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

**УДК 004.652.2**

# Анализ и классификация нереляционных баз данных

**И.А. Козлов**

*Студент, кафедра «Компьютерные системы и сети»,*
*МГТУ им. Н.Э. Баумана, г. Москва, Россия*

*Научный руководитель: Смирнова Е.В., зам. зав. кафедрой по учебной работе, доцент кафедры «Компьютерные системы и сети» МГТУ им. Н.Э. Баумана, г. Москва, Россия*

МГТУ им. Н.Э. Баумана
kozlovilya89@gmail.com
galiam@bmstu.ru

## 1 Введение

На настоящий момент реляционные системы управления базами данных (РСУБД) являются наиболее распространенной технологией для хранения структурированных данных. После публикации статьи Э. Кодда "Реляционная модель данных для больших, совместно используемых банков данных" [1] в 1970 г. такой подход к созданию баз данных, основанный на реляционном исчислении, получил широкое распространение и часто считается единственным возможным решением, обеспечивающим согласованный доступ к данным множества пользователей. Действительно, РСУБД предоставляют широчайшие возможности для выполнения сложных ad hoc запросов, кроме того, использование языка SQL позволяет формировать запросы в декларативной форме и не требует от пользователя описания необходимых программных инструкций.

Благодаря своим возможностям в отношении выполнения запросов и управления транзакциями реляционные СУБД еще недавно казались идеальным решением практически для любой задачи. Однако для эффективного использования этой технологии необходимо выполнение ряда требований к данным: наличие строго определенной структуры, четких взаимосвязей, возможность выделения ключей. Разреженные же данные с нечеткой структурой плохо соответствуют реляционной модели.

Новые проблемы РСУБД обнаружились с наступлением эры «больших данных». Перед современными бизнес-приложениями зачастую стоит задача обработки терабайтов и даже петабайтов данных. Для решения этой проблемы широко используются распределенные вычисления, что предъявляет к СУБД новые требования: хорошая масштабируемость и возможность распределённой обработки транзакций [2]. При возникновении необходимости обработки больших объемов данных РСУБД сталкиваются с рядом проблем: так, в распределенных системах выполнение транзакций и операций объединения, затрагивающих большое количество узлов кластера, сложно и неэффективно [3]. Это стало причиной роста популярности нереляционных баз данных, позволяющих добиться большей гибкости, лучшей масштабируемости и высокой отказоустойчивости. Такие базы данных получили название NoSQL.

Первоначально этот термин использовался для обозначения нереляционных баз данных, не поддерживающих SQL. Но он также может трактоваться как «Not Only SQL» [4], что выражает суть новой парадигмы: одна технология не может удовлетворять всем требованиям. Действительно, NoSQL-СУБД решают многие проблемы реляционных баз данных и упрощают работу с большими данными, но при этом приходится жертвовать характерными для РСУБД возможностями в отношении управления транзакциями, индексирования и использования ad hoc запросов [5]. Таким образом, для каждой конкретной задачи выбор используемой технологии следует осуществлять в зависимости от требований к проектируемой БД.

Рассмотрим основные причины перехода от РСУБД к NoSQL-технологиям.

# 2 Проблемы реляционных СУБД

## 2.1 Распределенная обработка транзакций

Все ведущие реляционные СУБД предоставляют широкие возможности по работе с транзакциями. Транзакция – логическая единица работы с базой данных, группа операций, которая может быть либо выполнена целиком, либо не выполнена вообще. Основная сложность работы с транзакциями в распределенных базах данных состоит в том, что необходимые обновления зачастую затрагивают несколько узлов кластера. Долгое время наиболее распространённым набором требований к транзакциям был набор ACID, однако в последнее время часто используется его альтернатива – набор BASE, получивший широкое применение в NoSQL технологиях [6].

### 2.1.1 Требования ACID

Важным свойством реляционных баз данных является их способность удовлетворять требованиям ACID. ACID представляет собой набор требований к транзакционной системе, обеспечивающих её надежную и предсказуемую работу:

- Атомарность (Atomicity): каждая транзакция осуществляется по принципу «всё или ничего». Сбой на каком-либо этапе транзакции означает сбой всей транзакции в целом. При этом данные в базе остаются в состоянии на начало транзакции;
- Согласованность (Consistency): каждая транзакция представляет собой переход из одного корректного состояния в другое. Данные, записываемые в базу в рамках транзакции, удовлетворяют всем установленным ограничениям;
- Изолированность (Isolation): на результат транзакции не должны оказывать влияние транзакции, выполняющиеся параллельно с ней;
- Надежность (Durability): изменения, успешно внесенные в базу, остаются в сохранности вне зависимости от сбоев в оборудовании, обесточивания системы и прочих проблем.

Однако не для всех задач требуется выполнение всех приведенных требований. Примером может служить приложение ConnectNow от Adobe, хранящее три копии данных о пользовательской сессии, не подвергая реплицируемые данные всем проверкам на согласованность.

Еще одна проблема реляционных баз данных, связанная с требованиями ACID, возникает при создании распределенных систем. Для выполнения первых трёх требований в РСУБД часто используется централизованное управление блокировками. Если приложение пытается получить доступ к данным, обрабатываемым другим приложением, ему необходимо ждать завершения текущей транзакции. Для распределенной системы централизованный менеджер блокировок является узким местом, поскольку все узлы системы должны взаимодействовать с ним во время каждой своей операции.

### 2.1.2 Теорема CAP

Основной областью применения нереляционных баз данных являются распределенные системы. Акроним CAP отражает три важных свойства таких систем:

- Согласованность (consistency): после выполнения каждой операции система находится в согласованном состоянии, то есть данные в различных узлах не противоречат друг другу;
- Доступность (availability): каждый запрос получает отклик, даже в случае выхода из строя узлов кластера;

- Устойчивость к разделению (partition tolerance): сохранение работоспособности при расщеплении системы на несколько изолированных частей.

Теорема CAP утверждает, что лишь два из этих трёх требований могут быть выполнены одновременно [7]. Поэтому при построении распределенной базы данных приходится сделать выбор в пользу двух из перечисленных характеристик и пожертвовать третьей [8]. Выбор в каждом случае осуществляется исходя из конкретной задачи и требований к проектируемой базе. В зависимости от выбранных свойств СУБД разделяются на 3 класса: CA(consistency + availability), CP(consistency + partition tolerance) и AP(availability + partition tolerance) (рисунок 1). К первой категории относятся традиционные РСУБД, однако от распределенных систем обычно требуется устойчивость к разбиению, а потому приходится идти на уступки в отношении согласованности или доступности данных. Поэтому большинство NoSQL технологий относится к классу CP или AP.

