Моделиране на взаимоотношения "едно към едно" с вградени документи

Ще разгледаме пример за модел на данни, който използва вградени документи за описване на взаимоотношения "едно към едно" между свързани данни. Вграждането на свързани данни в един документ може да намали броя на операциите за четене, необходими за получаване на данни. По принцип трябва да структурирате схемата си така, че приложението ви да получава цялата необходима информация с една операция за четене.

Модел на вграден документ

Разгледайте следния пример, който съпоставя връзките между клиент и неговия адрес. Примерът илюстрира предимството на вграждането пред препращането, ако трябва да видите една единица данни в контекста на другата. В тази едно към едно връзка между данните за клиента и адреса адресът принадлежи на клиента.

При нормализирания модел на данни документът за адрес съдържа препратка към документа за клиент.

```
// client document
{
  _id: "joe",
  name: "Joe Bookreader"
}
// address document
{
  client_id: "joe",  // препратка към client документа
  street: "123 Fake Street",
  city: "Faketon",
  state: "MA",
  zip: "12345"
}
```

Ако данните за адреса често се извличат заедно с информацията за името, тогава при препратките приложението ви трябва да прави множество заявки за разрешаване на препратката. По-добрият модел на данни би бил да вградите данните за адреса в данните за клиента, както е в следния документ:

```
{
  _id: "joe",
  name: "Joe Bookreader",
  address: {
       street: "123 Fake Street",
       city: "Faketon",
       state: "MA",
       zip: "12345"
   }
}
```

Благодарение на вградения модел на данни приложението ви може да извлича пълната информация за клиента с една заявка.

Модел за подмножество

Потенциален проблем с модела на вградения документ е, че той може да доведе до големи документи, които съдържат полета, от които приложението не се нуждае. Тези ненужни данни могат да доведат до допълнително натоварване на вашия сървър и да забавят операциите по

четене. Вместо това можете да използвате шаблона за подмножество, за да извлечете подмножеството от данни, които се достъпват най-често, с едно извикване на базата данни.

Нека да разгледаме приложение, което показва информация за филми. Базата данни съдържа колекция от филми със следната схема:

```
{
      "_id": 1,
      "title": "The Arrival of a Train",
      "year": 1896,
      "runtime": 1,
      "released": ISODate("01-25-1896"),
      "poster": "http://ia.media-imdb.com/images/M/MV5BMjEyNDk5MDYzOV5BMl5BanBnXkFtZ
                              TgwNjIxMTEwMzE@._V1_SX300.jpg",
      "plot": "A group of people are standing in a straight line along the platform of a railway station,
      waiting for a train, which is seen coming at some distance.
      When the train stops at the platform, ...",
      "fullplot": "A group of people are standing in a straight line along the platform of a railway
          station, waiting for a train, which is seen coming at some distance. When the train stops at
          the platform,      the line dissolves. The doors of the railway-cars open, and people on the
          platform help passengers to get off.",
      "lastupdated": ISODate("2015-08-15T10:06:53"),
      "type": "movie",
      "directors": [ "Auguste Lumière", "Louis Lumière" ],
      "imdb": {
            "rating": 7.3,
            "votes": 5043,
            "id": 12
      },
      "countries": [ "France" ],
      "genres": [ "Documentary", "Short" ],
      "tomatoes": {
             "viewer": {
                  "rating": 3.7,
                  "numReviews": 59
            },
           "lastUpdated": ISODate("2020-01-09T00:02:53")
      }
}
```

Понастоящем колекцията от филми съдържа няколко полета, от които приложението не се нуждае, за да покаже прост преглед на даден филм, като например пълен сюжет и информация за рейтинг. Вместо да съхранявате всички данни за филма в една колекция, можете да разделите колекцията на две колекции:

1. Колекцията с филми (**movie**) съдържа основна информация за даден филм. Това са данните, които приложението зарежда по подразбиране:

```
{
 "_id": 1,
 "title": "The Arrival of a Train",
 "year": 1896,
 "runtime": 1,
 "released": ISODate("1896-01-25"),
 "type": "movie",
 "directors": [ "Auguste Lumière", "Louis Lumière" ],
 "countries": [ "France" ],
 "genres": [ "Documentary", "Short" ],
}
```

