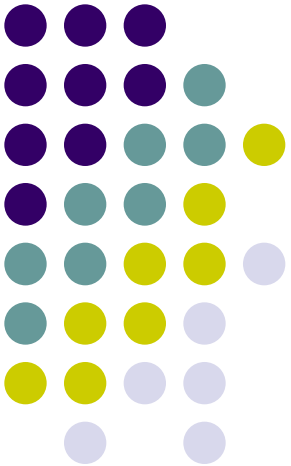
Введение в распределенные методы обработки информации



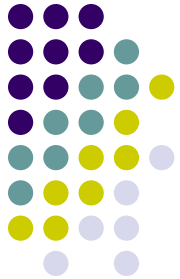
Лекция №6

Распределенные реляционные базы данных. SQL и распределенные базы данных.

NoSQL базы данных

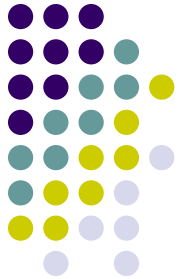
New SQL базы данных

Реляционная модель данных Обобщение



- Реляционная модель есть представление БД в виде совокупности упорядоченных нормализованных отношений
- Для реляционных отношений характерны следующие особенности.
  - Любой тип записи (кортежа) содержит только простые (по структуре) элементы данных.
  - Порядок записей в таблице несуществен.
  - Упорядочение значащих атрибутов в кортеже должно соответствовать упорядочению атрибутов в реляционном отношении.
  - Любое отношение должно содержать один атрибут или более, которые вместе составляют уникальный первичный ключ.
  - Если между двумя реляционными отношениями существует зависимость, то одно отношение является исходным, второе - подчиненным.
  - Чтобы между двумя реляционными отношениями существовала зависимость, атрибут, служащие первичным ключом в исходном отношении, должны также присутствовать в подчиненном отношении.

Достоинства и недостатки РМД



Достоинства РМД:

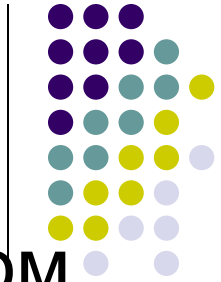
- простота представления и формирования базы данных
- универсальностью и удобством обработки данных, которая осуществляется с помощью декларативного языка запросов SQL (Structured Query Language)

Достоинства обеспечили широкое распространение РМД

Недостатки РМД:

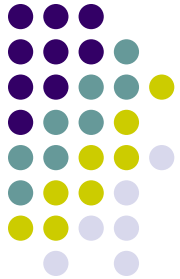
- сложность моделирование предметной области
- нет специальных средств для отображения различных типов связей и агрегатов (приходится проводить нормализацию отношений)
- отсутствие специальных механизмов навигации, что приводит к увеличению времени поиска определенных данных

Базовые операции SQL

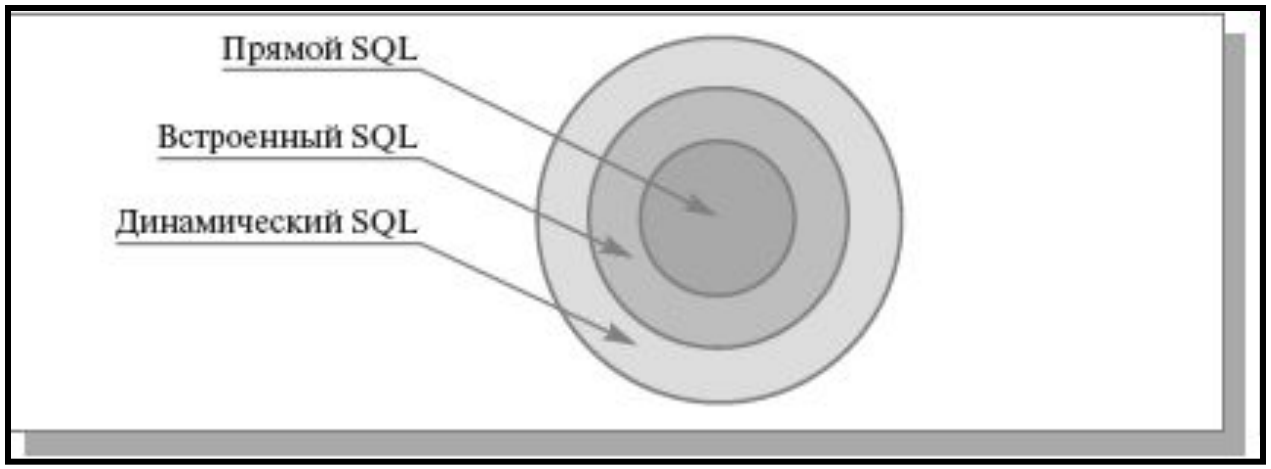


- Изначально, SQL был основным способом работы пользователя с базой данных и позволял выполнять следующий набор операций:
  - создание в базе данных новой таблицы;
  - добавление в таблицу новых записей;
  - изменение записей;
  - удаление записей;
  - выборка записей из одной или нескольких таблиц (в соответствии с заданным условием);
  - изменение структур таблиц

SQL - развитие

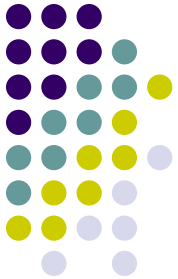


- обеспечиваются возможности описания и управления новыми хранимыми объектами (например, индексы, представления, триггеры и хранимые процедуры)
- стал приобретать черты, свойственные языкам программирования
- пользователю предоставляются развитые средства визуального построения запросов



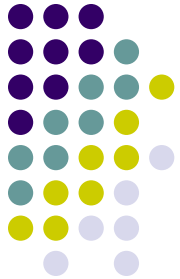
- ***прямой (direct) SQL***
  - конструкции языка используются при "прямом" взаимодействии пользователя с СУБД
  - является базовым уровнем
- ***встроенный (embedded) SQL***
  - содержит конструкции, позволяющие использовать возможности *прямого* SQL в программах, написанных на традиционных языках программирования
- ***динамический (dynamic)***
  - во *встраиваемый* SQL добавляются конструкции, позволяющие приложениям обращаться к СУБД с конструкциями прямого SQL, которые динамически образуются во время выполнения программы

