

Adam Biśta

Najpierw zimportujemy bazę Yelp na serwer lokalny. W tym celu po rozpakowaniu folderu yelp.zip wykonałem następujące komendy w cmd:

```
mongoimport --db yelp --collection business --file "C:\\<pełna ścieżka do folderu>\\yelp\\business.json"

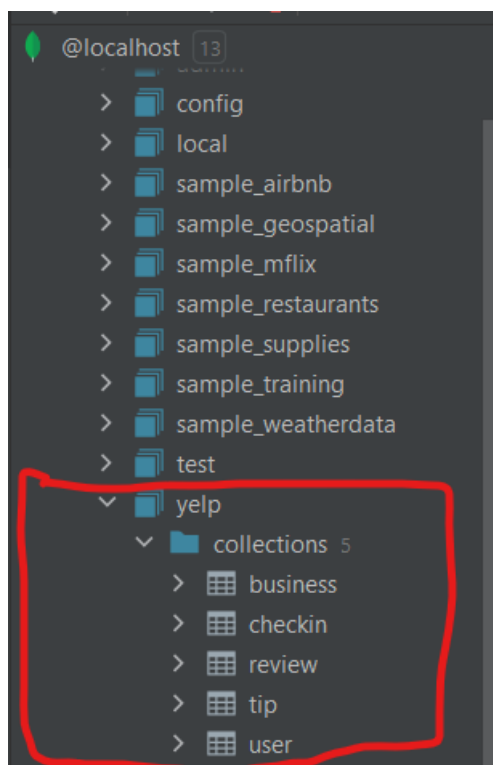
mongoimport --db yelp --collection checkin --file "C:\\<pełna ścieżka do folderu >\\yelp\\checkin.json"

mongoimport --db yelp --collection review --file "C:\\<pełna ścieżka do folderu >\\yelp\\review.json"

mongoimport --db yelp --collection tip --file "C:\\<pełna ścieżka do folderu >\\yelp\\tip.json"

mongoimport --db yelp --collection user --file "C:\\<pełna ścieżka do folderu >\\yelp\\user.json"
```

Widok bazy w datagripie:



Wpisujemy polecenia do konsoli:

```
use yelp;  
show collections;
```

Wynik:

	{} badge	{} name
1		business
2		checkin
3		review
4		tip
5		user

ZADANIA

1. Operacje wyszukiwania danych

a. Zapytanie:

```
use yelp;  
db.business.find(  
  "categories": { $in: ["Restaurants"] },  
  "hours.Monday": { $exists: true, $ne: "" },  
  "stars": { $gte: 4 }  
, {_id: 0,  
  name: 1,  
  address: 1,  
  categories: 1,  
  hours: 1,  
  stars: 1}  
) .sort(  
  name: 1  
)
```

wyniki:

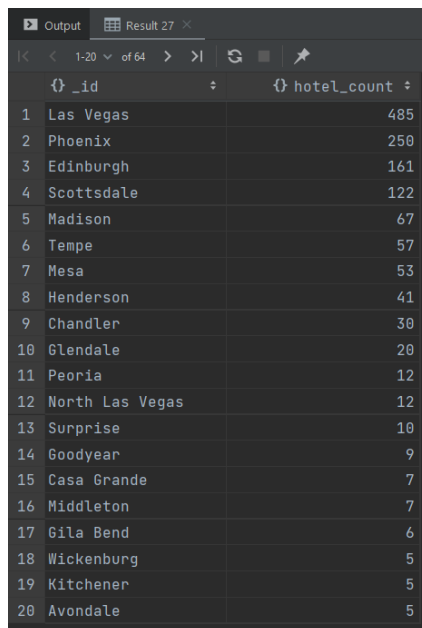
	{} categories	{} hours	{} name
1	["Food", "Desserts", "Coffee & Tea", "Indian", "Rest"]	{"Monday": {"close": "22:00", "open": "10:00"}, "Tue 10-to-10 In Delhi	
2	["Vietnamese", "Asian Fusion", "French", "Restaurant"]	{"Monday": {"close": "22:00", "open": "11:00"}, "Fri 188 Restaurant	
3	["Bars", "Asian Fusion", "Nightlife", "Lounges", "Ka"]	{"Monday": {"close": "05:00", "open": "18:00"}, "Tue 21 Restaurant & Lounge	
4	["Food", "Live/Raw Food", "Juice Bars & Smoothies"]	{"Monday": {"close": "14:00", "open": "08:00"}, "Tue 24 Carrots Juice Bar & Cafe	
5	["Breakfast & Brunch", "Gluten-Free", "Vegetarian"]	{"Monday": {"close": "14:00", "open": "08:00"}, "Tue 24 Carrots Juice Bar & Cafe	
6	["Asian Fusion", "Restaurants"]	{"Monday": {"close": "22:00", "open": "11:00"}, "Tue 2860 The Restaurant	
7	["Coffee & Tea", "Food", "Breakfast & Brunch", "Bars"]	{"Monday": {"close": "22:00", "open": "06:00"}, "Tue 32 Shea	
8	["Bakeries", "Food", "Breakfast & Brunch", "Coffee &"]	{"Monday": {"close": "14:30", "open": "07:00"}, "Tue 4620 Bakery & Cafe	
9	["Cafes", "American (Traditional)", "Diners", "Resta"]	{"Monday": {"close": "14:00", "open": "05:00"}, "Tue 48th Street Cafe	
10	["American (New)", "Restaurants"]	{"Monday": {"close": "14:00", "open": "11:30"}, "Tue 43 North	
11	["Vietnamese", "Restaurants"]	{"Monday": {"close": "20:00", "open": "09:00"}, "Fri 43rd Express	
12	["Bars", "Restaurants", "American (Traditional)", "S"]	{"Monday": {"close": "00:00", "open": "11:00"}, "Tue 4th Floor Grille & Sports Bar	
13	["Food", "Ethnic Food", "Thai", "Specialty Food", "R"]	{"Monday": {"close": "21:00", "open": "11:00"}, "Tue 5 R Cha Thai Bistro	
14	["Food", "Ethnic Food", "Thai", "Specialty Food", "R"]	{"Monday": {"close": "21:00", "open": "11:00"}, "Tue 5 R Cha Thai Diner	
15	["Caterers", "Event Planning & Services", "Barbeque"]	{"Monday": {"close": "20:00", "open": "11:00"}, "Tue 5 Star BBQ & Grill	
16	["Breakfast & Brunch", "Comfort Food", "Diners", "Re"]	{"Monday": {"close": "21:00", "open": "07:00"}, "Tue 58s Diner	
17	["Bars", "Nightlife", "British", "Restaurants"]	{"Monday": {"close": "01:00", "open": "11:00"}, "Tue 52 Canoes Tiki Den	
18	["Breakfast & Brunch", "American (Traditional)", "Re"]	{"Monday": {"close": "14:00", "open": "07:00"}, "Tue 5th Avenue Cafe	
19	["Wine Bars", "Bars", "American (New)", "Nightlife"]	{"Monday": {"close": "21:00", "open": "11:00"}, "Tue 5th and Wine	
20	["Chicken Wings", "Restaurants"]	{"Monday": {"close": "20:00", "open": "11:00"}, "Tue 702 Wing Spot	

