**NO SQL. Основные характеристики**

NoSQL системы весьма разнообразны и решения, которые они используют, не ограничены никакими формальными рамками. Тем не менее, попытаемся перечислить некоторые характеристики, которые дают основания хранилищам причислять себя к системам типа NoSQL.

**Языки запросов в NoSQL системах**

* **Декларативные (иногда SQL)**
* **Низкоуровневые (как правило)**

Одно из самых больших заблуждений состоит в том, что наиболее общей характеристикой всех этих систем является отказ от использования языка SQL. Это утверждение не соответствует действительности, так как многие NoSQL системы используют SQL-подобные языки запросов. Хотя надо признать, что в списке NoSQL есть системы, которые, действительно отказались от использования декларативных запросов и формулируют запросы на языках низкого уровня.

**Основные модели данных**

* **Ключ-значение**
* **Документные**
* **Колоночные**
* **Граф-ориентированные**

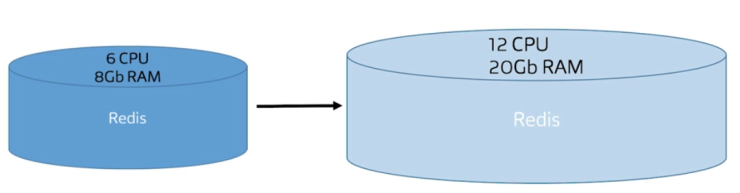
NoSQL системы используют, как правило, простые и, предназначенные для широкого круга задач. Современные NoSQL модели данных принято делить на четыре категории, а именно: хранилища типа ключ-значение, документные хранилища, колоночные хранилища и граф-ориентированные. На самом деле моделей значительно больше, однако остальные не так широко распространены. Кроме того, последнее время многие системы позволяют использовать несколько моделей. В этом случае их называют мультимодальными или гибридными.

### Горизонтальное масштабирование VS вертикальное масштабирование

* **Вертикальное масштабирование** **(scaling up) - увеличение количества доступных ресурсов за счет увеличения мощности применяемых серверов.**
* **Горизонтальное масштабирование**(**scaling out) - увеличение количества объединенных в кластер серверов при нехватке CPU, памяти или дискового пространства.**

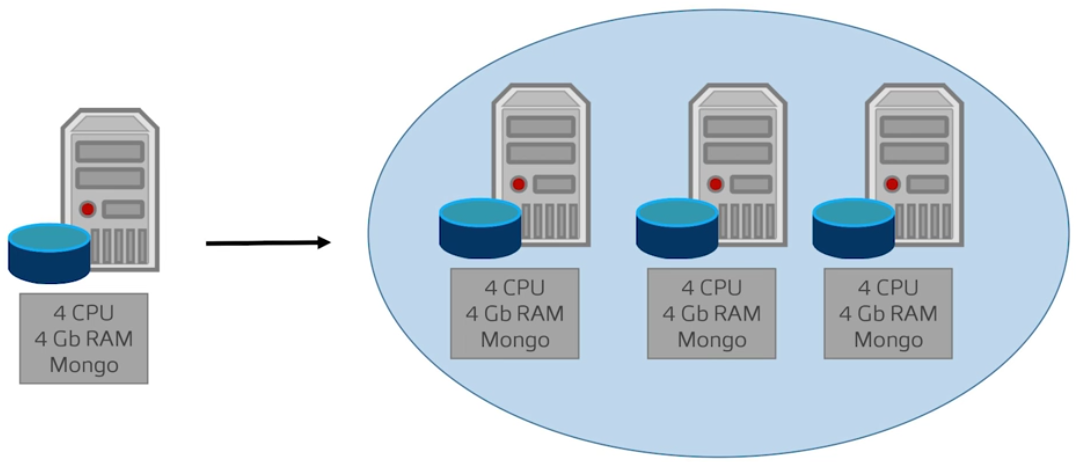
Для NoSQL систем характерно наличие возможностей для масштабирования как вертикального, так и горизонтального.

**Вертикальное масштабирование**



**Вертикальное масштабирование** (scaling up) означает увеличение количества доступных ресурсов за счет увеличения мощности применяемых серверов.

**Горизонтальное масштабирование**



Горизонтальное масштабирование означает разбиение системы на более мелкие структурные компоненты и разнесение их по отдельным серверам и увеличение количества серверов, параллельно выполняющих одну и ту же операцию.

