Технологии, основанные на данных сегодня окружают нас ежедневно.

Персонализированная реклама, умные роботы-помощники, смарт-устройства, рекомендации в сети и здравоохранении, все основано на результатах обработки данных в большом их объеме.

Добрый день уважаемые дамы и господа, сегодня мы начнем с вами курс лекций, посвященный основам обработки больших данных в вычислительных системах.

На данном лекционном курсе мы разберемся с основами обработки больших данных в разных проявлениях, начиная от сбора данных и заканчивая их анализом.

Лекционный курс состоит из восьми тем, среди которых

Технологии хранения информации и больших объемов данных.

В данной теме мы постараемся раскрыть истинную природу больших данных и рассмотреть то, где они хранятся и какие специалисты работают с большими данными.

Следующей темой будет «Технологии сбора информации и больших объемов данных».

В данной теме мы охватим как различные виды и типы данных, так и способы их хранения и представления.

Затем разберемся каким образом данные собираются из различных источников и охватим инструменты и принцип работы менеджеров сообщений.

По порядку дальше идет тема «Технологии структурирования больших объемов данных», в которой мы рассмотрим различные модели хранения данных, а также реляционную модель данных подробнее.

В теме «Технологии обработки больших данных: преобразование и агрегация» мы разберем ключевые моменты связанные с теорией шкал данных, принципов использования различных методик по очистке данных и преобразованию данных

В теме, связанной с обогащением больших данных, мы разберемся с операциями слияния данных, а также возможностями хранения такого большого массива информации.

Поговорим на тему облаков, модели услуг в облаках и вычислений в них.

Тема Технологии аналитики больших объединит в себе как технологии анализа так и построения предиктивных моделей по данным на основе машинного обучения

Тема технологий визуализации больших данных будет посвящена популярным методам визуализации данных, аналитики и BI-системам

В теме технологий обработки больших объемов данных мы разберемся с популярными технологиями NoSQL, файловыми хранилищами данных, Data Lake, Hadoop, Spark, Loginom и AirFlow

На протяжении всех лекций мы будем стараться рассматривать базовые принципы и концепции, изучив которые вы получите возможность дальнейшего изучения более сложного и обширного материала.

В рамках курса будут использоваться материалы различных практиков, аналитиков и компаний, практикующих разработку приложений на основе больших данных.

Большое количество информации по данной теме весьма противоречиво, что делает данную дисциплину еще более интересной в глазах учащегося.

Рекомендуется при прохождении курса конспектировать, зарисовывать и главное записывать материал, чтобы провести в голове определенный ассоциативный ряд множества сущностей, которые будут разбираться в данном курсе.

Итак, мы начинаем

Первой темой наших лекций будет тема «Технологии хранения информации и больших объемов данных».

Давайте разберемся что у нас внутри

Первым подразделом будет раздел введения в большие данные, в котором мы разберемся с природой больших данных, какие отличительные особенности выделяют большие данные от данных поменьше, какие противоречия есть в определении «Большие данные».

Предложим первый классический конвейер обработки больших данных с точки зрения работы и действий, производимых над данными, а также посмотрим кто и каким образом с ними работает.

В завершении сделаем краткий обзор трех главных категорий инструментов для работы с данными и кратко коснемся вычислительной инфраструктуры данных.

Далее мы вернемся к азам представления данных в вычислительной системе и вспомним каким образом числовые и символьные данные представляются в вычислительных устройствах.

Затем углубимся в различные виды современной вычислительной инфраструктуры на базовом уровне и разберемся где обрабатывать данные, а где их необходимо визуализировать.

Также увидим и на базовом уровне поймем устройство центров обработки данных.

И наконец, в 4 разделе нашей лекции мы сделаем краткий обзор современной классификации операционных систем для работы с вычислительными устройствами и разберемся в том какую роль выполняют серверные и пользовательские операционные системы

В современном мире интернет-сервисы уже стали обыденностью и их массовое использование послужило образованию нового витка аналитики данных, в котором данные собираются и обрабатываются в реальном времени в больших количествах.

Не только большое количество записей в таблицах и файлах характеризует большие данные, но и разнообразие собираемых значений со всех источников, а именно большое количество разнообразных параметров с разными типами и форматами.

Такое большое скопление информации и называется большими данными и их сбор, хранение и обработка — ключевая задача аналитики данных на сегодняшний день.

**Большие данные** — это **разнообразные данные, которые поступают с постоянно растущей скоростью и объем которых постоянно растет**.

Три основных свойства больших данных — **разнообразие**, **высокая скорость поступления** и **большой объем**

**Разнообразие данных** заключается как в количестве параметров, собираемых в рамках одной записи данных, так и в различных вариациях представления собираемой информации.

К примеру разнообразия параметров вспомним, что чек магазина состоит из таких параметров как: наименование товара, код товара, количество, цена за единицу, кассир, покупатель и метаданные магазина.

Для одного чека таких строчек несколько, и в каждой строчке таких данных измерения хранятся в различных типах и форматах.

Наименование товара – строковый формат, количество - числовой формат, цена – формат чисел с плавающей запятой и т.д.

Также к примеру разнообразия данных можем привести пример, что

В магазинах помимо чеков также хранятся изображения товаров на полках, которые хранятся в виде файлов-картинок, а также видео с камер наблюдения, которые затем магазины используют при оптимизации расстановки товаров в зале, решая задачу повышения спроса на определенные товарные позиции своего ассортимента.

Также большие данные характеризуются высокой скоростью поступления

**Высокая скорость поступления** данных характеризуется одновременным количеством участников в системе учета.

К примеру для большого магазина, с большим количеством точек распространения товара, учетная система старается обрабатывать поступающие данные в реальном времени со всех них, чтобы затем обрабатывать их без задержек.

Своевременность записи данных в систему хранения — актуальная задача области разработки высоконагруженных систем обработки и хранения данных.

**Большой объем** данных характеризуется в необходимости хранить данные, поступающие с большой скоростью на протяжении некоторого обязательного периода времени.

В различных предметных областях этот период времени разнится от нескольких лет до десятилетий.

Однако, все поступающие большие потоки данных, а также исторические данные необходимо обрабатывать и анализировать на современных вычислительных устройствах.

Эта проблема является очень существенной в рамках работы с данными, ввиду чего программисты используют различные методы архивации данных, не использовавшихся более определенного заданного времени.

Таким образом, объединяя все эти три характеристики получим то, что в современном мире и называется …

большими данными.

На картинке отображены основные проблемы, с которыми сталкиваются специалисты по работе с большими данными:

В большом объеме это:

- Здравоохранение

- Расчетные чеки

- Транзакции

- Мобильные датчики

В скорости поступления рассматривают:

- Пакетную обработку данных

- Холодную обработку данных во времени

- Горячую обработку данных в реальном времени

- Обработка потока данных

В разнообразии учтены:

- Структурированные табличные данные

- Полуструктурированные файловые данные

- Неструктурированные файловые данные

Процесс цифровизации сопряжен с интеграцией информации со всем, чем мы занимаемся.

Современным миром управляют данные.

Компания IDC прогнозирует рост глобального объема информации от 33 зеттабайт в 2018 году до 175 зеттабайт в 2025г.

Емкость носителей к тому времени сможет обеспечить только 59% этого объема. В облачной среде к 2025 году будет храниться 49% этой информации.

30% общего объема данных будет формироваться системами реального времени.

К 2025 году число пользователей этих данных достигнет 5 миллиардов. 90 зеттабайт в 2025г будет создаваться устройствами IoT.

На рисунке показан рост глобального объема данных по годам. Данные были получены от 1100 экспертов из 110 стран.

Рассмотрим в каких областях происходит самая большая нагрузка на сектор хранения данных

Умные устройства или Интернет вещей — область информационных технологий, в которой миллионы устройств по всему миру обмениваются информацией между собой, локальным сервером или хабом для подключения устройств, а также головным сервером, который следит за их состоянием, версиями и обновлениями посредством обратной связи с устройств.

Обратная связь реализуется отправкой необходимых сведений с устройства по каналу связи в реальном времени.

Такой поток данных необходимо уметь правильно и быстро обрабатывать.

Среди умных устройств в большом количестве собирающих наши данные в повседневности популярны следующие

Умные фитнес-трекеры, которые собирают данные

* О вашем местоположении и передвижении в реальном времени
* О состоянии вашего здоровья по датчикам пульса, давления, насыщенности крови кислородом с заданной дискретизацией по времени
* О ваших уведомлениях и используемых приложениях

Данные с таких устройств могут в фоновом режиме обрабатываться на вашем смартфоне или в головном офисе изготовителя, который сможет извлечь определенную пользу из этих данных.

Умные колонки и виртуальные голосовые ассистенты, которые собирают не только текстовую оперативную информацию о ваших поисковых запросах, но и биометрию вашего личного голоса, что в дальнейшем может быть использовано в задачах генерации текста и голоса соответственно

Умные помощники по дому собирают данные о вашем распорядке дня:

* Время приема пищи, количество потребляемых продуктов
* График уборки помещения, фото 3D сцен помещения и т.д.

Самостоятельно пользуясь данными устройствами с подключением к сети интернет, возможны сценарии в которых вы не намеренно помогаете бизнесу в дальнейшем предлагать вам определенного рода услуги посредством рекламы

Алгоритмы подбора музыки, видео, изображений, и прочих онлайн развлечений на вкус потребителя.

Автоматические алгоритмы загрузки медиаконтента с устройств в облачное хранилище.

Сжатие данных и формирование нового контента для пользователя на основе имеющихся предпочтений.

Персонализированная реклама на основе потребленного контента в интернете с использованием чтения цифрового следа.

В сельском хозяйстве большие данные могут использоваться для улучшения урожайности, оптимизации процессов выращивания и сбора урожая, прогнозирования погоды и рыночных тенденций, а также для улучшения управления ресурсами, такими как вода и удобрения.

Среди задач для больших данных также имеют место быть:

* Подсчет продукции
* Распознавание растений
* Мониторинг животных
* Автоматизация ферм
* Контроль условий в зоне посадок.
* Обнаружение поврежденной продукции

В медицине большие данные помогают улучшить диагностику и лечение, анализировать эффективность лекарств и процедур, прогнозировать распространение заболеваний и эпидемий, а также улучшать управление медицинскими учреждениями.

* Сегментация изображений по сканированным изображениям – здесь на основе данных снимков МРТ или Рентгена могут решаться задачи локализации источника внутреннего воспаления
* Анализ движения – по видео современными алгоритмами можно достраивать движение опорно-двигательной системы человека и решать задачу локализации проблем.
* Постановка диагноза – по данным симптомов обращения пациента ко врачу алгоритмы на основе данных могут быть обучены ставить предварительные диагнозы на основе исторических данных текстового описания жалоб на физическое состояние. Таким образом можно построить ассоциативный алгоритм классификации болезни на ранней стадии заболевания.

В розничной торговле большие данные используются для анализа покупательского поведения, персонализации предложений и рекламы, оптимизации запасов и ценообразования, а также для улучшения качества обслуживания клиентов.

* Предсказание повторного визита клиента
* Анализ профилей клиентов
* Анализ продуктовой корзины
* Логистика
* Современные тренды в обществе
* Создание новых брендов

Главной задачей обработки больших данных на сегодняшний день является максимизация пользы от накопленных данных о потреблении ресурсов или услуг.

Сбор данных

Накопленные исторические данные и оперативные данные о потреблении услуг обладают информацией о трендах, тенденциях и изменчивости вектора предпочтений пользователей

Обработка данных

Обработка больших данных позволяет получить пользу из исторических данных в сферах бизнеса, здравоохранения, сельского хозяйства, и т.д.

Но задачи сбора, хранения и обработки данных не являются ключевыми в больших данных.

Анализ данных.

Главной задачей обработки больших данных на сегодняшний день является максимизация пользы от накопленных данных о потреблении ресурсов или услуг.

Необходимо производить над данными операции, позволяющие получать из них новые знания о предметной области и её аспектах, меняющихся во времени.

Накопленные исторические данные и оперативные данные о потреблении услуг обладают информацией о трендах, тенденциях и изменчивости вектора предпочтений пользователей сервисов, услуг и приложений.

Анализ больших данных с помощью специфицированных алгоритмов позволяет получить пользу из исторических данных в сферах озвученных ранее.

Сфера аналитики больших данных базируется на 5 основных составляющих, среди которых:

Сфера аналитики больших данных базируется на 5 основных составляющих, среди которых:

* Генерация данных – личная активность, операции, движение транспортных средств, производственные процессы и тому подобное
* Базовая инфраструктура – Оборудование для сбора данных и сеть для передачи данных
* Технологические инструменты – Программное обеспечение, предназначенное для выполнения задач, таких как: анализ данных, искусственный интеллект, машинное обучение
* Цифровые платформы – Цифровые платформы, обеспечивающие поток данных, их хранение и вычисления над ними
* Вертикальные решения и услуги – Решения для конечного пользователя, приложения и услуги, направленные на решение конкретных проблем

Все эти составляющие, объединенные в единое целое позволяют работать с данными, для решения конечной поставленной цели.

Классификация специалистов, задействованных в работе с данными, на сегодняшний день, всё ещё размыта.

В общемировой практике по мере увеличения капитализации компаний, производящих продукты на основе больших данных распределение обязанностей по работе с данными между людьми достаточно сильно разбито на части.

На каждой стадии работы с данными найдется сотрудник или команда, которая отвечает за данную зону работы проекта.

Результатом становится строгая структуризация и качественное развитие новых стартап проектов, а также стабильное масштабирование существующих проектов.

На рисунке можно увидеть классические вопросы которые встают перед специалистами в области работы с данными:

* Загрузка сырых данных
* Извлечение доступных данных для бизнеса
* Приведение данных в консолидированный вид
* Настройка систем долгосрочного хранения данных

5. Разработка платформы данных для программистов прикладного ПО

6. Аналитика полученных бизнес-данных

Кто же работает на данных стадиях в основном у нас и за рубежом?

В работе с данными выделяют основных специалистов:

Совершенно точно на сегодняшний день в очень четком разделении существуют несколько специализаций в науке о данных.

* Инженер данных – обеспечивает данными команду аналитики данных
* Аналитик данных – вырабатывает бизнес-стратегию и рекомендации по бизнес-процессам организации на основе данных
* ML-инженер – решает задачи настройки и разработки моделей машинного обучения на основе данных
* DEVops, MLops - занимаются разработкой и поддержкой инфраструктуры и цепочки сотрудников для работы с данными и моделями машинного обучения, включая автоматизацию процессов и управление жизненным циклом моделей и распределением ресурсов внутри компании.
* Управляющий продуктом – роль ответственная за определение требований к продукту, управление бэклогом (списком задач) и обеспечение максимальной ценности продукта для заказчика и пользователей.

Управляющий продуктом обычно работает в тесном взаимодействии с командой разработки, заказчиком и другими заинтересованными сторонами, чтобы обеспечить успешное достижение целей продукта.

Посмотрим и разберем чем занимается в проекте инженер данных.

Инженер данных занимается предобработкой, обеспечением хранения и предоставлением данных для различных ситуаций.

Одно из направлений работы инженера данных - это предоставление данных для аналитики.

Это включает в себя создание и поддержку хранилищ данных, где информация собирается, хранится и обрабатывается для последующего анализа.

Также инженер данных занимается разработкой системы аналитики, которая позволяет извлекать ценные инсайты из данных и визуализировать их для принятия бизнес-решений.

Еще одно важное направление работы инженера данных - предоставление данных для машинного обучения и науки о данных.

Это включает в себя подготовку и очистку данных, создание платформы для обучения моделей машинного обучения, а также предоставление доступа к данным для специалистов по науке о данных.

Инженер данных также отвечает за создание инфраструктуры, которая позволяет эффективно использовать данные для обучения моделей и проведения анализа.

Инженер данных имеет ряд ключевых задач, связанных с обработкой и управлением данными. Вот подробное описание каждой из них:

1. Сбор данных из различных источников: Инженер данных отвечает за сбор данных из различных источников, таких как базы данных, веб-серверы, API и т.д.

Перемещение данных: Инженер данных занимается перемещением данных из одного места в другое с помощью потоков данных (например, Kafka) и конвейеров ETL (извлечение, трансформация, загрузка).

Инженер данных обеспечивает очистку, подготовку и трансформацию данных в соответствии с бизнес-правилами и требованиями.

Инженер данных выполняет агрегацию данных для создания сводных отчетов и размечает данные для дальнейшего анализа и построения моделей машинного обучения

Инженер данных занимается оптимизацией хранения и обработки данных, выбирая подходящие базы данных, хранилища данных и инструменты для обработки больших объемов информации.

Инженер данных участвует в построении платформ для обработки, хранения и анализа данных, включая создание инфраструктуры для работы с большими объемами информации.

Не все задачи сваливаются на одного инженера данных, но в данной выборке задач мы говорим лишь о спектре проблем, с которыми сталкивается инженер данных.

В команде по работе с данными возможны сценарии разграничения зоны ответственности.

Далее поговорим о навыках, которые необходимы для работы инженера данных.

На слайде представлены компетенции инженера данных в процентном соотношении от общего числа инженеров, работающих в компаниях.

То есть значение доли в процентах указывает на то какая доля от общего числа инженеров данных использует тот или иной инструмент или компетенции в своей работе.

Компетенции инженера данных, представленные в списке, имеют прямое отношение к задачам обработки данных. Вот как они могут быть использованы:  
1. SQL (Structured Query Language): Необходим для работы с базами данных. Инженеры данных используют SQL для извлечения, обновления и анализа данных, а также для создания отчетов и агрегирования информации.

2. Python: Язык программирования Python широко используется в инженерии данных для создания скриптов, автоматизации процессов обработки данных, разработки моделей машинного обучения и других задач.

3. Data Pipelines: Инженеры данных создают и поддерживают потоки данных (data pipelines), которые позволяют перемещать, трансформировать и обрабатывать данные из различных источников.

4. Data Warehouse: Знание о структуре и функционировании хранилищ данных (data warehouse) позволяет инженерам данных оптимизировать процессы загрузки, хранения и извлечения данных.

5. Hadoop: Hadoop - это фреймворк для распределенного хранения и обработки больших объемов данных. Инженеры данных могут использовать Hadoop для работы с большими массивами данных, выполнения распределенных вычислений и реализации сложных аналитических задач.

6. Hive: Hive - это инфраструктура для обработки и анализа больших наборов структурированных данных в Hadoop. Знание Hive помогает инженерам данных создавать запросы и выполнять аналитику на данных, хранящихся в Hadoop.

7. ETL (Extract, Transform, Load): ETL процессы являются основой для обработки данных. Инженеры данных используют ETL для извлечения данных из различных источников, их трансформации в нужный формат и загрузки в целевые системы.

8. Spark: Apache Spark - это фреймворк для обработки больших объемов данных параллельно и распределенно. Инженеры данных используют Spark для выполнения высокопроизводительной обработки данных, создания потоковой обработки (stream processing), анализа больших данных и машинного обучения.

9. AWS (Amazon Web Services): AWS предоставляет различные инструменты и сервисы для обработки данных, такие как Amazon S3 для хранения данных, Amazon Redshift для аналитики данных, Amazon EMR для обработки больших данных и многие другие. Инженеры данных используют AWS для создания и развертывания инфраструктуры обработки данных в облаке.

10. Redshift: Amazon Redshift - это облачное хранилище данных, предназначенное для аналитики больших объемов данных. Инженеры данных используют Redshift для хранения и анализа структурированных данных, выполнения сложных запросов и создания отчетов.

11. Java: Язык программирования Java широко используется в инженерии данных для разработки приложений обработки данных, создания инструментов для работы с базами данных, реализации алгоритмов обработки данных и многих других задач.

12. Kafka: Apache Kafka - это распределенная система потоковой обработки и передачи сообщений. Инженеры данных используют Kafka для создания потоков данных, передачи и обработки сообщений в реальном времени и интеграции различных источников данных.

13. MapReduce: MapReduce - это модель программирования для параллельной обработки больших объемов данных на кластерах серверов. Инженеры данных используют MapReduce для реализации распределенных вычислений, выполнения агрегации данных и выполнения сложных операций над данными.

14. Scala: Scala - это язык программирования, который часто используется в инженерии данных для разработки высокопроизводительных приложений обработки данных, создания инструментов для работы с распределенными системами и реализации алгоритмов машинного обучения.

15. Vertica: Vertica - это аналитическая база данных, специально оптимизированная для обработки больших объемов структурированных данных. Инженеры данных используют Vertica для выполнения сложных аналитических запросов, агрегации данных и создания отчетов.

16. NoSQL: NoSQL базы данных предоставляют гибкую модель хранения и обработки неструктурированных или полуструктурированных данных. Инженеры данных используют NoSQL для работы с разнообразными типами данных, такими как текст, изображения, видео и другие.

17. Statistics: Статистика играет важную роль в обработке данных, поскольку позволяет инженерам данным проводить анализ, извлекать информацию из данных, строить прогнозы и принимать научные решения на основе данных.

В целом, эти компетенции позволяют инженерам данным эффективно управлять различными типами данных, выполнять сложные операции обработки и анализа данных, создавать высокопроизводительные приложения и системы для работы с данными.

**Разберемся с направлениями работы аналитика данных.**

**Направления работы аналитика данных**

Формулировка бизнес-метрик для построения продуктовых решений на основе данных

Построение отчетов для построенных рекомендаций на основе данных

**Задачи аналитика данных**

Построение метрик качества принятия решений на основе бизнес-требований

Построение аналитических отчетов на основе данных с использованием агрегации разной глубины

Построение моделей предиктивной аналитики на основе бизнес-данных

Формулировка рекомендаций по данным

Направления работы ML-инженера:

Построение моделей машинного обучения по данным

Оптимизация моделей под бизнес-задачу

Обеспечение программного интерфейса взаимодействия с моделями

Задачи ML-инженера:

На основе данных и инструментов строить модели машинного обучения для решения конечных задач

Тестировать модели обучения на данных из тестовой выборки и данных смежной области

Интегрировать полученные решения в тестовый дизайн продукта посредством программного интерфейса

В базовом варианте все вышеперечисленные специалисты трудятся над тем, чтобы данные, собираемые с различных приборов и устройств приобрели вид, необходимый для решения конечной бизнес-задачи.

В дальнейшем данные задачи решаются инженерами машинного обучения и аналитики данных.

Поговорим о базовой модели жизненного цикла работы с данными.

Какие стадии проходят сырые данных, чтобы оказаться готовыми к применению в готовых решениях на базе интеллектуальных алгоритмов.

Перед извлечением из данных пользы их необходимо **собрать, очистить, сохранить в нужном виде** и затем работать с ними

В современной индустрии устоялся классический конвейер работы с данными, как основной шаблон по которому компании подстраивают поток работ под себя

Данный конвейер работы с данными получился в результате принятия лучших практик методологии CRISP-DM – методологии обработки данных в области Data Science. О данной методологии будем подробно говорить в одной из следующих лекций.

Под каждую новую задачу поток работ над данными видоизменяется при неизменной основе

Видоизменения набора задач по обработке данных зависит от **количества источников данных, сложности данных и целей обработки данных**

Большие компании хранят, обрабатывают и анализируют данные на серверных вычислительных устройствах или ЦОД (центры обработки данных) разной степени доступности.

**Вычислительная инфраструктура, на которой компании решают задачи вычислений следующая:**

* Локальный вычислительный кластер
* Частные облачные сервисы
* Общедоступное облако

**Популярные серверные ОС:**

Linux-серверные системы

Debian/CentOS

Инструменты для хранения данных

(ИМПРОВИЗАЦИЯ)

PostgreSQL или любые другие OLTP реляционные базы данных для решения задач хранения и обработки табличных транзакционных данных

Amazon Redshift или любые другие OLAP реляционные базы данных для решения задач быстрого извлечения ужа построенных аналитических агрегатов или плоских таблиц

GreenPlum это распределенная база данных с открытым исходным кодом, разработанная для анализа и обработки больших объемов данных. Его основные задачи включают: Аналитика структурированных данных, Масштабируемость, Поддержка SQL

Hadoop фреймворк для обработки и хранения больших объемов неструктурированных данных в распределенной среде. Его основные задачи включают: Хранение и обработка больших данных, Распределенная обработка данных, Отказоустойчивость

MongoDB - документо-ориентированная база данных, которая предназначена для хранения и обработки полуструктурированных и неструктурированных данных. Его основные задачи включают: Хранение и обработка неструктурированных данных, Гибкая модель данных

В целом, каждый из этих инструментов предназначен для работы с различными типами данных и выполняет задачи, связанные с хранением и обработкой больших объемов информации.

Инструменты управления потоками данных

(ИМПРОВИЗАЦИЯ)

Apache Kafka: Apache Kafka - это распределенная платформа для обработки и передачи потоковых данных. Его основные задачи включают: Сбор и передача потоков данных, Масштабируемость, Управление потоками данных

Apache NiFi - это инструмент для автоматизации процессов сбора, агрегации и доставки данных из различных источников. Его основные задачи включают: Сбор данных, Трансформация данных, Мониторинг и управление данными.

Apache Airflow - это инструмент для планирования, мониторинга и выполнения рабочих процессов. Его основные задачи включают: Планирование задач, Мониторинг выполнения задач, Управление расписанием

В целом, все эти инструменты предназначены для обработки, передачи, управления и мониторинга данных в больших и сложных системах.

Инструменты для обработки и анализа данных:

Python –

R –

Loginom –

Hadoop MapReduce –

Apache Spark –

О некоторых из вышеперечисленных инструментах мы будем говорить подробно в дальнейшем.

Информация и данные в вычислительных устройствах

**Информация** - нематериальная сущность, при помощи которой с любой точностью можно описывать реальные (материальные), виртуальные (возможные) и понятийные (абстрактные) сущности

Информация - это данные, факты или знания, которые передаются и используются для обучения, принятия решений или передачи сообщений.

Это может включать в себя текст, изображения, звуки, видео или любые другие формы представления знаний.

Информация имеет ценность, если она помогает людям понимать мир вокруг себя и взаимодействовать с ним.

Описываемому объекту(или понятию) ставится в соответствие некоторое число (код).

Информация может быть двух видов: *дискретная* информация и *непрерывная (аналоговая).*

Дискретная информация характеризуется последовательными точными значениями некоторой величины, а непрерывная - непрерывным процессом изменения некоторой величины.

При переводе непрерывной информации в дискретную важна *частота дискретизации* ν, определяющая период (Т=1/ν). Значимость баланса ν.

В современных пользовательских и серверных вычислительных устройствах общего назначения информация, хранимая на носителях и данные использующиеся в памяти представлены в виде набора дискретных состояний - битов

**Бит –** единица измерения количества информации, использующаяся в компьютерных системах (сигнал)

Физически Бит представляет собой перепад напряжения.

положительная фиксация напряжения на временной шкале отвечает единице, а перепад в отрицательную сторону отвечает нулю.

**Байт –** набор из 8-ми битов, представляющих собой удобный вид представления информации в вычислительных устройствах

**Машинное слово –** последовательность битов, рассматриваемых аппаратной частью вычислительного устройства как единое целое.

Машинными словами процессор кодирует выполняемые инструкции и поля данных в памяти над которыми необходимо совершить эти инструкции.

Разрядность процессора напрямую говорит о ширине машинного слова, которым оперирует вычислительное устройство.

В качестве примера приведем таблицу с информацией о возможных диапазонах принимаемых значений для различных вариаций целых чисел в компьютерной арифметике

**Пример**. Примитивные типы данных в языке программирования С++

В данной таблице явно указаны различные варианты представления целых чисел в памяти компьютера в соответствии со стандартами языка программирования C++ .

