MongoDB

mini przewodnik

v20180414

Plan

W tym przewodniku dowiesz się jak:

- utworzyć bazę danych, kolekcje i wstawić dokumenty,
- odpytać bazę danych o dokumenty (projekcja, selekcja),
- konstruować zapytania wyliczające agregaty,
- posługiwać się modelem opartym na referencjach i modelem opartym na zagnieżdżonych (osadzonych) dokumentach,
- modyfikować i usuwać dokumenty.

1 Środowisko pracy

1.1 Uruchomienie konsoli MongoDB

W celu uruchomienia konsoli *MongoDB* uruchom terminal i wydaj polecenie **mongo**.

```
student@lab-ai-sbd: mongo
MongoDB shell version v3.4.14
connecting to: mongodb://127.0.0.1:27017
MongoDB server version: 3.4.10
Server has startup warnings:
2018-04-13T16:52:41.294+0200 I CONTROL  [initandlisten]
2018-04-13T16:52:41.294+0200 I CONTROL  [initandlisten] ** WARNING: Access control is not enabled for the database.
2018-04-13T16:52:41.294+0200 I CONTROL  [initandlisten] **          Read and write access to data and configuration is unrestricted.
2018-04-13T16:52:41.294+0200 I CONTROL  [initandlisten]
>
```

Zignoruj ewentualne komunikaty dotyczące braku kontroli dostępu. W produkcyjnym systemie oczywiście należy uwzględnić w/w. Wyjście z konsoli *MongoDB* możliwe jest poprzez wydanie polecenia **exit**. Na razie jednak pozostań w konsoli.

1.2 Sprawdzenie środowiska

Zanim utworzymy nową bazę danych sprawdzimy, jakie bazy aktualnie są dostępne. W tym celu wywołaj polecenie **show dbs**.

```
show dbs
admin 0.000GB
local  0.000GB
```

Bazy danych **admin** i **local** to systemowe bazy danych. W tym przypadku nie ma żadnej bazy danych stworzonej przez użytkownika.

W tym tutorialu będziemy wykorzystywali bazę danych **student**. Niezależnie od tego, czy baza ta znajduje się na uzyskanej liście, wykonaj poniższe polecenie.

```
use student
switched to db student
```

Łatwo domyślić się, że polecenie **use <database name>** służy do wskazania bazy danych, z którą chcemy pracować. Nie ma znaczenia to, czy taka baza taka rzeczywiście istnieje. Jeżeli na liście istniejących baz była baza **student**, wykonaj poniższe polecenie.

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

Odpowiedź wskazuje, że baza danych **student** została usunięta. W przypadku, gdy baza nie istniała, odpowiedź wyglądałaby następująco:

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

W *MongoDB* tworzenie bazy danych jest realizowane niejawnie. Wystarczy wskazać poleceniem **use** bazę danych, a następnie utworzyć w niej kolekcję. Samą kolekcję tworzymy poprzez wstawienie do niej pierwszego dokumentu.

1.3 Tworzenie kolekcji i wstawianie dokumentów

W tym ćwiczeniu utworzymy kolekcję przechowującą dane o pracownikach. Do realizacji tego celu użyjemy polecenia **insert(<dokument>)** wywoływanego dla kolekcji **pracownicy**. Wstaw do kolekcji pracownika **WEGLARZ**:

```
db.pracownicy.insert({ id_prac : 100, "nazwisko" : "WEGLARZ", "placa_pod" : 1730 })
WriteResult({ "nInserted" : 1 })
```

W odpowiedzi otrzymaliśmy informację o liczbie wstawionych dokumentów. Zauważ, że w przeciwieństwie do pozostałych kluczy, klucz **id_prac** jest zapisany bez użycia cudzysłowów. O ile w nazwie klucza nie pojawia się znak specjalny . (kropka), nie trzeba stosować cudzysłowów.

Upewnijmy się, że kolekcja pracownicy została utworzona. W tym celu wydaj polecenie **db.getCollectionNames()** lub **show collections**. Oba polecenia służą do uzyskania listy dostępnych kolekcji w bazie danych.

```
db.getCollectionNames()
[ "pracownicy", "system.indexes" ]
```

Sprawdźmy zawartość kolekcji **pracownicy** korzystając z polecenia **find()**, którego zadaniem jest wyszukanie wszystkich dokumentów:

```
db.pracownicy.find()
{ "_id" : ObjectId("5a21d40fa997edfb630bc998"), "id_prac" : 100, "nazwisko" : "WEGLARZ", "placa_pod" : 1730 }
```

Jak łatwo zauważyć do podanego przez nas dokumentu, opisującego pracownika **WEGLARZ**, automatycznie wygenerowany został dodatkowy klucz **_id** oraz wartość dla tego klucza.

Spróbuj ponownie wstawić pracownika **WEGLARZ** korzystając ponownie z tego samego polecenia **insert**:

```
db.pracownicy.insert({ id_prac : 100, "nazwisko" : "WEGLARZ", "placa_pod" : 1730 })
```

Wyświetl zawartość kolekcji **pracownicy**. Czy drugi dokument został wstawiony? Jeżeli tak, to czy różni się od poprzedniego?

Wykonaj teraz poniższe polecenie, które jawnie specyfikuje wartość klucza **_id**:

```
db.pracownicy.insert({ _id:100, id_prac : 100, "nazwisko" : "WEGLARZ", "placa_pod" : 1730 })
```

Spróbuj ponownie wykonać to samo polecenie. Co się stało?

Dodajmy teraz innego pracownika. Tym razem chcemy przechować również datę zatrudnienia. Wykonaj poniższe polecenie dodające pracownika **BLAZEWICZ**.

```
db.pracownicy.insert({ id_prac : 110, "nazwisko" : "BLAZEWICZ", "placa_pod" : 1350, zatrudniony: new Date("1973-05-01") })
```

```
WriteResult({ "nInserted" : 1 })
```

Zauważ, że nie musieliśmy zmieniać definicji kolekcji, aby dodać pole **zatrudniony**. To podstawowa cecha kolekcji i dokumentów. W większości przypadków kolekcje zawierają dokumenty o podobnej, ale niekoniecznie takiej samej, strukturze.

