бюджетное профессиональное образовательное учреждение

Вологодской области

«Череповецкий лесомеханический техникум им. В.П. Чкалова»

Специальность 09.02.07 Информационные системы и программирование

ОТЧЁТ ПО ПРОИЗВОДСТВЕННОЙ ПРАКТИКЕ

ПП ПО ПП.07 СОАДМИНИСТРИРОВАНИЕ БАЗ ДАННЫХ И СЕРВЕРОВ

|  |  |
| --- | --- |
| Выполнила студентка: 4 курса, группы ИС – 41  Жукова Алёна Александровна  (ФИО)  подпись: \_\_\_\_\_\_\_\_\_\_\_\_\_  Место практики: ООО «Вэбест» | Период прохождения:  с «3» ноября 2021г.  по «16» ноября 2021г. |
| Руководитель практики от предприятия:\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_  должность\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_  подпись\_\_\_\_\_\_\_\_\_\_/\_\_\_\_\_\_\_\_\_\_\_\_\_/  МП | Руководитель практики от техникума:\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_  подпись\_\_\_\_\_\_\_\_\_\_/\_\_\_\_\_\_\_\_\_\_\_\_\_/  оценка « » \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_  дата « »\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_2021г. |

г. Череповец

2021г.

СОДЕРЖАНИЕ

[Введение 4](#_Toc87462629)

[1 Администратор базы данных – основные понятия 6](#_Toc87462630)

[1.1 Понятие, классификация и функции администратора базы данных 6](#_Toc87462631)

[1.2 Обязанности, связи и средства администратора современных систем управления базами данных 11](#_Toc87462632)

[2 NoSQL СУБД 15](#_Toc87462633)

[2.1 Зачем нужны не реляционные базы данных в Big Data: история появления и развития 16](#_Toc87462634)

[2.2 Какие бывают NoSQL – СУБД: основные типы не реляционных баз данных 17](#_Toc87462635)

[2.3 Чем хороши и плохи не реляционные базы данных: главные достоинства и недостатки 19](#_Toc87462636)

[3 Особенности nosql-систем 23](#_Toc87462637)

[3.1 Модели согласованности 24](#_Toc87462638)

[3.2 Теорема CAP 26](#_Toc87462639)

[3.3 Модели данных и классификация 27](#_Toc87462640)

[4 Системы «ключ-значение» 29](#_Toc87462641)

[4.1 Project Voldemort 29](#_Toc87462642)

[4.2 DynamoDB 30](#_Toc87462643)

[4.3 Redis 31](#_Toc87462644)

[4.4 Riak 32](#_Toc87462645)

[4.5 Aerospike 33](#_Toc87462646)

[4.6 Резюме 34](#_Toc87462647)

[5 Документные СУБД 35](#_Toc87462648)

[5.1 MongoDB 35](#_Toc87462649)

[5.2 CouchDB 37](#_Toc87462650)

[5.3 Couchbase Server 39](#_Toc87462651)

[5.4 Резюме 40](#_Toc87462652)

[6 Системы типа google bigtable 41](#_Toc87462653)

[6.1 HBase 42](#_Toc87462654)

[6.2 Cassandra 43](#_Toc87462655)

[6.3 Резюме 45](#_Toc87462656)

[7 Другие системы 46](#_Toc87462657)

[Заключение 47](#_Toc87462658)

[Список литературы 49](#_Toc87462659)

[Приложение 1 50](#_Toc87462660)

# ВВЕДЕНИЕ

Современные базы данных – это сложные многофункциональные программные системы, работающие в открытой распределенной среде. Они уже сегодня доступны для использования в деловой сфере и выступают не просто в качестве технических и научных решений, но как завершенные продукты, предоставляющие разработчикам мощные средства управления данными и богатый инструментарий для создания прикладных программ и систем.

Администрирование базами данных предусматривает выполнение функций, направленных на обеспечение надежного и эффективного функционирования системы баз данных, адекватности содержания базы данных информационным потребностям пользователей, отображения в базе данных актуального состояния предметной области.

Необходимость персонала, обеспечивающего администрирование данными в системе БД в процессе функционирования, является следствием централизованного характера управления данными в таких системах, постоянно требующего поиска компромисса между противоречивыми требованиями к системе в социальной пользовательской среде. Хотя такая необходимость и признавалась на ранних стадиях развития технологии баз данных, четкое понимание и структуризация функций персонала, занятого администрированием, сложилось только вместе с признанием многоуровневой архитектуры СУБД.

Проблема исследования «Администрирование базы данных» заключается в возможности дать исчерпывающие ответы на поставленные вопросы: что представляет собой администрирование базы данных, в чем заключаются его основные функции и задачи, его значение для стабильной и эффективной работы базы данных.

Актуальность исследования «Администрирование базы данных» несомненна. Можно провести аналогию между администратором баз данных и ревизором предприятия. Ревизор защищает ресурсы предприятия, которые называются деньгами, а администратор – ресурсы, которые называются данными. Нельзя рассматривать администратора баз данных только как квалифицированного технического специалиста, так как это не соответствует целям администрирования. Уровень администратора баз данных в иерархии организации достаточно высок: чтобы определять структуру данных и право доступа к ним, администратор должен знать, как работает предприятие и как используются соответствующие данные.

Проблеме администрирования баз данных внимание уделяется сравнительно недавно – с появлением и развитием современных баз данных. Однако в связи с тем, что совершенствование баз данных и систем управления данных – явление постоянное и непрерывное, проблема остается достаточно актуальной, следовательно, требует дополнительных исследований в данной области компьютерных технологий.

Цель исследования заключается в изучении администрирования базы данных.

Задачи исследования формируются исходя из его цели и заключаются в следующем:

1. Рассмотреть понятие, классификацию и функции администратора базы данных;
2. Рассмотреть обязанности, связи и средства администратора современных систем управления базами данных;
3. Изучить основные направления и принципы администрирования базы данных.

Данное исследование проведено с использованием теоретических положений, раскрывающих основные характеристики и элементы исследуемого явления.

Практическая значимость исследования заключается в его возможном использовании при изучении информационных технологий в высших учебных заведениях.

# АДМИНИСТРАТОР БАЗЫ ДАННЫХ – ОСНОВНЫЕ ПОНЯТИЯ

## Понятие, классификация и функции администратора базы данных

Функционирование базы данных (БД) невозможно без участия специалистов, обеспечивающих создание, функционирование и развитие базы данных. Такая группа специалистов называется администратором базы данных (АБД). Эта группа специалистов считается составной частью базы данных.

В зависимости от сложности и объема банка данных, от особенностей используемой системы управления базы данных (СУБД), общую схему которой можно увидеть в таблице (см. Приложение 1) служба администрации базы данных может различаться как по составу и квалификации специалистов, так и по количеству работающих в этой службе.

Администратор базы данных выполняют работы по созданию и обеспечению функционирования БД на протяжении всех этапов жизненного цикла системы. В составе группы администраторов банка данных можно выделить различные подгруппы в зависимости от выполняемых ими функций. Численность группы администрации, выполняемые ими функции, будут в значительной степени зависеть от масштаба банка данных, специфики хранимой в нем информации, типа банка данных, особенностей используемых программных средств и некоторых других факторов.

В составе администрации базы данных должны быть системные аналитики, проектировщики структур данных и внешнего по отношению к банку данных информационного обеспечения, проектировщики технологических процессов обработки данных, системные и прикладные программисты, операторы, специалисты по техническому обслуживанию. Если речь идет о коммерческом банке данных, то важную роль здесь будут играть специалисты по маркетингу.

Администраторы базы данных выполняют большой круг разнообразных функций:

* 1. Анализ предметной области: описание предметной области, выявление ограничений целостности, определение статуса информации, определение потребностей пользователей, определение статуса пользователей, определение соответствия «данные – пользователь», определение объемно-временных характеристик обработки данных.
  2. Проектирование структуры базы данных: определение состава и структуры информационных единиц, составляющих базу данных, задание связей между ними, выбор методов упорядочения данных и методов доступа к информации, описание структуры БД на языке обработки данных (ЯОД).
  3. Задание ограничений целостности при описании структуры базы данных и процедур обработки БД: задание ограничений целостности, присущих предметной области, определение ограничений целостности, вызванных структурой базы данных, разработка процедур обеспечения целостности БД при вводе и корректировке данных, обеспечение ограничений целостности при параллельной работе пользователей в многопользовательском режиме.
  4. Первоначальная загрузка и ведение базы данных: разработка технологии первоначальной загрузки и ведения (изменения, добавления, удаления записей) БД, проектирование форм ввода, создание программных модулей, подготовка исходных данных, ввод и контроль ввода.
  5. Защита данных от несанкционированного доступа:

− обеспечение парольного входа в систему: регистрация пользователей, назначение и изменение паролей;

− обеспечение защиты конкретных данных: определение прав доступа групп пользователей и отдельных пользователей, определение допустимых операций над данными для отдельных пользователей, выбор/создание программно-технологических средств защиты данных;

− шифрование информации с целью защиты данных от несанкционированного использования;

− тестирование средств защиты данных;

− фиксация попыток несанкционированного доступа к информации;

− исследование возникающих случаев нарушения защиты данных и проведение мероприятий по их предотвращению.

