteriów
- Modyfikacja dokumentów
- Usuwanie dokumentów**
- Indeksy
- Integracja z JAVA
- MongoDB University
- Podsumowanie

# Remove



```
> db.drivers.remove({KRYTERIA})
```

```
> db.drivers.remove(  
  {"points" : {$gte : 21.0}}  
)
```

# Usuwa kierowców, którzy  
# mają 21 punktów lub więcej

Zamias `find(...)` wywołujemy `remove(...)`. Dokumenty spełniające zadane kryterium zostaną usunięte.



```
> db.drivers.remove(  
  {"points" : {$gte : 21.0}}  
)  
# Usuwa kierowców, którzy  
# mają 21 punktów lub więcej
```

Zmienione w Polsce prawo. Wiek wymagany do kierowania pojazdami mechanicznymi został zwiększony do 20 lat.

**Ćwiczenie 1:** Znajdź kierowców, którzy nie mają 20 lat.

**Ćwiczenie 2:** Usuń z kolekcji kierowców, którzy nie mają 20 lat.

# Indeksy



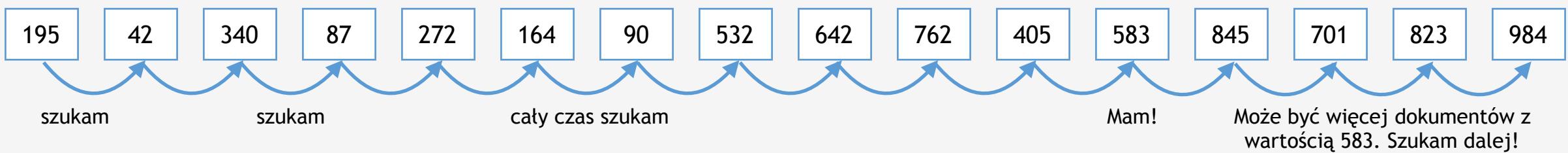
Baza danych i teoria CAP  
Bazy relacyjne vs NoSQL  
MongoDB  
Instalacja  
JSON  
Bazy i kolekcje  
Dodawanie nowych dokumentów  
Aktualizacja całego dokumentu  
Szukanie z wykorzystywaniem kryteriów  
Modyfikacja dokumentów  
Usuwanie dokumentów  
Indeksy  
Integracja z JAVA  
MongoDB University  
Podsumowanie



# Szukanie bez indeksu

```
> db.drivers.find( {"number": 583} )
```

Dokumenty mają następujące wartości



Aby znaleźć dokument z liczbą 583 musielibyśmy przejrzeć całą kolekcję dokumentów. Aktualnie jest to tylko 16 dokumentów. Bez indeksu, wraz z wzrostem zbioru danych, czas szukania dokumentów również rośnie.

# Indeks



W Indeks (bazy danych) – Wikipedia

Nie jesteś zalogowany Dyskusja Edycje Utwórz konto Zaloguj się

Artykuł Dyskusja Czytaj Edytuj Edytuj kod źródłowy Historia i autorzy Przeszukaj Wikipedię

[ukryj]

**Indeks (bazy danych)** [edytuj]

**Indeks** – struktura danych zwiększająca szybkość wykonywania operacji wyszukiwania na tabeli.

**Typy** [edytuj | edytuj kod]

**indeks główny**  
główny (na identyfikatorach wierszy) - *primary*

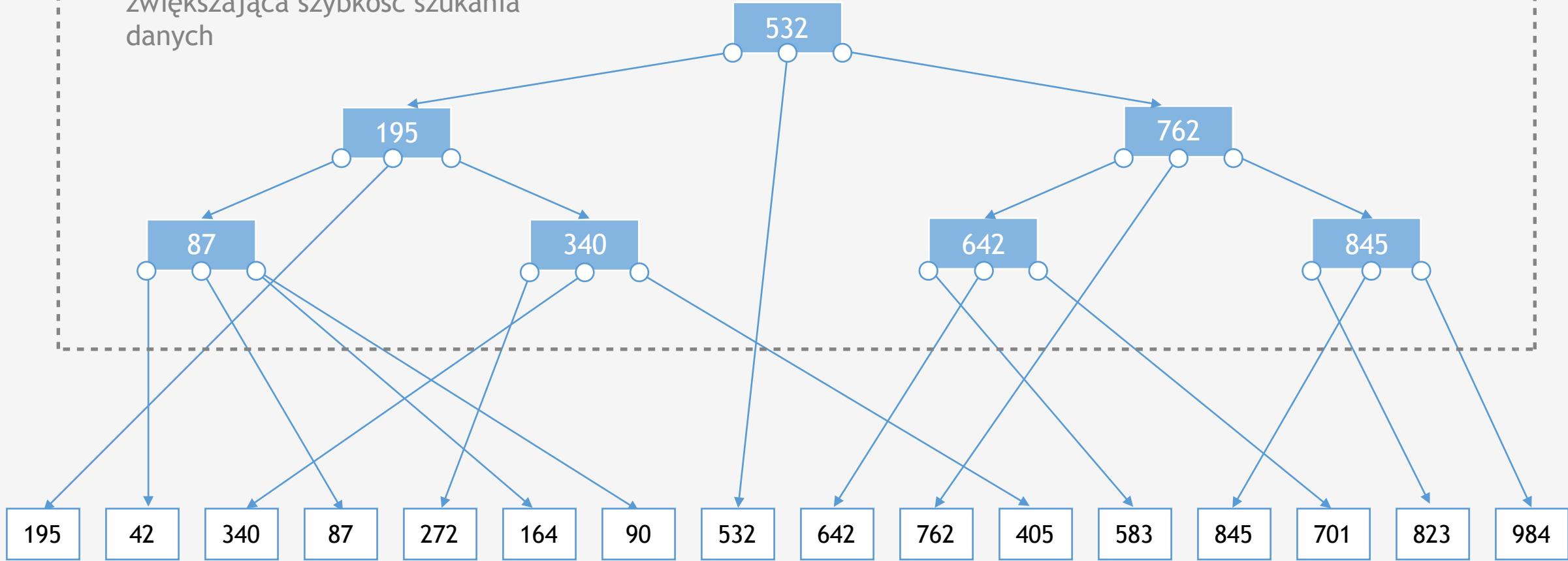
**indeks drugorzędny**  
pomocniczy (*secondary*)

