***3. Что такое Hadoop?***

С точки зрения аналитики Hadoop можно рассматривать как набор программ и процедур с открытым исходным кодом, который можно использовать в качестве «основы» для построения систем Big Data и работы с ними. Это фреймворк для разработки и выполнения распределённых программ, работающих на кластерах из множества узлов. Эта основополагающая технология хранения и обработки больших данных (Big Data) является проектом верхнего уровня фонда Apache Software Foundation.

Hadoop состоит из четырёх основных «модулей», каждый из которых выполняет определённую задачу, необходимую для системы аналитики больших данных.

***4. Что такое HDFS?***

Основной из четырех модулей Hadoop. Это распределённая файловая система, Hadoop Distributed File System – технология хранения файлов на различных серверах данных (узлах, DataNodes), адреса которых находятся на специальном сервере имен (мастере, NameNode)**.**За счет дублирования (репликации) информационных блоков, [HDFS](https://www.bigdataschool.ru/wiki/hdfs) обеспечивает надежное хранение файлов больших размеров, поблочно распределённых между узлами вычислительного кластера**.**

***5. \*Что такое YARN?***

YARN – система планирования заданий и управления кластером (Yet Another Resource Negotiator), которую также называют MapReduce 2.0 (MRv2) – набор системных программ (демонов), обеспечивающих совместное использование, масштабирование и надежность работы распределенных приложений. Фактически, YARN является интерфейсом между аппаратными ресурсами кластера и приложениями, использующих его мощности для вычислений и обработки данных.

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

HDFS обладает рядом отличительных свойств:

* большой размер блока по сравнению с другими файловыми системами (>64MB), поскольку HDFS предназначена для хранения большого количества огромных (>10GB) файлов;
* ориентация на недорогие и, поэтому не самые надежные сервера – отказоустойчивость всего кластера обеспечивается за счет репликации данных;
* зеркалирование и репликация осуществляются на уровне кластера, а не на уровне узлов данных;
* репликация происходит в асинхронном режиме – информация распределяется по нескольким серверам прямо во время загрузки, поэтому выход из строя отдельных узлов данных не повлечет за собой полную пропажу данных;
* HDFS оптимизирована для потоковых считываний файлов, поэтому применять ее для нерегулярных и произвольных считываний нецелесообразно;
* клиенты могут считывать и писать файлы HDFS напрямую через программный интерфейс Java;
* файлы пишутся однократно, что исключает внесение в них любых произвольных изменений;
* принцип WORM (Write-once and read-many, один раз записать – много раз прочитать) полностью освобождает систему от блокировок типа «запись-чтение». Запись в файл в одно время доступен только одному процессу, что исключает конфликты множественной записи.
* HDFS оптимизирована под потоковую передачу данных;
* сжатие данных и рациональное использование дискового пространства позволило снизить нагрузку на каналы передачи данных, которые чаще всего являются узким местом в распределенных средах;
* самодиагностика — каждый узел данных через определенные интервалы времени отправляет диагностические сообщения узлу имен, который записывает логи операций над файлами в специальный журнал;
* все метаданные сервера имен хранятся в оперативной памяти.

В связи с особенностями архитектуры и принципом действия, для HDFS характерны следующие недостатки:

* сервер имен является центральной точкой всего кластера и его отказ повлечет сбой системы целиком;
* отсутствие полноценной репликации Secondary NameNode;
* отсутствие возможности дописывать или оставить открытым для записи файлы в HDFS, за счет чего в классическом дистрибутиве Apache Hadoop невозможно обновлять блоки уже записанных данных;
* отсутствие поддержки реляционных моделей данных;
* отсутствие инструментов для поддержки ссылочной целостности данных, что не гарантирует идентичность реплик. HDFS перекладывает проверку целостности данных на клиентов. При создании файла клиент рассчитывает контрольные суммы каждые 512 байт, которые в последующем сохраняются на сервере имен. При считывании файла клиент обращается к данным и контрольным суммам. В случае их несоответствия происходит обращение к другой реплике.

***7. Что такое блок HDFS?***

При записи файла в HDFS файл разбивается и записывается в блоки, которые находятся на разных машинах. Размер блока по умолчанию 128 МБ. Такая разбивка способствует тому, что при сбое и отключении какой-то из машин файл не будет утерян полностью и другие его части сохранятся в блоках на других машинах.

