# Krótki przewodnik po pracy z bazą danych MongoDB

Opracował: Artur Gramacki

## 1. Uwagi wstępne

W niniejszym opracowaniu[1] pokazujemy tylko wybrane polecenia służące do pracy z **kolekcjami dokumentów**. Pomijamy np. całkowicie polecenia służące do zarządzania bazą danych i pracy z innymi obiektami niż kolekcje (np. indeksy, replikacja, sharding). Baza MongoDB, mimo pozornej prostoty, jest całkiem złożonym systemem informatycznym.

W niniejszym pracowaniu pokazujemy tylko niewielki wycinek możliwości MongoDB, skupiając się na naprawdę najbardziej podstawowych poleceniach. Tradycyjnie odsyłamy do dokumentacji źródłowej, która jest najbardziej kompletnym źródłowe wiedzy na ten temat.
https://docs.mongodb.com/manual/
Spis poleceń do pracy z kolekcjami znajdziemy tutaj:
https://docs.mongodb.com/manual/reference/method/js-collection/

## 2. Instalacja i uruchomienie MongoDB

Posiłkując się wiadomościami z wykładu zainstaluj serwer MongoDB i go uruchom (w konsoli tekstowej). Powinieneś zobaczyć m.in. taki tekst, który świadczy o tym, że serwer uruchomił się poprawnie.

```
NETWORK  [listener] Listening on 127.0.0.1
NETWORK  [listener] waiting for connections on port 27017
```

Połącz się z konsoli tekstowej do serwera. Wszystkie ćwiczenia będą wykonywane bezpośrednio w tej konsoli. Na tym etapie nie będziemy używali żadnego graficznego programu klienckiego (jak np. *Compass*, *Robo 3T* itp.).

Na początek wylistujemy aktualne bazy danych. Gdy dopiero, co zainstalowałeś oprogramowanie MongoDB powinny być widoczne 3 bazy systemowe. Nie należy do nich się łączyć ani wykonywać na nich jakichkolwiek działań, aby przypadkiem czegoś nie uszkodzić. Należy utworzyć swoją własną bazę lub bazy i na nich wykonywać wszystkie ćwiczenia. Poniżej pokazano wynik działania polecenia listującego aktualnie istniejące w systemie bazy danych.

**Uwaga**: w ramkach, w kolorze czarnym, podajemy wydawane polecenia natomiast na **niebiesko** pokazujemy otrzymywane wyniki.

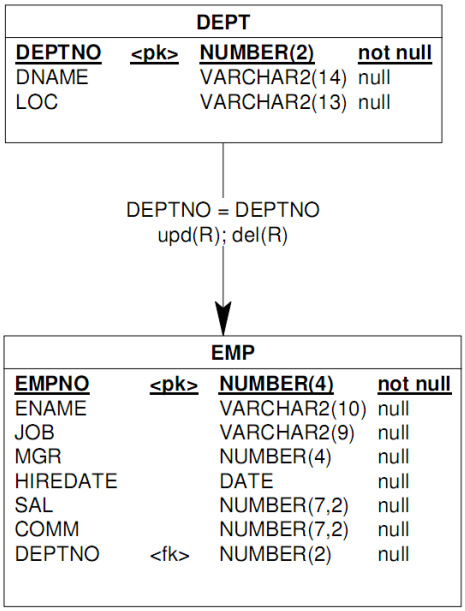
[1] Opracowując ten materiał korzystałem głównie z oryginalnej dokumentacji MongoDB oraz z
http://www.cs.put.poznan.pl/pboinski/files/SBD2/MongoDB_tutorial.pdf
przy opracowywaniu punktów dotyczących agregacji danych oraz łączenia kolekcji.

```
show dbs

admin   0.000GB
config  0.000GB
local   0.000GB
```

## 3. Tworzenie przykładowego modelu

Większość ćwiczeń będziesz wykonywał wykorzystując bazę danych *scott*, która „od zawsze” była obecna w relacyjnej bazie danych ORACLE. Jej struktura jest banalnie prosta. Składa się z 4 tabel, z których my wykorzystamy tylko dwie: *dept* oraz *emp*. Poniższy diagram zapewne wszystko wyjaśnia.




**Ważna uwaga:** zwróćmy uwagę, że do bazy NoSQL-owej przeniesiemy, *de facto* bez żadnych zmian, strukturę relacyjną, gdzie dane o pracownikach (*emp*) oraz oddziałach (*dept*) przechowywane są w dwóch kolekcjach połączonych niejawnie kluczem w polu *deptno* (piszemy *niejawnym*, gdyż MongoDB nie obsługuje kluczy w rozumieniu baz relacyjnych).

Baza MongoDB potrafi obsługiwać takie modele za pomocą operatora `$lookup`, jednak nie jest to działanie zgodne z duchem baz NoSQL-owych. W dalszej części ćwiczenia przebudujemy nasz model i dane z kolekcji *dept* zostaną „upakowane” do kolekcji *emp* (użyjemy dokumentów zagnieżdżonych).

