Финальное задание

Нужно либо ответить на вопросы ниже, либо развернуть etl на основе Apache NiFi и Cassandra (но можно и что-нибудь попроще, например PostgreSQL). Если будете создавать etl, жду описание шагов, что как и зачем делали.

1. **Что такое Hadoop?**

Hadoop — свободно распространяемый набор утилит, библиотек и фреймворк для разработки и выполнения распределённых программ, работающих на кластерах из сотен и тысяч узлов. Используется для реализации поисковых и контекстных механизмов многих высоконагруженных веб-сайтов. Разработан на Java в рамках вычислительной парадигмы MapReduce, согласно которой приложение разделяется на большое количество одинаковых элементарных заданий, выполнимых на узлах кластера и естественным образом сводимых в конечный результат.

2. **Что такое HDFS?**

HDFS (Hadoop Distributed File System) — файловая система, предназначенная для хранения файлов больших размеров, поблочно распределённых между узлами вычислительного кластера. Все блоки в HDFS (кроме последнего блока файла) имеют одинаковый размер, и каждый блок может быть размещён на нескольких узлах, размер блока и коэффициент репликации (количество узлов, на которых должен быть размещён каждый блок) определяются в настройках на уровне файла. Благодаря репликации обеспечивается устойчивость распределённой системы к отказам отдельных узлов. Файлы в HDFS могут быть записаны лишь однажды (модификация не поддерживается), а запись в файл в одно время может вести только один процесс. Организация файлов в пространстве имён — традиционная иерархическая: есть корневой каталог, поддерживается вложение каталогов, в одном каталоге могут располагаться и файлы, и другие каталоги.

3. ***Что такое YARN?**

YARN (Yet Another Resource Negotiator — «ещё один ресурсный посредник») — модуль, появившийся с версией 2.0 (2013), отвечающий за управление ресурсами кластеров и планирование заданий. Если в предыдущих выпусках эта функция была интегрирована в модуль MapReduce, где была реализована единым компонентом (JobTracker), то в YARN функционирует логически самостоятельный демон — планировщик ресурсов (ResourceManager), абстрагирующий все вычислительные ресурсы кластера и управляющий их предоставлением приложениям распределённой обработки. Работать под управлением YARN могут как MapReduce - программы, так и любые другие распределённые приложения, поддерживающие соответствующие программные интерфейсы; YARN обеспечивает возможность параллельного выполнения нескольких различных задач в рамках кластера и их изоляцию (по принципам мультиарендности). Разработчику распределённого приложения необходимо реализовать специальный класс управления приложением (Application Master), который отвечает за координацию заданий в рамках тех ресурсов, которые предоставит планировщик ресурсов; планировщик ресурсов же отвечает за создание экземпляров класса управления приложением и взаимодействия с ним через соответствующий сетевой протокол. YARN может быть рассмотрен как кластерная операционная система в том смысле, что выступает интерфейсом между аппаратными ресурсами кластера и широким классом приложений, использующих его мощности для выполнения вычислительной обработки.

4. **Какие минусы или опасные места HDFS?**

Каждый DataNode периодически отправляет сообщение Heartbeat в NameNode. Сетевой раздел может привести к тому, что подмножество DataNode теряет связь с NameNode. NameNode обнаруживает это состояние по отсутствию сообщения Heartbeat. NameNode помечает узлы данных без недавних сообщений как мертвые и не пересылает им новые запросы ввода-вывода. Любые данные, которые были зарегистрированы на мертвом DataNode, больше не доступны для HDFS. Сбой DataNode может привести к тому, что коэффициент репликации некоторых блоков упадет ниже заданного значения. NameNode постоянно отслеживает, какие блоки должны быть реплицированы, и инициирует репликацию при необходимости. Необходимость повторной репликации может возникнуть по многим причинам: узел данных может стать недоступным, реплика может быть повреждена, жесткий диск на узле данных может выйти из строя или коэффициент репликации файла может быть увеличен.

FsImage и EditLog являются центральными структурами данных HDFS. Повреждение этих файлов может привести к неработоспособности экземпляра HDFS. По этой причине NameNode может быть настроен для поддержки нескольких копий FsImage и EditLog. Любое обновление FsImage или EditLog приводит к тому, что каждый из FsImages и EditLogs обновляется синхронно. Это синхронное обновление нескольких копий FsImage и EditLog может снизить скорость транзакций в секунду, поддерживаемую NameNode. Однако это ухудшение допустимо, поскольку, хотя приложения HDFS по своей природе очень интенсивно используют данные, они не требуют интенсивного использования метаданных. Когда NameNode перезапускается, он выбирает последний совместимый FsImage и EditLog для использования. Машина NameNode является единственной точкой отказа для кластера HDFS. Если машина NameNode выходит из строя, необходимо ручное вмешательство. В настоящее время автоматический перезапуск и аварийное переключение программного обеспечения NameNode на другой компьютер не поддерживается.

Также HDFS может быть очень медленным. В HDFS записать можно, а изменить нет.

5. **Что такое блок HDFS?**

HDFS имеет концепцию хранения данных в блоках. Блоки - это физические разделы данных в HDFS. Всякий раз, когда файл загружается в HDFS, он физически разделяется на разные части, известные как блоки. Количество блоков зависит от значения dfs.block.size в hdfs-site.xml.

6. **Для чего используется NameNode?**

NameNode (управляющий узел, узел имен или сервер имен) – отдельный, единственный в кластере, сервер с программным кодом для управления пространством имен файловой системы, хранящий дерево файлов, а также мета-данные файлов и каталогов, который отвечает за открытие и закрытие файлов, создание и удаление каталогов, управление доступом со стороны внешних клиентов и соответствие между файлами и блоками, дублированными (реплицированными) на узлах данных. Сервер имён раскрывает для всех желающих расположение блоков данных на машинах кластера.

Развёртывание экземпляра HDFS предусматривает наличие центрального узла имён (NameNode), хранящего метаданные файловой системы и метаинформацию о распределении блоков, и серии узлов данных (DataNode), непосредственно хранящих блоки файлов. Узел имён отвечает за обработку операций уровня файлов и

каталогов — открытие и закрытие файлов, манипуляция с каталогами, узлы данных непосредственно отрабатывают операции по записи и чтению данных. Узел имён и узлы данных снабжаются веб-серверами, отображающими текущий статус узлов и позволяющими просматривать содержимое файловой системы. Административные функции доступны из интерфейса командной строки.

7. **Для чего используется DataNode?**

DataNode, Node (узел или сервер данных) – один их множества серверов кластера с программным кодом, отвечающим за файловые операции и работу с блоками данных. DataNode отвечает за запись и чтение данных, выполнение команд от узла NameNode по созданию, удалению и репликации блоков, а также периодическую отправку сообщения о состоянии (heartbeats) и обработку запросов на чтение и запись, поступающих от клиентов файловой системы HDFS. Данные проходят с остальных узлов кластера к клиенту мимо узла NameNode.

8. **Что будет, если записать много маленьких файлов в HDFS?**

В идеале размер блока должен быть большим, например 64/128/256 МБ (по сравнению с 4 КБ в нормальной файловой системе). Значение размера блока по умолчанию в большинстве дистрибутивов Hadoop 2.x составляет 128 МБ. Причина более высокого размера блока заключается в том, что Hadoop предназначен для обработки PetaBytes данных с каждым файлом в диапазоне от нескольких сотен мегабайт до порядка терабайт. Например, у вас есть файл размером 1024 МБ. Если размер вашего блока составляет 128 МБ, вы получите 8 блоков по 128 МБ каждый. Это означает, что ваш NameNode должен будет хранить метаданные 8 x 3 = 24 файла (3 - фактор репликации). Рассмотрим тот же сценарий с размером блока 4 КБ. Это приведет к 1 ГБ / 4 КБ = 250000 блоков, и это потребует от NameNode сохранить метаданные для 750000 блоков только для файла размером 1 ГБ. Поскольку вся эта информация, связанная с метаданными, хранится в памяти, предпочтительнее использовать больший размер блока, чтобы сэкономить этот бит дополнительной нагрузки на NameNode. Опять же, размер блока не установлен на очень высокое значение, как например 1 ГБ и т.д., потому что в идеале для каждого блока данных запускается 1 tapper. Поэтому, если вы установите размер блока на 1 ГБ, вы можете потерять параллелизм, что может привести к снижению общей пропускной способности. Создавая файл, клиент может явно указать размер блока файла (по умолчанию 64 Мб) и количество создаваемых реплик (по умолчанию значение равно 3-ем).

9. **Что будет, если несколько DataNode внезапно отключатся?**

В этом случае воспользуемся Secondary NameNode. Secondary NameNode (вторичный узел имен) — копирует образ HDFS и лог транзакций операций с файловыми блоками во временную папку, применяет изменения, накопленные в логе транзакций к образу HDFS, а также записывает его на узел NameNode и очищает лог транзакций. Secondary NameNode необходим для быстрого ручного восстанавления NameNode в случае его выхода из строя.

10. **Как проадпейдить несколько записи в большом файле на hdfs?**

HDFS предназначена для больших данных, поэтому размер файлов, которые хранится в ней, существенно выше чем в локальных файловых системах – более 10 GB. Информация записывается в потоковом режиме, за счет чего достигается высокая пропускная способность. Клиент, осуществляющий запись, кэширует данные во временном локальном файле, пока их объем не достигнет размера одного HDFS-блока (по умолчанию 64 MB). Накопив данные на один блок, клиент отправляет на сервер

имен NameNode запрос на создание файла, указав размер блока для создаваемого файла и количество реплик.
Сервер имен регистрирует новый файл, выделяет блок и возвращает клиенту идентификаторы узлов данных для хранения копий (реплик) этого HDFS-блока. Также сервер имен уведомляет другие узлы данных, на которые будут писаться реплики файлового блока. Все блоки в HDFS, кроме последнего блока файла, имеют одинаковый размер. Каждый блок может быть размещён на нескольких узлах.
Размер блока и коэффициент репликации – число узлов для размещения каждого HDFS-блока – определяются в файловых настройках.
Репликация обеспечивает устойчивость системы к отказам отдельных узлов. Клиент начинает передачу данных блока из временного файла первому по списку узлу данных, который сохраняет информацию на своем диске и пересылает ее следующему серверу данных. Второй узел передает данные третьему и далее: файловый поток передается в конвейерном режиме и автоматически реплицируется на нужном количестве узлов. Все HDFS-блоки реплицируются столько раз, сколько было указано клиентом (по умолчанию 3). Для повышения надежности узлы для хранения 2-ой и 3-ей реплики располагаются в разных серверных стойках. Расположение следующих реплик вычисляется произвольно. Для защиты от сбоев можно настроить кластер так, чтобы сервер имен знал, на каких серверных стойках расположены узлы данных. Для этого используется специальный механизм Hadoop — rack awareness.
По завершении записи HDFS-блока каждый узел данных из цепочки в обратном порядке (т.е. с конца) отправляет клиенту сообщения об успешной операции. После этих подтверждений клиент оповещает сервер имен об успешной записи блока и получает цепочку узлов данных для записи 2-го блока и т.д.
Если сервер имен не принимает от узла данных heartbeat-сообщений, он считает этот DataNode вышедшим из строя (умершим) и реплицирует данные, хранящиеся на этом узле, на другие сервера данных (живые). Если клиент смог успешно записать блок хотя бы на 1 узел, можно не беспокоиться за дальнейшую репликацию – это находится в сфере ответственности сервера имен, который сам обеспечит распространение информации на нужном уровне.
Окончив запись, клиент уведомляет сервер имен NameNode, который фиксирует транзакцию создания файла в своем журнале. После этого HDFS-файл становится доступным для использования: чтения, повторной репликации или удаления.