* 1. Защита данных от разрушений. Одним из способов защиты от потери данных является резервирование. Используется как при физической порче файла, так и в случае, если в БД внесены нежелательные необратимые изменения.
  2. Обеспечение восстановления БД: разработка программно-технологических средств восстановления БД, организация ведения системных журналов.
  3. Анализ обращений пользователей к БД: сбор статистики обращений пользователей к БД, ее хранение и анализ (кто из пользователей, к какой информации, как часто обращался, какие выполнял операции, время выполнения запросов, анализ причин безуспешных (в т.ч. и аварийных) обращений к БД).
  4. Анализ эффективности функционирования базы данных и развитие системы: анализ показателей функционирования системы (время обработки, объем памяти, стоимостные показатели), реорганизация и реструктуризация баз данных, изменение состава баз данных, развитие программных и технических средств.
  5. Работа с пользователями: сбор информации об изменениях в предметной области, об оценке пользователями работы базы данных, определение регламента работы пользователей с базой данных, обучение и консультирование пользователей.
  6. Подготовка и поддержание системных программных средств: сбор и анализ информации о СУБД и других прикладных программ, приобретение программных средств, их установка, проверка работоспособности, поддержание системных библиотек, развитие программных средств.
  7. Организационно-методическая работа: выбор или создание методики проектирования БД, определение целей и направлений развития системы, планирование этапов развития базы данных, разработка и выпуск организационно-методических материалов.

Классификация АБД

Существует несколько видов администраторов БД, а их обязанности вполне могут отличаться от компании к компании. Вот характеристики некоторых типов АБД и занимаемых ими положений: Оперативные (operational) АБД:

− манипулируют дисковым пространством;

− наблюдают за текущей производительностью системы;

− реагируют на возникающие неисправности БД;

− обновляют системное ПО и ПО базы данных;

− контролируют структурные изменения БД;

− запускают процедуры резервного копирования данных;

− выполняют восстановление данных;

− создают и управляют тестовыми конфигурациями БД.

Тактические (tactical) АБД:

− реализуют схемы размещения информации утверждают процедуры резервного копирования и восстановления данных;

− разрабатывают и внедряют структурные элементы БД: таблицы, столбцы, размеры объектов, индексацию и т.п.;

− сценарии(scripts) изменения схемы БД;

− конфигурационные параметры БД утверждают план действий в случае аварийной ситуации.

Стратегические (strategic) АБД:

− выбирают поставщика БД;

− устанавливают корпоративные стандарты данных;

− определяют корпоративную стратегию резервирования и восстановления данных;

− устанавливают корпоративный подход к ликвидации последствий аварии и обеспечению доступности данных.

Старшие (senior) АБД:

− досконально знают свой персонал;

− пользуются высоким спросом;

− могут написать скрипт, который освободит их из запертого сундука, брошенного в океан, и чрезвычайно гордятся своими произведениями;

− тратят уйму времени на подготовку младших АБД;

− очень ценятся руководством и получают бешеные деньги.

Младшие (junior) АБД:

− мечтают стать старшим АБД;

− не слишком сильны в написании скриптов, имеют большую склонность к использованию средств управления БД;

− тоже неплохо получают.

Прикладные (application) АБД:

− в курсе информационных нужд компании;

− помогают в разработке прикладных задач;

− отвечают за разработку схемы и ее изменения;

− вместе с системным АБД обеспечивают должный уровень резервирования/восстановления данных;

− занимаются построением тестовых БД.

Системные (system) АБД:

− отвечают за все необходимое для резервирования и восстановления данных;

− контролируют производительность системы в целом;

− осуществляют поиск и устранение неисправностей;

− в курсе нынешних и будущих потребностей БД в плане емкости в курсе текущего состояния и нужд БД.

Наемные (contract) АБД:

− приглашаются под конкретную задачу или в качестве консультантов передают персоналу необходимые знания;

− фиксируют свои действия;

− должны прекрасно разбираться в соответствующей области;

− хороши в качестве временного персонала, для оценки проекта или системы.

Администраторы – руководители: проводят еженедельные совещания определяют перечень первоочередных задач устанавливают и оглашают официальный курс и стратегию утверждают и корректируют должностные инструкции и список обязанностей следят за наличием соответствующей документации.

## Обязанности, связи и средства администратора современных систем управления базами данных

Поскольку система баз данных может быть весьма большой и может иметь много пользователей, должно существовать лицо или группа лиц, управляющих этой системой. Такое лицо называется администратором базы данных (АБД).

В любой базе данных должен быть хотя бы один человек, выполняющий административные обязанности; если база данных большая, эти обязанности могут быть распределены между несколькими администраторами.

В обязанности администратора могут входить:

− инсталляция и обновление версий сервера и прикладных инструментов;

− распределение дисковой памяти и планирование будущих требований системы к памяти создание первичных структур памяти в базе данных (табличных пространств) по мере проектирования приложений разработчиками приложений;

− создание первичных объектов (таблиц, представлений, индексов) по мере проектирования приложений разработчиками;

− модификация структуры базы данных в соответствии с потребностями приложений;

− зачисление пользователей и поддержание защиты системы;

соблюдение лицензионного соглашения управление и отслеживание доступа пользователей к базе данных;

− отслеживание и оптимизация производительности базы данных;

планирование резервного копирования и восстановления поддержание архивных данных на устройствах;

− хранения информации;

− осуществление резервного копирования и восстановления обращение в корпорацию за техническим сопровождением.

В некоторых случаях база данных должна также иметь одного или нескольких сотрудников службы безопасности. Сотрудник службы безопасности главным образом отвечает за регистрацию новых пользователей, управление и отслеживание доступа пользователей к базе данных, и защиту базы данных.

В обязанности разработчика приложений входит:

− проектирование и разработка приложений базы данных;

− проектирование структуры базы данных в соответствии с требованиями приложений;

− оценка требований памяти для приложения;

− формулирование модификаций структуры базы данных для приложения;

− передача вышеупомянутой информации администратору базы данных;

− настройка приложения в процессе его разработки установка мер по защите приложения в процессе его разработки.

В процессе своей деятельности администратор базы данных взаимодействует с другими категориями пользователей банка данных, а также и с «внешними» специалистами, не являющимися пользователями базы данных.

Прежде всего, если банк данных создается для информационного обслуживания какого-либо предприятия или организации, то необходимы контакты с администрацией этой организации. Как указывалось выше, внедрение БД приводит к большим изменениям не только системы обработки данных, но и всей системы управления организацией. Естественно, что такие большие проекты не могут быть выполнены без активного участия и поддержки руководителей организации. Руководство организации должно быть ознакомлено с возможностями, предоставляемыми базой данных, проинформировано об их преимуществах и недостатках, а также проблемах, вызываемых созданием и функционированием базы данных.

Так как база данных является динамическим информационным отображением предметной области, то желательно, чтобы администратор базы данных в свою очередь был своевременно информирован о перспективах развития объекта, для которого создается информационная система.

Руководством организации и администратором базы данных должны быть согласованы цели, основные направления и сроки создания БД и его развития, очередность подключения пользователей.

Очень тесная связь у АБД на всех этапах жизненного цикла базы данных наблюдается с конечными пользователями. Это взаимодействие начинается на начальных стадиях проектирования системы, когда изучаются потребности пользователей, уточняются особенности предметной области, и постоянно поддерживается как на протяжении процесса проектирования, так и функционирования системы.

Следует отметить, что в последнее время наблюдается активное перераспределение функций между конечными пользователями и администраторами банка данных. Это, прежде всего, связано с развитием языковых и программных средств, ориентированных на конечных пользователей. Сюда относятся простые и одновременно мощные языки запросов, а также средства автоматизации проектирования.

Администраторы базы данных взаимодействуют и с внешними по отношению к нему группами специалистов и, прежде всего, поставщиками СУБД и ППП, администраторами других баз данных.

Базы данных часто создаются специализированными проектными коллективами на основе договора на разработку информационной системы в целом или базой данных как самостоятельного объекта проектирования. В этом случае служба администрации базы данных должна создаваться как в организации-разработчике, так и в организации-заказчике.

На эффективность работы базы данных оказывают влияние множество внешних и внутренних факторов. Возрастание сложности и масштабов базы данных, высокая «цена» неправильных или запоздалых решений по администрированию БД, высокие требования к квалификации специалистов делают актуальной задачу использования развитых средствах автоматизированного (или даже автоматического) администрирования базы данных.