2. Колекцията **movie_details** съдържа допълнителни, по-рядко използвани данни за всеки филм:

```
{
    "_id": 156,
    "movie_id": 1,  // препратка към колекция movie
    "poster": "http://ia.media-imdb.com/images/M/MV5BMjEyNDk5MDYzOV5BMl5BanBnXkFtZ
        TgwNjIxMTEwMzE@._V1_SX300.jpg",
    "plot": "A group of people are standing in a straight line along the platform of a railway station,
        waiting for a train, which is seen coming at some distance. When the train stops at the
        platform, ...",
    "fullplot": "A group of people are standing in a straight line along the platform of a railway station,
        waiting for a train, which is seen coming at some distance.
        When the train stops at the platform, the line dissolves. The doors of the railway-cars
        open, and people on the platform help passengers to get off.",
    "lastupdated": ISODate("2015-08-15T10:06:53"),
    "imdb": {
    "rating": 7.3,
    "votes": 5043,
    "id": 12
    },
    "tomatoes": {
         "viewer": {
              "rating": 3.7,
              "numReviews": 59
         },
        "lastUpdated": ISODate("2020-01-29T00:02:53")
    }
}
```

Този метод подобрява производителността на четене, тъй като изисква от приложението да чете по-малко данни, за да изпълни най-често срещаната си заявка. При необходимост приложението може да направи допълнително извикване към базата данни, за да изтегли по-рядко използваните данни.

Компромиси на модела на подмножеството

Използването на по-малки документи, съдържащи по-често използвани данни, намалява общия размер на работното множество. Тези по-малки документи водят до подобрена производителност при четене и осигуряват повече памет за приложението.

Важно е обаче да разбирате вашето приложение и начина, по който то зарежда данни. Ако неправилно разделите данните си на няколко колекции, приложението ви често ще трябва да прави многократни заявки до базата данни и да разчита на операциите JOIN, за да извлече всички данни, от които се нуждае.

Освен това разделянето на данните ви на много малки колекции може да увеличи необходимата поддръжка на базата данни, тъй като може да стане трудно да се проследи кои данни в коя колекция се съхраняват.

Изводи:

1. Когато обмисляте къде да разделите данните си, най-често използваната част от тях трябва да бъде в колекцията, която приложението зарежда първо.

Моделиране на взаимоотношения "едно към много" с вградени документи

Ще разгледаме модел на данни, който използва вградени документи, за да опише връзка от типа "едно към много" между свързани данни. Вграждането на свързани данни в един документ може да намали броя на операциите за четене, необходими за получаване на данни. По принцип трябва да структурирате схемата си така, че приложението ви да получава цялата необходима информация с една операция за четене.

Модел на вграден документ

Разгледайте следния пример, който съпоставя връзките между клиент и множество адреси. Примерът илюстрира предимството на вграждането пред препращането. Тази връзка е от тип "едно към много" – всеки клиент има множество същности на адреса.

При нормализирания модел на данните документите с адреси съдържат препратка към документа на клиента.

```
// client document:
{
       _id: "joe",
       name: "Joe Bookreader"
}
// address documents:
{
     client_id: "joe",   // препратка към client документа
     street: "123 Fake Street",
     city: "Faketon",
     state: "MA",
     zip: "12345"
}
{
     client_id: "joe",
     street: "1 Some Other Street",
     city: "Boston",
     state: "MA",
     zip: "12345"
}
```

Ако приложението ви често извлича данни за адреса с информация за името, тогава приложението ви трябва да прави множество заявки за разрешаване на препратките. По-оптимална схема би била да се вградят същностите на адресните данни в данните за клиента, както е в следния документ:

```
{
"_id": "joe",
     "name": "Joe Bookreader",
     "addresses": [
          {
               "street": "123 Fake Street",
               "city": "Faketon",
               "state": "MA",
               "zip": "12345"
          },
          {
               "street": "1 Some Other Street",
               "city": "Boston",
               "state": "MA",
               "zip": "12345"
```

```
        }
    ]
}
```

Благодарение на вградения модел на данни приложението ви може да извлича пълната информация за клиента с една заявка.

Модел на подмножество

Потенциален проблем при образеца за вграден документ е, че той може да доведе до големи документи, особено ако вграденото поле е неограничено. В този случай можете да използвате шаблона за подмножество, за да получите достъп само до данните, които се изискват от приложението, вместо до целия набор от вградени данни.