Zadanie 1.
Używając metody **find()** sprawdź, ile jest dokumentów w kolekcji **pracownicy** oraz w jaki sposób przedstawiona jest data zatrudnienia.

2 Dokumenty z referencjami

2.1 Utworzenie zbioru kolekcji

Przed dalszą pracą usuńmy wszystkie dokumenty z kolekcji **pracownicy**.

```
db.pracownicy.drop()
```

```
true
```

Wstawimy teraz większą liczbę dokumentów (w trybie wsadowym) posługując się poleceniem **insert**. Wykonaj poniższy kod wstawiający dane o 14 pracownikach do kolekcji **pracownicy**.

```
db.pracownicy.insert([
{id_prac:100, nazwisko:"WEGLARZ", etat:'DYREKTOR', zatrudniony: new Date("1968-01-01"), placa_pod:1730.00, placa_dod:420.50, id_zesp:10},
{id_prac:110, nazwisko:'BLAZEWICZ', etat:"PROFESOR",id_szefa:100, zatrudniony: new Date("1973-05-01"), placa_pod:1350.00, placa_dod:210.00, id_zesp:40},
{id_prac:120, nazwisko:'SLOWINSKI', etat:"PROFESOR",id_szefa:100, zatrudniony: new Date("1977-09-01"), placa_pod:1070.00, id_zesp:30},
{id_prac:130, nazwisko:'BRZEZINSKI', etat:"PROFESOR",id_szefa:100, zatrudniony: new Date("1968-07-01"), placa_pod:960.00, id_zesp:20},
{id_prac:140, nazwisko:'MORZY', etat:"PROFESOR",id_szefa:130, zatrudniony: new Date("1975-09-15"), placa_pod:830.00, placa_dod:105.00, id_zesp:20},
{id_prac:150, nazwisko:'KROLIKOWSKI', etat:'ADIUNKT',id_szefa:130, zatrudniony: new Date("1977-09-01"), placa_pod:645.50, id_zesp:20},
{id_prac:160, nazwisko:'KOSZLAJDA', etat:'ADIUNKT', id_szefa:130, zatrudniony: new Date("1985-03-01"), placa_pod:590.00, id_zesp:20},
{id_prac:170, nazwisko:'JEZIERSKI', etat:'ASYSTENT', id_szefa:130, zatrudniony: new Date("1992-10-01"), placa_pod:439.70, placa_dod:80.50, id_zesp:20},
```

```
{id_prac:190, nazwisko:'MATYSIAK', etat:'ASYSTENT', id_szefa:140, zatrudniony: new Date("1993-09-01"), placa_pod:371.00, id_zesp:20},
{id_prac:180, nazwisko:'MAREK', etat:'SEKRETARKA', id_szefa:100, zatrudniony: new Date("1985-02-20"), placa_pod:410.20, id_zesp:10},
{id_prac:200, nazwisko:'ZAKRZEWICZ', etat:'STAZYSTA', id_szefa:140, zatrudniony: new Date("1994-07-15"), placa_pod:208.00, id_zesp:30},
{id_prac:210, nazwisko:'BIALY', etat:'STAZYSTA', id_szefa:130, zatrudniony: new Date("1993-10-15"), placa_pod:250.00, placa_dod:170.60, id_zesp:30},
{id_prac:220, nazwisko:'KONOPKA', etat:'ASYSTENT', id_szefa:110, zatrudniony: new Date("1993-10-01"), placa_pod:480.00, id_zesp:20},
{id_prac:230, nazwisko:'HAPKE', etat:'ASYSTENT', id_szefa:120, zatrudniony: new Date("1992-09-01"), placa_pod:480.00, placa_dod:90.00, id_zesp:30}
])
```

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

Zadanie 2.
Do wykonania naszych zadań wykorzystamy również informacje o zespołach. Wykorzystaj polecenie **`insert`** do wstawienia (w trybie wsadowym) danych o zespołach (kolekcja **`zespoly`**):

```
{"id_zesp":10,"nazwa":"ADMINISTRACJA","adres":"PIOTROWO 3A"}
{"id_zesp":20,"nazwa":"SYSTEMY ROZPROSZONE","adres":"PIOTROWO 3A"}
{"id_zesp":30,"nazwa":"SYSTEMY EKSPERCKIE","adres":"STRZELECKA 14"}
{"id_zesp":40,"nazwa":"ALGORYTMY","adres":"WLODKOWICA 16"}
{"id_zesp":50,"nazwa":"BADANIA OPERACYJNE","adres":"MIELZYNSKIEGO 30"}
```

Jak zapewne zauważyłaś/eś, wstawione dokumenty współdzielą ten sam „schemat” i odpowiadają zawartości tabel w relacyjnym modelu danych. W wielu przypadkach takie przechowywanie danych może nie być zgodne z ideą przyświecającą bazom dokumentowym. Zajmiemy się tym problemem w dalszej części przewodnika. Na razie zobaczmy, w jaki sposób możemy pobrać, zmodyfikować i usuwać dane przechowywane w postaci takich prostych dokumentów.