SQL и распределенные базы данных



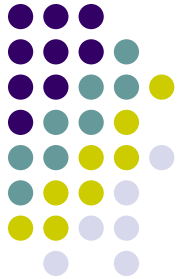
- распределенные БД определяют сегодня развитие технологий реляционных баз данных и языка SQL
- задачи SQL в распределенных БД:
  - ***удаленный запрос*** - отдельный SQL оператор обращается к одной удаленной базе данных, и каждый оператор – это отдельная транзакция. Таким образом, пользователь может выполнить несколько SQL операторов, обращающихся к разным базам данных, но каждый раз это будет отдельная транзакция;
  - ***удаленная транзакция*** – транзакция уже из нескольких SQL операторов, обращающихся к одной удаленной базе данных
  - ***распределенные транзакции*** – каждый оператор транзакции обращается к одной удаленной базе данных, но одна транзакция может обращаться к нескольким
  - ***распределенные запросы*** – отдельный SQL оператор может обращаться к нескольким удаленным базам данных, причем транзакция из таких операторов может обращаться также к нескольким базам данных

SQL и распределенные базы данных



- ***проблемы:***
  - ***план выполнения статического оператора SQL:***
    - встроенная статическая инструкция SQL компилируется и сохраняется в базе данных в виде плана выполнения
    - когда запрос объединяет данные из двух или более баз данных, в какой из них следует хранить план выполнения?
    - иметь два или более согласованных плана?
    - если изменяется структура одной базы данных, то как можно изменить план выполнения в другой базе данных?
  - ***Выход -*** применение динамического SQL в сетевой среде
  - ***НО!*** снижается производительность приложений из-за повышения сетевого трафика и многочисленных задержек.

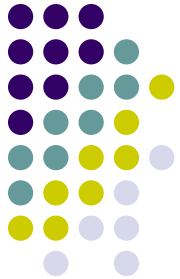
SQL и распределенные базы данных



- ***Проблема оптимизации:***
  - в распределенных БД нельзя применять обычные правила оптимизации инструкций SQL
  - например, полное сканирование локальной таблицы может оказаться оптимальнее, чем поиск по индексу в удаленной таблице
  - программа оптимизации должна знать параметры сети и, в частности, ее быстродействие

  В распределенных БД роль оптимизации становится более важной, а ее осуществление более трудным.

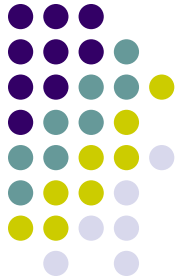
SQL и распределенные базы данных



- ***Проблема совместимости данных***
  - в различных вычислительных системах существуют разные типы данных
  - данные одного и того же типа в разных системах могут иметь разные форматы

  В распределенной СУБД эти различия должны быть незаметны
- ***Оборудование от разных поставщиков***
  - различные СУБД => различные диалекты SQL

SQL и облачные вычисления

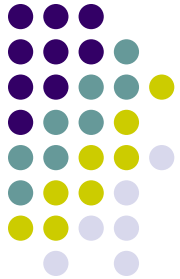


Задачи проекция традиционного SQL на облако:

- решить проблему масштабирования (произвольного увеличения количества серверов распределенных БД)
- предоставить возможность для работы с базой данных посредством интернет-сервисов
- построить в облаке проект реляционной базы данных со всеми преимуществами, предоставляемыми любой облачной технологией

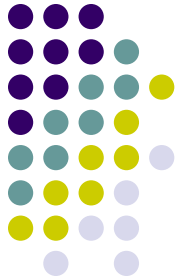
Одно из решений – новая технология Microsoft - SQL Azure

NoSQL и SQL



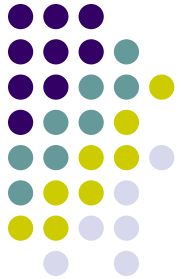
- концепция NoSQL (англ. not only SQL, не только SQL):
  - расширить возможности БД там, где SQL недостаточно гибок
  - оставить SQL там, где он справляется со своими задачами
- проблемы реляционных БД:
  - сложности при работе с данными очень большого объема
  - проблема масштабируемости
- основа концепции NoSQL:

NoSQL и SQL



- методологические обоснования – основа - теорема CAP:
  - в распределённой системе невозможно одновременно обеспечить:
    - согласованность данных,
    - доступность (англ. availability, в смысле наличия отклика по любому запросу)
    - устойчивость к расщеплению распределённой системы на изолированные части

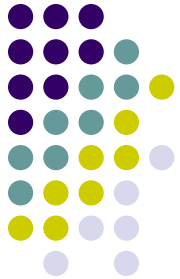
NoSQL и SQL



- предлагается:
  - обеспечить высокую доступность и устойчивости к разделению
  - не фокусироваться на средствах обеспечения согласованности данных, обеспечиваемых традиционными SQL-ориентированными СУБД с транзакционными механизмами на принципах ACID

То есть предлагается пожертвовать согласованностью данных ради доступности и масштабируемости

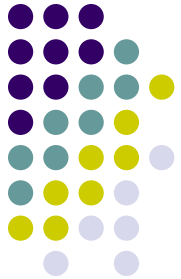
Что такое NoSQL? Предпосылки развития NoSQL технологий



- Появление в начале 2000-х
  - Google - поисковые системы
  - Facebook – социальные сети
  - И.т.д.
- Этим системам свойственно
  - Постоянно меняющаяся структура
  - Непредсказуемый рост количества данных
  - Огромное число пользователей

Реляционные базы данных не справлялись

Что такое NoSQL(***not only SQL***) СУБД?