Utworzymy więc nową bazę danych o nazwie *scott* (wcześniej ją dla pewności wykasujemy, aby rozpocząć pracę od pustej bazy) i wstawimy tam odpowiednie dokumenty do dwóch kolekcji: *emp* oraz *dept*. Poniżej podajemy odpowiednie polecenia.

```
use scott
switched to db scott

db.dropDatabase()
{ "dropped" : "scott", "ok" : 1 }

use scott
switched to db scott
```

```
db.emp.insert([
{"empno":7369, "ename":"SMITH", "job":"CLERK", "mgr":7902,
 "hiredate":"17-DEC-1980", "sal":800, "comm":null, "deptno":20},
{"empno":7499, "ename":"ALLEN", "job":"SALESMAN", "mgr":7698,
 "hiredate":"20-FEB-1981", "sal":1600, "comm":300, "deptno":30},
{"empno":7521, "ename":"WARD", "job":"SALESMAN", "mgr":7698,
 "hiredate":"22-FEB-1981", "sal":1250, "comm":500, "deptno":30},
{"empno":7566, "ename":"JONES", "job":"MANAGER", "mgr":7839,
 "hiredate":"02-APR-1981", "sal":2975, "comm":null, "deptno":20},
{"empno":7654, "ename":"MARTIN", "job":"SALESMAN", "mgr":7698,
 "hiredate":"28-SEP-1981", "sal":1250, "comm":1400, "deptno":30},
{"empno":7698, "ename":"BLAKE", "job":"MANAGER", "mgr":7839,
 "hiredate":"01-MAY-1981", "sal":2850, "comm":null, "deptno":30},
{"empno":7782, "ename":"CLARK", "job":"MANAGER", "mgr":7839,
 "hiredate":"09-JUN-1981", "sal":2450, "comm":null, "deptno":10},
{"empno":7788, "ename":"SCOTT", "job":"ANALYST", "mgr":7566,
 "hiredate":"19-APR-1987", "sal":3000, "comm":null, "deptno":20},
{"empno":7839, "ename":"KING", "job":"PRESIDENT", "mgr":null,
 "hiredate":"17-NOV-1981", "sal":5000, "comm":null, "deptno":10},
{"empno":7844, "ename":"TURNER", "job":"SALESMAN", "mgr":7698,
 "hiredate":"08-SEP-1981", "sal":1500, "comm":0, "deptno":30},
{"empno":7876, "ename":"ADAMS", "job":"CLERK", "mgr":7788,
 "hiredate":"23-MAY-1987", "sal":1100, "comm":null, "deptno":20},
{"empno":7900, "ename":"JAMES", "job":"CLERK", "mgr":7698,
 "hiredate":"03-DEC-1981", "sal":950, "comm":null, "deptno":30},
{"empno":7902, "ename":"FORD", "job":"ANALYST", "mgr":7566,
 "hiredate":"03-DEC-1981", "sal":3000, "comm":null, "deptno":20},
{"empno":7934, "ename":"MILLER", "job":"CLERK", "mgr":7782,
 "hiredate":"23-JAN-1982", "sal":1300, "comm":null, "deptno":10}
])

BulkWriteResult({
        "writeErrors" : [ ],
        "writeConcernErrors" : [ ],
        "nInserted" : 14,
        "nUpserted" : 0,
        "nMatched" : 0,
        "nModified" : 0,
        "nRemoved" : 0,
        "upserted" : [ ]
})
```

```
db.dept.insert([
{"deptno":10, "dname":"ACCOUNTING", "loc":"NEW YORK"},
{"deptno":20, "dname":"RESEARCH", "loc":"DALLAS"},
{"deptno":30, "dname":"SALES", "loc":"CHICAGO"},
{"deptno":40, "dname":"OPERATIONS", "loc":"BOSTON"}
])

BulkWriteResult({
        "writeErrors" : [ ],
        "writeConcernErrors" : [ ],
        "nInserted" : 4,
        "nUpserted" : 0,
        "nMatched" : 0,
        "nModified" : 0,
```

```
        "nRemoved" : 0,
        "upserted" : [ ]
})
```

## 4. Wyświetlanie danych

W kolejnym punkcie wyświetlimy w różnym układzie zapisane dane. Możemy wydać np. polecenie wyświetlające, jakie mamy w bieżącej bazie kolekcje. Bazę bieżącą wybieramy poleceniem **use**.

```
db.getCollectionNames()

[ "dept", "emp" ]
```

Możemy też wylistować zawartość kolekcji. Będzie to odpowiednik SQL-owego polecenie `SELECT * FROM tabela`.

```
db.dept.find()

{ "_id" : ObjectId("6029b35b99180152a1047260"), "deptno" : 10, "dname" :
"ACCOUNTING", "loc" : "NEW YORK" }
{ "_id" : ObjectId("6029b35b99180152a1047261"), "deptno" : 20, "dname" :
"RESEARCH", "loc" : "DALLAS" }
{ "_id" : ObjectId("6029b35b99180152a1047262"), "deptno" : 30, "dname" :
"SALES", "loc" : "CHICAGO" }
{ "_id" : ObjectId("6029b35b99180152a1047263"), "deptno" : 40, "dname" :
"OPERATIONS", "loc" : "BOSTON" }
```

Wartość unikalnego klucza **_id** zwykle nie jest nam potrzebna bezpośrednio, więc najczęściej będziemy pomijali go i nie wyświetlali.