The screenshot shows a web browser window displaying the Polish Wikipedia article titled "Indeks (bazy danych)". The page content includes a brief definition of an index as a data structure that speeds up search operations on tables. It then lists two types of indices: the primary key (main index, "indeks główny") and secondary keys ("indeks drugorzędny"). The primary key is described as being based on row identifiers, while the secondary key is described as being auxiliary. The page also features a sidebar with various links related to the site's navigation and user interaction.

# Indeks



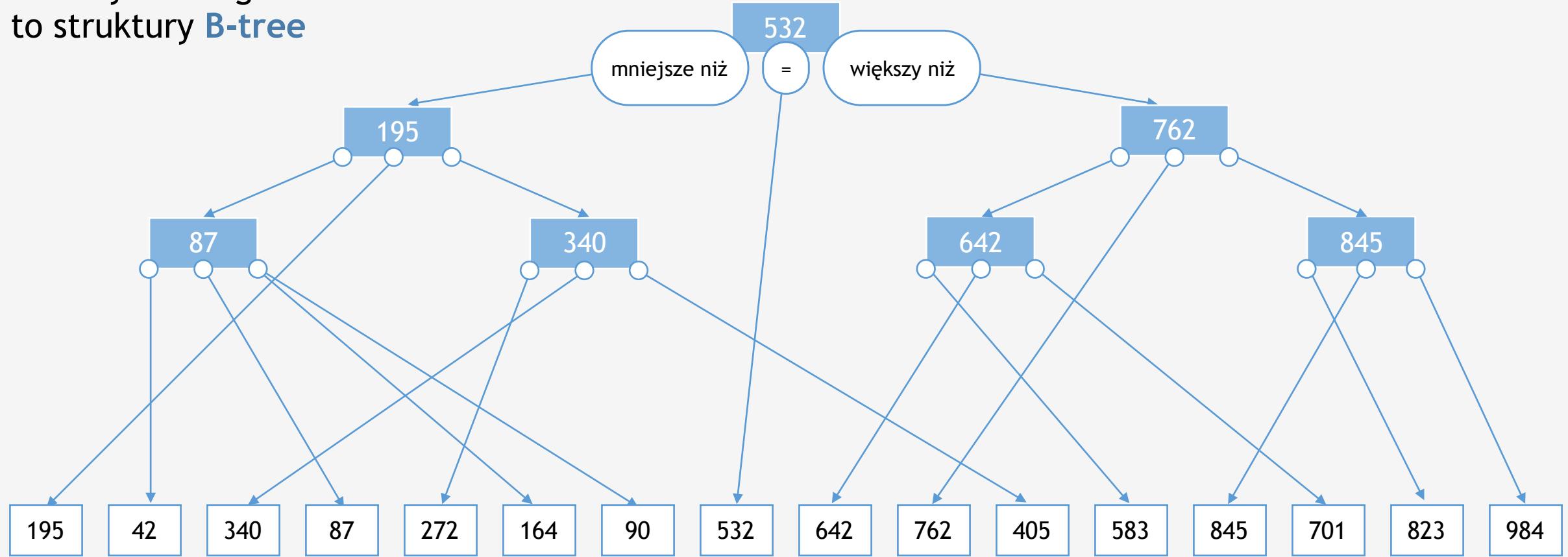
Indeks, czyli dodatkowa struktura zwiększająca szybkość szukania danych