2.2 Prosta selekcja

Wiemy już jak wybrać wszystkie dokumenty stosując bezparametrowe wywołanie metody **`find`**. Wersja parametryzowana pozwala na selekcję i projekcję danych. Najprostsze wyrażenie pozwalające na wyszukanie dokumentów opisujących profesorów wygląda następująco:

```
db.pracownicy.find({"etat":"PROFESOR"})
```

```
{ "_id" : ObjectId("5a31809c5e0e382af7b11db2"), "id_prac" : 110, "nazwisko" : "BLAZEWICZ", "etat" : "PROFESOR", "id_szefa" : 100, "zatrudniony" : ISODate("1973-05-01T00:00:00Z"), "placa_pod" : 1350, "placa_dod" : 210, "id_zesp" : 40 }
{ "_id" : ObjectId("5a31809c5e0e382af7b11db3"), "id_prac" : 120, "nazwisko" : "SLOWINSKI", "etat" : "PROFESOR", "id_szefa" : 100, "zatrudniony" : ISODate("1977-09-01T00:00:00Z"), "placa_pod" : 1070, "id_zesp" : 30 }
{ "_id" : ObjectId("5a31809c5e0e382af7b11db4"), "id_prac" : 130, "nazwisko" : "BRZEZINSKI", "etat" : "PROFESOR", "id_szefa" : 100, "zatrudniony" : ISODate("1968-07-01T00:00:00Z"), "placa_pod" : 960, "id_zesp" : 20 }
{ "_id" : ObjectId("5a31809c5e0e382af7b11db5"), "id_prac" : 140, "nazwisko" : "MORZY", "etat" : "PROFESOR", "id_szefa" : 130, "zatrudniony" : ISODate("1975-09-15T00:00:00Z"), "placa_pod" : 830, "placa_dod" : 105, "id_zesp" : 20 }
```

Łatwo zauważyć, że domyślnym operatorem jest porównanie. Wyszukaliśmy zatem dokumenty, których klucz **etat** ma wartość **PROFESOR**. Oczywiście można użyć innych operatorów selekcji. Najpierw jednak zapoznajmy się z projekcją.

2.3 Projekcja

Chociaż w poprzednim ćwiczeniu zostały wybrane odpowiednie dokumenty, wyświetlanie ich w całości może być mało czytelne. Można to poprawić wymuszając format czytelniejszy dla człowieka. Odpowiedzialna za to jest metoda **pretty()**.

```
db.pracownicy.find({"etat":"PROFESOR"}).pretty()
```

W praktyce często będziemy chcieli ograniczyć zakres wybieranych pól dokumentu. Służy do tego drugi parametr metody **find**. Załóżmy, że chcemy otrzymać tylko nazwiska profesorów.

```
db.pracownicy.find({"etat":"PROFESOR"}, {"nazwisko":1})
```

```
{ "_id" : ObjectId("5a31809c5e0e382af7b11db2"), "nazwisko" : "BLAZEWICZ" }
{ "_id" : ObjectId("5a31809c5e0e382af7b11db3"), "nazwisko" : "SLOWINSKI" }
{ "_id" : ObjectId("5a31809c5e0e382af7b11db4"), "nazwisko" : "BRZEZINSKI" }
{ "_id" : ObjectId("5a31809c5e0e382af7b11db5"), "nazwisko" : "MORZY" }
```

Wartość 1 przy specyfikacji nazwiska jest umowna (teoretycznie można podać inną wartość różną od 0), ale jest wymagana ponieważ poprawny format zawsze obejmuje nazwę atrybutu i jego wartość. Jeżeli chcesz wybrać więcej atrybutów, wymień je po przecinku.

```
db.pracownicy.find({"etat":"PROFESOR"}, {"nazwisko":1, "placa_pod":1})
```

```
{ "_id" : ObjectId("5a31809c5e0e382af7b11db2"), "nazwisko" : "BLAZEWICZ", "placa_pod" : 1350 }
{ "_id" : ObjectId("5a31809c5e0e382af7b11db3"), "nazwisko" : "SLOWINSKI", "placa_pod" : 1070 }
{ "_id" : ObjectId("5a31809c5e0e382af7b11db4"), "nazwisko" : "BRZEZINSKI", "placa_pod" : 960 }
{ "_id" : ObjectId("5a31809c5e0e382af7b11db5"), "nazwisko" : "MORZY", "placa_pod" : 830 }
```

Zadanie 3.
Zauważ, że dużą część odpowiedzi zajmują identyfikatory dokumentów (dodane automatycznie). Można uniknąć ich wybierania z bazy danych poprzez dodanie do listy pól pozycji `_id:0`. Przetestuj działanie poniższych zapytań:

```
db.pracownicy.find({"etat":"PROFESOR"}, {"nazwisko":1, "_id":0})
db.pracownicy.find({"etat":"PROFESOR"}, {"nazwisko":0, "_id":0})
```

Czym różnią się uzyskane wyniki?

2.4 Operatory $ne, $lt, $gt, $lte, $gte

Wróćmy teraz do problemu selekcji. Możemy ją przeprowadzić także z użyciem warunków innych niż porównanie. Poniższy przykład prezentuje wybranie pracowników, którzy nie są profesorami.

```
db.pracownicy.find({"etat":{$ne:"PROFESOR"}})
```

Operator **$ne** oznacza *not equal*, czyli *różny od*. W tabelce zestawiono informacje na temat popularnych operatorów, których możesz użyć w zapytaniach.

Operator	Opis
$ne	różny od
$lt	mniejszy niż
$gt	większy niż
$lte	mniejszy niż lub równy
$gte	większy niż lub równy

Wyszukanie nazwisk i płac podstawowych pracowników otrzymujących więcej niż 500 zł płacy podstawowej wygląda następująco:

```
db.pracownicy.find(
 {"placa_pod":{$gt:500}},
 {"nazwisko":1,"placa_pod":1}
)
```

2.5 Operatory $and i $or

Operator **$and** oznacza, że wszystkie warunki musza być spełnione by wybrać dokument. Formalnie specyfikowanie **$and** nie jest jednak konieczne. Wystarczy wymienić warunki po przecinku. Na przykład polecenie:

```
db.pracownicy.find(
 {"placa_pod":{$gt:1000}, "etat":"PROFESOR"},
 {"nazwisko":1,"placa_pod":1}
)
```

znajdzie profesorów zarabiających powyżej 1000 zł.

