Dokumentowe bazy danych – MongoDB

**Ćwiczenie 2 - zadanie do samodzielnego wykonania**

**Imię i nazwisko:**

**Materiały:**

Książki

Np.

* Shannon Bradshaw, Eoin Brazil, Kristina Chodorow,MongoDB: The Definitive Guide. Powerful and Scalable Data Storage, O'Reily 2019
* Alex Giamas, Mastering MongoDB 4.x., Pact 2019

Dokumentacja

* <https://www.mongodb.com/docs/manual/reference/program/mongo/>

MongoDB University Courses

* <https://university.mongodb.com/courses/catalog>
* MongoDB Basics
  + <https://university.mongodb.com/courses/M001/about>
* The MongoDB Aggregation Framework
  + <https://university.mongodb.com/courses/M121/about>
* Data Modeling
  + <https://university.mongodb.com/courses/M320/about>

**Yelp Dataset**

[www.yelp.com](http://www.yelp.com) - serwis społecznościowy – informacje o miejscach/lokalach

* restauracje, kluby, hotele itd. (*businesses*),
* użytkownicy piszą recenzje (*reviews*) o miejscach i wystawiają oceny oceny,
* użytkownicy odwiedzają te miejsca - "meldują się" (*check-in*)
* Przykładowy zbiór danych zawiera dane z 5 miast: Phoenix, Las Vegas, Madison, Waterloo i Edinburgh.

Kolekcje:

Obraz zawierający tekst

Opis wygenerowany automatycznie

Obraz zawierający tekst

Opis wygenerowany automatycznie

### Obraz zawierający tekst Opis wygenerowany automatycznie

### Obraz zawierający tekst Opis wygenerowany automatycznie

### Obraz zawierający tekst Opis wygenerowany automatycznie

### Obraz zawierający tekst Opis wygenerowany automatycznie

### Zadania

1. **Operacje wyszukiwania danych**

Dla zbioru Yelp wykonaj następujące zapytania

W niektórych przypadkach może być potrzebne wykorzystanie mechanizmu Aggregation Pipeline

<https://www.mongodb.com/docs/manual/core/aggregation-pipeline/>

1. Zwróć dane wszystkich restauracji (kolekcja businesss, pole *categories* musi zawierać wartość *Restaurants),* które są otwarte w poniedziałki(pole hours) i mają ocenę co najmniej 4 gwiazdki (pole *stars*). Zapytanie powinno zwracać: nazwę firmy, adres, kategorię, godziny otwarcia i gwiazdki. Posortuj wynik wg nazwy firmy.

Standardowe zapytanie spełniające kryteria podpunktu. Metoda find() składa się z 3 parametrów: zapytania, widoku zapytania oraz dodatkowych opcji, gdzie możemy np. posortować wyniki. Poniżej fragment otrzymanych dokumentów.

db.getCollection('business').find(

    {

        'categories': { $elemMatch: { $eq: 'Restaurants' } },

        'hours.Monday': { $exists: true },

        'stars': { $gte: 4 }

    },

    {

        'name': true,

        'full\_address': true,

        'categories': true,

        'hours': true,

        'stars': true

    },

    {

        sort: [ 'name' ]

    }

);

A picture containing text, screenshot, font

Description automatically generated

1. Ile hoteli znajduje się w każdym mieście. (pole *categories* musi zawierać wartość *Hotels & Travel* lub *Hotels)*. Wynik powinien zawierać nazwę miasta, oraz liczbę hoteli. Posortuj wynik malejąco wg liczby hoteli.

W tym zapytaniu korzystam z metody aggregate(), która przyjmuje tablicę obiektów odpowiadających za agregację zapytania. $match to filtr kategorii hoteli, natomiast $group, jak sama nazwa wskazuje, to agregator grupy – grupuje każde przedsiębiorstwo po mieście i zlicza liczbę takich hoteli w obrębie każdej grupy, a na koniec sortuje malejąco po tej liczbie malejąco, korzystając z $sort.

db.getCollection('business').aggregate(

    [

        { $match: { 'categories': {

            $in: [ 'Hotels & Travel', 'Hotels' ]

        } } },

        { $group: {

            '\_id': '$city',

            'hotel\_count': { $count: {} }

        } },

        { $sort: { 'hotel\_count': -1 } }

    ]

);

A screenshot of a computer program

Description automatically generated with medium confidence

1. Ile każda firma otrzymała ocen/wskazówek (kolekcja *tip* ) w 2012. Wynik powinien zawierać nazwę firmy oraz liczbę ocen/wskazówek Wynik posortuj według liczby wskazówek (*tip*).

