электронный журнал
МОЛОДЕЖНЫЙ НАУЧНО-ТЕХНИЧЕСКИЙ ВЕСТНИК
Издатель ФГБОУ ВПО "МГТУ им. Н.Э. Баумана". Эл No. ФС77-51038.

УДК 004.65

Введение в NoSql

***Картава А.И.**, студент*
Россия, 105005, г. Москва, МГТУ им. Н.Э. Баумана,
кафедра «Системы обработки информации и управления»

Научный руководитель: Гапанюк Ю.Е., к.т.н., доцент,
Россия, 105005, г. Москва, МГТУ им. Н.Э. Баумана
gapyu@bmstu.ru

Введение

Активная деятельность по отысканию способов обобщения быстро растущего объема данных была зафиксировано еще в 60-х. И привела эта деятельность к созданию программных комплексов, называемых «Системой управления базами данных» или СУБД.

СУБД предоставляет пользователю инструменты для ввода и хранения данных. Также эти программные комплексы позволяют описывать структуры данных. Файлы, снабженные описанием данных, которые в них хранятся, и находящиеся под управлением СУБД, стали называть базами данных [1].

В настоящее время все более актуальными становятся нереляционные базы данных. В данной статье они будут сравнены с реляционными базами.

Реляционные базы данных

Сотрудник фирмы IBM доктор Кодд привнес значимый вклад в развитие реляционных баз данных, написав статью, в которой обсуждались возможности применения привычных для человека способов представления данных.

«Будучи математиком по образованию Э.Кодд предложил использовать для обработки данных аппарат теории множеств (объединение, пересечение, разность, декартово произведение). Он показал, что любое представление данных сводится к совокупности двумерных таблиц особого вида, известного в математике как отношение – relation (англ.)»[2, 3, 4].

Реляционная база данных – это множество отношений , которые содержат информацию, хранящуюся в базе данных. Пользователи же могут воспринимать такую базу данных как совокупность таблиц.

Нереляционные базы данных

За последние 20 лет в мире корпоративных вычислений многое изменилось: языки, архитектуры, платформы, процессы. Но за все это время неизменным оставалась одно – реляционность баз данных. В свое время были некоторые претенденты, которые даже имели некоторый успех, но, в целом, вопрос хранения информации всегда оставался лишь вопросом, какую реляционную базу данных выбрать. То есть проблема реляционности все же оставалась.

Термин «NoSQL» плохо определен. Этот термин в основном относится к ряду недавно выпущенных нереляционных баз данных, таких как Cassandra, Mongo, Neo4J, Riak и BrightStardb. Они хранят данные, не используя схемы, запускаются на кластерах и имеют возможность расширения свойств (напрмер, полей) , хотя и за счет потери консистенции (согласованности данных). Сторонники NoSql-баз данных заявляют, что они могут построить системы, которые более производительны, масштабируемы и проще в программировании.

Не смотря на такие явные преимущества нереляционного подхода реляционные базы данных не теряют своей актуальности. Реляционные базы данных являются мощным инструментом, который будет еще использоваться достаточно долго. Тем не менее мы сейчас входим в эпоху, в которой коммерческие и даже частные приложения используют множество технологий управления данными. В результате проектировщики баз данных должны хорошо знать эти технологии и быть в состоянии выбрать нужную в соответствии с поставленной задачей [5].

Почему NoSQL интересен?

Автор книги “NoSQL Distilled” считает, что на это есть две основных причины:

-Продуктивное развития приложений. Развитие многих приложений требует определенных усилий для отображения структуры данных, находящихся, например, в оперативной памяти, на данные в реляционной базе данных. Таким образом, получаем решение, в котором решается такое явление, как Impedance Mismatch. NoSQL-базы данных предлагают модель данных, которая лучше отвечает требованиям приложения, в результате чего упрощается взаимодействие с базой. А это, в свою очередь, означает, что требуется написание кода, меньшего по объему, требующего меньшей отладки. Сама база нуждается в меньшем объеме изменений.