11. ***Почему задачи на YARN нестабильны?**

???

12. **Что такое Hive?**

Apache Hive — система управления базами данных на основе платформы Hadoop. Позволяет выполнять запросы, агрегировать и анализировать данные, хранящиеся в Hadoop. Apache Hive был создан корпорацией Facebook и передан под открытой лицензией в собственность фонду Apache Software Foundation. На сегодняшний день эта система используется компанией Netflix и доступна в Amazon Web Services через Amazon Elastic MapReduce.

13. **Что хранит HiveMetastore?**

Hive Metastore работает с процессором метаданных Hive и Hadoop FS или MapR FS как часть решения Drift Synchronization Solution для Hive.
Hive Metastore использует записи метаданных, сгенерированные процессором Hive Metadata, для создания и обновления таблиц Hive. Это позволяет адресатам Hadoop FS и MapR FS записывать дрейфующие данные Avro или Parquet в HDFS или MapR FS.

Hive Metastore сравнивает информацию в записях метаданных с таблицами Hive, а затем создает или обновляет таблицы по мере необходимости. Например, когда обработчик метаданных Hive встречает запись, для которой требуется новая таблица Hive, он передает запись метаданных в Hive Metastore, и Hive Metastore создает таблицу.
Имена Hive таблиц, имена столбцов и имена разделов создаются строчными буквами. Имена, содержащие прописные буквы, в Hive становятся строчными.
Hive Metastore не обрабатывает данные. Он обрабатывает только записи метаданных, сгенерированные процессором метаданных Hive, и должны быть ниже по потоку от выходного потока метаданных процессора.
При настройке Hive Metastore вы определяете информацию о подключении для Hive, расположение файлов конфигурации Hive и Hadoop и, при необходимости, указываете дополнительные обязательные свойства. Вы также можете включить аутентификацию Kerberos. Вы также можете установить максимальный размер кэша для Hive Metastore, определить, как создаются и хранятся новые таблицы, и настроить настраиваемые атрибуты заголовка записи.
Hive Metastore также может генерировать события для потока событий.
При использовании Hive Metastore в нескольких конвейерах старайтесь избегать одновременных или конфликтующих записей в одни и те же таблицы.

14. **Чем отличается external table и managed table?**

Hive fundamentally knows two different types of tables:

- *Managed (Internal)*
- *External*

Таблицы Hive бывают двух типов: внутренняя таблица, также известная как управляемая таблица, и внешняя таблица.
Данные внутренней таблицы управляются Hive. Hive не несет ответственности за управление данными внешней таблицы.
При удалении внутренней таблицы Hive удаляются и данные таблицы, и метаданные, а при удалении внешней таблицы Hive удаляются только метаданные таблицы.
Мы можем идентифицировать внутренние или внешние таблицы с помощью оператора DESCRIBE FORMATTED table_name в Hive, который будет отображать либо MANAGED_TABLE, либо EXTERNAL_TABLE в зависимости от типа таблицы.

1. Внутренняя таблица
Это таблица по умолчанию в Hive. Когда пользователь создает таблицу в Hive, не указывая ее как внешнюю, по умолчанию внутренняя таблица создается в определенном месте в HDFS. По умолчанию внутренняя таблица будет создана в пути к папке, аналогичном каталогу / user / hive / хранилище HDFS. Мы можем переопределить местоположение по умолчанию с помощью свойства location во время создания таблицы. Если мы отбросим управляемую таблицу или раздел, данные таблицы и связанные с ней метаданные будут удалены из HDFS.

2. Внешняя таблица
Hive не управляет данными внешней таблицы.
Мы создаем внешнюю таблицу для внешнего использования, когда хотим использовать данные вне Hive. Внешние таблицы хранятся вне каталога хранилища. Они могут получить доступ к данным, хранящимся в таких источниках, как удаленные расположения HDFS или тома хранилища Azure.
Каждый раз, когда мы удаляем внешнюю таблицу, удаляются только метаданные, связанные с таблицей, данные таблицы остаются нетронутыми Hive. Мы можем

создать внешнюю таблицу, указав ключевое слово EXTERNAL в операторе создания таблицы Hive. Семантика загрузки различается в обеих таблицах. Посмотрим на разницу в семантике загрузки между внутренней и внешней таблицами.
а. Внутренняя таблица
Когда мы загружаем данные во внутреннюю таблицу, Hive перемещает данные в каталог хранилища. Например: мы создаем таблицу «internaldemo». Когда мы загружаем данные в таблицу «internaldemo», Hive перемещает данные в каталог хранилища.
б. Внешняя таблица
С помощью ключевого слова EXTERNAL Hive знает, что он не управляет данными таблицы, поэтому не перемещает данные в свой каталог хранилища. Hive даже не проверяет, существует ли внешнее местоположение на момент его определения.
Например: мы создаем внешнюю таблицу «external_demo» в указанном месте, то есть «/ home / dataflair /». При загрузке данных во внешнюю таблицу Hive не перемещает данные таблицы в свой каталог хранилища. При удалении внешней таблицы удаляются только метаданные таблицы. Содержимое таблицы остается нетронутым.
Когда использовать внутреннюю и внешнюю таблицы?
1. Внутренняя таблица
Мы можем использовать внутреннюю таблицу в случаях:
При создании временных таблиц.
При необходимости, Hive должен управлять жизненным циклом таблицы.
И когда нам не нужны данные таблицы после удаления.
2. Внешняя таблица
Мы можем использовать внешнюю таблицу в случаях:
Когда мы не создаем таблицу на основе существующей таблицы.
Когда требуется использовать данные за пределами Hive. Например, файлы данных читаются и обрабатываются существующей программой, которая не блокирует файлы. Когда мы не хотим полностью удалять данные таблицы даже после DROP.
Когда данные не должны принадлежать Hive.

15. ***Какие форматы умеет читать Hive?**

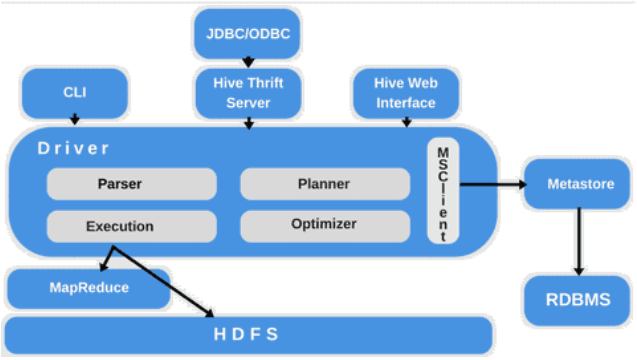
Hive поддерживает типы данных List, Map, DateTime, BigInt и Uint8List.
Таблица в hive представляет из себя аналог таблицы в классической реляционной БД. Основное отличие — что данные hive'овских таблиц хранятся просто в виде обычных файлов на hdfs. Это могут быть обычные текстовые csv-файлы, бинарные sequence-файлы, более сложные колоночные parquet-файлы и другие форматы.

16. ***Чем отличается управление ресурсов в Hive и Impala?**

10 ключевых отличий:

- ***Режим обработки данных*** *– Impala обрабатывает SQL-запросы на лету, реализуя интерактивные вычисления в режиме онлайн, тогда как Hive не походит для OLTP-задач, т.к. работает с пакетами данных не в реальном времени.*
- ***Языки разработки*** *– Hive написан на Java, тогда как Impala – на C++. Это обусловливает более эффективное использование памяти в Impala, несмотря на то в Hive повторно используются экземпляры JVM (Java Virtual Machine), чтобы частично снизить накладные расходы при запуске виртуальной машины. Однако, Impala, разработанный на C++, иногда может не работать с форматами, написанными на Java.*

- ***Форматы данных*** *– Impala работает с форматами LZO, Avro и Parquet, а Hive – с Plain Text и ORC. При этом обе рассматриваемые системы поддерживают форматы RCFIle и Sequence.*
- ***Вычислительная модель*** *– Impala основана на архитектуре массовой параллельной обработки (MPP, Massive Parallel Processing), благодаря чему реализуется распределенное многоуровневое обслуживание дерева для отправки SQL-запросов с последующей агрегацией результатов из его листьев. Также MPP позволяет Impala распараллеливать обработку данных, поддерживая интерактивные вычисления. В свою очередь, Hive используется технологию MapReduce, преобразуя SQL-запросы в задания Apache Spark или Hadoop.*
- ***Задержка обработки данных (latency)*** *– в связи с разными вычислительными моделями, рассматриваемые системы по-разному обрабатывают информацию. Cloudera Impala выполняет SQL-запросы в режиме реального времени. Для Apache Hive характерна высокая временная задержка и низкая скорость обработки данных. Impala способна работать в 6-69 раз быстрее Hive с простыми SQL-запросами. Однако, со сложными запросами Hive справляется лучше благодаря LLAP.*
- ***Пропускная способность*** *Hive существенно выше, чем у Impala. LLAP-функция (Live Long and Process), которая разрешает кэширование запросов в памяти, обеспечивает Hive хорошую производительность на низком уровне. LLAP включает долговременные системные службы (демоны), что позволяет напрямую взаимодействовать с узлами данных HDFS и заменяет тесно интегрированную DAG-структуру запросов (Directed acyclic graph) – графовую модель, активно используемую в Big Data вычислениях.*
- ***Кодогенерация*** *– Hive генерирует выражения запросов во время компиляции (compile time), тогда как Impala — во время выполнения (runtime). Для Hive характерна проблема «холодного старта», когда при первом запуске приложения запросы выполняются медленно из-за необходимости установки подключения к источнику данных. В Impala отсутствуют подобные накладные расходы при запуске, т.к. необходимые системные службы (демоны) для обработки SQL-запросов запускаются во время загрузки (boot time), что ускоряет работу.*
- ***Отказоустойчивость*** *– Hive является отказоустойчивой системой, которая сохраняет все промежуточные результаты. Это также положительно влияет на масштабируемость, однако приводит к снижению скорости обработки данных. В свою очередь, Impala нельзя назвать отказоустойчивой платформой.*
- ***Безопасность*** *– в Hive отсутствуют инструменты обеспечения кибер-безопасности, тогда как Impala поддерживает аутентификацию Kerberos.*
- ***Основные пользователи*** *– с учетом отказоустойчивости и высокой пропускной способности Hive, эта система более адаптирована для промышленной эксплуатации в условиях высоких нагрузок, когда допустима некоторая временная задержка (latency). Поэтому она в большей степени востребована у инженеров больших данных (Data Engineer) в рамках построения сложных ETL-конвейеров. А быстрая и безопасная, но не слишком надежная Impala лучше подходит для менее масштабных проектов и пользуется популярностью у аналитиков и ученых по данным (Data Analyst, Data Scientist).*



Архитектура Apache Hive

Таким образом, рассмотренные сходства и различия Cloudera Impala и Apache Hive подтверждают, что данные SQL-инструменты для аналитической обработки данных, хранящихся в экосистеме Hadoop, не конкурируют, а дополняют друг друга. Аргументы выбора того или иного решения мы приводим здесь, вместе с реальными примерами использования.

17. **Чем отличается колочный формат хранения данных от строчного?**

Все многообразие файловых форматов Big Data (AVRO, Sequence, Parquet, ORC, RCFile) можно разделить на 2 категории: линейные (строковые) и колоночные (столбцовые).