В инженерной практике данная классификация стала уже классической, в результате чего все современные системы обработки данных базируются на данной типизации

**Кодовые таблицы символов призваны для того, чтобы возможно было в формате битов декодировать написанные текстовые данные в файле в читаемом виде, среди кодировок как ранее встречавшихся, так и современных выделяем:**

**ASCII7** — первая кодировка, пригодная для работы с текстом. Помимо маленьких букв английского алфавита и служебных символов, содержит большие буквы английского языка, цифры, знаки препинания и другие символы. **(7 бит)**

**ASCII** — первая кодировка, в которой стало возможно использовать символы национальных алфавитов. **(8 бит)**

**КОИ8-R** — первая русская кодировка. Символы кириллицы расположены не в алфавитном порядке. **(8 бит)**

**CP866** — русская кодировка, использовавшаяся на компьютерах IBM в системе DOS. **(8 бит). До сих пор популярна и распространена в системах верстки российских научных журналов и статей.**

**Windows-1251** — русская кодировка, использовавшаяся в русскоязычных версиях операционной системы Windows в начале 90-х годов. Кириллические символы идут в алфавитном порядке. **(8 бит)**

**UTF8** — распространённый стандарт кодирования символов, позволяющий более компактно хранить и передавать символы Юникода, используя переменное количество байт (от 1 до 4), и обеспечивающий полную обратную совместимость с 7-битной кодировкой ASCII. **(8 бит)**

В различных реализациях стандарта UTF возможно кодировать различное количество символов ввиду ограниченности мощности множества, позволяющего себе биекцию на пространство целых чисел в двоичной системе счисления, среди стандартов реализации UTF можно выделить следующие особенности:

1. Реализация 8 бит - ASCII, в том числе английский алфавит, простейшие знаки препинания и арабские цифры

2. Реализация 16 бит - кириллица, расширенная латиница, арабский алфавит, армянский алфавит, греческий алфавит, еврейский алфавит и коптский алфавит; сирийское письмо; Международный фонетический алфавит; некоторые знаки препинания

3. Реализация 24 бита – все другие современные формы письменности, в том числе грузинский алфавит, индийское, китайское, корейское и японское письмо; сложные знаки препинания; математические и другие специальные символы

4. Реализация 32 бита – музыкальные символы, редкие китайские иероглифы, вымершие формы письменности

5. служебные символы c, d, e, f

Флаги 1 и 0 – это зарезервированные значения в разрядах кодов для однозначного декодирования последовательности.

Иксами указаны свободные места, которые можно заполнить битами. Варианты комбинаций можно сосчитать.

Таким образом с помощью кодировки мы имеем возможность явно декодировать машинный битовый код в виде удобочитаемых символов.

С помощью различных кодировок мы имеем возможность передавать осмысленную текстовую информацию в вычислительных системах.

Задача для вас, ибо данное знание является довольно существенным

Необходимо решить следующую задачу. И законспектировать результат:

**Задача:** подсчитать максимально возможное количество символов для первых трех вариантов реализации в стандарте UTF

Ответ необходимо дать в виде целых чисел.

Отлично, теперь мы переходим к аспектам вычислительных устройств в повседневном использовании и в обработке больших объемов данных.

Итак, раздел вычислительная инфраструктура и вычислительные устройства. В данном разделе мы по верхам зацепим важные особенности вычислительной инфраструктуры и уделим внимание критически важным моментам в ней для обработки данных.

Разберемся с основными характеристиками вычислительных устройств которые являются критическими в области работы с данными.

В этом разделе мы будем говорить о компьютерах, серверах и начинке, на основе которой данные вычисления производятся.

Основные характеристики вычислительного устройства:

* Вычислительная мощность (процессор)
* Оперативная память (ОЗУ)
* Хранилище (дисковое пространство)

Дисковый накопитель отвечает за долговременное хранение информации пользователя.

Это файлы, медиа и данные, которые должны храниться при отсутствии питания от сети.

Дисковый накопитель хранит данные для запуска операционной системы компьютера и данные прикладных программ для работы с ними.

**Жесткий диск (или HDD)** — устройство хранения данных, принцип записи информации в котором заключается в намагничивании областей на поверхности магнитных дисков (пластин).

Для организации хранения данных магнитный диск разбивается на дорожки и сектора, а совокупность дорожек, расположенных одна над другой (на нескольких пластинах), называется цилиндром.

В зависимости от объема памяти, внутри корпуса HDD могут находиться до восьми пластин.

Пластины крепятся к шпинделю, вращающемуся со скоростью от 4 до 15 тысяч оборотов в минуту (rpm).

Запись и чтение информации с пластины осуществляется при помощи магнитной головки.

Связывается с вычислительным устройством посредством SATA кабеля и порта

**Твердотельный накопитель (или SSD**) — устройство, использующее для хранения информации флеш-память.

**Флеш-память (или flash memory)** — разновидность твердотельной полупроводниковой энергонезависимой перезаписываемой памяти.

Она может быть прочитана сколько угодно раз (в пределах срока хранения данных, типично — 10-100 лет), но писать в такую память можно лишь ограниченное число раз (максимально — около миллиона циклов).

SSD 2.5 так же как и жесткий диск связывается с вычислительным устройством посредством SATA кабеля, что является узким горлом в чтении и записи данных с помощью технологии Флэш-памяти

SSD nvme позволяет читать и записывать данные на кратно высокой скорости по сравнению с жесткими дисками и твердотельными 2.5 накопителями за счет другого разъема, использующегося для передачи данных из вычислительной системы на диск.

В зависимости от версии PCI-e порта скорость записи и чтения могут значительно отличаться

Дисковый накопитель отвечает за долговременное хранение информации пользователя. Это файлы, медиа и данные, которые должны храниться при отсутствии питания от сети.

Дисковый накопитель хранит данные для запуска операционной системы компьютера и данные прикладных программ для работы с ними.

* **От вида накопителя будут зависеть:**

1. Долговечность (надежность), количество циклов перезаписи данных. Может зависеть от многих факторов, но зачастую определяется ценой, поскольку более дорогие накопители изготовлены из более качественных материалов с большей надежностью.

2. Скорость работы (чтение и запись),

**Скорость чтения** измеряет, насколько быстро накопитель может «читать» или получать доступ к файлам, хранящимся на нем. Например, SSD с более высокой скоростью чтения может запустить гигабайтный файл быстрее. Это помогает сократить время загрузки компьютера, так как чтение больших файлов, необходимых для загрузки операционной системы, займет меньше времени.

**Скорость записи** измеряет, насколько быстро файл может быть записан на диск. Чаще всего пользователь сталкивается со «скоростью записи», когда пытается скопировать файл из одного места в другое. Чем выше скорость чтения, тем меньше времени потребуется для копирования.

3. Ёмкость (общий размер данных),

4. Стоимость (цена за единицу памяти)

На слайде представлена таблица сравнения дисковых накопителей информации, их сравнение по скорости чтения и записи, качественной вариации цены за единицу информации, долговечности и поддержки популярного серверного порта передачи данных.

В настоящее время данное сравнение претерпело значительные изменения с выходом SSD накопителей, поддерживающих порт PCIe четвертой версии. Скорость NVME накопителя может значительно отличаться от той что приведена на слайде в большую сторону.

**Оперативная память компьютера** - энергозависимая часть системы компьютерной памяти, в которой во время работы компьютера хранится выполняемый машинный код (программы), а также входные, выходные и промежуточные данные, обрабатываемые процессором.

**Оперативное запоминающее устройство** (ОЗУ) — устройство, реализующее функции оперативной памяти.

Оперативная память (RAM) в вычислительных системах используется для временного хранения данных и инструкций, которые используются процессором.

Она играет ключевую роль в обеспечении быстрого доступа к данным и выполнения операций.

Оперативная память является формой "кратковременного" хранения информации, в отличие от постоянной памяти, такой как жесткие диски или флэш-накопители.

Оперативная память обычно имеет высокую скорость доступа к данным, что позволяет процессору быстро получать необходимые данные для выполнения задач.

Однако оперативная память имеет ограниченную емкость, поэтому ее использование требует оптимизации и управления ресурсами.

Существует несколько типов оперативной памяти, таких как динамическая оперативная память (DRAM) и статическая оперативная память (SRAM), каждая из которых имеет свои особенности и применение в различных вычислительных системах.

Современные ОЗУ отличаются стандартами хранения.

Более новые версии стандартов отличаются более высокими номерами DDR.

Самый современный стандарт – DDR5 отличается высокой пропускной способностью, максимальным размером памяти, шириной машинного слова, скоростью работы.

Каждый новый стандарт памяти претерпевает значительные инженерные хитрости связанные с изменением задержки постановки данных на шину

Центральный процессор – интегральная схема, исполняющая машинные инструкции (коды программ).

Машинный код - специфицированный набор битов, обозначающих номер инструкции и поля данных над которыми необходимо произвести инструкции.

* **Процессор с точки зрения обработки данных характеризуется:**
* Тактовая частота:

Количество машинных операций в секунду, измеряется в Герцах.

В современных устройствах процессор способен показывать производительность порядка двух-трех МГц на одно ядро.

Это скорость, с которой процессор выполняет инструкции.

Чем выше тактовая частота, тем быстрее процессор обрабатывает данные.

2. Количество ядер и потоков:

Ядра процессора отвечают за выполнение задач, поэтому чем больше ядер, тем больше задач может выполняться параллельно.

Потоки позволяют каждому ядру выполнять несколько задач одновременно.

Современные серверные процессоры могут иметь около 10 и до 192 ядер одновременно

3. Быстродействующая память (кэш): Кэш - это быстрая память, которая используется для временного хранения данных, наиболее часто используемых процессором.

Больший размер кэша может улучшить производительность процессора.

4. Архитектура: Архитектура процессора определяет его внутреннюю структуру и способ организации работы. Различные архитектуры могут иметь различное поведение при обработке данных.

- CISC (Complex Instruction Set Computing) - В CISC-процессорах используются сложные инструкции, которые могут выполнять несколько операций одновременно.

Это позволяет сократить количество инструкций, необходимых для выполнения определенных задач.

CISC-процессоры обычно имеют большой набор инструкций, включающий в себя различные операции и адресные режимы.

Примеры CISC-архитектуры включают процессоры Intel x86 и AMD.

- RISC (Reduced Instruction Set Computing) - RISC-процессоры используют простые инструкции, каждая из которых выполняет только одну операцию.

Это упрощает дизайн процессора и увеличивает скорость выполнения инструкций.

RISC-процессоры обычно имеют ограниченный набор инструкций, что упрощает их дизайн и улучшает производительность.

Примеры RISC-архитектуры включают процессоры ARM и MIPS.

5. Разрядность: Это количество бит, которые процессор может обрабатывать за один раз. 32-разрядный процессор обрабатывает машинное слово длиной в 32 бита данных за один такт,

в то время как 64-разрядный процессор обрабатывает машинное слово в 64 бита данных за такт.

Большая разрядность может улучшить производительность при работе с большими объемами данных.

Современный Российский процессор Эльбрус обладает возможностью работы с длинным машинным словом в 128 бит, что увеличивает пропускную способность множества решений по обработке данных

Поговорим уже об устройствах, которые содержат в себе ту самую начинку, о которой мы говорили ранее.

На данный момент выделяют следующие виды вычислительных инфраструктур:

На данный момент выделяют следующие виды вычислительных инфраструктур:

* Персональные компьютеры (терминал доступа к серверу)
* Локальные вычислительные сервера
* Частный облачный сервис
* Общедоступный облачный сервис

Персональный компьютер на сегодняшний день в обработке больших данных используется для, в основном, отображения информации и разработки программ под вычислительные серверы.

Программисты используют ПК как терминал для ввода команд и запуска программ на удаленном сервере посредством терминала команд.

Также большие графические станции позволяют делать небольшие модели для обработки данных и визуализировать информацию на экране монитора.

**Типичная конфигурация современного ПК:**

* Центральный процессор   
  (x64, 4 ядра, ~ 2.6 ГГц)
* Оперативная память DDR4 8-16 Гб
* Дисковый накопитель 500-1000Гб (HDD/SSD)
* Операционная система Windows/Linux/MacOS с GUI

**Серверный компьютер** – единица серверной вычислительной инфраструктуры

Производители серверов предлагают устанавливать серверный компьютер в стойки, а стойки в кластер с применением сетевого взаимодействия между устройствами для возможной передачи данных между ними

**Типичная конфигурация одной серверной стойки:**

* Центральный процессор   
  (x64, 20 ядер, ~ 3 ГГц, до 8 процессоров)
* Оперативная память DDR4 ~ 512-2048 Гб
* Дисковый накопитель ~ 10-100Тб (HDD/SSD)
* Операционная система Linux Server CLI (Debian / CentOS / Red Hat)

Сейчас наблюдается все больший переход от вычисления и хранения данных на локальных серверах в сторону обработки данных на облачных решениях с возможностью аренды оборудования и экономии места на производстве или в офисе.

**Центры обработки данных (ЦОД)** — это специализированное здание или помещение, в котором компания размещает серверное и сетевое оборудование с последующим подключением клиентов к сети.

Функции ЦОД — обеспечить стабильную и безотказную работу размещённого в нём оборудования.

Кроме этого, любой дата-центр предоставляет защищённые каналы связи, по которым происходит обмен данными.

ЦОД обслуживает корпоративных клиентов и обеспечивает их ресурсами для вычислений и организации бизнеса

**Разберем**

**Уровень качества**

**Особенности обеспечения вычилениями**

**Отказоустойчивость в процентах и времени простоя**

Tier 1

В случае отказов работа ЦОД прерывается. Не предусмотрено обязательное использование источников бесперебойного питания и возможность резервирования данных.

Уровень доступности ЦОД такого уровня составляет 99,67%, что эквивалентно 28 часам реального времени простоя в год

Tier 2

Предусматривает улучшенные условия размещения оборудования: резервные источники электроснабжения, фальшполы, избыточные системные ресурсы, улучшенные системы охлаждения.

Уровень доступности ЦОД такого уровня составляет 99,75%, что эквивалентно 21 часу реального времени простоя в год

Tier 3

Не нужно останавливать для ремонта и профилактических работ. Для соответствия Tier 3 ЦОД должен иметь полное резервирование всех систем жизнеобеспечения.

Уровень доступности ЦОД такого уровня составляет 99,98%, что эквивалентно 1 часу и 40 минут реального времени

Tier 4

В требования этого стандарта входит двойное резервирование и полное дублирование всей системы.

99,99% - в таком случае простой составляет 52 минуты в год реального времени

ЦОД является местом размещения вычислительных устройств в стойках, соединенных коммутаторами.

Каждая стойка состоит из нескольких узлов – серверных компьютеров, также объединенных в локальную сеть передачи данных.

Каждый узел обладает своей операционной системой для обеспечения работы внутренних компонентов.

Узлы в стойке связаны между собой посредством систем виртуализации общей операционной системы и динамического выделения места на устройствах хранения и ресурсов вычисления, что позволяет обрабатывать большие объемы данных корпоративных клиентов.

Управление ЦОД осуществляется как на физическом, так и на программном уровне, вследствие чего возможен максимальный контроль для доступа к данным.

Доступ внешних пользователей осуществляется посредством различных интерфейсов удаленного доступа, главным образом посредством протокола безопасного соединения SSH.   
Доступ к данным обеспечивается из специальной виртуализированной кластерной системы, объединяющей несколько устройств.

Операторы дата-центров и облачные сервисы на коммерческой основе предоставляют ресурсы для развертывания вычислений или платформ для обработки данных

Дата-центры предлагают десятки тысяч стоек для нужд бизнеса и других отраслей экономики.

В РФ функционируют 4-5 крупнейших оператора дата-центров и до десятка крупнейших облачных сервисов у которых напрямую можно развернуть облачные сервисы вычислений.

Среди главных: DataPro – крупнейший IT провайдер в России

Также SELECTEL

YandexCloud

SberCloud

DataSpace и другие

В данном крайнем разделе мы кратко поговорим о том, какие операционные системы используются на персональных устройствах и серверных устройствах для корректной работы компонентов физической системы.

Операционные системы для управления вычислительным устройством бывают различных видов, среди них:

1. Серверные ОС – обеспечивают высокую оптимизацию ресурсов вычислительной системы, фокусируясь на работе основных модулей вычислений в ядре ОС, и не обеспечивая пользователя графической оболочкой. Предназначены для пользовательских или коммерческих серверов, файловых серверов, а также систем, направленных на обработку данных. В основном популярные дистрибутивы серверных операционных систем базируются на файловой системе Linux, ввиду чего обладают мощными утилитами командной строки, позволяющими опытным пользователям и администраторам уверенно работать в системе без графического интерфейса

Среди популярных серверных ОС выделяют: Debian OS, CentOS, Red Hat

2. Домашние или офисные операционные системы предназначены в основном для визуализации, отчетности, игр, и моделирования и т.д., то есть для производства и воспроизведения медиаконтента. В обработке больших данных также важно использовать и данные связанные с медиаформатом, поэтому офисные ОС также применяются в больших данных

Среди популярных ОС выделяют: Windows, MacOS, Ubuntu, Fedora, ArchLinux и дистрибутивы собранные на их базе

3. Операционные системы для суперкомпьютеров сталкиваются с проблемой оперирования большим количеством ядер процессора, а также большим количеством процессоров в распределенной системе или системе с общей памятью. В данной категории систем присутствуют как серверные системы на базе Linux, так и специализированные операционные системы. Данный класс ОС пользуется очень низким спросом ввиду нишевости решения.

4. Исследовательские ОС на данный момент являются плацдармом для начала изучения ОС в вузах и других учебных заведениях, и представляют собой UNIX или FreeBSD подобные операционные системы.

5. Мобильные ОС являются на данный момент главным рыночным сегментом, поскольку смартфоны приносят большую прибыль ввиду доступности устройств на рынке. В классификацию были добавлены для полноты картины

**Серверные ОС –** предназначены для управления программным обеспечением, которое в свою очередь обслуживает всех пользователей сети, как внутренней, так и внешней

Серверные ОС не предоставляют специализированный графический инструментарий управления системой и управляются напрямую с использованием командной строки

Серверные ОС являются более предпочтительными с точки зрения экономии ресурсов и гибкости использования управления системой на основе команд

Наибольшее распространение получили Linux системы с UNIX-подобными утилитами командной строки

Операционные системы для вычислительных распределенных кластеров, позволяющие администрировать множество узлов использует Linux/Unix-подобную модель команд и структуру файловой системы за счет своей популярности и открытости.

**Офисные/пользовательские ОС** снабжены графической оболочкой (интерфейсом), удобной для взаимодействия с компьютером посредством координатного устройства ввода-вывода(мышь, графический планшет, и т.д.)

Пользовательские ОС предоставляют возможность пользователю взаимодействовать с компьютером посредством визуальной ориентации

Пользовательские ОС позволяют визуализировать результаты вычислений, анализа и предлагают возможность пользователю воспринимать мультимедийную информацию визуального характера

Так и каким же образом вместе функционируют два данных вида операционных систем?

Для начала мы имеем вычислительный сервер на котором развернутая серверная ОС, основная задача которой является администрирование участников вычислений и собственно запуск этих вычислений.

Также мы имеем пользовательскую ОС с графическим интерфейсом, пользователь которой имеет возможность удобно использовать средства разработки, визуализации данных, использования браузерных утилит и т.д.

Задача на вычисления над имеющимися на сервере данными поступает с пользовательской ОС на серверную ОС посредством протокола безопасного соединения (ssh).

Произведенные вычисления сохраняются в промежуточные файлы данных результата в оптимизированном или сыром виде и передаются обратно в пользовательскую ОС посредством обратной передачи данных.

Результат визуализируется посредством клиентских программ и утилит, в результате чего зоны ответственности равномерно распределяются.

Добрый день уважаемые коллеги и учащиеся, сегодня мы продолжаем лекционный курс по дисциплине «Большие данные»

У нас с вами на очереди вторая тема: «Технологии сбора информации и больших объемов данных»

Рассмотрим классические типы данных в структурированных системах хранения данных

Дальше разберемся на примерах в определениях структурированных и неструктурированных данных. Выделим значимы примеры структурированных данных

Произведем обзор проблемы сбора данных в реалиях конвейера больших данных

А также познакомимся с технологией обмена сообщениями в системах обработки больших данных

Начнем с обзора и классификации типов данных в современных системах, хранящих отчетную или бизнес-информацию

Типы данных

Тип данных — атрибут, определяющий, какого рода данные могут храниться в объекте: целые числа, символы, данные денежного типа, метки времени и даты, двоичные строки и так далее.

Названия типов данных зачастую различаются от системы к системе.

В зависимости от характера значений все типы данных можно разделить на группы:

* числовые;
* дата и время;
* строковые;
* бинарные;
* прочие.

Названия типов данных зачастую различаются от системы к системе.

В зависимости от характера значений все типы данных можно разделить на группы:

* числовые;
* дата и время;
* строковые;
* бинарные;
* прочие.

На рисунке представлены популярные типы данных для хранения в реляционных SQL базах данных, хранящих структурированные данные.

Данная классификация и система типов необходима для введения стандартов обработки данных в различных прикладных программах.

На основе данной системы типов наука о данных существует уже порядка 30-40 лет с момента основания данного языка структурированных запросов.

На рисунке продемонстрированы все популярные типы данных и их представители в рамках реляционной модели хранения данных

Все целые числа хранятся в памяти компьютера в формате с фиксированной запятой.

Формат представления числа – это способ отображения содержимого в ячейке памяти..

Целые числа могут представляться в компьютере со знаком и без знака, т.е. храниться как отрицательными, так и положительными.

Множество целых чисел, которое может быть представлено в памяти компьютера, ограничено. Диапазон значений зависит от размера области памяти, используемой для размещения чисел.

В таблице на слайде подробно описаны стандарты хранения различных числовых типов данных, которые можно использовать в системах хранения и обработки данных

В некоторых средствах хранения и обработки данных типы данных упрощены на уровне абстракции представления чисел в системе с автоматических расширением разрядной сетке по мере появления значения целого числа вне заявленного стандартом диапазона, в результате чего запоминать их нет большой необходимости.

Целые числа используют для хранения данных, например, о количестве товара (и вообще любом количестве чего-либо), о расстоянии между точками, о сумме покупки.

Вещественные числа обычно представляются в виде чисел с плавающей запятой.

Числа с плавающей запятой — один из возможных способов представления действительных чисел, который является компромиссом между точностью и диапазоном принимаемых значений.

Число с плавающей запятой состоит из набора отдельных двоичных разрядов, условно разделенных на так называемые знак (англ. sign), порядок (англ. exponent) и мантиссу (англ. mantis). В наиболее распространённом формате (стандарт IEEE 754) число с плавающей запятой представляется в виде набора битов.

**Decimal** в системах SQL чаще всего принимает значение в диапазоне: *(−﷐10﷮38﷯)÷(﷐10﷮38﷯−1)*

Строковый тип — тип данных, значениями которого является произвольная последовательность (строка) символов некоторого алфавита. Каждая переменная такого типа может быть представлена фиксированным количеством байтов либо иметь произвольную длину.

В представлении строк в памяти компьютера существует два разных подхода:

* представление массивом символов (**char**) фиксированной длины;
* метод «завершающего байта» (**varchar**) — одно из возможных значений символов алфавита выбирается в качестве признака конца строки. В качестве признака конца строки используются символ 0, байт 0xFF (255) или код символа «$».

Формат дата/время хранит информацию о времени снятия измерения.

Тип данных **DATE** используется для величин с информацией только о дате в формате **'YYYY-MM-DD'**. При обработке данных допускаются и другие форматы, которые обычно интерпретируются разработчиком или аналитиком.

Тип данных **DATETIME** используется для величин, содержащих информацию как о дате, так и о времени в формате 'YYYY-MM-DD HH:MM:SS'. Формат также может быть изменён в зависимости от стандарта хранения.

Также стоит обратить внимание на стандарт хранения времени **timestamp**

**timestamp** - это последовательность символов или закодированной информации, показывающей, когда произошло определённое событие. Обычно показывает дату и время (иногда с точностью до долей секунд).

Формат **UNIX timestamp** предусматривает отсчет в секундах от заданной даты. Применяется во многих задачах обработки данных.

Бинарный тип данных

Эти типы данных используются для хранения необработанных двоичных данных длиной до 8,000 байт. Содержимое файлов изображений (файлы формата BMP, TIFF, GIF или JPEG), текстовых файлов и т.д. являются примерами двоичных данных.

Основные бинарные типы:

* binary (последовательность байтов фиксированной длины n);
* varbinary (последовательность байтов переменной длины n).

BLOB – большой двоичный объект. Он представляет строку переменной длины, значение которой составляет до 2,147,483,647 символов.

При помощи BLOB можно хранить данные, которые не могут размещаться в полях иного типа. Сюда относят:

* музыку;
* аудиозаписи;
* картинки;
* анимацию;
* иные компоненты.

В объектно-ориентированном подходе (ООП) все сущности формализуются набором **полей** и **методов.**

**Поля** характеризуют параметры сущности, а **методы** возможности воздействия на процесс и другие объекты.

В хранении данных принято хранить состояние об объекте или процессе в виде набора значений его характеристик, которые можно назвать **полем** объекта.

Объектный подход распространяется как на парадигму программирования, так и на системы хранения данных.

**Структура данных** в обработке и хранении данных это **перечень полей и их типов данных**, которыми представлена структурированная таблица данных.

На основе структуры данных можно проектировать сценарии обработки данных без наличия непосредственно записей в таблице (выборок).

Структура данных является отражением объектно-ориентированного подхода в обработке данных, и связано с понятием **структурированных данных**

**Структурированными** называются данные, отражающие отдельные факты предметной области и упорядоченные определенным образом с целью обеспечения возможности применения к ним различных методов обработки.

В случае таблиц данных подразумевается, что данные упорядочены по вертикали в типизированные столбцы, называемые **полями**, а по горизонтали — в строки, называемые **записями**.

В случае таблиц данных подразумевается, что данные упорядочены по вертикали в типизированные столбцы, называемые **полями**, а по горизонтали — в строки, называемые **записями**.

Также предполагается, что записи в структурированном наборе данных не являются противоречивыми и полными, а значения на пересечении записи и поля являются – атомарными, т.е. единственными в клетке.  
Таким образом можно обеспечить структуру в наборах данных

В случае таблиц данных подразумевается, что данные упорядочены по вертикали в типизированные столбцы, называемые **полями**, а по горизонтали — в строки, называемые **записями**.

Также предполагается, что записи в структурированном наборе данных не являются противоречивыми и полными, а значения на пересечении записи и поля являются – атомарными, т.е. единственными в клетке.  
Таким образом можно обеспечить структуру в наборах данных

*Временной ряд - Измерения показателя во времени для одного обособленного объекта.*

Содержат зависимые от времени и последовательности измерений данные о показателе выбранного объекта.

**Таблица хранит:**

* Временные метки (дата/время)
* Значения показателей (целочисленные или вещественные)

**Применение:**

* Экономика
* Геология
* Прикладная физика
* Химия
* ...

**Транзакционные данные** — это любая информация, которая связана с транзакциями.

Ключевое отличие транзакционных данных от других типов — это фиксация даты и времени.

Показатели не зависят друг от друга в последовательности

Для данных из сферы продаж, которые чаще всего являются транзакционными, также сохраняется

* вид платежа,
* продукт,
* количество покупок,
* применяемые скидки и промокоды,

Учитывается поведение клиентов до и после конверсии.

Примеры транзакционных данных:  
- Чеки магазина - сумма покупки, дата и время транзакции, номер кредитной карты.

- Запрос на изменение кода в системах контроля версий

- Оплата счетов за коммунальные услуги - сумма оплаты, дата и время транзакции, номер лицевого счета.

- Перевод денежных средств на другой банковский счет - сумма перевода, дата и время транзакции, реквизиты получателя.

- Снятие наличных в банкомате - сумма снятия, дата и время транзакции, номер банковской карты.

- Покупка билетов на мероприятие - сумма покупки, дата и время транзакции, номер заказа.

Таблицы данных объектов используются для хранения и извлечения больших двоичных объектов, например

* изображений,
* текстовых файлов,
* видео- и аудиопотоков,
* объектов данных
* и документов приложений большого размера.

Объект состоит из сохраненных данных, метаданных и уникального идентификатора доступа к объекту.

Хранилища объектов поддерживают отдельные большие файлы, а также позволяют управлять всеми файлами.