# B-tree

Indeksy w MongoDB  
to struktury **B-tree**

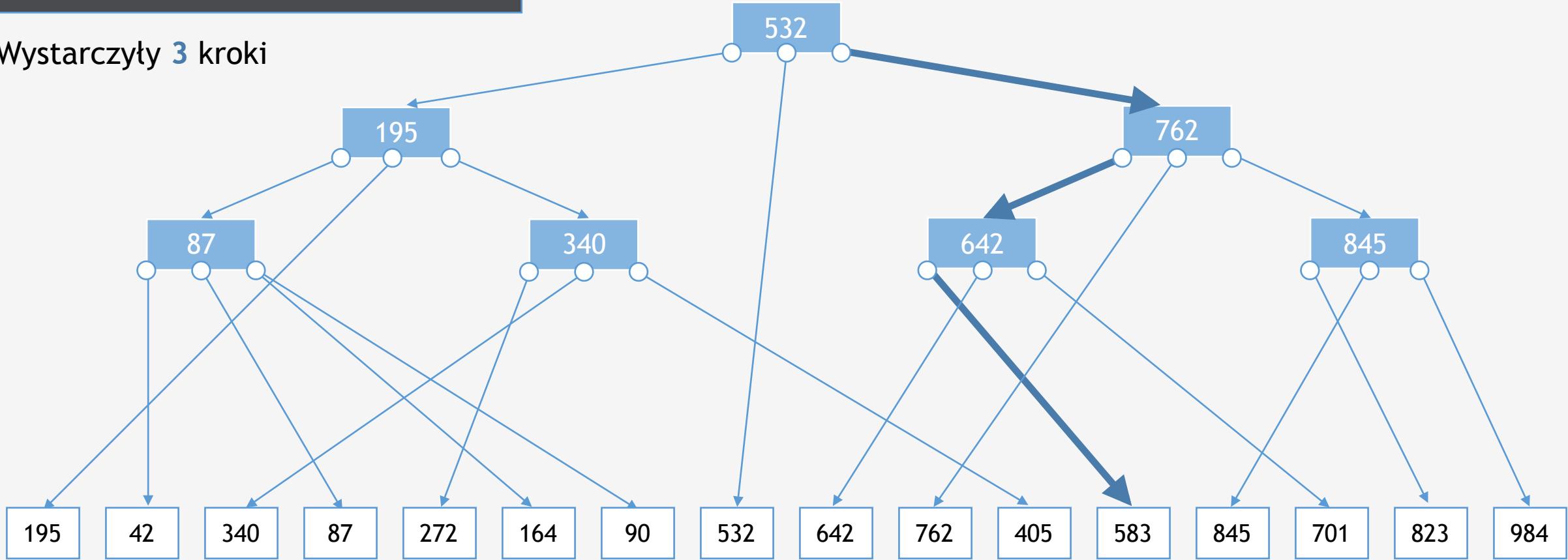




# B tree - szukamy 583

```
> db.drivers.find( { "number": 583 } )
```

Wystarczyły 3 kroki

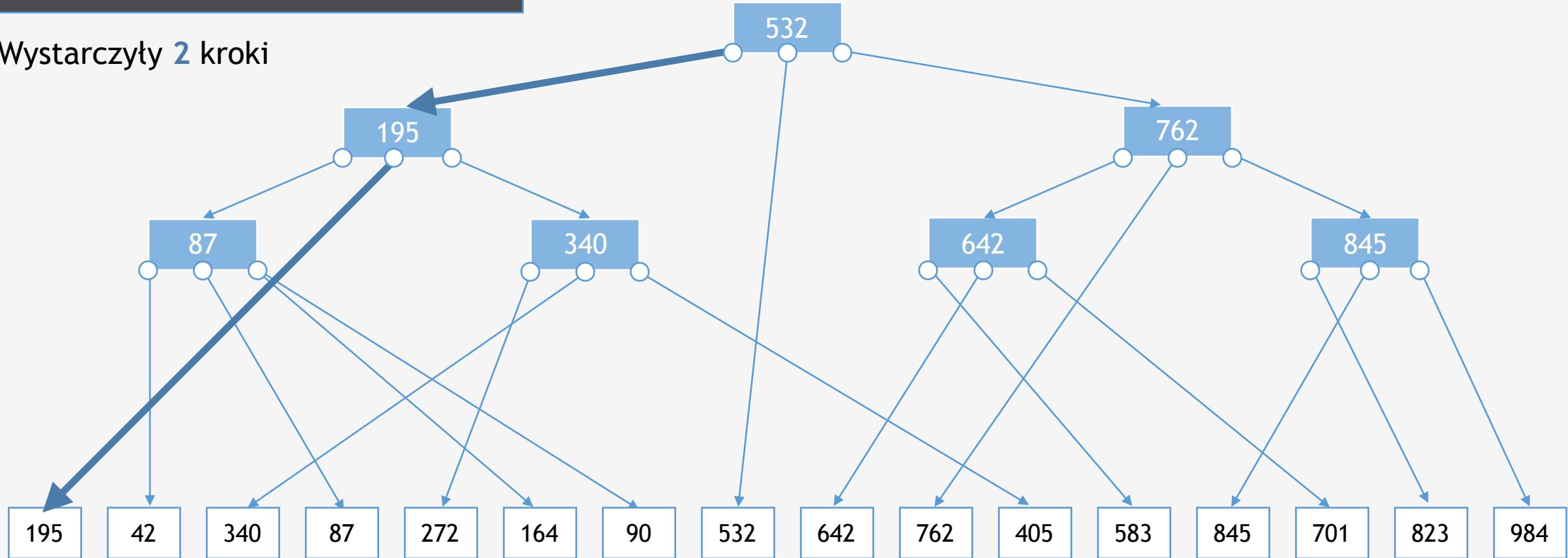




# B tree - szukamy 195

```
> db.drivers.find( { "number": 195 } )
```

Wystarczyły 2 kroki





# Generowanie danych

## Przygotowanie bazy danych

W przypadku małej liczby danych korzyści z indeksu nie są zauważalne.

Dodajmy do bazy danych większą liczbę kierowców. Tak aby na koniec każdy z nas miał ich **minimum 50 tys.**

W Robo-3T polecam ustawienie wyższego czasu oczekiwania aplikacji na wykonanie skryptu  
Menu: Options > Change Shell Timeout...

Skrypt: [add-drivers.js](#)

```
* Array.prototype.random = function () {
  localhost 27017 mongo_workshop
  Array.prototype.random = function () {
    return this[Math.floor((Math.random() * this.length))];
  };

  function getInteger(min, max) {
    return Math.floor(Math.random() * (max - min + 1)) + min;
  }

  function getFirstName() {
    return [
      "Paweł", "Ania", "Michał", "Józef", "Emi", "Korneliusz", "Aneta",
      "Julia", "Zofia", "Hanna", "Olivia", "Rita", "Justyna", "Filip",
      "Wojciech", "Szymon", "Jakub", "Antoni", "Jan", "Brajan", "Diana",
      "Marta", "Juri", "Bartosz", "Tymoteusz", "Grzegorz", "Robert"
    ].random();
  }

  function getLastname() {
    return this[Math.floor((Math.random() * this.length))];
  }
}

Logs
Key Value Type
(1) { 5 fields } Object
(2) { 5 fields } Object
(3) { 5 fields } Object
(4) { 5 fields } Object
```



# Listowanie założonych już indeksów

```
> db.drivers.getIndexes()
```

```
[ ← Lista składająca się z jednego elementu
{
    "v" : 2,
    "key" : {
        "_id" : 1 ← DOMYŚLNY
                    INDEKS ZAŁOŻONY
                    NA _id
    },
    "name" : "_id_",
    "ns" : "mongo_workshop.drivers",
}
] ←
```