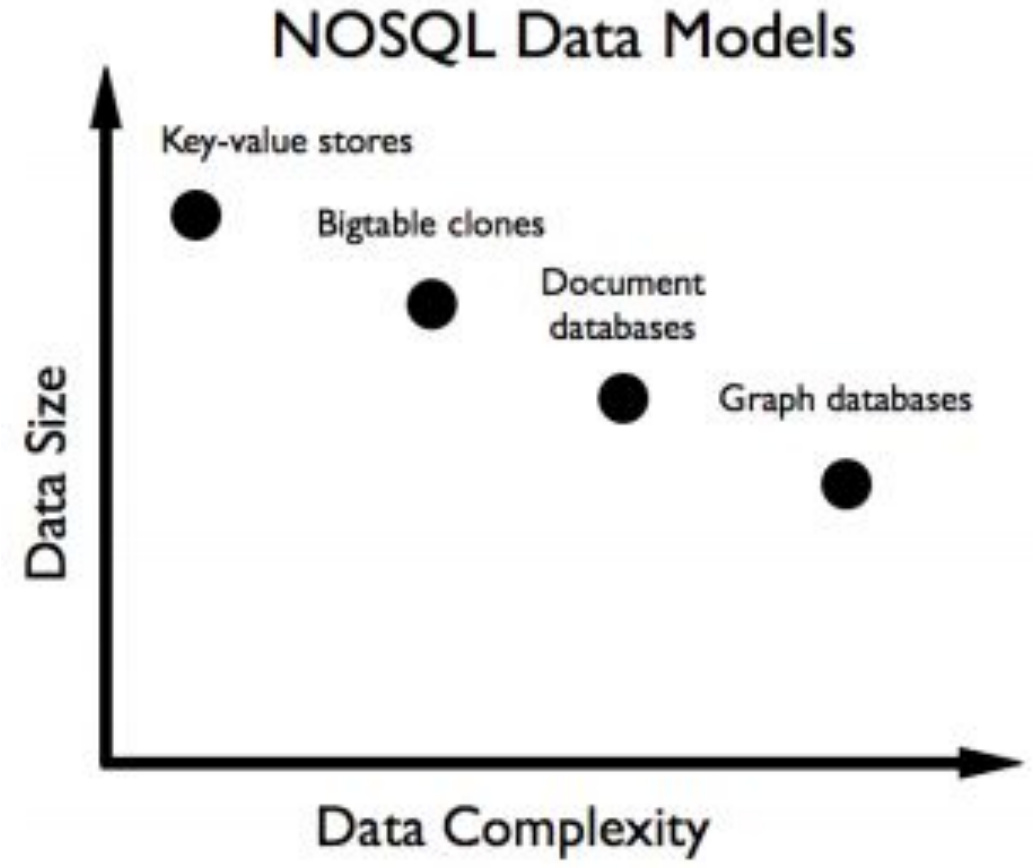


- специализация БД для конкретной области применения (позволяет обеспечить более высокую скорость обработки данных);
- разделение БД на системы хранения, системы обработки потоков данных в режиме реального времени и гибридные системы;
- целевая направленность на обработку распределенных данных (многие базы NoSQL используют разные серверные продукты, осуществляют резервное хранение, поддерживают географическую распределенность, не испытывают зависимости от центрального узла)
- На данный момент существует очень много различных NoSQL СУБД (около 130)

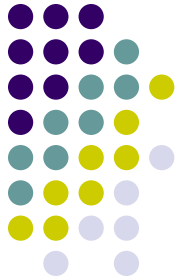
Виды NoSQL

- Все NoSQL СУБД разделяются на несколько категорий:
  - Key-value stores / Хранилища типа «ключ-значение»
  - Column Family (Bigtable) stores / Масштабируемые распределенные хранилища
  - Graph Stores / Графовые СУБД
  - Document Stores / Документно-ориентированные СУБД

На рисунке схематично обозначены объемы используемых данных и сложность этих данных в этих видах NoSQL



Языки запросов баз NoSQL

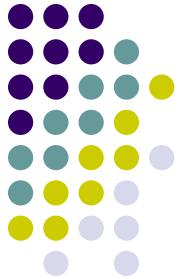


В качестве языка запросов баз NoSQL используется либо

- специализированные программные продукты (например, MapReduce, Sawzall),
- либо несколько расширенный SQL, позволяющий извлекать сложные объекты из одной таблицы без операций соединения.

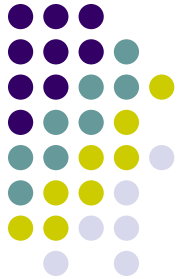
Назначение	*Формат хранения*	*Структура хранения*	*Способ обработки данных*	*Примеры*
Хранение структури-рованных данных	**Текст и примитивные типы данных**	**Двумерные таблицы**	**SQL**	**DB2, Oracle, MSSQL Server**
Хранение документов	**JSON (JavaScript Object Notation), XML**	**Деревья**	**MapReduce, XQuery**	**CouchDB, MongoDB, eXist**
Хранение неструктури-рованных данных	**Произвольный формат**	**Отображения типа ключ-значение**	**Sawzall**	**BigTable, Dynamo**
Хранение сложно взаимосвязан-ных объектов	**JSON (JavaScript Object Notation), XML и др.**	**Графы и гиперграфы**	**SPARQL**	**Neo4j, AllegroGraph, Sones graphDB**

Анализ таблицы: реляционные базы данных



- реляционные базы данных предназначены для хранения структурированной информации в виде двумерных таблиц, которая обрабатываются с помощь языка запросов SQL
- реляционные базы относятся к типу CP, т.е. они направлены на обеспечение согласованности данных и устойчивости системы к сбоям узлов
- именно их следует использовать для хранения бухгалтерской и финансовой информации, в частности для реализаций транзакций по банковским картам.
- использовать для этих целей базы NoSQL неразумно, поскольку в последних согласованность данных приносится в жертву доступности системы и ее устойчивости

Анализ таблицы: категории NoSQL баз данных



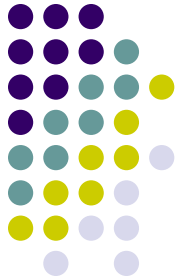
Первая категория — это базы данных (ключ значение).

- это очень большие хэш-таблицы[1], где каждому ключу поставлено в соответствие значение.
- могут очень быстро оперировать колоссальными объемами информации
- имеют серьезные ограничения в языке запросов.

Примеры:DynomiteПримеры:Dynomite, VoldemortПримеры:Dynomite, Voldemort, TokyoПримеры:Dynomite, Voldemort, Tokyo, Redis, BigTable.

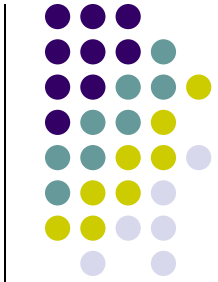
[1] Хеш-табли́ца — это структура данных, которая позволяет хранить пары (ключ, значение) и выполнять три операции: операцию добавления новой пары, операцию поиска и операцию удаления пары по ключу.

Что такое key-value БД?



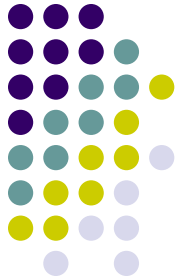
- Этот тип БД работает с данными типа ключ-значение. Здесь нет места ни структуре, ни связям.
- После подключения к серверу (например Redis) приложение может задать ключ и его значение, а в последствии получать эти данные по запросу.
- Такие СУБД обычно используются для быстрого сохранения базовых данных, а иногда не таких уж и базовых, если подсчитать затраты процессора и памяти.
- Они, обычно, очень быстры, работоспособны или легко масштабируемы (хорошо использовать такие БД для хранения сессий, кеша, счётчиков посещений или просмотров и т.д.).