***8. Для чего используется NameNode?***

Управляющий узел, узел имен или сервер имен (NameNode) – отдельный, единственный в кластере, сервер с программным кодом для управления пространством имен файловой системы, хранящий дерево файлов, а также мета-данные файлов и каталогов. NameNode – обязательный компонент кластера HDFS, 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.

***9. Для чего используется DataNode?***

Узел или сервер данных (DataNode, Node) – один их множества серверов кластера с программным кодом, отвечающим за файловые операции и работу с блоками данных. DataNode – обязательный компонент кластера HDFS, который 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. Стоит отметить, что данные проходят с остальных узлов кластера к клиенту мимо узла NameNode.

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

В п.6 обозначали, что HADOOP оптимизирован для пакетной загрузки и для хранения больших файлов. Это связано с тем, что Namenode является очень дорогой высокопроизводительной системой, поэтому нецелесообразно занимать пространство в Namenode ненужным количеством метаданных, которые генерируются для нескольких небольших файлов. Таким образом, когда в одном файле содержится большой объем данных, имя узла будет занимать меньше места. Следовательно, для получения оптимизированной производительности HDFS поддерживает большие наборы данных вместо нескольких маленьких файлов.

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

Secondary NameNode — вторичный узел имен, отдельный сервер, единственный в кластере, который 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.

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

Файлы пишутся однократно, что исключает внесение в них любых произвольных изменений. Т.е. теоретически это возможно, но слишком ресурсозатратно и нецелесообразно.

***13. \*Почему задачи на YARN нестабильны?***

В архитектуре YARN глобальный ResourceManager работает как master-демон (обычно на выделенной машине), распределяющий ресурсы кластера между конкурирующими приложениями. ResourceManager отслеживает работоспособные узлы и доступные ресурсы в кластере и распределяет их между приложениями, запускаемыми пользователями. Являясь единственным процессом, имеющим эту информацию, ResourceManager принимает решения о распределении с учетом совместного использования, безопасности и множественной аренды (например, согласно приоритету приложения, вместимости очереди, спискам управления доступом, местоположению данных и т.д.). Таким образом скорость выполнения одной и той же задачи может колебаться в зависимости от политики приоритетности, которая заложена на кластере.

***14. Что такое Hive?***

Apache Hive — это SQL-интерфейс доступа к данным, хранящимся в Hadoop (HDFS, HBase) . Hive позволяет выполнять запросы, агрегировать и анализировать данные используя SQL-синтаксис. Hive использует технологию MapReduce, преобразуя SQL-запросы в задания Apache Spark или Hadoop.

На практике Apache Hive используется инженерами данных (Data Engineer) в ETL-процессах для длительных пакетных заданий на больших наборах данных. Hive не поддерживает интерактивное выполнение запросов в режиме реального времени, а потому не может использоваться в OLTP-задачах.

***15. Что хранит HiveMetastore?***

Хранит метаданные для таблиц Hive — схему на чтение (schema-on-read) , расположение, информацию о столбцах в таблице, типы данных, ACL и тд.

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

* ARCHIVE/UNARCHIVE/TRUNCATE/MERGE/CONCATENATE работает только для managed tables
* DROP удаляет managed tables полностью и удаляет только метаданные для external
* ACID/Transactional работает только для managed tables
* Query Results Caching работает только для managed tables
* Для external tables разрешен только RELY constraint

Т.о. external tables используют в основном тогда, когда есть необходимость сохранить исходные файлы и таблицы при удалении рабочей таблицы.

***17. \*Какие форматы умеет читать Hive?***

В настоящее время в Hive поддерживаются 4 формата данных: TEXTFILE, SEQUENCEFILE, ORC и RCFILE.