**Uwaga:** w MongoDB trzeba się przyzwyczaić do dużej ilości nawiasów, głównie klamrowych :-)

```
db.dept.find({}, {"_id":0})

{ "deptno" : 10, "dname" : "ACCOUNTING", "loc" : "NEW YORK" }
{ "deptno" : 20, "dname" : "RESEARCH", "loc" : "DALLAS" }
{ "deptno" : 30, "dname" : "SALES", "loc" : "CHICAGO" }
{ "deptno" : 40, "dname" : "OPERATIONS", "loc" : "BOSTON" }
```

Możemy też wyświetlić tylko wybrane pole/pola i zawęzić ilość wyświetlanych dokumentów do tych, które spełniają warunek zapytania. Jest to odpowiednik polecenia SQL-owego
`SELECT kol1, kol2 FROM tabela WHERE kol1='warość'`

Zwróćmy uwagę, że tym razem polecenie zostało czytelnie sformatowane. Użyto wcięć oraz przejść do nowej linii. Zawsze warto tak robić, zwłaszcza, gdy polecenia są dłuższe i wówczas ciągnięcie ich w jednej linii jest bardzo niedobrym przyzwyczajeniem.

Wartość 0 w sekcji wyboru pól oznacza „nie wyświetlaj tego pola”, wartość 1 (lub jakakolwiek inna wartość różna od 0, w praktyce jednak najczęściej wpisuje się właśnie wartość 1) oznacza „wyświetl to pole”. Pozostałe, niewymienione jawnie pola, nie będą wyświetlane.

```
db.emp.find(
  {deptno:30},
  {_id:0, ename:1, job:1, deptno:1}
)
```

```
{ "ename" : "ALLEN", "job" : "SALESMAN", "deptno" : 30 }
{ "ename" : "WARD", "job" : "SALESMAN", "deptno" : 30 }
{ "ename" : "MARTIN", "job" : "SALESMAN", "deptno" : 30 }
{ "ename" : "BLAKE", "job" : "MANAGER", "deptno" : 30 }
{ "ename" : "TURNER", "job" : "SALESMAN", "deptno" : 30 }
{ "ename" : "JAMES", "job" : "CLERK", "deptno" : 30 }
```

Używając metody **pretty** wszystkie pola dokumentów można wyświetlić w wygodnie sformatowanej postaci (ale zajmuje to więcej linii na ekranie).

```
db.dept.find().pretty()

{
        "_id" : ObjectId("602a5fcf5e42f0590b4c5ac2"),
        "deptno" : 10,
        "dname" : "ACCOUNTING",
        "loc" : "NEW YORK"
}
{
        "_id" : ObjectId("602a5fcf5e42f0590b4c5ac3"),
        "deptno" : 20,
        "dname" : "RESEARCH",
        "loc" : "DALLAS"
}
{
        "_id" : ObjectId("602a5fcf5e42f0590b4c5ac4"),
        "deptno" : 30,
        "dname" : "SALES",
        "loc" : "CHICAGO"
}
{
        "_id" : ObjectId("602a5fcf5e42f0590b4c5ac5"),
        "deptno" : 40,
        "dname" : "OPERATIONS",
        "loc" : "BOSTON"
}
```

Jeżeli chcemy dowiedzieć się, ile mamy dokumentów w kolekcji możemy użyć metody **count**. Będzie to odpowiednik SQL-owego polecenia `SELECT COUNT(*) FROM tabela`.

```
db.emp.find().count()
14

db.dept.find().count()
4
```

Często zachodzi potrzeba posortowania danych. Używamy wówczas metody **sort**. Wartość **1** oznacza sortowanie do najmniejsze do największej (dla tekstów od A do Z) a wartość **-1** odwrotnie.

```
db.emp.find(
  {},
  {_id:0, ename:1, sal:1}
).sort(
  {sal:-1}
)

{ "ename" : "KING", "sal" : 5000 }
{ "ename" : "SCOTT", "sal" : 3000 }
```

```
{ "ename" : "FORD", "sal" : 3000 }
{ "ename" : "JONES", "sal" : 2975 }
{ "ename" : "BLAKE", "sal" : 2850 }
{ "ename" : "CLARK", "sal" : 2450 }
{ "ename" : "ALLEN", "sal" : 1600 }
{ "ename" : "TURNER", "sal" : 1500 }
{ "ename" : "MILLER", "sal" : 1300 }
{ "ename" : "WARD", "sal" : 1250 }
{ "ename" : "MARTIN", "sal" : 1250 }
{ "ename" : "ADAMS", "sal" : 1100 }
{ "ename" : "JAMES", "sal" : 950 }
{ "ename" : "SMITH", "sal" : 800 }
```

Na koniec zwróćmy uwagę, że w przypadku pomyłki (literówka) w nazwie pola MongoDB nie zasygnalizuje błędu, tylko po prostu nic się nie wyświetli. Trzeba więc uważać, bo takie zachowanie bazy może być mylące. Poniżej zamiast *deptno* oraz *dname* wpisano odpowiednio *depton* oraz *dnamme*. W drugim wypadku wyświetliły się 4 puste dokumenty.

```
db.dept.find( {deptno:10}, {_id:0} )
{ "deptno" : 10, "dname" : "ACCOUNTING", "loc" : "NEW YORK" }

db.dept.find( {depton:10}, {_id:0} )

db.dept.find( {}, {_id:0, dnamme:1} )
{  }
{  }
{  }
{  }
```

## 5. Wstawianie danych

Do kolekcji *dept* dopiszemy dwie pozycje. Proszę zwrócić uwagę, że dane, które wstawiamy mają inną strukturę (dodajemy kolejne pole *country*). W bazach relacyjnych takie coś jest niemożliwe, gdyż wszystkie wiersze muszą mieć taką samą strukturę. Tutaj możemy tak zrobić. Oczywiście w praktyce nie powinno być tak, że każdy dokument w kolekcji jest zupełnie inny, raczej powinny mieć albo taką samą albo podobną strukturę.

