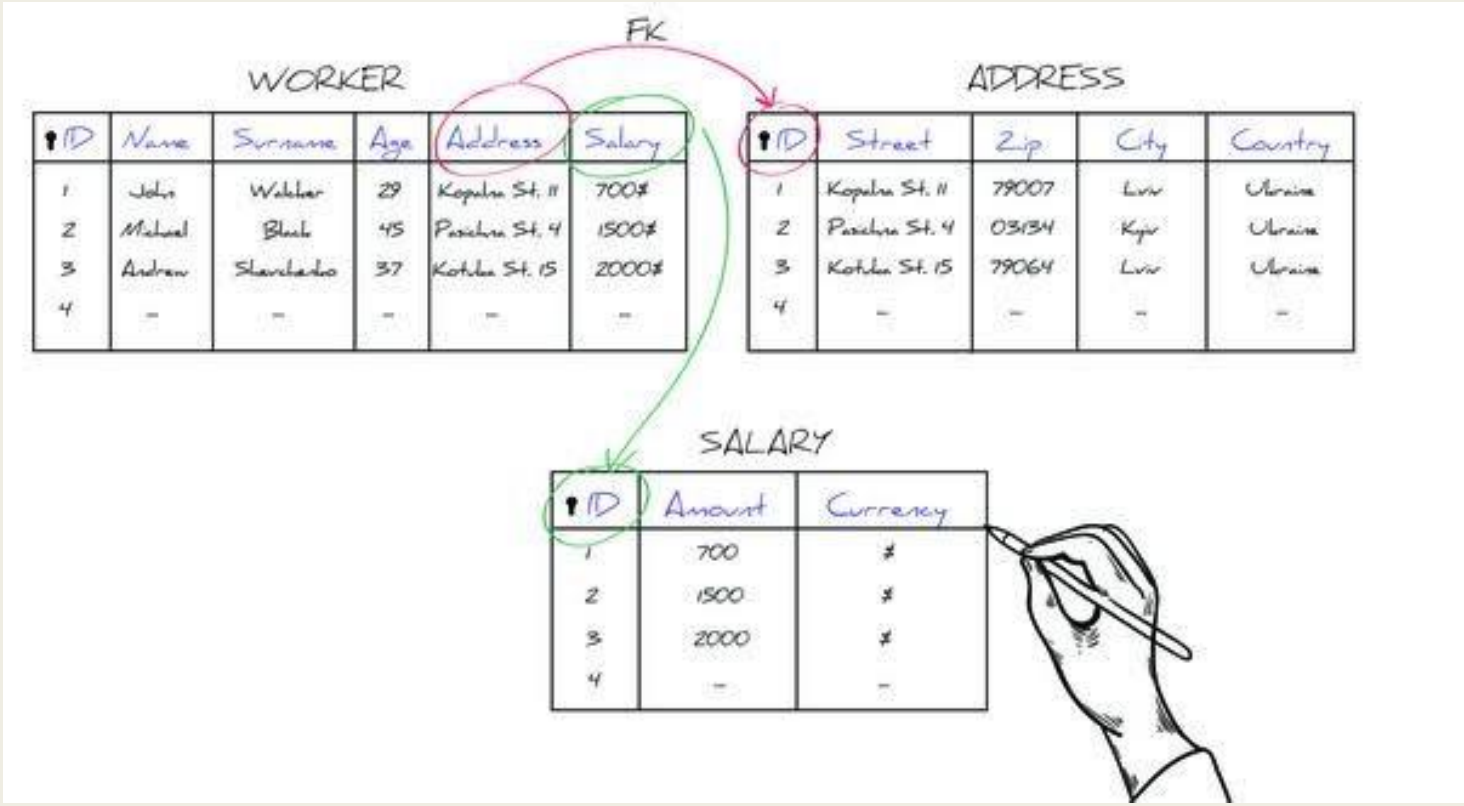
ЛЕКЦИЯ 7

Нереляционные базы данных

Реляционные базы данных (SQL)

- база, где данные хранятся в формате таблиц
- данные строго структурированы и связаны друг с другом
- в таблице есть строки и столбцы, каждая строка представляет отдельную запись, а столбец — поле с назначенным ей типом данных


