Проектиране на MongoDB бази данни

I. Въведение

Релационните бази данни имат фиксирана **схема** на описание на данните. Полетата от таблиците имат фиксиран тип и дори да не се налага да приемат в даден момент стойност, те трябва да имат стойност, дори тя да е null. Това води до неефективно използване на паметта. За нерелационните бази данни често се казва, че са бази без схема. На практика всяка база данни има схема, но при NoSQL базите данни схемата е *динамична*. Това означава, че не е наложително схемата твърдо да се прилага за всички документи в дадена колекция. Напълно допустимо е да липсват определени полета в част от документите. Схемата се дефинира на ниво приложение. Тя трябва да може да се развива с промяната на приложението. Такава схема се нарича и *полиморфна*.

Дизайнът на схемата на базата данни е важен процес тъй като от него до голяма степен зависи производителността на цялата услуга, както и мащабируемостта на базата данни. Преди да проектирате базата данни, трябва да отговорите на редица въпроси, например:

1. Кои са обектите с които приложението ви работи и които трябва да са част от база данни?
2. Колко и от какъв тип са връзките между тези обекти (1:1, 1:N или M:N)?
3. Колко е броят на потребителите на вашето приложение и колко често ще се извлича информация от базата данни?
4. Колко често се налага вмъкване на нови обекти в базата данни?
5. Колко често се налага изтриване на обекти от базата данни?
6. Колко често се налага модифициране на съдържанието на обекти от базата данни?
7. Какви заявки до базата данни ще реализира приложението ви?
8. Как ще се реализира достъпа до обектите от базата данни – чрез единичен ключ, чрез съставен ключ, чрез стойността на свойства или чрез прилагане на някакъв вид филтриране и сортиране на данните?

1.1 Видове връзки между обектите

Връзка 1:1

Връзката "едно към едно" е вид кардиналност, която описва връзката между две същности, при която един запис от същност A е свързан с един запис в същност B. Този тип връзка може да се моделира по два начина: чрез вграждане на връзката като поддокумент, или свързване с документ в отделна колекция. Всичко зависи от това колко често се прави достъп до данните, а също и от жизнения цикъл на набора от данни - ако обект A се изтрие, трябва ли обект B да продължи да съществува?

Връзка 1:N

Връзката “едно към много” се отнася до връзката между две същности A и B, при която едната страна може да има една или повече връзки с другата, докато обратната връзка е едностранна (един студент изучава множество дисциплини). Подобно на връзката 1:1, тя също може да бъде моделирана чрез използване на вграждане или свързване.

Връзка N:M

Тази връзка е от тип „много към много” се отнася до връзка между две същности A и B, като и двете страни могат да имат една или повече връзки към другата. Ако имате опасение, че моделирането на тази връзка с един документ ще доведе до силно нарастване на размера му, то в този случай се предпочита A и B да се моделират като самостоятелни колекции. Това е компромисно решение, тъй като ще трябва да извършим втора заявка, за да получим данните за същност B, така че скоростта на четене може да бъде намалена. Не трябва да се забравя за възможността за обединение между A и B на ниво приложение. При правилното индексиране (за оптимизиране на паметта) обединенията от страна на сървъра са малко по-скъпи от тези, които се реализират от самата база данни.

От изложеното по-горе стана ясно, че при дизайн на схемата на базата данни се използват два основни подхода: **вграждане** и **свързване**.

1.2 Денормализиран (вграден) модел на данните

При документния тип NoSQL бази данни можете да вграждате свързани данни в една структура на документ. Тези схеми формират *денормализирани* модел на данните. Денормализацията на базата е процес при който се търсят обекти с връзки от тип 1:1. В този случай тези обекти трябва да се моделират чрез един документ. Това обаче трябва да се направи само ако обектите не са големи по размер и не се обновяват често. Предимството на денормализацията е бързодействието на обслужване на заявките – необходимо е да се достъпи само един документ, който съдържа цялата необходима информация. Вградените модели на данни позволяват на приложенията да съхраняват свързани части от информация в един и същи запис в базата данни. В резултат на това на приложенията може да се наложи да правят по-малко заявки и актуализации за извършване на общи операции.