Jeżeli natomiast rzeczywiście poszczególne dokumenty mają zupełnie inną strukturę, to należy zastanowić się, czy na pewno nasza baza jest dobrze zaprojektowana od strony logicznej.

```
db.dept.insert({"deptno":50, "dname":"SERVIS", "loc":"ZIELONA GÓRA"})
WriteResult({ "nInserted" : 1 })

db.dept.insert({"deptno":60, "dname":"sprzedaż", "loc":"ZIELONA GÓRA",
"country":"Poland"})
WriteResult({ "nInserted" : 1 })

db.dept.find({},{_id:0, dname:1, loc:1, country:1})
{ "dname" : "SERVIS", "loc" : "ZIELONA GÓRA" }
{ "dname" : "sprzedaż", "loc" : "ZIELONA GÓRA", "country" : "Poland" }
```

Jeżeli z jakiegoś powodu zależy nam, aby unikalny klucz **_id** nie był generowany przez system, tylko chcemy wpisać tam swoją własną wartość możemy tak zrobić. Trzeba jednak wówczas samodzielnie zadbać o unikalność kluczy, gdyż **duplikaty nie są dopuszczalne**.

```
db.test.insert({"pole":"abc"})
WriteResult({ "nInserted" : 1 })

db.test.insert({"pole":"def"})
WriteResult({ "nInserted" : 1 })

db.test.insert({"_id":100, "pole":"xyz"})
WriteResult({ "nInserted" : 1 })

db.test.find()
{ "_id" : ObjectId("602c6a759f215a35ae4ca4c0"), "pole" : "abc" }
{ "_id" : ObjectId("602c6a759f215a35ae4ca4c1"), "pole" : "def" }
{ "_id" : 100, "pole" : "xyz" }

db.test.insert({"_id":100, "pole":"xyz"})
WriteResult({
        "nInserted" : 0,
        "writeError" : {
                "code" : 11000,
                "errmsg"   :   "E11000   duplicate   key   error   collection:
scott.test index: _id_ dup key: { _id: 100.0 }"
        }
})
```

## 6. Modyfikacja danych

Zmienimy zawartość wybranego (dokładnie jednego) dokumentu w kolekcji *dept*.

```
db.emp.update(
  {empno:7369},
  {$set:{sal:950}}
)

WriteResult({ "nMatched" : 1, "nUpserted" : 0, "nModified" : 1 })
```

```
db.emp.find({},{_id:0, ename:1, sal:1})

{ "ename" : "SMITH", "sal" : 950 }
{ "ename" : "ALLEN", "sal" : 1600 }
{ "ename" : "WARD", "sal" : 1250 }
{ "ename" : "JONES", "sal" : 2975 }
{ "ename" : "MARTIN", "sal" : 1250 }
{ "ename" : "BLAKE", "sal" : 2850 }
{ "ename" : "CLARK", "sal" : 2450 }
{ "ename" : "SCOTT", "sal" : 3000 }
{ "ename" : "KING", "sal" : 5000 }
{ "ename" : "TURNER", "sal" : 1500 }
{ "ename" : "ADAMS", "sal" : 1100 }
{ "ename" : "JAMES", "sal" : 950 }
{ "ename" : "FORD", "sal" : 3000 }
{ "ename" : "MILLER", "sal" : 1300 }
```

Tym razem chcemy zmodyfikować dane więcej niż jednego dokumentu (każdemu pracownikowi powiększamy zarobki o 1000). I tutaj napotkamy pewną **niespodziankę**. Okazuje się, że poniższe pole-

cenie zmodyfikuje tylko **pierwszy** dopasowany dokument. Aby polecenie zadziałało zgodnie z naszymi oczekiwaniami, należy dodatkowo uzupełnić je o fragment **multi:true**. Polecenie to wpisujemy w dodatkowej (trzeciej) sekcji z nawiasami klamrowymi. W tej sekcji możemy podawać różne dodatkowe opcje (szczegóły patrz dokumentacja).

```
db.emp.find({},{sal:1, _id:0})

{ "sal" : 800 }
{ "sal" : 1600 }
{ "sal" : 1250 }
{ "sal" : 2975 }
{ "sal" : 1250 }
{ "sal" : 2850 }
{ "sal" : 2450 }
{ "sal" : 3000 }
{ "sal" : 5000 }
{ "sal" : 1500 }
{ "sal" : 1100 }
{ "sal" : 950 }
{ "sal" : 3000 }
{ "sal" : 1300 }

db.emp.update( {}, {$inc:{sal:1000}} )
WriteResult({ "nMatched" : 1, "nUpserted" : 0, "nModified" : 1 })

db.emp.find({},{sal:1, _id:0})

{ "sal" : 1800 }
{ "sal" : 1600 }
{ "sal" : 1250 }
{ "sal" : 2975 }
{ "sal" : 1250 }
{ "sal" : 2850 }
{ "sal" : 2450 }
{ "sal" : 3000 }
{ "sal" : 5000 }
{ "sal" : 1500 }
{ "sal" : 1100 }
{ "sal" : 950 }
{ "sal" : 3000 }
{ "sal" : 1300 }
```

```
db.emp.update(
  {},
  {$inc:{sal:1000}},
  {"multi":true}
)
WriteResult({ "nMatched" : 14, "nUpserted" : 0, "nModified" : 14 })

db.emp.find({},{sal:1, _id:0}

{ "sal" : 2800 }
{ "sal" : 2600 }
{ "sal" : 2250 }
{ "sal" : 3975 }
{ "sal" : 2250 }
{ "sal" : 3850 }
```

```
{ "sal" : 3450 }
{ "sal" : 4000 }
{ "sal" : 6000 }
{ "sal" : 2500 }
{ "sal" : 2100 }
{ "sal" : 1950 }
{ "sal" : 4000 }
{ "sal" : 2300 }
```