Jawne użycie operatora **$and** będzie wyglądało następująco:

```
db.pracownicy.find(
 {$and:[{"placa_pod":{$gt:1000}}, {"etat":"PROFESOR"}]},
 {"nazwisko":1,"placa_pod":1}
)
```

Uwaga - stosując niejawny zapis operatora **$and** należy pamiętać, że warunki muszą dotyczyć różnych pól. W przeciwnym wypadku tylko ostatni z warunków dla powtarzającego się pola zostanie uwzględniony (poprzednie zostaną „nadpisane” przez ostatni warunek).

Na przykład chcąc zdefiniować zakres „od-do”, należy to zapisać używając **$and**:

```
db.pracownicy.find(
 {$and:[{"placa_pod":{$gt:500}}, {"placa_pod":{$lt:700}}]},
 {"nazwisko":1,"placa_pod":1}
)
```

, a nie:

```
db.pracownicy.find(
 {"placa_pod":{$gt:500}, "placa_pod": {$lt:700}},
 {"nazwisko":1,"placa_pod":1}
)
```

Sprawdź jakich pracowników znajdzie powyższe polecenie.

Dodajmy jeszcze, że najkrótszą (ale nadal poprawną) formą zapisu będzie:

```
db.pracownicy.find(
 {"placa_pod":{$gt:500, $lt:700}},
 {"nazwisko":1,"placa_pod":1}
)
```

Jeżeli chcemy wybrać dokumenty spełniające jeden z warunków, możemy użyć operatora **$or**. Na przykład, aby wybrać dokumenty opisujące pracowników, którzy zarabiają więcej niż 1000 zł lub są profesorami, napiszemy:

```
db.pracownicy.find(
 {$or:[{"placa_pod":{$gt:1000}}, {"etat":"PROFESOR"}]},
 {"nazwisko":1, "etat":1, "placa_pod":1}
)
```

Zadanie 4.
Znajdź nazwiska, etaty i place podstawowe wszystkich asystentów oraz pracowników, którzy zajmują dowolne stanowisko, ale zarabiają pomiędzy 200 a 500 zł.

```
{ "nazwisko" : "JEZIERSKI", "etat" : "ASYSTENT", "placa_pod" : 439.7 }
{ "nazwisko" : "MATYSIAK", "etat" : "ASYSTENT", "placa_pod" : 371 }
{ "nazwisko" : "MAREK", "etat" : "SEKRETARKA", "placa_pod" : 410.2 }
{ "nazwisko" : "ZAKRZEWICZ", "etat" : "STAZYSTA", "placa_pod" : 208 }
{ "nazwisko" : "BIALY", "etat" : "STAZYSTA", "placa_pod" : 250 }
{ "nazwisko" : "KONOPKA", "etat" : "ASYSTENT", "placa_pod" : 480 }
{ "nazwisko" : "HAPKE", "etat" : "ASYSTENT", "placa_pod" : 480 }
```

2.6 Operator $in

Znalezienie pracowników zatrudnionych w zespole 10 lub 20 wygląda następująco:

```
db.pracownicy.find(
 {"id_zesp":{$in:[10,20]}},
 {"nazwisko":1,"id_zesp":1}
)
```

2.7 Sortowanie

Do sortowania dokumentów można wykorzystać metodę **sort**. Ponieważ dłuższe polecenia mogą być nieczytelne, warto je rozdzielić na kilka linii. Często stosuje się następujący format zapisu:

```
db.pracownicy.find(
 {"etat":"PROFESOR"},
 {"nazwisko":1, "placa_pod":1}
).sort(
 {"placa_pod":-1})
```

```
SELECT ETAT, NAZWISKO
FROM PRACOWICY
WHERE ETAT = "PROFESOR"
ORDER BY PLACA_POD DESC;
```

Zadanie 5.
Znajdź etaty, nazwiska i płace podstawowe pracowników zarabiających więcej niż 400zł. Otrzymane dokumenty powinny być posortowane wg etatu (zgodnie z porządkiem leksykograficznym). W przypadku powtarzających się etatów sortowanie powinno być zgodne z malejącą wartością płacy podstawowej.

```
{ "nazwisko" : "KROLIKOWSKI", "etat" : "ADIUNKT", "placa_pod" : 645.5 }
{ "nazwisko" : "KOSZLAJDA", "etat" : "ADIUNKT", "placa_pod" : 590 }
{ "nazwisko" : "KONOPKA", "etat" : "ASYSTENT", "placa_pod" : 480 }
{ "nazwisko" : "HAPKE", "etat" : "ASYSTENT", "placa_pod" : 480 }
{ "nazwisko" : "JEZIERSKI", "etat" : "ASYSTENT", "placa_pod" : 439.7 }
{ "nazwisko" : "WEGLARZ", "etat" : "DYREKTOR", "placa_pod" : 1730 }
{ "nazwisko" : "BLAZEWICZ", "etat" : "PROFESOR", "placa_pod" : 1350 }
{ "nazwisko" : "SLOWINSKI", "etat" : "PROFESOR", "placa_pod" : 1070 }
{ "nazwisko" : "BRZEZINSKI", "etat" : "PROFESOR", "placa_pod" : 960 }
{ "nazwisko" : "MORZY", "etat" : "PROFESOR", "placa_pod" : 830 }
{ "nazwisko" : "MAREK", "etat" : "SEKRETARKA", "placa_pod" : 410.2 }
```

2.8 Ograniczanie zbioru wyników

Do ograniczania zbioru wynikowego można użyć metod **`skip`** i **`limit`**. Metoda **`skip(n)`** pomija n dokumentów, a metod **`limit(m)`** wybiera maksymalnie m dokumentów. Poniższy przykład wybiera trzech najlepiej zarabiających pracowników.

```
db.pracownicy.find(
 {},
 {"nazwisko":1, "placa_pod":1}
).sort(
 {"placa_pod":-1}
).limit(3)
```

```
SELECT ETAT, NAZWISKO
FROM PRACOWICY
ORDER BY PLACA_POD DESC
FETCH FIRST 3 ROWS ONLY;
```