Нека да създадем схема, която описва студенти при условие, че необходимата информация е следната: идентификационен номер на студента, име и фамилия, информация за връзка с него (e-mail адрес, номер на телефон, facebook профил). Ако информацията за връзка със студента се използва много често от вашето приложение, то тя трябва да се вгради в документа (виж Фиг. 4.1). Вграденият документ е поле “contact” и неговата стойност.

Вграждането осигурява по-добра производителност при операции четене, както и възможност за заявяване и извличане на свързани данни с една операция. Вградените модели на данни дават възможност да се актуализират свързаните данни с една атомична операция за запис. За да получите достъп до данни във вградени документи, използвайте точкова нотация за достигнете до вградените документи.

Фиг.4.1 Пример за схема, използваща вграждане

Вграден модел на данните е подходящо да се използва при:

- Наличие на връзки от тип “един към един”.
- Наличие на връзки от тип “един към много”. Необходимо е подчинените (“много”) документи да са в силна връзка с родителския документ (“един”).
- Гарантиране на интегритет при операции за четене и запис на данни от типа “едно към едно” и “едно към много”, които се изтриват заедно по подразбиране.
- Когато данните много по-често се четат отколкото обновяват.

Този модел не е подходящо да се използва при:

- Наличие на връзки от тип “много към много”.
- Тенденция за бързо увеличаване на размера на вградените документи. Трябва да се има предвид, че има физическо ограничение за размера на всеки документ и тенденция за намаляване на производителността при работа с много големи по размер файлове.
- Ако се налага много често модифициране на вградените данни.
- При независимост на заявките към документите от различните колекции

1.3 Нормализиран модел на данните

Нормализираните модели на данни описват взаимоотношенията с помощта на препратки между документите. Нормализацията е процес, който има за задача да намали излишъка и взаимозависимостта на информацията в базата. За целта трябва да се прецени кои обекти да се опишат в отделен документ. Едно от предимствата на нормализацията е, че базата ще стане по-малка по размер (няма да има повторение на обекти в рамките на една колекция. Един от недостатъците на нормализацията е, че се увеличават броя на заявките – получаването на крайният резултат може да изисква заявки към множество взаимосвързани документи. Когато връзката между два обекта е “1:N”, то най-логично е двата обекта да се моделират в два различни документа. Например, потребителят на една книжарница (1) и книгите (N), които той заема трябва да са отделни документи: всеки потребител може да заема множество книги и няколко потребители могат да заемат едни и същи книги. Книгите, които всеки потребител заема, ще се идентифицират с “bookId” на книгите, а за всяка книга ще има отделен документ, в който се описва идентификатора й, автора, наименованието, броя страници, жанра на книгата и др.

Ако използваме нормализиран модел за примера със описанието на студенти ще създадем две колекции. В едната с име “students” ще се съдържат документи, които описват студентите с идентификатор и име, а в другата с име “contact” ще бъдат всички документи, които описват информация за връзка със студентите. Връзката на документите от двете колекции ще се реализира чрез двойката “id-studentId” (виж Фиг. 4.2).

```
{
    "id": "1234567",
    "name":
     [
        {
            "firstname": "Иван",
            "lastname": "Георгиев"
        }
    ]
)
```

Документ от колекция “student”

```
{
    "studentId": "1234567",
    "mobile": "YYYYXXXXXX",
    "email": "you@domain.com",
    "facebook": "……"
}
```

Документ от колекция “contact”

Фиг.4.2 Пример за схема, използваща връзки между документите

Свързан модел на данните е подходящо да се използва при:

- Вграждането би довело до дублиране на данни, но не би осигурило достатъчно предимства в производителността при четене.
- Трябва да се моделират по-сложни взаимоотношения от тип "много към много".
- Трябва да се моделират големи йерархични масиви от данни.

Нека да разгледаме друг пример. Клиентите на дадена услуга могат да имат два адреса - домашен и на работното си място. Тук въпросът е дали тези адреси да са част от документа, който описва клиент или да бъдат в отделни документи. Тъй като домашният адрес е сравнително уникален за всеки клиент, нормално е той да е част от документа, който описва клиента. Множество клиенти обаче може да имат един и същ работен адрес. Ако вашата програма трябва да проверява често кои от вашите клиенти работят на едно и също място, то трябва да интегрирате и работния адрес в документа, който описва клиент. В противен случай, работното място трябва да е отделен документ, за да се намали размера на документите, описващи клиентите.

Нормализация или денормализация?