# Tworzenie nowego indeksu

```
> db.drivers.createIndex( { POLA } , { OPCJE } )
```

```
> db.drivers.createIndex( { "age": 1 } ) # Tworzy standardowy  
# indeks na polu age
```

```
> db.drivers.createIndex( { "first_name": 1, "last_name": 1 } ) # Tworzy indeks na  
# dwóch polach
```

```
> db.drivers.createIndex( { "age": 1 } , { "background": true } ) # Indeks zostanie  
# stworzony w tle
```

```
> db.drivers.createIndex( { "last_name": 1 } , { "unique": true } ) # Indeks zapewniający  
# unikalność
```



# Tworzenie nowego indeksu

```
{  
  "_id" : ObjectId("5b5e2f89529592926f70f022") ,  
  "first_name" : "Anna" ,  
  (...)  
  "vehicles" : [  
    { "number" : "PO3493" , "color" : "blue" , "seats" : 2 } ,  
    { "number" : "PO9831" , "color" : "black" , "seats" : 5 }  
  ]  
}
```

```
> db.drivers.createIndex( { "vehicles.seats": 1 } ) # Indeks na polu  
# zagnieżdżonym
```

# Tworzenie nowego indeksu

ĆWICZENIE



```
> db.drivers.createIndex( { "vehicles.seats": 1 } ) # Indeks na polu  
# zagnieżdżonym
```

```
> db.drivers.createIndex( {"last_name": 1}, { "unique": true } ) # Indeks zapewniający  
# unikalność
```

**Ćwiczenie 1:** Policz kierowców, którzy mają 28 lat. Jak długo trwało zapytanie?

**Ćwiczenie 2:** Załóż indeks na polu **"age"** i ponownie policz kierowców, którzy mają 28 lat. Jak długo teraz trwało zapytanie?

**Ćwiczenie 3:** Załóż indeks **unikalny** na polu **"vehicles.number"**. Spróbuj dodać samochód z numerem rejestracyjnym, który istnieje już w bazie danych. **Co się stało?**

# Integracja z JAVA



Baza danych i teoria CAP  
Bazy relacyjne vs NoSQL  
MongoDB  
Instalacja  
JSON  
Bazy i kolekcje  
Dodawanie nowych dokumentów  
Aktualizacja całego dokumentu  
Szukanie z wykorzystywaniem kryteriów  
Modyfikacja dokumentów  
Usuwanie dokumentów  
Indeksy  
**Integracja z JAVA**  
MongoDB University  
Podsumowanie



## Ćwiczenie

1. Sklonuj repozytorium

[Clone or download ▾](#)

2. Przeczytaj plik: **README.md**

3. Zaimplementuj brakujące metody w klasie  
wareq.mongoworkshop.repository.DriverRepository

Branche z zadaniami:

[https://github.com/pawelwar/mongo-workshop-java/tree/exercise\\_01](https://github.com/pawelwar/mongo-workshop-java/tree/exercise_01)

[https://github.com/pawelwar/mongo-workshop-java/tree/exercise\\_02](https://github.com/pawelwar/mongo-workshop-java/tree/exercise_02)

<https://github.com/pawelwar/mongo-workshop-java>

The screenshot shows the GitHub repository page for 'wareq / mongo-workshop'. At the top, there's a banner for 'Join GitHub today' with a 'Sign up' button. Below the banner, the repository name is displayed. The repository statistics are shown: 1 commit, 2 branches, 0 releases, and 1 contributor. The 'master' branch is selected. A 'Find file' and 'Clone or download' button are also present. The repository tree lists several files and folders, all of which were committed by 'wareq' just now. At the bottom, there's a section titled 'Mongo Workshop'.

File/Folder	Description	Time Ago
Exercises: Spring Boot application integrated with Spring Data MongoDB	Latest commit fac75ac an hour ago	
gradle/wrapper	Exercises: Spring Boot application integrated with Spring Data MongoDB	9 minutes ago
src	Exercises: Spring Boot application integrated with Spring Data MongoDB	9 minutes ago
.gitignore	Exercises: Spring Boot application integrated with Spring Data MongoDB	9 minutes ago
README.md	Exercises: Spring Boot application integrated with Spring Data MongoDB	9 minutes ago
build.gradle	Exercises: Spring Boot application integrated with Spring Data MongoDB	9 minutes ago
gradlew	Exercises: Spring Boot application integrated with Spring Data MongoDB	9 minutes ago
gradlew.bat	Exercises: Spring Boot application integrated with Spring Data MongoDB	9 minutes ago
settings.gradle	Exercises: Spring Boot application integrated with Spring Data MongoDB	9 minutes ago

# Integracja z JAVA

ROZWIĄZANIE



## Rozwiązania

```
/**  
 * Get single driver by id  
 */  
public Driver getById(String id) { return mongoTemplate.findById(id, Driver.class); }  
  
/**  
 * Get drivers with specific first and last name  
 */  
public List<Driver> get(String firstName, String lastName) {  
    Criteria criteria = Criteria  
        .where(Driver.FIRST_NAME_FIELD).is(firstName)  
        .and(Driver.LAST_NAME_FIELD).is(lastName);  
  
    return mongoTemplate.find(  
        new Query(criteria),  
        Driver.class  
    );  
  
/**  
 * Get drivers  
 * - older than 80 years old  
 * - sorted by age  
 * - with pagination (skip and limit parameter)  
 */  
public List<Driver> getOlderThan(Integer age, Integer skip, Integer limit) {  
    Criteria ageGreaterThanOrEqualCriteria = Criteria.where(Driver.AGE_FIELD).gt(age);  
    Sort oldestFirst = new Sort(Sort.Direction.DESC, Driver.AGE_FIELD);  
  
    return mongoTemplate.find(  
        new Query(ageGreaterThanOrEqualCriteria).skip(skip).limit(limit).with(oldestFirst),  
        Driver.class  
    );  
}
```