Да разгледаме сайт за електронна търговия, който има списък с отзиви за даден продукт:

```
{
    "_id": 1,
    "name": "Super Widget",
    "description": "This is the most useful item in your toolbox.",
    "price": { "value": NumberDecimal("119.99"), "currency": "USD" },
    "reviews": [
        {
            "review_id": 786,
            "review_author": "Kristina",
            "review_text": "This is indeed an amazing widget.",
            "published_date": ISODate("2019-02-18")
        },
        {
            "review_id": 785,
            "review_author": "Trina",
            "review_text": "Nice product. Slow shipping.",
            "published_date": ISODate("2019-02-17")
        },
        ... // други отзиви
        {
            "review_id": 1,
            "review_author": "Hans",
            "review_text": "Meh, it's okay.",
            "published_date": ISODate("2017-12-06")
        }
    ]
}
```

Отзивите са подредени в обратен хронологичен ред. Когато потребителят посети страница на продукт, приложението зарежда десетте най-нови прегледа. Вместо да съхранявате всички отзиви заедно с продукта, можете да разделите колекцията на две колекции:

1. В колекцията на продукта (**product**) се съхранява информация за всеки продукт, включително десетте най-нови прегледа на продукта:

```
{
 "_id": 1,
 "name": "Super Widget",
 "description": "This is the most useful item in your toolbox.",
 "price": { "value": NumberDecimal("119.99"), "currency": "USD" },
 "reviews": [
  {
   "review_id": 786,
   "review_author": "Kristina",
   "review_text": "This is indeed an amazing widget.",
```

```
    "published_date": ISODate("2019-02-18")
   }
   ...
   {
    "review_id": 777,
    "review_author": "Pablo",
    "review_text": "Amazing!",
    "published_date": ISODate("2019-02-16")
   }
 ]
}
```

2. В колекцията от прегледи (**review**) се съхраняват всички отзиви. Всеки отзив съдържа препратка към продукта, за който е написан.

```
{
 "review_id": 786,
 "product_id": 1,
 "review_author": "Kristina",
 "review_text": "This is indeed an amazing widget.",
 "published_date": ISODate("2019-02-18")
}
{
 "review_id": 785,
 "product_id": 1,
 "review_author": "Trina",
 "review_text": "Nice product. Slow shipping.",
 "published_date": ISODate("2019-02-17")
}
...
{
 "review_id": 1,
 "product_id": 1,
 "review_author": "Hans",
 "review_text": "Meh, it's okay.",
 "published_date": ISODate("2017-12-06")
}
```

Чрез съхраняването на десетте най-нови рецензии в колекцията от продукти при извикването на колекцията от продукти се връща само необходимото подмножество от общите данни. Ако потребителят желае да види допълнителни отзиви, приложението извършва повикване към колекцията с прегледи.

Компромиси на модела на подмножеството

Използването на по-малки документи, съдържащи по-често използвани данни, намалява общия размер на работното множество. Тези по-малки документи водят до подобряване на производителността при четене на данните, до които приложението има най-чест достъп.

Въпреки това моделът на подмножеството води до дублиране на данни. В примера прегледите се поддържат както в колекцията с продукти, така и в колекцията с прегледи. Трябва да се предприемат допълнителни стъпки, за да се гарантира, че прегледите са последователни между всяка колекция. Например, когато клиент редактира своето ревю, може да се наложи приложението да извърши две операции за запис: една за актуализиране на колекцията с продукти и една за актуализиране на колекцията с ревюта.

Също така трябва да приложите логика в приложението, за да гарантирате, че ревютата в колекцията на продукта са винаги десетте най-нови ревюта за този продукт.

Изводи:

1. Когато обмисляте къде да разделите данните си, най-често използваната част от тях трябва да бъде в колекцията, която приложението зарежда първо. В този пример схемата е разделена на десет прегледа, защото това е броят на прегледите, видими в приложението по подразбиране.

Моделиране на взаимоотношения "едно към много" с препратки към документи

Ще разгледаме модел на данни, който използва препратки между документи, за да опише връзките от типа "едно към много" между свързани данни. Ще разгледаме пример, който съпоставя връзките между издател и книга. Примерът илюстрира предимството на препратките пред вграждането, за да се избегне повтарянето на информацията за издателя.