Заявките към една база данни могат да бъдат Create, Read, Update и Delete (CRUD). Критични са операциите при които се реализира промяна на съдържанието на базата данни. Целта е всички клиенти на услугата е да получат последно модифицираните данни, дори когато базата е с разпределена архитектура. Следователно, операциите, свързани със запис, трябва да са атомарни. Това означава, че само един процес трябва да има права да обновява един документ или колекция в даден момент от време. Ако документа е денормализиран, атомарността (невъзможни са блокировки на данни) на операция запис се гарантира лесно. Това не е в сила за нормализираните документи. В този случай едно обновяване може да изисква запис в множество документи. Повечето NoSQL бази данни решават проблема с атоматността на операции запис чрез репликиране на базата данни (дублиране на данните на множество инстанции на базата, които формират клъстер).

Съществуват три основни **правила** на базата на които се определя дали да използвате нормализация или денормализация:

Правило № 1: Ако обект B трябва да бъде достъпен самостоятелно (извън контекста на родителския обект A), тогава използвайте връзка, в противен случай - вграждане.

Правило № 2: Масивите не трябва да растат неограничено:

- Ако броят на документите от страна B е по-малък от няколко стотин и са малки по размер, то не е проблем да ги вградите в обекта A.
- Ако броят на документите от страна B е по-голям от няколко хиляди използвайте препратки като отделите обектите в отделна колекция.

Правило № 3: Връзките на ниво приложение са обичайна практика и не трябва да се отхвърлят; в тези случаи изборът на полета за индексиране до голяма степен определя производителността на заявките.

MongoDB дава възможност да проектираме схемата така, че да отговаря на нуждите на приложенията на потребителите. Не трябва да се забравя, че проблемите с производителността често се дължат на неправилен подбор на схемата на базата данни.

Избор на правилния брой на документите

Основателен е въпросът дали да се работи с много на брой малки по размер документи, или с малко документи, които съдържат много на брой полета и техните стойности. Ако бързодействието е важният за вас параметър, то трябва да се работи с колкото се може по-малко на брой документи, тъй като дисковите операции са много бавни в сравнение с операциите в паметта. Ако един документ в даден момент не е необходим за функционирането на вашата система, то най-добре е той да бъде изтрит. Затова още при проектирането трябва да се дефинира времето на живот на всеки документ.

1.4 Шаблони за моделиране на данните

Шаблоните за моделиране на данните са пряко свързани с опита на дизайнерите на бази данни и добрите практики в тази област. Всеки шаблон позволява бързо и надеждно решаване на определен проблем или набор от проблеми. Познанията в тази област са ключови от гледна точка време на дизайн и качество на дизайна. За да изберете подходящия шаблон трябва да знаете не само какви шаблони вече съществуват, но и кой шаблон при кой сценарий е най-подходящ (use case scenario). В Табл. 4.1 са описани имената на 12 шаблони за дизайн, които се използват при MongoDB и тяхното практическо приложение.

1. Шаблон “Approximation”

Моделът “Approximation” е добро решение за приложенията, които работят с данни, чието изчисляване е скъпо, тъй като изисква много изчислителни ресурси, а точността не е от решаващо значение. Апроксимирането на входните данни позволява да извършваме по-малко записи в базата данни, което увеличава производителността на услугата, и все пак да поддържаме статистически валидни числа. Например, ако трябва да изчисляваме средната стойност на температурата на годишна база в даден град няма смисъл да записваме в базата данни информация за температурата през малък интервал от време, а например веднъж на ден. В този случай не е толкова важна и точността на резултата. От страна на услугата може да покажем, че средната годишна температура в избран град е 18°C, а не реалното число, например 18.03°C.

Табл. 4.1 Шаблони за дизайн на MongoDB бази данни

Примерно приложение / Шаблон за дизайн	Каталог	Управление съдържанието	Интернет на нещата	Мобилни приложения	Персонализация	Анализ в реално време	Общ изглед
1. Approximation	■		■	■		■	
2. Attribute	■	■					■
3. Bucket			■			■	
4. Computed	■		■	■	■	■	■
5. Document Versioning	■	■			■		■
6. Extended Reference	■			■		■	
7. Outlier			■	■	■		
8. Preallocated			■			■	
9. Polymorphic	■	■		■			■
10. Schema Versioning	■	■	■	■	■	■	■
11. Subset	■	■		■	■		
12. Tree and Graph	■	■					