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

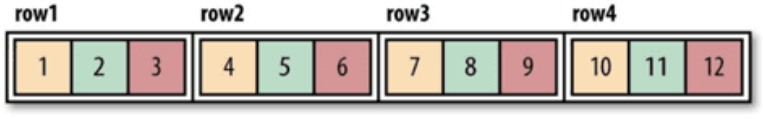
Zip
11217
95389
10013
04578
01970

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

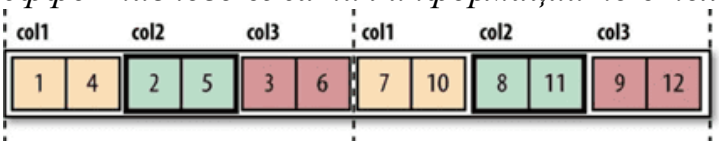
Age
34
52
35
43
57

Sales
100
500
200
10
250

***В линейных форматах** (AVRO, Sequence) строки данных одного типа хранятся вместе, образуя непрерывное хранилище. Даже если необходимо получить лишь некоторые значения из строки, все равно вся строка будет считана с диска в память. Линейный способ хранения данных обусловливает пониженную скорость операций чтения и выполнении избирательных запросов, а также больший расход дискового пространства. Линейные форматы, в отличие от колоночных, не являются строго типизированными: например, Apache AVRO хранит информация о типе каждого поля в разделе метаданных вместе со схемой, поэтому для чтения сериализованных данных информации не требуется предварительное знание схемы. Бинарный формат файлов последовательностей (Sequence File) для хранения Big Data в виде сериализованных пар ключ/значение в экосистеме Apache Hadoop также содержит метаданные в заголовке файла. Формат Sequence File отлично обеспечивает параллелизм при выполнении задач MapReduce, т.к. разные порции одного файла могут быть распакованы и использованы независимо друг от друга. Тем не менее, степень сжатия информации у строковых форматов ниже, чем у столбцовых. Однако, именно линейно-ориентированные форматы лучше всего подходят для потоковой записи, т.к. в случае сбоя информация может быть восстановлена (повторно синхронизирована) с последней точки синхронизации.*



В колоночно-ориентированных форматах *(Parquet, RCFile, ORCFile) файл разрезается на несколько столбцов данных, которые хранятся вместе, но могут быть обработаны независимо друг от друга. Такой метод хранения информации позволяет пропускать ненужные столбцы при чтении данных, что существенно ускоряет чтение данных и отлично подходит в случае, когда необходим небольшой объем строк или выполняются избирательные запросы, как, например, в СУБД Apache Hive. Но такой формат чтения и записи занимает больше места в оперативной памяти, поскольку, чтобы получить столбец из нескольких строк, кэшируется каждая строка. Также колоночно-ориентированные форматы не используются в средах потоковой обработки (Apache Kafka, Flume), т.к. после сбоя записи текущий файл не сможет быть восстановлен из-за отсутствия точек синхронизации. Однако, колоночные файлы занимают меньше места на жестком диске вследствие более эффективного сжатия информации по столбцам.*



18. **Чем отличается parquet/orc от csv?**

csv широко используется, большинство систем ввода имеет формат csv, кроме того, файл csv можно понимать и читать напрямую, файл csv также поддерживает добавление данных в конце, но файл csv, поскольку сжатие отсутствует, объем файла большой, то некоторые операции не так хороши, как хранение в столбцах.

Паркет также широко используется. По умолчанию используется сжатие gzip, которое является более громоздким и более эффективным. Однако паркет не может быть понят и прочитан, потому что он использует двоичное хранилище. Когда в системе нет особых требований, файлы данных должны быть открыты.

orc также представляет собой список сохраненных двоичных файлов, которые не могут быть непосредственно поняты и поняты при открытии, но потому что в формате файлов каждый блок строк имеет легкий индекс в начале, поэтому по сравнению с паркетом, при поиске строк, orc скорость относительно быстрее.

csv

Файл данных csv относится к режиму хранения текста. Spark поддерживает его по умолчанию и записывает его в файл в соответствии с текстом. Каждая строка имеет одну запись. Вообще говоря, метод хранения текста несжатый, а производительность относительно высокая.

parquet

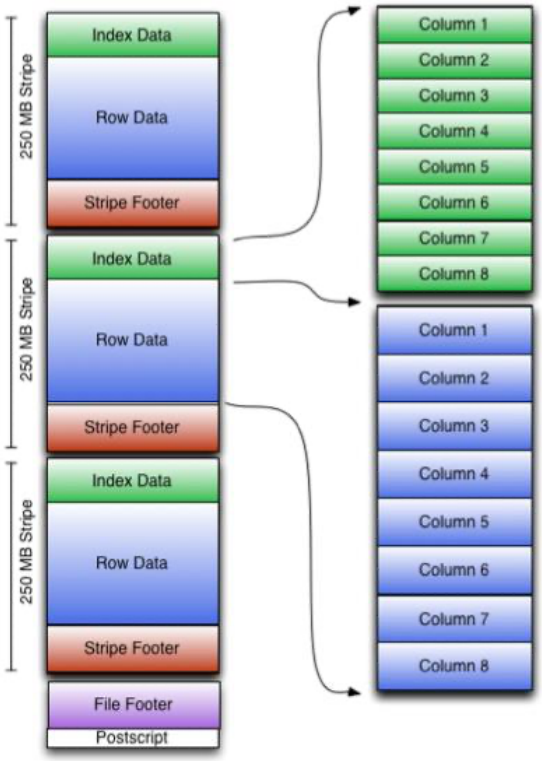
Apache Parquet - это новый тип столбчатого формата хранения в экосистеме Hadoop. Он совместим с большинством вычислительных сред (Mapreduce, Spark и т. д.) в экосистеме Hadoop и поддерживается несколькими механизмами запросов (Hive, Impala, Drill и т. Д.), И это не зависит от языка и платформы. Первоначально паркет был разработан Twitter и Cloudera в сотрудничестве и с открытым исходным кодом, а в мае 2015 года он окончил инкубатор Apache и стал главным проектом Apache.

Файлы паркета хранятся в двоичном формате и не могут быть непосредственно прочитаны и изменены. Файлы паркета разбираются автоматически, а данные и метаданные файла включаются в файл. В файловой системе HDFS и файлах Parquet существуют следующие концепции:

Блок HDFS (Блок): это самая маленькая единица копирования в HDFS. HDFS будет хранить блок в локальном файле и поддерживать несколько копий, разбросанных по разным машинам. Обычно размер блока составляет 256 МБ. 512M и т. Д.

Файл HDFS (Файл). Файл HDFS, включая данные и метаданные, данные распределяются и хранятся в нескольких блоках.
Группа строк: данные физически разделены на несколько блоков в соответствии со строками. Каждая группа строк содержит определенное количество строк. По крайней мере одна группа строк хранится в файле HDFS, Parquet При чтении и записи вся группа строк кэшируется в памяти, поэтому, если размер каждой группы строк определяется размером памяти.
Блок столбцов: каждый столбец в группе строк хранится в блоке столбцов, а все столбцы в группе строк постоянно хранятся в этом файле группы строк. Различные блоки столбцов могут использовать разные алгоритмы сжатия.
Страница: каждый блок столбца разделен на несколько страниц, одна страница является наименьшей единицей кодирования, и разные методы кодирования могут использоваться на разных страницах одного и того же блока столбца.
При нормальных обстоятельствах при хранении данных Parquet размер группы строк будет установлен в соответствии с размером блока HDFS, поскольку в общем случае наименьшая единица данных, обрабатываемая каждой задачей Mapper, представляет собой блок, поэтому Каждая группа строк может быть обработана задачей Mapper для увеличения параллелизма выполнения задачи. Информация об индексе метаданных в формате хранения сохраняется в конце, поэтому при чтении строки данных вам нужно найти последнюю информацию об индексе, и, наконец, вы можете прочитать соответствующие данные строки.
orc
Формат файла ORC - это столбчатый формат хранения в экосистеме Hadoop, созданный в начале 2013 года и изначально созданный из Apache Hive для сокращения пространства хранения данных Hadoop и ускорения Hive. Скорость запроса. Как и в Parquet, это не чисто столбчатый формат хранения, а первый разделитель всей таблицы по группе строк и сохранение ее в столбцах внутри каждой группы строк. Файл ORC имеет самоописание, его метаданные сериализуются с использованием буферов протокола, а данные в файле сжимаются настолько, насколько это возможно, чтобы уменьшить потребление памяти. В настоящее время он также поддерживается механизмами запросов, такими как Spark SQL и Presto. Без поддержки паркет по-прежнему используется в качестве основного столбчатого формата хранения. В 2015 году Apache Project Foundation выдвинул проект ORC на топовый проект Apache.
Как и в Parquet, файл ORC также хранится в двоичном режиме, поэтому его нельзя прочитать напрямую, файл ORC также анализируется самостоятельно, он содержит много метаданных, все эти метаданные Изоморфный ProtoBuffer сериализуется. Файловая структура ORC включает в себя следующие концепции:
Файл ORC: обычный двоичный файл, хранящийся в файловой системе. Файл ORC может содержать несколько полос, и каждая полоса содержит несколько записей. Эти записи хранятся независимо в соответствии со столбцами, что соответствует концепции группы строк в Parquet.
Метаданные на уровне файлов: включая информацию описания файла PostScript, метаинформацию файла (включая статистическую информацию всего файла), всю информацию о полосах и информацию о схеме файлов.
Полоса : группа строк образует полосу. Каждый раз, когда файл читается, блок представляет собой группу строк, которая обычно представляет собой размер блока HDFS, в котором хранятся индекс и данные каждого столбца.
метаданные полосы: сохраните положение полосы, статистику каждого столбца в полосе и все типы и позиции потока.

Группа строк : наименьшая единица индекса, полоса содержит несколько групп строк, значение по умолчанию состоит из 10000 значений.
Поток : поток представляет собой часть допустимых данных в файле, включая индекс и данные. Индексный поток хранит положение и статистическую информацию каждой группы строк. Поток данных включает в себя несколько типов данных, и конкретный тип зависит от типа столбца и метода кодирования.



Разница с паркетом заключается в том, что в начале каждой строки данных будет указана информация об индексе данных индекса. По сравнению с паркетом, который помещает информацию об индексе в конце, теоретическая скорость чтения в восходящем направлении немного выше.
Данные одинаковы, преобразованы из CSV в следующие различные форматы, эффективность чтения и записи выглядит следующим образом:

формат	***размер***	***Время чтения и записи***	***отсчет времени работы***
csv	36G	3.6min	59s
parquet	6G	1.1min	5s
orc	5.9G	1.2min	5s
avro	13G	1.3min	25s

Размер: видно, что по сравнению с csv, parque и orc напрямую уменьшают размер в 6 раз. Теоретически он может достичь 6-10-кратной эффективности сжатия. В этот раз использовался метод сжатия gzip по умолчанию для паркета.

19. **Чем отличается Avro от json?**

Avro можно отнести к категории «Платформы сериализации», а JSON - к категории «Языки».
Что такое Авро? Это ориентированная на строки структура удаленного вызова процедур и сериализации данных, разработанная в рамках проекта Apache Hadoop. Он использует JSON для определения типов данных и протоколов и сериализует данные в компактном двоичном формате.

Что такое JSON? Нотация объектов JavaScript - это облегченный формат обмена данными. Людям легко читать и писать. Машины легко анализируют и генерируют. Он основан на подмножестве языка программирования JavaScript.
Redsift, OTTLabs и Mon Style - одни из популярных компаний, использующих JSON, тогда как Avro используется Liferay, LendUp и BetterCloud. JSON пользуется более широким одобрением и упоминается в стеках 32 компаний и 161 стека разработчиков; по сравнению с Avro, который указан в 8 стеках компаний и 8 стеках разработчиков.