Вграждането на документа на издателя в документа на книгата би довело до повтаряне на данните на издателя, както показват следните документи:

```
{
  title: "MongoDB: The Definitive Guide",
  author: [ "Kristina Chodorow", "Mike Dirolf" ],
  published_date: ISODate("2010-09-24"),
  pages: 216,
  language: "English",
  publisher: {
         name: "O'Reilly Media",
         founded: 1980,
         location: "CA"
        }
}
{
  title: "50 Tips and Tricks for MongoDB Developer",
  author: "Kristina Chodorow",
  published_date: ISODate("2011-05-06"),
  pages: 68,
  language: "English",
  publisher: {
         name: "O'Reilly Media",
         founded: 1980,
         location: "CA"
        }
}
```

За да избегнете повторение на данните за издателя, използвайте препратки и съхранявайте информацията за издателя в отделна колекция от колекцията с книги.

Когато използвате препратки, растежът на връзките определя къде да се съхранява препратката. Ако броят на книгите на издател е малък с ограничен растеж, съхраняването на препратката към книгата вътре в документа на издателя понякога може да е полезно. В противен случай, ако броят на книгите на издател е неограничен, този модел на данни би довел до променливи, нарастващи масиви, както в следния пример:

```
{
  name: "O'Reilly Media",
  founded: 1980,
  location: "CA",
  books: [123456789,  234567890, ...]
}
{
   _id: 123456789,
   title: "MongoDB: The Definitive Guide",
   author: [ "Kristina Chodorow", "Mike Dirolf" ],
   published_date: ISODate("2010-09-24"),
   pages: 216,
   language: "English"
}
```

```
{
  _id: 234567890,
  title: "50 Tips and Tricks for MongoDB Developer",
  author: "Kristina Chodorow",
  published_date: ISODate("2011-05-06"),
  pages: 68,
  language: "English"
}
```

За да избегнете променящи се, нарастващи масиви, съхранявайте препратката към издателя в документа на книгата:

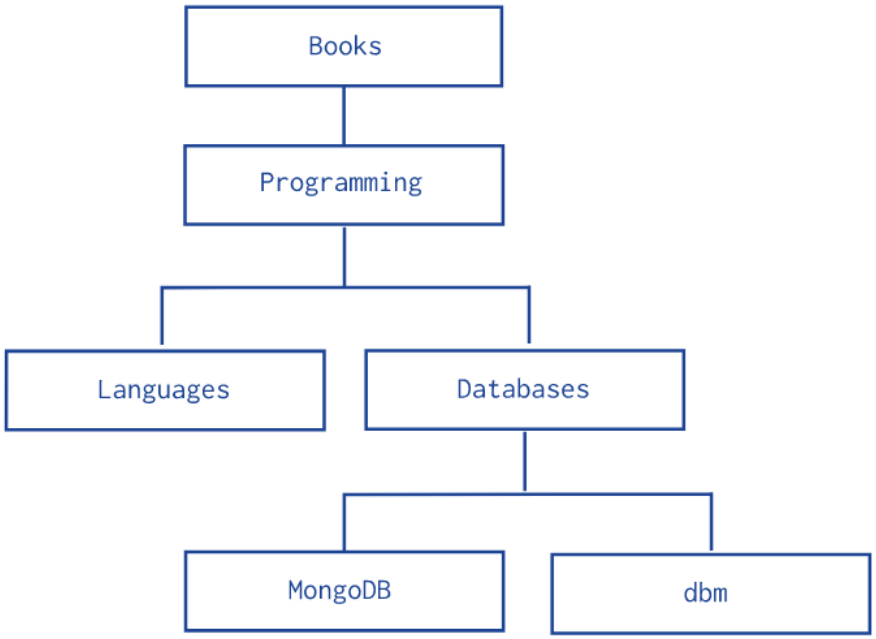
```
{
  _id: "oreilly",
  name: "O'Reilly Media",
  founded: 1980,
  location: "CA"
}
{
  _id: 123456789,
  title: "MongoDB: The Definitive Guide",
  author: [ "Kristina Chodorow", "Mike Dirolf" ],
  published_date: ISODate("2010-09-24"),
  pages: 216,
  language: "English",
  publisher_id: "oreilly"
}
{
  _id: 234567890,
  title: "50 Tips and Tricks for MongoDB Developer",
  author: "Kristina Chodorow",
  published_date: ISODate("2011-05-06"),
  pages: 68,
  language: "English",
  publisher_id: "oreilly"
}
```

Моделиране на дървовидни структури с родителски препратки

Ще разгледаме модел на данни, който описва дървовидна структура в документите на MongoDB чрез съхраняване на препратки към "родителски" възли в дъщерни възли.

