Нереляционные базы данных

Лабораторная работа 4

Аналоги JOIN

Первое и самое фундаментальное различие, с которым вам надо свыкнуться, это отсутствие у MongoDB аналога конструкции JOIN. Неизвестно почему именно MongoDB не поддерживает JOIN-синтаксиса, однако точно можно сказать, что JOIN-ы не масштабируемы. Это значит, что когда вы начнёте разделять данные горизонтально, вам всё равно придётся выполнять JOIN-ы на клиенте (которым является сервер приложений). Независимо от причин, факт остаётся фактом: данные реляционны по своей природе, но MongoDB не поддерживает JOIN-ов.

Мы должны делать JOIN-ы вручную, в коде своего приложения. По существу, мы должны делать второй запрос, чтобы найти связанные данные. Создание данных тут не сильно отличается от создания внешних ключей в реляционных базах. Теперь давайте от единорогов (unicorns) перейдём к сотрудникам (employees). Первым делом создадим сотрудника (я явным образом привожу здесь _id, чтобы наши примеры выполнялись как задумано)

```
db.employees.insert({_id: ObjectId("4d85c7039ab0fd70a117d730"), name: 'Leto'})
```

Теперь добавим пару сотрудников и сделаем Leto их менеджером:

```
db.employees.insert({_id: ObjectId("4d85c7039ab0fd70a117d731"), name: 'Duncan', manager: ObjectId("4d85c7039ab0fd70a117d730")});
db.employees.insert({_id: ObjectId("4d85c7039ab0fd70a117d732"), name: 'Moneo', manager: ObjectId("4d85c7039ab0fd70a117d730")});
```

(стоит повторить, что _id может быть любым уникальным значением. Поскольку в жизни вы скорее всего станете использовать ObjectId, мы также здесь используем его.)

Чтобы найти всех сотрудников, принадлежащих Leto, выполним просто:

```
db.employees.find({manager: ObjectId("4d85c7039ab0fd70a117d730")})
```

Никакой магии. В худших случаях отсутствие JOIN-ов чаще всего потребует дополнительного запроса (как правило индексированного).

Массивы и вложенные документы

Но тот факт, что у MongoDB нет JOIN-ов ещё не означает, что у неё не припасено пару козырей в рукаве. Помните, как мы вкратце поведали ранее о поддержке в MongoDB массивов, как объектов первого класса? Оказывается,

что она чертовски удобна, когда требуется смоделировать отношения «один-ко-многим» или «многие-ко-многим». Например, если у сотрудника есть несколько менеджеров, мы просто можем сохранить их в виде массива:

```
db.employees.insert({_id: ObjectId("4d85c7039ab0fd70a117d733"), name: 'Siona', manager: [ObjectId("4d85c7039ab0fd70a117d730"), ObjectId("4d85c7039ab0fd70a117d732")] })
```

А самое интересное, что в одних документах manager можно сделать скалярным значением, а в других — массивом. А наш предыдущий запрос find сработает в обоих случаях:

```
db.employees.find({manager: ObjectId("4d85c7039ab0fd70a117d730")})
```

Вскоре вы убедитесь, что массивы значений намного удобнее в использовании, нежели таблицы связи «многие-ко-многим».

Кроме массивов MongoDB также поддерживает вложенные документы. Попробуйте вставить документ со вложенным документом, например:

```
db.employees.insert({_id: ObjectId("4d85c7039ab0fd70a117d734"), name: 'Ghanima', family: {mother: 'Chani', father: 'Paul', brother: ObjectId("4d85c7039ab0fd70a117d730")}})
```

Вложенные документы можно запрашивать с помощью точечной нотации:

```
db.employees.find({'family.mother': 'Chani'})
```

Мы кратко обсудим, где могут использоваться вложенные документы, и как их следует применять.

DBRef

MongoDB поддерживает понятие под названием DBRef, которое является соглашением, принятым во многих драйверах. Когда драйвер видит DBRef, он может автоматически получить связанный документ. DBRef включает в себя коллекцию и _id документа, на который он ссылается. Это означает следующее — документы из одной и той же коллекции могут ссылаться на другие документы из различных коллекций. То есть документ 1 может ссылаться на документ из коллекции managers, в то же время документ 2 может ссылаться на документ из коллекции employees.

<поле объекта> : { "$ref" : <value>, "$id" : <value>, "$db" : <value> }

`$ref` — имя коллекции.
`$id` — id объекта.

`$db – имя базы данных (опционально).`

Денормализация

Ещё одна альтернатива использованию JOIN-ов — денормализация. Исторически денормализация использовалась для оптимизации производительности, или когда с данных (например, журнала аудита) необходимо было иметь возможность делать снимок. Однако с быстрым ростом NoSQL решений, многие из которых лишены JOIN-ов, денормализация стала в порядке вещей. Это не означает, что нужно дублировать всё подряд в любых документах. Можно остерегаться дублирования данных, а можно соответствующим образом продумать архитектуру своей базы.