Проблемът при този шаблон е да преценим колко често да записваме в базата данни. Дали да бъде всяка минута, всеки час, всеки ден, през седмица или всеки месец. Друг типичен пример за използване на този модел е при обработка на данни от сензори от безжична сензорна мрежа (WSN). Тук основният въпрос е колко често трябва да записваме данните от сензорите в базата. Правилният отговор зависи от това колко инертни са тези данни и какво е значимото им изменение при което трябва да се активира някакво действие. Ако сензорът например измерва температура и сме преценили, че значимото изменение е 5 градуса, то в базата трябва да записваме, само когато се достигне този предварително дефиниран праг.

Плюсове

- По-малко записи в базата данни.
- Поддържане на статистически валидни данни.

Недостатъци

- Не се представят точните стойности на данните.
- Изисква се изпълнение на код от страна на приложението (поддържане на брояч, генератор на случайни числа).

2. Шаблон “Attribute”

Шаблонът "Attribute" е полезен при документи с много сходни полета, но имат и подмножество от полета с общи характеристики по които информацията трябва да се сортира или да се правят заявки. Шаблонът е подходящ и когато трябва да сортираме по стойностите на полета, които са налични само в малка част от документите. Следователно, този шаблон трябва да се използва за схеми, които имат набори от полета с един и същ тип стойност, например списъци с дати. Той работи добре и при работа с характеристики на продукти. Някои продукти, като например дрехи и обувки имат размери, които се описват по различен на чин в различните държави. Нека да опишем артикули от електронен магазин за облекло. Всички артикули имат общи полета като код на артикула, наименование на артикула, фирма производител и др. Всеки артикул има размер, но стойността му се различава в зависимост от локацията на клиента. На Фиг. 4.3а е показан част от документ, който описва артикул “обувки”. Размерите на обувките в различните локации са описани като отделни полета. Търсенето на размера на обувките ще изисква търсене в много полета едновременно. Поради тази причина ще ни трябват няколко индекса за колекцията ни от артикули (виж Фиг. 4.3б). С помощта на шаблон “Attribute” можем да преместим информацията за размера на артикула в масив от двойки „ключ-стойност” и така да намалим нуждата от използването на множество индекси (виж Фиг. 4.3в). В този случай търсенето ще бъде много по-бързо, тъй като ще използваме само един съставен ключ (виж Фиг. 4.3г).

```
{
   category: "shoes",
   manufacturer: "Ermenegildo Zegna",
   model: "Derby",
   ...
   size_EU: 42,
   size_USA: 8,
   size_UK: 8.5,
   size_JP: 26.5,
   ...
}
```

а) Документ от колекция “артикул”

```
{ size_EU:1 }
{ size_USA:1 }
{ size_UK:1 }
...
```

б) Индекси

```
{
   category: "shoes",
   manufacturer: "Ermenegildo Zegna",
   model: "Derby",
   ...
   size: [
      {
         location: "EU",
         value: 42
      },
      {
         location: "USA",
         value: 8
      },
      ...
   ]
}
```

в) Документ използващ шаблон “Attribute”

```
{
   size.location:1,
   size.value:1
}
```

г) Нов индекс

Фиг.4.3 Използване на шаблон “Attribute”

Можем още да намалим размера на документите и да използваме само един ключ като използваме стойността на поле “location” за ключ. При този вариант на схемата документите ще имат следното съдържание:

```
{
   category: "shoes",
   manufacturer: "Ermenegildo Zegna",
   model: "Derby",
   ...
   size:
      {
         "EU": 42,
         "USA": 8,
         ...
      }
}
```

Фиг.4.4 Подобрен вариант на документа описващ обувки

3. Шаблон “Bucket”

Когато данните постъпват като поток за определен период от време (данни от сензори например), може да съхраняваме всяко измерване в отделен документ. Ако имаме сензор, който измерва температурата и я записва в базата данни всяка минута, нашият поток от данни може да изглежда по следния начин:

```
{
   sensor_id: 1,
   timestamp: ISODate("2021-09-01T08:00:00.000Z"),
   temperature: 23.4
}
{
   sensor_id: 1,
   timestamp: ISODate("2021-09-01T08:01:00.000Z"),
   temperature: 22.8
}
```