Зачем нужно такое решение, если есть MySQL, PostgreSQL, Oracle...?



- Решая такую простую задачу, как сохранение/чтение значений по ключу, система работает крайне эффективно, т.к. в ней отсутствуют тяжелые слои SQL обработчиков, систем индексации, профилирования, вакуумизации (для PostgreSQL) и т.д.
- Подобное решение дает наиболее эффективную производительность, минимальную стоимость внедрения и масштабирования.

Пример на основе авторизации пользователя

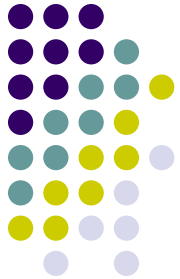


- Сейчас все представили себе стандартное решение — таблица в MySQL на три колонки

ID | login | password |

- Регистрация происходит следующим образом — мы проверяем, нет ли такого же логина в таблице и вставляем новую строку.
- Авторизация — делаем выборку по связке логин — пароль (или хеш пароля).
- А теперь вопрос — зачем мы использовали MySQL для решения такой задачи, если *мы не использовали 99%* возможностей этой РСУБД.

Пример на основе авторизации пользователя

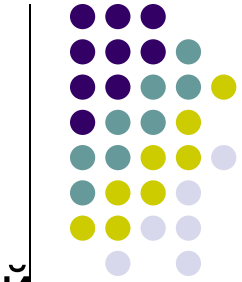


- Давайте ту же задачу рассмотрим в приближении БД ключ-значение:
  - *Регистрация*. У нас есть логин (уникальная колонка) и пароль.
  - *Авторизация* будет происходить следующим образом — делаем выборку по логину (это ключ) — получаем пароль, сравниваем с тем, что написал пользователь — готово.
- Как Вы успели заметить, у нас нет поля "user_id", без которого будет крайне трудно построить систему любой сложности. Как это решается?
- Ключом будет user_id, который будет увеличиваться при каждой регистрации на 1 (текущее значение autoincrement будем также хранить в паре ключ=значение)
- Т.к. во время авторизации нам нужно делать выборку по логину, то нужно хранить еще одну пару: логин-user_id

Реляционная БД	Хранилище типа ключ-значение
База данных состоит из таблиц Таблицы содержат колонки и строки Строки состоят из значений колонок. Все строки одной таблицы имеют единую структуру.	Для доменов можно провести аналогию с таблицами, однако в отличие от таблиц для доменов не определяется структура данных. Домен – это такая коробка, в которую вы можете складывать все что угодно. Записи внутри одного домена могут иметь разную структуру.
Модель данных определена заранее. Является строго типизированной, содержит ограничения и отношения для обеспечения целостности данных.	Записи идентифицируются по ключу, при этом каждая запись имеет динамический набор атрибутов, связанных с ней.

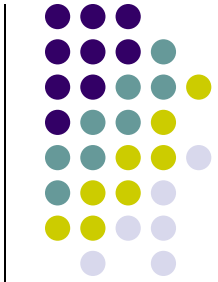
Реляционная БД	Хранилище типа ключ-значение
Модель данных основана на естественном представлении содержащихся данных, а не на функциональности приложения.	В некоторых реализация атрибуты могут быть только строковыми. В других реализациях атрибуты имеют простые типы данных, которые отражают типы, использующиеся в программировании: целые числа, массива строк и списки.
Модель данных подвергается нормализации, чтобы избежать дублирования данных. Нормализация порождает отношения между таблицами. Отношения связывают данные разных таблиц.	Между доменами, также как и внутри одного домена, отношения явно не определены.

Хранилища типа ключ-значение ориентированы на работу с записями



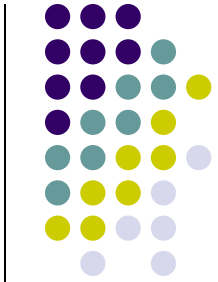
- Это значит, что вся информация, относящаяся к данной записи, хранится вместе с ней.
- Домен (о котором можно думать как о таблице) может содержать бессчетное количество различных записей.
- Например, домен может содержать информацию о клиентах и о заказах. Это означает, что данные, как правило, дублируются между разными доменами. Это приемлемый подход, поскольку дисковое пространство дешево.
- Главное, что он позволяет все связанные данные хранить в одном месте, что улучшает масштабируемость, поскольку исчезает необходимость соединять данные из различных таблиц.
- При использовании реляционной БД, потребовалось бы использовать соединения, чтобы сгруппировать в одном месте нужную информацию.

Доступ к данным



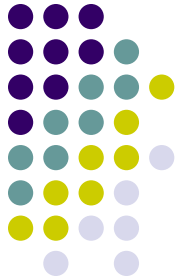
Реляционная БД	Хранилище типа ключ-значение
Данные создаются, обновляются, удаляются и запрашиваются с использованием языка структурированных запросов (SQL).	Данные создаются, обновляются, удаляются и запрашиваются с использованием вызова API методов.
SQL-запросы могут извлекать данные как из одиночной таблице, так и из нескольких таблиц, используя при этом соединения (join'ы).	Некоторые реализации предоставляют SQL-подобный синтаксис для задания условий фильтрации.

Доступ к данным



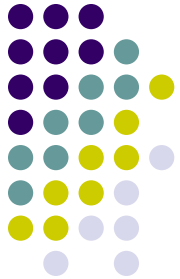
Реляционная БД	Хранилище типа ключ-значение
SQL-запросы могут включать агрегации и сложные фильтры.	Зачастую можно использовать только базовые операторы сравнений (=, !=, <, >, <= и =>).
Реляционная БД обычно содержит встроенную логику, такую как триггеры, хранимые процедуры и функции.	Вся бизнес-логика и логика для поддержки целостности данных содержится в коде приложений.

Хранилища типа ключ-значение: преимущества



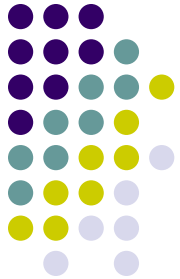
- **РСУБД** (RDBMS) слишком **медленные**, имеют **тяжелую прослойку** SQL движков, **тяжело масштабируются**
- РСУБД не достаточно хороши в плане показателя ***concurrency*** (обработка **одновременных запросов**)
- Слишком большая **стоимость** решения РСУБД для **хранения мелких порций данных**
- Нет необходимости в SQL запросах, индексах, триггерах, хранимых процедурах, временных таблицах, видах и т.д.
- БД **ключ-значение** легко **масштабируемы** и **высокопроизводительны** ввиду своей **легкости**

Хранилища типа ключ-значение: недостатки



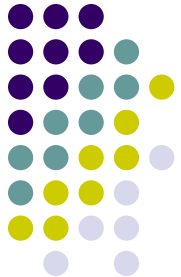
- Преимущество реляционных БД заключается в том, что они вынуждают вас пройти через **процесс разработки модели данных**.
- Ограничения в реляционных БД **гарантируют целостность** данных на самом низком уровне.
- Данные, которые не удовлетворяют ограничениям, физически не могут попасть в базу.
- **Контроль** целостности данных полностью **лежит на приложениях**. Однако в любом коде есть ошибки.