## 7. Kasowanie danych

Wszystkie dane z wybranej kolekcji wykasujemy za pomocą metody **DeleteMany**. Jest też metoda **DeleteOne**, która wykasuje tylko pierwszy dokument spełniający warunek selekcji.

```
db.emp.deleteMany({})
{ "acknowledged" : true, "deletedCount" : 14 }
```

## 8. Operatory

Pisząc polecenia do selekcji danych bardzo często chcemy w jakiś sposób zawęzić ilość wyświetlanych danych. Wówczas nieocenione stają się operatory. W bazie MongoDB operatorów jest bardzo dużo. Pełną ich listę znajdziemy w dokumentacji systemu tutaj
https://docs.mongodb.com/manual/reference/operator/query/index.html
Te najczęściej używane służą do porównywania. Nazwa operatora na początku zawsze zawiera **znak dolara**. Są to (przepisujemy za dokumentacją):

| | |
|---|---|
| **$eq** | Matches values that are equal to a specified value. |
| **$gt** | Matches values that are greater than a specified value. |
| **$gte** | Matches values that are greater than or equal to a specified value. |
| **$in** | Matches any of the values specified in an array. |
| **$lt** | Matches values that are less than a specified value. |
| **$lte** | Matches values that are less than or equal to a specified value. |
| **$ne** | Matches all values that are not equal to a specified value. |
| **$nin** | Matches none of the values specified in an array. |
| **$and** | Joins query clauses with a logical AND returns all documents that match the conditions of both clauses. |
| **$not** | Inverts the effect of a query expression and returns documents that do not match the query expression. |
| **$nor** | Joins query clauses with a logical NOR returns all documents that fail to match both clauses. |
| **$or** | Joins query clauses with a logical OR returns all documents that match the conditions of either clause. |
| **$type** | Selects documents if a field is of the specified type. |
| **$where** | Matches documents that satisfy a JavaScript expression. |
| **$exists** | Matches documents that have the specified field. |

Poniżej jeden wybrany przykład.

```
db.emp.find(
      {deptno:{$in:[10,20]}},
      {_id:0, ename:1, deptno:1}
)

{ "ename" : "SMITH", "deptno" : 20 }
{ "ename" : "JONES", "deptno" : 20 }
{ "ename" : "CLARK", "deptno" : 10 }
{ "ename" : "SCOTT", "deptno" : 20 }
{ "ename" : "KING", "deptno" : 10 }
{ "ename" : "ADAMS", "deptno" : 20 }
{ "ename" : "FORD", "deptno" : 20 }
{ "ename" : "MILLER", "deptno" : 10 }
```

# 9. Agregacje danych

Pracując z bazami danych często potrzebujemy nie tyle pobrać zgromadzone dane ale je też jakoś przetworzyć. Przykładowo moglibyśmy chcieć z kolekcji *emp* wyliczyć sumę zarobków (pole *sal*) pracowników pracujących na poszczególnych stanowiskach (pole *job*). Wówczas mówimy, że będziemy dane **agregować** (łączyć jakoś ze sobą, podsumować dane elementarne).

Uwagi:

a) pola dokumentów poprzedzamy znakiem **$**,
b) agregację uzyskujemy poprzez zastosowanie operatora **$group**. Pierwszym parametrem jest wyrażenie grupujące specyfikowane jako wartość pola **_id**. Jeżeli chcemy, aby wszystkie dokumenty były w jednej grupie ustawiamy wartość tego pola na **null**. Pozostałe parametry definiują, w jaki sposób będą obliczane agregaty,
c) jeżeli chcemy z wyniku agregacji odfiltrować pewne agregaty (grupy), możemy użyć operatora **$match**, który pełni podobna rolę, jak klauzula **HAVING** w języku SQL (na przykład wybrać do agregacji tylko dane pracowników z oddziału *deptno=20*),
d) zagadnienie tworzenia agregacji jest dość złożone (w zasadzie w każdej bazie danych, nie tylko w bazie MongoDB). Warto zapoznać się z dokumentacją źródłową, aby zobaczyć na przykład ile różnych operatorów możemy używać:
https://docs.mongodb.com/manual/reference/operator/aggregation/
https://docs.mongodb.com/manual/aggregation/

Wyświetlamy sumę zarobków wszystkich pracowników.

```
db.emp.aggregate(
   [{$group: {
         _id: null,
         suma_zarobkow: {$sum: "$sal"},
   }}]
)

{ "_id" : null, "suma_zarobkow" : 29025 }
```