Моделът "Препратки към родители" (**Parent References**) съхранява всеки дървовиден възел в документ; в допълнение към дървовидния възел документът съхранява идентификатора на родителя на възела.

Разгледайте следната йерархия от категории:



Следващият пример моделира дървото с помощта на препратки към родителите, като съхранява препратката към родителската категория в полето родител (parent):

```
[
   { _id: "MongoDB", parent: "Databases" },
   { _id: "dbm", parent: "Databases" },
   { _id: "Databases", parent: "Programming" },
   { _id: "Languages", parent: "Programming" },
   { _id: "Programming", parent: "Books" },
   { _id: "Books", parent: null }
]
```

Запитването за извличане на родителя на даден възел е бързо и лесно:

```
db.categories.findOne( { _id: "MongoDB" } ).parent
```

Можете да направите заявка по родителското поле, за да намерите непосредствените му дъщерни възли:

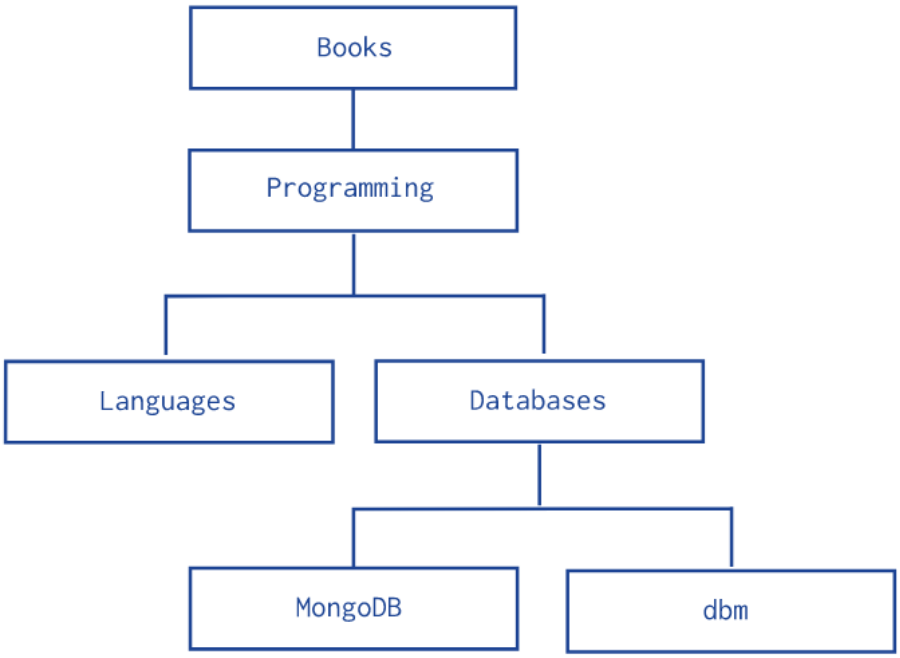
```
db.categories.find( { parent: "Databases" } )
```

Моделиране на дървовидни структури с препратки към дъщерните възли

Ще разгледаме модел на данни, който описва дървовидна структура в документите на MongoDB чрез съхраняване на препратки в родителските възли към дъщерните възли.

Моделът "Препратки към деца" (Child References) съхранява всеки възел на дърво в документ; в допълнение към възела на дърво документът съхранява в масив идентификаторите на децата на възела.

Разгледайте следната йерархия от категории:



Следващият пример моделира дървото с помощта на препратки към деца, като съхранява препратката към децата на възела в полето children:

```
[
   { _id: "MongoDB", children: [] },
   { _id: "dbm", children: [] },
   { _id: "Databases", children: [ "MongoDB", "dbm" ] },
   { _id: "Languages", children: [] },
   { _id: "Programming", children: [ "Databases", "Languages" ] },
   { _id: "Books", children: [ "Programming" ] }
]
```

Запитването за извличане на непосредствените деца на даден възел е бързо и лесно:

```
db.categories.findOne( { _id: "Databases" } ).children
```

Можете да направите заявка за възел в полето за деца, за да намерите родителския му възел, както и неговите братя и сестри:

```
db.categories.find( { children: "MongoDB" } )
```

Модел на данни за атомарни операции

Въпреки че MongoDB поддържа транзакции с много документи за набори от реплики (от версия 4.0) и клъстери с раздробени данни (от версия 4.2), за много сценарии денормализираният модел на данни ще продължи да бъде оптимален за вашите данни при определени случаи.