Хранилища типа ключ-значение: недостатки



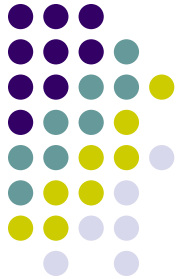
- Если ошибки в правильно спроектированной **реляционной** БД обычно **не ведут к проблемам** целостности данных, то ошибки в хранилищах типа **ключ-значение** обычно **приводят к** таким **проблемам**.
- Таким образом, в **РСУБД** данные становятся **независимы от приложения**. Это значит, что **другое приложение** сможет **использовать** те же самые **данные** и **логика приложения** может быть **изменена без** каких-либо **изменений** в **модели** базы.

Выводы



- Экономя время на анализе во время разработки, вы теряете время и деньги, масштабируя решения, которые не предусмотрены для возложенных на них задач.
- Использовать РСУБД для хранения пар ключ-значение и чтений значений по ключу — это показатель того, что Вы действуете крайне не эффективно.
- Тем не менее, думайте заранее — если Вам необходимо делать выборки списков или строить графики статистики, Вам *придется* использовать реляционные СУБД (для чего они собственно и нужны)

Документо-ориентированные базы данных

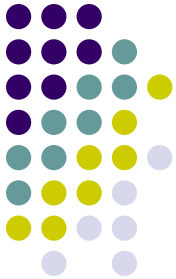


Такие базы немного напоминают базы (ключ-значение), но в данном случае, база данных знает, что из себя представляют значения.

- Обычно, значением является некоторый документ или объект, к структуре которого можно делать запросы.

Примерами таких баз являются CouchDBПримерами таких баз являются CouchDB и MongoDB.

Графовые базы данных

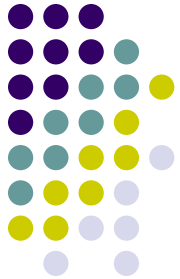


В особую категорию относят базы, данных построенные на графах.

- Такие базы ориентированы на поддержку сложных взаимосвязей между объектами, и основываются на теории графов.
- Структура данных представляет собой набор узлов, связанных между собой ссылками.
- При этом и узлы и ссылки могут обладать некоторым количеством атрибутов.

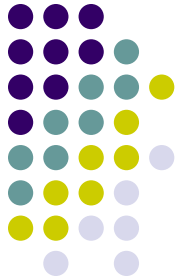
Примеры: Neo4j, AllegroGraph, Sones graphDB.

Объектно-ориентированные базы данных



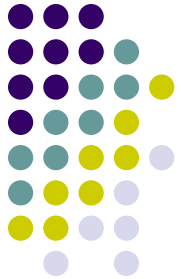
- Также существует еще одна категория, которую обычно не относят к NoSQL.
- Это - объектно-ориентированные базы данных
- Такие базы предназначены прежде всего для поддержки объектно-ориентированной парадигмы программирования.
- Их очень просто использовать в языках программирования, в которых поддерживается эта парадигма.

Преимущества постреляционных БД



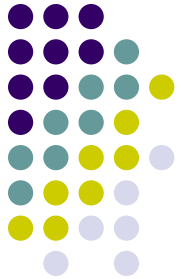
- Кроме отказа от нормализации, постреляционные СУБД позволяют хранить в полях отношений данные абстрактных, определяемых пользователями типов.
- Это дает возможность решать задачи нового уровня, хранить объекты и массивы данных, ориентированные на конкретные предметные области, а также роднит постреляционные СУБД с еще одним классом – объектно-ориентированными СУБД.

Объектно-ориентированные базы данных **(ООБД)**



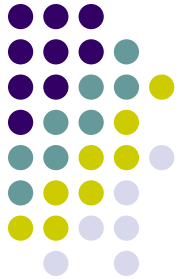
- ООБД - базы данных, в которых информация представлена в виде объектов,
- в ООБД любая сущность – объект и обрабатывается как объект;
- в ООБД используется понятие "объект" из объектно-ориентированного программирования

Объектно-ориентированная парадигма



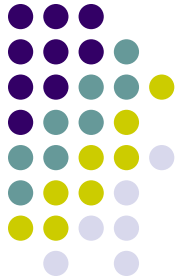
- Термин "объект" в программной индустрии впервые был введен в языке Simula (1967 г.) и означал какой-либо аспект моделируемой реальности.
- Сейчас под объектом понимается "нечто, имеющее четко определенные границы" (определение известного американского специалиста Г.Буча).
- В ООБД термин "объект" означает комбинацию "данных" и "программы", представляющих некоторую сущность реального мира.

Объектно-ориентированная парадигма



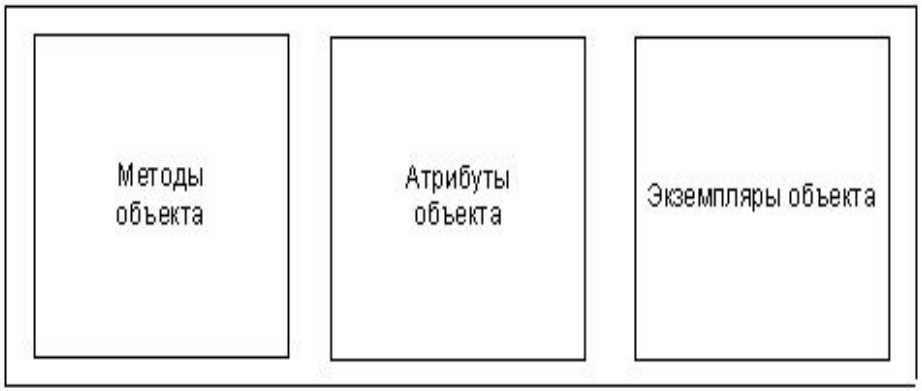
- "Данные" состоят из компонентов произвольного типа, называемых "атрибутами".
- Характеристики объекта моделируются его атрибутами.
- Каждая программа в программной части называется "методом".
- Идентификаторы объектов – дескрипторы.