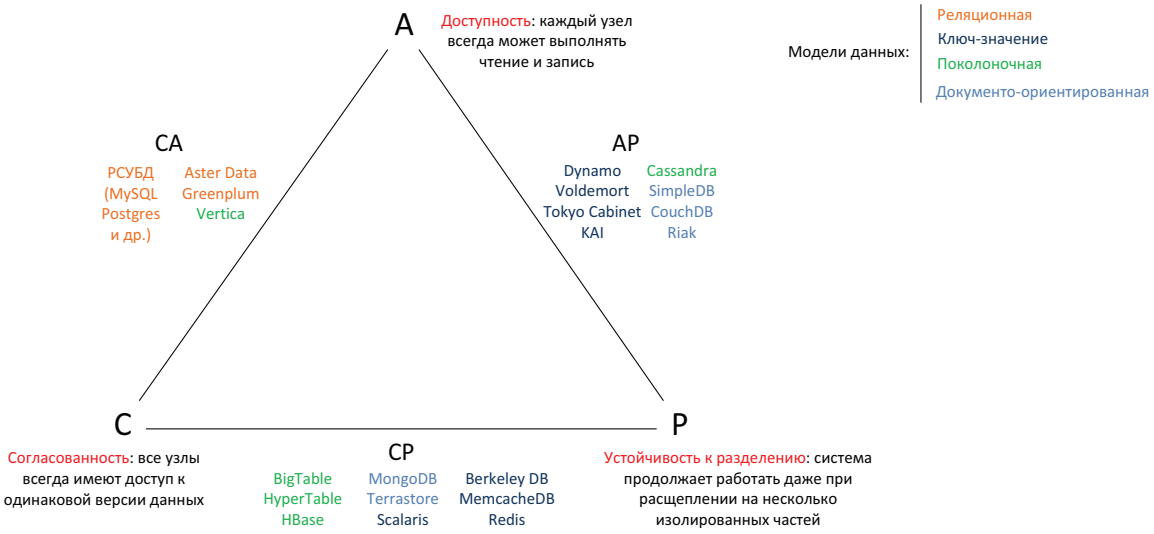



Рис. 1. Классы СУБД с точки зрения CAP-теоремы

### 2.1.3 Требования BASE

Чтобы выполнить требования ACID в распределенной системе необходимо использовать механизм двухфазовой фиксации (2PC – two-phase commit). При этом процедура фиксации изменений в базе данных состоит из двух этапов:

- Фаза голосования: координатор опрашивает каждую из баз данных, участвующих в транзакции, о готовности фиксации;
- Фаза фиксации: когда все узлы готовы, осуществляется фиксация. После этого обновленные данные становятся доступными пользователям.

Такой подход позволяет добиться согласованности, но как быть, если более важным является требование доступности? Решением этой проблемы является

альтернативный набор требований BASE (Basically Available, Soft-state, Eventually consistent — базовая доступность, неустойчивое состояние, согласованность в конечном счёте) [9]. В отличие от строгой согласованности, входящей в набор ACID, BASE включает более слабое требование – согласованность в конечном счёте. Это условие подразумевает наличие окна несогласованности, когда различные пользователи могут иметь доступ к разным версиям данных. Однако такой подход позволяет получить ряд преимуществ:

- Доступность: данные становятся доступными сразу после записи на узел;
- Отказоустойчивость: сбой одного из узлов не ведёт к сбою системы в целом (данные будут недоступны лишь для пользователей узла, на котором произошел сбой).

Наборы ACID и BASE представляют собой возможные варианты требований к распределенной системе, и выбор одного из них зависит от того, какое из свойств играет большую роль для разрабатываемой системы: согласованность или доступность. Большинство NoSQL-СУБД используют BASE-подход, поскольку отказ от обеспечения строгой согласованности позволяет добиться значительного преимущества в производительности и масштабируемости.

## 2.2 Масштабируемость

Одной из основных проблем реляционных баз данных, решение которых может быть достигнуто с помощью технологий NoSQL, является масштабируемость. Она представляет собой способность системы справляться с увеличением нагрузки посредством наращивания её ресурсов. Существует два основных подхода к масштабированию:

- Вертикальное масштабирование: увеличение производительности системы за счёт использования более мощного оборудования;
- Горизонтальное масштабирование: необходимый уровень производительности достигается посредством разбиения системы на структурные компоненты и распределения их по отдельным физическим машинам.

Горизонтальное масштабирование – более сложное решение, но оно имеет целый ряд преимуществ: гибкость, относительно низкая стоимость и высокий потенциал для дальнейшего увеличения кластера. В контексте баз данных этот подход заключается в распределении данных между несколькими базами, находящимися на различных узлах кластера. К основным способам горизонтального масштабирования баз данных относятся репликация и шардинг [10].

### 2.2.1 Репликация

В распределенных базах данных репликация состоит в хранении одних и тех же данных на нескольких узлах. Это позволяет увеличить пропускную способность чтения данных, поскольку операции чтения могут быть распределены между множеством машин. Кроме того, это повышает отказоустойчивость системы.

С другой стороны, репликация имеет недостаток, связанный с операциями записи. Каждая из таких операций должна быть выполнена для каждого узла, предназначенного для дублирования данных. При этом возможны следующие варианты:

- Данные становятся доступными только после записи на каждый из узлов;
- После поступления данные становятся доступными вне зависимости от того, были ли они записаны на все узлы.

В зависимости от выбранного варианта приходится пожертвовать либо доступностью, либо согласованностью данных.

### 2.2.2 Шардинг

Другим подходом к горизонтальному масштабированию является шардинг (Sharding). Он представляет собой разбиение данных в базе на отдельные порции и распределение их по узлам. Для реляционных баз данных при шардинге логически независимые записи таблицы хранятся на различных узлах. Таким образом, на разных серверах будет находиться таблица с одинаковой структурой, но разными данными. Для осуществления распределения данных по узлам может быть использована хеш-функция, применяемая к первичному ключу записи для определения сервера, на котором она должна быть размещена.

Чем больше узлов используется в шардированной базе данных, тем выше вероятность того, что один из них выйдет из строя. Поэтому шардинг обычно используется в сочетании с репликацией, что позволяет повысить отказоустойчивость системы.

Основным преимуществом шардинга является возможность добавления и удаления узлов из кластера по мере изменения нагрузки без необходимости каких-либо изменений в приложении. Однако есть и существенный недостаток: некоторые операции, типичные для баз данных, становятся слишком сложными и медленными. Одна из наиболее важных операций для реляционных баз данных – операция соединения, служащая для материализации логической связи между данными базы. Результатом операции является отношение (таблица), получаемая как декартово произведение исходных отношений. В распределенных системах оба исходных набора данных хранятся на множестве узлов, и для осуществления соединения потребуется выполнить большое количество запросов к

каждому из этих узлов, что вызовет значительный рост сетевого трафика. По этой причине в базах данных с возможностью шардирования обычно не поддерживается операция соединения.