20. ***Чем отличается документориетированный формат данных от реляционного?**

Реляционная (или основанная на таблицах) модель базы данных безусловно наиболее часто используется сегодня и представлена как большими коммерческими пакетами, как Oracle, Sybase, Informix, Ingres и Gupta, так и маленькими как DBaseIV, Access, FoxPro, Alpha4 и Paradox. Все они основаны на первой теоретической работе Кодда и др. 1970 года. Из трех моделей, реляционная модель имеет наиболее хорошую математическую базу, включающую реляционную алгебру и реляционное исчисление. Последнее является основой языка запросов SQL и расширения ODBC, которые широко применяются для подключения пользовательского интерфейса в 2-х и 3-х слойных приложениях клиент/сервер.
Объектно-ориентированные базы данных относительно новы и по-прежнему достаточно примитивны в некоторых отношениях. В большинстве из современных систем, таких как Objectivity, Poet и Versant ощущается недостаток удобства как для пользователя, так и для программиста, и сама по себе теория баз данных не имеет такой хорошей математической основы как реляционные или древовидные модели. Это указывает на молодость данной технологии. Серьезная работа ведется по исправлению обоих этих недостатков, и обе системы баз данных (и реляционная, и древовидная) стремятся применить объектно-ориентированные модели, как правило, поверх их собственных неотъемлемых структур.
Все реляционные базы данных используют в качестве модели хранения данных двумерные таблицы. Любая система данных, не имеет значения какой сложности, может быть сведена к набору таблиц (или "отношений" в терминологии СУРБД) с некоторой избыточностью. Избыточность контролируется путем приведения отношений к канонической "нормальной" форме, которая минимизирует ненужную избыточность без уменьшения связей между элементами данных.
Соединение (при котором временное отношение создается путем соединения информации двух отношений, используя общие поля). Выборка (в которой выбирается подмножество записей в отношении, основываясь на определенных значениях или ряде значений в выбранных полях). Другие операции с данными обычно не поддерживаются в базах с реляционной структурой. Добавление произвольных данных, которые, например, не соответствуют ни одному полю в описании данных, запрещено. Добавление поля для произвольных данных потребовало бы перестройки (реструктурирования) базы данных. А этот, зачастую очень длительный, процесс может выполняться только когда базой данных никто не пользуется.
В настоящее время модель реляционной базы данных - лидирующая в компьютерной индустрии и занимает эту позицию уже приблизительно 20 лет. Но эта позиция силы может превратиться в позицию слабости. Несмотря на то, что РБД стали доминирующими в индустрии баз данных за последние 20 лет, многие эксперты почувствовали, что РБД будут вытесняться более современными и гибкими моделями, в первую очередь моделью объектно-ориентированной базы данных.

Объектно-ориентированные БД - относительно новая концепция и, как таковая, не имеет четкого определения. Требования, которым должна отвечать " объектно-ориентированная " БД, целиком и полностью лежат на совести их авторов. Свойства, представляющиеся общими для большинства реализаций, таковы:

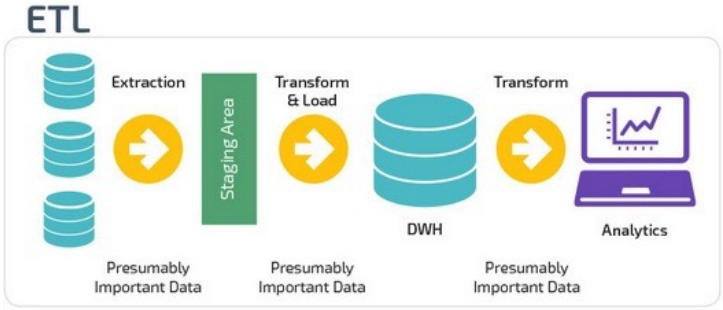
1. *Абстракция: Каждая реальная "вещь", которая хранится в БД, является членом какого-либо класса. Класс определяется как совокупность свойств (properties), методов (methods), общедоступных (public) и частных (private) структур данных, а также программы, применимых к объектам (экземплярам) данного класса. Классы представляют собой ни что иное, как абстрактные типы данных. Методы - это процедуры, которые вызывается для того, чтобы произвести какие-либо действия с объектом (например, напечатать себя или скопировать себя). Свойства - это значения данных, связанные с каждым объектом класса, характеризующие его тем или иным образом (например, цвет, возраст). Свойства присутствуют не во всех реализациях, по сути дела, они являются краткой записью методов без аргументов (таких как "сообщите свой цвет", "сообщите свой возраст").*
2. *Инкапсуляция: Внутреннее представление данных и деталей реализации общедоступных и частных методов (программ) является частью определения класса и известно только внутри этого класса. Доступ к объектам класса разрешен только через свойства и методы этого класса или его родителей (см. ниже "наследование"), а не путем использования знания подробностей внутренней реализации.*
3. *Наследование (одиночное или множественное): Классы определены как часть иерархии классов. Определение каждого класса более низкого уровня наследует свойства и методы его родителя, если они только они явно не объявлены ненаследуемыми или изменены новым определением. При одиночном наследовании класс может иметь только один родительский класс (т.е. классовая иерархия имеет древовидную структуру). При множественном наследовании класс может происходить от многочисленных родителей (т.е. иерархия классов имеет структуру ориентированного нециклического графа, не обязательно древовидную). Не все объектно-ориентированные СУБД поддерживают множественное наследование.*
4. *Полиморфизм: Несколько классов могут иметь совпадающие имена методов и свойств, даже если они считаются различными. Это позволяет писать методы доступа, которые будут правильно работать с объектами совершенно различных классов, лишь бы соответствующие методы и свойства были в этих классах определены.*
5. *Сообщения: Взаимодействие с объектами осуществляется путем посылки сообщений с возможностью получения ответов. Это отличается от традиционного для других моделей вызова процедур. Для того, чтобы применить метод к объекту, надо послать ему сообщение типа "примени к себе данный метод" (например, "напечатай себя"). Парадигма пересылки сообщений не всегда используется в объектно-ориентированных базах данных, однако типична для "истинно" ОО-реализаций.*

Успех объектно-ориентированного подхода лежит в смещении акцента со структуры данных (в особенности, от вида связей между данными) к процессу, с помощью которого эти данные создаются, изменяются и уничтожаются. Реальные структуры данных - деталь реализации, лучше всего относящаяся к внутренней работе каждого класса - и разные классы для обеспечения наивысшей эффективности могут быть реализованы совершенно по-разному. Ведение базы

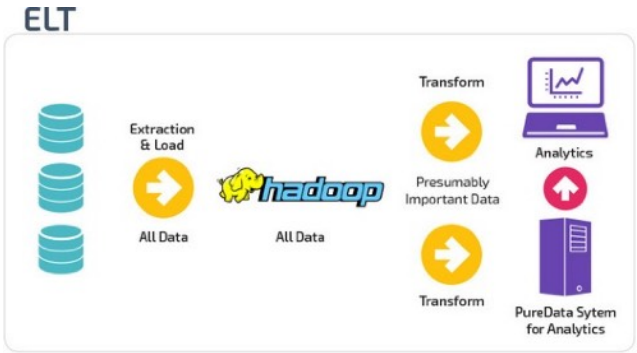
данных выполняется как раз за счет знания методов и свойств, предоставляемых классами.
Модель ООБД находится на более высоком уровне абстракции, чем реляционные или древовидные БД, поэтому классы можно реализовать, опираясь либо на одну из этих моделей, либо на какую-нибудь еще. Поскольку в центре разработки оказываются не структуры данных, а процедуры (методы), важно, чтобы выбиралась базовая модель, которая обеспечивает достаточную прочность, гибкость и производительность обработки.
Реляционные БД с их строгим определением структуры и ограниченным набором разрешенных операций, бесспорно, не подходят в качестве базовой платформы для ООБД.
Объектно-ориентированный подход к системам БД широко разрекламирован как грядущая парадигма, которая заменит реляционную модель через несколько лет. Он действительно дает большую гибкость по сравнению с реляционной моделью и позволяет сосредоточить внимание при разработке и документировании на описании отдельных типов предметов, а не на общей структуре базы данных. Это обещает быть особенно полезным при построении сложных систем БД, которые имеют различные типы данных, особенно, если часто встречаются связи "многие-ко-многим", которые по своей природе не годятся для таблиц.

6. **Чем отличается etl и elt?**

ETL и ELT — два разных способа загрузки данных в хранилище.
ETL (Extract, Transform, Load) сначала извлекают данные из пула источников данных. Данные хранятся во временной промежуточной базе данных. Затем выполняются операции преобразования, чтобы структурировать и преобразовать данные в подходящую форму для целевой системы хранилища данных. Затем структурированные данные загружаются в хранилище и готовы к анализу.



В случае ELT (Extract, Load, Transform) данные сразу же загружаются после извлечения из исходных пулов данных. Промежуточная база данных отсутствует, что означает, что данные немедленно загружаются в единый централизованный репозиторий.
Данные преобразуются в системе хранилища данных для использования с инструментами бизнес-аналитики и аналитики.



Ниже приведены основные ключевые различия между ETL и ELT:

- *ETL является более старой концепцией и существует на рынке более двух десятилетий, ELT - относительно новая концепция и сравнительно сложная для реализации.*
- *В случае ETL, большое количество инструментов имеет только одно из своих требований к оборудованию, которые являются шикарными. В случае ELT, поскольку это подпадает под стоимость оборудования Saas, это не проблема.*
- *Чтобы выполнить поиск, ETL использует шаблон строка за строкой, чтобы отобразить факт-значение с его ключевым элементом измерения из другой таблицы. В ELT мы можем напрямую отобразить факт-значение с ключевыми элементами измерения.*
- *В ETL реляционные данные имеют приоритет здесь, тогда как ELT легко поддерживает неструктурированные данные.*

Сравнительная таблица между ETL и ELT:

Основа сравнения ETL против ELT	ETL	ELT
использование	Подразумевает сложные преобразования включает в себя ETL	ELT вступает в игру, когда задействованы огромные объемы данных
преобразование	Преобразования выполняются в зоне подготовки	Все преобразования в целевых системах
Время	Поскольку этот процесс включает в себя загрузку данных сначала в системы ETL, а затем в соответствующую целевую систему, это тянет за сравнительно большее время.	Здесь, поскольку данные непосредственно загружаются в целевые системы изначально, и все преобразования выполняются в целевых системах.
Участие Datalake	Нет поддержки озера данных	Неструктурированные данные могут быть обработаны с озерами данных здесь.
техническое обслуживание	Обслуживание здесь высоко, так как этот процесс включает в себя два разных этапа	Техническое обслуживание сравнительно низкое
Стоимость	Выше в ценовом факторе	Сравнительно дешевле
вычисления	Либо нам нужно переопределить существующий столбец, либо необходимо отправить данные на целевую платформу.	Рассчитанный столбец можно легко добавить

7. **Далее - не различаем etl и elt:**

Каждая компания, соблюдающая требования к хранилищу данных, будет использовать ETL (Извлечение, Преобразование, Загрузка) или ELT (Извлечение, Загрузка, Преобразование) для передачи данных в хранилище данных, получаемых из разных источников. Исходя из отраслевых и технических потребностей, одна из вышеперечисленных процедур широко применяется.

8. **Какие основные челенджы etl?**

ETL — это процесс, который извлекает данные из различных исходных систем РСУБД, затем преобразует данные (например, применяет вычисления, объединения и т.д.) И загружает данные в систему хранилища данных.