Структура объектной модели



- Объекты, обладающие одинаковыми свойствами, составляют классы (например, курица, пингвин и чайка - объекты класса "птицы").
- Обычно класс описывается как новый тип данных, а объекты (экземпляры класса) - определенные на его основе переменные.
- Классы образуют иерархию наследования, заимствуя свойства друг друга;
- Каждый класс имеет определенную совокупность методов,
- Классы взаимодействуют друг с другом посредством механизма сообщений;

Схема представления класса объектов



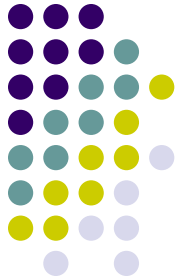
Объект ФАКУЛЬТЕТ

МЕТОДЫ	АТРИБУТЫ
Создать объект	Название
Модифицировать	Номер
Удалить	Декан
Вывести на экран	Подразделение - класс
Выдать значения атрибутов_____ у объекта_____	

Объект ПОДРАЗДЕЛЕНИЕ

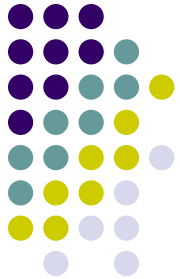
МЕТОДЫ	АТРИБУТЫ
Создать объект	Название
Модифицировать	Зав. кафедрой
Удалить	Декан
Вывести на экран	Сотрудник- класс
Выдать значения атрибутов_____ у объекта_____	

Структура объектной модели



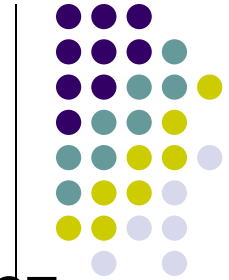
- Структура объектной модели описываются с помощью трех ключевых понятий:
  - ***инкапсуляция*** – свойство объекта скрывать свою внутреннюю структуру
  - ***наследование*** - возможность создавать из классов объектов новые классы объектов, которые наследуют структуру и методы своих предков, добавляя к ним черты, отражающие их собственную индивидуальность
  - ***полиморфизм*** -различные объекты могут по разному реагировать на одинаковые внешние события в зависимости от того, как реализованы их методы

Характеристики ООБД



- Объектно-ориентированные базы данных обычно рекомендованы для тех случаев, когда требуется высокопроизводительная обработка данных, имеющих сложную структуру.
- Выбор обязательных характеристик ООБД основан на 2 критериях: система должна быть объектно-ориентированной и представлять собой базу данных.

Преимущества ООБД

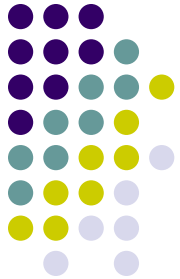


Объектно-ориентированный подход предоставляет мощные средства конструирования типов данных

Эти средства устраняют три важных недостатка РБД:

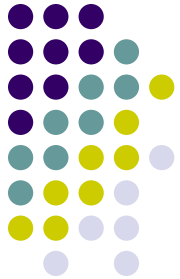
1. РБД вынуждают пользователей представлять иерархические данные (данные с большой вложенностью, составные данные и т.д.) в терминах кортежей многих отношений, что
   a) неудобно
   b) для выборки данных, разбросанных по многим отношениям, РБД должны выполнять трудоемкие операции соединения

Преимущества ООБД



2. РБД предлагают набор примитивных встроенных типов в качестве доменов столбцов отношений, без всяких средств добавления пользовательских типов.
   - Встроенные типы - в основном, численные и символьные. Поэтому для того, чтобы добавить новый тип данных, часто приходится перекраивать архитектуру и код системы.
   - Добавление же нового типа к системе баз данных означает возможность сохранения данных этого типа, запрашивания и обновления таких данных.

Преимущества ООБД

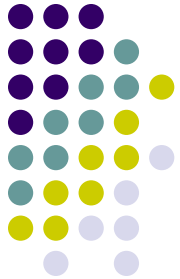


Инкапсуляция объектов в ООБД не накладывает никаких ограничений на типы.

В объектно-ориентированных языках тип атрибута объекта может быть как примитивным, так и произвольно определенным пользователем типом (классом).

Иерархические данные естественным образом представляются благодаря тому, что значение атрибута объекта, в свою очередь, может быть объектом.

Преимущества ООБД



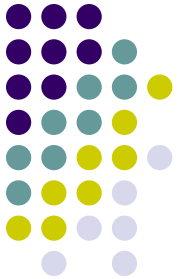
3. Инкапсуляция объектов - основа для хранения и управления программами как объектами, средствами баз данных.

Современные РБД поддерживают "хранение процедуры", т.е. позволяют писать программы на некотором процедурном языке и сохранять в базе данных, а затем их загружать и выполнять.

Однако хранимые процедуры не инкапсулируют данные, поэтому их нельзя соотнести с любым отношением или с любым кортежем.

Кроме того, поскольку РБД не поддерживают наследования, хранимые процедуры не допускают автоматического переиспользования.

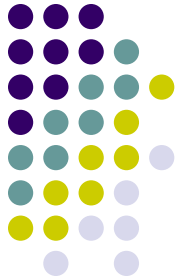
Преимущества ООБД



- Сила объектно-ориентированных концепций проистекает из объединения инкапсуляции и наследования.

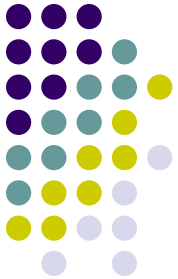
- Поскольку наследование делает возможным совместное использование различными классами одного набора атрибутов и методов, одна и та же программа может работать с объектами, принадлежащими ко всем этим классам.

Преимущества ООБД



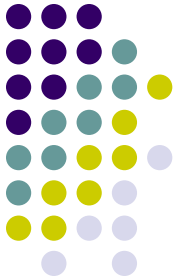
- На этом основывается объектно-ориентированный интерфейс пользователя современных оконных систем. Один и тот же набор программ (открыть, закрыть, создать, выбросить, переместить и т.д.) применим к различным типам данных (изображение, текст, аудио-данные, каталог и т.д.).
- Если пользователи определяют много классов, и каждый имеет много атрибутов и методов, выгода от разделения не только данных, но и программ может быть впечатляющей.
- Их не надо определять каждый раз заново.
- Новые классы могут создаваться добавлением атрибутов и методов к существующим классам, а не модифицируя их атрибуты и методы, что уменьшает вероятность появления новых ошибок.