Pasujących danych jest o wiele więcej, wyświetliłem tylko 20 pasujących.

b. Zapytanie:

```
use yelp;
db.business.aggregate([
  {
    $match: {
      $or: [
        { "categories": { $in: ["Hotels & Travel", "Hotels"] } }
      ]
    }
  },
  {
    $group: {
      _id: "$city",
      hotel_count: { $sum: 1 }
    }
  },
  {
    $sort: { hotel_count: -1 }
  }
])
```

Wynik:



	_id	hotel_count
1	Las Vegas	485
2	Phoenix	250
3	Edinburgh	161
4	Scottsdale	122
5	Madison	67
6	Tempe	57
7	Mesa	53
8	Henderson	41
9	Chandler	30
10	Glendale	20
11	Peoria	12
12	North Las Vegas	12
13	Surprise	10
14	Goodyear	9
15	Casa Grande	7
16	Middleton	7
17	Gila Bend	6
18	Wickenburg	5
19	Kitchener	5
20	Avondale	5

Jak widać najwięcej hoteli podanych kategorii znajduje się w Las Vegas, niemal 2 razy więcej niż w Phoenix

c. Zapytanie:

```
use yelp;
db.tip.aggregate([
  {
    $match: {
      date: { $regex: /2012/ }
    }
  },
  {
    $lookup: {
      from: "business",
      localField: "business_id",
      foreignField: "business_id",
      as: "business_doc"
    }
  },
  {
    $unwind: "$business_doc"
  },
  {
    $limit: 100
  },
  {
    $group: {
      _id: "$business_doc.name",
      no_tips: { $sum: 1 }
    }
  },
  {
    $sort: { no_tips: -1 }
  },
  {
    $project: {
      _id: 0,
      name: "$_id",
      no_tips: 1
    }
  }
])
```

Uwaga! W powyższym przykładzie tuż po połączeniu tip oraz business zrobiłem ograniczenie do pokazania 100 pierwszych wyników. Bez tego ograniczenia całe zapytanie trwałoby bardzo długo.

Wynik:

	name	no_tips
1	Monty's Blue Plate Diner	17
2	Dexter's Pub	11
3	Club Tavern & Grille	5
4	People's Bakery	4
5	Louisianne's Etc	4
6	Green Lantern Restaurant	3
7	Imperial Garden Chinese Restaurant	3
8	Mousehouse Cheesehaus	3
9	Cool Beans Coffee House and Cafe	2
10	International House of Pancakes	2
11	China Wok Buffet	2
12	LOFT	2
13	Bristled Boar Saloon & Grill	2
14	The Home Depot	2
15	New Orleans Take-Out	2
16	Barriques Wine & Spirits	2
17	Target	2
18	Marriott-Madison West	2
19	Culver's	2
20	Capital Brewery & Beer Garden	2

d. Zapytanie:

```
db.review.aggregate([
  {
    $project: {
      _id: 0,
      cool: { $cond: [{ $gte: ["$votes.cool", 1] }, 1, 0] },
      funny: { $cond: [{ $gte: ["$votes.funny", 1] }, 1, 0] },
      useful: { $cond: [{ $gte: ["$votes.useful", 1] }, 1, 0] }
    }
  },
  {
    $group: {
      _id: null,
      totalCool: { $sum: "$cool" },
      totalFunny: { $sum: "$funny" },
      totalUseful: { $sum: "$useful" }
    }
  },
  {
    $project: {
      _id: 0,
      totalCool: "$totalCool",
      totalFunny: "$totalFunny",
      totalUseful: "$totalUseful"
    }
  }
])
```

