Web-разработка на PHP-технологиях

Курс лекций и семинаров для студентов,
желающих научиться основам Web-разработки на PHP

Осень-Зима 2014

Лекция №1 0

NoSQL Базы Данных

Автор: Дмитрий Левин, Senior PHP Developer, Sibers

Sibers®

NoSQL Базы Данных

Характеристики NoSQL баз данных:

1. Не используется SQL

Имеется в виду ANSI SQL DML, так как многие базы пытаются использовать query languages похожие на общеизвестный любимый синтаксис, но полностью его реализовать не удалось никому и вряд ли удастся.

Список всех no-sql баз данных: http://nosql-database.org/

NoSQL Базы Данных

2. Неструктурированные (schemaless)

В NoSQL базах в отличие от реляционных структура данных не регламентирована (или слабо типизированна, если проводить аналогии с языками прогаммирования) — в отдельной строке или документе можно добавить произвольное поле без предварительного декларативного изменения структуры всей таблицы.

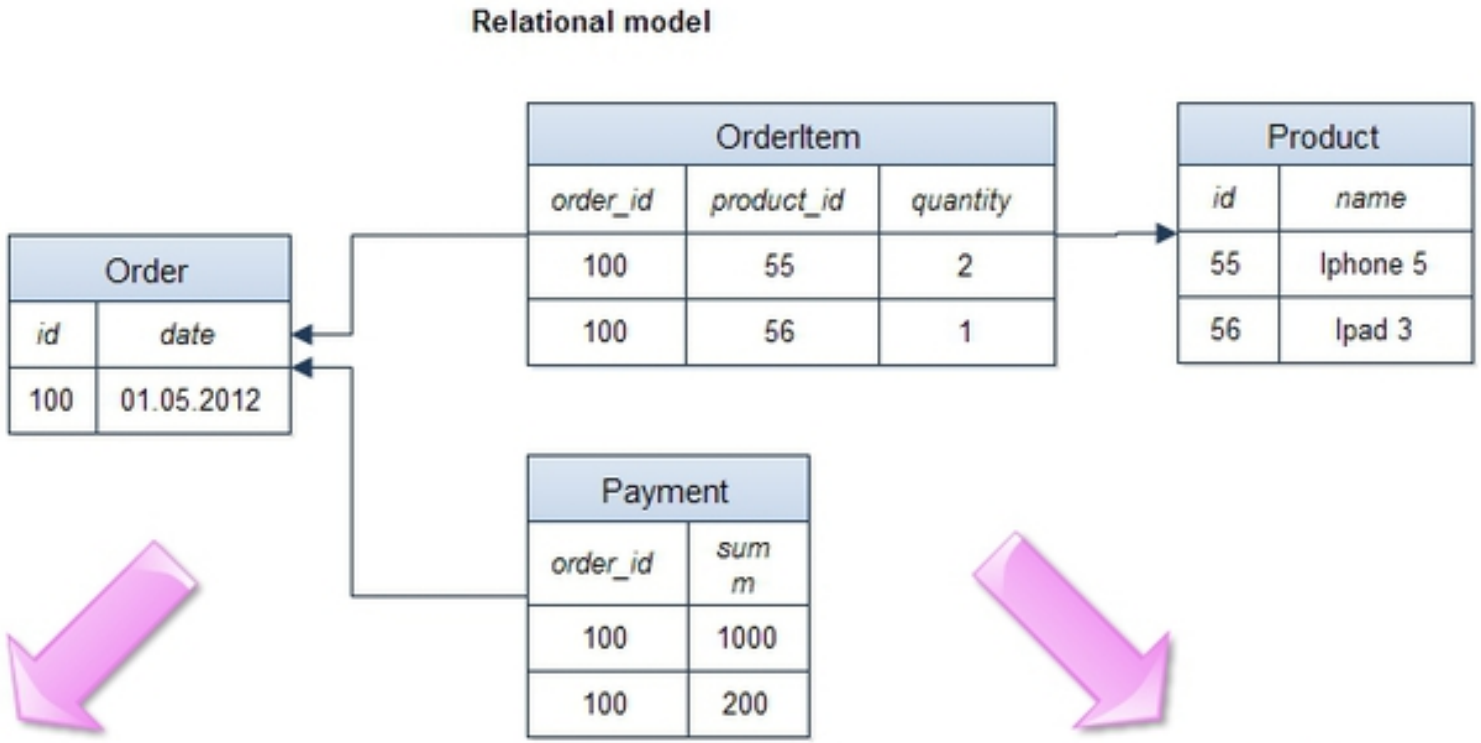
Таким образом, если появляется необходимость поменять модель данных, то единственное достаточное действие — отразить изменение в коде приложения.