Немного поговорим о полуструктурированных данных.

К полуструктурированным данным относятся данные, подлежащие структуризации, но не хранящиеся в виде таблиц.

Полуструктурированные данные отличаются от структурированных форматом хранения, не обязывающим их привязываться к какой-либо предметной области, то есть отсутствует предметная ориентированность данных, или их прикладной смысл.

Полуструктурированные данные чаще всего представляют собой различные файлы с определенной заранее заданной нетабличной структурой, допускающей свободное внесение изменений в несуществующие поля данных. Среди таких обычно выделяют:

* CSV, XLS таблицы

XLS (Excel Spreadsheet) - Это формат файла, используемый Microsoft Excel для хранения табличных данных. XLS-файлы содержат данные в виде ячеек, которые могут содержать текст, числа, формулы и другие типы информации. Они часто применяются в бухгалтерском учете, финансовом анализе и других областях, где необходимо работать с табличными данными.

CSV (Comma-Separated Values) - Формат хранения данных, в котором значения разных полей разделены запятыми. Этот формат часто используется для хранения табличных данных, таких как базы данных, таблицы и электронные таблицы. CSV-файлы могут быть открыты и редактированы в текстовых редакторах или программах электронных таблиц.

XML (eXtensible Markup Language) - Это формат хранения данных, который используется для структурирования информации с использованием тегов.

XML-файлы могут содержать иерархические данные и метаданные, и они широко применяются в веб-разработке, обмене данными между приложениями и других областях, где необходимо структурировать и обменивать данные.

JSON-LD (JSON for Linking Data) - Это формат хранения данных, основанный на JSON (JavaScript Object Notation), который используется для связывания данных с помощью контекста и разметки.

JSON-LD широко применяется в семантической разметке веб-страниц, ведении блогов и других онлайн-проектах для улучшения видимости и интерпретации данных поисковыми системами.

YAML (YAML Ain't Markup Language) - Это формат хранения данных, который используется для представления структурированных данных с помощью отступов и списков.

YAML-файлы часто используются в конфигурационных файлах, скриптах настройки и других областях, где необходимо представить данные в читаемом человеком формате.

Таблица данных документов управляет набором значений-документов.

Обычно данные в этих хранилищах содержатся в виде документов JSON.

Каждое значение поля документа может представлять собой скалярный элемент, например число, или сложный объект, например список или коллекция типа "родитель — потомок".

Популярным представлением документа является представление в виде коллекции данных «Ключ-Значение». Каким чаще всего и представляется файл JSON.

Данные в полях документа можно закодировать разными способами, например в формате XML, YAML, JSON, или хранить в виде обычного текста.

Приложение может получать документы по ключу документа.

**Перейдем к понятию неструктурированных данных.**

**Неструктурированные данные**  — данные, которые не соответствуют заранее определённой модели данных, и, как правило, представлены в форме текста с датами, цифрами, фактами, расположенными в нём в произвольной форме.

Такие техники, как интеллектуальный анализ данных (data mining), обработка естественного языка (Natural Language Processing) и интеллектуальный анализ текста, предоставляют методы поиска закономерностей с целью так или иначе интерпретировать неструктурированную информацию.

Термин «неструктурированные данные» может считаться неточным по нескольким причинам:

* структура, даже если она не определена формально, может подразумеваться;
* данные, обладающие структурой некоторой формы, могут по-прежнему характеризоваться как неструктурированные, если их структура не предназначена для машинной обработки;
* неструктурированная информация может иметь некоторую структуру (такая информация называется полуструктурированной) или даже быть хорошо структурированной, но теми способами, которые являются неочевидными без предварительного согласования.

Приведем итоговое сравнение структурированных и неструктурированных данных:

**Структурированные данные характеризуется:**

Организованной, типизированной информацией

Она хранит в основном количественные данные

Структурированные данные хранятся в долговременном хранилище, так как представляют большую ценность для поддержания работоспособности бизнеса

**Неструктурирован-ные данные**

Множество форматов данных

Качественные

Озёра данных, файловые базы данных

Текстовые данные представляют собой неструктурированный источник информации.

Примером неструктурированных данных также служат фото и видео данные, записанные на камеры или созданные в графических редакторах.  
Такие данные могут обладать информацией в потоке кадров, но данная информация запечатлена в виде набора пикселей.

Структуризация данных рассматривается как отдельный механизм преобразования неструктурированных данных в удобный для обработки данных вид информации.

Структуризация данных доступна для таких данных как **текстовые** данные и **графовые** данные.

Данные структуризации не обладают достаточной эффективностью хранения и обработки.

Для текстовых данных возможна структуризация в виде вектора смысла текста, которые поддерживают векторные модели Word2Vec переводя каждое слово из заранее заданного словаря в форму n-мерного вектора чисел, взаимно отражающего смысл слова относительно векторов других слов.

Для графовых данных доступно структурированное табличное представление в форме разреженной матрицы инцидентности. Такая форма представления графа не является оптимальной для хранения, но является хорошим представлением для алгоритмов обработки данных.

В данной части лекции мы поговорим о конвейере обработки больших данных, а также о проблемах сбора данных

Классически поток обработки больших данных состоит из следующих стадий:

* Сбор и извлечение данных из внешних источников

Источники данных отличаются своей вариативностью форматов и данных, которые поступают в систему

Затем

Происходит предобработка и структуризация данных.

На этой стадии первично решаются проблемы неполных данных, пропущенных значений, преобразования данных, вычисления агрегатов и решение проблемы синхронизации источников данных перед прохождением их дальше по конвейеру обработки данных

Загрузка данных в долговременное хранилище данных.

Долговременное хранилище данных состоит обязательно из нескольких компонент  
1. Вычислительный сервер для хранения структурированных данных

2. Сервер хранения сырых файловых данных  
3. Сервер для хранения агрегатов и вычислений группировок для аналитики больших данных в оптимизированном формате

Витрины данных, которые являются подготовленными хранилищами данных для различных пользователей системы с различными целями для анализа и обработки данных

И конечные пользователи, которые используют данные для решения своих задач аналитики, построения визуализации данных, мониторинга оперативных данных и решения задач машинного обучения

Представленный конвейер обработки больших данных является устоявшейся классикой. Вариации данного пайплайна являются крайне необходимыми в зависимости от решаемых задач и данных, которые необходимо обрабатывать. Данная схема является общеструктурной и не отражает инструментов, которые используются на каждой из стадий.

В данном разделе лекции мы сфокусируемся именно на сборе данных, задачах и проблемах которые возникают при получении данных из различных источников.

Данные в поток обработки попадают из различных источников. Настроенный источник данных также называют **подключением**

Данные в поток обработки попадают из различных источников. Настроенный источник данных также называют **подключением**

Среди источников данных выделяют:

Программный интерфейс приложения (API): Это специальный набор правил и протоколов, который позволяет различным программам взаимодействовать друг с другом.

API может быть использован для получения данных из различных источников, таких как веб-серверы, социальные сети, онлайн-сервисы и т.д.

2. SQL и NoSQL базы данных: SQL (Structured Query Language) и NoSQL (Not Only SQL) базы данных являются основными хранилищами структурированных данных.

SQL базы данных используют язык SQL для запросов и манипуляции данными, в то время как NoSQL базы данных предоставляют более гибкую модель хранения и доступа к данным.

Файлы данных могут быть использованы для хранения различных типов информации, таких как текстовые документы, изображения, видео, аудио и другие файлы.

Они могут быть использованы как источник данных для обработки и анализа.

Потоковый сервис предоставляет данные в реальном времени или в виде непрерывного потока.

Это может быть использовано для обработки событий, мониторинга, аналитики и других приложений, которые требуют оперативной обработки данных.

Потоковый сервис предоставляет данные в реальном времени или в виде непрерывного потока.

Это может быть использовано для обработки событий, мониторинга, аналитики и других приложений, которые требуют оперативной обработки данных.

На стадии **извлечения** и **сбора** данных ставится задача загрузки данных из нескольких внешних или внутренних источников организации в поток обработки данных.

Озвучим классические проблемы, которые возникают на этапе сбора данных

Количество внешних источников данных может быть колоссальным в реальном продукте.

Среди таких источников запросто могут быть потоковые сервисы, сайты официальных организаций, интерфейсы к открытым источникам данных и внутренние источники данных.

Для решения данной проблемы зачастую используют решения по менеджменту подключений, в основе которых лежит возможность распределенной обработки сообщений.

Таким образом все возможные потребители данных смогут в сортированном порядке выбрать необходимый источник данных из списка зарегистрированных и подготовленных в компании или команде.

Для каждого такого источника данных имеет место быть проблема, связанная с разнородностью интерфейса или формата предоставления данных.

В случае данных различного формата вид передачи данных, представление и синтаксис подключения к их источнику может значительно отличаться.

В результате чего применяют инструменты миграции данных, которые обладают встроенным функционалом базовой настройки извлечения данных.

При большом объеме собираемых данных нагрузка на долговременное хранилище данных при загрузке может быть велика, что приведет либо к сбоям, либо к потерям данных.

Лучше всего загружать подготовленные данные в систему долговременного хранения данных пакетами – то есть определенным количеством заранее подготовленных записей.

Таким образом возникает проблема согласованности времени сбора данных.

Когда данные являются операционными, т.е. критичными ко времени загрузки в операционное хранилище, то имеет место быть проблема согласованности времени сбора данных для их объективизации и своевременного обновления.

Данная проблема решается с помощью предварительной области, называемой staging area или зона хранения, где данные, преобразовываясь в реальном времени, складываются в единое множество и загружаются по расписанию в хранилище бизнес-данных, т.е. долговременное хранилище данных

Пропуски в данных являются ключевой проблемой при сборе данных. На данной стадии необходима автоматическая предобработка данных или отбрасывание недостоверных записей с критическим количеством ошибок

Количество собираемых данных это проблема на систему хранения данных на серверных устройствах.

Данная проблема решается периодической многоступенчатой архивацией неактуальных данных и удалением данных, выходящих за период обязательного хранения в рамках регламента предметной области.

В результате сбора данных из различных источников необходимо производить их слияние для обеспечения консолидации хранилища данных

API (Application Programming Interface) - это набор протоколов, инструментов и определений, которые позволяют различным программам взаимодействовать друг с другом.

API позволяет программистам использовать готовые функции и процедуры для обращения к внешним сервисам или компонентам приложения.

Оно важно для обработки данных, так как позволяет программам обмениваться информацией, выполнять запросы и получать ответы от удаленных серверов или сервисов.

API помогает отделить одно приложение от другого, находясь между системами.

API накладывает свои правила на запрос клиента и ответ от сервера.

Благодаря этому приложения работают согласованно и данные защищены с обеих сторон.

С помощью API мы имеем возможность

Создатели некоторых сервисов понимают, что находящаяся на их сайтах информация может быть полезна для аналитики.

Тогда они предоставляют специальный набор ссылок (**урлов**), на которые можно отправлять HTTP-запросы.

Тогда вместо получения html-страницы, как в браузере, сайт будет отдавать сразу структурированные данные, например, в формате JSON.

Совокупность форматов таких запросов образует API (Application Program Interface).

В таком случае сайт может отдавать только те данные, которыми хочет делиться, следить за количеством обращений к этим данным, а для аналитика такой вариант лучше, поскольку позволяет быстрее интерпретировать результат.

На слайде вы можете видеть различные популярные API для получения данных, с помощью которых можно проводить различные аналитические исследования на реальных данных.

Среди популярных API на российском рынке представлены API HeadHunter сервиса поиска работы, Kinopoisk сервиса подписки для просмотра кино и сериалов, VK API интерфейс к популярному сервису Вконтакте.

REST (Representational State Transfer) - это архитектурный стиль для построения веб-сервисов, который использует стандартные HTTP методы (GET, POST, PUT, DELETE) для доступа и изменения данных.

RESTful API использует унифицированный интерфейс для взаимодействия с удаленными ресурсами, предоставляя данные в формате JSON или XML.

Чаще всего передача данных по REST API работает на основе протокола HTTP. Он использует четыре HTTP-метода, которые позволяют произвести подключение по API и взаимодействовать с объектами на сервере:

* **GET**, чтобы получить список объектов;
* **DELETE**, чтобы удалить данные;
* **POST**, чтобы добавить или заменить данные;
* **PUT** для регулярного обновления данных.

Второе название этих методов — идентификаторы *CRUD*. Это аббревиатура, образованная от четырёх действий: создать (create), прочитать (read), обновить (update) и удалить (delete).

Метод **GET** позволяет получать данные с сервера предоставления данных по запросу.

**Пример**: **GET** https://api.hh.ru/vacancies/[идентификатор\_вакансии]

Вернет JSON файл с данными об интересующей вас вакансии

Запрашиваемый JSON файл содержит данные в формате ключ-значение, которые в дальнейшем подлежат определенной структуризации в виде таблицы данных.

Видим, что вернувшийся файл соответствтует данным вакансии “Менеджер по региональным продажам”

SOAP API (Simple Object Access Protocol) - это протокол обмена структурированными информационными сообщениями в распределенной среде. SOAP API позволяет клиентам обращаться к удаленным сервисам и выполнять операции над данными. SOAP API использует XML для формата сообщений и HTTP для передачи.

gRPC - это высокопроизводительный открытый фреймворк для удаленного вызова процедур (RPC), разработанный Google.

gRPC использует протокол HTTP/2 для передачи данных и Protocol Buffers для определения интерфейсов и сериализации данных.

Он обеспечивает эффективную передачу данных между клиентом и сервером и поддерживает множество языков программирования.

Благодаря оптимизации и режиму работы в формате потоковой передачи данных поддерживает большую пропускную способность и скорость интерпретации результатов чем API по причине передачи блоков данных в заранее объявленных типах без необходимости их декодирования.

GraphQL - это язык запросов и среда выполнения, разработанные Facebook. GraphQL позволяет клиентам запрашивать только необходимые данные и получать ответы в удобном формате. Он предоставляет более гибкий и эффективный способ получения данных по сравнению с традиционными REST API.

Apache Kafka - это распределенная система потоковой обработки и передачи сообщений. Kafka обеспечивает надежную и масштабируемую передачу данных между различными компонентами системы. Он широко используется для создания потоковых приложений, обработки журналов, мониторинга и аналитики данных.

В последнем разделе нашей лекции мы рассмотрим принципы работы такого инструмента, как системы обмена сообщениями в системах обработки больших данных.

Существует несколько систем обмена сообщениями, которые могут использоваться в обработке больших данных. Некоторые из них включают:

* Apache Kafka: Это распределенная платформа для обработки потоков данных и обмена сообщениями. Она может использоваться для передачи больших объемов данных между различными компонентами системы обработки данных в реальном времени.

Apache Kafka можно сравнить с почтой — одни сервисы передают туда сообщения-письма, а другие — получают.

Apache Kafka называют брокером сообщений, потому что она выступает в качестве посредника.

Платформу разработали в 2010 году инженеры компании LinkedIn.

Они столкнулись с проблемой обработки и передачи большого объёма данных в режиме реального времени.

Сервисы, которые в целом умели это делать, существовали, но не справлялись с огромными объёмами данных.

Изначально Kafka разрабатывали как систему для внутреннего пользования, но потом выпустили её на широкий рынок.

Постепенно она завоевала популярность благодаря своей масштабируемости, высокой пропускной способности и устойчивости к отказам.

И в 2012 году её передали компании Apache Software Foundation.

Сейчас она используется во многих отраслях: электронной коммерции, банковском деле, телекоммуникации и других.

2. Apache Pulsar: Это другая распределенная система обмена сообщениями, которая предоставляет масштабируемую и устойчивую платформу для обработки потоков данных.

3. RabbitMQ: Это популярная система обмена сообщениями, которая может быть использована для обмена сообщениями и событиями в системах обработки больших данных.

У Kafka есть множество способов применения, и у каждого способа есть свои особенности. В этой части лекции мы разберём, чем Kafka отличается от популярных систем обмена сообщениями; рассмотрим, как Kafka хранит данные и обеспечивает гарантию сохранности; поймём, как записываются и читаются данные

Для первого погружения в технологию сравним Kafka и классические сервисы очередей, такие как, например, RabbitMQ

Системы очередей обычно состоят из трёх базовых компонентов:

1) сервер на котором хранятся очереди сообщений и данные из этих сообщений,  
2) производители (producer), которые отправляют сообщения в некую именованную очередь, заранее сконфигурированную администратором на сервере. Продьюсеры являются поставщиками данных из различных источников. Их предназначение снизить затраты на менеджмент потоков входных данных, а также предотвратить такие проблемы как синхронизация данных во времени или количество внешних источников данных,  
3) потребители (consumer), которые считывают те же самые сообщения по мере их появления. Потребители данных являются обработчиками, которые передают данные из очереди сообщений в долговременное или операционное хранилище данных

В веб-приложениях очереди часто используются для отложенной обработки событий или в качестве временного буфера между другими сервисами, тем самым защищая их от всплесков нагрузки.

Потребители получают данные с сервера, используя две разные модели запросов: pull или push.

**pull-модель** — потребители сами отправляют запрос раз в n секунд на сервер для получения новой порции сообщений. При таком подходе клиенты могут эффективно контролировать собственную нагрузку. Кроме того, pull-модель позволяет группировать сообщения в пакеты по несколько сообщений, таким образом достигая лучшей пропускной способности. К минусам модели можно отнести потенциальную разбалансированность нагрузки между разными потребителями, а также более высокую задержку обработки данных.

**push-модель** — сервер делает запрос к клиенту, посылая ему новую порцию данных. По такой модели, например, работает RabbitMQ. Она снижает задержку обработки сообщений и позволяет эффективно балансировать распределение сообщений по потребителям.

Как правило, приложение пишет и читает из очереди с помощью нескольких источников сообщений продюсеров и потребителей. Это позволяет эффективно распределить нагрузку между участниками обмена.

**Типичный жизненный цикл сообщений в системах очередей:**

* Производитель отправляет сообщение на сервер.
* Потребитель запрашивает сообщение и его уникальный идентификатор сервера.
* Сервер помечает сообщение как in-flight (в пути). Сообщения в таком состоянии всё ещё хранятся на сервере, но временно не доставляются другим потребителям. Таймаут этого состояния контролируется специальной настройкой.
* Потребитель обрабатывает сообщение, следуя бизнес-логике. Затем отправляет ack или nack-запрос обратно на сервер, используя уникальный идентификатор, полученный ранее — тем самым либо подтверждая успешную обработку сообщения, либо сигнализируя об ошибке.
* В случае успеха сообщение удаляется с сервера навсегда. В случае ошибки или таймаута состояния in-flight сообщение доставляется потребителю для повторной обработки.

С базовыми принципами работы очередей разобрались, теперь перейдём к Kafka. Рассмотрим её фундаментальные отличия.

Как и сервисы обработки очередей, Kafka условно состоит из трёх компонентов:

1) сервер (по-другому ещё называется брокер),  
2) продюсеры — они отправляют сообщения брокеру,  
3) потребители — считывают эти сообщения, используя модель pull.

Пожалуй, фундаментальное отличие Kafka от очередей состоит в том, как сообщения хранятся на брокере и как потребляются потребителями.

* Сообщения в Kafka **не удаляются** брокерами по мере их обработки потребителями — данные в Kafka могут храниться днями, неделями, годами.
* Благодаря этому одно и то же сообщение может быть обработано **сколько угодно раз** разными потребителями и в разных контекстах.

В этом кроется главная мощь и главное отличие Kafka от традиционных систем обмена сообщениями.

Теперь давайте посмотрим, как Kafka и системы очередей решают одну и ту же задачу. Начнём с системы очередей.

Представим, что есть некий сайт, на котором происходит регистрация пользователя. Для каждой регистрации мы должны:

1) отправить письмо пользователю,  
2) пересчитать дневную статистику регистраций.

В случае с RabbitMQ или Amazon SQS функционал может помочь нам доставить сообщения всем сервисам одновременно. Но при необходимости подключения нового сервиса придётся конфигурировать новую очередь.

Kafka упрощает задачу. Достаточно послать сообщения всего один раз, а потребители сервиса отправки сообщений и потребители статистики сами считают его по мере необходимости.

Kafka также позволяет тривиально подключать новых подписчиков и сервисы к потоку регистрации.

Например, сервис архивирования всех регистраций в S3 для последующей обработки с помощью Spark или Redshift можно добавить без дополнительного конфигурирования сервера или создания дополнительных очередей.

Кроме того, раз Kafka не удаляет данные после обработки потребителями, эти данные могут обрабатываться заново, как бы отматывая время назад сколько угодно раз. Это оказывается невероятно полезно для восстановления после сбоев и, например, верификации кода новых потребителей. В случае с RabbitMQ пришлось бы записывать все данные заново, при этом, скорее всего, в отдельную очередь, чтобы не сломать уже имеющихся клиентов.

Наверняка возникает вопрос: «Раз сообщения не удаляются, то как тогда гарантировать, что потребитель не будет читать одни и те же сообщения (например, при перезапуске)?».

Для ответа на этот вопрос разберёмся, какова внутренняя структура Kafka и как в ней хранятся сообщения.

Каждое сообщение (event или message) в Kafka состоит из ключа, значения, таймстампа и опционального набора метаданных (так называемых хедеров).

Например (см. рисунок)

Ключ (Key): “Alice”

Значение (Value): “Registered on our website”

Временная метка (Timestamp): “Jun. 25, 2020 at 2:06 p.m.” (CreateTime, LogAppendTime)

Заголовки (Headers): [{“X-Generated-By”: “web-host-12.eu-west2.slurm.io”}]

Сообщения в Kafka организованы и хранятся в именованных топиках (Topics), каждый топик состоит из одной и более партиций P (Partition), распределённых между брокерами внутри одного кластера.

Подобная распределённость важна для горизонтального масштабирования кластера, так как она позволяет клиентам писать и читать сообщения с нескольких брокеров одновременно.

Когда новое сообщение добавляется в топик, на самом деле оно записывается в одну из партиций этого топика.

Сообщения с одинаковыми ключами всегда записываются в одну и ту же партицию, тем самым гарантируя очередность или порядок записи и чтения.

Для гарантии сохранности данных каждая партиция в Kafka может быть реплицирована n раз, где n — replication factor (гиперпараметр настройки Kafka).

Таким образом гарантируется наличие нескольких копий сообщения, хранящихся на разных брокерах.

У каждой партиции есть «лидер» (Leader) — брокер, который работает с клиентами.

Именно лидер работает с продюсерами и в общем случае отдаёт сообщения потребителям.

К лидеру осуществляют запросы подписчики - фолловеры (Follower) — это брокеры, которые хранят реплику всех данных партиций.

Сообщения всегда отправляются лидеру и, в общем случае, читаются с лидера.  
Чтобы понять, кто является лидером партиции, перед записью и чтением клиенты делают запрос метаданных от брокера.

Причём они могут подключаться к любому брокеру в кластере.

Основная структура данных в Kafka — это распределённый, реплицируемый лог.

Каждая партиция — это и есть тот самый реплицируемый лог, который хранится на диске.

Каждое новое сообщение, отправленное продюсером в партицию, сохраняется в «голову» этого лога и получает свой уникальный, монотонно возрастающий порядок - offset (64-битное число, которое назначается самим брокером).  
Как мы уже выяснили, сообщения не удаляются из лога после передачи потребителям и могут быть вычитаны сколько угодно раз.  
Время гарантированного хранения данных на брокере можно контролировать с помощью специальных настроек.

Длительность хранения сообщений при этом не влияет на общую производительность системы.

Поэтому совершенно нормально хранить сообщения в Kafka днями, неделями, месяцами или даже годами.

Группы потребителей  
Теперь давайте перейдём к потребителям и рассмотрим их принципы работы в Kafka.

Каждый потребитель Kafka обычно является частью какой-нибудь группы потребителей.

Каждая группа имеет уникальное название и регистрируется брокерами в кластере Kafka.

Данные из одного и того же топика могут считываться множеством групп потребителей одновременно.

Когда несколько потребителей читают данные из Kafka и являются членами одной и той же группы, то каждый из них получает сообщения из разных партиций топика, таким образом распределяя нагрузку.

Вернёмся к нашему примеру с топиком сервиса регистрации и представим, что у сервиса отправки писем есть своя собственная группа потребителей с одним потребителем *c1* внутри.

Значит, этот потребитель будет получать сообщения из всех партиций топика.

Если мы добавим ещё одного потребителя в группу, то партиции автоматически распределятся между ними, и *c1* теперь будет читать сообщения из первой и второй партиции, а *c2* — из третьей.

Добавив ещё одного потребителя (*c3*), мы добьёмся идеального распределения нагрузки, и каждый из потребителей в этой группе будет читать данные из одной партиции.

А вот если мы добавим в группу ещё одного потребителя (*c4*), то он не будет задействован в обработке сообщений вообще.  
Важно понять: внутри одной группы потребителей партиции назначаются потребителям уникально, чтобы избежать повторной обработки.  
Если потребители не справляются с текущим объёмом данных, то следует добавить новую партицию в топик. Только после этого потребитель c4 начнёт свою работу.

Механизм партиционирования является нашим основным инструментом масштабирования Kafka. Группы являются инструментом отказоустойчивости.

В случае если один из потребителей в группе перестанет работать, партиции автоматически распределятся между оставшимися потребителями в этой группе.

Добавлять партиции в Kafka можно на лету, без перезапуска клиентов или брокеров.

Клиенты автоматически обнаружат новую партицию благодаря встроенному механизму обновления метаданных.

Однако, нужно помнить две важные вещи:

* Гарантия очерёдности данных — если вы пишете сообщения с ключами и кодируете (например с помощью хэширования) номер партиции для сообщений, исходя из общего числа, то при добавлении новой партиции вы можете просто сломать порядок этой записи.

Партиции невозможно удалить после их создания, можно удалить только весь топик целиком.

Помимо этого, механизм групп позволяет иметь несколько несвязанных между собой приложений, обрабатывающих сообщения.

Как мы обсуждали ранее, можно добавить новую группу потребителей к тому же самому топику, например, для обработки и статистики регистраций.

Эти две группы будут читать одни и те же сообщения из топика тех самых событий регистраций — в своём темпе, со своей внутренней логикой.

А теперь, зная внутреннее устройство потребителей в Kafka, давайте вернёмся к изначальному вопросу: **«Каким образом мы можем обозначить сообщения в партиции, как обработанные?»**.

Для этого Kafka предоставляет механизм офсетов (порядков) потребителей.

Как мы помним, каждое сообщение партиции имеет свой собственный, уникальный, монотонно возрастающий офсет.

Именно этот офсет и используется потребителями для сохранения партиций.

Потребитель делает специальный запрос к брокеру, так называемый offset-commit (передача порядка) с указанием своей группы, идентификатора пары топик-партиции и, собственно, офсета, который должен быть отмечен как обработанный.

Брокер сохраняет эту информацию в своём собственном специальном топике.

При рестарте потребитель запрашивает у сервера последний запрошенный офсет для нужной топик-партиции, и просто продолжает чтение сообщений с этой позиции.

В примере потребитель в группе email-service-group, читающий партицию *p1* в топике registrations, успешно обработал три сообщения с офсетами 0, 1 и 2.

Для сохранения позиций потребитель делает запрос к брокеру, передавая офсет равный 3.

В случае рестарта потребитель запросит свою последнюю переданную позицию у брокера и получит в ответе 3.

После чего начнёт читать данные с этого офсета.

Потребители вольны коммитить (передавать) совершенно любой офсет (валидный, который действительно существует в этой топик-партиции) и могут начинать читать данные с любого офсета, двигаясь вперёд и назад во времени, пропуская участки лога или обрабатывая их заново.

Ключевой для понимания факт: в момент времени может быть только один записанный офсет для топик-партиции в группе потребителей.

Иными словами, мы не можем передать несколько офсетов для одной и той же топик-партиции, эмулируя каким-то образом выборочное ack-сообщение (как это делалось в системах очередей).

Представим, что обработка сообщения с офсетом 1 завершилась с ошибкой.