### DriverRepository

[https://github.com/pawelwar/mongo-workshop-java/tree/solution\\_01](https://github.com/pawelwar/mongo-workshop-java/tree/solution_01)  
[https://github.com/pawelwar/mongo-workshop-java/tree/solution\\_02](https://github.com/pawelwar/mongo-workshop-java/tree/solution_02)

The screenshot shows a GitHub repository named 'mongo-workshop' owned by 'wareq'. The repository has 1 commit, 2 branches, 0 releases, and 1 contributor. It was last updated an hour ago. The repository contains files like 'gradle/wrapper', 'src', '.gitignore', 'README.md', 'build.gradle', 'gradlew', 'gradlew.bat', and 'settings.gradle'. A note at the top encourages users to 'Join GitHub today'.

Autor: Paweł Warczyński

Prawa do korzystania z materiałów posiada Software Development Academy

# MongoDB University



Baza danych i teoria CAP  
Bazy relacyjne vs NoSQL  
MongoDB  
Instalacja  
JSON  
Bazy i kolekcje  
Dodawanie nowych dokumentów  
Aktualizacja całego dokumentu  
Szukanie z wykorzystywaniem kryteriów  
Modyfikacja dokumentów  
Usuwanie dokumentów  
Indeksy  
Integracja z JAVA  
MongoDB University  
Podsumowanie



# MongoDB University

The screenshot shows the MongoDB University website interface. At the top, there's a navigation bar with links for DOCS, OPEN SOURCE, COMPANY, CERTIFICATION, ONLINE COURSES, and TRAINING. A 'Sign in' button is also present. The main heading is 'Learn MongoDB from MongoDB'. Below it is a 'Get Started' button. A large orange box features the course 'M040: New Features and Tools in MongoDB 4.0' with a description: 'Get up to speed with everything this exciting release has to offer.' To the right, there's a section for the 'M001: MongoDB Basics' course, which includes a thumbnail of a laptop screen showing a database interface, the course title, a brief description, and a 'New Course' section for 'M103: Basic Cluster Administration'.

**M040:**  
New Features and  
Tools in MongoDB 4.0

Get up to speed with everything this exciting release has to

**Course:**  
M001: MongoDB Basics  
*Jumpstart your learning with this course*

**New Course:**  
M103: Basic Cluster  
Administration  
*Administer MongoDB deployments with confidence.*

**Certification Exam:**

<https://university.mongodb.com/courses/catalog>



# MongoDB University



## M001: MongoDB Basics

*Introductory*

Fundamentals of MongoDB: connecting to a MongoDB Cluster, using MongoDB Compass, MongoDB's document storage model and principles of flexible schema design, basic architecture of MongoDB clusters, CRUD operations.



## M101J: MongoDB for Java Developers

*Introductory*

Learn basic installation of MongoDB, JSON, schema design, querying, insertion of data, indexing and working with language drivers. The course project involves building a blogging platform with MongoDB. Code examples will be in Java.

<https://university.mongodb.com/courses/catalog>

Autor: Paweł Warczyński

Prawa do korzystania z materiałów posiada Software Development Academy



# MongoDB University

<p> <b>Course Completion Confirmation</b></p> <p>JULY 2013</p> <p><i>This confirms</i> <b>YOUR NAME</b> _____</p> <p><i>successfully completed</i></p> <p><b>M101J: MongoDB for Java Developers</b></p> <p><i>a course of study offered by MongoDB, Inc.</i></p> <p></p> <p>Shannon Bradshaw VP, Education MongoDB, Inc.</p>	<p> <b>Course Completion Confirmation</b></p> <p>JUNE 2013</p> <p><i>This confirms</i> <b>YOUR NAME</b> _____</p> <p><i>successfully completed</i></p> <p><b>M102: MongoDB for DBAs</b></p> <p><i>a course of study offered by MongoDB, Inc.</i></p> <p></p> <p>Shannon Bradshaw VP, Education MongoDB, Inc.</p>
--	--