NoSQL Базы Данных

3. Представление данных в виде агрегатов (aggregates).

В отличие от реляционной модели, которая сохраняет логическую бизнес-сущность приложения в различные физические таблицы в целях нормализации, NoSQL хранилища оперируют с этими сущностями как с целостными объектами.

NoSQL Базы Данных



NoSQL Базы Данных

Aggregate model 1

```
// Order document
{
"id": 100,
"customer_id": 1000,
"date": 01.05.2012,
"order_items": [
        {
        "product_id": 55,
        "product_name": Iphone5,
        "quantity": 2
        },
        {
        "product_id": 56,
        "product_name": Ipad3
        "quantity": 1
        }
],
"payments":[
        {
        "sum": 1000,
        "date": 03.05.2012
        }
]
}
// Product document here
{...}
```

Aggregate model 2

```
// Order document
{
"id": 100,
"customer_id": 1000,
"date": 01.05.2012,
"order_items": [
        {
        "product_id": 55,
        "product_name": Iphone5,
        "quantity": 2
        },
        {
        "product_id": 56,
        "product_name": Ipad3
        "quantity": 1
        }
]
}
// Payment document
{
"order_id": 100,
"sum": 1000,
"date": 03.05.2012
}
// Product document here
{...}
```

Sibers®

NoSQL Базы Данных

Плюсы и минусы обоих подходов:

Нормализация данных	Данные в виде агрегатов
• Целостность информации при обновлении (меняем запись в одной таблице, а не в нескольких) • Ориентированность на широкий спектр запросов к данных	• Оптимизация только под определенный вид запросов • Сложности при обновлении денормализованных данных
• Неэффективна в распределенной среде • Низкая скорость чтения при использовании объединений (joins) • Несоответствие объектной модели приложения физической структуре данных (impedance mismatch, решается с помощью Hibernate etc.)	• Лучший способ добиться большой скорости на чтение в распределенной среде • Возможность хранить физически объекты в том виде, в каком с ними работает приложение (легче кодировать и меньше ошибок при преобразовании) • Родная (native) поддержка атомарности на уровне записей

NoSQL Базы Данных

4. Слабые ACID свойства

Atomicity — Атомарность
Consistency — Согласованность
Isolation — Изолированность
Durability — Надежность

4.1 Atomicity — Атомарность

Атомарность гарантирует, что никакая транзакция не будет зафиксирована в системе частично. Будут либо выполнены все её подоперации, либо не выполнено ни одной.

NoSQL Базы Данных

4. Слабые ACID свойства.

4.2 Consistency — Согласованность

Транзакция достигающая своего нормального завершения (EOT – end of transaction, завершение транзакции) и, тем самым, фиксирующая свои результаты, сохраняет согласованность базы данных.

Другими словами, каждая успешная транзакция по определению фиксирует только допустимые результаты.

NoSQL Базы Данных

4. Слабые ACID свойства.

4.3 Isolation — Изолированность

Во время выполнения транзакции параллельные транзакции не должны оказывать влияние на её результат

NoSQL Базы Данных

4. Слабые ACID свойства.

4.4 Durability — Надежность

Независимо от проблем на нижних уровнях (к примеру, обесточивание системы или сбои в оборудовании) изменения, сделанные успешно завершённой транзакцией, должны остаться сохранёнными после возвращения системы в работу.

Другими словами, если пользователь получил подтверждение от системы, что транзакция выполнена, он может быть уверен, что сделанные им изменения не будут отменены из-за какого-либо сбоя.

NoSQL Базы Данных

4. Слабые ACID свойства.

NoSQL Базы Данных

4. Слабые ACID свойства.

На самом деле слабые ACID свойства не означают, что их нет вообще. В большинстве случаев приложение, работающее с реляционной базой данных, использует транзакцию для изменения логически связанных объектов (заказ — позиции заказа), что необходимо, так как это разные таблицы.

При правильном проектировании модели данных в NoSQL базе (агрегат представляет из себя заказ вместе с перечнем пунктов заказа) можно добиться такого же самого уровня изоляции при изменении одной записи, что и в реляционной базе данных.