Однако мы продолжили выполнение нашей программы в потребителе и обработали сообщение с офсетом 2 успешно. В таком случае перед нами будет стоять выбор: какой офсет передать — 1 или 3.

В настоящей системе мы бы рекомендовали передать офсет 3, добавив при этом функционал, отправляющий ошибочное сообщение в отдельный топик для повторной обработки (ручной или автоматической).

Подобные алгоритмы называются Dead letter queue.

Разумеется, потребители, находящиеся в разных группах, могут иметь совершенно разные переданные офсеты для одной и той же топик-партиции.

Существуют и другие брокеры сообщений, однако у Apache Kafka есть перед ними несколько преимуществ:

Существуют и другие брокеры сообщений, однако у Apache Kafka есть перед ними несколько преимуществ:  
  
✅ **Масштабируемость**. В Kafka можно легко добавлять дополнительные разделы, чтобы увеличивать пропускную способность системы и обрабатывать больше запросов. Это особенно полезно для систем, которые быстро растут или подвержены резким всплескам нагрузки — например, при распродажах в «Чёрную пятницу».  
  
Благодаря масштабируемости Apache Kafka отлично подходит для работы с большими объёмами данных, их хранения и аналитики. Поэтому умение работать с этой системой важно для всех, чья профессия связана с Big Data, — в том числе для инженеров данных.

✅ **Отказоустойчивость**. Благодаря механизму репликации Kafka сохраняет данные в случае сбоев и продолжает работать, даже если что-то сломается.

✅ **Универсальность**. В основе работы Kafka простая схема, которая подходит для разных сценариев: от объединения микросервисов внутри приложения до аналитики больших данных.

✅ **Удобные интеграции**. Как часть системы Apache, Kafka легко интегрируется с другими продуктами этой системы: Spark, Storm, Flink и пр.

✅ **Открытость**. Это open source проект — он бесплатен и открыт для изменений всеми пользователями. При желании его можно адаптировать под потребности конкретной компании.

✅ **Надежности доставки**. Брокер сообщений Kafka использует модель «точная однократная доставка». Это значит, что сообщение будет либо доставлено успешно, либо не будет доставлено вообще. Такое отсутствие дублирования критически важно для некоторых систем — например, для банков, чтобы с клиента дважды не списались деньги. Если же сообщение не будет доставлено, издатель попытается отправить его повторно и рано или поздно преуспеет.

✅ **Возможность хранения данных**. Kafka может не только доставлять, но и хранить сообщения длительное время, чтобы их можно было обработать и проанализировать позже. Время хранения можно конфигурировать в зависимости от потребностей.

✅ **Простота использования**. API Kafka удобно подключать к любой системе. Также в ней есть готовые клиентские библиотеки для различных языков программирования.

Producer API: Producer API используется для отправки сообщений в Kafka. Он позволяет приложениям записывать данные в темы Kafka.

Consumer API: Consumer API позволяет приложениям читать данные из Kafka тем. Он обеспечивает возможность подписки на темы и получения сообщений из них.

Admin Client API: Admin Client API предоставляет административные операции для Kafka кластера, такие как создание, удаление и конфигурирование тем и брокеров Kafka.

Connect API: Connect API используется для интеграции Kafka с внешними системами, позволяя им отправлять данные в Kafka или получать данные из Kafka.

Kafka Streams API: Kafka Streams API предоставляет библиотеку для обработки и анализа данных в реальном времени с использованием Kafka. Он позволяет разрабатывать приложения для обработки потоков данных, такие как агрегация, фильтрация и преобразование данных.

Kafka API Поддерживает такие языки программирования как Java, Scala и Python

**Где применяется Apache Kafka**

Apache Kafka используется практически в любой сфере: электронной коммерции, производстве, финансах, медицине. Вот какие задачи обычно поручают платформе:

**1. Обработка потоков данных в реальном времени.** То есть работа с данными ровно в тот момент, когда они появились или поступили, практически без задержки. Например, в финансовой сфере Kafka может использоваться для отображения и анализа цен акций и изменений рынка.

**2. Интеграция систем и сервисов между собой.** Часто в продукте используется несколько разных систем. Например, такое случается при микросервисной разработке, когда каждую функцию разрабатывают как отдельный сервис. Kafka обеспечивает связь между этими сервисами и системами, помогая им обмениваться информацией.

**3. Логирование.** Kafka может принимать и сохранять логи различных событий, а также предоставлять их по запросу.

**4. Аналитика больших данных.** С помощью брокера сообщений Kafka можно отправлять большие объемы данных на анализ, а также хранить их для будущей аналитики. Это делает Apache Kafka полезной для инженеров данных.

Добрый день уважаемые учащиеся и коллеги,

я рад вновь приветствовать вас на курсе по большим данным.

Сегодня мы проходим новую тему, а именно «Технологии структурирования данных. Таблицы данных»

В данной теме мы охватим следующие разделы

Хранение информации в виде структурированных данных в реляционной БД, а также охватим принципы работы систем управления базами данных

Коснемся особенностей реляционной модели данных и связям таблиц в базе данных, а также нормальных форм

Погрузимся в транзакции, согласованность данных, ACID и языка управления транзакциями

Начнем с простого, а именно с хранения информации в виде таблиц

**Начнем с понятия базы данных и модели данных, какие бывают модели данных и для какой информации лучше всего подходят разные модели**

**Таким образом.**

**База данных (БД)** – это совокупность данных, хранящихся и упорядоченных в соответствии с определенной **структурой.**

**Это могут быть данные которые хранят структурированные таблицы, изображения, видео, текст, объекты данных, код приложений и многое другое.**

**Модель данных** определяет то, как и каким образом данные будут **располагаться** в БД и как к ним будет **предоставляться доступ.**

Если проще, то БД это просто информация с которой мы работаем.

С базой данных нельзя полноценно взаимодействовать не используя систему управления базами данных.

**Перейдем к видам моделей данных для хранения информации в компьютерных системах**

Современная классификация насчитывает несколько моделей данных, среди которых самые важные три модели:

1. Реляционые данные – классические связанные табличные данные.

2. Сетевые – графовые данные, хранящие информацию о сущностях в которых важна связь и сила связи. Пример: контакты в социальных сетях, транспортные потоки между городами, и т.д.

3. Иерархическая модель – классическая модель хранения данных на дисках и в операционных системах.

Реляционная модель хранения данных:

* Принцип хранения данных: Данные хранятся в виде таблиц, где каждая строка представляет отдельную запись, а каждый столбец - отдельное поле.

Таблицы связаны между собой с помощью ключей, что позволяет эффективно организовывать и структурировать данные.

2. Вычисления над данными: Для выполнения вычислений над данными используются структурированные языки запросов, такие как языки SQL.

Эти языки позволяют осуществлять операции выборки, вставки, обновления и удаления данных из таблиц.

Подробнее о данной модели будет рассказано далее в этой лекции

Сетевая модель тесно сопряжена с понятиями граф и графовая база данных

В графовой модели данные представлены в виде узлов (вершин) и связей (ребер) между этими узлами.

Это позволяет эффективно моделировать и анализировать сложные взаимосвязи между данными.

Графовая база данных (Graph Database) - это тип базы данных, который использует графовую модель для хранения и обработки данных.

В графовых базах данных данные обычно хранятся в виде узлов и связей между этими узлами.

Узлы представляют объекты или сущности, а связи определяют отношения между этими объектами.

Каждый узел может иметь различные атрибуты, которые описывают его свойства,

а связи могут иметь направление и тип, что позволяет более точно моделировать взаимосвязи между данными.

Для выборки и обработки данных в графовых базах данных существуют различные способы.

Один из наиболее распространенных способов - использование языка запросов Cypher, который специально разработан для работы с графовыми данными.

С помощью Cypher можно выполнять запросы для поиска путей между узлами, анализировать структуру графа, находить паттерны и т.д.

Также существуют различные алгоритмы для анализа графовых данных, такие как

поиск кратчайшего пути,

выявление сообществ в графе,

поиск циклов и т.д.

Эти алгоритмы позволяют проводить сложный анализ данных в графовых базах данных и находить интересные закономерности и взаимосвязи.

Иерархическая модель данных - это структура организации данных, в которой информация представлена в виде иерархии, состоящей из уровней.

Каждый уровень содержит данные, связанные с более высоким уровнем, образуя древовидную структуру.

В иерархической модели данных каждый элемент данных может иметь только один родительский элемент, но может иметь несколько дочерних элементов.

В иерархической базе данных данные хранятся в виде набора записей, которые связаны между собой отношением "родитель-потомок".

Каждая запись может содержать набор полей, которые описывают свойства объекта, и указатель на родительскую запись.

Таким образом, данные организованы в виде древовидной структуры, где каждая запись может иметь несколько дочерних записей.

Это позволяет эффективно организовывать данные с иерархическими отношениями, например как хранение данных в операционных системах.

Иерархическая модель данных хорошо подходит для хранения в файловых базах данных, хранящих неструктурированные данные, и используется в таких популярных системах как Hadoop, MongoDB и других.

Базу данных невозможно было бы изменить или заполнить не будь **системы для её управления**

**Система управления базами данных (СУБД)** представляет из себя совокупность программных и языковых средств для создания, удаления, изменения и любых других манипуляций с данными в БД.

СУБД работает в соответствии с моделью, на которой строится база данных

**Все манипуляции с базой данных и с данными происходят через СУБД**

**Основными функциями СУБД являются:**

Управление данными во внешней памяти:

СУБД управляет данными, хранящимися на диске или других внешних устройствах.

Она отвечает за организацию данных, их хранение, доступ и обновление.

СУБД использует различные методы индексации, кэширования и оптимизации запросов для эффективного управления данными во внешней памяти.

Управление буферами оперативной памяти:

СУБД управляет буферами оперативной памяти для эффективного доступа к данным.

Она оптимизирует использование оперативной памяти, кэширует данные из внешней памяти для быстрого доступа и управляет вытеснением данных из буфера при необходимости.

Поддержка языков базы данных:

Она обеспечивает поддержку языков базы данных для выполнения запросов, транзакций и хранимых процедур.

Журнализация и резервное копирование базы данных:

СУБД обеспечивает механизмы для журнализации изменений в базе данных,

что позволяет восстановить состояние базы данных в случае сбоя.

Она также предоставляет средства для создания резервных копий данных,

чтобы обеспечить защиту от потери информации в случае аварии или ошибки.

СУБД состоит из нескольких компонентов среди которых: средства для создания базы данных, средства для работы с базой данных и сервисные средства.

* Средства для создания базы данных:

- Язык запросов (Query Language): Например, SQL (Structured Query Language) позволяет пользователям извлекать данные из базы данных, выполнять аналитические запросы и создавать отчеты.

- Инструменты администрирования баз данных: Эти инструменты позволяют администраторам баз данных создавать новые базы данных, управлять пользователями, настраивать параметры безопасности и мониторить производительность.

2. Средства для работы с базой данных:

Пользовательский интерфейс в прикладных программах,

Визуальные средства и прикладное программное обеспечение для определения таблиц и определения операций над данными

3. Сервисные средства:

- Механизмы безопасности: Эти механизмы позволяют управлять доступом пользователей к данным, обеспечивать конфиденциальность и защищать базу данных от несанкционированного доступа.

- Механизмы резервного копирования и восстановления: Эти механизмы обеспечивают возможность создания резервных копий базы данных и ее восстановления в случае сбоев или потери данных.

В реляционной БД вся информация хранится в **таблицах**, состоящих из **столбцов** и **строк.**

**Столбцы** – это атрибуты или характеристики объекта. Чаще всего хранит информацию об определенном атрибуте записи.

В столбцах должны содержаться данные одного типа данных.

Каждая **строка** хранит данные об отдельном объекте.

Все строки одной таблицы имеют одинаковую структуру и состоят из ячеек, содержащих описание того или иного атрибута объекта.

В качестве примера приведем

Таблицу, хранящую данные об автолюбителях.

Данная таблица имеет следующие **атрибуты** (столбцы):

**имя**: строковый тип,

**фамилия**: строковый тип,

**возраст**: числовой тип,

**профессия**: строковый тип,

**дата покупки**: дата,

**автомобиль**: строковый тип

Добавим данные в таблицу

**Первичный ключ** (PRIMARY key) – уникальный атрибут, идентифицирующий отдельную запись таблицы данных.

Первичные ключи нельзя менять.

Первичным ключом может выступать как **число** так и **строка**.

**Вторичный или внешний ключ** (FOREIGN key) – уникальный атрибут внешней таблицы, создающий связь с данной по совпадающим значениям в столбце.

Важной частью таблицы является **первичный ключ (PRIMARY KEY),** идентифицирующий каждую отдельную строку.

Двух одинаковых идентификаторов быть не может, а сам первичный ключ является отдельным атрибутом у каждой таблицы.

Создадим атрибут ID, который будет являться первичным ключом в таблице с автолюбителями из прошлого примера.

Первичные ключи изменять нельзя, поэтому у объектов всегда будет свой личный идентификатор по которому их легко найти.

Также, несмотря на то, что сейчас мы используем в ID ключа цифры, первичным ключом может быть и строка.

Попробуем соединить две таблицы с помощью ключей.

У автолюбителей есть машины, которые имеют свои характеристики.

Также машины у автолюбителей могут повторяться и чтобы решить эту проблему, мы создадим новую таблицу.

Таблица с машинами будет иметь следующие атрибуты: наименование, которое будет являться первичным ключом, длина, ширина и вес.

Теперь названия машин, расположенные в таблице автолюбителей будут являться вторичными или внешними ключами (FOREIGN key),

создающими связь с первичным ключом из таблицы с машинами.

Таблица с первичным ключом называется Родительской, а со вторичным – дочерней.

Ранее упомянутая **связь** устанавливается между двумя таблицами через первичный и вторичный ключи, но и она бывает разной.

Выделяют следующие свзяи:

* один к одному,
* один ко многим,
* многие ко многим.

Связь **один к одному** подразумевает, что **один** объект (строка) первой таблицы зависит от **одного** объекта второй таблицы и наоборот.

Данная связь встречается редко, поскольку в большинстве случаев можно просто объединить две таблицы и надобность в связи отпадет сама собой.

Например, каждый автомобилист относится к одной единственной профессии, к которой также может относиться лишь один единственный автомобилист.

Но так или иначе мы хотим описать профессию и в отдельной таблице.

Итак перейдем к определению связи один-ко многим

Связь один ко многим – связь при которой одна строка первой таблицы относится к нескольким строкам (нескольким объектам) второй таблицы, а одна строка второй таблицы относится к одной строке (одному объекту) первой.

Связь **один ко многим** мы уже использовали.

В ней одна машина относилась к нескольким автовладельцам, а один автовладелец имел лишь одну машину.

Таким образом, от одного объекта родительской таблицы зависят несколько объектов дочерней таблицы и один объект дочерней таблицы зависит от одного объекта родительской таблицы

Связь многие ко многим звучит следующим образом.

«**Один** объект первой таблицы зависит от **нескольких** объектов второй таблицы и **один** объект второй таблицы зависит от **нескольких** объектов первой таблицы».

Как пример, можно привести машины и создающие их фабрики.

Одна фабрика может делать несколько видов машин, в то время как один вид машин может делать несколько фабрик.

В связи многие ко многим есть одна отличительная особенность: в ней участвует не две, а три таблицы. А именно:

* Две основных
* Одна связующая, которая будет хранить два вторичных ключа от основных таблиц.

Также нет запрета на использование связующей таблицы для хранения данных, относящихся к конкретным машинам и фабрикам.

В использованных ранее рисунках с иллюстрациями связей таблиц мы использовали наглядный инструмент отображения схем таблиц.

**Схема БД** – список таблиц, их атрибутов, типов данных, ограничений, ключей и связей между таблицами,

необходимый для корректной организации хранения данных в памяти вычислительного устройства и доступа к данным извне,

как на запись, так и на чтение.

**Схема БД** является удобным инструментом унификации доступа к данным и помогает ускорять доступ к информации по сравнению с другими моделями хранения данных.

Также корректная схема и организация ограничений и связей таблиц помогают обеспечить высокую отказоустойчивость и целостность хранилища данных на автоматическом уровне.

В системах администрирования реляционных баз данных доступно представление схемы данных в виде концептуальных диаграмм,

что позволяет быстрее создавать и настраивать таблицы в базе данных.

Таким образом реализуется возможность управления базами данных посредством визуального интерфейса.

Таблицы на концептуальной схеме базы данных выражены в форме объектно-ориентированной модели, то есть в формате полей данных, которые содержатся в таблице.

На усмотрение разработчиков системы работы с базой данных предусмотрено также визуальное отображение типов данных для каждого из полей.

Разобранные ранее связи между таблицами отображаются в формате, показанном ранее на примерах связи один-ко-многим и многие-ко-многим.

То есть в случае варианта связи «много», отображается знак бесконечности, в случае варианта «один», отображается единичка.

На слайде показана концептуальная схема базы данных, представленная в системе MS ACESS, для более простой визуализации.

Разберем пример на слайде:

На концептуальной схеме отображены 6 таблиц, относящихся к предметной области принятия контроля в ВУЗе

Первая таблица представляет собой сущность «Студент» и имеет следующие поля:

1. Номер зачетки – первичный ключ, уникальный идентификатор учащегося, шифр

2. Фамилия, имя и отчество – строковый тип,

3. Группа, курс – строковый тип,

4. Фото – тип binary или BLOB.

Вторая таблица это база преподавателя и имеет такие же поля, за исключением отличий:

1. Кафедра, Должность – скорее всего строковый тип или внешний ключ на другую таблицу в базе данных, не указанную здесь

2. Адрес – строковый тип

3. Дата приема на работу в формате DATETIME – дата и время

Третья таблица – таблица предметов, преподаваемых в ВУЗе. Состоит из следующих полей:

* Номер предмета – первичный ключ
* Название предмета – строковый тип
* Количество часов и номер семестра в виде целого числа
* Номер преподавателя, который ведет дисциплину – внешний ключ на вторую таблицу с преподавателями, которые ведут занятия.

Преподаватель, может вести много предметов, поэтому связь реализуется как «один» преподаватель – «много» предметов

Четвертая таблица связывает таблицы предмета и студента посредством связи многие ко многим внешними ключами и она также обладает дополнительные поля:

* Номер ведомости – первичный ключ в виде строки
* Дата приема – DATETIME
* Оценка студенту по предмету – целое число или строка.

Таким образом мы сможем корректно масштабируемым образом на количество предметов в ВУЗе хранить данные ведомостей приема экзамена.

Данная реализация не является эталонной и служит только в качестве примера, однако имеет место быть при разработке подобных систем.

Также важно понимать, что однажды разработанная схема базы данных служит жестким фундаментом для хранения данных в долговременном хранилище данных,

так как в дальнейшем масштабирование данной схемы не представляется легкой задачей как в плане восстановления данных в записях таблиц, так и расширения схемы БД.

На практике при расширении структуры хранимой информации организации вынуждены прибегать к нескольким из возможных сценариев:

Прямое расширение данных с их поиском во внешних источниках.

Проблема здесь в том, что данные в новых столбцах будут пустыми для старых записей что может плохо сказаться на качестве данных в хранилище.

Осознавая риски, данное решение может быть принято и данные с пустыми значениями в новой структуре могут не использоваться в дальнейшей версии расширенного анализа

* Создание новой структуры хранения недостающих данных и их объединение с имеющимися посредством процедур преобразования и трансформации

Здесь мы неосознанно создаем новую схему данных рядом с имеющейся, что плохо влияет на консолидацию и данных способ чаще всего не применяется в реальной практике.

Создание новой базы данных, перенаправление запросов в первую версию базы данных при необходимости архивной сводки.

Похожий вариант был в первом случае, но здесь мы намеренно не вводим новые данные в схему первой «устаревшей» схемы БД.

Создав новую базу данных мы имеем возможность варьировать порядки запросов к текущему состоянию системы и к базе данных, собиравшихся по прошлой схеме полей и атрибутов.

Таким образом, из приведенных выше рассуждений можно сделать вывод, что реляционная модель данных не позволяет гибко хранить табличные данные и побуждает к поиску идеальной структуры хранения данных под вашу задачу с самого начала,.

Данный аспект отражен в очень долгой и кропотливой работе инженеров данных и управляющих проектом над созданием архитектуры хранения данных, необходимых для работы системы обработки больших данных.

Каким образом составлять архитектуру базы данных таким образом, чтобы схема не содержала в себе проблем, связанных с дублированием данных и несогласованной зависимостью.

На этот вопрос отвечают правила нормализации данных

Нормализация — это процесс организации данных в базе данных, Она включает в себя создание таблиц и установление связей между ними в соответствии с правилами, разработанными как для защиты данных, так и для повышения гибкости базы данных, устраняя избыточность и несогласованную зависимость.

Избыточность данных приводит к непродуктивному расходованию свободного места на диске и затрудняет обслуживание баз данных. Например, если данные, хранящиеся в нескольких местах, потребуется изменить, в них придется внести одни и те же изменения во всех этих местах

Всего правила нормализации данных насчитывают порядка шести нормальных форм, первые три из которых необходимо знать даже аналитикам данных

Попробуем разобраться в первых трех правилах нормализации данных:

**Первая нормальная форма таблицы достигается при следующих условиях:**

В каждой ячейке должно быть единственное атомарное значение.

Это значит что в ячейке должно хранится значение из ранее объявленных типов данных в системе хранения структурированных данных или СУБД.

В каждой СУБД мы имеем свою систему базовых типов, расширяющую главный стандарт SQL, и первая нормальная форма базируется именно на базовых типах в конкретной реализации СУБД

Далее

Расположение строк и атрибутов в таблице никак не должно влиять на работу с ними.

То есть записи в таблице данных не должны предусматривать необходимость по-отдельности обрабатывать каким-либо образом каждую отдельную строчку данных.

Это выражается в правилу унифицированности представления объектов через структуру данных.

Строки, как и идентификаторы не должны повторяться.

Таким образом первая нормальная форма накладывает ограничения на избыточность данных с точки зрения повторяющихся записей.

Приведем пример данных, которые необходимо привести к первой нормальной форме

Разберемся с понятием первой нормальной формы на простом примере.

У таблицы «Машины» и «Продавцы» связь многие ко многим, но есть способ убрать связующую таблицу, записав ID всех продавцов в таблицу к машинам через запятую.

Получим общую таблицу, в которой будут записаны как объекты для продажи так и способы достать желанный автомобиль.

Подобный метод может показаться неплохим, но он совсем ненадежен.

Посмотрим на правила нормализации по первой нормальной форме.

В случае, если один из продавцов перестанет работать с определенной маркой машин, нам придется разбивать ячейку с продавцами на составные части, что в разы усложнит работу с данными.

Данная таблица находится на грани первой нормальной формы, ввиду того, что последняя колонка может храниться как в формате списка чисел, так и в формате цельной строки символов.

Первый случай не допустим в реляционной модели данных.

Второй случай позволяет нам оставить данную таблицу и сказать, что она нормализована по первой нормальной форме.

Приведем пример возможного преобразования таблицы к первой нормальной форме.

В результате таблица приведена в вид реализации таблицы промежуточной таблицы в связи многие-ко-многим, т.е. вид таблицы фактов для справочных таблиц.

Продублируем строки в количестве, равном количеству продавцов машин и получим, что.

Строки не повторяются полностью, данные атомарны, а значит таблица находится в первой нормальной форме.

Перейдем к определению второй нормальной формы

Данные в базе данных находятся во второй нормальной форме если соблюдаются два условия:

Данные находятся в первой нормальной форме, а значит соблюдены все три условия которые мы разбирали ранее, а также

Все неключевые атрибуты таблицы полностью зависят от первичного ключа.

Это правило гласит о том, что если какой-либо атрибут, который находится в таблице не зависит от первичного ключа, то его необходимо вынести из таблицы во внешнюю таблицу со связью по внешнему ключу, т.е. «один-ко-многим».

Разберемся с данной формой на следующем примере:

Рассмотрим пример приведения БД ко второй нормальной форме

Пусть у нашей связующей таблицы между Машинами и Продавцами будет атрибут с объемом двигателя.

Отобразим нашу сущность в виде элемента концептуальной схемы БД.

В таком случае, объем двигателя относится только к машине и никак не относится к продавцу

Поэтому ее необходимо перенести в соответствующую таблицу Cars и установить,

что хранить такие зависимые от авто атрибуты мы будем только в данной внешней таблице.

То же самое будет работать и для второй таблицы, необходимо все данные, находящиеся только в зависимости от продавца, хранить во внешней таблице Sellers.

Таким образом наши данные были переведены в уже вторую нормальную форму.

Первые два правила нормализации данных кажутся довольно простыми и естественными с точки хранения данных в таблицах.

Третья нормальная форма очень похожа на вторую нормальную форму, но между ними есть отличие.

Третья нормальная форма достигается в случае,

когда достигнута вторая нормальная форма, т.е. соблюдены уже 4 вложенных условия, которые мы ввели ранее, а также

Все неключевые атрибуты таблицы напрямую не зависят от других неключевых атрибутов этой таблицы.

Данное условие говорит о том, что если среди неключевых атрибутов прослеживается явная зависимость, то данные атрибуты необходимо вынести во внешнюю таблицу с ключом.

Разберемся с данным видом нормальной формы на примере.

Представим новую таблицу с программистами, они пишут код на определенной платформе и языке программирования.

Таблица содержит следующие атрибуты:

1. Уникальный идентификатор: IDprog

2. Имя и фамилию

3. Язык программирования на котором работает программист

4. Тип исполнения команд

5. Возраст программиста

6. Платформа на которой программист разрабатывает решения

У таблицы с программистами есть атрибут со типом исполнения команд в программах (Style), который постоянно меняется вместе с атрибутом языка программирования (Language)

Для того чтобы вместо двух атрибутов менять лишь один, нужно вынести язык и тип исполнения команд в новую таблицу

А так будет выглядеть новая связь:

Для того чтобы вместо двух атрибутов менять лишь один, нужно вынести язык и тип исполнения команд в новую таблицу

А следующим образом будет выглядеть новая связь,

это связь «один-ко-многим», с внешним ключом на новую справочную таблицу

Разобравшись в том, каким образом выглядит реляционная модель данных, мы можем переходить к непосредственному изучению языка SQL со всеми его особенностями

В следующем разделе нашей темы мы поговорим о том, как получить доступ к данным в реляционной базе данных,

а также каким образом таблицы создавать, заполнять и изменять в них значения

На слайде приведена древовидная схема с подмодулями классической реализации стандарта языка SQL

На промежуточных узлах мы видим 4 подмодуля языка для работы с данными, позволяющими:  
  
1. Определять новую базу данных и таблицы, а также изменять их – это язык DDL

2. Изменять наполнение таблиц и обращаться к таблицам для получения данных – язык DML

3. Управлять доступом пользователей базы данных к различным таблицам и возможностям работы с ними – язык DCL

4. Управлять последовательностью операций по изменению базы данных – язык работы с транзакциями TCL.

На данном этапе нас интересует обеспечение доступа к данным, т.е. получение данных из существующей базы данных.

Для данной операции чаще всего аналитики данных применяют команду SELECT.

Данная команда относится к языку манипуляции над данными – DML

Функции языков DML определяются первым словом в предложении (часто называемом запросом), которое почти всегда является глаголом.

В случае с SQL эти глаголы —

«select» («выбрать»),

«insert» («вставить»),

«update» («обновить»),

«delete» («удалить»).

Или ка мы ранее уже упоминали аббревиатура CRUD

Языки DML могут несущественно различаться у различных производителей СУБД.

Получение данных из реляционных таблиц данных зачастую работает по принципу прямого запроса пользователя с помощью команды SELECT в базу данных

Используя директиву SELECT в терминале, подключенному к конкретной базе данных мы имеем возможность запросить у таблицы либо все содержащиеся в ней данные,

Либо совершить запрос данных со специальными параметрами, фильтрами, сортировками и т.д.