Сложности с распределенным выполнением операции соединения делают проблематичным горизонтальное масштабирование реляционных баз данных. Некоторые специалисты в области распределенных вычислений полагают, что использование шардирования ведёт к потере всех преимуществ РСУБД [11]. Причиной является то, что многие существующие СУБД разрабатывались без расчета на использование горизонтального масштабирования – эта возможность была добавлена в готовые системы позднее. В то же время большинство NoSQL-баз данных изначально включают шардинг как одну из ключевых возможностей. Некоторые из них даже поддерживают автоматическое партиционирование и балансировку нагрузки на узлы кластера [12].

Таким образом, использование NoSQL-технологий позволяет решить ряд проблем, присущих реляционным базам данных и связанных, прежде всего, с распределенной обработкой данных. Ориентация на распределенные системы и отказ от набора требований ACID поставили перед разработчиками БД новые задачи и привели к появлению оригинальных решений. И хотя в настоящее время существует множество типов нереляционных баз данных, различающихся по характеристикам, возможностям и назначению, можно выделить ряд концепций и методов, характерных для NoSQL-технологий в целом. Рассмотрим некоторые из этих особенностей.

## 3 Особенности нереляционных баз данных

### 3.1 Консистентное хеширование

Для многих NoSQL-СУБД шардинг представляет собой одну из ключевых возможностей. При этом важной задачей является выбор механизма для определения узла, на котором должна быть размещена та или иная запись. Очевидным решением является применение простой хеш-функции к ключу записи с последующим делением результата на количество узлов в кластере:

$$index = hash(key) \bmod n,$$

где *index* – индекс целевого узла, *key* - ключ записи, *n* - число узлов в кластере. Проблема такого подхода в том, что при добавлении/удалении хотя бы одного узла потребуется перераспределение всех данных в кластере, ведь значение *index* для любой записи может стать иным. Решением проблемы является механизм консистентного хеширования, используемый в некоторых NoSQL-базах данных, таких как Dynamo и

Project Voldemort. Идея состоит в присвоении узлам кластера идентификаторов из области значений хеш-функции. Если представить область значений функции в виде окружности, каждый узел отвечает за дугу окружности (соответствующую некоторому интервалу значений хеш-функции) между идентификаторами самого узла и его ближайшего соседа против часовой стрелки. Таким образом, чтобы получить доступ к объекту с ключом *Key* необходимо обратиться к узлу, ближайшему по часовой стрелке от *Key*.

На рисунке 2 приведен пример использования консистентного хеширования. Окружность представляет собой нормированную (приведенную к отрезку [0,1]) область значений хеш-функции, замкнутую в кольцо (максимальный элемент соединен с минимальным). Узел A отвечает за интервал, выделенный красным. При необходимости чтения/записи объекта, попадающего в этот интервал (например, объекта 1), необходимо обратиться к узлу A.

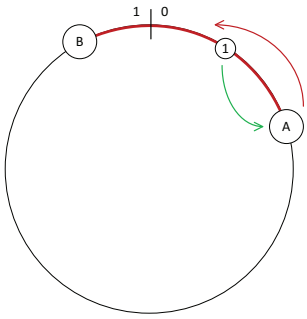
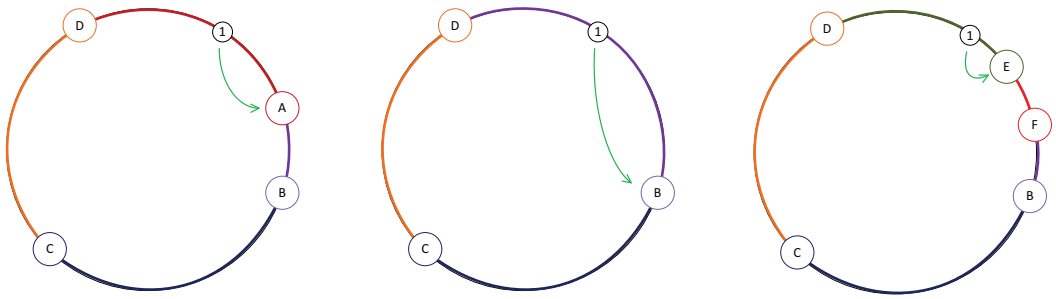



Рис. 2. Консистентное хеширование

При таком подходе изменения зон ответственности при добавлении/удалении компьютеров затрагивают наименьшее число узлов:

- Узел берет на себя ответственность за интервал значений, принадлежащий его ближайшему соседу против часовой стрелки, если тот покидает кластер (рисунок 3,*б*);
- Появившийся в кластере новый узел принимает на себя ответственность за ближайший против часовой стрелки интервал (его ближайший сосед по часовой стрелке, соответственно, теряет контроль над этим интервалом) (рисунок 3,*в*).




*а* *б* *в*

Рис. 3. Добавление и удаление узлов

В исходном виде механизм консистентного хеширования имеет ряд недостатков. Так, узлы распределены по окружности случайным образом, следовательно, размеры зон ответственности могут сильно различаться (например, зоны узлов B и C на рисунке 3,*а*). Кроме того, при удалении узла из кластера данные, хранящиеся на нём, становятся недоступными. При добавлении же новых компьютеров в кластер зона ответственности некоторых узлов уменьшается, и они перестают отвечать за хранящиеся на них файлы (то есть, запросы на доступ к этим файлам будут адресоваться уже другим узлам). В [13] предложены методы решения этих проблем:

- Использование виртуальных узлов. Идея состоит в создании большого количества виртуальных узлов, равномерно распределенных по кольцу. Каждый физический узел отвечает за некоторый набор виртуальных (мощность этих наборов может быть различной для физических узлов и зависеть от их производительности);
- Репликация. Запись данных производится не только на целевой узел, но и на несколько последующих (при обходе кольца по часовой стрелке).

## 3.2 Распределенная обработка данных с помощью MapReduce

Основной областью применения нереляционных баз данных являются распределенные системы, а потому разработчики многих NoSQL-СУБД уделяют особое внимание поддержке распределенных вычислений. Одним из наиболее популярных подходов к распределенной обработке данных является модель MapReduce, разработанная Google [14]. Использование этой парадигмы при создании распределенных приложений позволяет программисту избежать необходимости написания специального кода для распараллеливания и синхронизации. MapReduce предполагает разбиение задачи на два шага:

- *Map*-шаг: главный узел разбивает входные задачи на части и передает остальным узлам для предварительной обработки;
- *Reduce*-шаг: свертка результатов предварительной обработки. Главный узел получает данные от остальных узлов и формирует окончательный результат выполнения задачи.