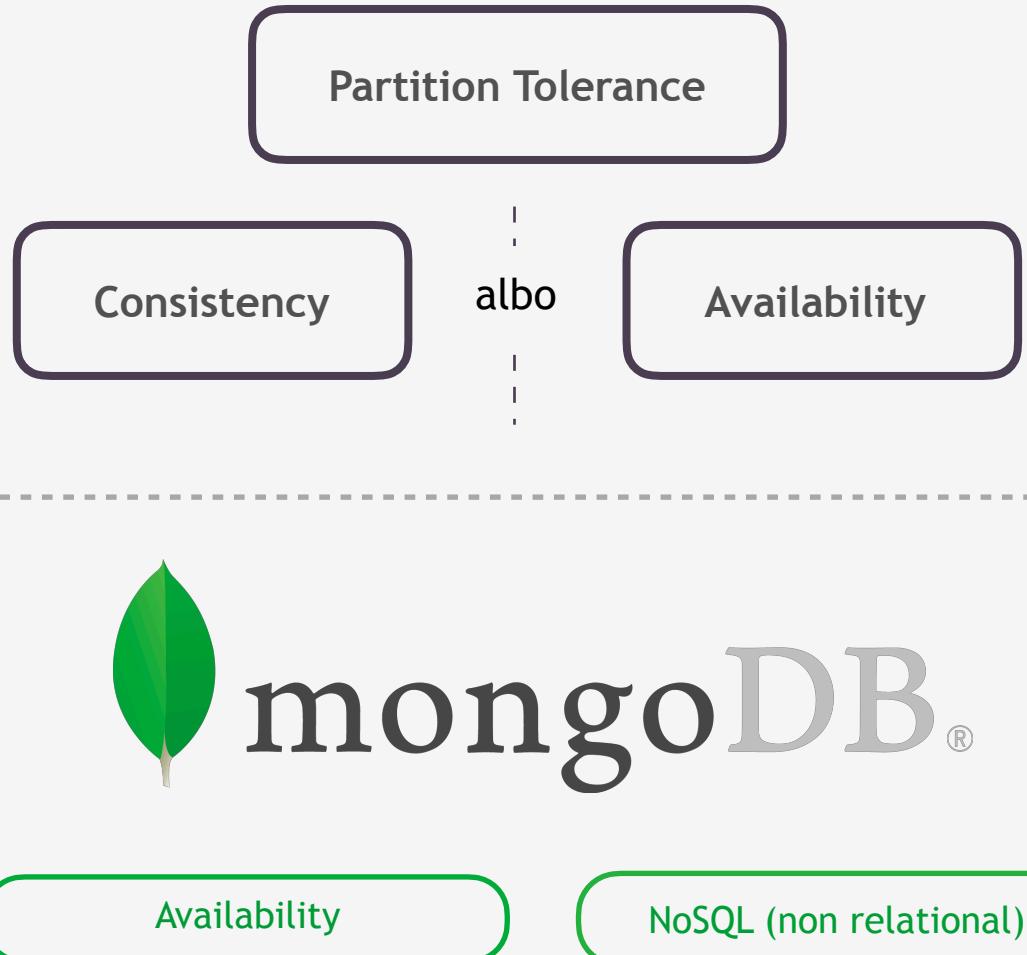
# Podsumowanie

## ...czyli 16 rzeczy, które chciałbym abyście zapamiętali

- Baza danych i teoria CAP
- Bazy relacyjne vs NoSQL
- MongoDB
- Instalacja
- JSON
- Bazy i kolekcje
- Dodawanie nowych dokumentów
- Aktualizacja całego dokumentu
- Szukanie z wykorzystywaniem kryteriów
- Modyfikacja dokumentów
- Usuwanie dokumentów
- Indeksy
- Integracja z JAVA
- MongoDB University
- Podsumowanie



# Podsumowanie

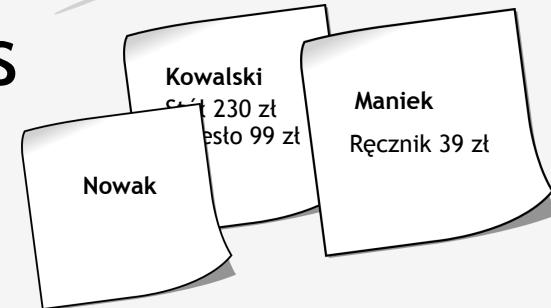


# mongoDB®

id	name	id	user
3	Kowalski	343	3
4	Nowak	344	3

id	user
343	3
344	3

VS

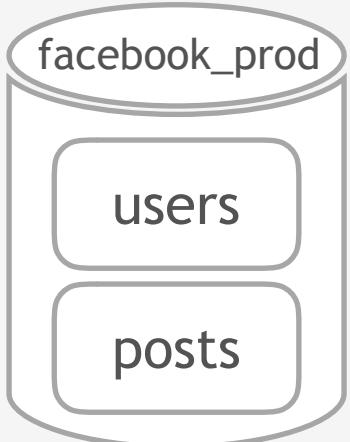


**Format: JSON**

```
{  
  "first_name" : "Jan",  
  "last_name" : "Nowak",  
  "age" : 38,  
  "vehicles" : ["ATS1234", "WAW3626"]  
}
```



# Podsumowanie



```
> show dbs  
facebook_prod
```

```
> show collections  
users  
news  
posts  
groups
```

```
> db.nazwakolekcji.insert(DOKUMENT)
```

```
> db.nazwakolekcji.save(DOKUMENT)
```

```
> db.drivers.find({ "points": 5 })
```

```
> db.drivers.count({ "age": 5 })
```

**\$gt** większy niż

**\$gte** większy niż lub równy

**\$lt** mniejszy niż

**\$lte** mniejszy niż lub równy

# Podsumowanie



```
> db.drivers.find( {} )  
    .sort({ "points": -1 })  
    .skip(2)  
    .limit(10)
```

```
"first_name": { $in: ["Emi", "Ada"] }
```

```
"first_name": { $nin: ["Xen", "DJ"] }
```

```
"vehicles": {  
    $elemMatch: { "number": "PO9831" }  
}
```

```
> db.drivers.find({  
    "age": { $gte : 20, $lte : 35},  
    "vehicles": { $elemMatch: { ... } },  
    "first_name": { $in: ["Paweł", ... ] } P#####  
})  
.sort({ "points": -1 }) 🔎
```





# Podsumowanie



+1800 zł



+200 zł

`save({OBJECT})`

vs

`update({K}, {Z})`

```
> db.drivers.update({KRYTERIA},  
{ZMIANA})
```

```
> db.drivers.updateMany({KRYTERIA},  
{ZMIANA})
```

```
> db.drivers.update(  
  {_id : ObjectId("5b5...2")},  
  { $set: {"last_name": "Kowalska"} })
```

```
> db.drivers.update(  
  { "first_name" : "Adam"},  
  { $inc: { "age": 1 } })
```