- Большие объемы данных. Организации считают ценным иметь в своем распоряжении как можно больше информации и процессы быстрого ее добавления, что в

случае с реляционными базами данных является дорогим удовольствием, не говоря уже о том, возможно ли это вообще обеспечить. Главной причиной этому является то, что реляционные базы данных спроектированы для работы на одной машине, в то время как более экономично работать с большой базой данных и распределять нагрузку на кластеры, состоящие из множества меньших и более дешевых машин. Большинство NoSQL-баз данных спроектировано как раз для запуска на кластерах, поэтому они лучше подходят для работы с большим количеством информации [5].

С ростом объемов данных обостряется вопрос о переходе на кластеры. И тем ярче становится заметно, что реляционные базы данных для кластеризации не подходят. Кластерные реляционные базы данных, такие как Oracle RAC или Microsoft SQL Server построены на концепции распределенных дисковых подсистем. Они используют кластер-подобную файловую систему, но это означает, что кластер все еще содержит дисковую подсистему, как единое устройство. Реляционные базы данных могут быть запущены также и на разделенных серверах для разных наборов данных, тем самым эффективно "дробя" базу. Это разделяет нагрузку на сервер, но, в свою очередь, такое дробление должно контролироваться приложением, которое должно хранить путь, по которому сервер базы данных должен обращаться к частям этого дробления. Также при этом мы теряем возможность делать запросы, ссылочную целостность, транзакционность или контроль консистенции (согласованности). На языке специалистов этой области это неестественно.

Коммерческие реляционные базы данных не рассчитаны на работу на кластерах, а запуск их на кластеры резко повышает их стоимость.

Из-за того, что реляционные базы данных не подходят для работы на кластерах, была сформирована организация для того, чтобы найти альтернативные пути развития баз данных. Две компании, такие как Google и Amazon, были очень влиятельны в этом вопросе. Обе эти компании были успешны в техническом смысле. И это не удивительно, что они хотели покончить с реляционностью своих баз. С наступлением 2000-го обе компании имели весьма краткие, но важные наброски о проделанной работе в направлении нереляционных баз данных: BigTable от Google и Dynamo от Amazon [5].

Термин NoSql

Возможно, это покажется удивительным, но термин "NoSql" появился еще в конце 90-х. Это было названием открытой реляционной базы данных, которая хранила таблицы как ASCII-файлы, то есть каждый кортеж хранился в виде строки с разделенными полями.

База данных была так названа, потому что она не использовала SQL в качестве языка запросов.

Использование термина "NoSql", которое мы имеем ввиду сегодня имеет отношение к встрече, организованной Джоном Оскарсоном 11 июня 2009 года в Сан-Франциско, на которой планировалось обсудить новые тенденции на ранке хранения и обработки информации. Главным стимулом для встречи были новые открытые продукты, такие как BigTable и Dynamo. Так как встрече требовалось краткое название, для того , чтобы разместить его в хэш-тэге твиттера, было решено применить термин «NoSQL». Этот термин не имел глубокой смысловой нагрузки, но на просторах интернета распространился, как вирусная реклама и стал названием целого направления ИТ-индустрии. Это названии скорее стоит воспринимать как вектор развития ИТ в сторону от реляционных баз данных, нежели уход от SQL [6].

Среди масс наиболее распространено мнение, что NoSQL расшифровывается как not only SQL. Так, например, считает автор статьи на Хабре. В свою очередь авторы книги “SQL Distilled”, с этим не согласны, аргументируя это, например, тем, что если бы создатель этого термина хотел в нем выразить именно not sql , он написал бы его так: NOSQL вместо NoSQL.

Но не стоит сильно заострять внимания на этом, главное понять , что под этим термином понимается направление развития баз данных.