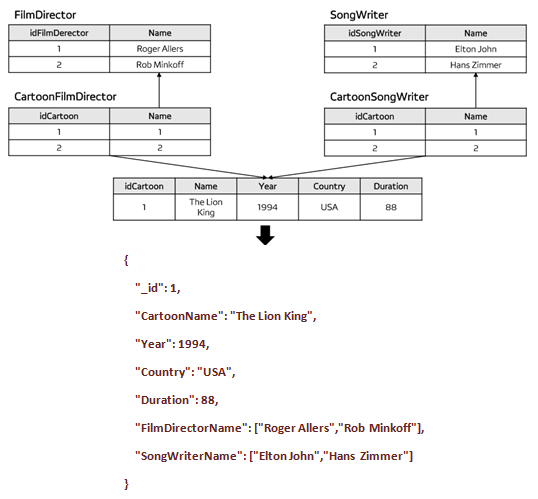
Горизонтальная масштабируемость в этом контексте означает возможность не изменять мощности используемых серверов (что характерно для вертикального масштабирования), а добавлять к системе новые узлы для увеличения общей производительности. Этот способ масштабирования может потребовать внесения изменений в программное обеспечение на стороне приложения или СУБД. Большинство NoSQL систем изначально проектируется с расчетом на горизонтальное масштабирование и решает эти проблемы на стороне СУБД.

**Сравнение хранения данных в NoSQL и SQL системах**

* **NoSQL – хранят только те данные, что есть на самом деле**
* **Реляционные – резервируют жесткие структуры даже для тех данных, которых нет в наличии**

В базах данных NoSQL хранение данных происходит значительно проще, чем в традиционных. Сторонники неструктурированных баз данных подчеркивают их свободу и гибкость. Хранилища типа "ключ-значение" позволяют сохранять и выбирать данные по ключу и не задумываться о содержании данных. Документные хранилища данных по существу делают то же самое, но при этом позволяют манипулировать с фрагментами слабоструктурированных данных. Графовые хранилища свободно добавляют новые узлы и ребра в граф, а также новые свойства к узлам и ребрам. Колоночные хранилища также позволяют легко добавлять новые поля к отдельным записям и удалять старые, не вынуждая при этом проводить глобальную перестройку всей базы. Помимо обеспечения удобных изменений, неструктурированные базы данных облегчают обработку нeoднородных данных, т.е. данных, в которых все записи имеют разные наборы полей. В базах с традиционной организаций схема вынуждает хранить все записи в одинаковом формате. Это может оказаться неудобным, если в разных строках хранятся разные данные. В этом случае множество неиспользуемых полей оказываются заполненными неопределенными или заданными по умолчанию значениями. Неструктурированные базы данных позволяют избежать этого, позволяя каждой записи хранить все, что требуется, - не больше и ни меньше.

**Представление данных в виде агрегатов (aggregates)**

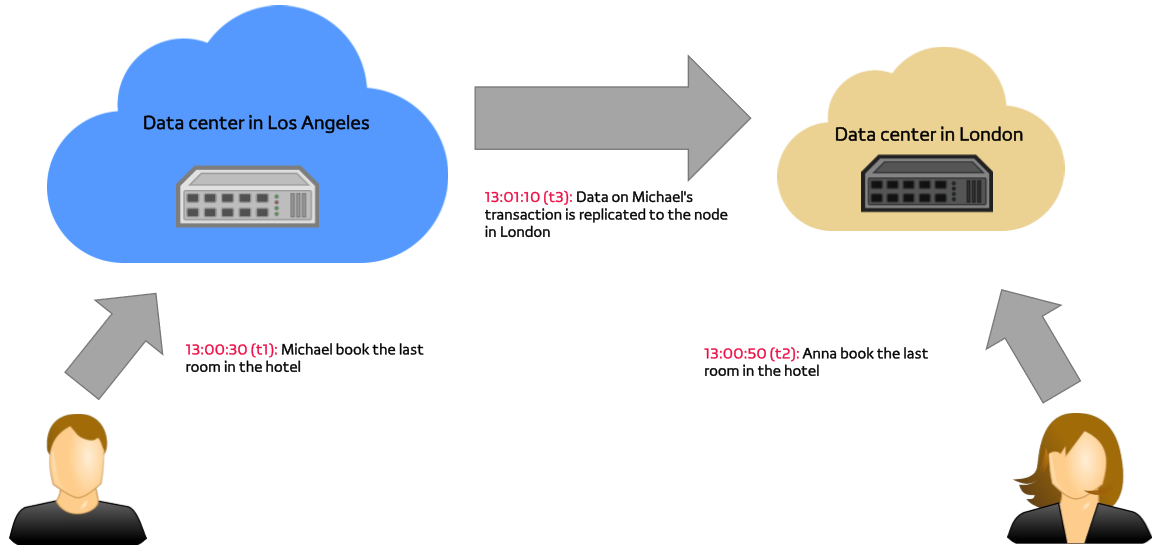


В отличие от реляционной модели, которая хранит сущности в нормализованном виде в различных таблицах, большинство NoSQL хранилищ оперируют с этими сущностями как с целостными объектами и хранят их в денормализованном виде. В примере на слайде продемонстрировано реляционное представление для базы “мультфильмы-режиссеры-композиторы” и соответствующий ему агрегат в формате JSON.