NoSQL Базы Данных

5. Распределенные системы, без совместно используемых ресурсов (share nothing).

С лавинообразным ростом информации в мире и необходимости ее обрабатывать за разумное время встала проблема вертикальной масштабируемости — рост скорости процессора остановился на 3.5 Ггц, скорость чтения с диска также растет тихими темпами, плюс цена мощного сервера всегда больше суммарной цены нескольких простых серверов.

В этой ситуации обычные реляционные базы, даже кластеризованные на массиве дисков, не способны решить проблему скорости, масштабируемости и пропускной способности.

Единственный выход из ситуации — горизонтальное масштабирование, когда несколько независимых серверов соединяются быстрой сетью и каждый владеет/обрабатывает только часть данных и/или только часть запросов на чтение-обновление. В такой архитектуре для повышения мощности хранилища (емкости, времени отклика, пропускной способности) необходимо лишь добавить новый сервер в кластер — и всё.

NoSQL Базы Данных

5. Распределенные системы, без совместно используемых ресурсов (share nothing).

Процедурами шардинга, репликации, обеспечением отказоустойчивости (результат будет получен даже если один или несколько серверов перестали отвечать), перераспределения данных в случае добавления ноды занимается сама NoSQL база

NoSQL Базы Данных

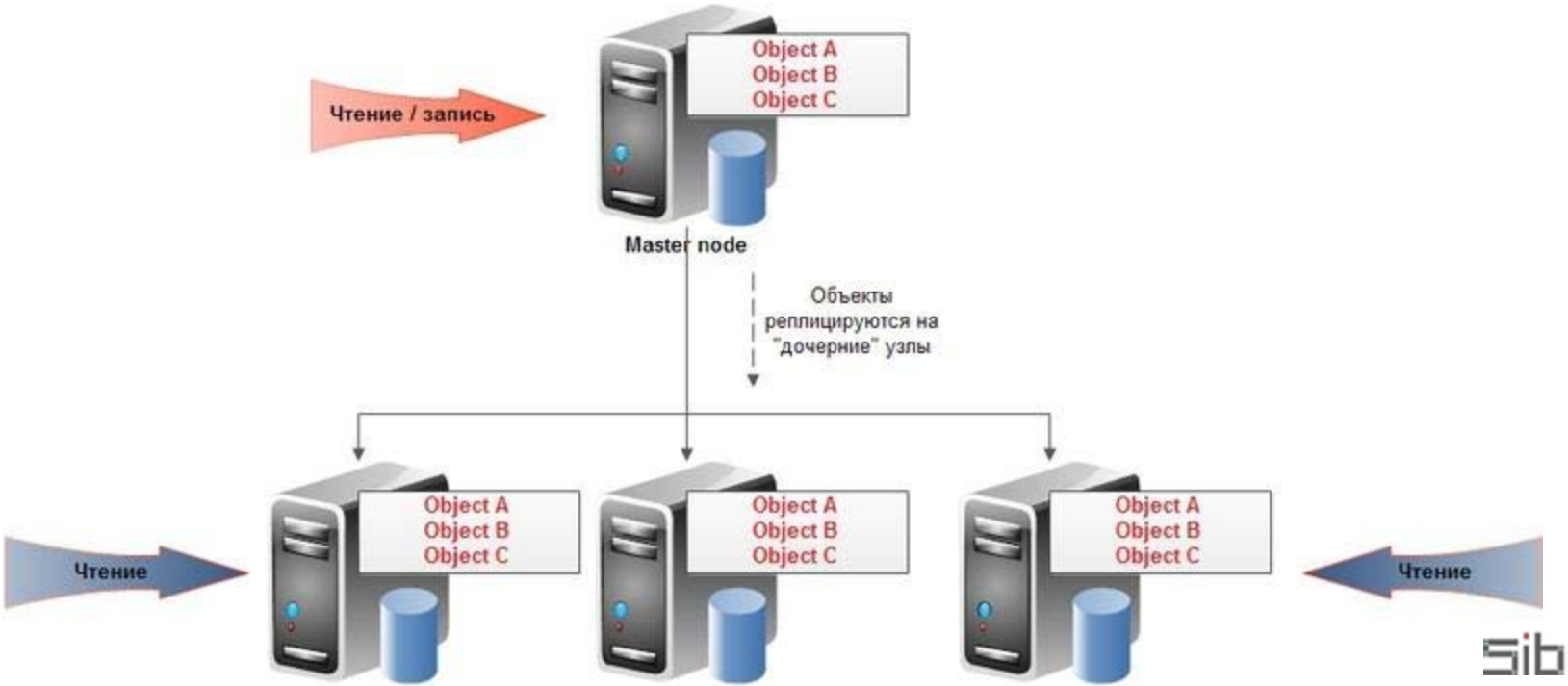
5. Распределенные системы, без совместно используемых ресурсов (share nothing).