Транзакции (преимущество реляционных баз данных)

Приложения промышленных масштабов позволяют обращаться к базе данных большому количество пользователей одновременно. И, хотя в основном эти пользователи работают с разными частями базы, все же бывают и совпадения, когда, например два пользователя обращаются к одним и тем же данным. Ситуацию можно разобрать на хорошо известном примере резервирования комнаты в отеле. Два человека могут одновременно забронировать одну и ту же комнату. Конечно, большинство отелей имеет пару комнат в резерве, но бывают более нетривиальные ситуации, когда просто резервированием не обойтись. Чтобы избежать таких ситуаций используют реляционные базы данных, предлагающие такую возможность, как транзакции. Но транзакции не являются панацеей: все равно придется обрабатывать транзакционные ошибки при попытке, например, забронировать комнату, которая только что ушла [5].

ACID

В течении долго времени консистентность данных была одним из главных преимуществ реляционных баз данных. Но с ростом объемов данных и с приходом распределенных систем стало ясно, что одновременно обеспечить транзакционность и быстродействие стало невозможно. Более того, даже транзакции не может обеспечить стопроцентную гарантию того, любой другой пользователь сразу же увидит изменения в системе. Это обусловлено тем, что изменение может быть произведено в главной(master)-ноде, а пользователь в это время работает с зависимой (slave) нодой, которая может обновляться асинхронно . Таким образом, пользователь увидит результат не сразу, а через некоторое время. Это называется eventual consistency.

Как показывает практика, на сегодняшний момент атомарная консистентность уже не так актуальна. Вот что сказал один разработчик финансовой банковской системы: “Если бы мы действительно ждали завершения каждой транзакции в мировой сети ATM (банкоматов), транзакции занимали бы столько времени, что клиенты убегали бы прочь в ярости. Что происходит, если ты и твой партнер снимаете деньги одновременно и превышаете лимит? — Вы оба получите деньги, а мы поправим это позже.”

На самом деле слабые ACID свойства не означают, что их нет вообще. Транзакции используются в реляционных базах данных при изменении логических связей. При правильном проектировании модели данных в NoSQL базе можно добиться уровня изоляции, не уступающему оному в реляционной базе.[6]

Несоответствие (Impedance Mismatch)

У реляционных баз данных есть множество преимуществ. Но не все так уж с ними хорошо. Уже начиная с первых дней эксплуатации реляционных баз данных, были обнаружены недостатки, которые заставили разочароваться в универсальности таких баз данных. Для разработчиков приложения для баз данных главным разочарованием было то, что обычно называют impedance mismatch. Impedance mismatch - это несоответствие между моделью базы данных и структурой тех данных, которые находятся в памяти. Как известно, в реляционной модели данные хранятся в структуре отношений и кортежей.

Такой подход обеспечивает простоту, но также привносит ограничения. Отчасти, это заключается в том, что реляционные кортежи должны быть простыми, то есть они не могут содержать структуру, будь то вложенная запись или список. Такое ограничение не является желательным, так как данные, хранящиеся в памяти могут иметь структуру, изменяющуюся в широких границах. В результате, для того , чтобы использовать ту

структуру данных, которая находится в памяти, необходимо тем или иным образом перевести ее в такое представление, которое может быть сохранено в базе [5].

Неструктурированные (schemaless)

В отличие от реляционных баз данных, в которых структура данных регламентирована, в NoSQL-базах такого регламента нет, то есть структура типизирована слабо. В отдельной строке можно добавить поле без предварительного декларирования изменений. То есть при необходимости поменять модель данных, достаточно изменить код в приложении.