Недостатки ООБД



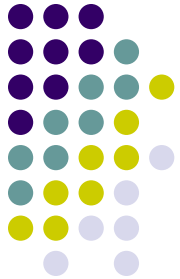
- Отсутствуют мощные непроцедурные средства извлечения объектов из базы.
- Все запросы приходится писать на процедурных языках, проблема их оптимизации возлагается на программиста
- Вместо чисто декларативных ограничений целостности (типа явного объявления первичных и внешних ключей реляционных таблиц с помощью ключевых слов PRIMARY KEY и REFERENCES) или полудекларативных триггеров для обеспечения внутренней целостности приходится писать процедурный код

Недостатки ООБД



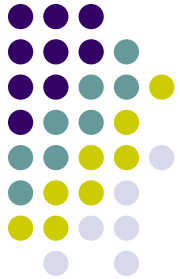
- Оба эти недостатка связаны с отсутствием развитых средств манипулирования данными.
- Эта задача решается двумя способами –
  - расширение ОО-языков в сторону управления данными (стандарт ODMG),
  - либо добавление объектных свойств в реляционные СУБД (SQL-3, а также так называемые объектно-реляционных СУБД).

Подход Стоунбрейкера



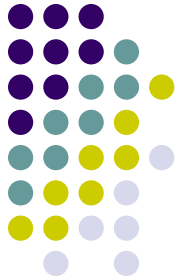
- Стоунбрейкер — главный архитектор Ingres и Postgres, активный участник разработки многих других систем. Он также является сооснователем компании Vertica, создателя одноименной столбцовой СУБД, которая феврале нынешнего года была приобретена Hewlett-Packard. Сейчас Стоунбрейкер — директор по технологиям новой компании VoltDB, предлагающей программную систему распределенной обработки данных.
- По убеждению патриарха СУБД, широко известный язык запросов SQL можно было бы после небольшого усовершенствования использовать с новыми горизонтально масштабируемыми NoSQL-системами, благодаря чему они могли бы обрести полноценную гибкость.
- Этот новый подход Стоунбрейкер называет NewSQL.

Подход Стоунбрейкера: реабилитация SQL баз данных



- Реляционные СУБД — действительно «вымирающий вид»,. Однако виноваты в этом поставщики СУБД, а не SQL как таковой. Реляционные СУБД, медлительны вовсе не из-за того, что поддерживают SQL.
- Большинство коммерческих реляционных систем управления базами данных, доступных на рынке, появились 30 лет назад или больше, Они не были рассчитаны на применение в условиях нынешних транзакционных сред, обрабатывающих гигантские объемы данных. Десятилетиями реляционные СУБД обрастали новыми возможностями сомнительной ценности, в результате превратившись в неимоверных размеров программные «пузыри».

Подход Стоунбрейкера: реабилитация SQL баз данных

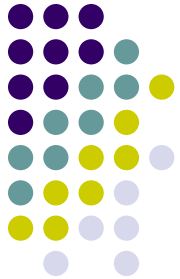


Медлительность баз данных можно отнести на счет нескольких факторов:

- реляционные системы обслуживают буферный пул,
- ведут журналы операций для нужд восстановления,
- управляют блокировками полей данных, предотвращающими их перезапись конкурирующими операциями.

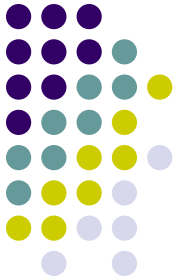
Согласно результатам испытаний все эти задачи отнимают до 96% системных ресурсов.

Причины перехода к NoSQL базам данным



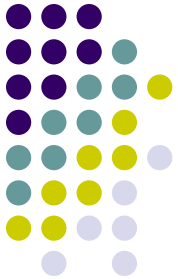
- Реляционные базы данных не обладают необходимой гибкостью.
- Их архитектура, разработанная еще в эпоху перфокарт, реализует фиксированный подход к моделированию данных.
- Если организации нужно добавить новый столбец к таблице, приходится изменять схему базы, что может вызвать определенные трудности. При этом сама схема не всегда точно отражает исходную модель данных
- Существует множество видов данных, которые нельзя разместить в таблице. Табличный формат накладывает слишком много ограничений.

Причины перехода к NoSQL базам данным



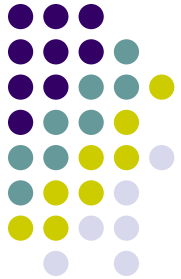
- Реляционные базы данных плохо масштабируются за пределами одиночного сервера.
- Когда объем данных превышает возможности одного сервера, их приходится делить между несколькими системами, а с этим могут быть определенные сложности.
- При исполнении СУБД на группе серверов могут возникнуть трудности с выполнением некоторых операций, например внешних соединений, при которых консолидируются данные из нескольких таблиц.

Недостатки NoSQL баз данных



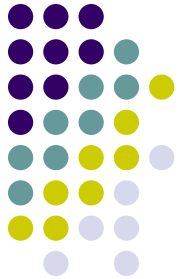
- Ввиду отсутствия поддержки SQL такие системы лишены способности выполнять структурированные запросы с математической точностью. Созданный на основе алгебры отношений и реляционного исчисления, SQL дает гарантию того, что хорошо структурированный запрос, даже очень сложный, захватит из базы все необходимые данные.
- NoSQL-системы не обеспечивают соответствия операций требованиям ACID (atomicity, consistency, isolation, durability — «атомарность, непротиворечивость, изолированность, долговечность») — стандарта, который гарантирует точность выполнения оперативных транзакций средствами СУБД, даже если работа системы прерывалась.

Недостатки NoSQL баз данных



- Средства обеспечения соответствия ACID можно реализовать на уровне приложения, однако написание соответствующего кода, по словам Стоунбрейкера, это «хуже смерти».
- Наконец, у каждой NoSQL-системы есть свой собственный язык запросов, в связи с чем затруднена стандартизация интерфейсов приложений.

Выход – NewSQL базы данных

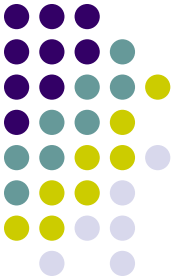


NewSQL обеспечивает гарантии качества выполнения транзакций, свойственные SQL-системам, и при этом обладает масштабируемостью на уровне NoSQL-систем.