а) Потоково генерирани документи – по един на всяка минута

```
{
    sensor_id: 1,
    measurements: [
       {
          timestamp: ISODate("2021-09-01T08:00:00.000Z"),
          temperature: 23.4
       },
       {
          timestamp: ISODate("2021-09-01T08:01:00.000Z"),
          temperature: 22.8
       }
    ],
    measuremets_count: 2,
    sum_temperature: 46.2
}
```

б) Един документ, който ще съдържа данните от сензора за определен период от време

Фиг.4.5 Пример за използване на шаблон “bucket”

Ако използваме шаблон “bucket”, данните от сензора ще се записват в масив “measurements” за зададен период от време, например 1 час. В нашия случай в масива има само два записа, но той ще расте, докато записите станат 60. При всеки запис в масива се инкрементира брояч на измерванията “measurement_count” и се актуализирана текущата сумата на всички температури в масива - “sum_temperature”. Ако в даден момент е необходима средната температура, можем лесно и бързо да я получим чрез разделяне на стойностите на двете полета, sum_temperature / measurement_count.

Плюсове

- Необходими са по-малко индекси.
- Заявките стават по-прости за писане и като цяло са по-бързи.

4. Шаблон “Computed”

Този шаблон се използва с цел намаляване на натоварването на процесора и увеличаване на производителността на приложението. Винаги, когато вашата система извършва едни и същи изчисления многократно и имате високо съотношение между броя на операции четене и запис, може да използвате шаблон “Computed”. За да намалите броя на изчисленията може да добавите времеви печат (timestamp) към документа, който да показва кога е бил последно актуализиран. След това, приложението може да определи кога трябва да се извърши изчислението. Друг вариант би могъл да бъде опашка от изчисления, които трябва да се извършат. Изборът на стратегията за актуализация е най-добре да се остави на разработчика на приложението.

Плюсове

- Намаляване на натоварването на процесора при чести изчисления.
- Заявките стават по-прости за писане и като цяло са по-бързи.

Минуси

- Прилагането или прекомерното използване на шаблона трябва да се избягва, освен ако не е необходимо.

5. Шаблон “Document Versioning”

Този шаблон се използва когато трябва да се запазят старите версии на някои документи, вместо да се използва втора система за управление. За да постигнем това, добавяме поле към всеки документ, което ни позволява да следим версията на документа. След това базата данни ще има две колекции: една, която съдържа най-новите (и най-търсените данни), и друга, която съдържа всички стари версии на документите. Типични примери за услуги при които трябва да се запазват старите версии на документи са тези, свързани с: финанси, здравна, застрахователната и правна системи. Трябва да се има предвид,че някои NoSQL бази данни като CouchDB например поддържат автоматично версиите на документите, тъй като това се използва при репликация на данните.

Плюсове

- Лесен за внедряване шаблон.
- Не оказва влияние върху производителността на заявките към колекцията с последната версия на документите.

Минуси

- Удвоява броя на записите.
- Заявките трябва да са насочени към правилната колекция.

6. Шаблон “Extended Reference”

Шаблонът “Extended Reference” се използва, за да намали броя на операции JOIN при често достъпвани данни от различни колекции. За да ускорим времето за изпълнение на заявките трябва да използваме влагане на поддокумент в базов документ. Вграждането на цялата информация е един документ ще намали броя на операции JOIN, но ще доведе до много дублирана информация. При този шаблон вместо да вграждаме цялата информация или да включваме препратка за свързване на информацията, вграждаме само полетата с най-висок приоритет и тези които най-често се достъпват. По този начин се подобрява производителността на услугата.

Плюсове

- Подобрява производителността, когато са необходими много операции JOIN.
- По-бързо четене и намаляване на общия брой на JOIN операциите.

Недостатъци

- Дублиране на данни.

7. Шаблон “Outlier”

Нека да е необходимо една социална мрежа да пази идентификаторите на всички клиенти, които са ви последователи. Ако имате няколко стотин последователи, техните идентификатори могат да се стойностите на поле “ followers” от тип масив. Ако броя на последователите ви надмине някакъв праг, например 1000, това решение не е приемливо. В този случай се казва, че имаме неограничено разрастващ се масив. Шаблон “outlier” решава проблема като първите 1000 последователи се вграждат в основния документ като масив. Добавя се ново поле, което показва дали имате още последователи, описани в друг документ, например поле “more_followers” което приема стойност true или false. При стойност на това поле true ще се знае, че при този документ има отклонение (outlier) от базовата схема за описание на клиентите. Остатъкът от вашите последователи се премества в документ от друга колекция, като за връзка между документите се използва идентификатора на потребителя.