прмер изменения имени поля в MongoDB

```
BasicDBObject order = new BasicDBObject();
order.put(“date”, orderDate); // это поле было давно
order.put(“totalSum”, total); // раньше мы использовали просто “sum”
```

(Пример взят из статьи на Хабре)

От проделанных изменений следует ожидать новое поле при чтении. Но, так как схема данных отсутствует, то поле totalSum отсутствует у уже существующих объектов. Автор статьи предлагает два варианта решения проблем. Первый заключается в том, чтобы обойти всю базу и обновить это поле во всех документах. Но так как объемы базы велики, то этот процесс происходит без каких-либо блокировок, в следствии этого данные могут уже считываться другими процессами. Поэтому второй вариант, предложенный автором неизбежен. Он заключается в проверке кода приложения.

```
BasicDBObject order = new BasicDBObject();
Double totalSum = order.getDouble(“sum”); // Это старая модель
if (totalSum  == null)
   totalSum = order.getDouble(“totalSum”); // Это обновленная модель
```

Смысл кода в том, что при повторной записи в базу, поле будет уже в новом формате.[5,6]

"В качестве примера можно привести Твиттер, который лет пять назад вместе с твиттом хранил лишь немного дополнительной информации (время, Twitter handle и еще несколько байтов метаинформации), однако сейчас в дополнение к самому сообщению в базе сохраняется еще несколько килобайт метаданных. "[6]

Данные в виде агрегатов

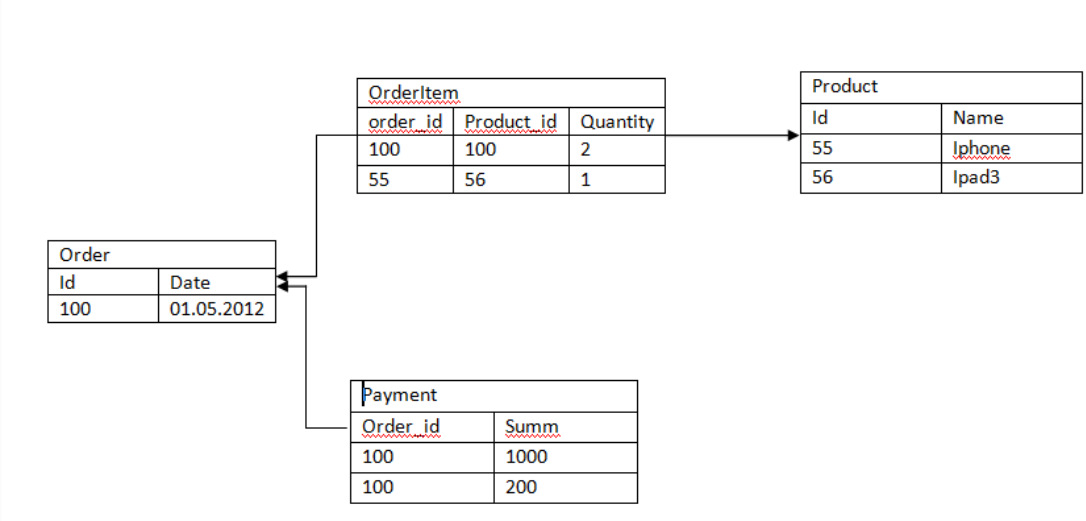

Рис. 1. Реляционная модель

В реляционной модели в целях нормализации бизнес-сущность приложения сохраняется в физические таблицы, NoSQL хранилища оперируют с сущностями, как с целыми объектами. (Пример взят с хабра в собственной нотации)

"На рисунке 2 продемонстрированы агрегаты для стандартной концептуальной реляционной модели e-commerce “заказ — позиции заказа — платежи — продукт”. В обоих случаях заказ объединяется с позициями в один логический объект, при этом каждая позиция хранит в себе ссылку на продукт и некоторые его атрибуты, например, название (такая денормализация необходима, чтобы не запрашивать объект продукта при извлечении заказа — главное правило распределенных систем — минимум “джоинов” между объектами). В одном агрегате платежи объединены с заказом и являются составной частью объекта, в другом — вынесены в отдельный объект. Этим демонстрируется главное правило проектирования структуры данных в NoSQL базах — она должна подчиняться требованиям приложения и быть максимально оптимизированной под наиболее частые запросы. Если платежи регулярно извлекаются вместе с заказом — имеет смысл их включать в общий объект.