Репликация — копирование данных на другие узлы при обновлении. Позволяет как добиться большей масштабируемости, так и повысить доступность и сохранность данных.

Принято подразделять на два вида:
master-slave и peer-to-peer

NoSQL Базы Данных

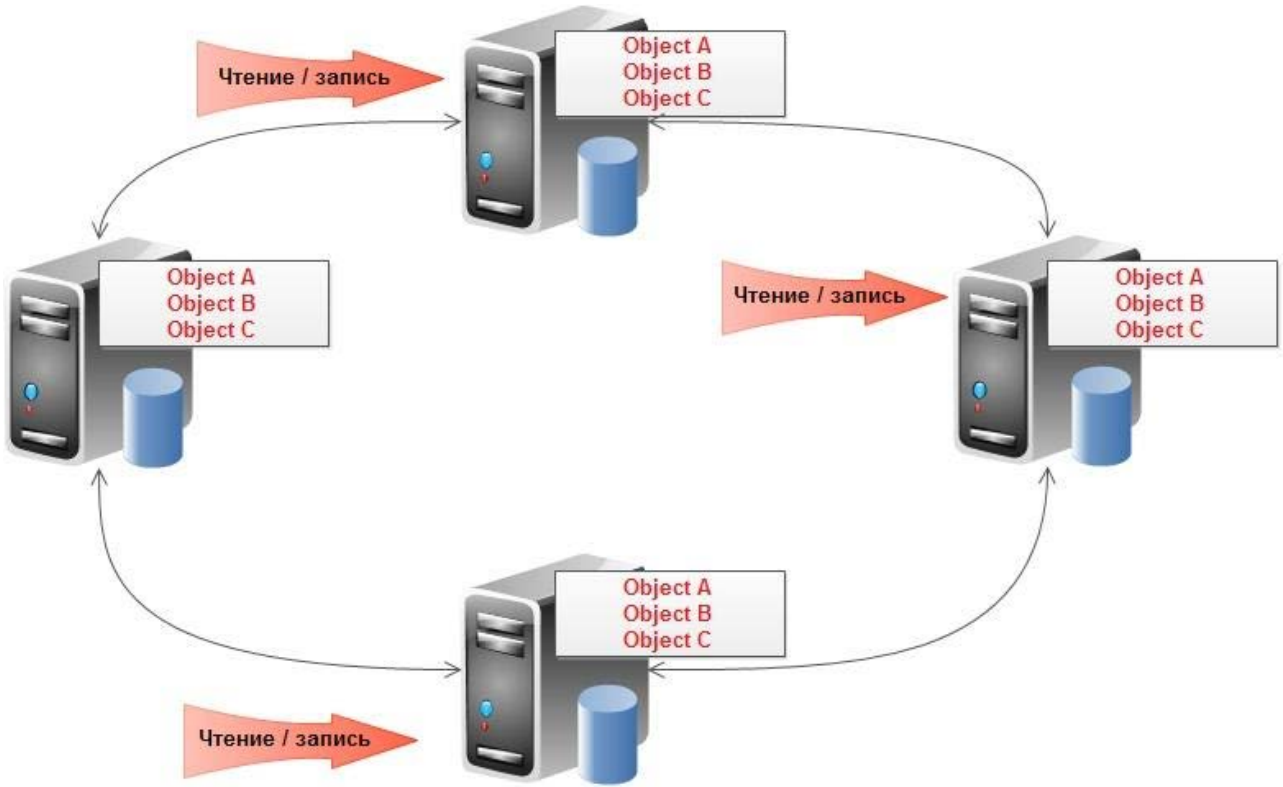
master-slave



Sibers®

NoSQL Базы Данных

peer-to-peer



NoSQL Базы Данных

5. Распределенные системы, без совместно используемых ресурсов (share nothing).

Первый тип предполагает хорошую масштабируемость на чтение (может происходить с любого узла), но немасштабируемую запись (только в мастер узел). Также есть тонкости с обеспечением постоянной доступности (в случае падения мастера либо вручную, либо автоматически на его место назначается один из оставшихся узлов).