Плюсове

- Запитванията са пригодени за базовата схема на документите, но отклоненията все пак са налични.

Минуси

- Изисква се писане на програмен код за реализация на шаблона.

8. Шаблон “Pre-allocated”

Когато познавате структурата на документа и приложението ви просто трябва да го запълни с данни, моделът за предварително разпределение е правилният избор. На този етап няма голям практически смисъл, тъй като при MongoDB 3.2 се използва двигателя WiredTiger с цел съхранение на данните.

Плюсове

- Опростяване на дизайна, когато структурата на документа е предварително известна.

Недостатъци

- Опростеност срещу производителност.

9. Шаблон Polymorphic

Когато всички документи в една колекция са с подобна, но не идентична структура, наричаме това полиморфен модел. Групирането на документи заедно въз основа на заявките, които искаме да изпълним, помага за подобряване на производителността. Нека да вземем за пример система, която трябва да обработва данните на спортисти. Всеки обект-спортист трябва да получава общите за всички спортисти свойства (полета), но и да съдържа и специфичните за неговия спорт характеристики. Шаблонът е лесно приложим ако сте изучавали обектно-ориентирано програмиране.

Плюсове

- Лесен за изпълнение.
- Запитванията могат да се изпълняват в една колекция.

10. Шаблон “Schema Versioning”

Този шаблон се използва ако често се налагат промени в схемата на данните. Възможно е за една част от документите в една колекция да се приложи една схема, а за други – друга. Този модел позволява предишни и текущи версии на документи да съществуват в една колекция. За целта е необходимо да се вмъкне поле “schema_version” със стойност версията на схемата, която трябва да се използва. Стойността на полето е необходима, за да може програмното осигуряване да се адаптира към полетата на документите (а не да ги проверява дали съществуват или не).

Плюсове

- Контрол на миграцията на схемата.

Недостатъци

- Може да са необходими два индекса за едно и също поле по време на миграцията.

11. Шаблон “Subset”

Шаблонът “Subset” се използва с цел оптимизиране на използването на паметта при работа с големи документи, основната част от данните от които приложението не използва в даден момент. Типичен пример за това са отзивите за всеки продукт, който се продава от даден електронен магазин. Вместо да запишем всички отзиви в масив от JSON обекти при шаблон “Subset” в документа на продукта се пазят само последните N отзива или подбрани N отзива на клиенти. Останалите отзиви се записват в отделен документ. Връзката на документите ще се реализира чрез идентификационния номер на продукта.

Плюсове

- По-кратко време за достъп до диска за най-често използваните данни.

Недостатъци

- Изтеглянето на допълнителни данни изисква допълнителни заявки до базата.

12. Шаблон “Tree”

Когато данните са с йерархична структура и често се търсят, шаблонът "Tree" е много подходящ за прилагане. Например, всеки продукт, който се продава в даден електронен магазин принадлежи на конкретна категория, която от своя страна е част от друга категория и така докато стигнем до върха (root) в тази йерархия. За да може лесно да се проследи йерархията на категориите, те най-често се записват в масив. Ако сте избрали

да разгледате мъжко сако, то в документа, който го описва, трябва да има поле, съдържащо йерархията от категории на които сакото принадлежи, например:

```
{
     _id: <ObjectId>,
     prod_code: "2YQS528U8_252",
    name: "BENETTON BLAZER IN LINEN BLEND",
    price: [
      value: "269,90",
      currency: "BGN"
    ],
    basic_features: [
    ...
    ],
    composition: "Outside 51% Linen 49% Cotton Lining 100% Cotton",
    clothing_care: "Machine washable at a maximum temperature of 30°C.",
    parent_category: "BLAZERS AND JACKETS",
     categories: [
      "BLAZERS AND JACKETS",
      "JACKETS",
      "JACKETS AND COATS",
      "MEN"
    ]
}
```

Фиг.4.6 Пример за използване на шаблон “Tree”

Плюсове

- Повишена производителност чрез избягване на множество операции JOIN.

Недостатъци

- Актуализациите на йерархията на категориите трябва да се управлява от страна на приложението.