# NoSQL СУБД

NoSQL – это подход к реализации масштабируемого хранилища (базы) информации с гибкой моделью данных, отличающийся от классических реляционных СУБД. В не реляционных базах проблемы масштабируемости (scalability) и доступности (availability), важные для Big Data, решаются за счёт атомарности (atomicity) и согласованности данных (consistency).

Термин «NoSQL» был впервые применен в 1998 году Карло Строцци (Carlo Strozzi) в качестве названия для его небольшой реляционной СУБД, которая не использовала язык SQL для манипулирования данными. С 2009 года термин «NoSQL» стал использоваться уже для обозначения растущего числа распределенных систем управления данными, которые отказывались от поддержки ACID-транзакций (Atomicity, Consistency, Isolation, Durability – Атомарность, Согласованность, Изолированность, Постоянство хранения) – одного из ключевых принципов работы с реляционными базами данных. По мнению Строцци, современные системы категории NoSQL точнее было бы называть не реляционными («NoREL»). В настоящее время термин «NoSQL» обычно расшифровывается как «Not Only SQL» , то есть «Не только SQL».

Главной предпосылкой к появлению систем, относимых к категории NoSQL, послужила растущая потребность в горизонтальной масштабируемости приложений, то есть в возможности наращивать производительность путём добавления новых вычислительных узлов к уже работающим. Таким образом, большинство NoSQL-систем изначально проектировались и создавались для работы в распределенной среде - кластере или облаке, где применение традиционных SQL-ориентированных систем связано с определёнными трудностями. Основной причиной отказа от поддержки транзакционной семантики послужила сложность эффективной реализации транзакций в распределённой среде: в общем случае приходится использовать двухфазный протокол фиксации транзакций, который требует пересылки большого количества сообщений по сети. Хотя NoSQL-системы обычно не поддерживают ACID-транзакции в полном объеме, в ряде случаев поддерживаются, например, атомарные операции для чтения и модификации, оптимистические блокировки и другие инструменты, помогающие упростить разработку приложений в условиях параллельного доступа к данным.

## Зачем нужны не реляционные базы данных в Big Data: история появления и развития

NoSQL – базы оптимизированы для приложений, которые должны быстро, с низкой временной задержкой (low latency) обрабатывать большой объем данных с разной структурой. Таким образом, не реляционные хранилища непосредственно ориентированы на Big Data. Однако идея баз данных такого типа зародилась гораздо раньше термина «большие данные», еще в 80-е годы прошлого века, во времена первых компьютеров (мейнфреймах) и использовалась для иерархических служб каталогов. Современное понимание NoSQL – СУБД возникло в начале 2000-х годов, в рамках создания параллельных распределённых систем для высоко масштабируемых интернет-приложений, таких как онлайн-поисковики.

Вообще термин NoSQL обозначает «не только SQL» (Not Only SQL), характеризуя ответвление от традиционного подхода к проектированию баз данных. Изначально так называлась опенсорсная база данных, созданная Карло Строззи, которая хранила все данные как ASCII-файлы, а вместо SQL-запросов доступа к данным использовала шелловские скрипты.

В начале 2000-х годов Google построил свою поисковую систему и приложения (GMail, Maps, Earth и прочие сервисы), решив проблемы масштабируемости и параллельной обработки больших объёмов данных.

Так была создана распределённые файловая и координирующая системы, а также колоночное хранилище (column family store), основанное на вычислительной модели MapReduce.

После того, как корпорация Google опубликовала описание этих технологий, они стали очень популярны у разработчиков открытого программного обеспечения. В результате этого был создан Apache Hadoop и запущены основные связанные с ним проекты.

Например, в 2007 году другой ИТ-гигант, Amazon.com, опубликовав статьи о своей высоко доступной базе данных Amazon DynamoDB. Далее в эту гонку NoSQL – технологий для управления большими данными включилось множество корпораций: IBM, Facebook, Netflix, eBay, Hulu, Yahoo! и другие ИТ – компаний со своими пропри тарными и открытыми решениями (рисунок 1).



Рисунок - Многообразие NoSQL – решений

## Какие бывают NoSQL – СУБД: основные типы не реляционных баз данных

Все NoSQL решения принято делить на 4 типа:

1. Ключ-значение (Key-value) – наиболее простой вариант хранилища данных, использующий ключ для доступа к значению в рамках большой хэш-таблицы. Такие СУБД применяются для хранения изображений, создания специализированных файловых систем, в качестве кэшей для объектов, а также в масштабируемых Big Data системах, включая игровые и рекламные приложения, а также проекты интернета вещей (Internet of Things, IoT), в т.ч. индустриального (Industrial IoT, IIoT). Наиболее известными представителями нереляционных СУБД типа key-value считаются Oracle NoSQL Database, Berkeley DB, MemcacheDB, Redis, Riak, Amazon DynamoDB, которые поддерживают высокую разделяемость, обеспечивая беспрецедентное горизонтальное масштабирование, недостижимое при использовании других типов БД.
2. Документно-ориентированное хранилище, в котором данные, представленные парами ключ-значение, сжимаются в виде полу структурированного документа из тегированных элементов, подобно JSON, XML, BSON и другим подобным форматам. Такая модель хорошо подходит для каталогов, пользовательские профилей и систем управления контентом, где каждый документ уникален и изменяется со временем.  Поэтому чаще всего документные NoSQL-СУБД используются в CMS-системах, издательском деле и документальном поиске. Самые яркие примеры документно-ориентированных нереляционных баз данных – это CouchDB, Couchbase, MongoDB, eXist, Berkeley DB XML.
3. Колоночное хранилище, которое хранит информацию в виде разреженной матрицы, строки и столбцы которой используются как ключи. В мире Big Data к колоночным хранилищам относятся базы типа «семейство столбцов» (Column Family). В таких системах сами значения хранятся в столбцах (колонках), представленных в отдельных файлах. Благодаря такой модели данных можно хранить большое количество атрибутов в сжатом виде, что ускоряет выполнение запросов к базе, особенно операции поиска и агрегации данных. Наличие временных меток (timestamp) позволяет использовать такие СУБД для организации счётчиков, регистрации и обработки событий, связанных со временем: системы биржевой аналитики, IoT/IIoT-приложения, систему управления содержимым и т.д. Самой известной колоночной базой данных является Google Big Table, а также основанные на ней Apache HBase и Cassandra. Также к этому типу относятся менее популярные ScyllaDB, Apache Accumulo и Hypertable.
4. Графовое хранилищепредставляют собой сетевую базу, которая использует узлы и рёбра для отображения и хранения данных. Поскольку рёбра графа являются хранимыми, его обход не требует дополнительных вычислений (как соединение в SQL). При этом для нахождения начальной вершины обхода необходимы индексы. Обычно графовые СУБД поддерживают ACID-требования и специализированные языки запросов (Gremlin, Cypher, SPARQL, GraphQL и т.д.). Такие СУБД используются в задачах, ориентированных на связи: социальные сети, выявление мошенничества, маршруты общественного транспорта, дорожные карты, сетевые топологии. Примеры графовых баз: InfoGrid, Neo4j, Amazon Neptune, OrientDB, AllegroGraph, Blazegraph, InfiniteGraph, FlockDB, Titan, ArangoDB.

Виды NoSQL – СУБД представлены на рисунке 2.

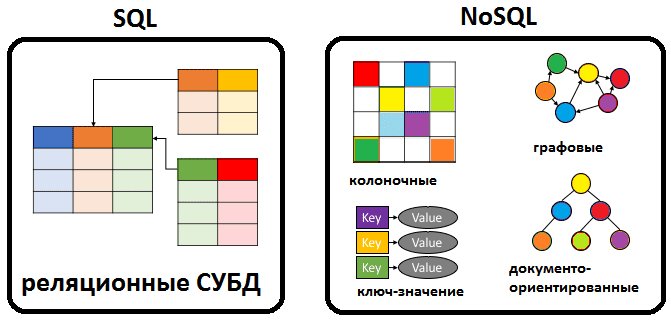


Рисунок – Виды NoSQL – СУБД

## Чем хороши и плохи не реляционные базы данных: главные достоинства и недостатки

По сравнению с классическими SQL-базами, не реляционные СУБД обладают следующими преимуществами:

* линейная масштабируемость – добавление новых узлов в кластер увеличивает общую производительность системы;
* гибкость, позволяющая оперировать полу-структурированные данные, реализуя, в. т.ч. полнотекстовый поиск по базе;
* возможность работать с разными представлениями информации, в т.ч. без задания схемы данных;
* высокая доступность за счет репликации данных и других механизмов отказоустойчивости, в частности, шаринга – автоматического разделения данных по разным узлам сети, когда каждый сервер кластера отвечает только за определенный набор информации, обрабатывая запросы на его чтение и запись. Это увеличивает скорость обработки данных и пропускную способность приложения.
* производительностьза счет оптимизации для конкретных видов моделей данных (документной, графовой, колоночной или «ключ‑значение») и шаблонов доступа;
* широкие функциональные возможности – собственные SQL-подобные языки запросов, RESTful-интерфейсы, API и сложные типы данных, например, map, list и struct, позволяющие обрабатывать сразу множество значений.