Zadanie 6.
Znajdź nazwisko i płacę podstawową pracownika zespołu nr 20, który jest na drugim miejscu pod względem zarobków (placa_pod) w swoim zespole.

```
{ "nazwisko" : "MORZY", "placa_pod" : 830 }
```

2.9 Wzorce

Selekcja dokumentów może odbywać się również na podstawie wyrażeń regularnych. Przykład wybrania wszystkich profesorów o nazwisku zawierającym ciąg znaków „SKI” wygląda następująco:

```
db.pracownicy.find
({"etat":"PROFESOR", "nazwisko":{$regex:"SKI"}},
 {"nazwisko":1}
)
```

```
SELECT ETAT, NAZWISKO
FROM PRACOWICY
WHERE ETAT = "PROFESOR" AND
NAZWISKO LIKE '%SKI%'
```

Zadanie 7.
Znajdź nazwiska i etaty pracowników zespołu 20 lub 30, którzy nie są asystentami, a ich nazwisko kończy się na literę I.

```
{ "nazwisko" : "SLOWINSKI", "etat" : "PROFESOR" }
{ "nazwisko" : "BRZEZINSKI", "etat" : "PROFESOR" }
{ "nazwisko" : "KROLIKOWSKI", "etat" : "ADIUNKT" }
```

2.10 Wartości puste

W przeciwieństwie do modelu relacyjnego, dokumenty w jednej kolekcji mogą się od siebie różnić strukturą. Na przykład mogą występować dodatkowe atrybuty. Możliwe jest również, że niektóre z atrybutów mają przypisaną specjalną wartość **`null`**. Poniższe polecenie znajdzie wszystkich pracowników, dla których nie zdefiniowano atrybutu **`placa_pod`**, lub gdy jego wartość to **`null`**.

```
db.pracownicy.find({"placa_dod":null})
```

Odpowiednikiem znanego z języka SQL operatora **`is not null`** jest warunek **`$ne:null`**:

```
db.pracownicy.find({"placa_dod":{$ne:null}})
```

2.11 Użycie języka JavaScript dla dopasowania dokumentów

Jeżeli nie możemy jakiegoś warunku wyrazić (lub w łatwy sposób wyrazić) używając składni *MongoDB*, przewidziano dla tego celu użycie języka *JavaScript*. Załóżmy, że chcemy wybrać pracowników, którzy dostają sumaryczną pensję (**`placa_pod`** + **`placa_dod`**) przekraczającą 1000 zł. W tym celu możemy wykorzystać operator **`$where`**, który umożliwia posłużenie się zapisem warunku w postaci kodu języka *JavaScript*:

```
db.pracownicy.find(
 {$where:
     "this.placa_pod + (this.placa_dod != null? this.placa_dod:0)  > 1000"},
 {"nazwisko":1, "placa_pod":1, "placa_dod":1}
)
```

```
SELECT NAZWISKO, PLACA_POD, PLACA_DOD
FROM PRACOWICY
WHERE PLACA_POD + NVL(PLACA_DOD,0) > 1000
```

W opisanym przykładzie uwzględniono możliwość braku zdefiniowanej wartości płacy podstawowej (i całkowitego braku tego atrybutu).

2.12 Mechanizm agregacji

Opisana wcześniej metoda projekcji nie pozwala np. na przetwarzanie odczytywanych wartości. Taki efekt można osiągnąć używając mechanizmu agregacji. Mimo swojej nazwy, nie oznacza on zawsze faktycznego agregowania wielu wartości do mniejszego zbioru. Przykład poniżej opisuje sposób wybierania pól oraz ich transformacji w zbiorze

wynikowym. Chcemy wyświetlić tylko etat (pod nazwą „stanowisko”) oraz dwunastokrotność płacy podstawowej (jako „pensja_roczna”) oraz miesiąc zatrudnienia każdego pracownika.

```
db.pracownicy.aggregate(
[{$project:{
       "_id":0,
       "stanowisko":"$etat",
       "pensja_roczna":{$multiply:["$placa_pod",12]},
       "miesiac_zatrudnienia": {$month:"$zatrudniony"}
  }}
])
```

```
SELECT ETAT AS STANIOWISKO
12*PLACA_POD AS PENSJA_ROCZNA
EXTRACT(MONTH FROM ZATRUDNIONY) AS
MIESIAC_ZATRUDNIENIA
FROM PRACOWNICY
```

Zauważ, że odniesienie do pól dokumentu poprzedzone jest znakiem **$**. Jeżeli chcielibyśmy dokonać selekcji dokumentów, możemy to uczynić operatorem **$match**. Na przykład ograniczmy się tylko do pracowników zespołu 20:

```
db.pracownicy.aggregate(
[{$match:{"id_zesp":20}},
 {$project:{
       "_id":0,
       "stanowisko":"$etat",
       "pensja_roczna":{$multiply:["$placa_pod",12]},
       "miesiac_zatrudnienia": {$month:"$zatrudniony"}
  }}
])
```

Mechanizm agregacji wywołuje kolejne operatory, które działają na wynikach swoich poprzedników.