Для второго типа репликации предполагается, что все узлы равны и могут обслуживать как запросы на чтение, так и на запись

NoSQL Базы Данных

Шардинг — разделение данных по узлам

NoSQL Базы Данных

5. Распределенные системы, без совместно используемых ресурсов (share nothing).

Шардинг часто использовался как «костыль» к реляционным базам данных в целях увеличения скорости и пропускной способности: пользовательское приложение партицировало данные по нескольким независимым базам данных и при запросе соответствующих данных пользователем обращалось к конкретной базе.

В NoSQL базах данных шардинг, как и репликация, производятся автоматически самой базой и пользовательское приложение обособленно от этих сложных механизмов.

NoSQL Базы Данных

6. NoSQL базы в основном оупенсорсные и созданы в 21 столетии

NoSQL движение набирает популярность гигантскими темпами. Однако это не означает, что реляционные базы данных становятся рудиментом или чем-то архаичным.

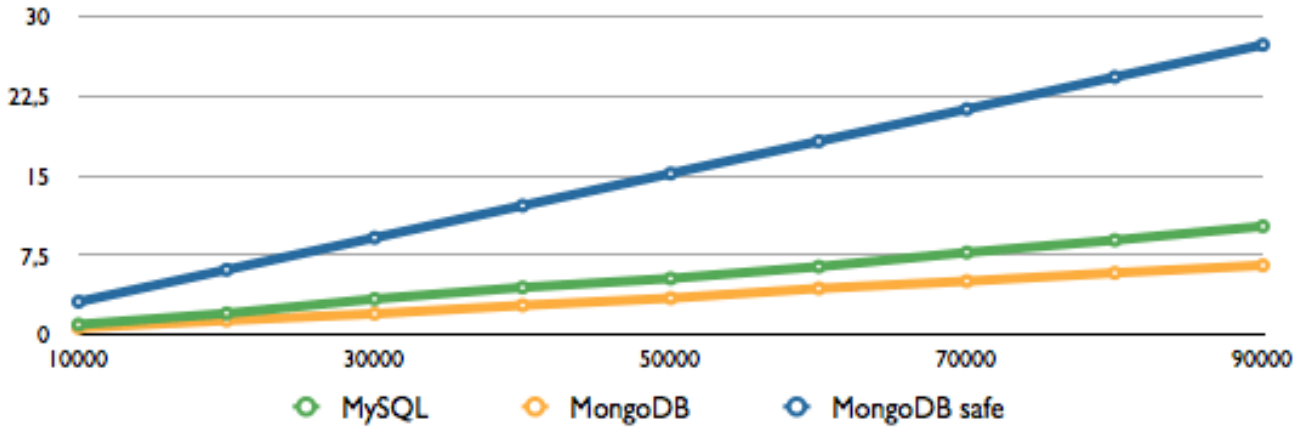
Скорее всего они будут использоваться и использоваться по-прежнему активно, но все больше в симбиозе с ними будут выступать NoSQL базы. Теперь нет монополизма реляционных баз данных, как безальтернативного источника данных.

Всё чаще архитекторы выбирают хранилище исходя из природы самих данных и того, как мы ими хотим манипулировать, какие объемы информации ожидаются. И поэтому все становится только интереснее.