***18. \*Чем отличается управление ресурсов в Hive и Impala?***

Что общего между Apache Hive и Cloudera Impala:

* прикладное назначение – Hive (Хайв) и Impala (Импала), в первую очередь, ориентированы на аналитическую обработку данных, хранящихся в экосистеме Hadoop (HDFS, HBase)
* Обе системы представляют собой решения с открытым исходным кодом и свободно распространяются по лицензии Apache Software Foundation (ASF). Тем не менее, изначально Импала является продуктом компании Cloudera, тогда как Хайв была разработана корпорацией Facebook в 2010 году, а затем передана фонду ASF
* SQL-подобный синтаксис (Hive SQL) — Impala использует те же метаданные, драйвер ODBC, пользовательский интерфейс и SQL-подобный язык запросов (HiveQL), что и Apache Hive
* Использование YARN в качестве кластерного менеджера ресурсов, который позволяет управлять всеми ресурсами обработки данных в кластере, назначая их запрашивающим приложениям, таким образом, поддерживая баланс выполнения заданий в распределенной среде

В чем разница между Hive и Impala:

* 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 – Хайв написан на Java, тогда как Импала – 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 – c 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 Импала 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 Хайв 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 Хайв характерна проблема «холодного старта», когда при первом запуске приложения запросы выполняются медленно из-за необходимости установки подключения к источнику данных. В Импала 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 – Хайв 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 Хайв отсутствуют инструменты обеспечения кибербезопасности, тогда как Impala поддерживает аутентификацию Kerberos.
* Основные пользователи – с учетом отказоустойчивости и высокой пропускной способности Хайв, 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).

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

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 колоночных. 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.

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

CSV - текстовый формат, предназначенный для представления табличных данных. Строка таблицы соответствует строке текста, которая содержит одно или несколько полей, разделенных запятыми.

Apache Parquet - это бинарный, колоночно-ориентированный (столбцовый) формат хранения данных, изначально созданный для экосистемы Hadoop. Apache Parquet был создан с целью сделать преимущества сжатого и эффективного столбцового представления данных доступными для любого проекта в экосистеме Hadoop. Он позволяет задавать схемы сжатия на уровне столбцов и рассчитан на возможность добавлять новые кодировки по мере их изобретения и реализации.

ORC (Optimized Row Columnar) – это колоночно-ориентированный (столбцовый) формат хранения Big Data в экосистеме Apache Hadoop. Он совместим с большинством сред обработки больших данных в среде Apache Hadoop и похож на другие колоночные форматы файлов: RCFile и Parquet.

|  |  |  |  |
| --- | --- | --- | --- |
|  | ***CSV*** | ***Parquet*** | ***ORC*** |
| ***Колоночный тип*** | - | + | + |
| ***Сжатие*** | + | ++ | +++ |
| ***Разделимость*** | + | + | ++ |
| ***Обновление*** | +- | + | +- |
| ***Читабельность*** | + | - | + |
| ***Скорость обработки и чтения*** | - | + | + |

***21. Чем отличается Avro от json?***

Достоинства схемы данных Avro для использования ее в Big Data системах:

* Прямое отображение в формат JSON, который часто используется для описания событий и объектов в Big Data системах;
* 2. Компактное представление, по сравнению с вышеупомянутым JSON-форматом, для которого характерно многократное повторение некоторых участков кода (имя поля в каждой записи), что может быть критичным в больших объемах.
* 3. Быстрота передачи и обработки данных благодаря компактности формата Авро;
* 4. Поддержка множества языков программирования (C, C++, C#, Go, Haskell, Java, Python, Scala, а также другие скриптовые и ООП-языки).
* 5. Широкие возможности по описанию объектов и событий, включая создание собственных схем данных, а также средства дополнения языка схемы, определяемые в чистом JSON.
* 6. Обеспечение совместимости с предыдущими версиями по мере развития данных с течением времени.

Json — каждая запись записывается вместе со схемой. Это также влияет и на скорость записи в файл: чем больше байтов необходимо записать, тем больше на это требуется времени. Для каждой записи создается json-объект, что тоже не убавляет время;

Avro - при сериализации объекта анализирует схему, чтобы в дальнейшем решить, каким образом обрабатывать то или иное поле. Это дополнительные затраты, ведущие к увеличению общего времени сериализации;

С точки зрения сжатия, то по экономии места Json существенно проигрывает Avro.