На слайде представлены возможные варианты простого использования SELECT в обычных уловиях

1. К примеру нам необходимо получить все строки данных из таблицы Apps в таком случае

Запрос SELECT \* FROM Apps позволит это сделать

2. Если нам нужно подсчитать количество всех строк в Apps, нам поможет агрегационная функция подсчета строк COUNT(),

В результате нам вернется лишь одно число

3. Мы можем выделить в таблице Apps все приложения, которые называются определенным именем с помощью параметра WHERE для проверки условия

В результате запроса SELECT \* FROM Apps WHERE AppName = “MoneyControl” мы получим только те строки которые соответствуют данному условию и т.д.

Каким образом мы можем наладить соединение с базой данных из различных инструментов?

Доступ к данным в **РСУБД** также может осуществляться посредством ODBC (контроллер базы данных) или API (прикладной интерфейс программы).

В прикладных пакетах анализа данных существуют возможности быстрого доступа к данным таблиц базы данных за счет разработанных библиотек, компонентов и утилит.

Данные утилиты позволяют прописывать в настройках несколько параметров для входа:

ip, port, имя пользователя, пароль, база данных

И на основе этих данных, именуемых credentials осуществлять соединение с базой данных

Таким же образом можно настроить соединение с помощью языка программирования на основе либо API к базе данных,

либо с помощью протокола настройки соединения с БД по ODBC

Пример на слайде показывает как в Python с помощью нескольких строк кода возможно получить таблицу из базы данных, хранящейся во вcтроенной базе данных sqlite3.

Для этого производится импорт необходимых программных компонентов (классов и функций) для связи с конкретной базой данных.

В дальнейшем за две-три команды с помощью SELECT можно по настроенному соединению получить таблицу в переменную Python

Приведем пример запроса на выборку данных из ранее показанных таблиц

Покажем запрос SELECT, который позволит выбрать некоторые столбцы из базы данных автолюбителей.

Через запятую указываем интересующие нас столбцы, переносы строки не являются обязательными в запросе.

Этот стиль оформления выбран для читаемости запроса.

Далее указываем источник данных в директиве FROM.

Обратите внимание на то, что выборка данных указывается поколоночно, если мы хотим выбрать какие-то специфицированные столбцы данных.

Получим уже известную выборку данных

Оператор SELECT состоит из нескольких предложений (разделов):

**SELECT** определяет список возвращаемых столбцов (как существующих, так и вычисляемых), их имена, ограничения на уникальность строк в возвращаемом наборе, ограничения на количество строк в возвращаемом наборе;

**FROM** задаёт табличное выражение, которое определяет базовый набор данных для применения операций, определяемых в других предложениях оператора;

**WHERE** задает ограничение на строки табличного выражения из предложения FROM;

**GROUP BY** объединяет ряды, имеющие одинаковое свойство с применением агрегатных функций, производя операцию группировки данных

**HAVING** выбирает среди групп, определённых параметром GROUP BY, значения удовлетворяющие условию, как и Where

**ORDER BY** задает критерии сортировки строк; отсортированные строки передаются в точку вызова.

Таким образом оператор SELECT имеет следующую структуру

Где

[] — необязательное предложение для оператора

{} — выбор из нескольких возможных аргументов предложения

Теперь рассмотрим способы изменять данные в существующих SQL таблицах, а также обсудим механизм транзакций, который позволяет соблюдать согласованность данных в системе хранения

Операторы, отвечающие за внесение изменений в наполнение реляционной базы данных находятся в языке DML.

Оператор INSERT Добавляет новые данные в таблицу.

Сразу же приведем пример команды INSERT.

Пусть мы рассматриваем таблицу данных «Persons» со следующей структурой:

1. ID – первичный ключ

2. Фамилия и Имя как строковые значения

3. А также рабочее подразделение тоже как строковые значения

В данном случае команда INSERT на добавление строк в базу данных выглядит следующим образом.

В запросе видно, что некоторые значения выставлены как NULL, что означает добавления пустых универсальных значений в рамках таблицы данных

Пустые данные не влияют на согласованность нормальных форм, но влияют на чистоту данных.

В результате в таблицу добавятся новые 6 строк данных с новыми экземплярами

Общий вид запроса является довольно простым и не нуждается в детальном разборе.

Перейдем к обновлению данных в реляционной базе данных

Обновление данных в реляционной БД происходит посредством запроса с предикатом UPDATE

Для обновления необходимо указать имя таблицы после UPDATE как и во вставке данных, затем

В директиве SET мы указываем то какой столбец обновить и на какое значение

Нам доступно указать либо конкретное значение, либо математическое выражение, либо NULL, или значение для типа данных или домена по умолчанию.

И после

Для того чтобы выбрать точечное изменение необходимо соблюсти нахождение в запросе оператора WHERE с условным выражением,

явно указывающим строки в которых значения будут претерпевать изменения.

Условие может быть как числовым, на порядок и равенство,

так и строковым на содержание части подстроки, совпадения строк и совпадения шаблонов регулярных выражений

Рассмотрим простой пример с оператором WHERE

Пусть у нас имеется таблица, содержащая ноутбуки и их характеристики:

Таблица с названием Laptop, хранящая данные о:

- Наименовании модели

- Скорости Диска

- Объему ОЗУ

- Объему дискового накопителя

- Цене ноутбука

- диагонали экрана

Тогда у нас имеется возможность обновить данные в данной таблице посредством оператора UPDATE

С оператором WHERE для обновления емкости накопителя на основе значения RAM, где значение емкости накопителя меньше 10, тогда записи с таким условием изменят свои значения в данных полях

Или без оператора WHERE как в примере с обновлением цены ноутбука.

Это не все возможности оператора WHERE но данные примеры покрывают большую часть задач по изменению данных в таблицах.

Также мы имеем возможность быстрого удаления записей по определенному условию из таблицы данных посредством оператора DELETE

Директива работает таким же образом как и UPDATE но без изменения данных, но с их удалением

Все операции по изменению данных в базе данных переводят базу данных из одного состояния в другое, хранящее уже другие данные.

Информация очень важна, иначе этого курса бы и не было, следовательно данные действия в БД необходимо контролировать различными способами

**Транзакция** – это любое изменение или последовательность операций по изменению данных, происходящих в БД. Таким образом

**Приходим к тому, что введенные нами выше операции с базой данных, а именно Добавление, изменение, удаление** данных – все это транзакции.

Перейдем к более формальному определению транзакции в БД

**Транзакция** — группа операторов определения, манипуляции данных, переводящих базу данных из одного согласованного состояния в другое согласованное состояние.

**Транзакция** — группа операторов определения, манипуляции данных, переводящих базу данных из одного согласованного состояния в другое согласованное состояние.

Транзакции сопровождают:

Создание таблиц

Изменения таблиц

Удаление таблиц

Вставку наблюдений (строк)

Изменение наблюдений

И удаление наблюдений, то есть строк данных в таблице

В определении транзакции важным термином было слово согласованное состояние.

Что называется согласованным состоянием в терминологии базы данных?

Требования ACID — набор требований, которые обеспечивают сохранность ваших данных.

ACID это аббревиатура, которая состоит из нескольких важнейших терминов в терминологии транзакций

Транзакции в базе данных должны подчиняться следующим свойствам:

Атомарность - любая транзакция должна быть выполнена полностью, либо не выполнена вовсе.

Например, команда на добавление данных в таблицу не может добавить лишь половину информации и остановиться.

Также существует пример двойной записи в базу данных, например когда происходит банковская транзакция.

Если Запрос A требует изменить данные в записи X, а затем запрос B требует изменить данные в записи Y на основе изменения X, но запрос B провалился с ошибкой, то атомарность транзакции нарушается.  
Два согласованных запроса были выполнены в разных транзакциях БД, но соответствовали одной реальной транзакции.

Решением данной проблемы будет объединение изменений в БД посредством связки двух запросов в единую транзакцию.

Согласованность - **база данных хранит в себе непротиворечивые данные, которые не являются ошибочными с точки зрения реляционной модели**

Таким образом имеется ввиду что при выполнении транзакции в базе данных, а именно действий по изменению схемы или таблицы мы не нарушим нормальные формы базы данных, что может повлечь за собой избыточность и недостоверность данных в БД.

Изолированность - **работа и результат каждой транзакции определяется независимо от других транзакций.**

Или другими словами, если речь идет о возможной параллельной работе транзакций:

Во время выполнения транзакции параллельные транзакции не должны оказывать влияния на её результат.

Если у нас система строго для одного человека, проблем не будет.

А если пользователей несколько?

Тогда транзакции запускают в параллель — для ускорения работы системы.

А иначе представьте себе, что вы делаете заказ в интернет-магазине и система вам говорит: «Вы в очереди, перед вами еще 100 человек хотят заказ оформить, подождите».

Для изолированности существует несколько проблем в случае параллельной обработки запросов:

Потерянная запись возникает в результате быстрого выполнения операций по обработке одной и той же сущности, будь то таблица или строка данных

Если два пользователя пытаются одновременно работать с одной и той же сущностью базы данных, то есть риск, что одно из изменений не достигнет таблицы и данные потеряются.

Данная проблема решается блокировкой значения, в которое пытается записать данные первый из пользователей.

Грязное чтение состоит в том, что при записи данных в БД возникает эффект быстрого сохранения данных и параллельный пользователь может считать промежуточное значение записанных изменений во время транзакции

Пример: Транзакция A читает данные, которые были изменены транзакцией B, но еще не зафиксированы. Если транзакция B откатится, то транзакция A прочитает "грязные" данные.

Повторимое чтение

Причиной повторимого чтения является:

Изменение данных другой транзакцией между повторным чтением одних и тех же данных.

Пример: Транзакция A читает значение X, затем транзакция B изменяет значение X, и транзакция A снова читает значение X, получая другой результат.

Фантомное чтение возникает в результате вставки или удаления данных другой транзакцией между повторным выполнением запроса

Пример: Транзакция A выполняет запрос на выборку определенных строк из таблицы, затем транзакция B вставляет новые строки в эту таблицу, и транзакция A выполняет тот же запрос, получая больше строк, чем ожидалось

Как бороться с этими проблемами?

Нужно изолировать транзакцию.

Способов есть несколько, но основные — блокировки и версии.

Блокировки — это когда мы блокируем данные в базе.

Версии – это резервное копирование данных, которое скрыто от разработчика.

Можно заблокировать одну строку в таблице, а можно всю таблицу.

Можно заблокировать данные на редактирование, а можно и на чтение тоже.

Базово все СУБД решения должны поддерживать разумный уровень блокировок транзакций на уровне приведенных выше проблем.

**Долговечность – гарантирует, что результат транзакции будет сохранен, даже если произойдет ошибка или сбой системы**

Если пользователь получил подтверждение от системы, что транзакция выполнена, он может быть уверен, что сделанные им изменения не будут отменены из-за какого-либо сбоя.

Обесточилась система, произошел сбой в оборудовании?

На выполненную транзакцию это не повлияет.

Таким образом мы объяснили принципы ACID которым должны следовать как транзакции, написанные разработчиком,

так и система хранения структурированных данных по мере своих технических возможностей.

Аббревиатура TCL расшифровывается как **Transaction Control Language** in SQL. Transaction Control Language (TCL) — это набор специальных команд, которые **работают с транзакциями** в базе данных.

Команды TCL также используются для **поддержания согласованности базы данных.**

В команды TCL входят:

Commit - Команда COMMIT в SQL используется для **постоянного сохранения любой транзакции в базе данных**. Как правило, всякий раз, когда мы используем какую-либо команду DML, такую как INSERT, UPDATE или DELETE, изменения, внесенные этими командами, не являются **постоянными**. Следовательно, перед закрытием текущего сеанса мы можем откатить любые изменения, сделанные с помощью этих команд.

По вышеуказанной причине необходимо использовать **команду COMMIT**, чтобы пометить любые внесенные изменения как необратимые.

СУБД чаще всего по умолчанию сохраняют изменения в базе данных при выполнении команд в диалоговом режиме, чтобы в SQL скриптах команды выполнялись слитно необходимо использовать TCL

После написания нашего запроса мы пишем "COMMIT", чтобы сохранить и сделать постоянными все изменения, которые сделаны нашими DML-командами в базе данных.

Кроме того, обратите внимание, что после того, как мы сделали COMMIT, мы не можем его отменить. Тем не менее, мы можем выполнить откат, но выполнение отката отменит всю транзакцию.

**Основные моменты:**

* COMMIT сохранит все изменения (все транзакции), сделанные командами DML в базе данных.
* Мы должны написать команду COMMIT до и после каждой команды DDL, чтобы навсегда сохранить изменения в базе данных.
* После выполнения COMMIT его нельзя отменить, пока не будет выполнен откат.

Как мы уже знаем, команда COMMIT используется в TCL для того, чтобы сделать любые изменения в базе данных постоянными;

Рассмотрим пример использования commit в нашем коде.

Например, предположим, что мы хотим изменить название страны для одной из наших таблиц, скажем в таблице "Клиенты".

Теперь предположим, что мы хотим изменить название страны для конкретного клиента, а затем зафиксировать транзакцию. Для этого мы напишем следующий фрагмент кода:

После выполнения приведенной выше команды транзакция обновления завершается командой commit, как описано выше.

И благодаря приведенным выше условиям, вместо имени клиента «Давид» будет установлено в «ИНДИЯ», и это изменение будет сохранено в базе данных навсегда.

Теперь, если мы захотим снова посмотреть на таблицу нашего клиента, она будет отображаться следующим образом:

Приведенный выше вывод является нашей обновленной таблицей после выполнения запроса "update".

Вы можете заметить, что название страны для клиента с именем «Дэвид» изменено на «Индия».

Команда rollback в TCL используется для **восстановления базы данных до последнего зафиксированного состояния**.

Другими словами, команда rollback восстанавливает базу данных в исходное состояние с момента последней фиксации с помощью commit.

**Например**, предположим, что мы использовали команду UPDATE в любой момент для внесения определенных изменений в нашу базу данных,

а затем поняли, что эти изменения должны быть отменены,

в этом случае мы можем использовать команду ROLLBACK.

Команда rollback в основном откатывает или отменяет любые изменения,

которые не были зафиксированы во время нашей транзакции с помощью команды COMMIT.

После написания запроса мы можем написать "ROLLBACK;",

чтобы откатить или отменить группу транзакций с момента последнего COMMIT.

**Основные моменты:**

* ROLLBACK используется для отмены транзакций, которые еще не были окончательно сохранены (или зафиксированы) в базе данных.
* Команда ROLLBACK восстанавливает ранее сохраненное значение, то есть данные, присутствующие до выполнения транзакций.

Как мы узнали выше, команда rollback обычно используется для отмены любых изменений, которые мы внесли до нашего последнего коммита;

Давайте теперь посмотрим на пример использования команды rollback.

Предположим, что в таблице наших клиентов при выполнении каких-то операций мы по ошибке обновили имя клиента как "JOHN",

тогда как должно быть "PETER".

Пусть перед откатом у нас есть следующая таблица, получившаяся в результате обновления данных с кодом ниже

Следовательно, мы можем откатить нашу транзакцию к исходному состоянию.

Нам необходимо выполнить ROLLBACK до того, как мы зафиксируем изменения данных с помощью COMMIT

Команда SAVEPOINT в TCL в основном используется для временного сохранения транзакции, чтобы мы могли откатиться к этой точке (сохраненной точке) при необходимости.

**Основные моменты:**

* SAVEPOINT используется для создания точки в группах транзакций для последующего сохранения или отката.
* SAVEPOINT очень выгоден, когда мы хотим откатить транзакции до определенной точки, не откатывая всю группу транзакций.

После написания запроса мы можем написать "SAVEPOINT ;", а затем имя savepoint\_name точки сохранения, чтобы создать точку сохранения

Кроме того, мы можем освободить точку сохранения, то есть удалить любую конкретную точку сохранения c помощью RELEASE SAVEPOINT

Давайте посмотрим на пример, где мы используем все эти 3 команды TCL вместе.

Предположим, что у нас есть некоторая таблица **Employee**, как показано на слайде, которая состоит всего лишь из уникального ключа и имени работника

Давайте выполним несколько SQL-запросов к приведенной выше таблице, используя команды Commit, Rollback и Savepoint, и посмотрим на результаты:

Итак, в приведенном выше примере мы написали кучу кода, добавив некоторые значения в нашу таблицу, а также создав несколько точек сохранения, которые будут сохранять наши данные до этого момента.

После каждой транзакции по изменению таблицы мы ставили точки сохранения с различными метками, что позволит нам вернуться к данным точкам сохранения.

Во всех кроме первой, где уже произошла фиксация с помощью commit

В результате наша таблица увеличится в количестве записей

После того, как мы это сделали, теперь давайте воспользуемся командой **ROLLBACK** для отката данных до точки сохранения B. Код для этого будет следующим:

Итак, вы можете видеть в приведенном на слайде выводе, что мы вернулись к нашим данным до точки сохранения B.

Потому что после точки сохранения B мы вставили еще одно значение с идентификатором emp 115.

Но, так как мы откатились до точки сохранения B, данные с идентификатором emp 115 не отражаются.

После этого воспользуемся командой **ROLLBACK** для отката данных до **точки сохранения A**. Код для этого будет следующим:

В приведенном выше примере мы выполнили откат к нашей последней точке сохранения, то есть A.

Таким образом на примерах мы разобрались с возможностями проведения транзакций в языке SQL

Приветствую вас уважаемые учащиеся и коллеги,

Мы продолжаем разбираться с основами больших данных на нашем курсе.

Данная тема посвящена технологиям вычислений в базах данных, обработкам пропусков, фильтрациям, агрегациям и преобразованию данных.

Внутри у нас темы по операциям обработки структурированных данных

Шкалы данных и возможная обработка данных в шкалах с точки зрения аналитика

Процедуры очистки данных и в целом очистка данных

Фильтрация и сортировка данных для приведения таблицы к желаемым подвыборкам

Группировка или агрегация данных в больших массивах

Главной темой нашей сегодняшней лекции будет обработка табличных данных.

Так что будем разбираться в том, что же подразумевается под данными операциями

Начнем наш раздел с определения трансформации данных

Трансформация данных - это выполнение различных преобразований данных с целью их подготовки к анализу или моделированию.

Среди наиболее часто используемых операций - отбор строк или столбцов таблицы данных, вычисление новых столбцов, подсчет итогов, группировка и ранжирование.

Трансформация необходима в конечном счете для решения конечной задачи получения пользы из данных.

Существует базовый перечень методов обработки данных, относящихся к трансформации таблиц:

Выборка данных (выборка столбцов или атрибутов)

Сортировка данных – упорядочивание записей в таблице по определенному показателю

Фильтрация данных – отбор данных по определенному совпадающему значению

Вычисления новых столбцов на основе имеющихся для подсчета итогов в группах или в целом на значениях

Группировка данных, которая подразумевает вычисление обобщающего значения для выборки строк по определенному показателю на основе агрегационной функции

Обогащение данных, состоящее в соединении таблиц посредством ключей или соединение таблиц таблиц друг рядом с другом

Транспонирование данных, состоящее в процедуре приведения повторяющихся меток внешних ключей в разряд столбцов (атрибутов), сопоставляя им данные по определенному правилу

Трансформация данных заключается в оптимизации их представлений и форматов с точки зрения решаемых задач и целей анализа.

Трансформация не ставит целью изменить информационное содержание данных.

Ее задача — представить эту информацию в таком виде, чтобы она могла быть использована наиболее эффективно.

Вообще, трансформация данных — очень широкое понятие, не имеющее четко очерченных границ.

В различных направлениях обработки данных этот термин иногда распространяют на любые манипуляции с данными, независимо от их целей и методов.

Однако в контексте аналитики, трансформация данных имеет вполне конкретные цели и задачи, а также использует достаточно стабильный набор методов.

В результате применения методов трансформации данных таблица приобретает новую структуру, или меняет порядок и состав объектов в таблице.

В **OLTP системах** (системы оперативной обработки данных) трансформация занимается обеспечением поддержки корректности форматов и типов данных,

а также оптимизацией процессов доступа и выгрузки данных.

На этапе **ETL-процесса** трансформация производится с целью приведения данных в соответствие с моделью, которая используется в хранилище,

а также с целью обеспечения процесса консолидации данных и их загрузки в хранилище.

В **аналитическом приложении** производится

- непосредственная подготовка данных к анализу,

- объединение и выделение наиболее ценной информации,

- обеспечение корректной работы аналитических алгоритмов, методов и моделей.

Методы трансформации данных активно применяются на стадиях:

очистка и подготовка данных перед загрузкой в хранилище данных  
На данной стадии применяют в основном методы фильтрации данных и вычисления над данными, в результате чего возможно выделение проблемных мест и локализация проблемы с пропусками и невалидными значениями

организация витрин данных на стадии подготовки схем витрин данных для разных отделов разработки

оптимизация данных для моделирования

организации проведения аналитических отчетностей

На самом деле, в реальных приложениях, картина немного другая.

Большая часть данных, поступающих на конвейер обработки данных представляет собой информацию,

полученную посредством использования внешнего API сервера организации или сайта крупной компании, датчиков интернета вещей, данные прихода пользователя и т.д.   
Все эти данные представляют собой, в основном, формат ключ-значение,

то есть файлы JSON, YAML или XML формата, которые в общем случае не являются структурированными.

В результате мы приходим к необходимости все эти данные структурировать с помощью алгоритмов преобразования или структуризации данных, выделяя структуру записей и сканируя файл, записывая данные в таблицу.  
После этого мы применяем методы трансформации данных.

Важным в обработке данных является физическое соответствие операций над данными с возможными по логике операциями над ними. Метаданные измерений, хранящихся в структурированных таблицах данных предписывают определенную природу данных, в соответствии с которой невозможно над определенными измерениями производить некоторые арифметические операции, хоть это и является технически верным решением.

Термином для такого соответствия является шкала данных

Шкала измерения в статистике — это способ представления переменных и их группировки в различные категории.

Она определяет характер значений, присвоенных переменным в наборе данных.

Типы шкал данных, которые определяются в статистике:

**Номинальная** и **порядковая** шкалы — измерение качественных данных (категории).

**Интервальная** и шкала **отношений —** измерение количественных данных.

Шкалы измерений в свою очередь обладают определенными свойствами, среди которых:

Идентифицируемость:

Это свойство шкалы измерений, которое означает, что каждый элемент или объект, находящийся на этой шкале, может быть однозначно идентифицирован.

Например, если мы измеряем рост людей в сантиметрах, каждый человек имеет уникальное значение на этой шкале и может быть однозначно идентифицирован по своему росту.

Величина:

Это свойство шкалы, которое позволяет сравнивать объекты и устанавливать отношения между ними на основе их значений на этой шкале.

То есть на множестве значений шкалы мы имеем возможность установить отношение порядка.

Например, если мы измеряем температуру в градусах Цельсия, мы можем сказать, что 20 градусов тепла в два раза больше, чем 10 градусов тепла.

Равенство интервалов:

Это свойство означает, что разница между любыми двумя значениями на шкале одинакова в терминах измеряемой характеристики.

Например, если мы измеряем возраст в годах, разница между 10 и 20 годами равна разнице между 30 и 40 годами.

Абсолютный ноль:

Это свойство шкалы, которое означает наличие точки, которая обозначает отсутствие измеряемой характеристики.

Например, абсолютный ноль для температуры в градусах Цельсия означает полное отсутствие тепла, что соответствует -273,15 градусам Цельсия.

**В Номинальную шкалу входят:** описания групп статистик, подписи визуализации.

Значения из номинальной шкалы отражают те или иные свойства объекта, выраженные словесно.

Их элементы могут только совпадать или не совпадать друг другом, Их нельзя сопоставлять по принципу «больше-меньше».

Недопустимы также и арифметические действия.

Характерным примером может служить группа крови, сорт мороженного, тип операционной системы и т.д.

Мерой среднего может служить **мода**. Мода – значение с самым максимальным употреблением на множестве.

Порядковую шкалу данных наполняют описательные значения, на которых можно поставить отношение «больше-меньше».

Порядковая шкала поддерживает расчет квантилей, а также позволяет исследовать градации оценки качества.

По ней можно ранжировать и сравнивать объекты, по какому — либо признаку.

Мерой среднего может служить медиана

Примером порядковой шкалы могут служить места, занятые на соревновании, или первичный ключ в структурированной таблице данных. На данных примерах мы можем установить соответствие «больше-меньше» или «раньше-позже».

Интервальная шкала состоит из числовых значений, для которых имеется возможность совершать базовые арифметические операции, не противоречащие смыслу измеряемых значений

Среди операций интервальная шкала поддерживает сравнение с эталоном, линейное преобразование (сдвиг), сложение и вычитание.

Является метрической шкалой, то есть данная шкала является представителем шкал, значениями которых можно измерять физические явления.

Мерой среднего может являться среднее арифметическое.

Пример: измерение времени, широта и долгота.

**Для шкалы отношений** присутствует дополнительное свойство — естественное и однозначное присутствие нулевой точки

Точкой начала отсчета является точка, в которой значение параметра равно нулю.

Появляется возможность отсчитывать от нее абсолютное значение параметра, определять разницы значений и во сколько раз одно больше другого.

Присутствуют операции сложения, вычитания, умножения, деления и наличие абсолютного нуля.

Примерами шкал отношений могут быть: измерение температуры, расстояния, массы тела.

Таким образом, на основе свойств шкал измерений мы имеем возможность ввести классификацию типов шкал на основе присущих им свойств, которые заранее определяют перечень возможных операций над ними

Получаем, что номинальная шкала обладает только свойством идентифицируемости

Порядковая шкала обладает свойствами идентифицируемости и величины

Интервальная шкала не обладает только свойством наличия абсолютного нуля

Шкала отношений обладает всеми перечисленными свойствами

Также в обработке данных вводится технически более понятное варьирование переменных по их характеру.

По характеру варьирования переменные делятся на дискретные и непрерывные.

**Дискретные данные** являются значениями признака, общее число которых конечной или бесконечно, но может быть подсчитано при помощи натруальных чисел.

С дискретными данными не могут быть произведены никакие арифметические действия, либо они не имеют смысла.

Дискретными данными являются все данные строкового и бинарного типа, а также некоторые числовые данные и данные даты и времени.

Примеры: код товара, образование, город, тип скидки, пол, категория.

**Непрерывные данные** – это данные, которые могут принимать любые значения в некотором интервале.

Над непрерывными данными можно производить арифметические операции: сложение и вычитание и они имеют смысл.

**Примеры**: возраст, рост, стоимость, количество.

В следующей серии слайдов разберем методы и подходы к очистке данных, а также поймем для чего необходима очистка данных и по какой причине это является важным

**Грязными** данными называют данные, которые содержат ошибки, пропуски, дубликаты, несогласованность в формате или некорректную информацию.

Они могут привести к неверным результатам при анализе и искажению выводов.

**Грязные данные** - это неверные, недостаточные, не несущие никакой пользы. К таковым относится информация, представленная в некорректном формате или несоответствующая критериям. Они появились вместе с системой ввода данных.

Причиной их появления может быть что угодно:

Ошибка во время ввода;

Если чтение в базу данных происходит из форм сбора данных, например анкетированный опрос, то данные могут быть искажены на стороне адресанта,

В данном случае может быть необходимо вводить систему проверок на уровне ввода определенных атрибутов, критичных для аналитики данных.

Противоречие критериям;

Если данные, которые вводятся внешним пользователем не проверяются, но удовлетворяют системе проверок, и физический смысл измерений установлен,

То имеет место быть наличие выбросов или экстремальных значений в записи.

Отсутствие оперативного обновления;

В случае оперативного ввода данных в составную таблицу, при приходе новых данных лишь из одного источника происходит запись неполной информации в строке,  
То необходимо проводить процедуры синхронизации поступления информации в транзакции, чтобы избежать необходимости обновлять уже введенную запись, в противном случае могут возникать грязные данные.

Неправильное обновление копий данных;

Чтобы очистить данные, нужно проводить процедуры предобработки данных, которые включают действия по выявлению, исправлению и удалению грязных данных.

**Чистые данные** представляют собой табличный набор наблюдений в котором каждой строке данных соответствует полный перечень атрибутов c адекватными значениями.