Teraz robimy podobnie, jak poprzednio ale sumujemy zarobki na poszczególnych stanowiskach (pole *job*). Dodatkowo wyświetlamy minimalne zarobki na każdym stanowisku.

```
db.emp.aggregate(
    [{$group: {
          _id: "$job",
          suma_zarobkow: {$sum: "$sal"},
          zarobki_minimalne: {$min: "$sal"}
    }}]
)

{ "_id" : "CLERK", "suma_zarobkow" : 4150, "zarobki_minimalne" : 800 }
{ "_id" : "SALESMAN", "suma_zarobkow" : 5600, "zarobki_minimalne" : 1250 }
{ "_id" : "MANAGER", "suma_zarobkow" : 8275, "zarobki_minimalne" : 2450 }
{ "_id" : "PRESIDENT", "suma_zarobkow" : 5000, "zarobki_minimalne" : 5000 }
{ "_id" : "ANALYST", "suma_zarobkow" : 6000, "zarobki_minimalne" : 3000 }
```

Modyfikujemy poprzedni przykład tak, aby odfiltrować grupy, gdzie suma zarobków jest mniejsza niż 5000. W stosunku do poprzednio otrzymanego wyniku widzimy, że nie wyświetliła się pozycja pierwsza (gdzie suma zarobków wynosi 4150, czyli mniej niż założone 5000). Pozycja 4 natomiast wyświetla się, bo mamy zarobki dokładnie równe 5000 (użyliśmy operatora **$gte**, czyli „greater than or equal").

```
db.emp.aggregate(
    [{$group: {
          _id: "$job",
          suma_zarobkow: {$sum: "$sal"},
          zarobki_minimalne: {$min: "$sal"}
    }},
    {$match: {
          suma_zarobkow: {$gte: 5000}
    }}]
)

{ "_id" : "SALESMAN", "suma_zarobkow" : 5600, "zarobki_minimalne" : 1250 }
{ "_id" : "MANAGER", "suma_zarobkow" : 8275, "zarobki_minimalne" : 2450 }
{ "_id" : "PRESIDENT", "suma_zarobkow" : 5000, "zarobki_minimalne" : 5000 }
{ "_id" : "ANALYST", "suma_zarobkow" : 6000, "zarobki_minimalne" : 3000 }
```

## 10. Usuwanie kolekcji i usuwanie bazy danych

Mamy tutaj dwa proste w użyciu polecenia, niewymagające żadnego komentarza. Jedynie uczulamy, że polecenie kasujące bazę danych kasuje bazę **bieżącą**, a więc tą ustawioną poleceniem **use**.

```
db.emp.drop()
true

db.dept.drop()
true

db.dropDatabase()
{ "dropped" : "scott", "ok" : 1 }
```

# 11. Dokumenty z zagnieżdżeniami

Jak pokazano na początku opracowania bezpośrednie przeniesienie modelu relacyjnego do MongoDB jest technicznie jak najbardziej możliwe. Wówczas każda tabela staje się kolekcją. Działanie takie jednak przeczy istocie baz klasy NoSQL. W gruncie rzeczy po to one zostały wymyślone, aby dać większą swobodę w gromadzeniu danych o bardzo różnej, i często zmieniającej się w czasie, strukturze. Jak bowiem pamiętamy bazy relacyjne najlepiej sprawdzają się tam, gdzie raz ustalona (na etapie projektu) struktura danych nie ulega zbyt wielkim i zbyt częstym modyfikacjom. Koncepcja kluczy głównych i obcych nie nadaje się do zastosowania w bazach NoSQL. A przecież utworzenie w naszym przypadku kolekcji *emp* oraz *dept* w gruncie rzeczy zachowało pierwotny układ kluczy.

Aby przechowywać dokładnie te same dane, co w oryginalnych tabelach relacyjnych *dept* oraz *emp* zdecydowanie naturalniej w bazie NoSQL jest zrealizować to z wykorzystaniem **dokumentów zagnieżdżonych (osadzonych; embedded)**, gdzie zapisane będą dane z tabeli *dept*. Nowa struktura danych będzie więc wyglądała jak niżej (pokazano tylko dwa pierwsze dokumenty).

Dostrzegamy natychmiast dużą ilość powtórzonych danych. Pozbywając się tabeli słownikowej *dept* godzimy się jednocześnie na dużą ilość **dubli danych**. Ale na to już nic nie poradzimy, taka jest istota baz NoSQL, że spójność i swego rodzaju kompresja danych poświęcana jest na rzecz większej elastyczności modelu. W modelu z zagnieżdżeniami nie ma żadnego problemy, aby dla każdego pracownika wpisać nieco inne dane w polu tablicowym *deptno*.

Dodatkowo naszą kolekcję rozbudowaliśmy wprowadzając **tablicę osadzonych dokumentów**, gdzie zapiszemy dane, kiedy pracownik wziął urlop i na ile dni. W innej **tablicy łańcuchów znakowych** wpisujemy języki programowania, które opanował pracownik.