FK
WORKER
ID
Name
Surname
Age
Address
Salary
1
John
Walker
29
Kopalna St. 11
700$
2
Michael
Black
45
Pasichna St. 4
1500$
3
Andrew
Slavchenko
37
Kotska St. 15
2000$
4
-
-
-
-
-
ADDRESS
ID
Street
Zip
City
Country
1
Kopalna St. 11
79007
Lviv
Ukraine
2
Pasichna St. 4
03134
Kyiv
Ukraine
3
Kotska St. 15
79064
Lviv
Ukraine
4
-
-
-
-
SALARY
ID
Amount
Currency
1
700
$
2
1500
$
3
2000
$
4
-
-

Особенности реляционных БД

- основная особенность - надежность и неизменяемость данных, низкий риск потери информации
- при обновлении данных их целостность гарантирована
- реляционные БД соответствуют требованиям, предъявляемым к транзакционным системам (ACID)
- лучшее применение: работа со структурированными данными, структура которых не подвержена частым изменениям

NoSQL

- появление ~1998, начали распространяться в конце 2000-х
- сам термин впервые появился как = No ("Не") и SQL, т.е. вообще никакого SQL-а нет, а есть какой-то другой механизм, для работы с этой базой данных
- позже стали использовать как "Not Only SQL", не только SQL, т.е. может быть SQL, но есть что-то помимо
- нереляционная база данных

Нереляционная база данных (NoSQL)

- схема данных является динамической и может меняться в любой момент времени
- к данным сложнее получить доступ, то есть найти внутри базы что-то нужное
- отличаются производительностью и скоростью
- подходят для хранения больших объемов неструктурированной информации
- хороши для быстрой разработки и тестирования гипотез

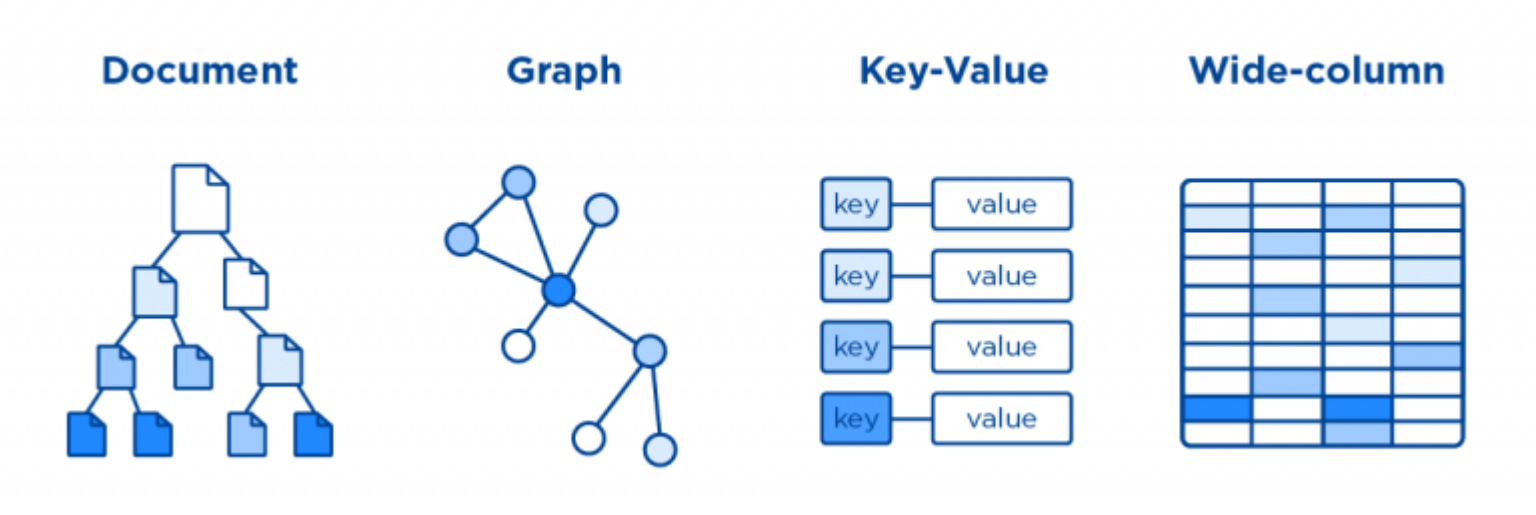
Причины появления

- увеличение объемов хранимых данных
- информация перестала быть изолированной, большая связанность между сущностями (страница в интернете ссылается на другие, теги связывают помеченную информацию из разных источников и т.д.)
- использование слабоструктурированной информации (например, описание товара в магазине, разные параметры у товаров одной категории)
- переход на микросервисную архитектуру (много сервисов, каждый со своим бэкендом и БД)