Для использования фреймворка MapReduce программисту необходимо написать функции *Map* и *Reduce.* Эти функции работают с данными, структурированными в виде пар «ключ-значение». *Map* принимает на вход пару одного типа и возвращает набор пар другого типа (промежуточный результат):

$$Map(Key,Value) \rightarrow list(iKey,iValue),$$

где *(Key,Value)* – одна запись из набора входных данных, *(iKey,iValue)* – промежуточная пара ключ/значение. Каждая пара, сгенерированная *Map*-функцией, отправляется на некоторый узел для выполнения *reduce*-шага. Целевой узел определяется посредством применения хеш-функции к ключу пары (*iKey*). На каждом узле полученные пары группируются по ключу. Затем к каждой группе применяется функция *Reduce*, при выполнении которой формируется окончательный результат *fValue*:

$$Reduce(iKey,list(iValue)) \rightarrow list(fValue)$$

На рисунке 4 приведен классический пример использования модели MapReduce для подсчёта частоты встречаемости слов в наборе документов. *Map*-функция возвращает набор пар «слово - количество» для каждого документа. Затем пары группируются по ключу, в результате чего в каждую группу попадают пары, соответствующие одному и тому же слову. *Reduce*-функция суммирует значения всех пар в группе и возвращает окончательный результат, равный количеству вхождений этого слова в текст всех документов набора.

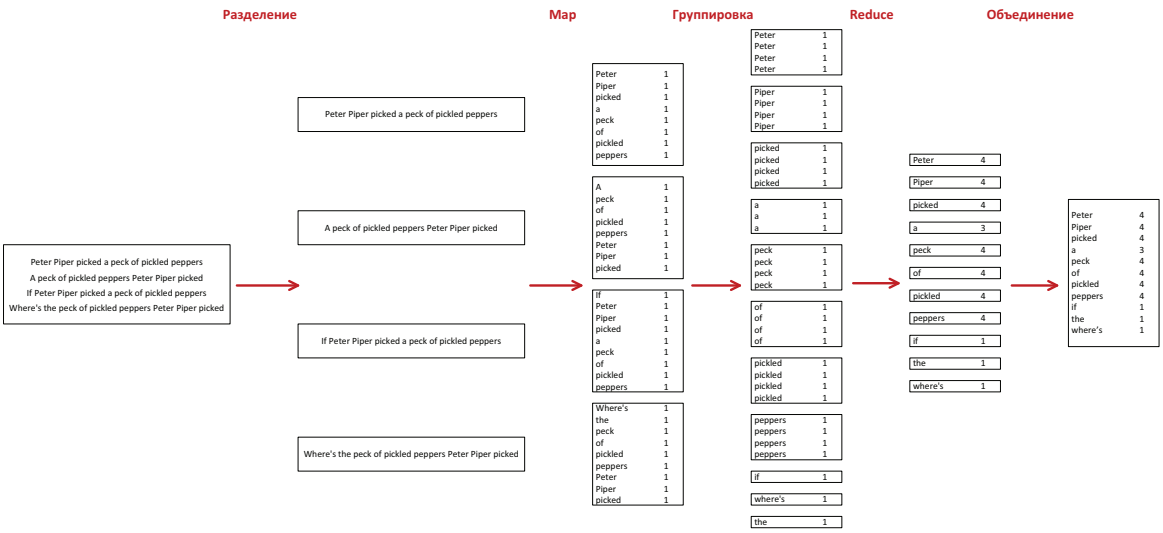



Рис. 4. Использование MapReduce для подсчёта частоты встречаемости слов в наборе документов

Модель MapReduce, предоставляющая широкие возможности для распределенной обработки данных, внедрена в ряд NoSQL-баз данных, таких как MongoDB и CouchDB. В СУБД для определения узлов, выполняющих операции *Map* и *Reduce* для пар (*Key,Value*) исходных данных и (*iKey,iValue*) промежуточных результатов соответственно, может быть использована консистентная хеш-функция (рисунок 5). Возможность узлов самостоятельно определять, куда следует передать данные для последующей обработки, снимает потребность в узле-координаторе.

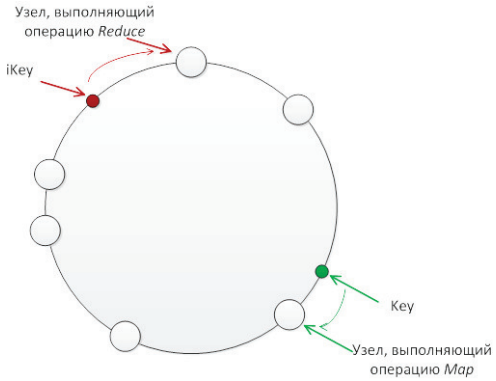



Рис. 5. Выполнение *Map* и *Reduce* с использованием консистентной хеш-функции

## 3.3 Управление конкурентным доступом с помощью многоверсионности

Использование BASE-подхода с отказом от строгой согласованности предполагает, что изменения в данных, внесенные на одном узле, постепенно распространяются к остальным. Но что если изменения внесены на нескольких узлах? Какой вариант данных должен быть, в конечном счёте, принят всеми узлами? Одним из подходов к решению этого вопроса является использование механизма MVCC (Multiversion Concurrency Control - Управление конкурентным доступом с помощью многоверсионности). Он позволяет множеству процессов осуществлять параллельный доступ к данным без риска порчи данных и возможности клинчей.

Для обеспечения согласованности каждый элемент данных сопровождается меткой времени последнего изменения или номером версии. При чтении процесс получает не только сами данные, но и метку. Когда процесс пытается изменить данные, он сравнивает текущее значение метки в БД со значением, полученным им при чтении. При их совпадении изменение может быть осуществлено, при этом номер версии элемента данных инкрементируется. Отсутствие совпадения говорит о том, что другой процесс уже внёс свои изменения. В этом случае рассматриваемый процесс должен быть прерван или перезапущен.

Однако обнаружение конфликтов при выполнении записи возможно только в том случае, если конкурирующие процессы пытаются изменить данные на одном и том же узле. В противном случае конфликт может быть обнаружен только после синхронизации данных между узлами. Для детектирования конфликта нужно знать:

- были ли изменения данных параллельными или последовательными;
- какая из версий была создана раньше.

Для этого могут быть использованы временные метки, но при этом необходимо, чтобы узлы кластера были синхронизированы во времени. Если это требование не выполняется, для обнаружения конфликтов используются векторные часы. Каждому элементу данных ставится в соответствие не одна метка, а вектор $V$ меток, по одной на каждый узел кластера. Для $i$-ого узла $i$-ый элемент вектора $V_i[i]$ означает текущую версию данных, а $j$-ый элемент $V_i[j]$ (где $j \neq i$) – версию на момент синхронизации с $j$-ым узлом. При взаимодействии узлов выполняется синхронизация: из двух векторов выбирается наибольший (полагается, что $V_i > V_j$, если $\forall k: V_i[k] \geq V_j[k]$), если таковой имеется. В противном случае фиксируется конфликт, который должен быть разрешен клиентским приложением.