Filtruję za pomocą wyrażenia regularnego wszystkie dokumenty, które mają w dacie 2012 rok (posiadają na początku tekstu 2012). Grupuję i zliczam wszystkie przefiltrowane wskazówki, dołączam kolekcję business do zwróconych wyników, aby wydobyć nazwę firmy (odpowiednik JOIN z relacyjnych baz danych), następnie sortuję malejąco po liczbie sugestii, a na końcu zmieniam widok, żeby zawierał tylko pola: id, nazwę firmy oraz liczbę posiadanych wskazówek. W tym podpunkcie można zauważyć, że kolejność agregacji ma olbrzymie znaczenie na wydajność zapytania, w tym przypadku złe ułożenie przyczyniało się do wydłużenia czasu obliczania zapytania do ponad 15 minut, np. $sort po $project z $lookup.

db.getCollection('tip').aggregate([

    { $match: { 'date': { $regex: /^2012/ } } },

    { $group: {

        '\_id': '$business\_id',

        'tip\_count': { $count: {} }

    } },

    {

        $lookup: {

            from: 'business',

            localField: '\_id',

            foreignField: 'business\_id',

            as: 'business'

        }

    },

    { $sort: { 'tip\_count': -1 } },

    {

        $project: {

            'business\_name': { $first: '$business.name' },

            'tip\_count': 1

        }

    }

]);

A screenshot of a computer program

Description automatically generated with medium confidence

1. Recenzje mogą być oceniane przez innych użytkowników jako *cool, funny* lub *useful* (kolekcja r*eview*, pole *votes*, jedna recenzja może mieć kilka głosów w każdej kategorii). Napisz zapytanie, które zwraca dla każdej z tych kategorii, ile sumarycznie recenzji zostało oznaczonych przez te kategorie (np. recenzja ma kategorię *funny* jeśli co najmniej jedna osoba zagłosowała w ten sposób na daną recenzję)

Zamieniam obiekt votes na tablicę klucz-wartość, następnie rozpakowywuję tablicę za pomocą $unwind, pozbywam się wszystkich dokumentów z liczbą ocen z danej kategorii równą 0. count to liczba wszystkich niezerowych ocen, natomiast total sumuje liczbę wszystkich ocen, tj. jeśli w jednym dokumencie v będzie miało wartość 2, a w drugim 3 (k będzie równe) to count zwróci 2, a total 5.

db.getCollection('review').aggregate([

    {

        $project: {

            'votes': { $objectToArray: '$votes' }

        }

    },

    { $unwind: '$votes' },

    { $match: { 'votes.v': { $gt: 0 } } },

    {

        $group: {

            '\_id': '$votes.k',

            'count': { $count: {} },

            'total': { $sum: '$votes.v' }

        }

    }

]);

A picture containing text, screenshot, font

Description automatically generated

1. Zwróć dane wszystkich użytkowników (kolekcja *user*), którzy nie mają ani jednego pozytywnego głosu (pole *votes*) z kategorii (*funny lub useful*), wynik posortuj alfabetycznie według nazwy użytkownika.

db.getCollection('user').aggregate([

    {

        $match: {

            'votes.funny': 0,

            'votes.useful': 0,

            'type': 'user'

        }

    },

    { $sort: { 'name': 1 } }

]);

A screenshot of a computer program

Description automatically generated with medium confidence

1. Wyznacz, jaką średnia ocenę uzyskała każda firma na podstawie wszystkich recenzji (kolekcja *review*, pole *stars*). Ogranicz do firm, które uzyskały średnią powyżej 3 gwiazdek.

przypadek 1: Wynik powinien zawierać id firmy oraz średnią ocenę. Posortuj wynik wg id firmy.