Виды нереляционных БД

Самая популярная классификация - по типу данных

- Базы данных ключ-значение (key-value)
- Колоночные СУБД
- Документ-ориентированные базы данных
- Графовые базы данных



Базы данных ключ-значение

- каждая запись имеет ключ и значение
- основное применение: данные не слишком сложные, а скорость является приоритетом
- например, для хранения данных конфигурации
- сохраненным данным не назначается никакой схемы, а сама база данных намного легче по сравнению с реляционной
- Пример такой базы данных: Memcashed, Redis, Riak

Колоночные СУБД

- основное применение: обработка больших данных
- отличаются высокой производительностью, эффективным сжатием данных и отличной масштабируемостью
- данные хранятся в виде разреженной матрицы, строки и столбцы которой используются как ключи
- Пример: HBase, Cassandra, Vertica, ClickHouse

Документ-ориентированные базы данных

- данные хранятся в коллекциях документов, обычно с использованием форматов JSON, XML или BSON
- запись может содержать столько данных, сколько нужно - ограничений нет
- документы можно вкладывать друг в друга
- вместо столбцов и строк мы описываем все данные в одном документе
- Примеры: MongoDB, CouchDB, ElasticSearch

Графовые базы данных

- состоят из узлов и связей между ними
- узлы обозначают элементы в базе данных, а связи между ними определяют их отношения между собой
- основное применение: приоритетными являются различные взаимосвязи между данным
- Пример: Neo4j, OrientDB

Базы данных ключ-значение (подробно)

Key	Value
K1	AAA,BBB,CCC
K2	AAA,BBB
K3	AAA,DDD
K4	AAA,2,01/01/2015
K5	3,ZZZ,5623

Сфера применения

- хранение пользовательских сессий (товары корзины интернет-магазина)
- хранение промежуточных данных (поток сообщений на стене, голосования, счётчики посещений)
- СУБД для небольших приложений, блогов
- кэширование данных из основного хранилища, что значительно снижает нагрузку на реляционную базу данных
- хранение «быстрых» данных — когда важны скорость и критичны задержки передачи (аналитика и анализ данных, финансовые и торговые сервисы)

Основное про key-value БД

- все данные находятся в оперативной памяти
- key-value БД очень быстрые для любых операций
- некоторые системы сохраняют данные на диск, но не синхронно с тем, как они записываются
- т.е. если сервер откажет между двумя точками синхронизации, будет потеряно всё, что осталось в памяти, но не сохранено на диск
- хранить все данные в памяти дорого
- из-за этого key-value БД обычно используют как кэш, но не как основное хранилище

Memcached vs Redis

- memcached (2003), redis (2009)
- скорость чтения/записи: практически одинаковая
- использование памяти:

redis - есть настройка максимума используемой памяти, redis никогда не будет использовать больше необходимого, и возвращает обратно неиспользуемую память

memcached - указывается максимальный размер кэша, по мере вставки элементов процесс занимает памяти чуть больше, чем ему было выделено. Но получить эту память обратно можно только перезапустив процесс

Memcached vs Redis

- сброс на диск:

  Memcached не умеет сбрасывать на диск без сторонних утилит, при перезапуске memcached, данные исчезают, это нормально для кэша redis - имеет встроенные механизмы (можно настроить, чтобы данные не сбрасывались)

- масштабирование: redis легко превращается в кластер, memcached - нет
- размер данных: memcached хранит строки до 1Мб, redis - разные данные до 512Мб

Основные команды redis

- SELECT N, где N число - выбор номера БД
- SET - установка ключа и значения. Дополнительно можно установить срок действия записи
- GET - получить значение по ключу. По умолчанию все значения - строки. Если значения нет, возвращается nil
- KEYS - возвращает все ключи по шаблону. Если ключей много, команда медленная
- EXISTS - проверка на существование значения. 1 - объект существует, 0 - нет. Типа Boolean в redis нет