Wynik:

	totalCool	totalFunny	totalUseful
1	346519	269256	549519

W powyższym zapytaniu użyłem **\$cond**, w którym zdefiniowałem warunek oznaczania recenzji. Jeśli liczba głosów danej kategorii przy wybranej recenzji jest większa lub równa 1 to powinien zwrócić 1, gdyż dana kategoria posiada jakiegokolwiek głosy. W przeciwnym wypadku 0. Jest to potrzebne do zliczania kategorii wziętych pod uwagę w **\$group**.

e. Zapytanie:

```
db.user.find({
  $or: [
    { "votes.funny": { $lt: 1 } },
    { "votes.useful": { $lt: 1 } }
  ]
}).sort({ name: 1 })
```

Wynik:

	elite	fans	friends	name	review_count	type	user_id
1	0	0	0	Bernard	1	user	xP3SPgfgW2vc5
2	0	0	0	Anastacia	10	user	qJLc0rYytqzeVl
3	0	0	0	Brandon	2	user	CfWe8CqVvHze8l
4	0	0	0	David	5	user	M5IQ6r6omqgmF
5	0	0	0	Jenelle	6	user	LI5yKFJdqKSTF
6	0	0	0	Persian	4	user	bSoPY5UR-fxSKn
7	Int("1")	1	1	Viridiana	4	user	wP-nekfpfZE2p
8	0	0	0	Maria	1	user	0s5f3TNpM7_A8
9	0	0	0	rick	10	user	28s2J8yWuR6u6l
10	0	0	0	006969123	3	user	FtTid0SF_puL0l
11	0	0	0	A	2	user	MjLcGNLsSUzZel
12	0	0	0	A	2	user	aHuT-eiSBLORL
13	0	0	0	A	1	user	kanljLBdaadQ_l
14	0	0	0	A	8	user	UuV8IZ9BPdLmq
15	0	0	0	A	1	user	ArZiH7p8LiMnh
16	0	1	1	A	9	user	SUmjgKXWh3UM
17	0	0	0	A	2	user	4UjsyyEuJKB6j
18	0	0	0	A	4	user	70yC-GN-rH2y6
19	Int("1")	0	0	A	2	user	7biUX-jsGryiI
20	0	0	0	A	2	user	grJY8BBpMa5_1

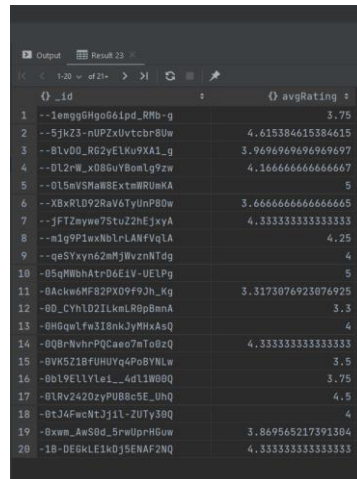
Wyniki są posortowane względem pola name alfabetycznie. Nazwy A znajdują się pod innymi imionami, gdyż przed nimi znajdują się dodatkowe znaki, które powodują wcześniejsze wyświetlanie.

f. Zapytanie:

i. Przypadek:

```
db.review.aggregate([
  { $group: { _id: "$business_id", avgRating: { $avg:
"$stars" } } },
  { $match: { avgRating: { $gt: 3 } } },
  { $sort: { _id: 1 } }
])
```

Wyniki:



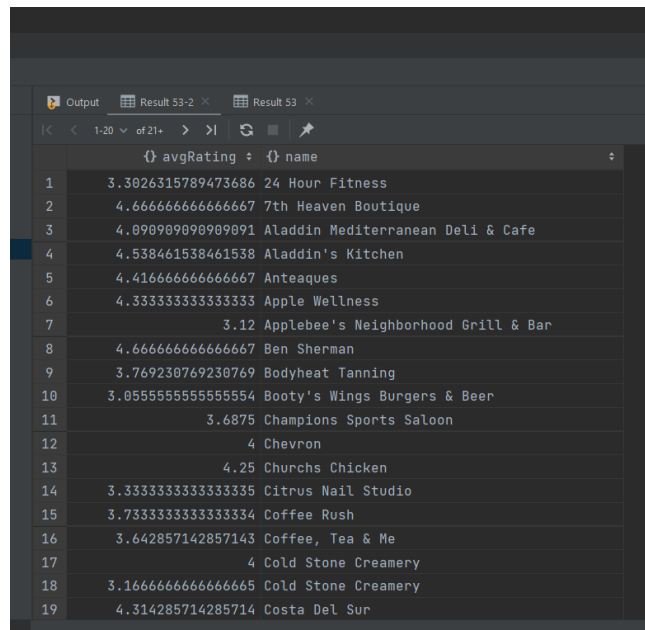
_id	avgRating
--1eaggDHge661pd_RMb+g	3.75
--5jkZ3-nUPZxUvtcbr8Uw	4.615384615384615
--BLvD0_R62yELku9XA1_g	3.9696969696969697
--DL2rW_x08GuYBomlg9zw	4.166666666666667
--DL5mVSMa8ExtmWRUmKA	5
--XBxRLD92RaV6TyUnP8Dw	3.6666666666666665
--JFTZmywe7Stu2ZhEjxyA	4.333333333333333
--m1g9P1exMbLrLANFVqLA	4.25
--qeSYxyn62mMjWvznNTdg	4
--B5qMBbAtrD6E1V-UeLPg	5
--6A6k6dMF92PX09F9Jh_Kg	3.3173076923076925
--B0_CVhLD2ILkmLR0pBmNA	3.3
--0HGqLfw3I8nkJyMHxAsQ	4
--0QB8rWvhrPQCaeo7mToBzQ	4.333333333333333
--0VK5Z1BFUHUyq4PoBYNLw	3.5
--0bL9ELLYle1_dL1W0BQ	3.75
--0LRvZ42DzyPU8Bc5E_UHQ	4.5
--0tJ4FweNtJj1L-ZUTy3BQ	4
--0xwm_AwS0d_5rwUpRHGuw	3.869565217391304
--1B-DEGkLE1k0J5ENAF2NQ	4.333333333333333