Обратной стороной вышеуказанных достоинств являются следующие недостатки:

* ограниченная емкость встроенного языка запросов.

Например, HBase предоставляет всего 4 функции работы с данными (Put, Get, Scan, Delete), в Cassandra отсутствуют операции Insert и Join, несмотря на наличие SQL-подобного языка запросов.

Для решения этой проблемы используются сторонние средства трансляции классических SQL-выражений в исполнительный код для конкретной, не реляционной базы. Например, Apache Phoenix для HBase или универсальный Drill.

* сложности в поддержке всех ACID-требований к транзакциям(атомарность, консистентность, изоляция, долговечность) из-за того, что NoSQL-СУБД вместо CAP-модели (согласованность, доступность, устойчивость к разделению) скорее соответствуют модели BASE (базовая доступность, гибкое состояние и итоговая согласованность). Впрочем, некоторые не реляционные СУБД пытаются обойти это ограничение с помощью настраиваемых уровней согласованности, о чем мы рассказывали на примере Cassandra.

Аналогичным образом Riak позволяет настраивать требуемые характеристики доступности-согласованности даже для отдельных запросов за счет задания количества узлов, необходимых для подтверждения успешного завершения транзакции. Подробнее о CAP-и BASE-моделях мы расскажем в отдельной статье.

* сильная привязка приложения к конкретной СУБД из-за специфики внутреннего языка запросов и гибкой модели данных, ориентированной на конкретный случай;
* недостаток специалистов по NoSQL-базам по сравнению с реляционными аналогами.

Подводя итог описанию основных аспектов нереляционных СУБД, стоит отметить некоторую некорректность запроса «NoSQL vs SQL» в связи с разными архитектурными подходами и прикладными задачами, на которые ориентированы эти ИТ-средства. Традиционные SQL-базы отлично справляются с обработкой строго типизированной информации не слишком большого объема. Например, локальная ERP-система или облачная CRM. Однако, в случае обработки большого объема полу структурированных и неструктурированных данных, т.е. Big Data, в распределенной системе следует выбирать из множества NoSQL-хранилищ, учитывая специфику самой задачи. В частности, для самостоятельных решений интернета вещей (Internet of Things), в т.ч. промышленного, отлично подходит Cassandra, о чем мы рассказывали здесь. А в случае многоуровневой ИТ-инфраструктуры на базе Apache Hadoop стоит обратить внимание на HBase, которая позволяет оперативно, практически в режиме реального времени, работать с данными, хранящимися в HDFS.

Не реляционные СУБД находят больше областей приложений, чем традиционные SQL-решения – рисунок 3.

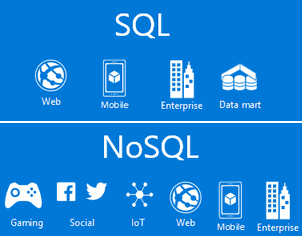


Рисунок – Области приложений

# ОСОБЕННОСТИ NOSQL-СИСТЕМ

Как уже было отмечено, большинство NoSQL-систем являются распределенными. Распределённая архитектура позволяет достичь не только горизонтальной масштабируемости (в идеале – линейного роста производительности при добавлении новых узлов), но и увеличить надежность системы с помощью поддержания нескольких копий данных. В распределённых системах управления данными используются два основных приёма (практически всегда применяемых совместно):

Разделение данных (шардинг, sharding) – подход, при котором каждый узел системы содержит свою часть данных и выполняет операции над ними. Этот метод является основным средством обеспечения горизонтальной масштабируемости, однако теперь операции над несколькими объектами могут вовлечь в работу несколько узлов, что требует активной передачи данных по сети. Кроме того, при увеличении числа узлов, хранящих данные, увеличивается вероятность сбоев, что требует наличия избыточности – применения репликации (см. далее). Шардинг также порождает такие задачи, как распределение данных по узлам, балансировка нагрузки, оптимизация сетевых взаимодействий и т.д.

Репликация (replication) – подход, при котором одни и те же данные хранятся на нескольких узлах в сети. Этот метод помогает повысить надежность системы и справляться как со сбоями отдельных узлов, так и с потерей целого кластера. Кроме того, репликация позволяет масштабировать операции чтения (несколько реже - записи). Серьезной задачей является поддержка согласованного состояния копий данных (реплик): при синхронном обновлении реплик увеличивается время ответа системы, а при асинхронном - возникает промежуток времени, когда реплики находятся в несогласованном состоянии.

Существуют две основные схемы репликации (поддерживающие как синхронные, так и асинхронные варианты):

Ведущий-ведомый (master-slave) – при такой схеме репликации операции модификации данных обрабатывает только ведущий узел (master), а сделанные изменения синхронно или асинхронно передаются на ведомого (slave). Чтения могут осуществляться как с ведущего узла (гарантированно содержит последнюю версию данных), так и с ведомого (данные могут быть несколько устаревшими при асинхронной репликации и «отставать» от ведущего).

Ведущий-ведущий (master-master, multi-master) – при этой схеме все узлы могут обрабатывать операции записи и передавать обновления остальным. В этом случае реализовать синхронную репликацию достаточно сложно, к тому же резко возрастают задержки, связанные с сетевыми взаимодействиями. При асинхронном обновлении возникает другая проблема – могут появиться конфликтующие версии данных, которые требуют наличия механизма для определения и разрешения конфликтов (автоматически или на уровне приложения).

## Модели согласованности

Применение репликации в распределённой системе порождает задачу поддержания идентичного состояния копий данных на разных узлах (и, соответственно, видимых разными клиентами). Для обозначения гарантий согласованности данных, которые предоставляются системой, используют термин «модель согласованности» (consistency model). Следует отметить, что слово «согласованность» в этом контексте отличается от свойства согласованности из определения ACID-транзакций. Модель согласованности определяет физическую согласованность состояния данных на разных узлах системы, в то время как в случае ACID имеется в виду логическая согласованность (в рамках ограничений целостности, определённых для базы данных). Ниже приводятся примеры моделей, часто применяемых в NoSQL-системах:

Согласованность, в «конечном счёте» (eventual consistency) гарантирует, что при отсутствии новых обновлений в течение некоторого времени все копии данных (реплики) станут согласованными.

Монотонные чтения (monotonic reads) являются усилением согласованности «в конечном счёте», гарантируя, что если какое-либо значение было прочитано клиентом, то последующие чтения никогда не вернут предыдущие значения.

Чтение своих записей (read your writes) также усиливает согласованность «в конечном счёте», давая гарантии того, что клиент всегда увидит данные, которые он до этого записал. Эта модель может также комбинироваться с монотонными чтениями.

Мгновенная согласованность (immediate consistency) означает, что как только операция модификации данных успешно завершена, все клиенты мгновенно увидят это изменение. Такую модель согласованности поддерживают, например, системы с синхронной репликацией.

Наиболее часто NoSQL-системы поддерживают согласованность «в конечном счёте» или её вариации. При этом возникает период времени, в течение которого данные находятся в несогласованном состоянии, называемый «окном несогласованности» (inconsistency window). Его величина зависит (при отсутствии сбоев) от скорости распространения обновлений, загруженности системы и количества узлов, содержащих реплики данных. Приложения, использующие системы управления данными, допускающие несогласованность, должны уметь справляться с появлением несколько «устаревших» данных. Кроме того, при одновременных обновлениях реплик данных возникают конфликтующие версии. Для обнаружения конфликтов применяются такие подходы, как временные метки и векторные часы. Для разрешения конфликтов существуют различные стратегии, но обычно наиболее точное решение о том, какую версию выбрать, может сделать только само приложение (о различных стратегиях разрешения конфликтов).

В некоторых системах уровень поддержки согласованности можно варьировать, используя различные конфигурации, настройки и/или операции. Рассмотрим, как этого можно добиться на практике с помощью кворума. Пусть данные реплицируются по N-узлам; обозначим через R – число узлов, к которым обращается клиент при чтении данных, а через W – число узлов, от которых ожидается подтверждение успешной записи. Изменяя эти параметры, можно добиться различного поведения распределённой системы:

При R + W > N множества реплик для записи и чтения пересекаются, а значит, чтение всегда возвратит результат успешно завершившейся операции записи (мгновенная согласованность).

При R + W ≤ N невозможно гарантировать возврат последней версии данных, гарантируется лишь согласованность «в конечном счёте».