Основные команды redis

- FLUSHALL - полностью удаляет данные
- GETSET - возвращает текущее значение и устанавливает новое
- DEL - удаление ключа и соответствующего значения
- TTL - позволяет посмотреть оставшееся время для ключа (если ему установлено время жизни)
- RENAME - переименование ключа

Базовые структуры redis

- строки
- списки
- хеш-таблицы
- упорядоченные множества

Хеш-таблицы

- для записи объекта используется команда HSET

  HSET имя_ключа имя_атрибута значение
- для чтения используется команда HGET

  HGET имя_ключа имя_атрибута
- HGETALL имя_ключа - получить все атрибуты и значения

Множества

- неупорядоченная коллекция неуникальных элементов
- SADD - добавить элемент в множество
- SMEMBERS - получить все элементы
- SUNION, SDIFF, SINTER - разные операции работы над множествами

Списки

- последовательность значений, упорядоченных по порядку их создания
- списки применяют для создания очередей
- LPUSH - добавить элемент в список
- LRANGE — возвращает срез списка с левой стороны

  LRANGE имя_списка индекс_начало индекс_конца

  Указание -1 означает до конца списка
- LPOP — вернуть одно значение с левой стороны и удалить его из списка. Формат LPOP имя_списка. Аналогичные команды RPUSH, RRANGE, RPOP только для правой стороны

Транзакции в redis

- Для реляционных баз: транзакция - группа команд, которые должны либо полностью выполнится или, в случае возникновение ошибки, вернуть состояние БД в исходное
- Транзакции в Redis - последовательное выполнение ранее записанных команд без возможности полноценного возвращения исходного состояния в случае ошибки исполнения
- MULTI - начать запись команд
- EXEC - выполнить ранее записанные команды: все ранее введенные команды будут исполнены один за другим
- DISCARD - отмена записи команд транзакций

Key-value БД: выводы

- Key-value базы данных очень быстрые, но могут потерять данные
- Скорость превыше всего
- Предназначены для простых операций (добавить по ключу, получить по ключу)

Колоночные СУБД

- способность хранить большое количество данных с большим количеством атрибутов
- реляционные БД хранят данные по строкам, колоночные БД хранят данные в колонках
- колонки хранятся в отдельных файлах
- это способствует улучшенному сжатию за счет информации о типе данных колонки
- это может ускорять запросы, если, например, нужны 3 колонки из 300, то не обязательно грузить остальные 297

Row-Oriented vs Column-Oriented

Row-oriented: rows stored sequentially in a file

Key	Fname	Lname	State	Zip	Phone	Age	Sales
1	Bugs	Bunny	NY	11217	(123) 938-3235	34	100
2	Yosemite	Sam	CA	95389	(234) 375-6572	52	500
3	Daffy	Duck	NY	10013	(345) 227-1810	35	200
4	Elmer	Fudd	CA	04578	(456) 882-7323	43	10
5	Witch	Hazel	CA	01970	(567) 744-0991	57	250

Column-oriented: each column is stored in a separate file
Each column for a given row is at the same offset.

Key
1
2
3
4
5

Fname
Bugs
Yosemite
Daffy
Elmer
Witch

Lname
Bunny
Sam
Duck
Fudd
Hazel

State
NY
CA
NY
CA
CA

Zip
11217
95389
10013
04578
01970

Phone
(123) 938-3235
(234) 375-6572
(345) 227-1810
(456) 882-7323
(567) 744-0991

Age
34
52
35
43
57

Sales
100
500
200
10
250

Сценарии работы с данными

- Разный порядок хранения данных лучше подходит для разных сценариев работы
- Сценарий работы с данными
  - *какие производятся запросы*
  - *как часто и в каком соотношении*
  - *сколько читается данных на запросы каждого вида - строк, столбцов, байт*
  - *как соотносятся чтения и обновления данных*
  - *какой рабочий размер данных и насколько локально он используется*
  - *используются ли транзакции и с какой изолированностью*
  - *какие требования к дублированию данных и логической целостности*
  - *требования к задержкам на выполнение и пропускной способности запросов каждого вида*