ii. Przypadek:

```
use yelp;
db.review.aggregate([
  { $group: { _id: "$business_id", avgRating: { $avg:
"$stars" } } },
  { $match: { avgRating: { $gt: 3 } } },
  { $lookup: { from: "business", localField: "_id",
foreignField: "business_id", as: "business_doc" } },
  { $unwind: "$business_doc" },
  { $limit: 100 },
  { $sort: { "business_doc.name": 1 } },
  { $project: { _id: 0, name: "$business_doc.name",
avgRating: 1 } }
])
```

Jak widać użyłem ograniczenia do wyświetlenia 100 danych od razu po połączeniu review z business. Zrobiłem to dlatego, gdyż liczba danych jest ogromna i czas oczekiwania na wyniki był bardzo długi.

Wyniki:



The screenshot shows a MongoDB query result in a web interface. The query is `{ avgRating : { $gt: 4.0 } }`. The result is a list of 19 documents, each containing an `avgRating` and a `name`. The documents are sorted by `avgRating` in descending order. The interface includes tabs for 'Output', 'Result 53-2', and 'Result 53'. The 'Output' tab is active, showing the results in a table format with a search bar and navigation controls.

	avgRating	name
1	3.3026315789473686	24 Hour Fitness
2	4.666666666666667	7th Heaven Boutique
3	4.090909090909091	Aladdin Mediterranean Deli & Cafe
4	4.538461538461538	Aladdin's Kitchen
5	4.416666666666667	Anteaques
6	4.333333333333333	Apple Wellness
7	3.12	Applebee's Neighborhood Grill & Bar
8	4.666666666666667	Ben Sherman
9	3.769230769230769	Bodyheat Tanning
10	3.0555555555555554	Booty's Wings Burgers & Beer
11	3.6875	Champions Sports Saloon
12	4	Chevron
13	4.25	Churchs Chicken
14	3.3333333333333335	Citrus Nail Studio
15	3.7333333333333334	Coffee Rush
16	3.642857142857143	Coffee, Tea & Me
17	4	Cold Stone Creamery
18	3.1666666666666665	Cold Stone Creamery
19	4.314285714285714	Costa Del Sur

2. Tworzenie bazy

a. Podejście znormalizowane

i. Wady i zalety oraz struktura kolekcji:

W tym podejściu będziemy mieli cztery osobne kolekcje: **'Professors'**, **'Subjects'**, **'Students'** i **'Grades'**. Każdy dokument będzie zawierał id referencyjne do innych dokumentów, co pozwoli na tworzenie złożonych relacji.

1. Zalety:

- Separacja danych: każda kategoria danych jest przechowywana oddzielnie – zwiększa czytelność bazy
- Redukcja redundancji – zmniejszone ryzyko niezgodności danych

2. Wady:

- Konieczność łączenia danych – zapytania mogą być bardziej skomplikowane
- Zmniejszona wydajność – większa ilość zapytań do bazy

Struktura bazy danych:

Professors:

```
{
  "_id": ObjectId(),
  "name": "Jan Kowalski",
  "email": "jan.kowalski@example.com",
  "subjects": [ObjectId(), ObjectId(), ...] // Referencje do
dokumentów z kolekcji Subjects
}
```

Subjects:

```
{
  "_id": ObjectId(),
  "name": "Fizyka",
  "professor": ObjectId() // Referencja do dokumentu z
kolekcji Professors
}
```

Students:

```
{
  "_id": ObjectId(),
  "name": "Anna Nowak",
  "email": "anna.nowak@example.com",
  "subjects": [ObjectId(), ObjectId(), ...] // Referencje do
dokumentów z kolekcji Subjects
}
```

Grades:

```
{
  "_id": ObjectId(),
  "student": ObjectId(), // Referencja do dokumentu z
kolekcji Students
  "subject": ObjectId(), // Referencja do dokumentu z
kolekcji Subjects
  "grade": 5,
  "date": ISODate("2023-05-20")
}
```

ii. Wprowadzanie przykładowych danych:

```
//Tworzenie bazy danych
use university;
```

```
//Tworzenie kolekcji
db.createCollection('Professors')
db.createCollection('Subjects')
db.createCollection('Students')
db.createCollection('Grades')
```

```
//wstawianie danych do kolekcji Professors:
```

```
db.Professors.insertMany([
  {
    "_id": ObjectId(),
    "name": "Jan Kowalski",
    "email": "jan.kowalski@example.com",
    "subjects": []
  },
  {
    "_id": ObjectId(),
    "name": "Maria Nowak",
    "email": "maria.nowak@example.com",
    "subjects": []
  },
]);
```

```
//wstawianie danych do kolekcji Subjects:
```

```
db.Subjects.insertMany([
  {
    "_id": ObjectId(),
    "name": "Fizyka",
    "professor": db.Professors.findOne({name: "Jan
Kowalski"})._id
  },
  {
    "_id": ObjectId(),
    "name": "Matematyka",
    "professor": db.Professors.findOne({name: "Jan
Kowalski"})._id
  },
]);
```

```
//Teraz zaktualizujemy dane dla kolekcji Professors dodając
odpowiednie identyfikatory do przedmiotów:
```

```
db.Professors.updateOne(
  { name: "Jan Kowalski" },
  { $push: { subjects: db.Subjects.findOne({name:
"Fizyka"})._id } }
);
db.Professors.updateOne(
  { name: "Jan Kowalski" },
  { $push: { subjects: db.Subjects.findOne({name:
"Matematyka"})._id } }
);
```

```
//Wstawianie danych do kolekcji Students:
```

```
db.Students.insertMany([
  {
    "_id": ObjectId(),
    "name": "Anna Kowalczyk",
    "email": "anna.kowalczyk@example.com",
```

```

        "subjects": [
            db.Subjects.findOne({name: "Fizyka"})._id,
            db.Subjects.findOne({name: "Matematyka"})._id
        ]
    },
    {
        "_id": ObjectId(),
        "name": "Piotr Nowakowski",
        "email": "piotr.nowakowski@example.com",
        "subjects": [
            db.Subjects.findOne({name: "Fizyka"})._id,
            db.Subjects.findOne({name: "Matematyka"})._id
        ]
    }
]);