Na początku grupuję po polu business\_id, agregując dodatkowo średnią opinii, wyświetlam tylko te opinie, które posiadają średnią większą niż 3, a na koniec sortuję wyniki po id firmy (\_id).

db.getCollection('review').aggregate([

    {

        $group: {

            '\_id': '$business\_id',

            'stars\_mean': { $avg: '$stars' }

        }

    },

    { $match: { 'stars\_mean': { $gt: 3 } } },

    { $sort: { '\_id': 1 } }

]);

A screen shot of a computer

Description automatically generated with medium confidence

przypadek 2: Wynik powinien zawierać nazwę firmy oraz średnią ocenę. Posortuj wynik wg nazwy firmy.

Do zapytania z przypadku 1 dodałem relację z kolekcją business, wyłuskując tylko nazwę firmy za pomocą pipeline. Czas oczekiwania na zapytanie jest bardzo długi z powodu natury zapytania i struktury bazy Yelp, mianowicie sortowanie i pozyskiwanie danych spoza bazowej kolekcji za pomocą $lookup zdecydowanie zmniejsza wydajność.

db.getCollection('review').aggregate([

    {

        $group: {

            '\_id': '$business\_id',

            'stars\_mean': { $avg: '$stars' }

        }

    },

    { $match: { 'stars\_mean': { $gt: 3 } } },

    {

        $lookup: {

            from: 'business',

            localField: '\_id',

            foreignField: 'business\_id',

            pipeline: [{ $group: { '\_id': '$name' } }],

            as: 'business'

        }

    },

    {

        $project: {

            '\_id': 0,

            'business\_name': { $first: '$business.\_id' },

            'stars\_mean': 1

        }

    },

    { $sort: { 'business\_name': 1 } }

]);

A picture containing text, screenshot, software

Description automatically generated

W sprawozdaniu należy umieścić zrzuty ekranów (z kodem poleceń oraz z uzyskanymi wynikami). Dodatkowo należy dołączyć plik tekstowy (najlepiej z rozszerzeniem .js) zawierający kod poleceń

1. **Modelowanie danych**
   * Zaproponuj strukturę bazy danych dla wybranego/przykładowego zagadnienia/problemu
   * Należy wybrać jedno zagadnienie/problem (A lub B)

Przykład A

* Wykładowcy, przedmioty, studenci, oceny
* Wykładowcy prowadzą zajęcia z poszczególnych przedmiotów
* Studenci uczęszczają na zajęcia
* Wykładowcy wystawiają oceny studentom
* Studenci oceniają zajęcia

Przykład B

* Firmy, wycieczki, osoby
* Firmy organizują wycieczki
* Osoby rezerwują miejsca/wykupują bilety
* Osoby oceniają wycieczki

Wybieram model A – baza danych uczelni.

1. Warto zaproponować/rozważyć różne warianty struktury bazy danych i dokumentów w poszczególnych kolekcjach oraz przeprowadzić dyskusję każdego wariantu (wskazać wady i zalety każdego z wariantów)

Wariant 1

Struktura bazy danych jest podzielona na 3 kolekcje: wykładowców, przedmioty i studentów. W tym modelu położyłem nacisk na redundancję danych i spójność, stosując podobny do relacyjnych baz danych model.

Zalety:

* Redundancja danych;
* Spójność;
* Duża wydajność dla poszczególnych kolekcji;
* Niewielki rozmiar dokumentów.

Wady:

* Średnia lub mała wydajność dla poszczególnych kolekcji przy zapytaniach agregacyjnych, które łączą kilka kolekcji;
* Zamodelowanie typowych dla tej bazy danych zapytań wymaga już łączenia kilku kolekcji;
* Mała czytelność z powodu posiadanych licznych obiektów ObjectId.

*// Lecturers*

*const* exampleLecturer1 = {

    "personalDetails": {

        "firstName": "John",

        "lastName": "Doe",

        "realLifeId": "532214321",

        "nationality": "Scottish"

*// pozostałe dane, nr telefonu, email uczelniany itd.*

*// ...*

    },

    "faculty": "Faculty of Computer Science, Electronics and Telecomunications",

    "degree": "Doctor of Philosophy",

    "roles": [

        "Head of the department"

*// ...*

    ],

    "taughtCourses": [

        ObjectId("\_id\_course\_0"),

        ObjectId("\_id\_course\_1")

*// ...*

    ]

};