## 3.4 Модели запросов

По сравнению с реляционными базами данных NoSQL-технологии предоставляют существенно более скромные возможности в отношении построения запросов. В большинстве NoSQL-СУБД поддержка сложных запросов ограничена ради увеличения производительности и масштабируемости. Так, хранилища «ключ-значение» обычно предоставляют лишь возможность поиска по первичному ключу. Другие базы данных, например, документо-ориентированные, поддерживают использование сложных запросов, но лишь предопределенных. Однако существует несколько способов расширения возможностей NoSQL-технологий:

1. Использование дополнительной SQL-базы данных, в которую копируются значения атрибутов, по которым предполагается производить поиск. Возможности этой реляционной БД в отношении построения запросов используются для осуществления поиска, результатом которого являются первичные ключи записей в исходной базе.

2. Для хранилищ «ключ-значение» используются префиксные деревья, где каждый путь от корня до листа содержит префиксы некоторого ключа, а сам лист – значение, соответствующее этому ключу (рисунок 6).

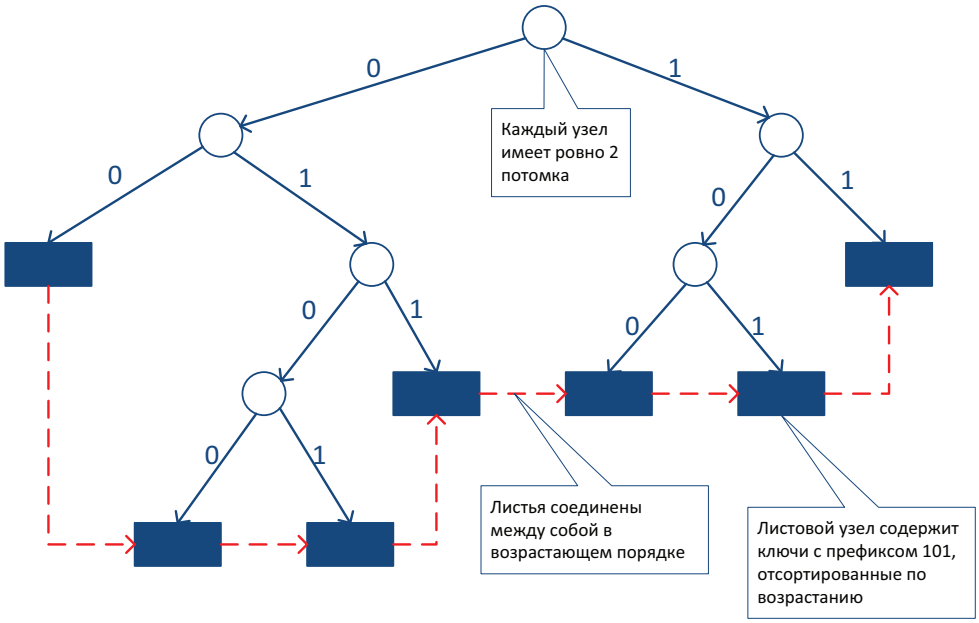



Рис. 6. Двоичное префиксное дерево

При выборе механизма построения запросов необходимо явным образом определить, как должны выполняться операции AND и OR. Простейшая реализация операции AND состоит в отправке всех записей, найденных по каждому из запросов, на сервер, где выполняется операция пересечения множеств. Если наборы данных велики, это вызовет значительный рост нагрузки на сеть. Поэтому следует использовать более эффективные подходы, такие как:

- Использование фильтра Блума для отбрасывания записей, гарантированно не принадлежащих результирующему множеству;
- Использование пошаговой выборки, если клиентскому приложению не требуется всё результирующее множество сразу.

## 4 Классификация нереляционных баз данных

В последнее время базы данных NoSQL активно развивались веб-компаниями. Каждая из них решала свою задачу и создавала технологию в соответствии со своими требованиями к производительности, масштабируемости и набору необходимых функций. Некоторые воспользовались идеями, реализованными Amazon и Google в базах данных Dynamo и Bigtable, другие предложили абсолютно новые концепции.

Обилие и многообразие подходов к созданию нереляционных баз данных вызвало появление различных способов их классификации. Один из наиболее полных вариантов, предложенный Стефаном Йеном [15], приведен в таблице.

Классификация NoSQL-СУБД

| Тип базы данных | Пример реализации |
|---|---|
| Хранилища «ключ-значение» | RAMCloud |
| Хранилища «ключ-значение» с кэшированием | Memcached |
| Упорядоченные хранилища «ключ-значение» | Tokyo Tyrant |
| Хранилища «ключ-значение» с согласованностью в конечном счёте | Dynamo |
| Хранилища структур данных | Redis |
| Кортежные хранилища | Apache River |
| Объектно-ориентированные БД | DB40 |
| Документо-ориентированные БД | MongoDB |
| Колоночные БД | Cassandra |

## 4.1 Хранилища «ключ-значение»

Хранилища «ключ-значение» - базы данных, построенные на основе простой модели ассоциативного массива, при которой пользователь записывает данные в базу и извлекает их из неё по соответствующему ключу. Современные хранилища «ключ-значение» в основном отдают предпочтение масштабируемости в ущерб согласованности, также они обычно не предоставляют широких возможностей аналитической обработки данных. Сложные ad hoc запросы с использованием операции объединения, как правило, не поддерживаются – лишь операции добавления и удаления записи, а также поиска по ключу.

Хранилища «ключ-значение» существуют несколько десятилетий (так, Berkeley DB была создана в 1986 г.), однако большое количество баз данных этого класса появились в течение нескольких последних лет под влиянием Dynamo от Amazon.

Наиболее известными СУБД этого типа являются:

- Project Voldemort – технология, используемая компанией LinkedIn. Одной из ключевых особенностей системы является высокая доступность операций записи, для чего поддерживается возможность параллельного изменения данных (используется управление конкурентным доступом с помощью многоверсионности (MVCC) с алгоритмом векторных часов).
- Dynamo. В основу СУБД от Amazon положен принцип «сбой оборудования – стандартный режим работы системы», поэтому особое внимание при разработке было уделено отказоустойчивости. Dynamo использует алгоритм партиционирования с использованием консистентного хеширования. При записи данных осуществляется их репликация на несколько ближайших (с точки зрения значений хеш-функции) узлов.