```

```

//Wstawianie danych do kolekcji Grades:

db.Grades.insertMany([
    {
        "_id": ObjectId(),
        "student": db.Students.findOne({name: "Anna
Kowalczyk"})._id,
        "subject": db.Subjects.findOne({name: "Fizyka"})._id,
        "grade": 5,
        "date": new Date()
    },
    {
        "_id": ObjectId(),
        "student": db.Students.findOne({name: "Piotr
Nowakowski"})._id,
        "subject": db.Subjects.findOne({name:
"Matematyka"})._id,
        "grade": 4,
        "date": new Date()
    }
]);

```

iii. Przykłady

Założmy, że chcemy wyświetlić imię i nazwisko studenta, jego przedmioty i oceny z tych przedmiotów. Nasze całe zapytanie będzie wyglądać tak:

```

use university;
let student_id = db.Students.findOne({name:"Anna Kowalczyk"})._id
db.Grades.aggregate([
    { $match: { "student": student_id } },
    { $lookup: {
        from: "Subjects",
        localField: "subject",
        foreignField: "_id",
        as: "subjectInfo"
    }
    }
]);

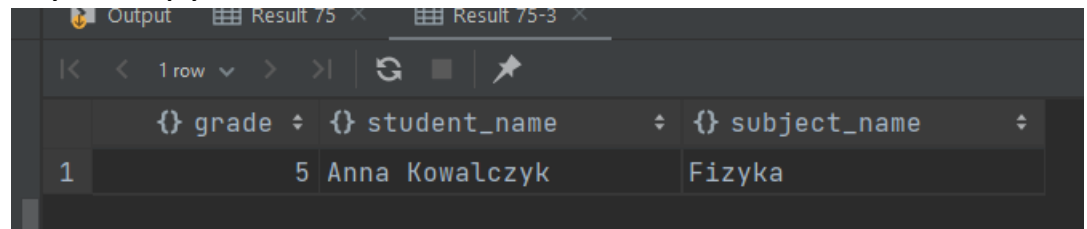
```

```

    } },
    {$unwind:"$subjectInfo"},
    { $lookup: {
      from: "Students",
      localField: "student",
      foreignField: "_id",
      as: "studentInfo"
    } },
    {$unwind:"$studentInfo"},
    {
      $project: {
        _id:0,
        student_name: "$studentInfo.name",
        subject_name:"$subjectInfo.name",
        grade:"$grade"
      }
    }
  ]
}

```

Wynik zapytania:



	grade	student_name	subject_name
1	5	Anna Kowalczyk	Fizyka

Aby uzyskać wyświetlenie podanych trzech informacji musieliśmy wykonać złożone zapytanie - połączyliśmy ze sobą aż 3 różne kolekcje. Dodatkowo wydajność jest zmniejszona, gdyż trzeba było wykonać skomplikowane zapytanie (**podpunkty a i b do wad**)

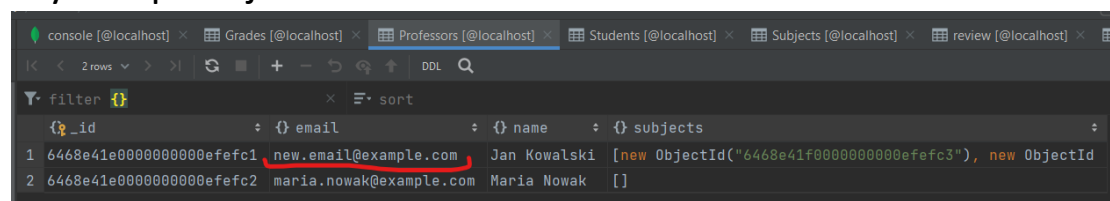
Jeśli chcielibyśmy zaktualizować adres e-mail profesora, to wystarczy zrobić to tylko w jednym miejscu (**separacja danych**):

```

db.Professors.updateOne(
  { "name": "Jan Kowalski" },
  { $set: { "email": "new.email@example.com" } }
);

```

Wynik operacji:



	_id	email	name	subjects
1	6468e41e0000000000efefc1	new.email@example.com	Jan Kowalski	[new ObjectId("6468e41f0000000000efefc3"), new ObjectId("6468e41e0000000000efefc2")]
2	6468e41e0000000000efefc2	maria.nowak@example.com	Maria Nowak	[]

Jak widać, wystarczyło w jednym miejscu dokonać zmiany, aby rezultat był widoczny w każdym miejscu.