К примеру, мы разрабатываем форум. Традиционный путь ассоциировать пользователя с его постом — это колонка userid в таблице posts. с такой моделью нельзя отобразить список постов без дополнительного извлечения данных (JOIN) из таблицы пользователей. Возможное решение — хранить имя пользователя (name) вместе с userid для каждого поста. Можно также вставлять небольшой встроенный документ, например,user: {id: ObjectId('Something'), name: 'Leto'}. Да, если позволить пользователям изменять своё имя, нам придётся обновлять каждый документ (пост) — это один лишний запрос.

Не всем легко приспособиться к такому подходу. Во многих случаях даже не имеет смысла этого делать. Всё же не бойтесь экспериментировать с таким подходом. Иногда это бывает полезным — чуть ли не единственным правильным — решением.

Что выбрать?

Также полезной стратегией в случаях отношения «один-ко-многим» или «многие-ко-многим» является массив идентификаторов. Бытует мнение, что DBRef используется не так часто, но конечно вы можете поэкспериментировать с ним. Обычно начинающие разработчики не уверены что подойдёт им лучше — вложенные документы или DBRef.

Во-первых, следует помнить, что одиночный документ ограничен в размере до 4 мегабайт. Факт ограничения (пусть и такого щедрого) размера документа даёт понимание о том, как их следует использовать. Теперь понятно, что большинство разработчиков склоняются к использованию заданных вручную ссылок. Вложенные документы используются часто, но для небольших объёмов данных, если их желательно всегда извлекать вместе с родительским

документом. Примером из жизни может быть документ accounts, сохраняемый с каждым пользователем, например:

db.users.insert({name: 'leto', email: 'leto@dune.gov', account: {allowed_gholas: 5, spice_ration: 10}})

Это не означает, что можно недооценивать мощь вложенных документов, либо отбрасывать их, как мелкую, второстепенную утилиту. Намного проще живётся, когда структура ваших данных напрямую отображает структуру ваших объектов. Особенно ценным является то, что MongoDB позволяет запрашивать и индексировать поля вложенных документов.

Мало или много коллекций

Учитывая то, что коллекции не привязывают нас к конкретной схеме, вполне возможно обойтись одной коллекцией, имеющей документы разной структуры. Построенные на MongoDB системы, с которыми мне приходилось сталкиваться, как правило, были похожи на реляционные базы данных. Другими словами, то, что являлось бы таблицей в реляционной базе данных, скорее всего реализуется, как коллекция в MongoDB (таблицы-связки «многие-ко-многим» являются важным исключением).

Дело принимает интересный оборот, если воспользоваться вложенными документами. Пример, который первым делом приходит на ум, это блог. Допустим, есть коллекция postsи коллекция comments, и каждый пост должен иметь вложенный массив комментариев. Если оставить в стороне ограничение 4Мб («Гамлет» на английском едва дотягивает до 200 килобайт, насколько же должен быть популярным ваш блог?), большинство разработчиков предпочитают разделять сущности. Так понятнее и яснее.

Нет какого бы то ни было строгого правила (ну, кроме 4MB). Поэкспериментируйте с различными подходами, и вам станет ясно, что будет правильнее, а что — нет.

Выводы к лабораторной работе

Целью этой лабораторной работы было представить некоторые полезные рекомендации для моделирования данных в MongoDB. Если угодно, стартовую точку. Моделирование в документ-ориентированных системах отличается от такового в реляционных, но не так уж сильно. Здесь намного больше гибкости, но есть одно ограничение, хотя для разработки новой системы это подходит, как правило, неплохо. Не выходит только у тех, кто не пробует.

Требования к сдаче лабораторной работы

1. Отчет к лабораторной работе, содержащий пошаговое исполнение заданий и промежуточные результаты

Задания к лабораторной работе

1. Создайте коллекцию из 5-10 документов, добавьте связь один-ко-многим. Проведите несколько запросов, чтобы продемонстрировать работу.
2. Создайте коллекцию из 5-10 документов, добавьте связь многий-ко-многим с помощью массива. Проведите несколько запросов, чтобы продемонстрировать работу.
3. Создайте коллекцию из 3-5 документов, добавьте связь один-к-одному с помощью вложенных документов. Проведите несколько запросов, чтобы продемонстрировать работу.
4. Создайте две коллекции из 3-5 документов, добавьте связь один-к-одному с помощью DBRef. Проведите несколько запросов, чтобы продемонстрировать работу.
5. Приведите собственный пример денормализации, сделайте вывод о ее достоинствах и недостатках.
6. Сделайте выводы о применимости различных замен JOIN.