Специализация БД

- Чем больше нагрузка на систему, тем более важной становится специализация под сценарий работы, и тем более конкретной становится эта специализация
- Не существует системы, одинаково хорошо подходящей под существенно различные сценарии работы
- Если система подходит под широкое множество сценариев работы, то при достаточно большой нагрузке, система будет справляться со всеми сценариями работы плохо, или справляться хорошо только с одним из сценариев работы

Популярность колоночных БД

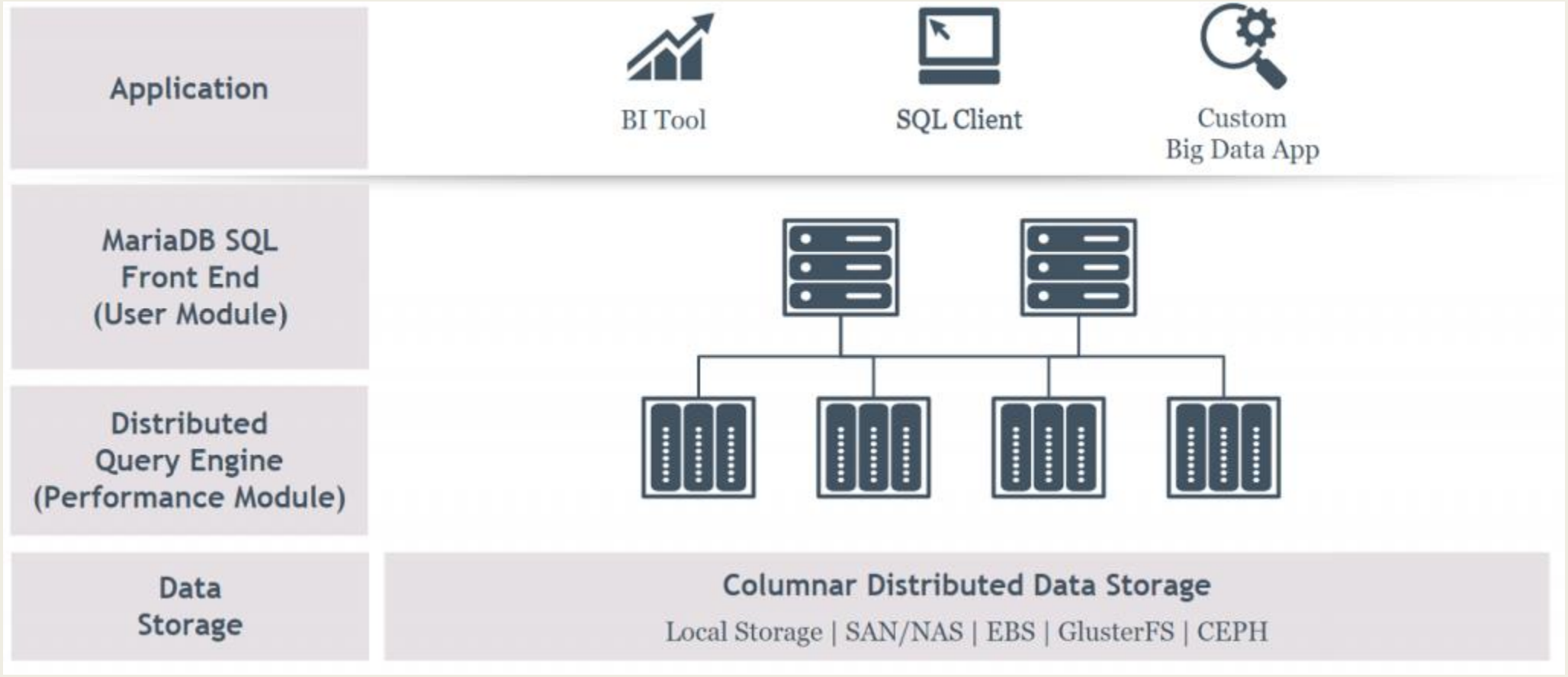
- ClickHouse - 1384 вакансии
- Cassandra - 652 вакансии
- MariaDB - 368 вакансий
- Hbase - 249 вакансий