При R = N, W = 1 поддерживается мгновенная согласованность, а операции записи работают быстро за счёт более медленных и дорогих чтений (с N реплик). При W = N, R = 1 получается обратная ситуация.

Следует отметить, что чем ближе значения R и W к общему числу реплик N, тем больше вероятность, что операция может завершиться ошибкой из-за выхода каких-либо узлов из строя.

Вариации согласованности «в конечном счёте» (например, наличие монотонных чтений и чтений своих записей) во многом зависят от реализации взаимодействия клиентов с системой (привязывается ли клиент к конкретному серверу, используется ли узел-маршрутизатор, поддерживаются ли версии объектов и т.д.). Более подробно о нюансах моделей согласованности в NoSQL-системах.

## Теорема CAP

Для обоснования компромиссов, выбираемых NoSQL-системами, часто приводится эмпирическое утверждение, известное как теорема CAP, или теорема Брюэра (Eric Brewer). Это утверждение гласит, что распределенная система не может гарантировать одновременного выполнения следующих трёх свойств:

Согласованность (Consistency) - все узлы в каждый момент времени имеют согласованные данные (все пользователи в любой момент времени видят одинаковые данные).

Доступность (Availability) - при выходе из строя каких-либо узлов оставшиеся узлы должны продолжать функционировать.

Устойчивость к разделению (Partition Tolerance) - если из-за сбоя сети система распадается на группы узлов, не связанные между собой, то каждая группа должна продолжать функционировать.

Несмотря на то, что данное утверждение само по себе недостаточно формализовано, оно было уточнено и доказано для некоторых частных случаев. Однако желание обеспечить устойчивость к разделению сети и высокую доступность системы – не единственные причины для ослабления согласованности данных. Как отмечает Дэниэл Абади (Daniel Abadi), формулировка теоремы CAP не учитывает такого важного свойства системы, как величина задержки при ответе на запрос пользователя. Усиление модели согласованности ведёт в общем случае к более высоким задержкам (достаточно вспомнить, например, случай синхронной репликации, когда обновления должны распространиться по большинству узлов или даже по всем узлам для того, чтобы операция изменения данных считалась завершенной). Особенно важно минимизировать передачу данных по сети между узлами в случае большой географической распределённости кластеров.

## Модели данных и классификация

Еще одним важным отличием систем категории NoSQL от реляционных баз данных являются их не реляционные модели данных и способы осуществления запросов. В целом модели данных, лежащие в основе NoSQL, значительно проще, чем классическая реляционная модель, что в ряде случаев облегчает работу с ними. Обычно (хотя и с некоторыми вариациями) NoSQL-системы делят, исходя из модели данных, на следующие основные классы:

Системы «ключ-значение» (Key-Value Stores)

Документные СУБД (Document Stores)

Системы типа Google BigTable (Extensible Record Stores/Wide Column Stores/Column Families)

Иногда под термином «NoSQL» понимают также вообще все системы, не являющиеся реляционными (SQL-ориентированными), однако чаще всего имеются в виду именно представители приведённых трёх классов систем.

Это деление является достаточно общим, так как внутри каждой группы системы значительно различаются и в плане поддержки согласованности данных, и в нюансах работы с ними (например, атомарность операций, использование блокировок или мультиверсионного доступа (MVCC, Multi-Version Concurrency Control), однако оно отражает основные аспекты и область применения соответствующих NoSQL-систем.

Кроме того, многие системы имеют черты более чем одного класса, и иногда их трудно классифицировать по такому принципу. Далее будет дана общая характеристика каждого из классов и рассмотрены примеры некоторых конкретных систем, которые могут быть к ним отнесены.

# СИСТЕМЫ «КЛЮЧ-ЗНАЧЕНИЕ»

NoSQL-системы типа «ключ-значение» хранят данные (неструктурированные или структурированные) и позволяют иметь доступ к ним при помощи единственного уникального ключа. Работа с данными обычно осуществляется с помощью простых операций вставки, удаления и поиска по ключу. Вторичные ключи и индексы в таких системах не поддерживаются. При этом может поддерживаться некоторая структура данных, позволяющая менять отдельные поля объекта, но не позволяющая строить по ним запросы (в этом заключается основное отличие систем типа «ключ-значение» от документных СУБД). В этом отношении системы «ключ-значение» похожи на популярную распределённую систему кэширования в оперативной памяти Memcached, но предоставляют постоянное хранение данных и ряд дополнительных возможностей. Существуют NoSQL-системы, обладающие обратной совместимостью с Memcached, что позволяет использовать тот же интерфейс и те же клиентские библиотеки, облегчая переход с Memcached (например, MemcacheDB, Couchbase Server и др.). Далее будут подробнее рассмотрены некоторые системы класса «ключ-значение».

## Project Voldemort

Project Voldemort – система управления данными типа «ключ-значение» с открытым исходным кодом, реализованная на Java и активно используемая в социальной сети LinkedIn. Помимо скалярных значений, Project Voldemort допускает также списки значений и записи с именованными полями, ассоциируемые с одним ключом. Самим значением и ключом может являться любая сущность, для которой определена сериализация (правила преобразования объекта в последовательность байтов и обратно). Вся работа с данными осуществляется с использованием трёх операций: put, get и delete. Project Voldemort использует механизм MVCC при модификации данных.

Project Voldemort поддерживает шардинг и репликацию, а также несколько способов физической организации системы. Для распределения ключей по логическому кольцу узлов используется метод консистентного хэширования (consistent hashing). Шардинг осуществляется прозрачно для приложения, узлы могут быть добавлены или удалены в процессе работы, восстановление при сбоях также происходит автоматически. Project Voldemort использует асинхронную репликацию и поддерживает согласованность «в конечном счёте», конфликты разрешаются при чтении (read-repair). Также используется механизм «hinted handoff», который позволяет сохранить данные даже при выходе хранящих их узлов из строя, используя соседние узлы. Project Voldemort может хранить данные в оперативной памяти и обеспечивать постоянство хранения с помощью одного из поддерживаемых механизмов, например, Berkeley DB.

## DynamoDB

DynamoDB – облачный сервис, представленный компанией Amazon в начале 2012 года. Эта система является последователем таких технологий Amazon, как Dynamo и SimpleDB. DynamoDB предоставляет пользователям быструю и масштабируемую систему управления данными, репликация и шардинг в которой осуществляются автоматически. Высокая производительность системы достигается за счёт использования твердотельных накопителей (SSD) и ряда других оптимизаций. Использование облачного сервиса снимает с пользователей необходимость в установке и администрировании серверов, позволяя оплачивать лишь потребляемые ресурсы, такие как трафик и объем хранимой информации.

Модель данных DynamoDB является достаточно гибкой и богатой для систем типа «ключ - значение». Данные хранятся в так называемых таблицах, обладающих первичным ключом (простым или составным) и набором атрибутов (заранее зафиксированной схемы нет, поддерживаются скалярные типы данных и множества). Для работы с данными используются операции поиска, вставки и удаления по первичному ключу, условные операции (например, обновить, если выполнено условие), атомарные модификации (например, увеличение значения атрибута на единицу) и поиск по неключевым атрибутам путём полного сканирования таблицы. Последняя операция делает эту систему еще ближе к документным СУБД, однако эффективного способа запрашивать данные по неключевым атрибутам DynamoDB не имеет. Желаемый уровень согласованности может быть указан при чтении данных («eventually consistent reads» или «strongly consistent reads»). Полностью согласованные чтения гарантированно возвращают последнюю версию данных, однако это сказывается на производительности. Клиентские библиотеки для работы с DynamoDB доступны для большого числа языков программирования, включая Java, .NET, PHP, Perl, Python, Ruby и др.

## Redis

Redis – система управления данными типа «ключ-значение» с открытым исходным кодом, написанная на C и также поддерживающая достаточно богатую для таких систем модель данных. Значения могут содержать не только строки, но и множества, списки и другие структуры данных. Помимо обычных операций получения, сохранения и удаления данных по ключу, Redis поддерживает атомарные операции, такие как увеличение числа на единицу, добавление элемента в список и т. д. Кроме того, поддерживаются транзакции, содержащие группы операций и обладающие свойствами изолированности и атомарности, а также оптимистические блокировки.

Redis работает в оперативной памяти, за счёт чего достигается высокая производительность. Для обеспечения долговременного хранения могут применяться снимки данных в определённые моменты времени или постоянная запись операций модификации данных на диск; также возможна работа вообще без использования дисков.

Горизонтальная масштабируемость может быть достигнута с помощью разделения данных (логика реализуется на стороне клиента). Также поддерживается асинхронная репликация со схемой «ведущий-ведомый».

Клиентские библиотеки для работы с Redis доступны для большинства языков программирования, а сама эта система используется в таких крупных проектах, как Twitter, Instagram, Digg, Github, StackOverflow, Flickr и других.