MySQL vs NoSQL

Сравнительные тесты MySQL и MongoDB

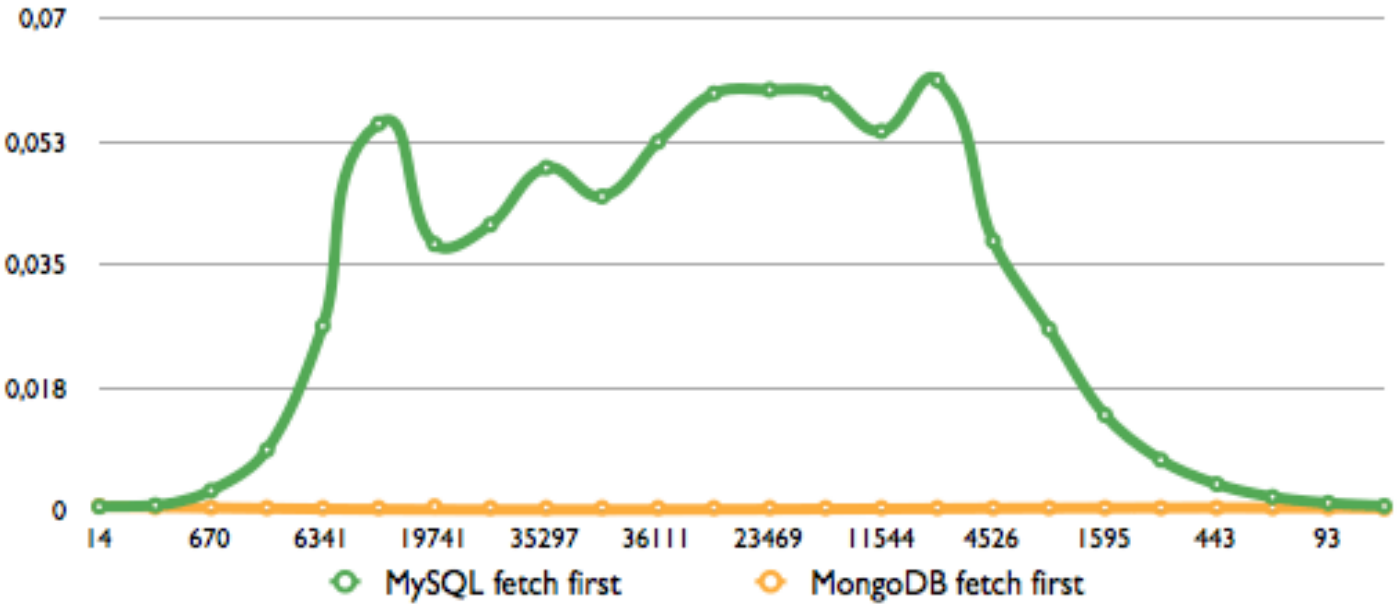
1. Вставка данных (90к) - MongoDB опережает MySQL на 40%