Что означает процесс E, T, L:

- *Извлечение: исходные данные извлекаются из пула данных на этапе извлечения, пул может быть неструктурированным. Далее идет процесс загрузки данных в промежуточное хранилище данных.*
- *Преобразование: это процедура создания или повышения данных, чтобы они стали подходящими для целевого источника.*
- *Загрузка: это маршрут острых данных в хранилище данных, поэтому поверх них можно применять необходимые инструменты бизнес-аналитики.*

ETL: процесс ETL включает извлечение данных из классифицированных источников данных, а затем преобразование и привязывание данных подходящим способом, наконец, данные загружаются в системы хранилища данных. Этот метод целесообразен до тех пор, пока многие разнородные базы данных не будут вовлечены в среду хранилища данных здесь перемещение данных из одного места в другое должно происходить в любом случае, поэтому ETL является наилучшей практикой в таких ситуациях для выполнения преобразований, поскольку передача данных в любом случае происходит здесь

ELT: Это немного другой процесс, здесь используется та же техника извлечения, затем данные загружаются непосредственно в целевые системы. На предыдущем этапе объективные системы отвечают за применение преобразований к загруженным данным. Основным недостатком здесь является то, что обычно требуется больше времени для получения данных в хранилище данных, и, следовательно, с помощью промежуточных таблиц добавляется дополнительный шаг в процессе, который требует больше дискового пространства.

ELT играет свою роль в следующих случаях,

- *Когда основным приоритетом является скорость приема пищи. Поскольку загрузка за пределы сайта здесь не происходит, это считается очень быстрым процессом, поэтому необходимая информация передается здесь намного быстрее, чем ETL. ELT также имеет преимущество, заключающееся в уменьшении диспенсации, происходящей в источнике, ввиду того, что преобразование не выполняется.*
- *Преимущество данных об отключении, которые интересуют бизнес-аналитика, заключается в возможности преобразования невидимых шаблонов в действенную информацию. Соблюдая все биты исторических данных о тендере, организации могут копаться в сроках, сезонных тенденциях, моделях продаж или любых перспективных показателях, которые оказываются важными для организации. Поскольку преобразование данных до их загрузки не выполняется, существует доступ ко всем доступным необработанным данным.*
- *Когда есть необходимость в масштабируемости. Когда в игру вступают топовые механизмы обработки данных, ELT является лучшим вариантом для использования, ELT может добиться улучшения мощности диспетчера для повышения масштабируемости.*

Преимущество ELT заключается в уменьшении выдачи, происходящей в источнике, ввиду того, что преобразование не выполняется, это очень важно учитывать, если источником является система PROD. Основным недостатком здесь является то, что обычно требуется больше времени для получения данных в хранилище данных, и, следовательно, с помощью промежуточных таблиц добавляется дополнительный шаг в процессе, который требует больше дискового пространства.

9. ***Какие инструменты etl вы знаете?**

Список инструментов ETL с их популярными функциями:

1) AWS Glue

AWS Glue — это сервис ETL, который помогает вам подготовить и загрузить их данные для аналитики. Он помогает создавать и запускать различные типы задач ETL в Консоли управления AWS.

Особенности:

- *Автоматическое обнаружение схемы*
- *Этот инструмент ETL автоматически генерирует код для извлечения, преобразования и загрузки ваших данных.*
- *Задания AWS Glue позволяют запускать по расписанию, по запросу или на основе определенного события.*

2) Alooma

Alooma — это продукт ETL, который позволяет команде иметь видимость и контроль. Он предлагает встроенные сети безопасности, которые помогут вам справиться с ошибкой, не останавливая конвейер.

alooma *Особенности:*

- *Обеспечить современный подход к миграции данных*
- *Инфраструктура Alooma соответствует вашим потребностям.*
- *Это поможет вам решить ваши проблемы с конвейером данных.*
- *Создавайте коллажи для анализа транзакционных или пользовательских данных с любым другим источником данных.*
- *Объедините хранилища данных в одном месте, независимо от того, находятся они в облаке или локально.*
- *Легко помогает захватить все взаимодействия.*

3) Stitch

Stitch — это облачная платформа с открытым исходным кодом, которая позволяет быстро перемещать данные. Это простой расширяемый ETL, созданный для групп данных.

Особенности:

- *Он предлагает вам возможность защищать, анализировать и управлять вашими данными, централизуя их в вашей инфраструктуре данных.*
- *Обеспечить прозрачность и контроль вашего конвейера данных*
- *Добавить несколько пользователей в вашей организации*

4) Fivetran

Fivetran — это инструмент ETL, который поддерживает изменения. Он автоматически адаптируется к изменениям схемы и API, что делает доступ к вашим данным простым и надежным способом.

Особенности:

- *Помогает создавать надежные автоматизированные конвейеры со стандартизированными схемами.*
- *Добавление новых источников данных так быстро, как вам нужно*
- *Не требуется обучение или пользовательское кодирование*
- *Поддержка BigQuery, Snowflake, Azure, Redshift и т. Д.*

- *Доступ ко всем вашим данным в SQL*
- *Полная репликация по умолчанию*

5) Matillion

Matillion — это передовое решение ETL, созданное для бизнеса в облаке. Это позволяет вам извлекать, загружать и преобразовывать ваши данные с простотой, скоростью и масштабом.

Особенности:

- *ETL-решения, которые помогут вам эффективно управлять своим бизнесом*
- *Программное обеспечение поможет вам разблокировать скрытые значения ваших данных.*
- *Достигайте результатов своего бизнеса быстрее с помощью решений ETL*
- *Помогает вам подготовить ваши данные для аналитики данных и инструментов визуализации*

6) Streamsets

Программное обеспечение StreamSets ETL, которое позволяет доставлять непрерывные данные в каждую часть вашего бизнеса. Он также обрабатывает смещение данных с помощью современного подхода к проектированию и интеграции данных.

Особенности:

- *Превратите большие данные в идеи всей организации с помощью Apache Spark.*
- *Позволяет выполнять массовую обработку ETL и машинного обучения без необходимости использования языка Scala или Python.*
- *Работайте быстро с единым интерфейсом, который позволяет проектировать, тестировать и развертывать приложения Spark*
- *Он предлагает лучшую видимость исполнения Spark с помощью дрейфа и обработки ошибок.*

7) Talend

Open Studio — это инструмент ETL с открытым исходным кодом, разработанный Talend. Он построен для преобразования, объединения и обновления данных в разных местах. Этот инструмент предоставляет интуитивно понятный набор инструментов, которые облегчают работу с данными. Это также обеспечивает интеграцию больших данных, качество данных и управление основными данными.

Особенности:

- *Поддерживает обширные преобразования интеграции данных и сложные рабочие процессы*
- *Обеспечивает бесперебойную связь для более чем 900 различных баз данных, файлов и приложений.*
- *Он может управлять проектированием, созданием, тестированием, развертыванием и т. Д. Интеграционных процессов.*
- *Синхронизировать метаданные между платформами баз данных*
- *Инструменты управления и мониторинга для развертывания и контроля работ*

8) Информатика PowerCenter

Informatica PowerCenter — это инструмент ETL, разработанный Informatica Corporation. Инструмент предлагает возможность подключения и извлечения данных из разных источников.

Особенности:

- *Он имеет централизованную систему регистрации ошибок, которая облегчает регистрацию ошибок и отклонение данных в реляционные таблицы.*
- *Встроенный интеллект для повышения производительности*
- *Ограничить журнал сеанса*
- *Возможность расширения интеграции данных*
- *Фонд модернизации архитектуры данных*
- *Лучшие проекты с применением передовых методов разработки кода*
- *Интеграция кода с внешними инструментами настройки программного обеспечения*
- *Синхронизация среди географически распределенных членов команды.*

9) Blendo

Blendo синхронизирует готовые аналитические данные в вашем хранилище данных с помощью нескольких щелчков мыши. Этот инструмент поможет вам сэкономить значительное время внедрения. Инструмент предлагает полнофункциональные 14-дневные бесплатные пробные версии.

Особенности:

- *Получите готовые данные аналитики из облачной службы в хранилище данных*
- *Это поможет вам объединить данные из разных источников, таких как продажи, маркетинг или поддержка, и поверхностные ответы, связанные с вашим бизнесом.*
- *Этот инструмент позволяет ускорить исследование, чтобы получить представление о времени с помощью надежных данных, схем и аналитических таблиц.*

10) IRI Voracity

IRI Voracity — это высокопроизводительное, универсальное программное обеспечение ETL для управления данными. Этот инструмент помогает вам контролировать ваши данные на каждом этапе жизненного цикла и извлекать из них максимальную выгоду.

Особенности:

- *IRI Voracity предлагает более быстрые решения для мониторинга и управления данными.*
- *Это поможет вам создавать и управлять тестовыми данными.*
- *Этот инструмент помогает объединить обнаружение данных, интеграцию, миграцию и аналитику в одной платформе*
- *Комбинируйте и оптимизируйте преобразования данных, используя механизмы CoSort или Hadoop.*

11) фабрика Azure Data

Фабрика данных Azure — это гибридный инструмент интеграции данных, который упрощает процесс ETL. Это экономичное и бессерверное решение для интеграции облачных данных.

Особенности:

- *Не требует обслуживания для создания гибридных трубопроводов ETL и ELT*
- *Повышение производительности за счет сокращения времени выхода на рынок*
- *Меры безопасности Azure для подключения к локальным, облачным и программным приложениям*
- *Среда выполнения интеграции служб SSIS помогает переоснастить локальные пакеты служб SSIS.*

12) Logstash

Logstash — это инструмент для сбора данных. Он собирает входные данные и каналы в Elasticsearch. Это позволяет собирать все типы данных из разных источников и делает их доступными для дальнейшего использования.

Особенности:

- *Logstash может объединить данные из разрозненных источников и нормализовать данные в соответствии с желаемым местом назначения.*
- *Это позволяет вам очистить и демократизировать все ваши данные для аналитики и визуализации вариантов использования.*
- *Предлагает централизовать обработку данных*
- *Он анализирует большое разнообразие структурированных / неструктурированных данных и событий*
- *Предлагает плагины для подключения к различным типам источников ввода и платформ*

13) SAS

SAS — это ведущий инструмент ETL, который позволяет получать доступ к данным из нескольких источников. Он может выполнять сложный анализ и предоставлять информацию по всей организации.

Особенности:

- *Деятельность управляется из центральных мест. Следовательно, пользователь может получить доступ к приложениям удаленно через Интернет*
- *Доставка приложений обычно ближе к модели «один ко многим», а не к модели «один к одному».*
- *Централизованное обновление функций позволяет пользователям загружать исправления и обновления.*
- *Позволяет просматривать файлы необработанных данных во внешних базах данных*
- *Помогает вам управлять данными с использованием традиционных инструментов ETL для ввода, форматирования и преобразования данных.*
- *Отображение данных с использованием отчетов и статистической графики*

14) Интеграция данных Pentaho

Pentaho — это платформа для хранения данных и бизнес-аналитики. Инструмент имеет упрощенный и интерактивный подход, который помогает бизнес-пользователям получать доступ, обнаруживать и объединять данные всех типов и размеров.


Особенности:

- *Корпоративная платформа для ускорения конвейера данных*
- *Community Dashboard Editor позволяет быстро и эффективно разрабатывать и развертывать*
- *Это комплексная платформа для решения всех задач интеграции данных.*
- *Интеграция больших данных без необходимости кодирования*
- *Упрощенная встроенная аналитика*
- *Возможность подключения практически к любому источнику данных.*
- *Визуализация данных с помощью пользовательских панелей*
- *Поддержка массовой загрузки для известных облачных хранилищ данных.*
- *Простота использования с возможностью интеграции всех данных*
- *Оперативная отчетность для Монго дБ*
- *Платформа для ускорения конвейера данных*

15) Etleap

Инструмент Etleap помогает организациям получать централизованные и надежные данные для более быстрого и качественного анализа. Этот инструмент помогает вам создавать конвейеры данных ETL.


Особенности:

- *Помогает вам уменьшить инженерные усилия*
- *Создание, поддержка и масштабирование конвейеров ETL без кода.*
- *Предлагает легкую интеграцию для всех ваших источников*
- *Etleap отслеживает конвейеры ETL и помогает решать такие проблемы, как изменения схемы и ограничения исходного API*
- *Автоматизируйте повторяющиеся задачи с помощью оркестровки и планирования конвейера.*

16) Singer

Singer обеспечивает извлечение и консолидацию данных в вашей организации. Инструмент передает данные между базами данных, веб-API, файлами, очередями и т.д.