## Riak

Riak – мощная система управления данными типа «ключ-значение» с открытым исходным кодом, написанная на Erlang. На архитектуру Riak оказала значительное влияние система Amazon Dynamo.

Организация кластера Riak похожа на Project Voldemort: также используется консистентное хэширование и hinted handoff. Такая архитектура обеспечивает надежность, децентрализацию и легкое добавление новых физических узлов. На каждом физическом узле (node) может работать несколько виртуальных узлов (vnode), что помогает при балансировке нагрузки; репликация выполняется асинхронно, количество реплик может быть гибко настроено (N). Riak позволяет указывать число реплик для чтения (R) и записи (W) при соответствующих операциях и, таким образом, варьировать уровень согласованности, однако атомарных операций не предусмотрено. Riak использует вариант MVCC для реализации параллельного доступа. Конфликты обновлений могут разрешаться либо по принципу «последний выигрывает», либо на уровне приложения (в этом случае приложению возвращаются конфликтующие версии объектов). Поддерживаются триггеры (называемые «commit hooks»), позволяющие запускать функции на JavaScript или Erlang перед или после модификации объекта.

Ключи в Riak организуются в корзины (bucket»), что позволяет задавать такие параметры, как число реплик, список триггеров и метод разрешения конфликтов, на уровне корзины. Таким образом, объект однозначно идентифицируется парой (корзина, ключ). Объекты в Riak представляются в виде JSON и могут иметь несколько полей данных. Кроме того, объект содержит метаданные, в том числе поддерживаются ссылки на другие объекты. Riak поддерживает вторичные индексы (возможность приписать объекту пару атрибут-значение для последующих запросов по ним) и MapReduce для более сложных запросов. Функциональность Riak делает эту систему близкой к документным СУБД, однако в Riak не поддерживаются запросы к полям объектов (единственный способ – MapReduce), поэтому обычно эту систему относят к классу систем «ключ-значение».

Подсистема хранения данных в Riak является подключаемым модулем, что позволяет использовать различные реализации, подходящие для разных задач. Взаимодействовать с Riak пользователь может с помощью REST-интерфейса или с помощью программного интерфейса, доступного для большинства языков программирования. Riak используется в достаточно большом количестве проектов.

## Aerospike

Aerospike (ранее эта система называлась Citrusleaf) интересна тем, что, являясь NoSQL- системой типа «ключ-значение», поддерживает оптимистические блокировки, атомарные операции, синхронную репликацию и мгновенную согласованность. При сбоях кластер Aerospike может работать в режиме устойчивости к разделению («partition tolerant mode», система продолжает работу в разделенном состоянии) или в режиме согласованности («high consistency mode», часть кластера может быть отключена, чтобы не допустить несогласованности данных).

Система Aerospike оптимизирована для работы в оперативной памяти и на твердотельных носителях (Flash, SSD), при этом ключи всегда находятся в оперативной памяти. Для оптимизации сетевых взаимодействий применяются клиентские библиотеки, использующие сведения о состоянии кластера. Модель данных в Aerospike на самом верхнем уровне содержит пространства имён (namespaces), внутри которых содержатся множества (sets), которые хранят записи (records), обладающие уникальным ключом и типизированными атрибутами (bins).

## Резюме

На момент написания данной работы список NoSQL-систем класса «ключ-значение» насчитывал более 30 систем, включая такие системы, как Scalaris, Tokyo Cabinet, GenieDB, LevelDB и др. Все эти системы обладают своими особенностями и нюансами и подходят для различных задач. При выборе конкретной системы приходится принимать во внимание множество факторов, таких как желаемый уровень согласованности данных, наличие атомарных операций, легкость масштабирования и администрирования, надежность, наличие клиентских библиотек для используемого языка программирования и т.д. В целом, достоинствами систем типа «ключ-значение» являются хорошая горизонтальная масштабируемость, простота и производительность, однако часто их моделей данных оказывается недостаточно для построения серьезных приложений, где возникает потребность, например, в поиске по сочетанию атрибутов, поддержка вложенных объектов, индексы на полях объектов и т.д. В этом случае нужно обратить внимание на системы с более сложной моделью данных, например, документные СУБД.

# ДОКУМЕНТНЫЕ СУБД

Документные СУБД предоставляют больше возможностей, чем системы типа «ключ - значение». Единицей хранения данных в таких системах является документ – некоторый объект, обладающий произвольным набором атрибутов (полей), который может быть представлен, например, в JSON. Документные системы поддерживают поиск по полям документов, индексы, часто допускаются вложенные документы и массивы, а заранее предопределённой схемы данных, как правило, нет. В отличие от систем типа «ключ- значение», документные СУБД позволяют запрашивать коллекции документов на основании нескольких ограничений на атрибуты, могут осуществлять агрегатные запросы, сортировку результатов, поддерживают индексы на полях документов и т.д. Документные системы обычно не поддерживают семантику ACID, однако между собой они значительно различаются в плане поддержки согласованности данных, наличии атомарных операций, а также в способах контроля параллельного доступа к документам и во многих других аспектах.

## MongoDB

MongoDB – документная СУБД с открытым исходным кодом, написанная на C++ и разрабатываемая компанией 10gen. MongoDB обладает достаточно богатой функциональностью и является одной из самых популярных NoSQL-систем на данный момент.

MongoDB позволяет оперировать JSON-документами (хранимыми и передаваемыми в виде BSON – более компактного двоичного представления JSON), объединяемыми в коллекции, которые, в свою очередь, объединяются в базы данных. Каждый документ в коллекции должен содержать уникальный идентификатор (сгенерированный автоматически или пользователем), который не может изменяться после создания документа. Кроме идентификатора, документ также может содержать произвольный набор полей, которые могут содержать массивы и вложенные документы. Заранее предопределенной схемы данных нет: документы в одной коллекции могут содержать разные наборы полей.

Для работы с документами предусмотрены операции поиска, вставки, удаления и обновления документов. Для поиска документов в коллекции используется метод запросов по образцу, поддерживаются сортировка, проекция, просмотр результатов запроса с помощью курсора. Кроме того, поддерживается MapReduce, а также Aggregation Framework – способ формирования запроса из последовательных шагов, таких как агрегация, проекция, сортировка и т.д., что позволяет выполнять достаточно сложные аналитические запросы. Обновление документа может производиться либо полной заменой документа (с сохранением идентификатора), либо изменением полей существующего документа (в том числе добавление элемента в массив, увеличение числа и т.д.); при этом операция модификации одного документа всегда является атомарной (если обновление затрагивает несколько документов, то это уже не так). Кроме того, поддерживается атомарная операция «findAndModify», которая находит и изменяет документ, возвращая старую или новую версию. MongoDB использует блокировки для синхронизации параллельного доступа в пределах одного узла.

Для ускорения поиска документов поддерживается создание индексов на одном или нескольких полях документов в коллекции (реализованные с помощью B-деревьев), а также двумерные пространственные индексы. Существует возможность «подсказать» оптимизатору запросов, какой индекс использовать (hint), и проанализировать план выполнения (explain).

Масштабируемость в MongoDB достигается за счёт разделения документов из коллекции по узлам на основании выбранного ключа (shard key). Поддерживается асинхронная репликация в режиме «главный-подчиненный»: операции записи обрабатываются только главным узлом, а чтения могут осуществляться как с главного узла, так и с одного из подчиненных. Клиент может работать в разных режимах: неблокирующем (не дожидаясь подтверждения) или блокирующем (ожидая подтверждения от заданного количества узлов). Таким образом, MongoDB поддерживает различные модели согласованности в зависимости от того, разрешены ли чтения с вторичных узлов и от скольких узлов ожидаются подтверждения при записи. MongoDB может быть использована также и в качестве распределённой файловой системы благодаря функционалу GridFS.

Клиентские библиотеки для работы с MongoDB доступны для большого числа языков программирования, кроме того, поддерживается REST-интерфейс. Эта система используется в большом числе крупных компаний и проектов, среди которых SourceForge, Foursquare, The Guardian, Forbes, The New York Times и другие.

## CouchDB

CouchDB – проект Apache Software Foundation с открытым исходным кодом, реализованный на Erlang. CouchDB является распределённой документной СУБД, также оперирующей JSON-документами.