Rzeczywistą agregację można uzyskać poprzez zastosowanie operatora **$group**. Pierwszym parametrem jest wyrażenie grupujące specyfikowane jako wartość pola **_id**. Jeżeli chcemy by wszystkie dokumenty znajdowały się w tej samej grupie, należy wartość tego pola ustawić na **null**. Pozostałe parametry definiują obliczane agregaty. Na przykład chcąc znaleźć sumę płac podstawowych oraz maksymalną wartość płacy podstawowej napiszemy:

```
db.pracownicy.aggregate(
  [{$group: {
       _id: null,
       wyplaty_razem: {$sum: "$placa_pod"},
       maksymalna: {$max: "$placa_pod"},
   }}]
)
```

```
SELECT SUM(PLACA_POD) AS
WYPLATY_RAZEM,
       MAX(PLACA_DOD) AS MAKSYMALNA
FROM PRACOWICY
```

W kolejnym przykładzie zastosujemy grupowanie zgodnie z numerem zespołu:

```
db.pracownicy.aggregate(
  [{$group: {
       _id: "$id_zesp",
       wyplaty_razem: {$sum: "$placa_pod"},
       maksymalna: {$max: "$placa_pod"},
   }}]
)
```

```
SELECT ID_ZESP AS _ID, SUM(PLACA_POD)
       AS WYPLATY_RAZEM,
       MAX(PLACA_DOD) AS MAKSYMALNA
FROM PRACOWICY
GROUP BY ID_ZESP
```

Mechanizm agregacji pozwala również odfiltrować niechciane grupy (odpowiednik klauzuli **HAVING** z języka SQL). W tym celu użyjemy operatora **$match**. Będzie on operował na wynikach grupowania.

```
db.pracownicy.aggregate(
  [{$group: {
       _id: "$id_zesp",
       wyplaty_razem: {$sum: "$placa_pod"},
       maksymalna: {$max: "$placa_pod"},
   }},
   {$match:{
      wyplaty_razem: {$lt:2100}
   }}]
)
```

```
SELECT ID_ZESP AS _ID, SUM(PLACA_POD)
       AS WYPLATY_RAZEM,
       MAX(PLACA_DOD) AS MAKSYMALNA
FROM PRACOWICY
GROUP BY ID_ZESP
HAVING SUM(PLACA_POD) < 2100;
```

Kolejny przykład pokazuje grupowanie oparte na wyliczonej wartości. Poniższe polecenie dla każdego roku, w którym został zatrudniony choć jeden pracownik oblicza liczbę zatrudnionych w tymże roku pracowników.

```
db.pracownicy.aggregate(
[
  {$group:{
       _id: {rok_zatrudnienia:{$year:"$zatrudniony"}},
       liczba_zatrudnionych: {$sum: 1}
    }}
]
)
```

```
SELECT COUNT(*) AS LICZBA_ZATRUDNIONYCH
FROM PRACOWICY
GROUP BY EXTRACT(YEAR FROM ZATRUDNIONY)
```

Więcej informacji o operatorach dla mechanizmu agregacji można znaleźć tutaj:

https://docs.mongodb.com/manual/reference/operator/aggregation/

Zadanie 8.

We frameworku agregacji, sortowanie oraz ograniczanie zbioru wynikowego przyjmuje postać dodatkowych operatorów **$sort**, **$skip**, **$limit**:

```
collection.aggregate(
[
  {$sort: { <field1>: <sort order>, <field2>: <sort order> ... } }
  {$skip : <numer>},
  {$limit: <numer>}
])
```

Wykorzystaj mechanizm agregacji do znalezienia nazwiska, etatu i roku zatrudnienia pracownika, który jest trzeci pod względem wysokości zarobków.

```
{ "stanowisko" : "PROFESOR", "nazwisko" : "SLOWINSKI", "rok_zatrudnienia" : 1977 }
```

Zadanie 9.

Znajdź dla każdego identyfikatora zespołu liczbę pracowników zatrudnionych w tym zespole. Wynik ogranicz do zespołów, które zatrudniają więcej niż 3 pracowników. Wskazówka: użyj wyrażenia **{$sum: 1}** do zliczania pracowników.

```
{ "_id" : 20, "liczba" : 7 }
{ "_id" : 30, "liczba" : 4 }
```

2.13 Połączenie danych z wielu kolekcji

Informacje o zespołach i pracownikach zatrudnionych w tych zespołach są przechowywane w osobnych kolekcjach. Mimo, że w ogólności może nie być to najlepszy sposób przechowywania danych dokumentowych, w nowszych wersjach *MongoDB* wspierane jest przetwarzanie takich zbiorów danych. Od wersji 3.2 możliwe jest używanie odwołań do innych kolekcji poprzez operator **$lookup**. Jest to operacja zbliżona do operacji połączenia w modelu relacyjnym.

Poniższe zapytanie realizuje zadanie powiązania każdego pracownika z odpowiednim zespołem.

```
db.pracownicy.aggregate(
[{$lookup:{from: "zespoly",
           localField: "id_zesp",
           foreignField: "id_zesp",
           as:"dept"}}
]
)
```

Dodanie projekcji wygląda następująco:

```
db.pracownicy.aggregate(
[{$lookup:{from: "zespoly",
           localField: "id_zesp",
           foreignField: "id_zesp",
           as: "dept"}},
 {$project:{"id_prac":1,"nazwisko":1, "dept":1}}
]
)
```

W powyższym przypadku otrzymaliśmy identyfikator i nazwisko pracownika oraz wszystkie dane o jego zespole. Zauważ, że dane o zespole są reprezentowane jako element tablicy. Jeżeli chcemy ograniczyć się tylko do pozyskania nazwy zespołu można użyć następującej konstrukcji:

```
db.pracownicy.aggregate(
[{$lookup:{from: "zespoly",
           localField: "id_zesp",
           foreignField: "id_zesp",
           as: "dept"}},
    {$project: {"id_prac":1,"nazwisko":1, "dept": {$arrayElemAt:["$dept.nazwa",0]}}}
]
)
```

Operator **$arrayElemAt** dla wskazanej tablicy elementów wybiera element znajdujący się pod wskazanym indeksem. Ponieważ każdy pracownik jest zatrudniony w jednym zespole, należy wybrać pierwszy element (indeks 0). Wyrażenie **$dept.nazwa** wybiera wartość klucza **nazwa** dla powiązanego z pracownikiem zespołu.

Zadanie 10.