- Redis. В отличие от других хранилищ «ключ-значение» позволяет осуществлять поиск не по конкретному ключу, а по их диапазону (поиск по диапазону чисел или по регулярному выражению). Также позволяет хранить в качестве значений списки и множества.
- Scalaris. Также позволяет осуществлять поиск по диапазону ключей (поскольку они хранятся в лексикографическом порядке). В отличие от большинства СУБД типа «ключ-значение» удовлетворяет требованиям ACID (включая строгую согласованность).

## 4.2 Документо-ориентированные базы данных

Документо-ориентированные базы данных позволяют хранить данные более сложной структуры, чем хранилища «ключ-значение». Центральным понятием этого подхода является «документ». Он инкапсулирует данные, как правило, в виде одного из широко распространенных форматов: XML, JSON, YAML. Также в этом качестве могут выступать и документы в более традиционном смысле – файлы PDF или Microsoft Word.

Документо-ориентированные СУБД обычно поддерживают вторичные индексы, а также вложенные документы и списки. Такие базы данных больше, чем хранилища «ключ-значение», напоминают РСУБД, но имеют ряд важных отличий: данные не привязаны к жесткой схеме, а набор полей (ключей, секций) у документов может различаться.

Так же, как и другие NoSQL-системы, документо-ориентированные базы данных обычно не удовлетворяют ACID-требованиям. Однако они предоставляют относительно широкие возможности построения запросов, позволяя осуществлять поиск не только по ключу, но и по содержимому документов.

Двумя крупнейшими представителями этого класса являются CouchDB и MongoDB.

- CouchDB – NoSQL-СУБД, разработанная Apache. Позволяет использовать в качестве полей документа скалярные значения (числа, строки), а также списки и вложенные документы. Возможно использование вторичных индексов, которые должны создаваться явным образом. Индексы реализованы с помощью Вдеревьев, что позволяет упорядочивать результаты выборки, а также осуществлять выборку по диапазону значений. Запросы могут выполняться параллельно к множеству узлов – для этого используется MapReduce. Масштабируемость достигается посредством репликации, шардинг не используется. Также CouchDB позволяет управлять параллельным доступом к БД с помощью MVCC.

- MongoDB имеет множество общих черт с CouchDB, такие как вторичное индексирование и механизм запросов. Но есть и ряд важных отличий. Репликация в MongoDB используется для повышения отказоустойчивости, масштабируемость же достигается с помощью автоматического шардинга. Технология MVCC не используется, вместо этого поддерживается механизм атомарных операций над полями. Также MongoDB использует MapReduce для выполнения агрегации данных из различных документов.

### 4.3 Колоночные базы данных

Идея хранить и обрабатывать данные по колонкам, а не по рядам, зародилась в области бизнес-аналитики, где колоночная обработка данных в системах с массово-параллельной архитектурой используется для создания высокопроизводительных приложений (одним из наиболее известных продуктов в этой области является Vertica). Стремительное развитие баз данных этого класса началось после появления BigTable. Под влиянием этой СУБД от Google появились HBase, Hypertable и Cassandra.

СУБД данного типа хранят данные в строках и столбцах, причем масштабируемость достигается посредством разделения между узлами как строк, так и столбцов:

- строки разделяются между узлами с помощью шардинга по первичному ключу;
- для разделения колонок используется группирование столбцов. При этом каждая из групп колонок обрабатывается как отдельная таблица (так, для каждой из них шардинг осуществляется отдельно).

Колоночные хранилища, как правило, позволяют использовать лишь скалярные значения полей – вложенные объекты не поддерживаются. Рассмотрим основных представителей этого класса СУБД:

- BigTable. В системе от Google с каждым элементом данных ассоциированы 3 значения: имя строки, имя колонки и временная метка. Несмотря на использование строк и столбцов, BigTable не является РСУБД, а может быть охарактеризован как разреженный распределенный многомерный ассоциативный массив. Строковые ключи хранятся в лексикографическом порядке и используются для партиционирования данных. Колоночные ключи разбиваются на группы по префиксу ключа. Временная метка показывает время изменения данных. Данные в БД отсортированы по убыванию значений временных меток, чтобы наиболее свежая информация была доступна в первую очередь.

- HyperTable во многом похожа на BigTable. Эта СУБД также осуществляет группировку колонок и использует временные метки и MVCC. Однако для функционирования HyperTable требуется наличие распределенной файловой системы (например, Hadoop Distributed File System).
- Cassandra также имеет множество характеристик, типичных для других колоночных БД. Однако есть и отличия: так, она использует концепцию «суперколонок», позволяющую ввести дополнительный уровень группировки столбцов. Группа столбцов содержит либо колонки, либо суперколонки (но не те и другие вместе). Суперколонки в свою очередь содержат некоторое множество столбцов. Так же, как и в других системах, строки не ограничены жесткой схемой и могут иметь произвольный набор полей.

## 5 Иерархические СУБД на примере IBM IMS

Важное место среди нереляционных баз данных занимают иерархические СУБД. В отличие от многих NoSQL-технологий, такой подход появился до возникновения реляционной модели: в 1968 г. компания IBM выпустила иерархическую СУБД IMS.

### 5.1 Построение иерархической модели данных

В основе систем этого класса лежит иерархическая модель данных. Она представляет собой совокупность записей, соединенных ссылками. Каждая запись является набором полей (атрибутов), каждое из которых содержит некоторое значение. Ссылка соединяет две записи друг с другом, причем при соединении ссылками записи образуют набор деревьев (в отличие от сетевой модели, где записи могут организовывать произвольные графы).

Рассмотрим диаграмму «сущность-связь», показывающую связь «один-ко-многим» между двумя сущностями (рис. 7,*а*). Пусть каждая книга может быть написана лишь одним автором, но каждый писатель может быть автором множества книг. Соответствующая иерархическая модель данных представлена на рис. 7,*б* и представляет собой дерево, корнем которого является запись, соответствующая писателю, а листьями – записи, соответствующие его книгам.