Jeśli wykładowca wystawi ocenę z jakiegoś przedmiotu, to nie musimy przechowywać tych samych informacji w dokumentach studenta, wykładowcy i przedmiotu. Informacje te są przechowywane tylko w dokumencie z ocenami (**zmniejszenie redundancji**)

```
db.Grades.insertOne({
  "student": db.Students.findOne({name: "Anna Kowalczyk"})._id,
  "subject": db.Subjects.findOne({name: "Matematyka"})._id,
  "grade": 4,
  "date": new Date()
});
```

Wykonajmy wcześniejsze zapytanie:

```
let student_id = db.Students.findOne({name: "Anna Kowalczyk"})._id
db.Grades.aggregate([
  { $match: { "student": student_id } },
  { $lookup: {
    from: "Subjects",
    localField: "subject",
    foreignField: "_id",
    as: "subjectInfo"
  } },
  { $unwind: "$subjectInfo" },
  { $lookup: {
    from: "Students",
    localField: "student",
    foreignField: "_id",
    as: "studentInfo"
  } },
  { $unwind: "$studentInfo" },
  {
    $project: {
      _id: 0,
      student_name: "$studentInfo.name",
      subject_name: "$subjectInfo.name",
      grade: "$grade"
    }
  }
])
```

Wynik:

	{ } grade ÷ { } student_name	÷ { } subject_name ÷
1	5 Anna Kowalczyk	Fizyka
2	4 Anna Kowalczyk	Matematyka

Wszystkie pliki odnośnie tego podejścia znajdują się w katalogu „podejście znormalizowane”, baza danych znajduje się w folderze **dump**

Na koniec usuwamy bazę:

```
use university;  
db.dropDatabase();
```

c. Podejście odnormalizowane (embedded documents)

i. Wady i zalety oraz tworzenie kolekcji:

W podejściu odnormalizowanym (embedded documents) dokumenty są zagnieżdżone wewnątrz innych dokumentów. Kolekcje mogą wyglądać tak:

- **Professors** - zawierająca listę przedmiotów prowadzonych przez danego profesora
- **Students** - zawierająca listę przedmiotów, które student obecnie uczęszcza, wraz z ocenami otrzymanymi od wykładowców

1. Zalety

- Mniejsza ilość zapytań do bazy - zapytania są prostsze i szybsze, ponieważ większość informacji można uzyskać z pojedynczego dokumentu
- Wyższa wydajność - zmniejsza się ilość operacji, ponieważ dane są często przechowywane w taki sposób, że do najczęściej używanych informacji można uzyskać dostęp za pomocą pojedynczego zapytania

2. Wady

- Zwiększona redundancja danych - te same informacje mogą być przechowywane w wielu miejscach
- Może być trudno zarządzać aktualizacjami - jeśli te same dane są przechowywane w wielu miejscach, trzeba je zaktualizować we wszystkich miejscach, co może być trudne i prowadzić do niezgodności

Struktura bazy danych:

Professors:

```
{
  "_id" : ObjectId(),
  "name" : "Jan Kowalski",
  "email" : "jan.kowalski@example.com",
  "subjects" : [
    {
      "_id" : ObjectId(),
      "name" : "Fizyka",
      "studentsGrades" : [
        {
          "studentId" : ObjectId(),
          "grade" : 5
        },
        // more grades
      ]
    },
    // more subjects
  ]
}
```

Students:

```
{
  "_id" : ObjectId(),
  "name" : "Anna Kowalczyk",
  "email" : "anna.kowalczyk@example.com",
  "subjects" : [
    {
      "_id" : ObjectId(),
      "name" : "Fizyka",
      "professorId" : ObjectId(),
      "grade" : 5
    },
    // more subjects
  ]
}
```

ii. Wprowadzanie przykładowych danych

```
//tworzymy bazę
use university;
```

```
//dodajemy kolekcje
db.createCollection("Professors")
db.createCollection("Students")
```

```
//dodawanie profesorów
db.Professors.insertMany([
  { "name": "Jan Kowalski", "email": "jan.kowalski@example.com" },
  { "name": "Piotr Nowak", "email": "piotr.nowak@example.com" }
]);
```

```
//dodawanie studentów
db.Students.insertMany([
  { "name": "Anna Nowak", "email": "anna.nowak@example.com" },
  { "name": "Piotr Kowalski", "email": "piotr.kowalski@example.com" },
  { "name": "Jakub Nowak", "email": "jakub.nowak@example.com" },
  { "name": "Ewa Kowalczyk", "email": "ewa.kowalczyk@example.com" }
]);
```