Пропуски в данных и невалидные данные не являются допустимым сценарием для качественной обработки данных.

**Грязные данные** же являются антиподом чистых данных.

Грязные данные – табличный набор наблюдений, подверженный пропускам и искаженям.

Адекватность данных измеряется шкалами измерений.

Требования первых трех нормальных форм предписывают структурированным данным, следующие три свойства:

Первое, чтобы внутри колонок находились значения,

все соответствующие своим типам данных,

в таком случае произойдет соответствие объектной структуре реляционной модели данных

Второе, чтобы наблюдения полностью покрывали весь диапазон столбцов, то есть все значения в полях записи были заполнены что также отвечает реляционной модели данных

И третье, чтобы значения в ячейках были атомарными, что предписывает первая нормальная форма из нормализации таблиц данных.

**Профайлинг данных** – процесс изучения данных с целью достижения понимания их структуры, содержимого и оценки качества.

Профайлинг данных является составной частью общего процесса по аудиту бизнес-процесса, в котором производится мониторинг систем принятия решений компании на основе данных.

Профайлингом данных в компании занимаются несколько сотрудников.

Инженер данных на этапе загрузки данных в хранилище регулярно должен проверять требования по качеству данных для обеспечения отсутствия грязных данных, насколько это возможно.

Аналитик данных, на этапе просмотра витрины данных должен проверять выборку данных на соответствие решаемой задачи и измерять необходимые метрики качества данных перед проведением анализа

Профайлинг данных может выполняться в разных режимах, будь то ручной выборочный процесс, или же процесс автоматизированный с регулярной выгрузкой отчетности или фиксации в операционном дашборде.

Поскольку источники данных имеют особенность регулярно ломаться, изменять правила доставки и формы предоставления данных,

то проведение регулярного профайлинга данных является чаще необходимостью чем просто рекомендацией

**Профайлинг данных включает в себя следующие этапы**:

Подведение общих описательных статистик по выборке.

В расчет данных статистик входят различные агрегаты по данным, например

- диапазон уникальных значений измерения атрибута,

- количество уникальных значений,

- минимум,

- максимум,

- среднее

- и квантили,

- а также построение тестовых гистограмм распределений данных в столбцах.

Это необходимо для верификации адекватности данных решаемой бизнес-задаче.

В случае обнаружения неточностей на данной стадии, аналитик отражает данные неточности в отчете о качестве данных.

Обнаружение пропусков.

Пропуски подсчитываются и количество пропущенных значений в столбце таблицы данных будет в дальнейшем играть роль в отборе признаков для анализа

Обнаружение выбросов и экстремальных значений.

Производится на основе статистических методов исследования распределений, или на основе квантилей.

Различные методы обнаружения выбросов приводят к разным результатам, в результате чего данный аспект профайлинга данных остается на плечах специалиста предметной области

Обнаружение дубликатов и противоречий.

Производится на основе NP-полных алгоритмов перебора и крайне нежелательны для проверки на всем диапазоне данных как долговременного хранилища данных, так и витрин данных.

Требования к нормализации таблиц данных приводят к отсутствию возможности дублирования данных,

а также ввода противоречивых данных, поскольку они будут отсеиваться уже на этапе транзакций в базу данных.

Поэтому данный этап необходим только в случае получения данных из внешних источников.

Сложные проверки, проводимые вручную

По результатам профайлинга данных аналитик составляет отчет о качестве данных с целью выявления критических проблем с системой сбора,

преобразования и загрузки данных в долговременное хранилище.

Составление отчета о качестве данных является многоступенчатым процессом и включает в себя следующие стадии:

Отчет по данным характеристик полей, который содержит данные об:

1. Имени поля и его физическом смысле измерений,

2. Типе собираемых данных,

3. Виде данных,

4. Количестве пропусков,

5. Количестве выбросов,

Собранные метрики позволят поверхностно оценить качество данных на уровне атрибутов таблицы

Далее идет составление отчета по характеристикам набора данных как множества записей, измеряются:

- Количество строк данных и полей в выборке,

- Количество строк и полей с пропусками, что позволит судить о процентном соотношении чистоты данных к общему числу наблюдений.

В результате аналитик выносит вердикт о пригодности данных в рамках возможности их использования в аналитике данных.

Одной из причин снижения качества данных – пропуски

Пропуски вызывают проблемы при анализе данных.

Пропуски обусловлены плохим сбором данных либо в процессе опроса, либо в процессе автоматизированного сбора данных с форм или датчиков IOT.

Запись в базу данных грязных данных, которые не проходят какие-либо проверки перед поступлением необходимо в системах с быстрым откликом.

Для систем в которых полнота данных является критичной возникают проблемы анализа и предобработки пропусков

Различают два разных способа обработки данных:

Первый способ - Построчный анализ.

Обработка пропущенных данных вручную с использованием инструментов работы с данными

Каждая запись обрабатывается отдельно и построчно во избежание случаев неадекватной обработки.

Такая обработка важна в сферах, где происходит работа с Персональными данными и транзакционными данными.

Пример. Разбор адресов. В таких данных в каждой строке могут содержаться независимые данные требующие индивидуальной обработки.

Второй способ - Моделирование.

В задачах моделирования допускается построение зависимостей только с полными данными о параметрах.

В этом случае неполные строки либо удаляются из рассмотрения, либо данные в них восстанавливаются посредством механизмов,

так же моделирования или заполнения пропусков.

По результатам отчета о качестве данных выносится решение о принятой стратегии борьбы с пропусками,

Рекомендации по борьбе с пропусками в данных могут быть следующими:

Если число пропусков в данных насчитывает до 1%, то примеры для анализа можно удалить

В случае незначительного количества пропусков рекомендуется применять ранее оговоренные механизмы и методы восстановления пропущенных значений

В случае среднего и большого количества пропусков, модель или анализ проведенный по данным может стать неадекватным предметной области,

что может повлечь за собой ошибки при принятии решений на основе данных

При очень большом количестве пропусков лучше отказаться от анализа набора данных вообще

В данных могут возникнуть помимо пропусков также выбросы и экстремальные значения.

Выбросы и экстремальные значения - это аномальные или необычные значения в данных, которые могут искажать результаты анализа или моделирования.

Они могут возникать из-за ошибок в данных, случайных событий или систематических причин.

Выбросы представляют собой высокие значения, которые являются адекватными для измерения данных в пределах атрибута.

Экстремальные же значения это значения, выходящие за пределы физической адекватности измерений.

Наличие выброса в выборке данных может быть констатировано с помощью эксперта предметной области или на основе документации к данным измерений.

Примером выброса может служить ситуация на заправке, когда легковая машина заправляется на 65 литров бензина при среднем показателе заправки в 25.

Это не является неадекватным значением ввиду наличия возможности запасти топливо сверх меры в дополнительных тарах.

А вот значение заправки легкового авто в 800 литров в ученой системе явно является экстремальным или аномальным для предметной области.

Чаще всего именно эксперт в предметной области выставляет физические ограничения по диапазонам адекватных измерений в атрибуте, либо в его связке с другим атрибутом.

Например все же заправка грузового авто на 100 литров не будет являться экстремальным значением в транзакции по точке АЗС.

Различают следующие способы идентификации выбросов и экстремальных значений:

Визуальный анализ: графическое представление данных с помощью диаграмм разброса (scatter plots), ящиков с усами (box plots) и гистограмм.

Именно на основе изображенного на слайде графика ящик с усами чаще всего и принимаются решения о выбросах и экстремумах в выборках данных

Видно что при составлении графика, большая часть данных находится внутри ящика, отображая гистограмму наблюдений в виде коробочки.

О графиках подробнее мы поговорим в следующих лекциях

Статистические методы: использование статистических критериев для определения выбросов, таких как Z-оценка, квартили или межквартильный размах.

В данных методах основой является предположение о том, что данные могут быть распределены согласно нормальному закону распределения случайной величины.

В случае если это так, для данных рассчитывается средний квадрат отклонения, хорошо аппроксимирующий стандартное отклонение распределения,

что позволяет выставить нужный диапазон принятия решения о признании определенной записи выбросом или экстремумом.

На графике можем видеть, что чаще всего для выбросов принимается значение в 3 стандартных отклонения и выше, для экстремумов от 4 стандартных отклонений и выше.

Не все величины, которые мы можем реально измерить подчиняются нормальному закону.

Поэтому применимость метода на основе Z-оценки меньше чем у предыдущего метода с ящиком, ввиду того что предыдущий метод основан именно на понятиях квантилей.

Машинное обучение: применение алгоритмов машинного обучения, таких как методы кластеризации или классификации, для выявления выбросов.

Способы обработки ситуаций с выбросами и экстремальными значениями:

Удаление аномальных значений из набора данных.

Этот метод может быть применен, если выбросы являются результатом ошибок в данных или являются незначительными.

Замена аномальных значений на более типичные или средние значения.

Применяется если необходимо обучать модели на основе машинного обучения и количество выбросов и экстремумов невелико.

Использование специализированных моделей: использование моделей, которые устойчивы к выбросам, таких как робастные методы регрессии, например метод медианной регрессии или оценки параметров.

Интерпретация выбросов: иногда выбросы могут содержать ценную информацию, поэтому их следует тщательно изучить и проанализировать.

Перейдем к сортировке и фильтрации данных как к самым базовым методам преобразования данных

Начнем с сортировки

**Сортировка табличных данных** – преобразование, упорядочивающее набор объектов (строк) или наблюдений в связи с правилом упорядочивания по выбранным атрибутам.

Сортировка таблицы производит упорядочивание строк по соответствующим значениям в сортируемом столбце.

Для одинаковых значений упорядочивание локально не происходит и порядок внутри данных с одинаковыми значениями в сортируемом столбце остается исходным.

Для числовых данных и данных даты и времени упорядочивание данных происходит в естественном порядке по возрастанию или убыванию (на выбор).

Для порядковых данных упорядочивание логически происходит по порядку определенного уровня фактора (первый, второй, третий, ...).

Для строковых данных упорядочивание осуществляется по лексикографическому правилу в порядке следования символов в кодовой странице символов (ASCII, UTF-8, ...).

Лексикографический порядок предписывает сравнивать строки сначала по первым символам строк, затем по вторым,

пока не установится лексикографический порядок следования записей в строках по убыванию или возрастанию соответственно

Сортировка по нескольким выбранным атрибутам определяется также лексикографическим порядком в соответствии с порядком выбранных атрибутов в сортировке.

Сначала данные сортируются по первому атрибуту, затем внутри строк с одинаковыми значениями порядок разрешается за счет сортировки уже по второму атрибуту и т.д.

На примере видим, что сначала в данных происходит сортировка по столбцу числа цилиндров по возрастанию значений,

в последствии внутри отсортированных по лексикографическому порядку группах выстраивается сортировка значений водоизмещения внутри групп с одинаковыми числами цилиндров.

Таким образом данные в столбце водоизмещения были отсортированы, но по тройкам с перепадами.

Пример на языке SQL по сортировке таблицы **машины** (кириллица для простоты).

SELECT

марка машины,

миль на галлон топлива,

число циллиндров,

водоизмещение,

лошадиные силы

FROM машины

ORDER BY

число циллиндров ASC,

водоизмещение DESC;

**Сортировка** необходима для:

Визуализации данных, столбчатых диаграмм, чартов и других данных категорий или описаний

Вычисления определенных статистических процедур (ABC-анализ, XYZ-анализ)

Упорядочивания данных для обеспечения логической адекватности набора данных, если данные собираются не синхронно

Теперь перейдем к фильтрации данных измерений

Фильтрация данных — операция выборки строк (объектов) или наблюдений из таблицы данных в соответствии с логическим правилом сравнения значений выбранного атрибута с определенным значением.

Среди условий фильтрации доступны как простые сравнения чисел с числами и строк со строками,

так и некоторые условия на нахождение значения в перечне или интервале.

В качестве примера фильтрации можно привести операцию WHERE в SQL

На слайде представлен пример выборки данных из таблицы Workers с атрибутами: ID, зарплаты, профессия

Выборке данных подлежат записи, в которых условием фильтрации была заработная плата меньше 50 тысяч

Результатом такой выборки служит, соответственно таблица слева.

Для фильтрации данных на поиск строк возможна постановка следующих условий:

Сравнения

>=, больше равно

>, строго больше

= (==), равно

!=, не равно

<, строго меньше

<=; меньше или равно

Проверка на нахождение значений

В интервале,

вне интервала,

в полуинтервале,

вне полуинтвервала (сложные фильтры по одному столбцу);

То есть принимаются во внимание составные сравнения, указаные ранее.

Для строк и дискретных значений доступны сравнения в списке и вне списка,

когда проверяемое значение проверяется на схожесть хотя бы с одним значением из предложенного списка поиска.

Содержит или не содержит выбранное поле подстроку заданную с помощью регулярного выражения

А также возможно составление сложных условий на фильтрацию данных с применением алгебры логики с И и ИЛИ и НЕ

В качестве примера приведем выборку проверки со сложным условием и регулярным выражением над строкой PersonID на содержание указанного шаблона

Показанный шаблон велит выбрать записи таблицы данных, где ключ содержит подстроку AD8 или AD9 слитно.

Результатом также служит поставленная рядом таблица с меньшим числом записей.

Пример на языке SQL по фильтрации таблицы users, пример которой мы приводили выше выглядит следующим образом.

SELECT

PersonID,

Salary,

Profession

FROM users

WHERE

Salary < 55000 AND

PersonID LIKE “AD[89]\_%”;

Примеры применения фильтрации наблюдений:

Выборка актуальных наблюдений по временному периоду

Выборка данных с заранее заданными значениями категориальных атрибутов или столбцов, а также возможна выборка строк с пропущенными значениями атрибута

Выборка данных с определенными номерами строк индексированной строковой таблицы данных

**Что же такое регулярные выражения?**

**Регулярные выражения** — язык построения шаблонов для поиска подстрок в строковых данных.

Регулярные выражения строятся по принципу написания шаблона, который будет находить желаемую подстроку внутри строки.

При нахождении шаблона внутри значения конкретной строки может быть выбран один из способов обработки строк:

1. сравнение строки с шаблоном,

2. поиск места подстроки в шаблоне,

3. извлечение подстроки из строки по совпадающему шаблону,

4. разделение строки на подстроки по шаблону.

Пример сложного регулярного выражения для декодирования и поиска email адреса в тексте показан на слайде

Для создания регулярного выражения необходимо использовать определенный синтаксис: специальные символы и конструкцию.

* **Указатели** ^, $

^ — начало текстовой строки

$ — окончание текстовой строки

* **Метасимволы** ., |, \

. — любой отдельный символ, кроме новой строки

| — оператор «или»

\ — указатель на то, что следующий символ является литералом

Основные классы символов:

* **Классы символов** [.], [^.], [:alpha:], ...

[...] — любой символ из набора символов ...

[^...] — любой символ, не входящий в набор символов ...

[:alnum:] — буквенно-цифровые символы (a–я, A–Я или 0–9.)

[:alpha:] — буквенные символы (a–я или A–Я.)

[:digit:] — цифры (0-9)

[:punct:] — знаки пунктуации (! " # $ % & ' ( ) \* + , \ -. / : ; < = > ? @ [ ] ^ \_ ` { | })

[:print:] — отображаемые символы и пробелы

[:space:] — пробелы, табуляция и разрывы строк

[:word:] — буквы, цифры и знаки подчеркивания (а–я, А–Я, 0–9 или \_)

* **Классы условных символов**\w, \s, \d, \W, \S, \D

\w — эквивалент [[:word:]], буквы, цифры и знаки подчеркивания (а–я, А–Я, 0–9 или \_)

\s — эквивалент [[:space:]], пробелы, табуляция и разрывы строк

\d — эквивалент [[:digit:]], цифры (0-9)

\W — эквивалент [^[:word:]]

\S — эквивалент [^[:space:]]

\D — эквивалент [^[:digit:]]

* **Группа** (...)

(...) — объединяет элементы выражения, которые могут быть связаны метасимволом «|» ИЛИ

* **Квантификаторы** {n}, {n, m}, ?

{n} — поиск n вхождений предыдущего выражения

{n, m} — Поиск не менее n и не более m вхождений предыдущего выражения

? — Указывает на то, что предыдущий символ или выражение могут входить в строку 0 или 1 раз.

\* — Указывает на то, что предыдущий символ или выражение могут входить в строку 0 или более раз.

+ — Указывает на то, что предыдущий символ или выражение могут входить в строку 1 или более раз.

Для выполнения анализа данных часто является необходимым вычислить новый столбец на основе исходных при помощи математических выражений или арифметических операций.

Операция вычисления новых столбцов позволяет на основе существующих столбцов поэлементыми операциями вычислять новый столбец с таким же количеством элементов, сколько и строк в исходном наборе данных

Вычисление новых столбцов расширяет исходную структуру таблицы и является продуктом анализа. Данный столбец не является необходимым для хранения и важен при анализе и обработке данных.

Вычисления происходят поэлементно, что позволяет как быстро вычислять новые столбцы на процессоре, так и эффективно распараллеливать вычисления на различных исполнителях.

На слайде представлен качественный пример преобразования важной информации даты последнего визита в количество дней, прошедших между заданной датой и датой последнего визита, на основе чего высчитывается новый столбец давности.

Давность визита важна при построении моделей клиентской аналитики.

**Аспекты вычисления новых атрибутов:**

- Вычисление новых атрибутов на основе исходных с помощью функций.

- Выполнение поэлементными операциями

- Используются эффективные алгоритмы вычисления

- Нет необходимости хранить в хранилище, могут вычисляться на лету в витринах данных

- Решают задачи анализа или моделирования на данных

Так для примера мы имеем возможность рассчитать удельный объем провезенной продукции через таможню в 2016 году на основе данных суммарного объема провезенной продукции и количества продукции

Удельный объем подсчитаем обычным делением.

Однако существует важная оговорка, если в столбце количество будет указано значение, равное нулю, то данное преобразование будет невалидным ввиду деления на ноль.

Данное исключение нужно обработать заранее

Предложим также пример SELECT запроса для получения такой таблицы

SELECT

Отчетная дата,

Суммарный объем,

Количество,

(Суммарный объем / Количество) AS Удельный объем

FROM grouped\_table;

Далее перейдем к аспектам агрегации данных в системах обработки и анализа данных или системах построения витрин данных

**Группировка данных** – процесс получения обобщенных статистик для некоторой большой выборки табличных данных с целью получить важную информацию по уникальным группам категорий объектов.

**Группа** – столбец, значения которого выбираются за уникальные сущности в пределах которого считается агрегированный показатель.

**Синоним:** измерение, категория

В качестве группы могут быть выбраны любые столбцы любого вида и шкалы данных.

**Показатель** – столбец, значения которого берутся за основу подсчитанной агрегированной **меры** на основе статистических функций агрегации

**Группа** – столбец, значения которого выбираются за уникальные сущности в пределах которого считается агрегированный показатель.

**Синоним:** измерение, категория

В качестве группы могут быть выбраны любые столбцы любого вида и шкалы данных.

Однако такое поведение нежелательно по множеству причин, начиная от бессмысленной операции группировки по вещественным атрибутам,

заканчивая огромным множеством целых значений для возможных атрибутов, измеряющих количество чего-либо

**Показатель** – столбец, значения которого берутся за основу подсчитанной агрегированной **меры** на основе статистических функций агрегации

Данные статистические функции ставят в соответствие группе значений в записях одно единственное значение, характеризующее это множество значений.

Приведем пример группировки данных для подсчета итогов по чеку.

Дана таблица транзакций, которая содержит в себе записи чека в пиццерии.

Внутри чека с одним номером имеется несколько заказов пиццы разных категорий и стоимости.

Наверное люди в заведении иногда любят сидеть большой компанией…

Так вот логичной операцией группировки в данной ситуации будет группировка по чеку с подсчетом например суммы заказа целиком по его детализации.

В качестве группы или измерения выберем ID чека, а в качестве показателя Сумму стоимости.

Получим результат в виде агрегированной таблицы данных с суммой чека.

Рассмотрим запрос, который приводит нас к такому результату, итак,

**Подсчёт суммарного чека по транзакционным данным**

SELECT

IDчека,

SUM(Стоимость),

FROM pizza\_transactions,

GROUP BY IDчека;

В операции используется GROUP BY для подведения именно группы по полю ID чека.

Оператора GROUP BY используется исключительно с агрегационной функцией в SELECT, коей является SUM().

SUM() – не единственная функция, чтобы узнать их полный список можно обратиться к документации реализации СУБД.

С точки зрения обработки данных, группировка данных имеет смысл по

**дискретным данным** или **квантованным данным**, как на примере ранее.

С точки зрения аналитики данных, группировка данных имеет смысл также по:

данным с фиксированным перечнем значений и заранее подготовленным атрибутам

Как именно готовить атрибуты для группировки расскажем даллее

Вспомним немного про дискретизацию информации

**Дискретизация данных** – процесс измерения показателя в дискретные моменты времени с заранее заданной частотой дискретизации,

так чтобы перевести непрерывную или аналоговую информацию в цифровой вид.

По факту, вся информация в компьютерной системе представляется в виде дискретной информации по способу организации или хранения, но не по смыслу.

На графике еще раз продемонстрирован пример дискретизации непрерывной зависимости.

Отмеченные точки могут быть записаны в таблицу данных на пересечении атрибута X, измеренного через равные интервалы времени и Y – амплитуды значения показателя в моменты X.

Тогда мы можем получить таблицу ИЗМЕРЕНИЙ непрерывной зависимости.

Квантование же данных напротив, вводит так необходимое нам дискретное значение для интересующего нас показателя, превращая измерения на оси Y в набор дискретных состояний

**Квантование данных** – разбиение диапазона значений непрерывной или дискретной величины на конечное число интервалов.

Шаг по оси измерения имеет фиксированный **шаг**.

Получаем процесс замены непрерывной величины на дискретные равные уровни измерения, ближайшие к исходным, с целью представить данные в виде похожих групп.

На примере на графике представлена визуализация процесса амплитудного квантования зависимости.

Уровни по Y строго фиксированы и значения, находящиеся внутри диапазона, выбранного по Y фиксируются в нем, и принимают метку конкретного диапазона.

Примером может служить введение возрастных групп по диапазонам возрастов, где каждому диапазону возраста например предписаны нормы измерения, например показателей анализа крови.

Такого рода квантование необходимо именно для группировки огромных массивов данных с большой вариабельностью значений.

Рассмотрим задачи, которые преследует группировка данных

Визуализация данных, например в виде столбчатых диаграмм

Глубокое понимание структуры данных (аудит данных), например как раз для построения гистограмм распределений показателей или подсчета уровней значений

Уменьшение гранулярности данных для получения адекватных и понятных показателей в выборке,

то есть квантование и подсчет итогов для квантов

Подведение итогов за определенный временной период

Подсчёт статистик по уникальным категориям объектов с целью дальнейшего анализа

Функции агрегации табличных данных ставят в соответствие ряду данных одно характеризующее их значение.

Среди популярных агрегационных функций встречаются:

- Сумма,

- Среднее,

- Средний квадрат отклонения,

- Медиана – а именно середина отсортированной выборки,

- Квантили – как значения находящиеся в определенной доле позиции отсортированной выборки атрибута,

- Минимум

- Максимум

- Количество

- Количество уникальных

- Количество пропусков

- Первое

- Последнее

**Кросс-таблица** – многозначная агрегация данных по нескольким измерениям фактов (категорий, дискретных данных).

Если в таблице присутствует несколько категориальных атрибутов, то можно произвести операцию агрегации данных по уникальным парам фактов.

Отображение может быть произведено в виде структурированной таблицы или матрицы пересечения фактов.

Приветствую вас уважаемые учащиеся и коллеги,

Мы продолжаем разбираться с основами больших данных на нашем курсе.

Данная тема является последней в рамках серии тем преобразования структурированных данных и начинает разделы понятий инфраструктуры их обработки и аналитики подготовленных данных.

В этой теме мы узнаем:

Каким образом структурированные таблицы можно объединять в хранилищах данных или на этапе трансформации данных

Разберемся в том, что такое ETL процесс, как он помогает унифицировать процессы обработки и хранения данных, а также поймем разницу между классическим ETL процессом и новым ELT процессом обработки данных.

Разберемся в особенностях проектирования хранилищ данных, а также поймем основные его свойства и задачи.

Перейдем к разделу обогащения данных, где разберем:

1. Какие задачи решает обогащение данных и для чего оно необходимо

2. Основные операции обогащения данных в SQL

3. Примеры основных действий по слиянию данных

**Начнем с определения обогащения данных**

**Обогащение данных** — это процесс дополнения сырых данных той информацией, которая в исходном виде в них отсутствует, но необходима для качественного анализа.

Конечными потребителями такой информации чаще всего являются аналитики.

Сам термин "обогащение данных" — это перевод англоязычного Data enrichment, который проводит аналогию между данными и... ураном.

Точно так же, как промышленники насыщают урановую руду, увеличивая долю изотопа урана 235, чтобы её можно было использовать (хочется надеяться, в мирных целях),

в процессе обогащения данных мы насыщаем их информацией.

Для чего необходимо обогащение данных

**Задача обогащения данных** напрямую связана с темой их обработки и анализа.

Обогащение нужно для того, чтобы конечные потребители данных получали качественную и полную информацию относительно исследуемого процесса или процессов работы организации или бизнес-процесса.

Существует только два вида обогащения данных как было показано на иллюстрации ранее:

Первый вид – локальное обогащение данных, которое состоит

в получении отсутствующей информации путём обработки уже имеющихся данных.

Главной особенностью является полная доступность такой информации, а также независимость от внешних факторов,

Недостатком же является сложность в выявлении новой информации из уже собранного массива данных.

Этот вид обогащения подразумевает собой итерационных процесс обработки собранных данных пакетами с целью выжать из них максимальную информацию,

которая не была видна ранее,

и соответственно принести пользу для оптимизируемого бизнес-процесса.

Второй вид обогащения данных – это внешнее обогащение.

Внешнее обогащение состоит в получении отсутствующей информации из иных информационных систем.

Данный вид обогащения является простым с точки зрения логики обработки.

Скорее всего в организации уже имеется стратегия или готовый пайплайн для преобразования внешних данных.

Главным недостатком этой информации как раз является их нелокальность, то есть в ее удаленность, а вместе с тем

время доступа,

а так же формат её представления

и полнота.

Локальное обогащение данных по своей природе может быть вызвано разными причинами,

которые определяются природой необходимости дополнения информации,

а также способами обработки той самой уже имеющейся выборки.

Среди локального обогащения данных выделяют:

Структурное обогащение. Оно состоит в корректировке записей с заданной точностью.

**Структурное** **обогащение** предполагает изменение параметров сообщения, отображающего информацию в зависимости от

- частотного спектра исследуемого процесса,

- скорости обслуживания источников информации

- и требуемой точности.

Статистическое обогащение.

При статистическом **обогащении** осуществляют накопление статистических **данных** и обработку выборок из генеральных совокупностей накопленных **данных**.

Например, если у нас имеется процесс, измеряющийся во времени, временной ряд, то статистическое обогащение текущего временного ряда в выбранном настоящем периоде заключается в его обработке на основе статистических показателей временного ряда за предыдущий временной диапазон, т.е. период времени.

Семантическое обогащение.

*Семантическое* обогащение означает

- минимизацию логической формы, исчислений и высказываний,

- выделение и классификацию понятий содержания информации,

- переход от частных понятий к более общим.

В итоге семантического обогащения удается обобщенно представить обрабатываемую либо передаваемую информацию и устранить логическую противоречивость в ней.

Для более близких к нам примеров можем предложить регулярные выражения для обработки строк и выделения из них основной смысловой части, кодирование категориальных переменных, и т.д.

Прагматическое обогащение данных.

*Прагматическое* обогащение является важной ступенью использования информации для принятия решения,

при котором из полученной информации отбирается наиболее ценная, отвечающая целям и задачам пользователя.

Примером будет служить действие по выборке определенных данных с помощью фильтрации или совмещение признаков.

В дальнейшем будем говорить именно о прагматическом обогащении.