Заранее предопределённой схемы данных в CouchDB также не предусмотрено – документы могут содержать различные наборы полей (поддерживаются скалярные поля, массивы, вложенные документы и т.д.) и имеют уникальный идентификатор, а также номер ревизии (revision); документы организуются в базы данных. Отчёты и запросы к базам данных строятся с использованием MapReduce-представлений (views) – специальных функций на JavaScript, позволяющих задавать вид возвращаемых данных, а также выполнять агрегацию; эти функции помещаются в специальные документы (design documents). Запросы к представлениям позволяют задавать ограничения на возвращаемые данные, осуществлять сортировку, ограничивать количество возвращаемых результатов и т.д. Для представлений строятся индексы на основе B-деревьев, обновляемые при модификациях данных. Операции модификации на уровне документа обладают свойствами ACID, а читатели никогда не блокируются благодаря использованию MVCC; CouchDB поддерживает оптимистические блокировки при работе с документами. После каждой операции, изменяющей данные, происходит немедленный сброс данных на диск. Данные дописываются в конец файла, старые ревизии документов сохраняются, поэтому требуется периодически проводить сжатие (compaction) базы данных (в процессе сжатия система остается доступной для чтения и записи). Также поддерживаются JavaScript-функции для валидации данных и проверки прав доступа к ним при обновлениях.

CouchDB может масштабироваться только за счет репликации, выполняемой асинхронно. Поддерживается как схема репликации «ведущий-ведомый», так и «ведущий-ведущий». Кроме того, существует механизм условной (filtered) репликации, когда реплицируются только определённые документы. Каждый клиент видит согласованное состояние базы данных, однако эти состояния могут различаться для разных клиентов (усиленный вариант согласованности «в конечном счёте»). Когда один и тот же документ изменяется на разных узлах, возникает конфликт. При обнаружении конфликта одна из версий документа автоматически становится «победителем», а «проигравшая» версия сохраняется и может быть использована для разрешения конфликта. Существует также проект CouchDB Lounge, предоставляющий возможности шардинга для CouchDB.

Работа с CouchDB осуществляется через REST-интерфейс, клиентские библиотеки доступны для большого числа различных языков, в том числе Java, .NET, Python, PHP, Ruby и др. CouchDB используется в достаточно большом количестве проектов.

## Couchbase Server

Couchbase Server – проект, являющийся слиянием проектов Membase (система типа «ключ-значение», совместимая с Memcached) и системы CouchDB, рассмотренной ранее.

Couchbase Server может быть использован как в качестве системы управления данными типа «ключ-значение», совместимой с протоколом Memcached, так и в качестве документной СУБД, работающей с JSON-документами через REST-интерфейс. Документы могут содержать произвольный набор полей, имеют уникальный идентификатор и хранятся в «корзинах» (data bucket). Запросы осуществляют с помощью MapReduce-представлений (views) на JavaScript аналогично CouchDB. Представления строятся инкрементально и асинхронно, поэтому по умолчанию моделью согласованности является согласованность «в конечном счёте», однако на уровне операции можно указать, чтобы данные индексировались сразу же. Подсистема хранения данных также функционирует аналогично CouchDB – данные записываются в конец файла, требуется периодическое сжатие базы данных.

Важной особенностью Couchbase Server по сравнению с CouchDB является поддержка автоматического и прозрачного для приложения шардинга. Кроме того, Couchbase поддерживает два различных вида репликации – внутри кластера (intra-cluster) и межкластерную (inter-cluster, XDCR – Cross Datacenter Replication). Первый вид репликации осуществляется в пределах кластера, где узлы содержат как свои собственные данные, так и реплики других узлов, и поддерживает мгновенную согласованность на уровне документа – репликация в стиле Membase. Второй вид репликации предназначен для географически распределённых кластеров, соединённых с помощью WAN, и выполняется асинхронно, обеспечивая согласованность, в «конечном счёте» между кластерами; разрешение конфликтов в этом случае осуществляется аналогично CouchDB – в кластерах выбирается один и тот же «победитель».

Couchbase Server является новой и активно развивающейся системой с богатыми возможностями.

## Резюме

Документные СУБД имеют гибкую модель данных, которая в ряде случаев является более удобной, чем фиксированная схема, и лучше сочетается с объектно-ориентированным программированием, сокращая прослойку между языком программирования и СУБД. Системы этого класса могут достаточно легко масштабироваться, хотя делают это немного по-разному. Способы построения запросов также различаются, но в целом поддерживаются достаточно сложные выборки, включающие ограничения на значения полей, агрегацию, сортировку и т.д. К вопросу согласованности данных документные системы также подходят по-разному, обычно позволяя в определенной степени варьировать их в зависимости от конфигурации и потребностей приложения. Кроме того, могут быть реализованы дополнительные механизмы, например, оптимистические блокировки, атомарные операции и т.д., что позволяет усилить гарантии согласованности данных для тех приложений, где это необходимо. Документные СУБД обычно поддерживают постоянство хранения данных, используя запись на жесткие диски или SSD-накопители, а надежность обеспечивается с помощью журналирования и репликации; индексы и часто используемые документы при этом обычно хранятся в оперативной памяти для быстрого доступа. Транзакционная семантика на уровне нескольких документов в таких системах обычно не поддерживается, единственной возможностью является реализация на уровне приложения. Тем не менее документные СУБД по функциональности постепенно приближаются к традиционным SQL- ориентированным СУБД.

# СИСТЕМЫ ТИПА GOOGLE BIGTABLE

Разработка Google BigTable была начата в 2004 году для поддержки различных сервисов Google, таких как Google Earth, Google Maps, Google Analytic и др. BigTable базируется на Google File System (GFS, используется для хранения данных и журнала), Chubby (используется для координации и хранения некоторых метаданных) и других разработках компании и не распространяется за пределами Google, но возможность её использования предоставляется в рамках Google App Engine. BigTable проектировалась таким образом, чтобы легко масштабироваться на сотни и тысячи узлов и работать с петабайтами данных.

Таблица BigTable представляет собой отображение ключа ряда (row key), ключа столбца (column key) и временной метки (timestamp) в значение в виде строки. Ключ ряда и ключ столбца также являются обычными строками. Ключи рядов упорядочены в лексикографическом порядке, а столбцы объединены в семейства столбцов (column family), которые должны быть определены до использования, после чего в каждое семейство столбцы могут быть добавлены динамически. Семейства столбцов обычно хранят однотипные данные, и их число невелико (не более сотни), в то время как столбцов в семействе может быть неограниченное количество. Каждая ячейка таблицы может содержать несколько версий данных, помеченных временными метками и упорядоченными по ним, текущее значение имеет наибольшую временную метку, поддерживается автоматическое удаление старых версий; ячейки также могут вообще не содержать данных. Таким образом, каждая строка таблицы (ряд) может содержать произвольное число атрибутов (столбцов), входящих в заранее определённые семейства.

Ряды таблицы разделяются по диапазонам ключей, формируя относительно небольшие по размеру сегменты («tablets» в терминологии BigTable), являющиеся единицами распределения при балансировке нагрузки. Кластер BigTable содержит один главный сервер (master) и сервера, непосредственно хранящие сегменты. Главный сервер отвечает за распределение сегментов по узлам, балансировку нагрузки, операции со схемой и т.д. Семейства столбцов являются единицами контроля прав доступа и параметров хранения. Данные хранятся по столбцам, а семейства столбцов, доступ к которым обычно осуществляется вместе, могут быть выделены в группы локальности (locality groups), что позволяет оптимизировать чтения. На уровне группы можно указать, например, чтобы данные её семейств столбцов постоянно находились в оперативной памяти и читались из неё, а также настроить сжатие данных. BigTable поддерживает асинхронную репликацию между кластерами, гарантируя при этом согласованность «в конечном счёте».

В BigTable предусмотрены операции для создания и удаления таблиц и семейств столбцов, изменения метаданных (например, прав доступа), записи и удаления значений, чтения определенных строк, просмотра подмножеств данных (например, столбцов из определённого семейства). Также поддерживаются атомарные операции над строками таблицы и исполнение скриптов для обработки данных. Клиентские библиотеки кэшируют метаданные о расположении сегментов и большую часть времени обращаются непосредственно к узлам, хранящим данные. Также BigTable может быть использована с MapReduce.

Успех BigTable положил начало новому семейству систем, применяющих схожие подходы для обеспечения масштабируемости и высокой производительности.

## HBase

HBase – проект с открытым исходным кодом на Java, разрабатываемый Apache Software Foundation. HBase следует принципам BigTable и использует Apache Hadoop – набор библиотек и инструментов для разработки и выполнения распределённых вычислений.

Вместо GFS в HBase используется HDFS (Hadoop Distributed File System) – распределённая файловая система, являющаяся частью Apache Hadoop и предназначенная для надежного хранения больших файлов, распределённых по блокам между узлами. Кроме того, могут быть использованы и другие файловые системы. Также HBase поддерживает работу с Hadoop MapReduce. Роль сервиса Chubby в HBase выполняет Apache ZooKeeper.