***22. \*Чем отличается документориетированный формат данных от реляционного?***

Документориетированный формат данных или база данных NoSQL

В базе данных NoSQL запись обычно хранится как документ JSON. Для каждой записи, или элемента, значения хранятся в качестве атрибутов в едином документе. В такой модели данные оптимизированы для интуитивно понятной разработки и горизонтальной масштабируемости. Документные базы данных позволяют разработчикам хранить и запрашивать данные в БД с помощью той же документной модели, которую они используют в коде приложения. Гибкий, полуструктурированный, иерархический характер документов и документных баз данных позволяет им развиваться в соответствии с потребностями приложений. Документная модель хорошо работает в каталогах, пользовательских профилях и системах управления контентом, где каждый документ уникален и изменяется со временем.

В реляционной базе данных запись часто разделяется на несколько частей (или «нормализуется») и хранится в отдельных таблицах, отношения между которыми определяются ограничениями первичных и внешних ключей. Реляционная модель создана таким образом, чтобы обеспечить целостность ссылочных данных между таблицами в базе данных. Данные нормализованы для снижения избыточности и в целом оптимизированы для хранения.

***23. Чем отличается etl и elt?***

Основное различие между ETL и ELT заключается в том, где происходит обработка:

ETL обработка данных происходит в инструменте ETL (обычно по записи и в памяти).

ELT обработка данных происходит в ядре базы данных.

ETL — это сокращение от Extract, Transform и Load. В этом процессе инструмент ETL извлекает данные из разных исходных систем РСУБД, затем преобразует данные, например, применяет вычисления, конкатенации и т. д., а затем загружает данные в систему хранилища данных. В ETL данные поступают из источника в цель. В ETL механизм преобразования процессов заботится о любых изменениях данных.

ELT — это другой метод рассмотрения инструментального подхода к перемещению данных. Вместо преобразования данных перед их записью ELT позволяет целевой системе выполнить преобразование. Данные сначала копируются в цель, а затем преобразуются на место. ELT обычно используется с базами данных без Sql, такими как кластер Hadoop, устройство обработки данных или облачная установка.

* ETL обозначает извлечение, преобразование и загрузку, в то время как ELT обозначает извлечение, загрузку, преобразование.
* ETL загружает данные сначала на промежуточный сервер, а затем в целевую систему, тогда как ELT загружает данные непосредственно в целевую систему.
* Модель ETL используется для локальных, реляционных и структурированных данных, в то время как ELT используется для масштабируемых облачных структурированных и неструктурированных источников данных.
* ETL в основном используется для небольшого количества данных, тогда как ELT используется для больших объемов данных.
* ETL не обеспечивает поддержку озера данных, в то время как ELT обеспечивает поддержку озера данных.
* ETL легко внедрить, в то время как ELT требует навыков для внедрения и поддержки.

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

???

***25. Какие основные челленджи etl?***

???

* извлечение данных из внешних источников;
* их трансформация и очистка, чтобы они соответствовали потребностям бизнес-модели;
* и загрузка их в хранилище данных.

***26. \*Какие инструменты etl вы знаете?***

Из всего списка инструментов, которые я нашла, я знакома только с ORACLE, слышала про Numetric. Также Informatica обсуждали на уроке

CData

QuerySurge

Xplenty

CloverDX

Oracle

Amazon RedShift

Domo

Teradata

SAP

SAS

IBM data Stage

Informatica

MS SSIS

Talend Open Studio

Ab Initio

Dundas

Sisense

Tableau

MicroStrategy

Pentaho

BigQuery

Numetric

Solver BI360

MarkLogic

***27. Для чего нужны key-value СУБД?***

Кey-value database - система хранения данных, предназначенная для хранения, извлечения и управления ассоциативными массивами, структура данных, более известная сегодня как словарь или хеш-таблица. Словари содержат коллекцию объектов или записей, которые, в свою очередь, содержат множество различных полей, каждое из которых содержит данные. Эти записи хранятся и извлекаются с использованием ключа, который однозначно идентифицирует запись и используется для быстрого поиска данных в базе данных.