Если же многие запросы работают только с платежами — значит, лучше их вынести в отдельную сущность."[6]

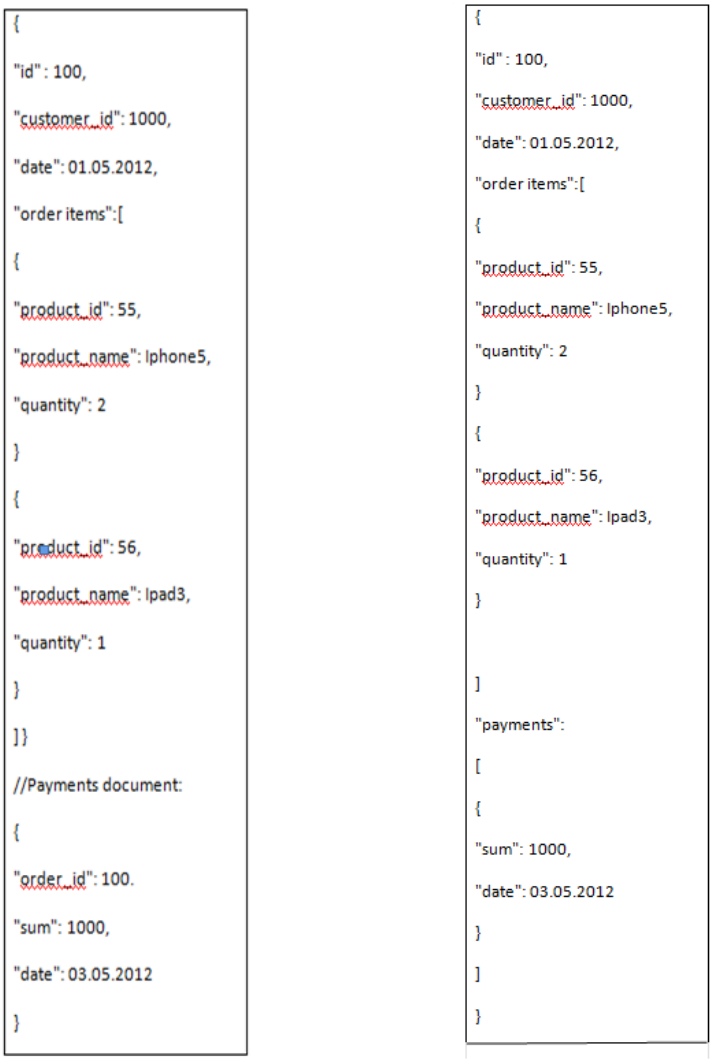

Рис. 2. Данные в виде агрегатов

Плюсы и минусы обоих подходов отображены в таблице.

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

Возможные варианты реализации NoSQL-баз данных

"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:"[6]