Sibers®

MySQL vs NoSQL

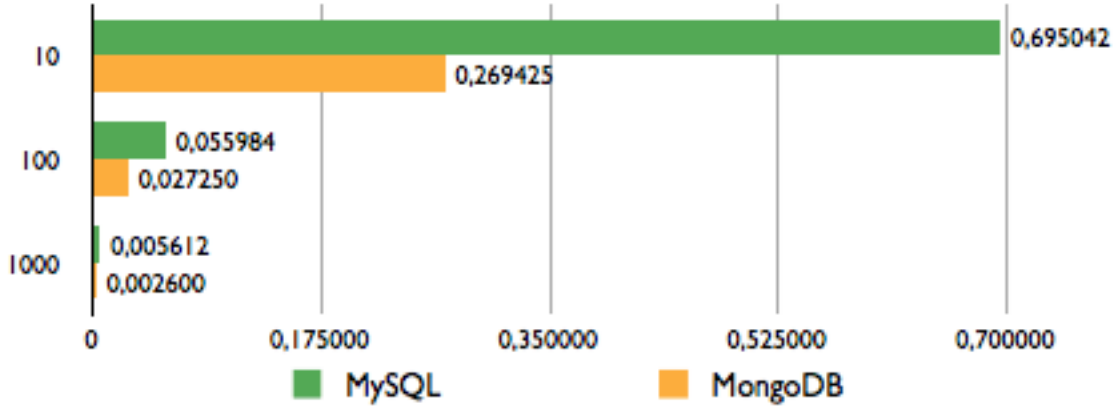
2. Выборка данных



MySQL vs NoSQL

3. Сравнительные результаты обновления каждой 10ой/100ой/1000ой строки

	MySQL	MongoDB
10	0,695042	0,269425
100	0,055984	0,027250
1000	0,005612	0,002600

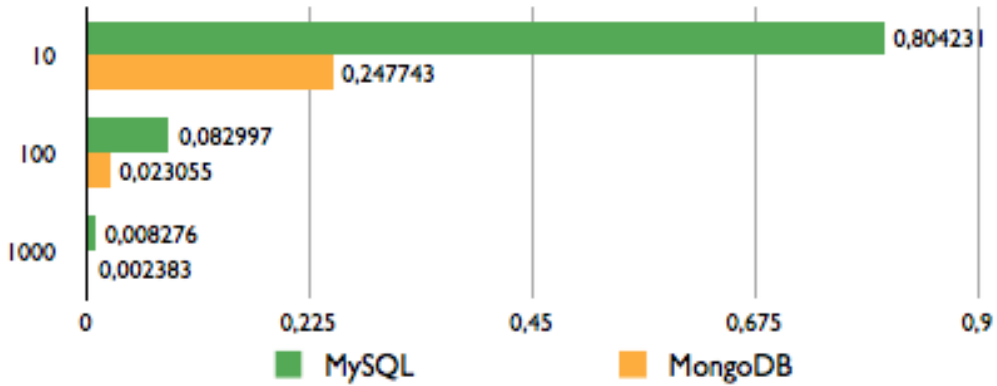


Sibers®

MySQL vs NoSQL

4. Ниже приведены результаты удаления каждой 10ой/100ой/1000ой строки

	MySQL	MongoDB
10	0,804231	0,247743
100	0,082997	0,023055
1000	0,008276	0,002383



MongoDB

nosql document-oriented database

Sibers®

MongoDB

Как MongoDB хранит данные?

MongoDB — это документарная база данных. Вместо хранения данных в таблицах, состоящих из отдельных строк, как в реляционных базах, MongoDB сохраняет данные в **коллекциях**, состоящих из **документов**.

Документ — это большой JSON объект без заранее определенного формата и схемы.

MongoDB

Пример документа

```
{title: 'Babylon 5',
 seasons: [
  {season_number: '1',
    episodes: [
     {ordinal_within_season: '1',
      title: 'Midnight on the Firing Line'
      reviews: [{...}],
      cast_members: [{...}]
     }
    ]
  }
 ]
}
```

MongoDB

У сериала есть название и массив сезонов. Каждый сезон — объект с метаданными и массивом эпизодов. В свою очередь каждый эпизод имеет метаданные и массивы отзывов и актеров. Похоже на огромную фрактальную структуру данных.

MongoDB

Все данные нужные для сериала хранятся одним документом, так что можно очень быстро получить всю информацию сразу, даже если документ очень большой.

Есть сериал, называемый “General Hospital”, который насчитывает уже 12000 эпизодов в течение 50+ сезонов.

MySQL работает около минуты, чтобы получить денормализованные данные для 12000 эпизодов, в то время как извлечение документа по ID в MongoDB занимает доли секунды.

MongoDB

MongoDB и социальные данные?

Когда вы попадаете в социальную сеть, есть только одна важная часть страницы: ваша лента активности.

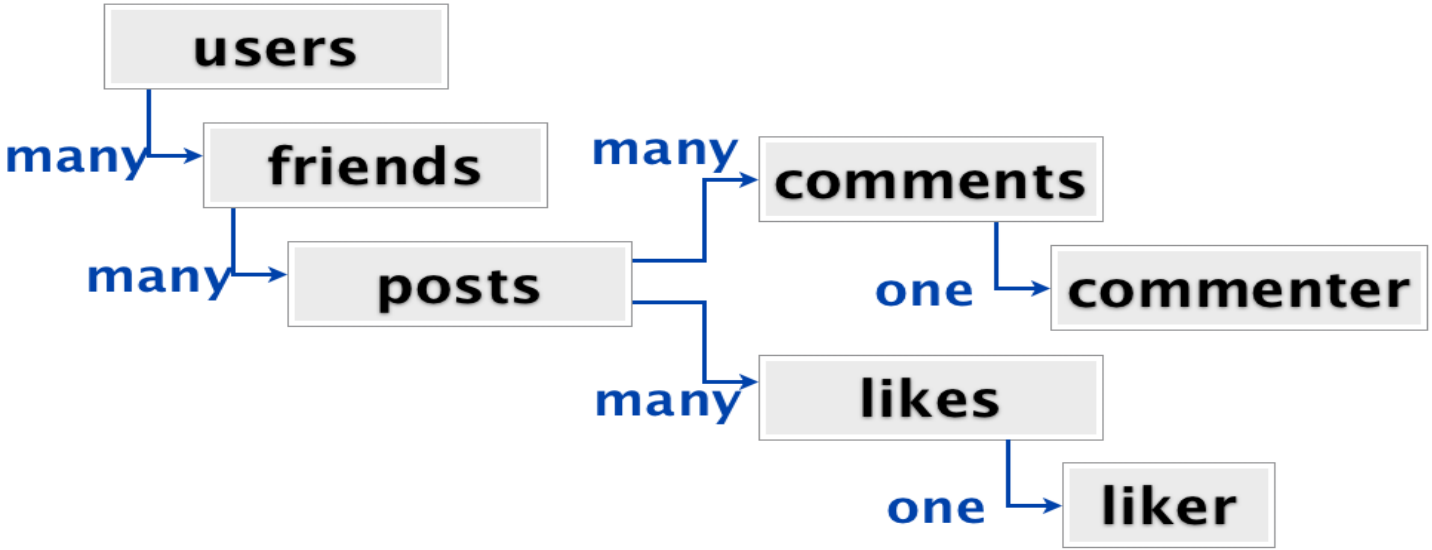
Запрос ленты активности получает все посты от ваших друзей, отсортированные по убыванию даты. Каждый пост содержит вложения, такие как фотографии, лайки, репосты и комментарии.