Hbase

- HBase - это база данных, построенная на основе HDFS
- HDFS - это распределенная файловая система, подходящая для хранения больших файлов
- обеспечивает быстрый поиск для больших таблиц
- обеспечивает доступ с низкой задержкой к отдельным строкам из миллиардов записей (произвольный доступ)
- HBase линейно масштабируется.
- имеет автоматическую поддержку при сбоях.
- интегрируется с Hadoop как в качестве источника, так и в качестве пункта назначения
- обеспечивает репликацию данных по кластерам
- используется всякий раз, когда есть необходимость писать тяжелые приложения

MariaDB

- MariaDB — ответвление СУБД MySQL
- добавлены оптимизации, которые повышают производительность СУБД по сравнению с оригинальным MySQL
- таблицы представлены не в форме построчного хранилища, а в форме колоночного хранилища
- легко заменяет MySQL (по словам разработчиков)
- для каждой версии MySQL они выпускают тот же номер версии MariaDB
- поддерживает SQL


Application
BI Tool
SQL Client
Custom
Big Data App
MariaDB SQL
Front End
(User Module)
Distributed
Query Engine
(Performance Module)
Data
Storage
Columnar Distributed Data Storage
Local Storage | SAN/NAS | EBS | GlusterFS | CEPH

Cassandra: Идеальные условия

- запись превосходит чтение на порядок
- данные редко обновляются
- чтение происходит по известному первичному ключу
- данные можно разбить на партиции по ключу, что позволит БД быть распределённой по многим узлам
- не нужно соединять таблицы или использовать агрегатные функции

Cassandra: Лучшие варианты использования

- логирование транзакций (заказы и т.д.)
- хранение временного ряда данных
- отслеживание статусов заказа
- данные о прогнозе погоды
- данные от умных часов

ClickHouse: условия

- подавляющее большинство запросов - на чтение
- данные обновляются достаточно большими пачками (> 1000 строк), а не по одной строке, или не обновляются вообще
- данные добавляются в БД, но не изменяются
- при чтении, вынимается достаточно большое количество строк из БД, но только небольшое подмножество столбцов
- значения в столбцах достаточно мелкие - числа и небольшие строки (пример - 60 байт на URL)
- требуется высокая пропускная способность при обработке одного запроса (до миллиардов строк в секунду на один сервер)
- транзакции отсутствуют
- результат выполнения запроса существенно меньше исходных данных - то есть, данные фильтруются или агрегируются

ClickHouse: сфера применения

- веб-аналитика и контекстная реклама
- real time мониторинг бизнес-метрик, например, анализ потребительского поведения на сайте
- интерактивное взаимодействие с пользователями, к примеру, онлайн-игры
- контроль технических показателей, в т.ч. интернет вещей

OLAP-сценарий

- Столбцовые СУБД лучше (от 100 раз по скорости обработки большинства запросов) подходят для OLAP сценария работы

https://clickhouse.tech/docs/ru/

- Для выполнения аналитического запроса, требуется прочитать небольшое количество столбцов таблицы. В столбцовой БД для этого можно читать только нужные данные. Например, если вам требуется только 5 столбцов из 100, то следует рассчитывать на 20-кратное уменьшение ввода-вывода.

ClickHouse

- Плюсы
  - *база данных для аналитики, исследовательских работ и real-time отчётов*
  - *легко обрабатывает большие потоки входящих данных*
  - *может работать с петабайтами данных*
  - *эффективно работает с HDD (а не только SSD)*
- Минусы
  - *жёсткая схема данных (как в реляционной схеме)*
  - *неэффективные большие JOIN'ы. Ограничения на sub-select'ы. Совсем произвольный отчёт из ста больше таблиц не так просто построить*
  - *свой SQL. Нужно вручную различать и понимать разницу между LOCAL \ GLOBAL JOIN, ANY \ ALL, WHERE \ PREWHERE. Непросто будет мигрировать на другой SQL-совместимый софт.*
  - *нет транзакций, нет апдейтов в простой форме*

https://play.clickhouse.tech/

Колоночные БД: выводы

- сильная сторона в способности хранить большое количество данных с большим количеством атрибутов
- основное применение: аналитика; большой поток данных