Ponieważ przy pracy z dłuższymi dokumentami JSON łatwo się pomylić i np. nie w tym miejscy wstawić nawias, warto czasem skorzystać z dostępnych w sieci **walidatorów i formaterów JSON-a.** Jeden przykładowy można znaleźć tutaj https://jsonformatter.org/

```
db.emp2.insert(
[{
   "empno":7369,
   "ename":"SMITH",
   "job":"CLERK",
   "mgr":7902,
   "hiredate":"17-DEC-1980",
   "sal":800,
   "comm":null,
   "deptno":{
         "deptno":20,
         "dname":"RESEARCH",
         "loc":"DALLAS"
   },
   "languages":["R", "Python", "C++", "JavaScript"],
   "vacation":[
         {"from":"25-JAN-1982", "days":5},
         {"from":"12-FEB-1988", "days":12},
         {"from":"11-DEC-1989", "days":3}
   ]
},{
   "empno":7499,
   "ename":"ALLEN",
   "job":"SALESMAN",
   "mgr":7698,
```

```
    "hiredate":"20-FEB-1981",
    "sal":1600,
    "comm":300,
    "deptno":{
          "deptno":30,
          "dname":"SALES",
          "loc":"CHICAGO"
    },
    "languages":["JavaScript"],
    "vacation":[
          {"from":"01-MAY-1982", "days":4},
          {"from":"12-FEB-1988", "days":15}
    ]
}]
)
```

Po wyświetleniu w konsoli jednego dokumentu z użyciem metody **pretty** otrzymujemy ładnie sformatowanego JSON-a (choć to formatowanie nie jest zbyt zwarte).

```
{
        "_id" : ObjectId("602c62d69f215a35ae4ca4be"),
        "empno" : 7369,
        "ename" : "SMITH",
        "job" : "CLERK",
        "mgr" : 7902,
        "hiredate" : "17-DEC-1980",
        "sal" : 800,
        "comm" : null,
        "deptno" : {
                "deptno" : 20,
                "dname" : "RESEARCH",
                "loc" : "DALLAS"
        },
        "languages" : [
                "R",
                "Python",
                "C++",
                "JavaScript"
        ],
        "vacation" : [
                {
                        "from" : "25-JAN-1982",
                        "days" : 5
                },
                {
                        "from" : "12-FEB-1988",
                        "days" : 12
                },
                {
                        "from" : "11-DEC-1989",
                        "days" : 3
                }
        ]
}
```

Dostęp do pól zagnieżdżonych można uzyskać używając operatora . (kropka). Dwa przykłady podano poniżej. W pierwszym przykładzie zostaną znalezione dokumenty, gdzie pracownik zna język R a w drugim, gdzie długość urlopu wynosiła 4 dni.

```
db.emp2.find(
   {"languages":"R"},
   {_id:0,ename:1,languages:1}
).pretty()

{
        "ename" : "SMITH",
        "languages" : [
                "R",
                "Python",
                "C++",
                "JavaScript"
        ]
}
```

```
db.emp2.find(
   {"vacation.days":4},
   {_id:0,ename:1, "vacation.days":1}
).pretty()

{
        "ename" : "ALLEN",
        "vacation" : [
                {
                        "days" : 4
                },
                {
                        "days" : 15
                }
        ]
}
```

## 12. Łączenie danych z kilku kolekcji, referencje

Wspominano już kilka razy, że bezpośrednie przeniesienie modelu relacyjnego do bazy NoSQL jest technicznie wykonalne, jednak nieco przeczy idei baz nierelacyjnych. Czasami jednak wygodniej jest użyć referencji, gdy np. danych w strukturach zagnieżdżonych zaczyna być bardzo dużo i bardzo wiele z nich powtarza się.

Aby dać użytkownikowi możliwość pracy z kilkoma kolekcjami naraz twórcy bazy MongoDB (od wersji 3.2) zaimplementowali operator **`$lookup`**. Poniżej pokazujemy przykład jego użycia.

Pierwszy przykład pokazuje, jak połączyć ze sobą dane z dwóch kolekcji. Wyświetlane są wszystkie dane. Na wydruku pokazujemy tylko jeden dokument, reszta jest analogiczna.

```
db.emp.aggregate([
  {
    $lookup:
    {
      from: "dept",
      localField: "deptno",
      foreignField: "deptno",
      as: "dept_data"
    }
```

```
  }
]).pretty()

{
        "_id" : ObjectId("602d7a3cc42f9ba327345a5e"),
        "empno" : 7369,
        "ename" : "SMITH",
        "job" : "CLERK",
        "mgr" : 7902,
        "hiredate" : "17-DEC-1980",
        "sal" : 800,
        "comm" : null,
        "deptno" : 20,
        "dept_data" : [
                {
                        "_id" : ObjectId("602d7a3cc42f9ba327345a6d"),
                        "deptno" : 20,
                        "dname" : "RESEARCH",
                        "loc" : "DALLAS"
                }
        ]
}
```