Вложенная структура ленты активности выглядит очень похоже на сериалы.

MongoDB

Пользователи имеют друзей, друзья имеют посты, посты имеют комментарии и лайки, каждый из которых имеет связан с одним комментатором или лайкером.

С точки зрения связей это не сильно сложнее структуры телесериалов. И как в случае сериалов мы хотим получить всю структуру разом как только пользователь войдет в соцсеть. В реляционной СУБД это было бы соединение семи таблиц, чтобы вытащить все данные.



MongoDB

Дублирование данных

Мы можем по-разному смоделировать это в MongoDB.

Самый простой способ — дублирование данных. Вся информация о пользователе копируется сохраняется в лайке к первому посту, а затем отдельная копия сохраняется в комментарии ко второму посту.

Преимущество в том, что все данные присутствует везде, где вам это нужно, и вы все еще можете вытащить весь поток активности еще в одном документе.

MongoDB

Дублирование данных

Примерно так выглядит плотностью денормализованная лента активности:

```
{
 name: Joe,
 url: '...',
 stream:
 [{
   user: {name: Jane, url: '...'},
   title: 'today',
   body: 'go fly a kite',
   likes: [
    {user: {name: Lu, url: '...'}},
    {user: {name: Joe, url: '...'}}
   ],
 }]
}
```

MongoDB

Существует другой подход к решению проблемы в MongoDB, который будет знаком тем, кто имеет опыт работы с реляционными СУБД. Вместо дублирования данных вы можем сохранять ссылки на на пользователей в ленте активности.

При этом подходе вместо встраивания данных там, где они нужны, вы даёте каждому пользователю ID. После этого вместо встраивания данных пользователя вы сохраняете только ссылки на пользователей

MongoDB

Дублирование данных

```
{id: 1,
 name: Joe,
 url: '...',
 stream:
 [{
   user: 2,
   title: 'today',
   body: 'go fly a kite',
   likes: [
    {user: 3},
    {user: 1},
   ],
 }]
}
```

MongoDB

Это исключает нашу проблему дублирования. При изменении данных пользователей, есть только один документ, который нужно изменить. Тем не менее, мы создали новую проблему для себя.

Потому что мы больше не можем построить ленту активности из одного документа. Это менее эффективное и более сложное решение.

Построение ленты активности в настоящее время требует, чтобы мы:

1) получили документ ленты активности, а затем
2) получили все документы пользователей, чтобы заполнить имена и аватары.

MongoDB

Чего не хватает MongoDB — это операции соединения как в SQL, которая позволяет написать один запрос, объединяющий вместе ленту активности и всех пользователей, на которых есть ссылки из ленты. В конечном итоге приходится вручную делать джоины в коде приложения.

Единственное, что удобно хранить в MongoDB — произвольные JSON фрагменты.

«Произвольные» в этом контексте означает, что вам абсолютно всё равно что внутри JSON.

Каждый документ — набор байт, и вы не делаете никаких предположений о том, что внутри.

MongoDB

Найдите ценность

Когда вы выбираете хранилище данных, самое главное понять где в данных и связях находится ценность для клиента.

Если вы пока ещё не знаете, то нужно выбирать то, что не загонит вас в угол. Запихивание произвольных JSON данных в базу выглядит гибким решением, но настоящая гибкость заключается в простом добавлении функций для бизнеса.

Делайте ценные вещи простыми.

Заключение:

В десятой, заключительной, лекции мы расссмотрели характиристики NoSQL баз данных, сравнили NoSQL и MySQL, познакомились с NoSQL базой данных MongoDB, её преимуществами и недостатками

Спасибо за внимание!

На нашем сайте также доступны

Материалы курса лекций по разработке под iOS:

http://www.sibers.ru/career/student-opportunities/ios-lectures-materials