Внешнее обогащение, в отличие от локального опирается на внешних поставщиков данных.

Среди внешних поставщиков выделяют:

Системы с похожей схемой хранения основной бизнес-информации, т.е. с хранилищами той же предметной области

Специализированные базы данных, учетные системы, нормативно-справочная информация и другие источники стандартов предметной области.

Графы знаний

Перейдем к механизмам прагматического обогащения данных, а именно объединению данных в структурированных БД, или реляционной модели

Поговорим об объединении данных в реляционной модели, как завершающей теме трансформации данных в реляционных базах данных

Рассмотрим различные варианты слияния данных, среди которых различные варианты Join-ов в SQL, которые так же реализуются и в других системах анализа и обработки табличных данных

А также варианты объединения записей в таблицах данных

Поговорим о вариантах соединения таблиц по ключу отношений в связах реляционной базы данных.

В SQL присутствует четыре варианта соединения данных по связи двух таблиц, а именно:

- внутреннее соединение по ключам похожих значений,

- левое соединение по похожим ключам правой таблицы, полностью содержащихся в левой

- обратное для левого правое соединение двух таблиц

- полное соединение по ключам как первой так и второй таблицы данных

Давайте рассмотрим данные варианты подробнее

Пусть у нас имеются две таблицы данных с одинаковыми ключами

Пусть у нас имеются две таблицы данных с одинаковыми ключами

- Либо это две таблицы которые содержат разную информацию о, например, клиентах, что может происходить при внешнем обогащении данных, то есть при обновлении информации из внешних источников.

- Либо это две таблицы, связанные в реляционной базе данных одним из видов связей на таблицах.

Ключи в нашем примере – целые числа, данные здесь это символьные выражения X1, X2, X3 в первой таблице и Y1, Y2, Y3 во второй.

В результате выборки данных этих таблиц выяснилось, что в каждой из таблиц существует запись, которая не содержится в другой таблице.

Внутреннее соединение по своему механизму производит следующую операцию,

оно сливает данные двух таблиц только по тем ключам, которые совпадают о двух таблиц одновременно

С помощью прямых линий мы выделяем связи соединения по похожим ключам

Внутреннее соединение по своему механизму производит следующую операцию,

оно сливает данные двух таблиц только по тем ключам,

которые совпадают о двух таблиц одновременно.

Такое соединение может привести к определенным проблемам при его осуществлении, например

При неосторожном внутреннем слиянии данных может произойти потеря критически важных записей одной из таблиц.

К примеру, вы хотите в таблице чеков онлайн-магазина дополнить информацию по чеку адресом доставки из внешней справочной таблицы клиентов.  
  
При использовании внутреннего соединения возможна утеря записей заказов для клиентов,

которые еще не зарегистрированы в вашей системе и не оставили запись в справочной таблице о себе.

К приведенному выше объяснению покажем как осуществляется внутреннее соединение в SQL

C помощью выражения SELECT выбрав все интересующие нас поля каждой из таблиц, указав во FROM главную таблицу,

С помощью INNER JOIN указываем какую таблицу мы хотим соединить с главной по ключу,

а затем в продолжении JOIN пишем по какому совпадающему атрибуту значений мы хотим соединить эти две таблицы.

Полученный SELECT выглядит так:

**SELECT** Key, Val\_x, Val\_y

**FROM** Table\_A**A**

**INNER JOIN** Table\_B**B**

**ON   A.**Key**=   B.**Key;

Таким образом из примера получим результат слияния с двумя записями, данные ключей в которых совпадали.

На слайде показана таблица результата соединения.

Видим, что в результирующей таблице присутствуют не все ключи и наблюдения как в случае с первой и со второй таблицей.

Приведем еще один пример внутреннего соединения таблиц.

Пусть у нас имеется две таблицы:

Таблица купленных товаров клиентами и информация о наших клиентах

Мы хотим дополнить наши чеки информацией о клиентах для проведения возможного дальнейшего статистического исследования, тогда

Заметим, что в таблице чеков клиент 2 не принимал участия, и внутреннее соединение будет происходить только по совпадающим значениям ключей, покажем запрос и результат

SQL запрос, который будет необходим для совершения внутреннего соединения данных будет следующим:  
  
SELECT

table1.товар,

table2.имя,

table2.возраст

FROM table1

INNER JOIN table2 ON table1.клиент = table2.id

В запросе так же указываем все атрибуты по которым необходимо произвести выборку, затем в INNER JOIN описываем способ по которому мы будем это слияние производить

И тогда наш итоговый результат будет таким:

Убрав за ненадобностью значение ключа покупателя, как излишнее,

поскольку всю информацию о клиенте мы приводим в результате на основе денормализации таблицы,

и получаем выборку для проведения дальнейшего исследования.

Например для такой структуры данных, при большом числе покупателей, можно провести исследование числа купленных товаров от возраста.

Опишем правила проведения левого соединения на реляционной базе данных таблиц.

Пусть у нас имеются две таблицы данных с одинаковыми ключами

Ключи в нашем примере – целые числа, данные здесь это символьные выражения X1, X2, X3 в первой таблице и Y1, Y2, Y3 во второй.

В результате выборки данных этих таблиц выяснилось, что в первой (левой) таблице присутствуют записи, которые не содержались во второй (правой) таблице.

Левое соединение по своему механизму производит следующую операцию,

Дополняет записи первой таблицы значениями атрибутов по совпадающим ключам второй таблицы

С помощью прямых линий мы выделяем связи соединения по похожим ключам.

Видим что для ключа 3 в левой таблице не нашлось похожего ключа в правой, поэтому к нему дополняем по умолчанию пропущенное значение.

Левое соединение по своему механизму производит следующую операцию,

оно сливает данные двух таблиц только по тем ключам,

которые совпадают в левой таблице.

Если в правой таблице не находится похожих ключей, то запись дополняется пропуском.

Левое соединение не влечет за собой уменьшение числа записей в главной таблице,

но влечет за собой уменьшение качества данных.

Чаще всего левое соединение применяется для транзакций, где критично не потерять значимые записи в таблице прецедентов.

C помощью выражения SELECT выбрав все интересующие нас поля каждой из таблиц, указав во FROM главную таблицу,

С помощью LEFT JOIN указываем какую таблицу мы хотим соединить с главной по ключу,

а затем в продолжении JOIN пишем по какому совпадающему атрибуту значений мы хотим соединить эти две таблицы.

Полученный SELECT выглядит так:

**SELECT** Key, Val\_x, Val\_y

**FROM** Table\_A**A**

**LEFT JOIN** Table\_B**B**

**ON   A.**Key**=   B.**Key;

Таким образом из примера получим результат слияния с тремя записями, данные ключей в которых совпадают с левой таблицей.

На слайде показана таблица результата соединения.

Видим, что в результирующей таблице присутствуют все ключи и наблюдения из главной левой таблицы, но значение в последней записи пустое.

Приведем еще один пример левого соединения таблиц, как это было и с внутренним.

Пусть у нас имеется две таблицы:

Таблица купленных товаров клиентами и информация о наших клиентах

Мы хотим дополнить наши чеки информацией о клиентах для проведения возможного дальнейшего статистического исследования, тогда

SQL запрос, который будет необходим для совершения внутреннего соединения данных будет следующим:  
  
SELECT

table1.товар,

table2.имя,

table2.возраст

FROM table1

LEFT JOIN table2 ON table1.клиент = table2.id

В запросе так же указываем все атрибуты по которым необходимо произвести выборку, затем в LEFT JOIN описываем способ по которому мы будем это слияние производить

И тогда наш итоговый результат будет таким:

В таблице видим, что для покупки 3 товара записи стали неполными, ввиду чего проведение полноценного анализа по данной строке в дальнейшем не будет возможным.

Опишем правила проведения правого соединения на реляционной базе данных таблиц.

Правое соединение является симметричным правилом для левого соединения и может быть описано в тех же терминах

Пусть у нас имеются две таблицы данных с одинаковыми ключами

В результате выборки данных этих таблиц выяснилось, что в первой (правой) таблице присутствуют записи, которые не содержались во второй (левой) таблице.

Правое соединение по своему механизму производит следующую операцию,

Дополняет записи первой таблицы значениями атрибутов по совпадающим ключам второй таблицы

С помощью прямых линий мы выделяем связи соединения по похожим ключам.

Видим что для ключа 4 в правой таблице не нашлось похожего ключа в левой, поэтому к нему дополняем по умолчанию пропущенное значение.

Перейдем сразу к механизму соединения

C помощью выражения SELECT выбрав все интересующие нас поля каждой из таблиц, указав во FROM главную таблицу,

С помощью RIGHT JOIN указываем какую таблицу мы хотим соединить с главной по ключу,

а затем в продолжении JOIN пишем по какому совпадающему атрибуту значений мы хотим соединить эти две таблицы.

Полученный SELECT выглядит так:

**SELECT** Key, Val\_x, Val\_y

**FROM** Table\_A**A**

**RIGHT JOIN** Table\_B**B**

**ON   A.**Key**=   B.**Key;

Таким образом из примера получим результат слияния с тремя записями, данные ключей в которых совпадают с правой таблицей.

На слайде показана таблица результата соединения.

Видим, что в результирующей таблице присутствуют все ключи и наблюдения из главной правой таблицы, но значение в последней записи пустое.

Полное соединение призвано соединить таблицы без потери любой информации от обоих таблиц в соединении, в результате чего появляются большие проблемы с качеством данных. Однако существуют разные приложения и такой схеме соединения таблиц

Полное соединение опишем так же как и остальные

Пусть у нас имеются две таблицы данных с одинаковыми ключами

В каждой из таблиц существуют записи, которых нет в соединяемой

Полное соединение по своему механизму производит следующую операцию,

Создает таблицу в которой присутствуют все записи ключей из первой и второй таблицы, а значения, которые в таблицах отсутствовали, принимают пустые значения.

С помощью прямых линий мы выделяем связи соединения по похожим ключам.

Видим что для ключа 3 и 4 в правой таблице не нашлось похожего ключа в левой и правой таблицах, поэтому к ним дополняем по умолчанию пропущенное значение.

Перейдем сразу к механизму соединения

C помощью выражения SELECT выбрав все интересующие нас поля каждой из таблиц, указав во FROM главную таблицу,

С помощью FULL JOIN указываем какую таблицу мы хотим соединить с главной по ключу,

а затем в продолжении JOIN пишем по какому совпадающему атрибуту значений мы хотим соединить эти две таблицы.

Полученный SELECT выглядит так:

**SELECT** Key, Val\_x, Val\_y

**FROM** Table\_A**A**

**FULL JOIN** Table\_B**B**

**ON   A.**Key**=   B.**Key;

На слайде показана таблица результата соединения.

Видим, что в результирующей таблице присутствуют все ключи и наблюдения из главной правой и левой таблицы,   
но значение в предпоследней и последней записи пустое.

Пусть у нас имеется две таблицы:

Таблица купленных товаров клиентами и информация о наших клиентах

Мы хотим дополнить наши чеки информацией, тогда

SQL запрос, который будет необходим для совершения полного соединения данных будет следующим:  
  
SELECT

table1.товар,

table2.имя,

table2.возраст

FROM table1

FULL OUTER JOIN table2 ON table1.клиент = table2.id

В запросе так же указываем все атрибуты по которым необходимо произвести выборку,

затем в FULL JOIN описываем способ по которому мы будем это слияние производить

И тогда наш итоговый результат будет таким:

Результатом операции полного соединения будет таблица, содержащая все записи из первой и второй таблицы.

В местах, где ключ не был найден в соединяемой таблице, дополняемое значение было заменено на пустое NULL

В реляционных базах данных, операции Union и Union All используются для объединения результатов двух или более запросов SELECT в один результат.

Они работают только в том случае, если каждый запрос SELECT возвращает одинаковое количество столбцов и эти столбцы имеют схожие типы данных.

Union: Оператор UNION используется для объединения двух наборов данных в один.

Он автоматически удаляет все дублирующиеся строки из результата.

Важно отметить, что оператор UNION выполняет сортировку данных, что может замедлить выполнение запроса на больших объемах данных.

Union All: Оператор UNION ALL также объединяет два набора данных в один, но, в отличие от оператора UNION, он не удаляет дубликаты из результата.

Кроме того, UNION ALL не выполняет сортировку данных, поэтому он работает быстрее, чем UNION, особенно на больших объемах данных.

Рассмотрим примеры использования Union и Union ALL

Рассмотрим данные чеков магазина. Пусть у нас имеются детализации продаж из магазина. Данные поступают в систему ежесекундно и в данных бывают повторения в результате грязного транзакционного чтения и записи. В реальных системах такое может происходить в результате параллельной обработки данных.

Объединим данные в одну таблицу, чтобы дополнить данные актуальными записями продаж, воспользовавшись Union и Union All

Объединение

Объединение

Объединение

Объединение

Видим, что запрос на объединение данных состоит из нескольких SELECT с одинаковыми столбцами, в противном случае запрос выполнится с ошибками  
(SELECT чек,

товар,

количество

FROM transactions1)

UNION (SELECT чек,

товар,

количество

FROM transactions2)

Результатом будет служить таблица с отсортированными записями по ключу чека

Как видим, полные повторения, подсвеченные ранее не представлены в результирующей таблице

При использовании UNION ALL на похожие схемы таблиц получаем в результате объединенную выборку с дубликатами и в неотсортированном порядке по ключу

Рассмотрим ETL и ELT процессы в обработке больших данных. Для чего они применяются и в чем между ними разница.

Процесс **ETL** представляет собой комплекс операций, реализующих процесс переноса первичных данных из различных источников в аналитическое приложение или поддерживающее его хранилище данных.

**ETL-приложения** извлекают информацию из одного или нескольких источников, преобразуют ее в формат,

поддерживаемый системой хранения и обработки, которая является получателем данных, а затем загружают в нее преобразованную информацию.

Любая ETL-система должна обеспечивать выполнение трех основных этапов процесса переноса данных:

**Извлечение данных** — на этом шаге данные извлекаются из одного или нескольких источников и подготавливаются к этапу преобразования.

На стадии **извлечения** и **сбора** данных ставится задача загрузки данных из нескольких внешних или внутренних источников организации в поток обработки данных.

Как ранее мы уже упоминали, проблемы, возникающие на этапе сбора могут быть следующими:

* количество внешних источников данных,
* разнородность формата и интерфейса предоставления данных,
* согласованность времени сбора данных,
* пропуски в данных и проблемы при интерпретации результатов,
* объем собираемых данных,
* консолидация данных.

**Преобразование данных** — производится преобразование форматов и кодировки данных, а также их интеграция и очистка;

Все операции над данными в процессе ETL производятся в так называемой промежуточной области, где для этого создаются временные таблицы.

Стадия предобработки данных решает задачу преобразования полученных данных в необходимую форму.

Среди направлений предобработки данных выделяют:

* Этап очистки данных
* Этап оптимизации данных

В результате данные представляются в формате структуры для хранения в долговременном хранилище и в форме, подходящей для анализа.

Очистка данных — этап удаления нерелевантных значений показателей или записей данных с нетипичными значениями.

В очистке данных выделяют стадии:

* обработка пропусков,
* удаление дубликатов и противоречий,
* обработка выбросов (нетипичных значений)
* восстановление структуры данных,
* верификация целостности данных.

После этапа очистки данные готовы к загрузке в хранилище данных.

**Загрузка данных** — запись преобразованных, интегрированных и очищенных данных в соответствующую систему хранения.

Процесс загрузки заключается в переносе данных из промежуточных таблиц в структуры хранилища данных (ХД).

После завершения загрузки выполняются дополнительные операции над данными (тестирование), только что загруженными в ХД.

К ним относятся: переиндексация, верификация данных и т. д.

Если тестирование показало, что несоответствия,

позволяющие заподозрить потерю или недостоверность данных, отсутствуют, то можно считать загрузку данных в ХД успешной.

Одной из основных проблем данного шага является то, что далеко не всегда данные загружаются полностью: в загрузке некоторых записей может быть отказано.

Отклонение записей происходит по следующим причинам:

на этапе преобразования данных не удалось исправить все критичные ошибки, которые блокируют загрузку записей в ХД;

В процессе обработки данных перед загрузкой в хранилище данных могут быть обнаружены ошибки или несоответствия, которые делают данные непригодными для загрузки.

Это могут быть, например,

- недостающие значения,

- некорректные форматы данных,

- нарушения ограничений целостности или другие проблемы.

Если эти ошибки не исправлены на этапе преобразования данных, то загрузка данных в хранилище может быть заблокирована.

некорректный порядок загрузки данных;

Порядок загрузки данных важен, особенно если между данными есть зависимости.

Например, если в таблице есть внешние ключи, которые ссылаются на другую таблицу,

то эта другая таблица должна быть загружена первой.

Если порядок загрузки не соблюдается, то это может привести к ошибкам и отклонению записей.

внутренние проблемы ХД, например недостаток места в нем;

Если в хранилище данных нет достаточно места для загрузки новых данных,

то процесс загрузки будет прерван и записи будут отклонены.

Кроме того, могут быть и другие внутренние проблемы, такие как проблемы с производительностью, ошибки в системе управления базами данных или проблемы с сетью.

прерывание процесса загрузки или остановка его пользователем.

Если процесс загрузки данных прерывается по какой-либо причине (например, из-за сбоя в системе или если пользователь останавливает процесс),

то незагруженные записи будут отклонены.

В зависимости от настроек системы, может потребоваться повторная загрузка всех данных или только незагруженных записей.

ETL процесс соответствует архитектуре обработки бизнес-данных, то есть данных не критичных к скорости загрузки в хранилище,

но критичных к качеству и полноте их представления,

а также возможности извлечения из них качественных выводов.

Поэтому

ELT (Extract, Load, Transform) — это процесс переноса данных из разнородных источников в хранилище данных с целью их дальнейшего анализа.

Также это подход к обработке и анализу данных, который включает в себя три основные стадии:

- извлечение данных из различных источников,

- загрузка этих данных в целевую систему (обычно это хранилище данных)

- и преобразование данных для последующего анализа или отчетности.

В целом, процесс ELT выполняет те же функции, что и ETL с той только разницей, что этапы загрузки и преобразования меняются местами.

ELT извлекает данные из исходных местоположений, но вместо перемещения их в промежуточную область для преобразования,

загружает необработанные данные непосредственно в приемник, где их можно преобразовать по мере необходимости в соответствии с конкретными целями и задачами анализа.

Extract (Извлечение):

На этом этапе данные извлекаются из различных источников, которые могут включать в себя базы данных, файлы, потоки данных, API и другие.

Данные могут быть структурированными, полуструктурированными или неструктурированными.

Извлеченные данные обычно сохраняются во временное хранилище перед загрузкой в целевую систему.

Load (Загрузка):

Загруженные данные затем перемещаются в целевую систему.

В большинстве случаев это хранилище данных или data lake.

Загрузка может происходить в режиме реального времени или пакетной обработки в зависимости от требований бизнеса и объема данных.

Transform (Преобразование):

После загрузки данные преобразуются для последующего анализа или отчетности.

Это может включать в себя очистку данных, объединение данных из различных источников, преобразование данных в нужный формат,

создание новых полей или метрик на основе существующих данных и другие операции.

Преобразование данных обычно выполняется с помощью SQL или других языков запросов.

Помимо порядка проведения операций, между процессами ETL и ELT встречаются следующие различия:

технология ELT новее ETL;

ELT позволяет работать с неструктурированной, слабоструктурированной и структурированной информацией. ETL только со структурированной;

в ELT данные загружаются в хранилище сразу после извлечения. Их преобразование производится по мере необходимости. Это экономит время ожидания загрузки данных.

Неочевидные отличия ETL и ELT процессов

1. Обработка данных: В ETL процессе, данные обрабатываются перед загрузкой в хранилище, в то время как в ELT процессе, данные сначала загружаются в хранилище, а затем обрабатываются.

2. Место обработки: В ETL процессе, преобразование данных происходит на стороне источника данных или на специальном сервере преобразования данных. В ELT процессе, преобразование данных происходит непосредственно в целевом хранилище данных.

3. Скорость загрузки: ELT обычно быстрее, поскольку он загружает данные в хранилище без предварительной обработки.

4. Сложность данных: ETL лучше подходит для структурированных данных, тогда как ELT может обрабатывать и структурированные, и неструктурированные данные.

5. Производительность: Поскольку ETL требует предварительной обработки, она может замедлить производительность источника данных. ELT не влияет на производительность источника данных, поскольку обработка происходит после загрузки данных.

6. Гибкость: ELT обеспечивает большую гибкость, поскольку позволяет обрабатывать и анализировать данные прямо в хранилище данных.

7. Контроль качества данных: В ETL процессе, контроль качества данных происходит до загрузки данных в хранилище. В ELT процессе, контроль качества данных происходит после загрузки данных в хранилище.

8. Безопасность: В ETL процессе, данные могут быть защищены и анонимизированы до загрузки в хранилище. В ELT процессе, это может быть более сложной задачей, поскольку данные уже находятся в хранилище.

Теперь поговорим о том, что представляет собой хранилище данных как функциональная единица потока обработки больших данных.

**Хранилище данных** — это цифровая система хранения, которая выполняет объединение и согласование больших объемов данных из разных источников.

Оно предоставляет данные для бизнес-аналитики, отчетов и анализа, а также обеспечивает поддержку нормативных требований.

С помощью ХД компании превращают свои данные в ценную информацию и принимают взвешенные решения на основе данных.

Хранилище данных объединяет данные из различных источников в одном месте, что облегчает доступ и управление данными.

Хранилище данных

- обеспечивает защиту данных от несанкционированного доступа, потерь или повреждений.

- может включать инструменты для очистки данных, чтобы улучшить их точность и согласованность.

- позволяет анализировать большие объемы данных, чтобы получить ценные бизнес-инсайты и информацию для принятия решений

- может увеличить скорость обработки запросов и уменьшить нагрузку на операционные системы

Хранилища данных позволяют хранить исторические данные на протяжении длительного времени для анализа трендов и паттернов

- а также может помочь организациям выполнять требования по отчетности и соблюдению законодательства

Хранилища могут использовать реляционную модель, когда данные в них нормализованы, или многомерную, использующую так называемые измерения.

В нормализованных хранилищах данные содержатся в таблицах третьей нормальной формы.

Преимущество нормализованных ХД заключается в простоте разработки и управления.

Недостатком является необходимость денормализации данных «на лету» при их извлечении из множества таблиц при выполнении сложных аналитических запросов.

При формировании больших выборок это приводит к значительным задержкам в получении данных,

а если хранилище и аналитическая платформа интегрированы в информационную систему предприятия, то возрастает нагрузка на всю систему, что может осложнить работу многих пользователей.

Данную проблему частично удается решить, используя в хранилище модель данных, основанную на измерениях.

Применяются две разновидности многомерных моделей данных — «звезда» и «снежинка».

Все загружаемые в хранилище данные обязательно должны быть определены как измерение, атрибут либо факт.

Хранилище данных решает ряд важных задач:

предоставление оперативного доступа и хранение информации (структурированной и нет);

Основная задача хранилища данных - обеспечивать быстрый и надежный доступ к информации для пользователей и приложений.

Это означает, что данные должны быть организованы таким образом, чтобы их можно было легко и быстро извлекать и обновлять.

Кроме того, хранилище данных должно обеспечивать надежное хранение информации, чтобы предотвратить потерю данных в случае сбоя системы или других проблем.

расширение и масштабирование данных при растущем увеличении объема информации;

С ростом объема данных, которые нужно хранить и обрабатывать, хранилище данных должно иметь возможность расширяться и масштабироваться, чтобы справиться с этими требованиями.

Это может включать в себя добавление дополнительного пространства для хранения, увеличение производительности системы или распределение данных на несколько серверов или местоположений.

функции безопасности (отражение локальных и сетевых атак, борьба с компроментацией данных);

Хранилище данных должно обеспечивать защиту информации от несанкционированного доступа, модификации или удаления.

Это может включать в себя использование шифрования, контроля доступа, аудита и других мер безопасности.

Кроме того, хранилище данных должно иметь возможность восстановления после сбоя или атаки.

репликация (дублирование данных);

Репликация данных - это процесс создания и поддержания нескольких копий данных в разных местах.

Это может быть необходимо для обеспечения высокой доступности данных, балансировки нагрузки или для предотвратения потери данных в случае сбоя одного из серверов.

виртуализация (распределение трафика между пользователями);

Виртуализация в контексте хранилища данных обычно означает абстракцию физического хранения данных, что позволяет управлять данными на более высоком уровне.

Это может облегчить управление данными, расширение и масштабирование, а также может повысить эффективность использования ресурсов хранения.

сжатие данных

Сжатие данных - это процесс уменьшения объема данных для экономии пространства хранения или увеличения скорости передачи данных.

Это может быть особенно полезно для больших наборов данных или для ситуаций, когда пространство для хранения ограничено.

Однако сжатие может потребовать дополнительных ресурсов для обработки данных и может увеличить сложность управления данными.

**А также Хранилище данных** должно обладать следующими свойствами:

Предметная ориентированность — создается с ориентацией на решаемую проблему хранения и анализа

Это свойство хранилища данных означает, что данные в нем организованы вокруг определенных предметных областей или тем, а не вокруг приложений или источников данных.

Например, в банковском секторе предметные области могут включать счета клиентов, кредиты, транзакции и т.д.

Такой подход позволяет упростить доступ и анализ данных, так как пользователи могут легко найти все данные, относящиеся к конкретной предметной области.

Консолидированность — данные объединены в схемы для удобной итеграции и решения задач

В хранилище данных данные собираются из различных источников и объединяются в единое целое.

Это может включать в себя процесс очистки данных, преобразования их в единый формат и решения проблем с качеством данных.

Консолидированность данных позволяет упростить анализ данных и обеспечивает более точные и полные результаты.

Задача консолидации данных и заключается в соблюдении ранее упомянутых в ETL свойств.

В процессе консолидации данных решаются следующие задачи:

- выбор источников данных;

- разработка стратегии консолидации;

- оценка качества данных;

- обогащение;

- очистка;

- перенос в хранилище данных.

Энергонезависимость

Это свойство означает, что данные в хранилище не теряются при отключении питания. Э

то достигается за счет использования постоянной памяти, такой как жесткие диски или SSD.

Энергонезависимость является важным свойством для любого хранилища данных, так как оно обеспечивает безопасность и надежность хранения данных.

Поддержка изменений во времени — настроенность на постоянную поддержку внесения новых данных

Хранилище данных способно отслеживать и сохранять исторические данные, позволяя пользователям анализировать тренды и изменения во времени.

В отличие от оперативных систем, которые обычно содержат только текущие данные, хранилища данных сохраняют большой объем исторических данных.

Это позволяет проводить более глубокий анализ и принимать более обоснованные решения.

Хранилища данных – сложная структура хранения как бизнес-информации, которые проходят огромные процедуры предобработки,   
так и оперативных данных, которые проходят по быстрому пути обработки и выдаются пользователям или автоматическим системам.

Для проектирования хранилищ данных были созданы паттерны концептуального проектирования схем реляционных таблиц и связей баз данных.

Кратко пройдемся по данным паттернам проектирования систем хранения данных.

Star Schema:

Это наиболее простая схема проектирования хранилища данных.

Она называется так из-за своей звездообразной структуры, где центральная таблица фактов окружена таблицами измерений.

Таблица фактов содержит квантитативную информацию для анализа и ключи к каждой из связанных таблиц измерений.

Таблицы измерений содержат подробности об интересующих вас атрибутах.

Star Schema – это модель данных, которая используется для организации данных в хранилище данных (Data Warehouse) или в витринах (Data Mart).

Она называется "звездой" из-за своей структуры, которая напоминает звезду.

Star Schema состоит из двух типов таблиц: фактов и измерений.

1. Таблицы фактов:

Это центральная таблица в Star Schema, которая содержит ключевые метрики или измерения бизнес-процесса.

Например, в розничной торговле таблица фактов может содержать информацию о продажах, такую как количество проданных единиц, общая стоимость продаж и т.д.