Архитектура и функциональность HBase во многом соответствует BigTable, хотя имеются и некоторые отличия. Например, HBase поддерживает несколько главных (master) серверов, чтобы повысить надежность системы. В HBase не поддерживается концепция групп локальности (locality groups), вся конфигурация выполняется на уровне семейств столбцов. Как и BigTable, HBase не поддерживает семантику ACID в полном объеме, однако определённые свойства, усиливающие гарантии согласованности, обеспечиваются. HBase не поддерживает вторичные индексы: записи могут быть запрошены только с помощью первичного ключа или сканирования таблицы. Индексы, тем не менее, могут быть построены вручную с помощью дополнительных таблиц.

Работа с HBase может осуществляться через Java API, REST-интерфейс, а также с помощью Avro и Thrift. HBase используется в крупных и высоконагруженных приложениях и проектах, таких как Facebook (сервис Facebook Messages) и Twitter (для поддержки MapReduce, поиска по людям и других задач).

## Cassandra

Система Cassandra была разработана и использовалась в Facebook. В её основе лежат идеи Google BigTabl и Amazon Dynamo. В настоящее время Cassandra является проектом с открытым исходным кодом (на Java), поддерживаемым Apache Software Foundation.

По организации модели данных Cassandra схожа с BigTable и HBase, однако терминология и детали несколько отличаются. База данных в Cassandra называется «пространством ключей» (keyspace) и содержит семейства столбцов (column family), которые являются аналогом таблиц и служат контейнерами для строк (рядов, rows), идентифицируемых уникальными ключами (row key). Строки состоят из столбцов (column) или супер-столбцов (super column). Столбец является минимальной единицей данных в Cassandra и состоит из имени, значения и временной метки (все эти поля предоставляются клиентом), хранится только последняя версия данных (в противоположность BigTable и HBase). Супер-столбцы, в свою очередь, содержат внутри себя столбцы, добавляя тем самым еще один уровень вложенности. Кроме того, поддерживаются специальные столбцы, такие как счётчики или столбцы с указанным временем жизни (TTL). Разным строкам необязательно должен соответствовать один и тот же набор столбцов или супер-столбцов. Семейства столбцов хранятся в отдельных файлах с сортировкой по ключам строк и должны содержать столбцы, доступ к которым в запросах предполагается осуществлять вместе.

Для работы с данными Cassandra поддерживает SQL-подобный язык CQL (Cassandra Query Language), кроме того, есть поддержка Hadoop MapReduce. Для ускорения запросов поддерживается создание вторичных индексов. Операции модификации данных являются атомарными на уровне одной строки таблицы, постоянство хранения обеспечивается с помощью записи в журнал, поддерживается сжатие данных. Cassandra позволяет гибко варьировать уровень согласованности данных на уровне операций. Конфликты разрешаются на основании временных меток (выигрывает последняя версия).

Cassandra проектировалась так, чтобы обеспечить хорошую масштабируемость и надежность на большом количестве недорогих (и ненадежных) машин. В отличие от BigTable и HBase, в кластере Cassandra нет выделенных узлов, все они равноправны и выполняют одни и те же функции. Для распределения данных по узлам применяется консистентное хеширование и hinted handoff, новые узлы могут быть легко добавлены в кластер, а обнаружение сбоев и восстановление происходят автоматически. Разделение строк по узлам может осуществляться как случайным образом, так и с сохранением порядка. Репликация поддерживается как в пределах кластера, так и между географически распределёнными кластерами.

Клиентские библиотеки для работы с Cassandra доступны для большинства языков программирования (основаны на Thrift). Эта система используется во многих проектах с высокой нагрузкой.

## Резюме

Рассмотренные системы во многом следуют архитектуре и подходам, применённым в BigTable. Эти системы созданы для поддержки высоконагруженных приложений и работы на больших кластерах недорогих машин, что достигается за счёт несколько более сложной модели данных, чем документная модель: требуется внимательное проектирование семейств столбцов и выбор ключей строк, а реализация некоторых функций (например, вторичные индексы в HBase) перекладывается на разработчика.

# ДРУГИЕ СИСТЕМЫ

Как было отмечено, иногда под термином «NoSQL» понимают также вообще все системы управления данными, не являющиеся реляционными (SQL-ориентированными). Многие из этих систем появились еще до зарождения и популяризации NoSQL-движения, а также часто поддерживают ACID-транзакции и не всегда являются распределёнными, что нетипично для новых систем. Тем не менее, перечислим некоторые классы систем, не вошедшие в данный обзор, но иногда относимые к NoSQL: объектно-ориентированные СУБД, графовые системы, XML-ориентированные СУБД, многомерные системы и др.

# ЗАКЛЮЧЕНИЕ

В данной работе была дана общая характеристика достаточно нового направления в области управления данными, получившего название «NoSQL». Мотивацией к созданию этих систем послужило активное развитие Web, что привело к появлению приложений с гигантской нагрузкой и огромными объемами данных.

Таким образом, NoSQL-системы фокусируются в основном на том, чтобы обеспечить требуемую масштабируемость (и отказоустойчивость) даже за счёт снижения гарантий согласованности данных и отказа от привычной транзакционной семантики. Модели данных, поддерживаемые NoSQL-системами в целом проще, чем реляционная модель, а жестко определённой схемы данных и ограничений целостности, как правило, нет.

При использовании NoSQL-систем разработка приложений часто упрощается за счёт более простых и гибких моделей данных (например, документной) и меньшего «impedance mismatch», то есть несоответствия объектно-ориентированной модели языка программирования и модели данных используемой СУБД. К сожалению, в целом NoSQL-системы плохо подходят для задач, где требуется транзакционная семантика.

Большое разнообразие систем класса NoSQL обусловлено общей тенденцией к специализации в области СУБД: каждая NoSQL-система приспособлена для решения определённого класса задач. Таким образом, выбор конкретных решений обусловлен спецификой решаемой задачи: предполагаемая нагрузка, соотношение интенсивности чтений и записи, вид хранимых данных и типы запросов к ним, желаемый уровень согласованности, требования к надежности, наличие клиентских библиотек для выбранного языка и т.д.

В этом отношении традиционные SQL-ориентированные СУБД претендуют на некоторую универсальность, хотя их масштабируемость ограничена. Кроме того, NoSQL-системы являются достаточно молодыми по сравнению с SQL- ориентированными СУБД, поэтому значительного опыта их применения еще не накоплено.

В настоящее время также появляются новые распределённые SQL-ориентированные системы, обладающие лучшей масштабируемостью и сохраняющие поддержку SQL и ACID- транзакций.

# СПИСОК ЛИТЕРАТУРЫ

1. C. Strozzi, «NoSQL: A Relational Database Management System» [В Интернете]. URL: [http://www.strozzi.it/cgi-bin/CSA/tw7/I/en\_US/nosql/Home%20Page.](http://www.strozzi.it/cgi-bin/CSA/tw7/I/en_US/nosql/Home%20Page)
2. «NOSQL Databases,» [В Интернете]. URL: [http://nosql-database.org/.](http://nosql-database.org/)
3. R. Baldoni и M. Raynal, «Fundamentals of Distributed Computing - A Practical Tour of Vector Clock Systems» 2002. [В Интернете]. URL: [http://net.pku.edu.cn/~course/cs501/2008/reading/a\_tour\_vc.html.](http://net.pku.edu.cn/~course/cs501/2008/reading/a_tour_vc.html)
4. Веретенникова Е.Г., Патрушина С.М., Савельева Н.Г. Информатика: Учебное пособие. Серия «Учебный курс», – М., 2002.
5. Дунаев С. Доступ к базам данных и техника работы в сети. Практические приемы современного программирования. – М., 2005.
6. Мейер Д. Теория реляционных баз данных: пер. с англ. – М., 2005.
7. Ревунков Г.И., Самохвалов Э.Н., Чистов В.В. Базы и банки данных и знаний. Учебник для вузов//Под ред. В.Н.Четверикова. – М., 2003.
8. Фаронов В.В., Шумаков П.В. Руководство разработчика баз данных. – М.: Нолидж, 2000.
9. Фундаментальные основы информатики: социальная информатика.: Учебное пособие для вузов / Колин К.К. – М.: Академ.проект: Деловая книга Екатеринбург, 2000.

# Приложение 1

Таблица – Типичные функции средств DBA

|  |  |  |  |
| --- | --- | --- | --- |
| Мониторинг работы БД, реакция на нештатные ситуации | Наблюдение за объектами БД, анализ, сопоставление характеристик | Оптимизация хранения данных, оптимизация работы сервера | Сопровождение БД, файлов, табличных пространств, откатных сегментов |
| Слежение за использованием ресурсов, выдача статистики | Планирование необходимых вычислительных мощностей | Анализ свободного пространства, устранение дефрагментации | Перенос таблицы на новое пространство, в другую СУБД, на другой компьютер |
| Обнаружение и исправление возникающих неполадок | Задание пороговых значений для слежения за нужными объектами | Наблюдение за параметрами, влияющими на производительность БД | Перенос содержимого базы данных в другую СУБД |