# Podsumowanie

Indeks, czyli dodatkowa struktura zwiększająca szybkość szukania danych

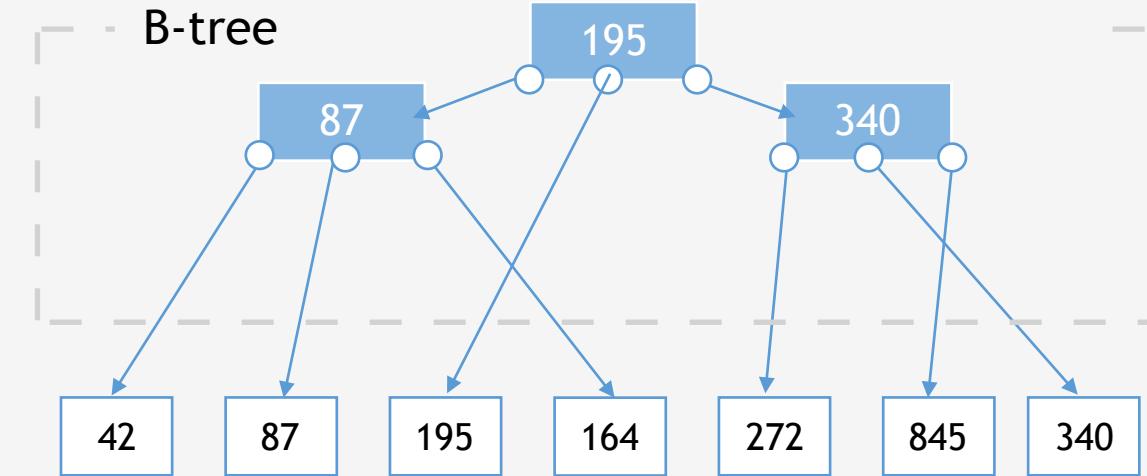
```
/*
 * Get single driver by id
 */
public Driver getById(String id) { return mongoTemplate.findById(id, Driver.class); }

/*
 * Get drivers with specific first and last name
 */
public List<Driver> get(String firstName, String lastName) {
    Criteria criteria = Criteria
        .where(Driver.FIRST_NAME_FIELD).is(firstName)
        .and(Driver.LAST_NAME_FIELD).is(lastName);

    return mongoTemplate.find(
        new Query(criteria),
        Driver.class
    );
}

/*
 * Get drivers
 * - older than 80 years old
 * - sorted by age
 * - with pagination (skip and limit parameter)
 */
public List<Driver> getOlderThan(Integer age, Integer skip, Integer limit) {
    Criteria ageGreaterThanOrEqualCriteria = Criteria.where(Driver.AGE_FIELD).gt(age);
    Sort oldestFirst = new Sort(Sort.Direction.DESC, Driver.AGE_FIELD);

    return mongoTemplate.find(
        new Query(ageGreaterThanOrEqualCriteria).skip(skip).limit(limit).with(oldestFirst),
        Driver.class
    );
}
```





Dziękuję!  
życzę wam powodzenia



# Materiały dodatkowe



# Szukanie z wykorzystaniem kryteriów

## Część #2



# \$exists

```
> db.drivers.find({ "supervision": { $exists: true } }) # zwraca dokumenty, które  
# mają pole supervision
```

```
> db.drivers.find({ "supervision": { $exists: false } }) # zwraca dokumenty, które  
# NIE mają pole supervision
```

**\$exists: true / false**

sprawdza czy dokument posiada lub  
nie posiada wskazanego pola



```
> db.drivers.find({ "supervision": { $exists: true } }) # zwraca dokumenty, które  
# mają pole supervision
```

Wykonajcie następujące polecenie

```
> db.drivers.update( {} , { $set: { "professionalRacerSince" : new Date() } } )
```

Do jednego z dokumentów zostało dodane pole **professionalRacerSince** z ustawioną aktualną datą. Kierowca został oznaczony jako profesjonalny rajdowiec.

**Ćwiczenie 1:** Znajdź tego kierowcę korzystając z kryterium **\$exists**.

**Ćwiczenie 2:** Ilu kierowców **nie** jest profesjonalnymi rajdowcami?



# \$size

```
> db.drivers.find({ "vehicles": { $size: 2 } }) # kierowcy tylko z dwoma samochodami
```

```
{
  "_id" : ObjectId("5b5e2f89529592926f70f022"),
  (...),
  "vehicles" : [
    { "number" : "PO3493", "color" : "blue", "seats" : 2 },
    { "number" : "PO9831", "color" : "black", "seats" : 5 }
  ]
}
```

**\$size**  
tylko dla list [...]

znajduję dokumenty, w których wskazana listą ma określoną liczbę elementów



```
> db.drivers.find({ "vehicles": { $size: 2 } }) # kierowcy tylko z dwoma samochodami
```

**Ćwiczenie 1:** Który kierowca z czterema samochodami ma najwięcej punktów?  
podpowiedź: [sort\(...\)](#)



# \$or

```
> db.drivers.find({$or : [  
    {"points": {$lt : 0}}, # szukamy anomalii  
    {"age": {$lt: 15}}, # mniej niż 0 punktów np. -3  
    {"first_name": "Daniel", "last_name": "Olbrychski"} # lub kierowca ma mniej niż 15 lat  
] }) # lub kierowca to Daniel O.
```

## \$or

Wystarczy, że jeden z warunków zwróci true by całe wyrażenie również zwróciło true



```
> db.drivers.find({$or : [  
    {"points": {$lt : 0}},  
    {"age": {$lt: 15}},  
    {"first_name": "Daniel", "last_name": "Olbrychski"}  
] })
```

**Ćwiczenie 1:** Ilu kierowców ma na imię Adam lub na nazwisko Adamek (38)

**Ćwiczenie 2:** Ilu kierowców ma więcej niż (\$gt) 80 lat lub więcej niż (\$gt) 15 punktów (470)

**Ćwiczenie 3:** Znajdź Grzegorzów, którzy mają od (\$gte) 18 do (\$lte) 20 lat  
lub od (\$gte) 80 do (\$lte) 100 lat (6)



# \$not

```
> db.drivers.find({ "points": { $not : { $gt: 20 } } })      # zagnieżdżony warunek
```

```
> db.drivers.find({ "points": { $not : { $not : { $gt: 20 } } } })
```