*// Courses*

*const* exampleCourse1 = {

    "name": "Data Structures and Algorithms",

    "academicYear": [ 2021, 2022 ],

    "description": "Algorithms, yeah.",

    "participants": [

        ObjectId("\_id\_student\_0"),

        ObjectId("\_id\_student\_1"),

        ObjectId("\_id\_student\_2"),

        ObjectId("\_id\_student\_3"),

        ObjectId("\_id\_student\_4")

*// ...*

    ],

    "reviews": [

        {

            "value": 10,

            "date": new Date(),

            "comment": "Yeah, cool.",

            "issuedBy": ObjectId("\_id\_anne\_kowalski"),

        }

    ]

};

*// Students*

*const* exampleStudent1 = {

    "personalDetails": {

        "firstName": "Anne",

        "lastName": "Kowalski",

        "realLifeId": "00211388482",

        "nationality": "Polish"

*// pozostałe dane, nr telefonu, email uczelniany itd.*

*// ...*

    },

    "major": "Computer Science",

    "faculty": "Faculty of Computer Science, Electronics and Telecomunications",

    "participatedCourses": [

        ObjectId("\_id\_course\_0"),

        ObjectId("\_id\_course\_1"),

        ObjectId("\_id\_course\_2")

*// ...*

    ],

    "grades": [

        {

            "value": 4.5,

            "date": new Date(),

            "issuedBy": ObjectId("\_id\_john\_doe"),

            "courseId": ObjectId("\_id\_dsa\_course")

        }

*// ...*

    ]

};

Wariant 2

Struktura bazy danych jest podzielona także na 3 kolekcje: wykładowców, przedmioty i studentów. W tym modelu położyłem nacisk na wydajność modelu, stosując zagnieżdżone dokumenty, które zawierają pełne informacje.

Zalety:

* Duża wydajność zapytań bez względu na ich rodzaj;
* Posiadanie wszystkich danych w jednej kolekcji bez potrzebnej agregacji;

Wady:

* Bardzo duża powtarzalność tych samych danych poprzez powielenie dokumentów;
* Bardzo duży rozmiar dokumentów jak i samej bazy;
* Integralność między danymi jest utrudniona, w wypadku modyfikacji pojedynczego dokumentu w kolekcji należy zmodyfikować inne, gdzie występuje ten sam dokument także.
* Mała czytelność przez ilość danych w pojedynczym dokumencie.

Wariant 3

Wariant ten to wariant 2 z redukowaną liczbą pól, tablice obiektów w postaci grades, reviews, itd. zostały usunięte.

Zalety:

* Duża wydajność zapytań większości standardowych zapytań;
* Posiadanie wszystkich kluczowych danych w jednej kolekcji bez potrzebnej agregacji;
* Czytelność.

Wady:

* Duża powtarzalność tych samych danych poprzez powielenie dokumentów;
* Duży rozmiar dokumentów jak i samej bazy;
* Integralność między danymi jest utrudniona, w wypadku modyfikacji pojedynczego dokumentu w kolekcji należy zmodyfikować inne, gdzie występuje ten sam dokument także.
* Czasami trzeba zastosować agregację danych, łącząc kilka kolekcji, by pozyskać odpowiednie informacje.

Wariant 4

Struktura bazy danych danych z poprzednich punktów zostaje zmieniona, powstają 2 nowe kolekcje w postaci ocen i opinii. W tym modelu można zastosować łączenie relacji poprzez ich identyfikatory lub częściowe zagnieżdżanie dokumentów tak jak w wariancie 3.

1. Kolekcje należy wypełnić przykładowymi danymi

Zrobiłem to w punkcie a.

1. W kontekście zaprezentowania wad/zalet należy zaprezentować kilka przykładów/zapytań/zadań/operacji oraz dla których dedykowany jest dany wariantów

W sprawozdaniu należy zamieścić przykładowe dokumenty w formacie JSON ( pkt a) i b)), oraz kod zapytań/operacji (pkt c)), wraz z odpowiednim komentarzem opisującym strukturę dokumentów oraz polecenia ilustrujące wykonanie przykładowych operacji na danych

Do sprawozdania należy kompletny zrzut wykonanych/przygotowanych baz danych (taki zrzut można wykonać np. za pomocą poleceń mongoexport, mongdump …) oraz plik z kodem operacji zapytań (załącznik powinien mieć format zip).

**Punktacja za zadanie (razem 2pkt)**