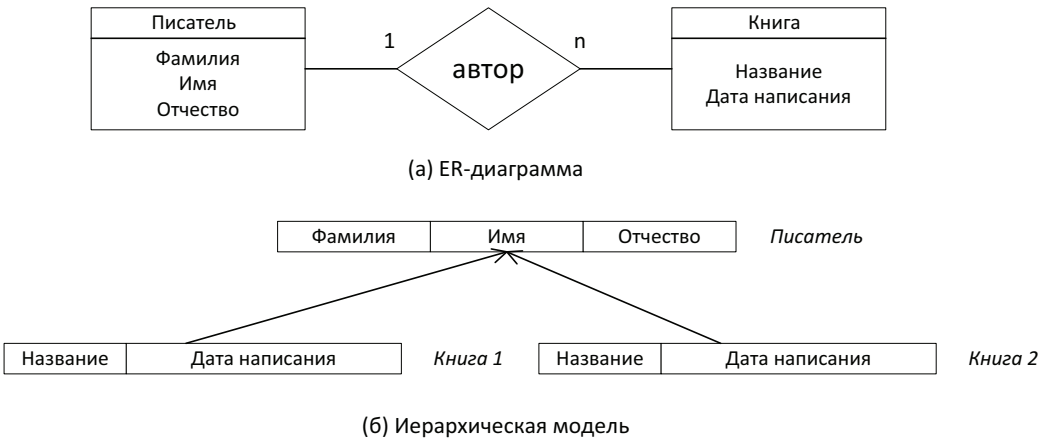



Рис. 7. Иерархическая модель данных для отношения «один-ко-многим»

Несколько сложнее осуществляется преобразование для отношения «многие-ко-многим» (рис. 8,*а*), поскольку лишь отношения «один-к-одному» и «один-ко-многим» могут быть напрямую отображены на иерархическую модель. Если в нашем примере рассмотреть возможность соавторства, модель будет состоять уже из двух деревьев, каждое из которых содержит записи типа «книга» и «писатель». Но если в одном из деревьев корнем является запись, соответствующая писателю, то в другом – запись, соответствующая книге (рис. 8,*б*).

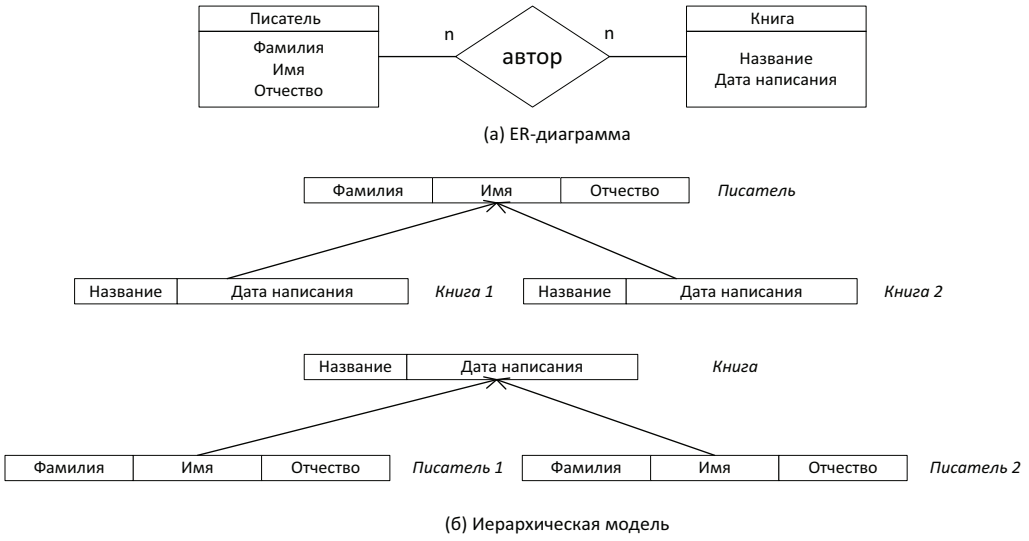



Рис. 8. Иерархическая модель данных для отношения «многие-ко-многим»

## 5.2 Система управления иерархическими базами данных IBM IMS

Старейшей и наиболее широко используемой иерархической СУБД является система IMS от IBM. Разработчики IMS были одними из первых, кому пришлось столкнуться с проблемами целостности, параллельного доступа к данным, а также эффективного выполнения запросов. Рассмотрим основные особенности этой СУБД.

Поскольку производительность играет важнейшую роль в больших системах, IMS предоставляет разработчику базы данных широкий спектр возможностей в области определения данных. Разработчик определяет физическую иерархию как схему БД. Он также может создавать подсхемы (представления), конструируя логическую иерархию типов записей, составляющих БД.

Оригинальная версия IMS была разработана до появления первых подходов к управлению параллельным доступом, поэтому ранние версии системы имели простую стратегию в этом отношении. В каждый момент времени мог быть запущен лишь один пишущий процесс. При этом не было ограничения на число запущенных процессов, осуществляющих чтение. Единственной опцией, доступной процессам, требовавшим большую защиту от аномалий, связанных с параллельным доступом, было предоставление таким процессам эксклюзивного доступа к БД. В более поздних версиях IMS был предложен сложный механизм, обеспечивающий как более совершенное управление параллельным доступом, так и возможность восстановления транзакций. Важность этих нововведений обусловлена переходом большого числа пользователей на использование онлайн-транзакций.

## 6 Использование нереляционных баз данных в МГТУ им. Н.Э. Баумана

Каждый университет стремится постоянно совершенствовать свои образовательные процессы. Так, в МГТУ им. Н.Э. Баумана в последнее время появился целый ряд решений в этой области, в частности, в направлении автоматизации образовательных процессов (с использованием системы управления обучением Moodle) и внедрения новых подходов к оценке качества подготовки специалистов [16]. Еще одним перспективным направлением является организация управления образовательными процессами с использованием семантических сетей и онтологий (рисунок 9).

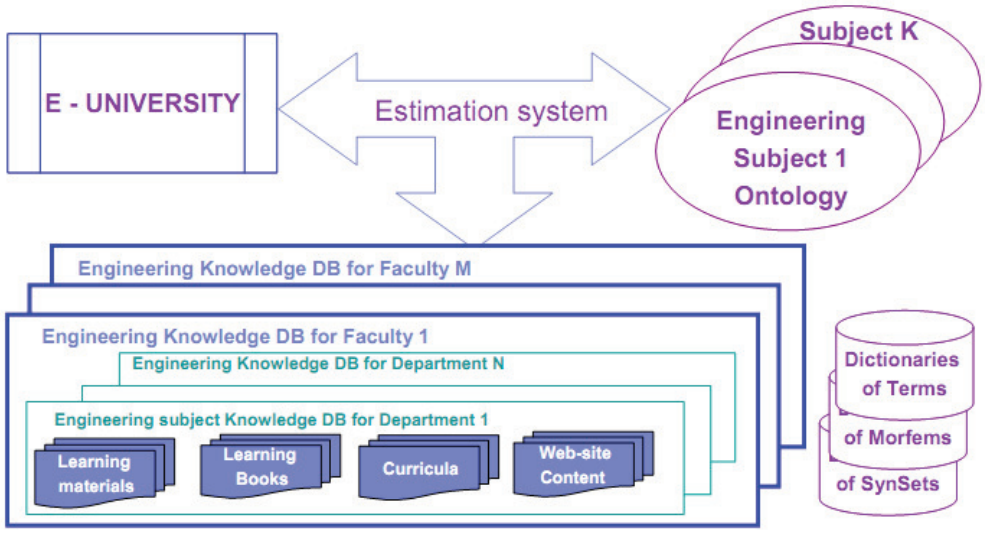



Рис. 9. Схема системы управления качеством образования с использованием онтологий

Семантические технологии помогают выделять полезную информацию из данных, таких как учебные и методические материалы и учебные планы, опираясь на открытые стандарты [17]. Семантические технологии позволяют представить знания с помощью онтологии. Онтология – форма представления знаний о некоторой предметной области, включающая основные понятия этой области и семантические (смысловые) связи между ними. В контексте управления образовательными процессами в качестве предметных областей выступают различные учебные дисциплины.