Następnie rozbudujmy powyższy przykład i dodajmy projekcję (wskazujemy, które pola chcemy wyświetlić). Pewną niedogodnością tego sposobu prezentacji wyniku jest to, że nazwy zespołów (pole *dname*) umieszczone są jako elementy tablicy.

```
db.emp.aggregate([
  {
    $lookup:
    {
      from: "dept",
      localField: "deptno",
      foreignField: "deptno",
      as: "dept_data"
    }
  },
  {$project:{"_id":0, "empno":1,"ename":1, "dept_data.dname":1}}
])

{ "empno" : 7369, "ename" : "SMITH", "dept_data" : [ { "dname" : "RESEARCH" } ] }
{ "empno" : 7499, "ename" : "ALLEN", "dept_data" : [ { "dname" : "SALES" } ] }
{ "empno" : 7521, "ename" : "WARD", "dept_data" : [ { "dname" : "SALES" } ] }
{ "empno" : 7566, "ename" : "JONES", "dept_data" : [ { "dname" : "RESEARCH" } ] }
{ "empno" : 7654, "ename" : "MARTIN", "dept_data" : [ { "dname" : "SALES" } ] }
{ "empno" : 7698, "ename" : "BLAKE", "dept_data" : [ { "dname" : "SALES" } ] }
{ "empno" : 7782, "ename" : "CLARK", "dept_data" : [ { "dname" : "ACCOUNTING" } ] }
{ "empno" : 7788, "ename" : "SCOTT", "dept_data" : [ { "dname" : "RESEARCH" } ] }
{ "empno" : 7839, "ename" : "KING", "dept_data" : [ { "dname" : "ACCOUNTING" } ] }
{ "empno" : 7844, "ename" : "TURNER", "dept_data" : [ { "dname" : "SALES" } ] }
{ "empno" : 7876, "ename" : "ADAMS", "dept_data" : [ { "dname" : "RESEARCH" } ] }
{ "empno" : 7900, "ename" : "JAMES", "dept_data" : [ { "dname" : "SALES" } ] }
{ "empno" : 7902, "ename" : "FORD", "dept_data" : [ { "dname" : "RESEARCH" } ] }
{ "empno" : 7934, "ename" : "MILLER", "dept_data" : [ { "dname" : "ACCOUNTING" } ] }
```

Aby nie wyświetlać tablicy tylko samą nazwę działu (pole *dname*) w nieco inny sposób należy „dobrać" się do danych. Użyjemy operatora **`$arrayElemAt`**. W tablicy mamy tylko jeden element (mający indeks 0) więc wpisujemy jako drugi parametr właśnie tą wartość 0.

```
db.emp.aggregate([
  {
```

```
    $lookup:
    {
      from: "dept",
      localField: "deptno",
      foreignField: "deptno",
      as: "dept_data"
    }
  },
  {
    $project:
    {
      "_id":0,
      "empno":1,
      "ename":1,
      "dept_data": {$arrayElemAt:["$dept_data.dname", 0]}
   }
  }
])
```

```
{ "empno" : 7369, "ename" : "SMITH", "dept_data" : "RESEARCH" }
{ "empno" : 7499, "ename" : "ALLEN", "dept_data" : "SALES" }
{ "empno" : 7521, "ename" : "WARD", "dept_data" : "SALES" }
{ "empno" : 7566, "ename" : "JONES", "dept_data" : "RESEARCH" }
{ "empno" : 7654, "ename" : "MARTIN", "dept_data" : "SALES" }
{ "empno" : 7698, "ename" : "BLAKE", "dept_data" : "SALES" }
{ "empno" : 7782, "ename" : "CLARK", "dept_data" : "ACCOUNTING" }
{ "empno" : 7788, "ename" : "SCOTT", "dept_data" : "RESEARCH" }
{ "empno" : 7839, "ename" : "KING", "dept_data" : "ACCOUNTING" }
{ "empno" : 7844, "ename" : "TURNER", "dept_data" : "SALES" }
{ "empno" : 7876, "ename" : "ADAMS", "dept_data" : "RESEARCH" }
{ "empno" : 7900, "ename" : "JAMES", "dept_data" : "SALES" }
{ "empno" : 7902, "ename" : "FORD", "dept_data" : "RESEARCH" }
{ "empno" : 7934, "ename" : "MILLER", "dept_data" : "ACCOUNTING" }
```

## 13. Polecenia do samodzielnego wykonania

Przygotuj samodzielnie plik w strukturze JSON, w którym wpiszesz dane **jednej kolekcji** zawierającej około **30-40** dokumentów. Dokumenty powinny zawierać przynajmniej:

a) jeden dokument zagnieżdżony,
b) dwie tablice.

Dobierz tak dane i model, aby były uzasadnione utworzenie poszczególnych dokumentów o **podobnej ale jednak nie identycznej strukturze**. Przykładowo jeżeli w naszej kolekcji będziemy chcieli gromadzić dane o sprzedanych towarach to prawdopodobnie nieco inne informacje będą zarejestrowane o telewizorach, inne o sprzęcie komputerach a jeszcze inne o kartach graficznych.

Zanim rozpoczniesz realizację zadań **skonsultuj swój wybór z prowadzącym, który zatwierdzi go** (lub poprosi o wprowadzenie poprawek).

Następnie wykonaj analogiczne ćwiczenia co w punktach 1-12, tyle, że na swoim zestawie danych, czyli:

a) wyświetlanie danych,

b) wstawianie nowych danych,
c) modyfikacja istniejących danych,
d) kasowanie danych,
e) agregowanie danych,
f) łączenia kilku kolekcji (musisz stworzyć dodatkową kolekcję, oprócz tej podstawowej, o której mowa wyżej.

Postaraj się, aby utworzone polecenia nie były zbyt banalne, korzystaj np. z różnych operatorów a pisząc agregacje nie ograniczaj się tylko do najprostszych.