Подход NewSQL основан на ряде инновационных архитектурных решений:

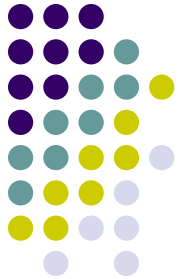
- В NewSQL не используется ресурсоемкий буферный пул, поскольку база данных целиком находится в основной памяти.
- Новый метод также устраняет потребность в краткосрочных блокировках данных, поскольку система исполняется на сервере строго в виде одиночного потока (хотя иные типы блокировки все же будут создавать определенную нагрузку)
- А «дорогостоящие» операции восстановления исключаются за счет применения дополнительных серверов для тиражирования и переключения нагрузки при отказе.

NewSQL базы должны удовлетворять следующим критериям:



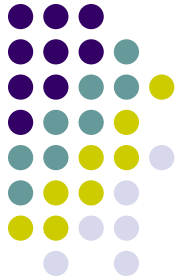
- поддержка реляционной модели и транзакционности
- SQL как основной интерфейс доступа к данным
- горизонтальная масштабируемость
- совершенно новый движок, не унаследованный от классических СУБД
- появились в последние 3-5 лет

Технические характеристики решений NewSQL



- SQL как основной механизм для взаимодействия.
- ACID поддержка транзакций.
- Механизм управления без применения блокировок, таким образом считывающие данные в реальном времени не будут находится в противоречии с записывающими, что исключает конфликт.
- Архитектура, обеспечивающая намного выше производительность узла, чем доступный из традиционных решений RDBMS.
- Удобное масштабирование, способное управлять большим количеством узлов, не перенося узкие места.

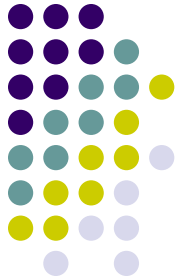
Классификация NewSQL



1. Новые базы данных
2. Новый движок базы данных MySQL
3. Прозрачное объединение в кластеры

Данная Классификация основана на различных подходах, разработанных для того, чтобы ***сохранить SQL интерфейс***, а также решить ***масштабируемость*** и ***производительность***, являющиеся проблемами традиционных решений OLTP.

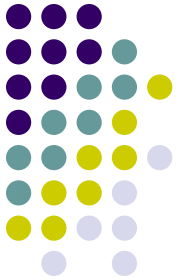
Новые базы данных



NewSQL система разрабатывается полностью с нуля с целью достижения масштабируемости и производительности.

- Одним из ключевых факторов в повышении производительности является использование оперативной памяти или новых видов дисков (флэш-память/SSD), которые являются хранилищем первичных данных.
- Данное решение может осуществляться программно (VoltDB, NuoDB) либо на уровне железа (Clustrix, Translattice)
- Примеры разработок являются Clustrix, NuoDB и Translattice (коммерческие) и VoltDB, (Open Source).

Новые базы данных



Многие NewSQL БД — это in-memory БД.

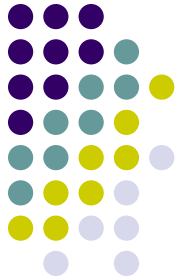
Они хранят все данные в оперативной памяти.

Создатели этих БД считают фатальным недостатком существующих БД стремление все хранить на диске.

Если вам нужно обрабатывать данные быстро, вам нужно хранить их в ОЗУ. А относительно небольшой объем ОЗУ на одной машине можно легко компенсировать, собрав кластер из сотен узлов.

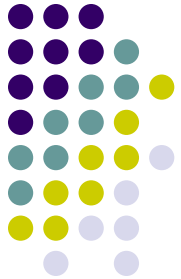
Хранение в ОЗУ не означает, что данные будут потеряны при выключении питания. Все эти БД ведут журнал операций и периодически скидывают снимки данных на диск.

Новый движок базы данных MySQL



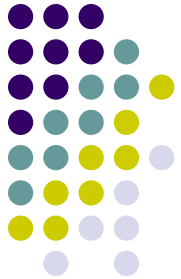
- Чтобы преодолеть проблемы масштабируемости MySQL, было создано ряд движков основанных на MySQL.
- Положительная сторона — использование интерфейса MySQL, но есть плохая сторона — не поддерживается миграция данных из других баз данных (включая старый MySQL).
- Примеры реализации — Xeround, GenieDB (коммерческие) TokuTek; и Akiban, MySQL Группа NDB и др. (opensource).

Новый движок базы данных MySQL



- Самый популярный — TokuDB — движок для MySQL.
- Он использует индексы на так называемых фрактальных деревьях.
- Эти индексы значительно быстрее B-деревьев в случаях, когда индексы не помещаются в оперативной памяти и их приходится читать с диска.

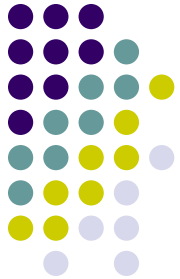
Прозрачное объединение в кластеры



Обычные SQL базы объединяются в кластере из нескольких физических узлов для хранения и обработки больших объемов данных. Тут может быть два подхода:

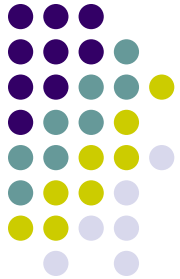
1. Базы данных OLTP сохраняются в своем оригинальном виде, но обеспечивают масштабируемость за счет особенностей группировки данных.
2. Другой подход должен обеспечить прозрачность фрагментации, чтобы улучшить масштабируемость.

Прозрачное объединение в кластеры



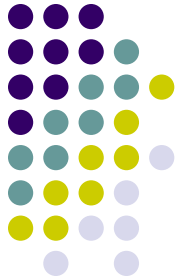
- БД Schooner MySQL, Continuent Tungsten и ScalArc следуют первому подходу, тогда как ScaleBase и dbShards следуют второму подходу.
- Оба подхода позволяют повторное использование существующих наборов и экосистемы, и избегают потребности переписать код или выполнить любые миграции данных.
- Примеры реализаций — ScalArc, Schooner MySQL, dbShards (коммерческий) ScaleBase; и Continuent Tungsten (opensource).

Прозрачное объединение в кластеры



- Сюда можно отнести также MySQL Cluster, Postgres-XC, Oracle RAC и прочие.
- Все промышленные СУБД пытаются собраться в кластер. Все эти решения представляют собой промежуточный слой (middleware), который распределяет запросы между узлами, хранящими данные (обычными БД) и объединяет результаты, собственно, осуществляя фрагментацию.
- Как правило, на запросы накладываются сильные ограничения на использование внешних ключей и джойнов.

NewSQL подходит для фирм, которые планируют:



- миграцию существующих приложений для адаптации к работе с данными большого объема
- разработку новых приложений на хорошо масштабируемых системах OLTP
- использовать на имеющиеся системы, основанные на традиционной системе обработки данных OLTP