Системы «ключ-значение» обрабатывают данные как одну непрозрачную коллекцию, которая может иметь разные поля для каждой записи. Это обеспечивает значительную гибкость и более точно следует современным концепциям, таким как объектно-ориентированное программирование. Поскольку необязательные значения не представлены заполнителями или входными параметрами, как в большинстве РБД, базы данных «ключ-значение» часто используют гораздо меньше памяти для хранения одной и той же базы данных, что может привести к значительному увеличению производительности при определенных рабочих нагрузках. Самой популярной реализацией базы данных «ключ-значение», согласно рейтингу DB-Engines, на данный момент (2019 год) является Redis

***28. \*Какие сложности стриминга в hdfs?***

Процесс стриминга это непрерывный процесс, в результате которого данные и входного топика считываются, затем происходит их обработка и запись в выходной топик. Если в это время на кластере возникнет сбой, то часть необработанных данных может быть утеряна, также возможна утеря записей о проведенных операциях и возникновение дублей при повторном запуске стриминга.

***29. \*Какие минусы key-value хранилищ?***

* отсутствие принципа нормализации, что затрудняет обработку данных, которые хранятся в разных форматах
* явно проигрывают в согласованности данных
* чтобы прочитать значение, нужно знать и ключ, и блок, поскольку на самом деле ключ является хешем (блок + ключ). В разных блоках могут быть идентичные ключи.
* При увеличении количества записей уникальность ключей становится проблемой
* Невозможность транзакций из-за отсутствия принципа ACID

***30. Из чего состоит хранилище данных?***

Довольно часто традиционная архитектура хранилища данных имеет трехуровневую структуру, состоящую из следующих уровней:

* Нижний уровень: этот уровень содержит сервер базы данных, используемый для извлечения данных из множества различных источников, например, из транзакционных баз данных, используемых для интерфейсных приложений.
* Средний уровень: средний уровень содержит сервер OLAP, который преобразует данные в структуру, лучше подходящую для анализа и сложных запросов. Сервер OLAP может работать двумя способами: либо в качестве расширенной системы управления реляционными базами данных, которая отображает операции над многомерными данными в стандартные реляционные операции (Relational OLAP), либо с использованием многомерной модели OLAP, которая непосредственно реализует многомерные данные и операции.
* Верхний уровень: верхний уровень — это уровень клиента. Этот уровень содержит инструменты, используемые для высокоуровневого анализа данных, создания отчетов и анализа данных.

***31. Какие виды хранилищ данных вы знаете?***

Существуют два основных архитектурных направления — нормализованные хранилища данных и хранилища с измерениями.

В нормализованных хранилищах, данные находятся в предметно ориентированных таблицах третьей нормальной формы. Нормализованные хранилища характеризуются как простые в создании и управлении, недостатки нормализованных хранилищ — большое количество таблиц как следствие нормализации, из-за чего для получения какой-либо информации нужно делать выборку из многих таблиц одновременно, что приводит к ухудшению производительности системы. Для решения этой проблемы используются денормализованные таблицы — витрины данных, на основе которых уже выводятся отчетные формы. При громадных объемах данных могут использовать несколько уровней «витрин»/«хранилищ».

Хранилища с измерениями используют схему «звезда» или схему «снежинка». При этом в центре «звезды» находятся данные (таблица фактов), а измерения образуют лучи звезды. Различные таблицы фактов совместно используют таблицы измерений, что значительно облегчает операции объединения данных из нескольких предметных таблиц фактов (пример — факты продаж и поставок товара). Таблицы данных и соответствующие измерения образуют архитектуру «шина». Измерения часто создаются в третьей нормальной форме, в том числе, для протоколирования изменения в измерениях. Основным достоинством хранилищ с измерениями является простота и понятность для разработчиков и пользователей, также, благодаря более эффективному хранению данных и формализованным измерениям, облегчается и ускоряется доступ к данным, особенно при сложных анализах. Основным недостатком является более сложные процедуры подготовки и загрузки данных, а также управление и изменение измерений данных.

* Блочные хранилища
* Файловые хранилища
* Объектные хранилища

***32. \*Основные задачи Data governance?***

* Повышение качества данных. 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. Нужно обеспечить работоспособность системы, которая предоставляет доступ пользователям к информации. 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.
* Унификация. Накопление информации из множества источников приводит к получению данных в разрозненном виде, их нужно приводить к единому виду и формату. 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.