В MongoDB операцията за запис на един документ е атомична. За полета, които трябва да бъдат актуализирани заедно, вграждането на полетата в един и същ документ гарантира, че полетата могат да бъдат актуализирани атомично.

Например, разгледайте ситуация, в която трябва да поддържате информация за книги, включително броя на наличните за закупуване копия, както и текущата информация за закупуване. Наличните копия на книгата и информацията за касовата наличност трябва да бъдат синхронизирани. По този начин вграждането на полето за налични копия и полето за касова наличност в един и същи документ гарантира, че можете да актуализирате двете полета атомично.

```
{
    _id: 123456789,
    title: "MongoDB: The Definitive Guide",
    author: [ "Kristina Chodorow", "Mike Dirolf" ],
    published_date: ISODate("2010-09-24"),
    pages: 216,
    language: "English",
    publisher_id: "oreilly",
    available: 3,
    checkout: [
        {
          by: "joe",
          date: ISODate("2012-10-15")
        }
    ]
}
```

След това, за да актуализирате с нова информация за касата, можете да използвате метода db.collection.updateOne(), за да актуализирате атомично полето за наличност и полето за каса:

```
db.books.updateOne (
    { _id: 123456789, available: { $gt: 0 } },
    {
      $inc: { available: -1 },
      $push: { checkout: { by: "abc", date: new Date() } }
    }
)
```

Операцията връща документ, който съдържа информация за състоянието на операцията:

```
{ "acknowledged" : true, "matchedCount" : 1, "modifiedCount" : 1 }
```

Полето **matchedCount** показва, че 1 документ е отговарял на условието за актуализация, а полето **modifiedCount** показва, че операцията е актуализирала 1 документ.

Ако нито един документ не отговаря на условието за актуализация, тогава matchedCount и modifiedCount ще бъдат 0 и ще показват, че не можете да закупите книгата.

Модел на данни в подкрепа на търсенето по ключови думи

Търсенето по ключова дума не е същото като търсенето на текст или на пълен текст и не предоставя функции за обработка на текст. При версия 2.4 на MongoDB се предоставя функция за търсене на текст.

Ако приложението ви трябва да извършва заявки върху съдържанието на поле, което съдържа текст, можете да извършвате точни съвпадения на текста или да използвате $regex, за да използвате съвпадения на шаблони с регулярни изрази. За много операции върху текст обаче тези методи не удовлетворяват изискванията на приложението.

Този шаблон описва един метод за поддържане на търсене по ключови думи с помощта на MongoDB за поддържане на функционалността за търсене на приложения, който използва ключови думи, съхранявани в масив в същия документ като текстовото поле. В комбинация с индекс с няколко ключа този модел може да поддържа операциите за търсене по ключови думи на приложението.

За да добавите структури към вашия документ, за да поддържате заявки, базирани на ключови думи, създайте поле с масив във вашите документи и добавете ключовите думи като низове в масива. След това можете да създадете индекс с няколко ключа върху масива и да създадете заявки, които избират стойности от масива.

Нека е дадена е колекция от библиотечни томове, за която искате да осигурите тематично търсене. За всеки том добавяте темите на масива и добавяте толкова ключови думи, колкото са необходими за дадения том.

За книгата Moby-Dick може да имате следния документ:

```
{
    title : "Moby-Dick" ,
    author : "Herman Melville" ,
    published : 1851 ,
    ISBN : 0451526996 ,
    topics : [ "whaling", "allegory", "revenge , "American",  "novel", "nautical", "voyage", "Cape Cod" ]
}
```

След това създавате индекс с няколко ключа върху масива от теми:

```
db.volumes.createIndex( { topics: 1 } )
```

Индексът с няколко ключа създава отделни индексни записи за всяка ключова дума в масива с теми. Например индексът съдържа един запис за лов на китове и друг за алегория. След това се прави заявка въз основа на ключовите думи. Например:

```
db.volumes.findOne( { topics : "voyage" }, { title: 1 } )
```

Ограничения на индексите на ключови думи

MongoDB може да поддържа търсене по ключови думи, като използва специфични модели на данни и многоключови индекси.