Особенности:

- *Singer поддерживает JSON Schema, чтобы обеспечить богатые типы данных и жесткую структуру при необходимости.*
- *Он предлагает легко поддерживать состояние между вызовами для поддержки постепенного извлечения.*
- *Извлеките данные из любого источника и запишите их в формате на основе JSON.*

17) Apache Camel

Apache Camel — это инструмент ETL с открытым исходным кодом, который помогает вам быстро интегрировать различные системы, использующие или производящие данные.


Особенности:

- *Помогает вам решать различные типы шаблонов интеграции*
- *Инструмент Camel поддерживает около 50 форматов данных, что позволяет переводить сообщения в различные форматы.*

- *Содержит несколько сотен компонентов, которые используются для доступа к базам данных, очередям сообщений, API и т.д.*

18) Actian DataConnect

Actian DataConnect — это гибридное решение для интеграции данных и ETL. Этот инструмент помогает вам проектировать, развертывать и управлять интеграциями данных на месте или в облаке.

Особенности:

- *Подключайтесь к локальным и облачным источникам с помощью сотен готовых разъемов*
- *Простой в использовании и стандартизированный подход к API веб-сервисов RESTful*
- *Быстрое масштабирование и полная интеграция с помощью многократно используемых шаблонов с помощью среды IDE.*
- *Работайте напрямую с метаданными, используя этот инструмент для опытных пользователей*
- *Это обеспечивает гибкие варианты развертывания*

19) Qlik в реальном времени ETL

Qlik — это инструмент интеграции данных / ETL. Это позволяет создавать визуализации, информационные панели и приложения. Это также позволяет увидеть всю историю, которая живет в данных.

Особенности:

- *Предлагает интерфейсы перетаскивания для создания гибких, интерактивных визуализаций данных*
- *Позволяет использовать естественный поиск для навигации по сложной информации*
- *Мгновенно реагировать на взаимодействия и изменения*
- *Поддерживает несколько источников данных и типов файлов*
- *Обеспечивает безопасность данных и контента на всех устройствах.*
- *Он делится релевантным анализом, который включает приложения и истории с использованием централизованного центра.*

20) IBM Infosphere DataStage

IBM Data Stage — это программное обеспечение ETL, которое поддерживает расширенное управление метаданными и универсальное подключение к бизнесу. Он также предлагает интеграцию данных в реальном времени.

InfoSphere DataStage *Особенности:*

- *Поддержка больших данных и Hadoop*
- *Доступ к дополнительному хранилищу или службам возможен без необходимости установки нового программного и аппаратного обеспечения.*
- *Интеграция данных в реальном времени*
- *Предоставляет надежные и высоконадежные данные ETL*
- *Решать сложные проблемы больших данных*
- *Оптимизация использования оборудования и расстановка приоритетов для критически важных задач*

- *Развертывание локально или в облаке*

21) Oracle Data Integrator

Oracle Data Integrator — это программное обеспечение ETL. Это набор данных, который рассматривается как единое целое. Целью этой базы данных является хранение и получение соответствующей информации. Это помогает серверу управлять огромными объемами данных, чтобы несколько пользователей могли получить доступ к одним и тем же данным.

Особенности:

- *Распределяет данные одинаково по дискам, обеспечивая равномерную производительность*
- *Работает для единичных и реальных кластеров приложений*
- *Предлагает реальное тестирование приложений*
- *Высокоскоростное соединение для перемещения обширных данных*
- *Работает без проблем с платформами UNIX / Linux и Windows*
- *Обеспечивает поддержку виртуализации*
- *Позволяет подключиться к удаленной базе данных, таблице или представлению*

22) Службы интеграции SQL Server

Службы интеграции SQL Server — это инструмент хранилища данных, который используется для выполнения операций ETL. Интеграция с SQL Server также включает в себя богатый набор встроенных задач.

Особенности:

- *Тесно интегрируется с Microsoft Visual Studio и SQL Server*
- *Проще поддерживать и настраивать пакет*
- *Позволяет удалить сеть как узкое место для вставки данных*
- *Данные могут быть загружены параллельно и в разных местах*
- *Он может обрабатывать данные из разных источников данных в одном пакете*
- *SSIS использует трудные данные, такие как службы FTP, HTTP, MSMQ, службы анализа и т. Д.*
- *Данные могут быть загружены параллельно многим различным адресатам*

10. Для чего нужны key-value СУБД?

Ключ СУБД — это атрибут или набор атрибутов, который помогает идентифицировать строку (кортеж) в отношении (таблице). Они позволяют найти связь между двумя таблицами. Клавиши помогают однозначно идентифицировать строку в таблице по комбинации одного или нескольких столбцов в этой таблице.

Вот причины использования ключей в системе СУБД.

- *Ключи помогают идентифицировать любую строку данных в таблице. В реальном приложении таблица может содержать тысячи записей. Более того, записи могут быть продублированы. Ключи гарантируют, что вы сможете однозначно идентифицировать запись таблицы, несмотря на эти проблемы.*
- *Позволяет установить связь между и определить связь между таблицами*
- *Помочь вам обеспечить идентичность и целостность в отношениях.*

СУБД имеет следующие семь типов ключей, каждый из которых имеет свои различные функции:

- *Супер Ключ*

- *Основной ключ (PRIMARY KEY)*
- *Ключ-кандидат (CANDIDATE KEY)*
- *Альтернативный ключ (ALTERNATE KEYS)*
- *Внешний ключ (FOREIGN KEY)*
- *Составной ключ*
- *Композитный ключ*
- *Суррогатный ключ*

Супер ключ — это группа из одного или нескольких ключей, которая идентифицирует строки в таблице. Супер ключ может иметь дополнительные атрибуты, которые не нужны для уникальной идентификации.

PRIMARY KEY — это столбец или группа столбцов в таблице, которые уникально идентифицируют каждую строку в этой таблице. Первичный ключ не может быть дубликатом, то есть одно и то же значение не может появляться в таблице более одного раза. Таблица не может иметь более одного первичного ключа.

Правила определения первичного ключа:

- *Две строки не могут иметь одинаковое значение первичного ключа*
- *Для каждой строки должно быть значение первичного ключа.*
- *Поле первичного ключа не может быть пустым.*
- *Значение в столбце первичного ключа никогда не может быть изменено или обновлено, если какой-либо внешний ключ ссылается на этот первичный ключ.*

ALTERNATE KEYS — это столбец или группа столбцов в таблице, которые однозначно идентифицируют каждую строку в этой таблице. Таблица может иметь несколько вариантов выбора первичного ключа, но в качестве первичного ключа может быть задан только один. Все ключи, которые не являются первичными ключами, называются альтернативными ключами.

CANDIDATE KEY — это набор атрибутов, которые однозначно идентифицируют кортежи в таблице. Ключ-кандидат — это супер-ключ без повторяющихся атрибутов. Первичный ключ должен быть выбран из возможных ключей. В каждой таблице должен быть хотя бы один ключ-кандидат. Таблица может иметь несколько ключей-кандидатов, но только один первичный ключ.

Свойства ключа-кандидата:

- *Он должен содержать уникальные значения*
- *Ключ-кандидат может иметь несколько атрибутов*
- *Не должен содержать нулевые значения*
- *Он должен содержать минимальные поля для обеспечения уникальности*
- *Уникальная идентификация каждой записи в таблице*

FOREIGN KEY — это столбец, который создает связь между двумя таблицами. Назначение внешних ключей заключается в поддержании целостности данных и возможности навигации между двумя различными экземплярами объекта. Он действует как перекрестная ссылка между двумя таблицами, так как ссылается на первичный ключ другой таблицы.

Составной ключ имеет два или более атрибута, которые позволяют однозначно распознавать конкретную запись. Возможно, что каждый столбец не может быть уникальным сам по себе в базе данных. Однако при объединении с другим столбцом или столбцами комбинация составных ключей становится уникальной. Целью составного ключа является уникальная идентификация каждой записи в таблице.

КОМПОЗИЦИОННЫЙ КЛЮЧ — это комбинация двух или более столбцов, которые однозначно идентифицируют строки в таблице. Комбинация столбцов гарантирует уникальность, хотя индивидуальная уникальность не гарантируется. Следовательно, они объединены, чтобы однозначно идентифицировать записи в таблице.

Искусственный ключ, предназначенный для уникальной идентификации каждой записи, называется суррогатным ключом. Такие ключи уникальны, потому что они создаются, когда у вас нет естественного первичного ключа. Они не придают никакого значения данным в таблице. Суррогатный ключ обычно является целым числом.

Суррогатные ключи разрешены, когда

- *Ни одно свойство не имеет параметра первичного ключа.*
- *В таблице, когда первичный ключ слишком большой или сложный.*

Разница между первичным и внешним ключом

Основной ключ	**Внешний ключ**
Помогает вам однозначно идентифицировать запись в таблице.	Это поле в таблице, которое является первичным ключом другой таблицы.
Первичный ключ никогда не принимает нулевые значения.	Внешний ключ может принимать несколько нулевых значений.
Первичный ключ — это кластеризованный индекс, а данные в таблице СУБД физически организованы в последовательности кластерного индекса.	Внешний ключ не может автоматически создавать индекс, кластеризованный или некластеризованный. Однако вы можете вручную создать индекс по внешнему ключу.
Вы можете иметь один первичный ключ в таблице.	Вы можете иметь несколько внешних ключей в таблице.

Резюме

- *Ключ СУБД — это атрибут или набор атрибутов, который помогает вам идентифицировать строку (кортеж) в отношении (таблице)*
- *Ключи СУБД позволяют установить связь и идентифицировать связь между таблицами*
- *Семь типов ключей СУБД: супер, первичный, кандидатный, альтернативный, внешний, составной, композитный и суррогатный ключ.*
- *Супер ключ — это группа из одного или нескольких ключей, которая идентифицирует строки в таблице.*
- *Столбец или группа столбцов в таблице, которая помогает нам однозначно идентифицировать каждую строку в этой таблице, называется первичным ключом.*
- *Все ключи, которые не являются первичными ключами, называются альтернативными ключами.*
- *Супер ключ без повторяющегося атрибута называется ключом-кандидатом*
- *Составной ключ — это ключ, который имеет много полей, которые позволяют однозначно распознать конкретную запись*
- *Ключ, имеющий несколько атрибутов для уникальной идентификации строк в таблице, называется композитным ключом.*
- *Искусственный ключ, предназначенный для уникальной идентификации каждой записи, называется суррогатным ключом.*
- *Первичный ключ никогда не принимает нулевые значения, в то время как внешний ключ может принимать несколько нулевых значений.*

28. *Какие сложности стриминга в hdfs?

Может сложности связаны с задержкой информации или что бывает она пропадает кусками?

29. *Какие минусы key-value хранилищ?

Нужно обязательно знать ключ, чтобы добраться до информации.

30. Из чего состоит хранилище данных?

Хранение данных — одно из важнейших направлений развития компьютеров, возникшее после появления энергонезависимых запоминающих устройств. Системы хранения данных разных масштабов применяются повсеместно: в банках, магазинах, предприятиях. По мере роста требований к хранимым данным растет сложность хранилищ данных.

Надежно хранить данные в больших объемах, а также выдерживать отказы физических носителей — весьма интересная и сложная инженерная задача.

Под хранением обычно понимают запись данных на некоторые накопители данных, с целью их (данных) дальнейшего использования. Опустим исторические варианты организации хранения, рассмотрим подробнее классификацию систем хранения по разным критериям. Я выбрал следующие критерии для классификации: по способу подключения, по типу используемых носителей, по форме хранения данных, по реализации.