Dla każdego pracownika zespołu nr 20 lub 30, uzyskaj nazwisko tego pracownika oraz adres jego zespołu.

```
{ "nazwisko" : "SLOWINSKI", "dept" : "STRZELECKA 14" }
{ "nazwisko" : "BRZEZINSKI", "dept" : "PIOTROWO 3A" }
{ "nazwisko" : "MORZY", "dept" : "PIOTROWO 3A" }
{ "nazwisko" : "KROLIKOWSKI", "dept" : "PIOTROWO 3A" }
{ "nazwisko" : "KOSZLAJDA", "dept" : "PIOTROWO 3A" }
{ "nazwisko" : "JEZIERSKI", "dept" : "PIOTROWO 3A" }
{ "nazwisko" : "MATYSIAK", "dept" : "PIOTROWO 3A" }
{ "nazwisko" : "ZAKRZEWICZ", "dept" : "STRZELECKA 14" }
{ "nazwisko" : "BIALY", "dept" : "STRZELECKA 14" }
{ "nazwisko" : "KONOPKA", "dept" : "PIOTROWO 3A" }
{ "nazwisko" : "HAPKE", "dept" : "STRZELECKA 14" }
```

Zadanie 11.

Znajdź pracowników, którzy pracują przy ul. Strzeleckiej. Oprócz nazwiska podaj również nazwę zespołu, do którego pracownik należy. Uwaga! Nie używaj identyfikatora zespołu do przeprowadzenia selekcji. Nie używaj również numeru budynku.

2.14 Modyfikacja danych

Zmodyfikujmy płacę dodatkową pracownika o nazwisku **BLAZEWICZ** ustawiając ją na wartość 100 zł. Uproszczona wersja metody **update** ma schemat:

```
db.collection.update(
  <query>,
  <update>,
)
```

W naszym przypadku przyjmie ona postać:

```
db.pracownicy.update(
  {nazwisko:"BLAZEWICZ"},
  {$set:{placa_dod:100}}
)
```

Sprawdź zawartość kolekcji **pracownicy**.

Zwiększmy jeszcze pensję podstawową wszystkim pracownikom o 500:

```
db.pracownicy.update(
  {},
  {$inc:{placa_pod:500}}
)
```

Sprawdź, czy rzeczywiście pensje podstawowe się zwiększyły. Co zauważyłaś(eś)?

Domyślnie polecenie update wykonywane jest dla pierwszego dopasowanego dokumentu. Jeżeli chcemy zmodyfikować wszystkie dopasowane dokumenty powinniśmy dodatkowo dopisać jako ostatni parametr polecenia parę **multi:true**.

```
db.pracownicy.update(
  {},
  {$inc:{placa_pod:500}},
  {"multi": true}
)
```

W omówionych przypadkach operacja update polega na aktualizacji odpowiednich pól. Możliwe jest także wykorzystanie metody update w wersji, która zastępuje cały dokument. Szersze informacje na ten temat możesz znaleźć tutaj: https://docs.mongodb.com/v3.4/reference/method/db.collection.update.

Zadanie 12.
Zauważ, że wartości identyfikatora zespołu (**id_zesp**) w dokumentach kolekcji **pracownicy** nie są powiązane z identyfikatorami dokumentów kolekcji **zespoly**.
Do wykonania połączenia danych z dwóch kolekcji możemy wykorzystać następujący kod w języku *JavaScript*:

```
var pracownicy = db.pracownicy.find();
while (pracownicy.hasNext()) {
  prac = pracownicy.next();
  zesp = db.zespoly.findOne({"id_zesp": prac.id_zesp});
  print(prac.nazwisko + ":" + zesp.nazwa);
}
```

Pierwsza linia programu to utworzenie kursora, który jest następnie odczytywany w ramach pętli **while**. Każda iteracja pętli to realizacja kolejnego zapytania, które wyszukuje w tym przypadku zespół pracownika.
Wykorzystując powyższy kod i wiedzę na temat metody **update** napisz skrypt, który wartość klucza **id_zesp** w dokumentach kolekcji **pracownicy** ustawi na wartość odpowiadającą identyfikatorowi (**_id**) dokumentu odpowiedniego zespołu.

3 Dokumenty zagnieżdżone/osadzone (embedded)

Stwórzmy teraz kolekcję dotyczącą produktów. Każdy produkt oprócz podstawowych informacji będzie zawierał także wystawione dla niego oceny. Oceny te będą przechowywane w postaci zagnieżdżonego dokumentu. Dodatkowo będziemy przechowywać listę tagów przypisanych do produktu.

```
db.produkty.insert(
[{
    nazwa: "Kosiarka spalinowa",
    cena: 1000,
    cechy: {
      zbiornik_paliwa_pojemnosc: 0.8,
      waga: 23
    },
    tagi: ["maszyna", "ogrod", "dom", "kosiarka"],
    oceny:
    [ {osoba: "Jurek", ocena: 3},
      {osoba: "Ania", ocena: 4},
      {osoba: "Basia", ocena: 3.6}
    ]
 },
 {
```

```
    nazwa: "Wiertarka udarowa",
    cena: 1200,
    cechy: {
      moc_udaru: 4,
      maksymalne_obroty: 4000,
      uchwyt: "SDS"
    },
    tagi: ["wiertarka"],
    oceny:
    [ {osoba: "Michał", ocena: 5},
      {osoba: "Roman", ocena: 4.8},
    ]
  },
  {
    nazwa: "Wiertarko - wkrętarka",
    cena: 450,
    cechy: {
      pojemnosc_akumulatora: 1.3,
      czas_ladowania: 60
    },
     tagi: ["wiertarka", "dom"],
     oceny:
     [ {osoba: "Ania", ocena: 5},
       {osoba: "Robert", ocena: 4},
       {osoba: "Janusz", ocena: 4},
       {osoba: "Julita", ocena: 3}
     ]
  },
  {
    nazwa: "Kosiarka elektryczna",
    cena: 900,
    cechy: {
      moc: 1700,
      waga: 17
    },
    tagi: ["kosiarka", "ogrod", "dom"],
    oceny:
    [ {osoba: "Monika", ocena: 3},
      {osoba: "Karol", ocena: 4}
    ]
  }
])
```

Przyglądnij się uważnie dokumentom produktów. W przeciwieństwie do kolekcji pracowników, dokumenty przechowywane w kolekcji **produkty** różnią się od siebie strukturą i są znacznie bardziej złożone.