Одним из способов описания онтологий является описание с помощью иерархических классификаторов [18]: на каждом иерархическом уровне поддерживается отношение эквивалентности на множестве классифицируемых сущностей, обеспечивающее его разбиение на попарно непересекающиеся классы. При этом сущности соседних уровней иерархии обычно находятся в отношении «целое — часть» или «род — вид». Это позволяет предположить, что удобным средством хранения онтологий учебных дисциплин является иерархическая база данных, такая как IBM IMS. Действительно, использование IMS для этих целей имеет целый ряд преимуществ [19]:

- Простота модели. Принципы IMS просты для понимания – иерархическая модель напоминает структуру компании или генеалогическое дерево;
- Использование связей «предок-потомок». IMS легко позволяет представлять связи вида «целое — часть» или «род — вид»;
- Высокая производительность. Связи «предок-потомок» реализуются с помощью ссылок между записями, что позволяет осуществлять быстрые транзакции. Простота структуры БД позволяет IMS размещать записи предка и потомка близко друг от друга, что уменьшает число операций чтения/записи.

Помимо предоставления средства хранения онтологий IMS используется в МГТУ для множества других задач, таких как:

- Удаленное проведение лабораторных работ;
- Оценка качества подготовки студентов.

Таким образом, использование технологии IBM IMS открывает широкие возможности для усовершенствования образовательных процессов и повышения качества образования.

## 7 Заключение

В статье рассмотрены слабые стороны реляционных баз данных и причины отказа от них в пользу NoSQL-технологий. Несмотря на множество достоинств РСУБД, в некоторых ситуациях эти системы сталкиваются с серьезными проблемами. В частности, сложности возникают при использовании распределенных вычислений, а также при обработке данных с нечетко определенной структурой.

Решение этих проблем может быть достигнуто с использованием нереляционных СУБД. В статье рассмотрены основные особенности таких систем и ряд широко используемых алгоритмов, таких как консистентное хеширование и управление параллельным доступом с помощью многоверсионности. Описаны основы фреймворка MapReduce, часто применяемого при распределенной обработке данных.

Приведена классификация NoSQL-систем и описаны основные классы нереляционных СУБД: хранилища «ключ-значение», документо-ориентированные и колоночные базы данных. Отдельно рассмотрена иерархическая модель данных. Наиболее широко распространенной системой, использующей эту модель, является IBM IMS. Эта СУБД используется в МГТУ им. Н.Э. Баумана для ряда задач, связанных с обеспечением качества образования. Рассмотрены перспективы использования этой технологии для усовершенствования образовательных процессов, в частности, для создания онтологий учебных дисциплин.

## Список литературы

1. Codd E.F. A Relational Model of Data for Large Shared Data Banks // Communications of the ACM, 13 (6), 1970. p. 377–387.
2. Padhy R.P., Patra R.M., Satapathy S.C. RDBMS to NoSQL: Reviewing Some Next-Generation Non-Relational Database's // International Journal of Advanced Engineering Sciences and Technologies, Vol No. 11, Issue No. 1, 2011, p. 15 – 30.

3. Cattell R. Scalable SQL and NoSQL Data Stores // SIGMOD Record, December 2010 (Vol. 39, No. 4) p. 12-27.
4. Orend K. Analysis and Classification of NoSQL Databases and Evaluation of their Ability to Replace an Object-relational Persistence Layer: Master's Thesis, Technische Universitat Munchen, 2010, 100 p.
5. Tiwari S. Professional NoSQL. Wiley, 2011.
6. Berndt D.J., Lasa R., McCart J. SiteWit Corporation: SQL or NoSQL that is the Question. University of South Florida, 2012
7. Brewer E.A. Towards Robust Distributed Systems // Keynote at the ACM Symposium on Principles of Distributed Computing (PODC), 2000.
8. Демидов А.А. Проектирование распределённых систем обработки объектных структур данных // Электронные библиотеки: перспективные методы и технологии, электронные коллекции: Труды 12-ой Всероссийской научной конференции (RCDL'2010). – Казань, 2010. – С. 441-447.
9. Pritchett D. BASE: An Acid Alternative // ACM Queue. — N. Y.: ACM, 2008. — Т. 6. — № 3. — p. 48-55.
10. Stensby A.M. Scalability and Horizontal Partitioning // JavaBin. 2009. URL: http://java.no/web/files/S%F8rlandet/moter/2009/03/scalability_and_horizontal_partitioning_javabin.pdf (дата обращения: 11.03.2013).
11. Lipcon, T. Design Patterns for Distributed Non-Relational Databases. 2009. URL: http://static.last.fm/johan/nosql-20090611/intro_nosql.pdf (дата обращения: 21.03.2013 ) .
12. MongoDB team: MongoDB 1.6 Released // MongoDB.org. 2010. URL: http://blog.mongodb.org/post/908172564/mongodb-1-6-released (дата обращения: 21.03.2013 )
13. White T. Consistent Hashing // Java.net. 2007. URL: http://weblogs.java.net/blog/tomwhite/archive/2007/11/consistent_hash.html (дата обращения: 21.03.2013 )
14. Dean J., Ghemawat S. MapReduce: simplified data processing on large clusters. Berkeley, CA, USA, 2004. URL: http://static.usenix.org/event/osdi04/tech/full_papers/dean/dean.pdf (дата обращения: 21.03.2013).
15. Yen S. NoSQL is a horseless carriage. 2009 URL: http://dl.getdropbox.com/u/2075876/nosql-steve-yen.pdf (дата обращения: 21.03.2013).
16. Козлов И.А., Шайхутдинов А.А., Маслов И.Д. Система оценки качества подготовки студентов на основе компетентностного подхода // Молодежный научно-технический вестник #05, 2012. 10 с.

17. Balani N. The future of the Web is Semantic // IBM.com. 2005. URL: http://www.ibm.com/developerworks/library/wa-semweb/index.html (дата обращения: 21.03.2013) .
18. Абросимова М.А. Информационные технологии управления. Учебное пособие. Уфимская государственная академия экономики и сервиса, 2007. 259 с.
19. Smirnova E.V. IBM IMS & Mainframe for Education Quality System // IBM IMS Technical Symposium, Konigstein, 2012.