По способу подключения есть следующие варианты:

- *Внутреннее. Сюда относятся классическое подключение дисков в компьютерах, накопители данных устанавливаются непосредственно в том же корпусе, где и будут использоваться. Типовые шины для подключения — SATA, SAS, из устаревших — IDE, SCSI.*

подключение дисков в сервере

- *Внешнее. Подразумевается подключение накопителей с использованием некоторой внешней шины, например FC, SAS, IB, либо с использованием высокоскоростных сетевых карт.*

дисковая полка, подключаемая по FC

По типу используемых накопителей возможно выделить:

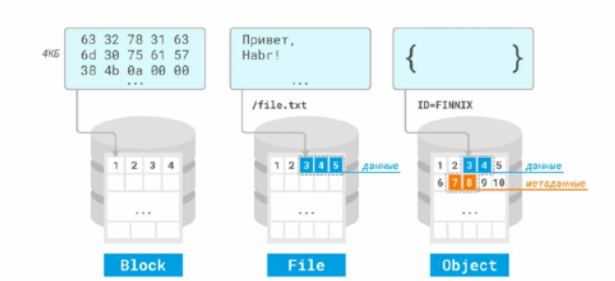
- *Дисковые. Предельно простой и вероятно наиболее распространенный вариант до сих пор, в качестве накопителей используются жесткие диски*
- *Ленточные. В качестве накопителей используются запоминающие устройства с носителем на магнитной ленте. Наиболее частое применение — организация резервного копирования.*
- *Flash. В качестве накопителей применяются твердотельные диски, они же SSD. Наиболее перспективный и быстрый способ организации хранилищ, по емкости SSD уже фактически сравнялись с жесткими дисками (местами и более емкие). Однако по стоимости хранения они все еще дороже.*
- *Гибридные. Совмещающие в одной системе как жесткие диски, так и SSD. Являются промежуточным вариантом, совмещающим достоинства и недостатки дисковых и flash хранилищ.*

Если рассматривать форму хранения данных, то явно выделяются следующие:

- *Файлы (именованные области данных). Наиболее популярный тип хранения данных — структура подразумевает хранение данных, одинаковое для пользователя и для накопителя.*
- *Блоки. Одинаковые по размеру области, при этом структура данных задается пользователем. Характерной особенностью является оптимизация скорости*

доступа за счет отсутствия слоя преобразования блоки-файлы, присутствующего в предыдущем способе.

- *Объекты. Данные хранятся в плоской файловой структуре в виде объектов с*



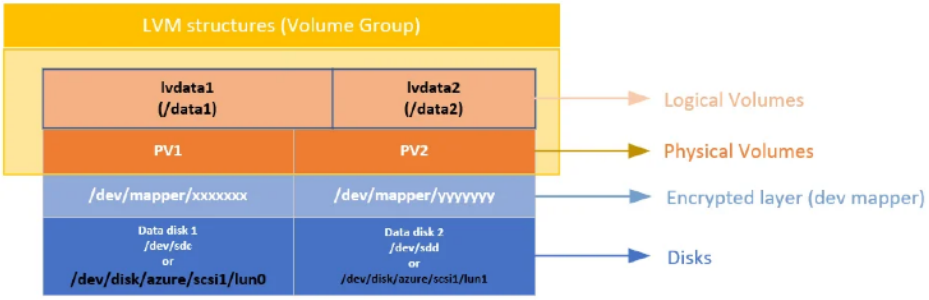
метаданными.

По реализации достаточно сложно провести четкие границы, однако можно отметить:

- *аппаратные, например RAID и HBA контроллеры, специализированные СХД.*


RAID контроллер от компании Fujitsu

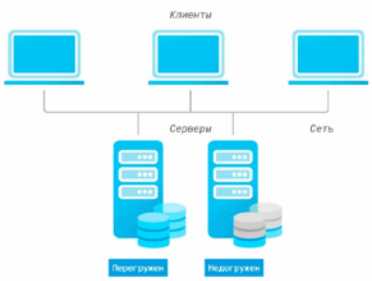
- *Программные. Например реализации RAID, включая файловые системы (например, BtrFS), специализированные сетевые файловые системы (NFS) и протоколы (iSCSI), а также SDS*



пример организации LVM с шифрованием и избыточностью в виртуальной машине Linux в облаке Azure

Детально рассмотрим некоторые технологии, их достоинства и недостатки:

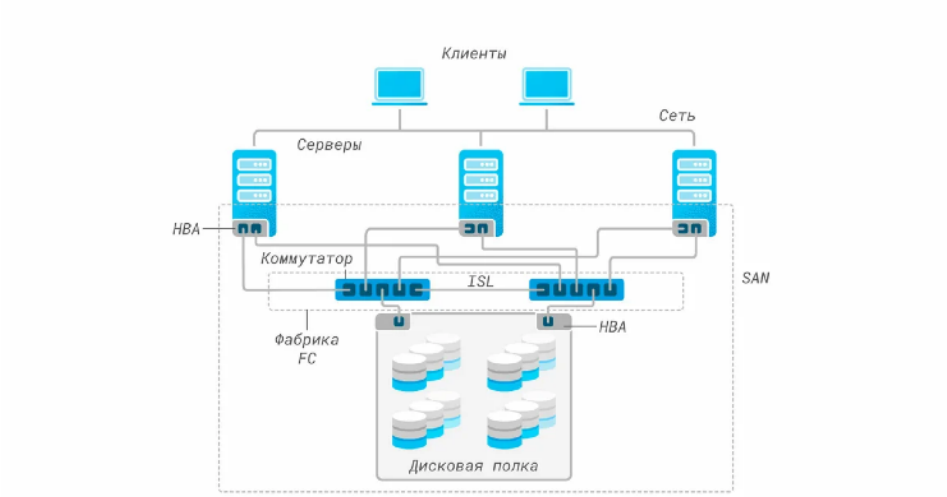
DAS (Direct Attached Storage) — это исторически первый вариант подключения носителей, применяемый до сих пор. Накопитель, с точки зрения компьютера, в котором он установлен, используется монопольно, обращение с накопителем происходит поблочно, обеспечивая максимальную скорость обмена данными с накопителем с минимальными задержками. Также это наиболее дешевый вариант организации системы хранения данных, однако не лишенный своих недостатков. К примеру если нужно организовать хранение данных предприятия на нескольких серверах, то такой способ организации не позволяет совместное использование дисков разных серверов между собой, так что система хранения данных будет не оптимальной: некоторые сервера будут испытывать недостаток дискового пространства, другие же — не будут полностью его утилизировать:



Конфигурации систем с единственным накопителем применяются чаще всего для нетребовательных нагрузок, обычно для домашнего применения. Для

профессиональных целей, а также промышленного применения чаще всего используется несколько накопителей, объединенных в RAID-массив программно, либо с помощью аппаратной карты RAID для достижения отказоустойчивости и\или более высокой скорости работы, чем единичный накопитель. Также есть возможность организации кэширования наиболее часто используемых данных на более быстром, но менее емком твердотельном накопителе для достижения и большой емкости и большой скорости работы дисковой подсистемы компьютера.

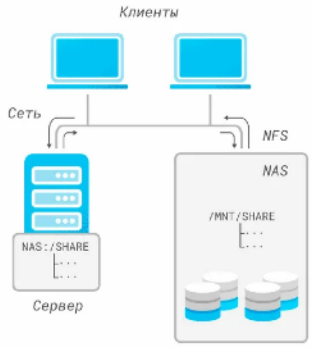
SAN (Storage area network), она же сеть хранения данных, является технологией организации системы хранения данных с использованием выделенной сети, позволяя таким образом подключать диски к серверам с использованием специализированного оборудования. Так решается вопрос с утилизацией дискового пространства серверами, а также устраняются точки отказа, неизбежно присутствующие в системах хранения данных на основе DAS. Сеть хранения данных чаще всего использует технологию Fibre Channel, однако явной привязки к технологии передачи данных — нет. Накопители используются в блочном режиме, для общения с накопителями используются протоколы SCSI и NVMe, инкапсулируемые в кадры FC, либо в стандартные пакеты TCP, например в случае использования SAN на основе iSCSI.



Недостатками такой системы являются большая стоимость и сложность, поскольку для обеспечения отказоустойчивости требуется обеспечить несколько путей доступа (multipath) серверов к дисковым полкам, а значит, как минимум, задублировать фабрики. Также в силу физических ограничений (скорость света в общем и емкость передачи данных в информационной матрице коммутаторов в частности) хоть и существует возможность неограниченного подключения устройств между собой, на практике чаще всего есть ограничения по числу соединений (в том числе и между коммутаторами), числу дисковых полок и тому подобное.

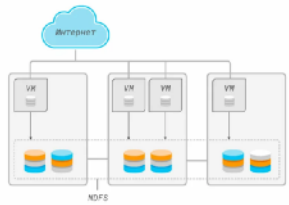
NAS (Network attached storage), или сетевое файловое хранилище, представляет дисковые ресурсы в виде файлов (или объектов) с использованием сетевых протоколов, например NFS, SMB и прочих. Принципиально базируется на DAS, но ключевым отличием является предоставление общего файлового доступа. Так как работа ведется по сети — сама система хранения может быть сколько угодно далеко от потребителей (в разумных пределах разумеется), но это же является и недостатком в случае организации на предприятиях или в датацентрах, поскольку для работы утилизируется полоса пропускания основной сети — что, однако, может быть нивелировано с использованием выделенных сетевых карт для доступа к NAS. Также по сравнению с SAN упрощается работа клиентов, поскольку сервер NAS берет на

себя все вопросы по общему доступу и т.п.



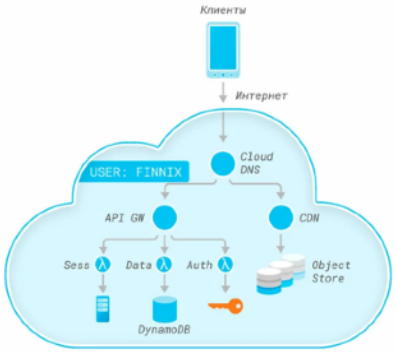
Unified storage (Универсальные системы), позволяющие совмещать в себе как функции NAS так и SAN. Чаще всего по реализации это SAN, в которой есть возможность активировать файловый доступ к дисковому пространству. Для этого устанавливаются дополнительные сетевые карты (или используются уже существующие, если SAN построена на их основе), после чего создается файловая система на некотором блочном устройстве — и уже она раздается по сети клиентам через некоторый файловый протокол, например NFS.
SDS (Software-defined storage) — программно определяемое хранилище данных, основанное на DAS, при котором дисковые подсистемы нескольких серверов логически объединяются между собой в кластер, который дает своим клиентам доступ к общему дисковому пространству.
Наиболее яркими представителями являются GlusterFS и Ceph, но также подобные вещи можно сделать и традиционными средствами (например на основе LVM2, программной реализации iSCSI и NFS).
Из преимуществ SDS — можно построить отказоустойчивую производительную реплицируемую систему хранения данных с использованием обычного, возможно даже устаревшего оборудования. Если убрать зависимость от основной сети, то есть добавить выделенные сетевые карты для работы SDS, то получается решение с преимуществами больших SAN\NAS, но без присущих им недостатков. Я считаю, что за подобными системами — будущее, особенно с учетом того, что быстрая сетевая инфраструктура более универсальная (ее можно использовать и для других целей), а также дешевеет гораздо быстрее, чем специализированное оборудование для построения SAN. Недостатком можно назвать увеличение сложности по сравнению с обычным NAS, а также излишней перегруженностью (нужно больше оборудования) в условиях малых систем хранения данных.
Гиперконвергентные системы
Подавляющее большинство систем хранения данных используется для организации дисков виртуальных машин, при использовании SAN неизбежно происходит удорожание инфраструктуры. Но если объединить дисковые системы серверов с помощью SDS, а процессорные ресурсы и оперативную память с помощью гипервизоров отдавать виртуальным машинам, использующим дисковые ресурсы этой SDS — получится неплохо сэкономить. Такой подход с тесной интеграцией хранилища совместно с другими ресурсами называется гиперконвергентностью. Ключевой особенностью тут является способность почти бесконечного роста при нехватке ресурсов, поскольку если не хватает ресурсов, достаточно добавить еще один сервер с дисками к общей системе, чтобы нарастить ее. На практике обычно есть ограничения, но в целом наращивать получается гораздо проще, чем чистую

SAN. Недостатком является обычно достаточно высокая стоимость подобных решений, но в целом совокупная стоимость владения обычно снижается.