Обратите внимание, что в отличие от реляционной модели, где данные “размазаны” по пяти таблицам в агрегате NoSQL данные о мультфильме объединяются со списком режиссеров и композиторов в один логический объект. Такая денормализация нужна, чтобы не запрашивать имена режиссеров и композиторов при извлечении фильма. Основной смысл такого объединения – минимизация количества соединений между различными объектами. Этим демонстрируется главное правило проектирования структуры данных во многих NoSQL базах — они должны подчиняться требованиям приложения и быть максимально оптимизированы под наиболее частые запросы. Если режиссеры регулярно извлекаются вместе с фильмом — имеет смысл их включать в общий объект, если же большинство запросов работает только с режиссерами — значит, лучше их вынести в отдельную сущность. Очевидно, что работа с большими денормализованными объектами чревата многочисленными проблемами при выполнении произвольных запросов к данным, когда запросы не укладываются в структуру агрегатов. В этом случае при получении требуемой информации нам придется извлекать массу информации, которая абсолютно не нужна. К сожалению, это компромисс, на который приходится идти в распределенной системе: мы не можем проводить нормализацию данных как в обычной односерверной системе, так как это создаст необходимость объединения данных с разных узлов и может привести к значительному замедлению работы базы.

**Согласованность в конечном счете (Еventual Сonsistency)**

Долгое время согласованность (Consistency) данных была “священной” для разработчиков баз данных. Все реляционные базы обеспечивали тот или иной уровень изоляции транзакций. С приходом огромных массивов информации и распределенных систем стало ясно, что обеспечить для них изолированность транзакций с одной стороны и получить высокую доступность и быстрое время отклика с другой — невозможно. Более того, даже обновление одной записи не гарантирует, что любой другой пользователь моментально увидит изменения в системе, ведь изменение в распределенной системе может произойти, например, в мастер-узле, а реплика асинхронно скопируется на узлы, с которыми работают другие пользователи. В таком случае пользователи увидят результат через какой-то, пусть минимальный, но все же промежуток времени. Это свойство принято называть “согласованностью в конечном счете” (Eventual Consistency) и это то, на что идут сейчас все крупнейшие интернет-компании мира, включая Facebook и Amazon. Последние, например, декларируют, что максимальный интервал, в течение которого пользователь может видеть несогласованные данные, составляет не более секунды. Пример такой ситуации показан на слайде.



Два пользователя с разницей в несколько миллисекунд забронировали один и тот же номер. Второй пользователь умудрился попасть в момент заказа именно в тот временной интервал, когда данные на несколько миллисекунд оказались несогласованными. Такое событие крайне маловероятно, но оно все же случилось! Что делать в этом случае? Разумеется, бизнес, который осознанно выбрал приложение и хранилище, обеспечивающие быстрый отклик системы для конечного пользователя и свойство “согласованности в конечном счете” (Eventual Consistency) должен предусматривать какие-то сценарии отката операции как с точки зрения приложения, так и самого бизнеса. В приведенном примере – в гостиницах принято держать “пул” свободных номеров на непредвиденный случай, а в приложениях выполнять откат несогласованного заказа.  
На самом деле слабая поддержка согласованности не означают, что ее нет вообще. NoSQL системы поддерживают ее в той мере, в какой она необходима для целей конкретных приложений.

**BASE-свойства**

* **Basic Availability**
* **Soft State**
* **Eventual Consistency**

Многие NoSQL системы поддерживают, так называемые, BASE-свойства (Basic Availability, Soft State, Eventual Consistency). В этом названии Basic Availability означает, что данные доступны всегда, когда к ним происходит обращение, даже если часть из них в настоящий момент недоступна; *Soft State* означает, что данные могут находиться в рассогласованном состоянии какой-то период времени, а Eventual Consistency означает, что в конечном счете, после некоторого периода времени данные в хранилище окажутся в согласованном состоянии. Тем не менее, некоторые NoSQL хранилища данных, например CouchDB обеспечивают полную поддержку согласованности, как это принято в традиционных СУБД.

В следующем фрагменте лекции мы обсудим технологии, используемые для NoSQL систем.

Опрос:

Вопрос 1.

**Какие утверждения верны?**

Ответ: (множественный выбор)

* **SQL никогда не используется в NoSQL системах.**
* **SQL иногда используется в NoSQL системах. (+)**
* **В NoSQL системах часто используются низкоуровневые языки запросов.(+)**
* **Хранилища данных NoSQL могут быть неструктурированными или слабоструктурированными. (+)**
* **Агрегаты не используются в NoSQL хранилищах.**
* **ACID-свойства всегда выполняются в NoSQL системах.**
* **Согласованность в конечном счете (Eventual Consistency) – это свойство, характерное для NoSQL систем. (+)**
* **Вертикальное масштабирование - это увеличение количества доступных для ресурсов за счет увеличения мощности применяемых серверов (+)**
* **Горизонтальное масштабирование - это повышение производительности за счет увеличение количества объединенных в кластер серверов.(+)**