## \$not

Jeżeli zagnieżdżony warunek zwróci TRUE zostanie zamieniony na FALSE. Zwrócone FALSE zostanie zamienione na TRUE.



```
> db.drivers.find( { "points": { $not : { $gt: 20 } } } )
```

**Ćwiczenie 1:** Ilu istnieje kierowców, którzy **nie** mają dokładnie 20 lat



# Modyfikacja dokumentów

Część #2



# Update - \$mul

```
> db.drivers.updateMany(  
    {},  
    { $mul: { "age": 3 } }  
)  
# Wszyscy stają się 3 razy starsi
```

```
> db.drivers.update(  
    { "_id" : ObjectId("5b5e2...0f022") } ,  
    { $mul: { "score": 1.234 } }  
)  
# We wskazanym dokumencie score  
# zostanie przemnożony o 1.234
```

**\$mul** Mnoży wartość z wskazanego pola o przekazaną liczbę

# Update - \$mul

ĆWICZENIE



```
> db.drivers.updateMany(  
  {},  
  { $mul: { "age": 3 } }  
)
```

**Ćwiczenie 1:** Powiększ dwukrotnie liczbę punktów karnych kierowcą, którzy mają ich aktualnie 20 lub więcej



# Update \$addToSet, \$push

```
> db.drivers.update(  
  { "_id" : ObjectId("5b6df04211561d86bfe9700e") } ,  
  { $addToSet: { "vehicles" :  
    NOWY  
    SAMOCHÓD { "number" : "ZZ0101", "color" : "red", "points" : 5.0 }  
  } }  
)
```

# Dla wybranego kierowcy  
# do listy vehicles  
# zostanie dodany  
# nowy samochód

**\$addToSet**  
tylko dla list [...]

Dodaje nowy element do listy. Jeżeli istnieje on już na liście - wtedy **nie zostanie dodany podwójnie**.

**\$push**  
tylko dla list [...]

Dodaje nowy element do listy. Jeżeli istnieje on już na liście - zostanie dodany **duplikat**.

# Update \$addToSet, \$push

ĆWICZENIE



```
> db.drivers.update(
    {"_id" : ObjectId("5b6df04211561d86bfe9700e") },
    { $addToSet: { "vehicles" :
        { "number" : "ZZ0101", "color" : "red", "points" : 5.0 }
    } }
)
```

**Ćwiczenie 1:** Paweł Kukiz kupił nowy samochód osobowy. Dodaj do jego dokumentu samochód pięcioosobowy, koloru czarnego, o numerach rejestracyjnych POPIWO.



# Update \$pull, \$pop

```
> db.drivers.update(  
    { "_id" : ObjectId("5b6df04211561d86bf9700e") } ,  
    { $pull: { "vehicles" :  
        KRYTERIA { "points" : { $gte : 4.0 } }  
    } }  
)
```

# Kasujemy wszystkie  
# samochody, które mają  
# 4 lub więcej miejsc  
# i należą do kierowcy  
# o identyfikatorze  
# ObjectId(5b6df0...9700e)

## \$pull

tylko dla list [...]

Kasuje z wskazanej listy element, który spełnia zdefiniowane przez nas kryteria.

## \$pop

tylko dla list [...]

W zależności od wartości kasuje pierwszy lub ostatni element na wskazanej liście.

# Update \$pull, \$pop

ĆWICZENIE



```
> db.drivers.update(
    {"_id" : ObjectId("5b6df04211561d86bfe9700e") },
    { $pull: { "vehicles" :
        { "points" : { $gte : 4.0 } }
    }
}
```

# Kasujemy wszystkie  
# samochody, które mają  
# 4 lub więcej miejsc  
# i należą do kierowcy  
# o identyfikatorze  
# ObjectId(5b6df0...9700e)

**Ćwiczenie 1:** Usuń wszystkie samochody koloru czarnego, których właściciel ma na imię "Emi".

**Ćwiczenie 2:** Roman Trajkowski ma 6 samochodów. Usuń ostatni samochód, który posiada.  
podpowiedź: usuń pierwszy **\$pop (nazwa\_pola: 1)**, usuń ostatni **\$pop (nazwa\_pola: -1)**



# Update - \$min, \$max

```
> db.drivers.update(  
  {"insurance" : true},  
  {$min: {"points": 2}}  
)  
# Wszyscy kierowcy, którzy mają  
# wykupione ubezpieczenie  
# ustawiamy liczbę punktów na 2  
# chyba, że wcześniej mieli  
# zgromadzone mniej punktów
```

```
> db.drivers.updateMany(  
  {"insurance" : false},  
  {$max: {"points": 15}}  
)  
# Wszyscy kierowcy, którzy NIE mają  
# wykupione ubezpieczenie  
# ustawiamy liczbę punktów na 15  
# chyba, że wcześniej mieli  
# zgromadzone więcej punktów(16, 17...)
```

## \$min

Porównuje dwie liczby: aktualną ustawioną w polu z tą, którą przekazał użytkownik. W polu zostanie ustawiona ta mniejsza.

# Update - \$min, \$max

ĆWICZENIE



```
> db.drivers.update(  
  {"insurance" : true},  
  {$min: {"points": 2}}  
)  
  # Wszyscy kierowcy, którzy mają  
  # wykupione ubezpieczenie  
  # ustawiamy liczbę punktów na 2  
  # chyba, że wcześniej mieli  
  # zgromadzone mniej punktów (0 lub 1)
```

**Ćwiczenie 1:** Wszyscy kierowcy, którzy mają na imię Paweł powinni mieć ustawione 15 punktów karnych, chyba, że aktualnie mają ich więcej.  
podpowiedź: skorzystaj z **\$max**

**Ćwiczenie 2:** W jaki sposób zrobić ćwiczenie 1 nie używając **\$min** i **\$max**?