W dokumentach produktów tagi reprezentowane są w postaci tablicy łańcuchów znaków. Zawarto również osadzony dokument opisujący pewne cechy produktu. Pola tego osadzonego dokumentu mogą być różne dla różnych produktów. Dodatkowo z każdym produktem związana jest lista ocen w postaci tablicy osadzonych dokumentów. Każda ocena ma autora.

3.1 Przeglądanie z uwzględnieniem tablic i dokumentów osadzonych

Zapoznajmy się teraz z metodami dostępu do tych złożonych pól dokumentów. Do wyszukania nazw wszystkich wiertarek użyjemy następującego polecenia:

```
db.produkty.find(
  {"tagi":"wiertarka"},
  {_id:0, nazwa:1}
)
```

Jak łatwo zauważyć wystarczyło wskazać konieczność występowania słowa „wiertarka" na liście tagów. W odpowiedzi powinniśmy otrzymać dwa produkty. Możemy zawęzić ten zbiór do wiertarek, ale przeznaczonych dla zastosowań domowych. W tym celu wydaj polecenie:

```
db.produkty.find(
  {"tagi": {"$all":["wiertarka","dom"]}},
  {_id:0, nazwa:1}
)
```

Użyliśmy operatora **$all** by sprawdzić występowanie wszystkich elementów. Przydatne operatory zostały opisane w tabelce poniżej.

Operator	Opis
$in	dowolny argument podany jako wartość znajduje się w zbiorze
$all	wszystkie argumenty podane jako wartość znajdują się w zbiorze
$nin	nie znajduje się w zbiorze (not in)

Dostęp do dokumentów zagnieżdżonych można uzyskać stosując operator **.** (kropka). Na przykład wyszukanie produktów ważących 23 kg wygląda następująco:

```
db.produkty.find(
  {"cechy.waga": 23},
  {_id:0, nazwa:1}
)
```

Zadanie 13.
Znajdź nazwy, dla których oceny nie wystawiła Ania lub Karol.
`{"nazwa" : "Wiertarka udarowa"}`

Do konstruowania warunków selekcji dokumentów opartych na rozmiarach tablic można wykorzystać operator **$size**. Poniższe polecenie pokazuje w jaki sposób wybrać produkty, dla których istnieją 2 opinie.

```
db.produkty.find(
  {"oceny": {$size:2}}
)
```

Zbadajmy jeszcze, jakie są średnie oceny dla składowanych w bazie danych produktów. Ponieważ jest to operacja agregacji, skorzystamy z mechanizmu agregacji *MongoDB*. Odpowiednie polecenie znajdziesz poniżej. Operator **$unwind** powoduje rozbicie tablicy **oceny** na pojedyncze elementy. Każdy taki element jest dołączany do pozostałych informacji o produkcie i tworzy nowy dokument. W rezultacie, z jednego dokumentu produktu, powstaje tyle dokumentów produktu, ile jest dla niego ocen. Każdy para uzyskanych dokumentów różni się zawartością pola oceny. Można to porównać do denormalizacji. Ponieważ chcemy uzyskać średnią ocenę, grupujemy dokumenty dotyczące tego samego produktu i obliczamy średnią używając operatora **$avg**.

```
db.produkty.aggregate(
  [
  { $unwind : "$oceny" },
  {$group: {
       _id:"$_id",
       srednia_ocena: {$avg: "$oceny.ocena"}
   }}]
)
```

Zadanie 14.
Znajdź najlepiej oceniony produkt (z najwyższą średnią ocen). Wybierz nazwę produktu (użyj aliasu „produkt") oraz jego średnią ocenę.
`{ "produkt" : "Wiertarka udarowa", "srednia_ocena" : 4.9 }`

3.2 Wstawianie danych

Wiertarko – wkrętarka posiada obecnie dwa tagi: „wiertarka" i „dom". Brakuje tagu „wkrętarka". Dodajmy go. Do tego celu możemy użyć operatora **$push**, który powoduje dodanie elementu do listy.

```
db.produkty.update(
  {"nazwa":"Wiertarko - wkrętarka"},
  {$push: {tagi: "wkretarka"}}
)
```

Zadanie 15.
Dodaj nową ocenę kosiarki spalinowej. Niech tą oceną będzie 4 i niech będzie wystawiona przez Jacka.

3.3 Usuwanie danych

Usuwanie danych możliwe jest poprzez wykorzystanie metody **remove**. W swojej podstawowej wersji przyjmuje ona jeden parametr – opis warunków, jakie musi spełnić usuwany dokument. Na przykład usunięcie kosiarki spalinowej można przeprowadzić w następujący sposób:

```
db.produkty.remove(
  {"nazwa":"Kosiarka spalinowa"}
)
```

Do usuwanie elementów tablic można użyć polecenia update z operatorem **$pull**. Poniższe polecenie usunie ocenę wystawioną przez „Romana" dla produktu „Wiertarka udarowa".

```
db.produkty.update(
  {"nazwa":"Wiertarka udarowa"},
  { $pull: { "oceny": { "osoba" : "Roman" } } }
)
```

Zadanie 16.
Usuń oceny nie większe niż 4 dla wszystkich produktów.

3.4 Usuwanie kolekcji bazy danych

Usunięcie całej kolekcji pracownicy można wykonać poleceniem:

```
db.produkty.drop()
```

Pracę zakończymy usuwając stworzoną przez nas bazę danych. W tym celu wydaj następujące polecenie:

```
db.dropDatabase()
```