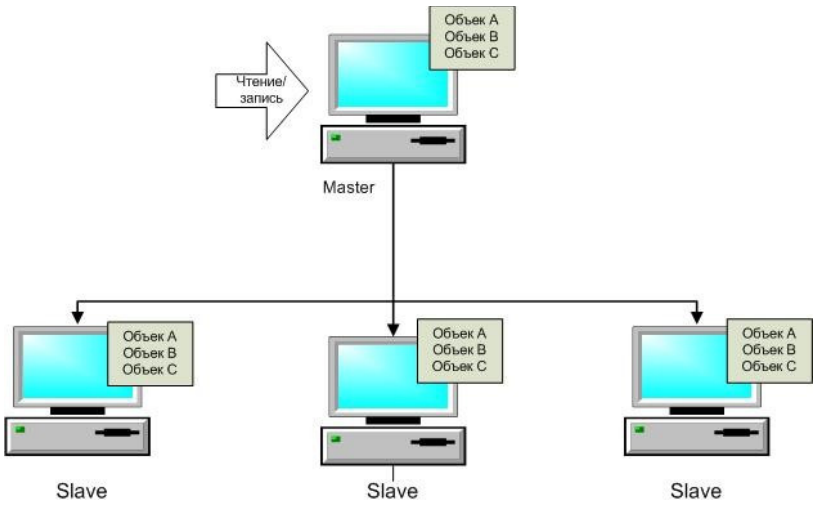

Рис. 3. Master-slave

peer-to-peer

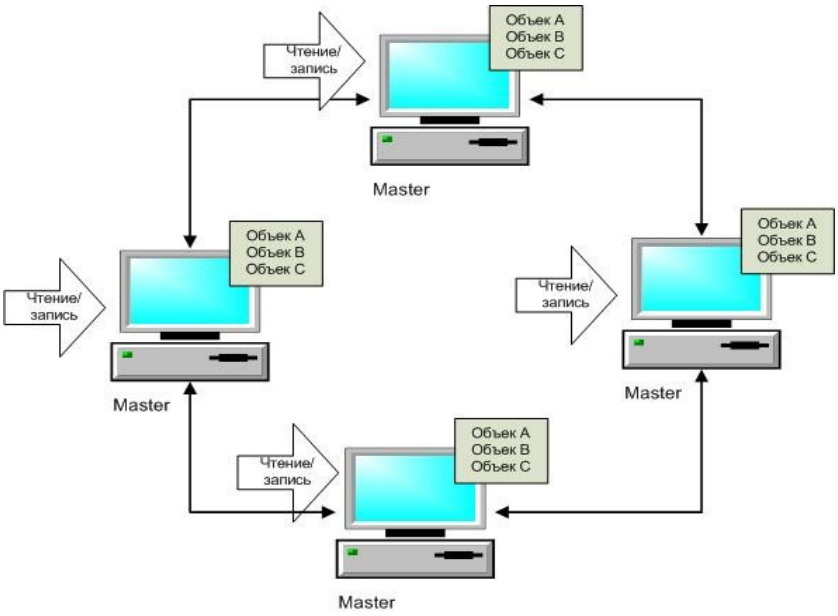

Рис. 4. Peer-to-peer

Первый тип предполагает хорошую масштабируемость на чтение, но слабую масштабируемость на запись, так запись производится в master-node'e. Для второго типа репликации предполагается, что все узлы равны и могут обслуживать как запросы на чтение, так и на запись.

Шардинг — разделение данных по узлам:

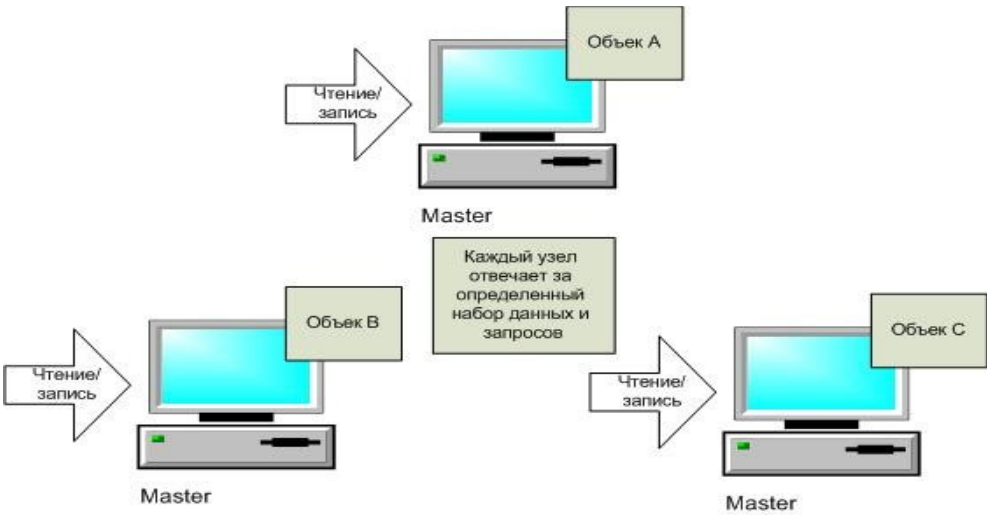

Рис. 5. Шардинг

В качестве реального примера могу привести некоторое подобие своей NoSQL-базы данных. Она реализована по принципу peer-to-peer:

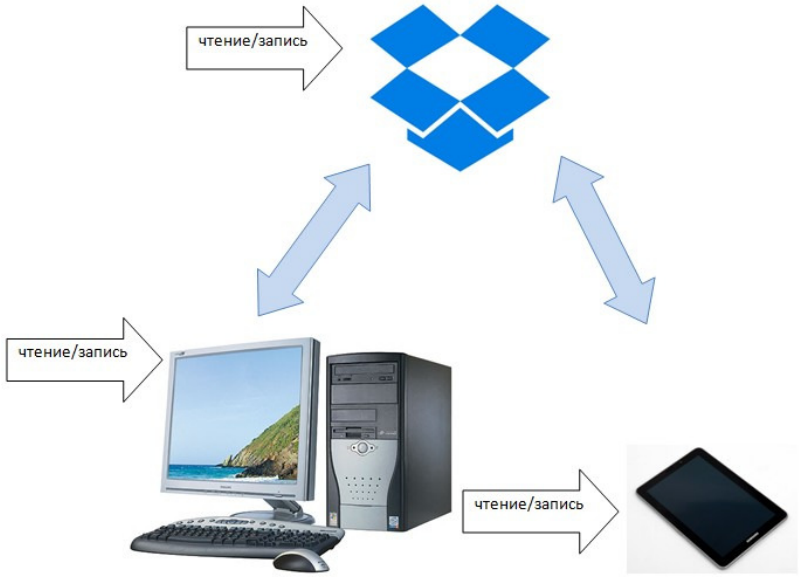

Рис. 6. Peer-to-peer

Такая "база" способна обеспечить меня всей той информацией дома(ПК), в пути(планшет), где-либо еще при наличии любого ПК(dropbox). Синхронизация происходит в двухстороннем режиме, то есть где бы информация не создавалась (записывалась), удалялась или редактировалась вся база будет полностью консистентна, а точнее eventual-консистентна, так как данные, например на планшете, обновляются при подключении его к сети wifi.

Вывод

NoSql-базы данных представляют собой перспективную технологию, которая позволяет оперировать с огромными объемами данных, распределенных между серверами. По мнению экспертов, перед проектированием базы данных надо определиться с тем какую модель выбрать: реляционную или нет. В случае, если по всем критериям подходит именно реляционная модель или нереляционная выбод очевиден, но если по совокупности критериев получается так, что можно использовать и ту, и другую модель, то следует выбрать нереляционную.

Статистическая справка:

Выше 50% программистов из США уже активно используют NoSQL, причем на уровне своих компаний. Остальные разработчики в ближайшие 2-3 года тоже планируют взяться за эту технологию.

В странах Азии программисты пока не спешат с переходом на нереляционные базы данных, так как считают традиционные средства разработки оправдывающими себя. Таким образом, 70% разработчиков в Азии предпочитают реляционные решения.

В Европе еще меньше приверженцев новой технологии – 39% потенциальных разработчиков.

Список литературы

1. Концепция баз данных. Режим доступа: http://citforum.ru/database/dbguide/1-2.shtml (дата обращения 15.03.2014).
2. Дейт К. Руководство по реляционной СУБД DB2. М.: Финансы и статистика, 1988. 320 с.
3. Мейер М. Теория реляционных баз данных. М.: Мир, 1987. 608 с.
4. Ульман Дж. Базы данных на Паскале. М.: Машиностроение, 1990. 386 с.
5. Sadalage P.J., Fowler M. NoSQL Distilled: A Brief Guide to the Emerging World of Polyglot Persistence. Wesley: Addison-Wesley Professional, 2012. 192c.
6. NoSQL базы данных: понимаем суть. Режим доступа: http://habrahabr.ru/post/152477 (дата обращения 02.03.2014).