//Teraz, gdy mamy ID dla każdego profesora i studenta,
//możemy dodać informacje o przedmiotach i ocenach. Pobierzmy najpierw ID dla profesorów i studentów:

```
let janKowalskiId = db.Professors.findOne({ name: "Jan Kowalski" })._id;
let piotrNowakId = db.Professors.findOne({ name: "Piotr Nowak" })._id;
let annaNowakId = db.Students.findOne({ name: "Anna Nowak" })._id;
let piotrKowalskiId = db.Students.findOne({ name: "Piotr Kowalski" })._id;
let jakubNowakId = db.Students.findOne({ name: "Jakub Nowak" })._id;
let ewaKowalczykId = db.Students.findOne({ name: "Ewa Kowalczyk" })._id;
```

```
//dodajemy przedmioty dla profesorów i dla studentów
db.Professors.updateOne(
  { _id: janKowalskiId },
  {
    $set: {
      "subjects": [
        {
          "name": "Fizyka",
          "studentsGrades": [
            { "studentId": annaNowakId, "grade": 5 },
            { "studentId": piotrKowalskiId, "grade": 4.5 }
          ]
        },
        {
          "name": "Matematyka",
          "studentsGrades": [
            { "studentId": ewaKowalczykId, "grade": 4.5 },
            { "studentId": jakubNowakId, "grade": 5 }
          ]
        }
      ]
    }
  }
);

db.Professors.updateOne(
  { _id: piotrNowakId },
  {
    $set: {
      "subjects": [
```



```

        {
            "name": "Biologia",
            "studentsGrades": [
                { "studentId": annaNowakId, "grade": 4 },
                { "studentId": ewaKowalczykId, "grade": 3.5 }
            ]
        },
        {
            "name": "Chemia",
            "studentsGrades": [
                { "studentId": piotrKowalskiId, "grade": 4 },
                { "studentId": jakubNowakId, "grade": 5 }
            ]
        }
    ]
}
);

db.Students.updateOne(
    { _id: annaNowakId },
    {
        $set: {
            "subjects": [
                {
                    "name": "Fizyka",
                    "professorId": janKowalskiId,
                    "grade": 5
                },
                {
                    "name": "Biologia",
                    "professorId": piotrNowakId,
                    "grade": 4
                }
            ]
        }
    }
);

```

```

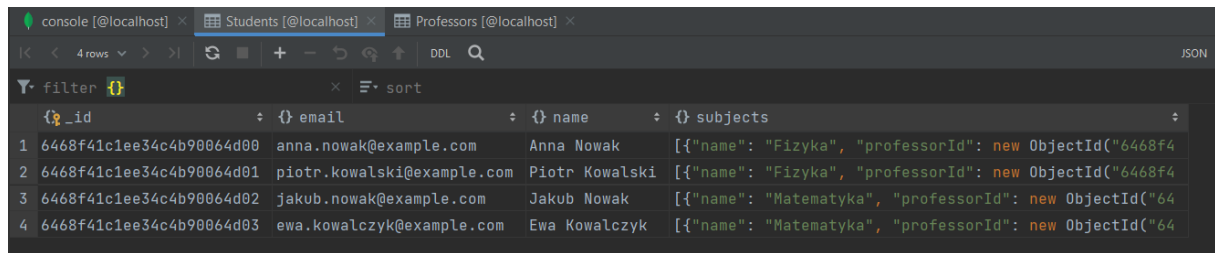
// Podobnie aktualizujemy resztę studentów
db.Students.updateOne(
    { _id: piotrKowalskiId },
    {
        $set: {
            "subjects": [
                {
                    "name": "Fizyka",
                    "professorId": janKowalskiId,
                    "grade": 4.5
                },
                {
                    "name": "Chemia",
                    "professorId": piotrNowakId,
                    "grade": 4
                }
            ]
        }
    }
);

```

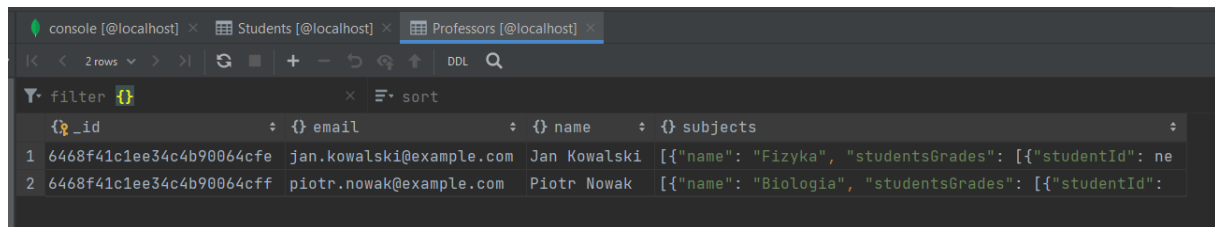
```
db.Students.updateOne(
  { _id: jakubNowakId },
  {
    $set: {
      "subjects": [
        {
          "name": "Matematyka",
          "professorId": janKowalskiId,
          "grade": 5
        },
        {
          "name": "Chemia",
          "professorId": piotrNowakId,
          "grade": 5
        }
      ]
    }
  }
);

db.Students.updateOne(
  { _id: ewaKowalczykId },
  {
    $set: {
      "subjects": [
        {
          "name": "Matematyka",
          "professorId": janKowalskiId,
          "grade": 4.5
        },
        {
          "name": "Biologia",
          "professorId": piotrNowakId,
          "grade": 3.5
        }
      ]
    }
  }
);
```

Wyniki w tabelach:



{_id}	{email}	{name}	{subjects}
1 6468f41c1ee34c4b90064d00	anna.nowak@example.com	Anna Nowak	[{"name": "Fizyka", "professorId": new ObjectId("6468f41c1ee34c4b90064d01")}]
2 6468f41c1ee34c4b90064d01	piotr.kowalski@example.com	Piotr Kowalski	[{"name": "Fizyka", "professorId": new ObjectId("6468f41c1ee34c4b90064d02")}]
3 6468f41c1ee34c4b90064d02	jakub.nowak@example.com	Jakub Nowak	[{"name": "Matematyka", "professorId": new ObjectId("6468f41c1ee34c4b90064d03")}]
4 6468f41c1ee34c4b90064d03	ewa.kowalczyk@example.com	Ewa Kowalczyk	[{"name": "Matematyka", "professorId": new ObjectId("6468f41c1ee34c4b90064d04")}]