Облака и эфемерные хранилища

Логическим продолжением перехода на виртуализацию является запуск сервисов в облаках. В предельном случае сервисы разбиваются на функции, запускаемые по требованию (бессерверные вычисления, serverless). Важной особенностью тут является отсутствие состояния, то есть сервисы запускаются по требованию и потенциально могут быть запущены столько экземпляров приложения, сколько требуется для текущей нагрузки. Большинство поставщиков (GCP, Azure, Amazon и прочие) облачных решений предлагают также и доступ к хранилищам, включая файловые и блочные, а также объектные. Некоторые предлагают дополнительно облачные базы, так что приложение, рассчитанное на запуск в таком облаке, легко может работать с подобными системами хранения данных. Для того, чтобы все работало, достаточно оплатить вовремя эти услуги, для небольших приложений поставщики вообще предлагают бесплатное использование ресурсов в течение некоторого срока, либо вообще навсегда.

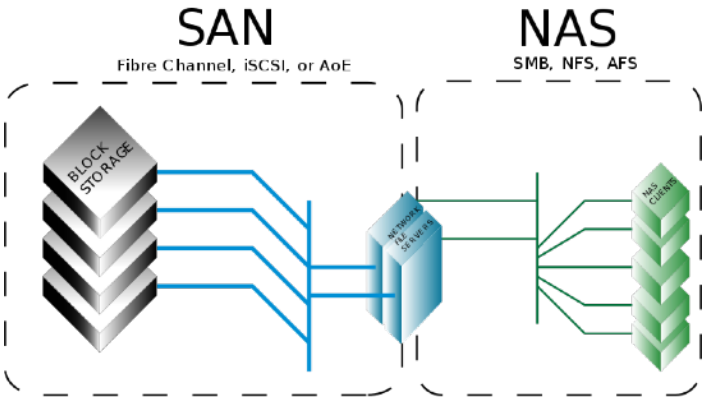


Из недостатков: могут заблокировать аккаунт, на котором все работает, что может привести к простоям в работе. Также могут быть проблемы со связностью и\или доступностью таких сервисов по сети, поскольку такие хранилища полностью зависят от корректной и правильной работы глобальной сети.

31. Какие виды хранилищ данных вы знаете?

Хранение на уровне блоков лежит в основе работы традиционного жесткого диска или магнитной ленты. Файлы разбиваются на «кусочки» одинакового размера, каждый с собственным адресом, но без метаданных. Пример — ситуация, когда драйвер HDD пишет и считывает блоки по адресам на отформатированном диске. Такие СХД используются многими приложениями, например, большинством реляционных СУБД, в списке которых Oracle, DB2 и др. В сетях доступ к блочным хостам организуется за счет SAN с помощью протоколов Fibre Channel, iSCSI или AoE.

Файловая система — это промежуточное звено между блочной системой хранения и вводом-выводом приложений. Наиболее распространенным примером хранилища файлового типа является NAS. Здесь, данные хранятся как файлы и папки, собранные в иерархическую структуру, и доступны через клиентские интерфейсы по имени, названию каталога и др.

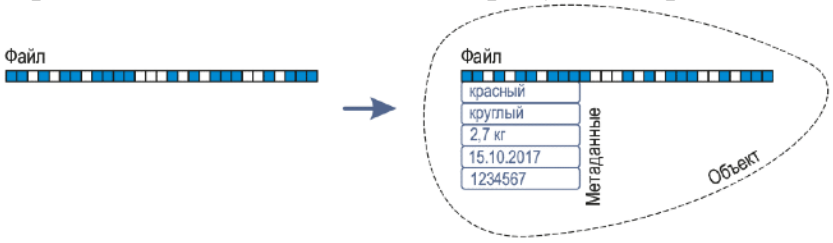


При этом следует отметить, что разделение «SAN — это только сетевые диски, а NAS — сетевая файловая система» искусственно. Когда появился протокол iSCSI, граница между ними начала размываться. Например, в начале нулевых компания NetApp стала предоставлять iSCSI на своих NAS, а EMC — «ставить» NAS-шлюзы на SAN-массивы. Это делалось для повышения удобства использования систем.

Что касается объектных хранилищ, то они отличаются от файловых и блочных отсутствием файловой системы. Древовидную структуру файлового хранилища здесь заменяет плоское адресное пространство. Никакой иерархии — просто объекты с уникальными идентификаторами, позволяющими пользователю или клиенту извлекать данные.

Марк Горос (Mark Goros), генеральный директор и соучредитель Carnigo, сравнивает такой способ организации со службой парковки, предполагающей выдачу автомобиля. Вы просто оставляете свою машину парковщику, который увозит её на стояночное место. Когда вы приходите забирать транспорт, то просто показываете талон — вам возвращают автомобиль. Вы не знаете, на каком парковочном месте он стоял.

Большинство объектных хранилищ позволяют прикреплять метаданные к объектам и агрегировать их в контейнеры. Таким образом, каждый объект в системе состоит из трех элементов: данных, метаданных и уникального идентификатора — присвоенного адреса. При этом объектное хранилище, в отличие от блочного, не ограничивает метаданные атрибутами файлов — здесь их можно настраивать.



Блочные хранилища

Блочные хранилища обладают набором инструментов, которые обеспечивают повышенную производительность: хост-адаптер шины разгружает процессор и освобождает его ресурсы для выполнения других задач. Поэтому блочные системы хранения часто используются для виртуализации. Также хорошо подходят для работы с базами данных.

Недостатками блочного хранилища являются высокая стоимость и сложность в управлении. Еще один минус блочных хранилищ (который относится и к файловым, о которых далее) — ограниченный объем метаданных. Любую дополнительную информацию приходится обрабатывать на уровне приложений и баз данных.

Файловые хранилища

Среди плюсов файловых хранилищ выделяют простоту. Файлу присваивается имя, он получает метаданные, а затем «находит» себе место в каталогах и подкаталогах. Файловые хранилища обычно дешевле по сравнению с блочными системами, а иерархическая топология удобна при обработке небольших объемов данных. Поэтому

с их помощью организуются системы совместного использования файлов и системы локального архивирования.
Пожалуй, основной недостаток файлового хранилища — его «ограниченность». Трудности возникают по мере накопления большого количества данных — находить нужную информацию в куче папок и вложений становится трудно. По этой причине файловые системы не используются в дата-центрах, где важна скорость.

Объектные хранилища
Что касается объектных хранилищ, то они хорошо масштабируются, поэтому способны работать с петабайтами информации. По статистике, объем неструктурированных данных во всем мире достигнет 44 зеттабайт к 2020 году — это в 10 раз больше, чем было в 2013. Объектные хранилища, благодаря своей возможности работать с растущими объемами данных, стали стандартом для большинства из самых популярных сервисов в облаке: от Facebook до DropBox.
Такие хранилища, как Haystack Facebook, ежедневно пополняются 350 млн фотографий и хранят 240 млрд медиафайлов. Общий объем этих данных оценивается в 357 петабайт.
Хранение копий данных — это другая функция, с которой хорошо справляются объектные хранилища. По данным исследований, 70% информации лежит в архиве и редко изменяется. Например, такой информацией могут выступать резервные копии системы, необходимые для аварийного восстановления.
Но недостаточно просто хранить неструктурированные данные, иногда их нужно интерпретировать и организовывать. Файловые системы имеют ограничения в этом плане: управление метаданными, иерархией, резервным копированием — все это становится препятствием. Объектные хранилища оснащены внутренними механизмами для проверки корректности файлов и другими функциями, обеспечивающими доступность данных.
Плоское адресное пространство также выступает преимуществом объектных хранилищ — данные, расположенные на локальном или облачном сервере, извлекаются одинаково просто. Поэтому такие хранилища часто применяются для работы с Big Data и медиа. Например, их используют Netflix и Spotify. Кстати, возможности объектного хранилища сейчас доступны и в сервисе 1cloud.
Благодаря встроенным инструментам защиты данных с помощью объектного хранилища можно создать надежный географически распределенный резервный центр. Его API основан на HTTP, поэтому к нему можно получить доступ, например, через браузер или cURL. После отправки к файлу добавляются необходимые метаданные. Богатая метаинформация объектов позволит оптимизировать процесс хранения и минимизировать затраты на него. Эти достоинства — масштабируемость, расширяемость метаданных, высокая скорость доступа к информации — делают объектные системы хранения оптимальным выбором для облачных приложений.
Однако важно помнить, что для некоторых операций, например, работы с транзакционными рабочими нагрузками, эффективность решения уступает блочным хранилищам. А его интеграция может потребовать изменения логики приложения и рабочих процессов.

32. *Основные задачи Data governance?

Data Governance или управление данными – это совокупность практик, которые помогают администрировать данные компании.
Основные задачи управления данными:
- *Повышение качества*

Низкое качество данных – это проблема не только для сотрудников отделов информационных технологий и аналитики, а бизнеса в целом. Хотя технология обеспечения качества данных внедряется и обслуживается IT-отделом, основную выгоду получают другие подразделения. Например, неверная разметка акционных продаж отделом маркетинга может привести к ошибкам при прогнозировании объёма товара на следующую акцию, поэтому важно внедрить в рабочий процесс контроль за качеством данных. Это позволит сократить время, которое тратится на то, чтобы привести данные в порядок: убрать дубликаты, привести к одному формату, заполнить пропуски, почистить шумы и т.д.

- *Обеспечение целостности и доступности*

Ни одна система не будет эффективно работать, если у неё нет доступа к необходимой информации. Кроме того, нужно обеспечить работоспособность системы, которая предоставляет доступ пользователям к информации. Для этого рекомендуется использовать облачные хранилища и отказоустойчивые базы данных. Данные могут быть потеряны – случайно или намеренно стёрты с носителя, поэтому необходимо делать резервные копии.

- *Контроль*

Внедрение процессов Data Governance позволяет получить полный контроль над данными. Можно контролировать, где и в каком формате они хранятся, обеспечивать версионность, поддерживать актуальность данных, определить правила доступа к данным и т.д.

- *Обеспечение согласованности*

Если разные сотрудники будут работать с одними и теми же данными, но при этом не синхронизировать их между собой, это может привести к неверным результатам. Можно разместить всю информацию в одном общем хранилище, например, в базе данных или облаке. Это позволит сотрудникам работать с актуальной информацией, которая едина для всех.

- *Унификация*

Накопление информации из множества источников приводит к получению данных в разрозненном виде, их нужно приводить к единому виду и формату. Задача усложняется, если речь идёт о тысячах строк. Каждый сотрудник может по-своему записать одно и то же значение. Например, «500 грамм» и «0.5 кг» воспринимаются человеком одинаково, но для машины это совершенно разные значения. И чтобы система понимала их правильно, необходимо привести все данные к одному формату, а это может занять много времени. Продуктивнее заранее определить политику работы с данными, стандарт, которому будут следовать все сотрудники. Таким образом, даже новый сотрудник или аналитик на стороне сможет понимать данные, и работа будет эффективнее.