{_id}	{email}	{name}	{subjects}
1 6468f41c1ee34c4b90064cfe	jan.kowalski@example.com	Jan Kowalski	[{"name": "Fizyka", "studentsGrades": [{"studentId": new ObjectId("6468f41c1ee34c4b90064d00"), "grade": 5}]}]
2 6468f41c1ee34c4b90064cff	piotr.nowak@example.com	Piotr Nowak	[{"name": "Biologia", "studentsGrades": [{"studentId": new ObjectId("6468f41c1ee34c4b90064d01"), "grade": 4}]}]

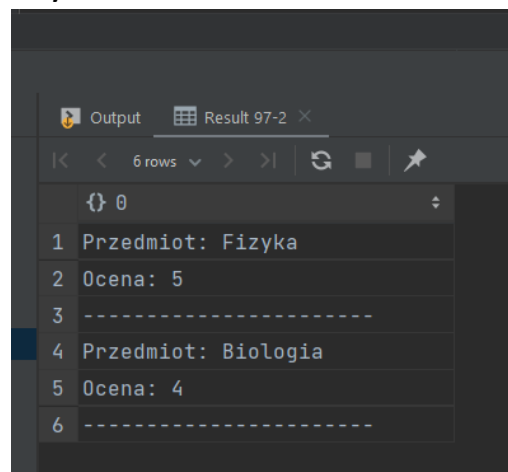
iii. Przykłady

Jeżeli chcielibyśmy sprawdzić oceny studenta, wszystko jest zawarte w jednym dokumencie, a nasze zapytanie jest bardzo proste: **(Zalety a i b)**

```
use university;

var student = db.Students.findOne({ name: "Anna Nowak" });
for (var i = 0; i < student.subjects.length; i++) {
    var subject = student.subjects[i];
    print("Przedmiot: " + subject.name);
    print("Ocena: " + subject.grade);
    print("-----");
}
```

Wynik:



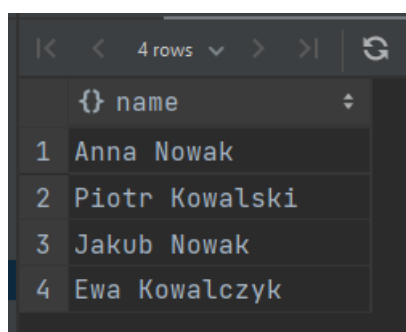
{}
1 Przedmiot: Fizyka
2 Ocena: 5
3 -----
4 Przedmiot: Biologia
5 Ocena: 4
6 -----

(Zmniejszona ilość operacji) Chcielibyśmy znaleźć wszystkich studentów, którzy są uczeni przez konkretnego profesora. Dzięki temu, że mamy ID profesora zapisane bezpośrednio przy przedmiotach, zapytanie jest bardzo proste i szybkie:

```
use university;

let professorId = db.Professors.findOne({ "name": "Jan Kowalski" })._id;
db.Students.find({ "subjects.professorId": professorId }, {name:1, _id:0});
```

Wynik:



	name
1	Anna Nowak
2	Piotr Kowalski
3	Jakub Nowak
4	Ewa Kowalczyk

(Wady a i b) Jeśli na przykład zmieni się ocena, musimy zaktualizować ją zarówno w dokumencie profesora, jak i studenta:

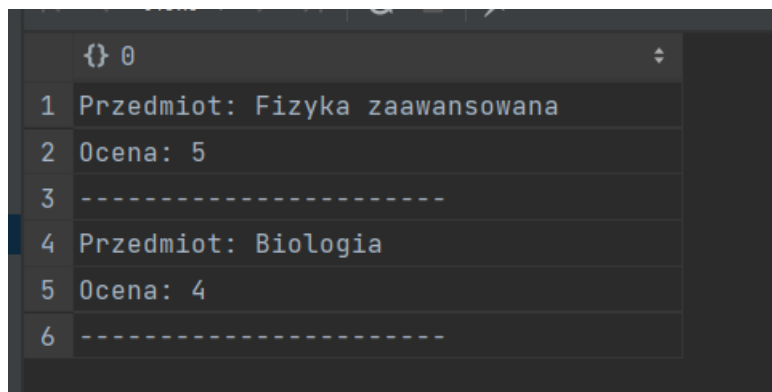
```
db.Professors.updateMany(
  { "subjects.name": "Fizyka" },
  { $set: { "subjects.$.name": "Fizyka zaawansowana" } }
);
db.Students.updateMany(
  { "subjects.name": "Fizyka" },
  { $set: { "subjects.$.name": "Fizyka zaawansowana" } }
);
```

W pewnych sytuacjach może to doprowadzić wielu błędów, jeśli się nie zna dokładnej struktury tabel.

Teraz wykonując poprzednie zapytanie:

```
var student = db.Students.findOne({ name: "Anna Nowak" });
for (var i = 0; i < student.subjects.length; i++) {
  var subject = student.subjects[i];
  print("Przedmiot: " + subject.name);
  print("Ocena: " + subject.grade);
  print("-----");
}
```

W wyniku otrzymamy:



A screenshot of a terminal window with a dark background. It displays a list of subjects and their corresponding grades, separated by dashed lines. The list is numbered 1 through 6.

1	Przedmiot: Fizyka zaawansowana
2	Ocena: 5
3	-----
4	Przedmiot: Biologia
5	Ocena: 4
6	-----

Wszystkie pliki odnośnie tego podejścia znajdują się w katalogu „**podejście odnormalizowane**”, baza danych znajduje się w folderze **dump**

Na koniec usuwamy bazę:

```
use university;  
db.dropDatabase()
```