Каждая строка в таблице фактов представляет отдельную транзакцию.

2. Таблицы измерений:

Это таблицы, которые окружают таблицу фактов и содержат детализированную информацию об атрибутах, связанных с фактами.

Они содержат контекстную или описательную информацию.

Например, для розничной торговли таблицы измерений могут включать:

- детали о продуктах (например, имя продукта, категория продукта),

- детали о магазинах (например, адрес магазина, город)

- и детали о времени (например, год, квартал, месяц).

В Star Schema каждая таблица измерений связана напрямую с таблицей фактов через первичный ключ.

Это обеспечивает простоту понимания и высокую производительность запросов.

Однако, стоит отметить, что Star Schema не всегда является идеальным решением для всех ситуаций.

Она хорошо работает для простых или средних по сложности запросов, но может быть неэффективной для сложных запросов, которые требуют анализа многих измерений.

Snowflake Schema:

Это расширение звездной схемы, где таблицы измерений нормализованы.

Каждая таблица измерений разбивается на дополнительные таблицы, что создает структуру, похожую на снежинку.

Это уменьшает объем хранимых данных, но увеличивает сложность запросов, так как теперь нужно соединять больше таблиц.

Например, вместо одной таблицы "Дата", которая содержит поля "День", "Месяц", "Квартал", "Год", в схеме снежинки будет четыре связанные таблицы — по одной для каждого из этих полей.

Преимущества схемы снежинки:

1. Эффективное использование дискового пространства за счет нормализации данных.

2. Улучшенная производительность запросов за счет уменьшения объема данных для обработки.

3. Возможность более детального анализа данных за счет дополнительной детализации измерений.

Недостатки схемы снежинки:

1. Сложность проектирования и поддержки из-за большого количества связанных таблиц.

2. Увеличение времени выполнения запросов из-за необходимости обрабатывать большее количество связей между таблицами.

3. Усложнение процесса загрузки данных в хранилище из-за необходимости обновления большего количества таблиц.

Data Vault - это схема проектирования хранилища данных, разработанная Дэном Линстедтом.

Она представляет собой гибкую, масштабируемую и простую в поддержке структуру для хранения больших объемов данных.

Data Vault состоит из трех основных типов компонентов: hubs, links и satellites.

1. Hubs (хабы):

Это основные объекты или деловые ключи в системе.

Они представляют собой уникальные идентификаторы, которые обычно являются первичными ключами в таблицах.

Например, клиент, товар или продажа могут быть хабами в БД розничной торговли.

2. Links (связи):

Связи представляют отношения между hubs.

Они представляют собой ассоциации или транзакции между двумя или более хабами.

Например, покупка товара клиентом может быть представлена как связь между хабом клиента и хабом товара.

3. Satellites (спутники):

Спутники предоставляют контекстную информацию или атрибуты для хабов и списков.

Они содержат все дополнительные данные, которые связаны с хабом или списком.

Например, адрес клиента или описание товара могут быть спутниками.

Основное преимущество Data Vault заключается в том, что он позволяет легко интегрировать новые источники данных без необходимости перестраивать всю схему.

Благодаря этому, Data Vault особенно полезен для компаний, которые работают с большими объемами данных и имеют необходимость быстро адаптироваться к изменениям в бизнес-требованиях.

Важно отметить, что Data Vault не предназначен для непосредственного использования конечными пользователями для отчетности или анализа данных.

Он служит для хранения данных на промежуточном уровне перед тем, как они будут перенесены в более подходящую для конечного пользователя структуру, такую как звездообразная схема или снежинка.

Anchor модель проектирования баз данных - это уникальный подход к моделированию данных, созданный Ларсом Роннбаком.

Этот подход отличается от традиционных методов моделирования, таких как ER (Entity-Relationship) и UML (Unified Modeling Language),

и предлагает ряд преимуществ, особенно при работе с большими и сложными наборами данных.

В Anchor моделировании, все данные организованы вокруг "якорей".

Якорь - это абстрактное представление объекта в предметной области.

Каждый якорь имеет уникальный идентификатор, который не зависит от его атрибутов.

Это позволяет легко отслеживать и управлять объектами, даже если их атрибуты меняются со временем.

Атрибуты объекта представлены в виде "атомов".

Атом - это минимальная единица информации, которую можно присвоить якорю.

Атомы могут быть связаны с одним или несколькими якорями, и они могут быть связаны друг с другом, чтобы образовать более сложные структуры данных.

Anchor модель также использует понятие "связей" для описания отношений между объектами.

Связь - это набор правил, которые определяют, как атомы могут быть связаны друг с другом и с якорями.

Одним из ключевых преимуществ Anchor моделирования является его гибкость.

Поскольку все данные организованы вокруг якорей, можно легко добавлять, удалять или изменять атрибуты объектов без необходимости перестраивать всю структуру базы данных.

Это делает Anchor модель особенно подходящей для работы с большими и динамическими наборами данных.

Кроме того, Anchor модель обеспечивает высокую нормализацию данных, что уменьшает дублирование информации и улучшает эффективность хранения данных.

Она также поддерживает временные аспекты данных, что позволяет отслеживать изменения в данных со временем.

В целом, Anchor модель предлагает мощный и гибкий подход к моделированию данных, который может быть очень полезен при работе с большими и сложными наборами данных.

Приветствую вас уважаемые учащиеся и коллеги,

Мы продолжаем разбираться с основами больших данных на нашем курсе.

Данная тема является последней в рамках серии тем преобразования структурированных данных и начинает разделы понятий инфраструктуры их обработки и аналитики подготовленных данных.

В данной теме подробно поговорим о методологии проведения анализа и исследования данных CRISP-DM

*Сегодня мы детально изучим методологию проведения аналитического исследования, а также что стоит за каждым этапом методологии CRISP-DM как самой популярной методологии, применяющейся в интеллектуальном анализе данных.*

*Что есть аналитика данных?*

*Аналитика данных — область занимающаяся преобразованием «сырых» данных в практические выводы*

Использует определенный набор процессов и технологий для поиска тенденций и закономерностей в данных

Задача аналитики данных — получение выгоды из данных путем их преобразований, визуализации и построения описательных и предиктивных моделей, а также их оценки.

Аналитика данных преобразует необработанные данные в практические выводы.

Она включает в себя ряд инструментов, технологий и процессов, используемых для поиска тенденций и решения проблем с помощью данных. Аналитика данных может формировать бизнес-процессы, повышать эффективность принятия решений и способствовать росту бизнеса.

Аналитика больших данных – это процесс поиска закономерностей, тенденций и взаимосвязей в огромных массивах данных.

Для такой сложной аналитики требуются специальные инструменты и технологии, вычислительные мощности и хранилища данных, поддерживающие масштаб.

Аналитика является сложной междисциплинарной областью исследования процессов на основе статистических или сырых наблюдений и объединяет в себе подобласти различных дисциплин, связанных

- как с вычислениями, объясняя самые продвинутые алгоритмы обработки и вычислений над наблюдениями,

- так и с информационными технологиями, изучающими принципы оптимальной реализации вычислительных методов математики и выстраивая эффективные системы хранения и представления информации

- а также системный анализ, который позволяет объединять причинно-следственные связи множества различных сущностей и явлений, сопровождающих предметную область исследования

Забирая лучше практики своих смежных областей аналитика данных является базой для проведения работы над данными с целью получения определенного рода выводов из них.

С процессом развития вычислительной техники в различных ее проявлениях алгоритмы вычислительной математики позволили обучать на данных модели и методы, способные в полуавтоматическом режиме обнаруживать знания в имеющихся данных. В последствие, область аналитики данных, в которой исследования ведутся с применением таких алгоритмов стали называть интеллектуальным анализом данных

Интеллектуальный анализ данных (data mining) исследует закономерности, которые нельзя выявить при обычном просмотре данных, и использует методы математического и статистического анализа.

На основе выявленных закономерностей формируется модель интеллектуального анализа данных.

Интеллектуальный анализ данных представляет собой процесс обнаружения пригодных к использованию сведений в крупных наборах данных.

В интеллектуальном анализе данных применяется математический анализ для выявления закономерностей и тенденций, существующих в данных.

Обычно такие закономерности нельзя обнаружить при традиционном просмотре данных, поскольку связи слишком сложны, или из-за чрезмерного объема данных.

Основу методов data mining составляют всевозможные методы классификации, моделирования и прогнозирования, основанные на применении деревьев решений, искусственных нейронных сетей, генетических алгоритмов, эволюционного программирования, ассоциативной памяти, нечёткой логики.

Знания, добываемые методами data mining, принято представлять в виде закономерностей (паттернов).

В качестве таких выступают:

- ассоциативные правила (корреляционный анализ);

- деревья решений;

- кластеры (кластерный анализ);

- математические функции (регрессионный анализ).

Алгоритмы поиска таких закономерностей находятся на пересечении областей: Искусственный интеллект, Математическая статистика, Математическое программирование, Визуализация, OLAP.

На известном аналитическом портале kdnuggets.org периодически публикуется опрос (например, здесь),

согласно которому среди методологий анализа данных первое место по популярности регулярно занимает именно CRISP-DM,

дальше с большим отрывом идет SEMMA и реже всего используется KDD Process.

В целом, эти три методологии очень похожи друг на друга (здесь сложно придумать что-то принципиально новое). Однако CRISP-DM заслужила популярность как наиболее полная и детальная.

По сравнению с ней KDD является более общей и теоретической, а SEMMA – это просто организация функций по целевому предназначению в инструменте SAS Enterprise Miner и затрагивает исключительно технические аспекты моделирования, никак не касаясь бизнес-постановки задачи.

**Методология CRISP-DM (Cross-Industry Standard Process for Data Mining) -** Открытый стандарт моделирования процессов, описывающий общие подходы, используемые в интеллектуальном анализе данных.

Стандарт CRISP-DM начал разрабатываться 1996 году консорциумом пяти компаний, которые имели значительный собственный опыт использования аналитических технологий:

Integral Solutions, Teradata, Daimler AG, NCR Corporation и OHRA.

Первая версия методологии была представлена на 4-м семинаре CRISP-DM SIG в Брюсселе в марте 1999 года и опубликована в виде пошагового руководства по интеллектуальному анализу данных в конце того же года.

Между 2006 и 2008 годами был выпущен CRISP-DM 2.0 и обсуждались вопросы обновления модели процесса.

CRISP-DM стал наиболее широко используемой формой модели интеллектуального анализа данных.

Недостатком этой модели является отсутствие средств поддержки управления проектами.

Преимущество CRISP-DM заключается в том, что он нейтрален в отношении предметных областей, инструментов и приложений.

Стандарт CRISP-DM представляет процесс интеллектуального анализ данных в виде шести фаз:

**- Понимание бизнес-целей** (Business Understanding)

**- Понимание данных** (Data Understanding)

- **Подготовка данных** (Data Preparation)

- **Моделирование** (Modeling)

- **Оценка** (Evaluation)

**- Внедрение** (Deployment)

**Подробнее о каждой из фаз будем говорить далее в этой лекции**

**Начнем с простого, в начале инженеры данных подготавливают сырые данные, поступающие из внешних источников на основе результатов бизнес-анализа, проведенного в преддверии разработки аналитического проекта**

**Понимание бизнес-целей** (Business Understanding).

На данном этапе производится исследование бизнес-процессов компании и предлагаются идеи относительно применения анализа данных для их совершенствования, формулируются конечные цели анализа.

Для этого к обсуждению приглашается как можно больше заинтересованных специалистов и экспертов.

Результатом этапа должен стать план аналитического проекта.

Кроме этого, необходимо убедиться в целесообразности проекта, прежде чем тратить на него ресурсы.

На этапе бизнес-анализа решаются следующие задачи

Первой задачей становится определение бизнес-целей

В первую очередь, мы инициируем встречу с заказчиком с целью выяснения его конкретных требований и предпочтений относительно проекта (или предоставляем ему информацию).

Ответы на следующие вопросы являются важными для нашего понимания.

Структура организации: кто из представителей заказчика принимает участие в проекте, кто выделяет финансирование для проекта, кто принимает ключевые решения, кто будет основным пользователем? На данном этапе мы формируем список контактов и поток информации о проекте.

Узнаем какова бизнес-цель данного проекта? Например, это может быть снижение оттока клиентов.

Есть ли какие-либо уже разработанные решения? Если они существуют, то каковы они и почему текущее решение не соответствует требованиям заказчика?

Данные три пункта помогают отсеять более половины проектов по анализу данных от полного провала на рынке во избежание ненужной разработки существующего продукта который уже решает существующую задачу лучше и экономнее

В дальнейшем происходит стадия оценки текущей ситуации

Когда вместе с заказчиком разобрались, что мы хотим, нужно оценить, что мы можем предложить с учетом текущих реалий.  
  
Оцениваем, хватает ли ресурсов для проекта.

* Есть ли доступное железо или его необходимо закупать?
* Где и как хранятся данные, будет ли предоставлен доступ в эти системы, нужно ли дополнительно докупать/собирать внешние данные?
* Сможет ли заказчик выделить своих экспертов для консультаций на данный проект?

Нужно описать вероятные риски проекта, а также определить план действий по их уменьшению.  
  
Типичные риски следующие.

Не уложиться в сроки.

Финансовые риски (например, если спонсор потеряет заинтересованность в проекте).

Малое количество или плохое качество данных, которые не позволят получить эффективную модель.

Данные качественные, но закономерности в принципе отсутствуют и, как следствие, полученные результаты не интересны заказчику.

Важно, чтобы заказчик и исполнитель говорили на одном языке, поэтому перед началом проекта лучше составить глоссарий и договориться об используемой в рамках проекта терминологии.

Так, например, если мы делаем модель оттока для телекома, необходимо сразу договориться, что именно мы будем считать оттоком – например, отсутствие значительных начислений по счету в течение 4 недель подряд.

Далее стоит (хотя бы грубо) оценить зоны интереса.

В machine-learning проектах обоснованную оценку окупаемости часто можно получить только по завершению проекта (либо пилотного моделирования),

но понимание потенциальной выгоды может стать хорошим драйвером для всех.

Определение целей аналитики

После того, как задача поставлена в бизнес-терминах, необходимо описать ее в технических терминах.

В частности, отвечаем на следующие вопросы.

Какую метрику мы будем использовать для оценки результата моделирования

Для различных задачи интеллектуального анализа данных существуют различные метрики качества, измеряющие пользу в определенных аспектах.

Одни из этих метрик качества являются важными для задач моделирования, другие нет, третьи вовсе вредят пониманию качества модели.

Выбрать есть из чего:

метрики точности в задаче классификации поступающих записей бывают разные Точность, Полнота, их Гармоническое среднее, ROC-AUC

Для задачи регрессии: LogLoss, средний квадрат ошибок, коэффициент детерминации R^2

Для ассоциации это различные метрики: лифт, достоверность, поддержка

Какое значение выбранного ранее

критерия качества модели

мы будем считать оптимальным.

Например, считаем AUC равный 0.65 — минимальным порогом, 0.75 — оптимальным.

Если объективный критерий качества использовать не будем, то как будут оцениваться результаты

Далее необходимо составить план проекта.

Как только получены ответы на все основные вопросы и ясна цель проекта, время составить план проекта.

План должен содержать оценку всех шести фаз внедрения выбранной нами методологии, а именно   
Бизнес-анализ (Business understanding)  
Анализ данных (Data understanding)  
Подготовка данных (Data preparation)  
Моделирование (Modeling)  
Оценка результата (Evaluation)  
Внедрение (Deployment)

После проведения бизнес-аналитики и составления плана всех шести фаз методологии появляется возможность приступить к следующей стадии, а именно к стадии понимания данных.

**Понимание данных** (Data Understanding).

Данная фаза включает в себя более детальное изучение имеющихся данных.

Ее цель — избежать непредвиденных проблем на стадии подготовки данных, которая, как правило, является самой сложной частью проекта.

Начальное изучение данных предполагает организацию доступа к ним, их исследование с использованием таблиц и графиков, оценку качества данных и разработку соответствующей документации.

**Понимание данных** производится на этапе их аналитики базовыми средствами для определения необходимого перечня информации для решения конечной задачи, а также выработки требований к их качеству.

Данная стадия производится совместно инженерами и аналитиками данных, чтобы обеспечить все процессы консолидации данных и организации витрин.

На данной стадии мы решаем следующие задачи:

Сбор данных (Data collection)

Для начала нужно понимать, какими данными располагает заказчик. Данные могут быть:  
- собственные (1st party data),

- сторонние данные (3rd party),  
- «потенциальные» данные (для получения которых необходимо организовать сбор).  
  
Необходимо проанализировать все источники, доступ к которым предоставляет заказчик.

Если собственных данных недостаточно, возможно, стоит закупить сторонние или организовать сбор новых данных.

Данная стадия нами подробно разбиралась ранее в рамках данного курса.

Далее идет задача описания данных:

Описание данных (Data description)  
Далее смотрим на доступные нам данные.  
- Необходимо описать данные во всех источниках (таблица, ключ, количество строк, количество столбцов, объем на диске).

- Если объем слишком велик для используемого ПО, создаем сэмпл данных.

- Считаем ключевые статистики по атрибутам (минимум, максимум, разброс, кардинальность и т.д.).

После идет процесс исследования данных

Исследование данных (Data exploration)  
С помощью графиков и таблиц исследуем данные, чтобы сформулировать гипотезы относительно того, как эти данные помогут решить задачу.  
В мини-отчете фиксируем, что интересного нашли в данных, а также список атрибутов, которые потенциально полезны.

Крайним этапом анализа понимания данных проводится исследование качества данных

Важно еще до моделирования оценить качество данных, так как любые несоответствия могут повлиять на ход проекта. Какие могут быть сложности с данными?  
  
- Пропущенные значения.  
*К примеру, мы делаем модель классификации клиентов банка по их продуктовым предпочтениям, но, поскольку анкеты заполняют только клиенты-заемщики, атрибут «уровень з/п» у клиентов-вкладчиков не заполнен.*

- Ошибки данных (опечатки)  
- Несогласованная кодировка значений (например кодировка мужского пола символами «M» и «male» в разных системах)

В результате производим отчет о качестве данных по этапам, которые были оговорены ранее в рамках предыдущих лекций.

Процесс понимания данных постоянно ссылается на предыдущую стадию и требует постоянных уточнений по принципу улучшения понимания бизнес-идеи на основе новых поступающих данных о предметной области.

Полученная информация в результате понимания данных также провоцирует изменения в бизнес-анализ предметной области на основе новых результатов визуализации и качества данных.

На практике чаще всего происходит ситуация, когда первичные предположения о возможности влияния данных на улучшение бизнес-процессов не оправдываются и в таком случае необходимо поднимать тему изменения задач в рамках анализа данных.

По прошествии нескольких итераций, получив определенное представление о природе данных, получив описание и стабильный приток информации, инженеры данных формируют хранилище данных как операционные базы данных так и бизнес-хранилища и OLAP.

Следующей стадией после понимания данных становится стадия подготовки данных.

**Подготовка данных** (Data Preparation). Является одним из наиболее важных и зачастую трудоемких этапов аналитического проекта, который может поглощать 50-70% времени, усилий и ресурсов.

Цель этапа – подготовить обучающую выборку для использования в моделировании.

В зависимости от специфики компании и направления ее деятельности подготовка данных обычно включает:

консолидацию данных;

Задача консолидации данных и заключается в соблюдении ранее упомянутых свойств.

В процессе консолидации данных решаются следующие задачи

- выбор источников данных;

- разработка стратегии консолидации;

- оценка качества данных;

- очистка;

- перенос в хранилище данных.

формирование выборок;

Для начала нужно отобрать данные, которые мы будем использовать для обучения моделей.  
Отбираются как атрибуты, так и кейсы.  
  
*Например, если мы делаем продуктовые рекомендации посетителям сайта, мы ограничиваемся анализом только зарегистрированных пользователей.*  
При выборе данных аналитик отвечает на следующие вопросы.  
  
- Какова потенциальная релевантность атрибута решаемой задаче?  
*Так, электронная почта или номер телефона клиента как предикторы для прогнозирования явно бесполезны. А вот домен почты (mail.ru, gmail.com) или код оператора в теории уже могут обладать предсказательной способностью.*

- Достаточно ли качественный атрибут для использования в модели?  
*Если видим, что большая часть значений атрибута пуста, то атрибут, скорее всего, бесполезен.*

- Стоит ли включать зависящие друг от друга атрибуты?

- Есть ли ограничения на использование атрибутов?  
*Например, политика компании может запрещать использование атрибутов с персональной информацией в качестве возможных предикторов*.

Очистка данных:

Когда отобрали потенциально интересные данные, проверяем их качество.  
- Пропущенные значения => нужно либо их заполнить, либо удалить из рассмотрения

- Ошибки в данных => необходимо попробовать исправить вручную либо удалить из рассмотрения

- Несоответствующая кодировка => необходимо привести к единой кодировке

На выходе получается 3 списка атрибутов – качественные атрибуты, исправленные атрибуты и забракованные.

Часто вычисление признаков (feature engineering) – это наиболее важный этап в подготовке данных: грамотно составленный признак может существенно улучшить качество модели.

К генерации данных можно отнести:

- агрегацию атрибутов (расчет sum, avg, min, max, var и т.д.),

- конвертацию типов данных для использования в разных моделях (например, одни алгоритмы традиционно работают с интервальными данными, а другие с номинальными),

- нормализацию атрибутов (feature scaling), - процесс приведения измерений в диапазон значений, близкий друг другу, во избежание долгого обучения моделей или неадекватных метрик качества

- заполнение пропущенных данных (missing data imputation).

обогащение данных;

Хорошо, когда данные берутся из корпоративного хранилища (КХД) или заранее подготовленной витрины.

Однако часто данные необходимо загружать из нескольких источников и для подготовки обучающей выборки требуется их интеграция.

Под интеграцией понимается как «горизонтальное» соединение (Merge), так и  «вертикальное» объединение (Append), а также агрегация данных.

На выходе, как правило, имеем единую аналитическую таблицу, пригодную для поставки в аналитическое ПО в качестве обучающей выборки.

очистку данных;  
  
Наконец, нужно привести данные к формату, пригодному для моделирования (только для тех алгоритмов, которые работают с определенным форматом данных). *Так, если речь идет об анализе временного ряда – к примеру, прогнозируем ежемесячные продажи торговой сети – возможно, его нужно предварительно отсортировать.*

В результате этапа подготовки бизнес-данных к моделированию возникает задача формирования витрин данных.

Разберемся что такое витрины данных и для чего они необходимы

**Витрина данных** — это часть хранилища данных, секционированная для отделов или направлений бизнеса (например, продажи, маркетинг или финансы).

Витрина строится из данных, которые запрашиваются чаще других или нужны для выполнения бизнес-задач.

В одном хранилище данных часто развертывается несколько витрин.

Витрины данных хранятся в виде многомерной схемы, которая служит основой для анализа пользователями хранилища данных.

Существует три основных типа витрин данных.

Разница между ними определяется их отношением к хранилищу данных и источникам данных, использованным для их создания.

К ним относятся:

**зависимые** — зависят от информации, извлеченной из корпоративных хранилищ данных;

**независимые** — не связаны с хранилищем данных. Данные извлекаются из внутренних или внешних источников и загружаются в независимый репозиторий витрин данных.

**гибридные** — объединяют источники первичных данных из существующего хранилища данных и других внешних источников данных..

После загрузки данных в хранилище формируем витрины данных по следующему алгоритму:

Создаем представления данных (временные таблицы на основе существующих) из баз данных хранилища;

Представления данных (VIEW) — специальные конструкции в реляционных СУБД, позволяющие хранить предметно-ориентированные таблицы, составленные из исходной схемы базы данных с использованием запроса (SELECT).

Данные представления могут как храниться в виде запроса и исполняться «на лету», так и храниться в виде отдельной таблицы, связанной с исходными.

Вставка данных в представления может сопровождаться триггером на вставку данных в базу данных таблиц.

Определяем архитектуру схемы витрин

Две основные архитектуры витрин данных — это звезда, и снежинка.

Встречается также реализация в виде многомерного OLAP-куба.

Сегментируем данные по отделам

Запрещаем доступ к хранилищу данных для всех пользователей по умолчанию.

И далее предоставляем пользователям доступ только к тематическим витринам.

На этом шаге могут быть исключения — некоторым пользователям будет необходимо иметь права доступа как к хранилищу, так и к витринам.

Аналитические приложения при обработке конкретно *бизнес-данных* аналитики полагаются на OLAP хранилища. Для объяснения почему это происходит и что представляют собой OLAP хранилища.  
Начнем с определения многомерных данных

Чаще всего бизнес-данные, то есть исторические табличные данные представляют собой плоские таблицы, называемые многомерными данными

**Многомерные данные** – дополнительное определение для табличных данных с большим количеством категориальных атрибутов (столбцов).

Данные называются многомерными из-за большого числа факторов, по которым можно производить агрегацию данных.

Многомерные данные хранятся в OLAP системах денормализованных аналитических баз данных

вместо множества таблиц в хранилище данных, согласно правилам нормализации.

Многомерные данные чаще всего изображаются в виде кубиков, символизируя их многомерность.

На гранях куба при изображении рисуют различные измерения категориальных атрибутов **(измерение)**.

На пересечении атрибутов размещаются результаты вычисления **агрегационных функций над данными**,

в общем случае для того, чтобы привести информацию к построенной размерности **(мера)**

На слайде представлены многомерные таможенные данные о перевозках товаров из или в регионы зарубежья (Код страны) за границей России (Направление торговли) по времени.

Также приведен суммарный объем перевозок. Необходимо представить данные в виде трехмерного куба.

В данном случае нам остается только разместить на осях следующие координаты:

- Код страны,

- направление торговли,

- месяц и год.

А в качестве меры использовать сумму суммарного объема перевозки товаров по данным схожим тройкам категорий.

Вместо суммы можно использовать и другие агрегационные функции исходя из задачи анализа данных

Таким образом работает OLAP таблицы и данные в этих системах.

Построенное нами многомерное представление данных называется OLAP-куб.

Данный вид анализа позволяет с помощью агрегаций по многомерным категориальным измерениям динамически строить бизнес-анализ на данных не только с большим количеством категорий,

но и с большим количеством наблюдений, за счет агрегационных функций.

Степень агрегаций и их перечень также выбираются исходя из задачи анализа.

OLAP (Online Analytical Processing) – это система аналитической обработки данных.

Она предназначена для подготовки отчетов, построения прогностических сценариев и выполнения статистических расчетов.

Является классическим применением витрин данных и является продолжением конвейера обработки данных.

OLAP системы основаны на **аналитических базах данных**, которые оптимизируют запросы на доступ к данным с целью ускорения таких операций

**Аналитические базы данных (также именуемые OLAP базы данных)** — специализированные колоночные РСУБД, оптимизированные для быстрой выборки данных из витрин.

Они Основаны на системах массивных параллельных вычислений (MPP) и базах данных, поддерживающих такой режим работы.

Предназначены такие БД для обработки данных на стадии работы непосредственно с витринами больших данных.

Среди представителей OLAP систем выделяются GreenPlum и SnowFlake работающие по принципу массовых параллельных вычислений и хранящие данные в колоночном виде, что облегчает доступ к данным через выборку SELECT

**Колоночные СУБД** — системы управления базами данных в которых данные хранятся и индексируются столбцами.

Преимущества:

- Быстрая выборка данных

Для выборки в SELECT при перечислении столбцов необходимо просто взять выбранные колонки и соответствующие позиции в них после фильтрации.

- Более качественное сжатие данных в колонках ввиду однородности измерений

- Гибкость схемы данных

Поскольку данные хранятся в виде колонок, то добавление или удаление их из таблиц происходит быстро и по ссылке даже при больших объемах данных

Недостатки:

- Сложности при внесении новых данных

Необходимо сканирование колонки сверху вниз перед вставкой, а значит необходимо проделывать это каждый раз для каждого атрибута в

Не подходят для транзакционных систем ввиду сложности вставки наблюдения в таблицу.

OLTP — системы хранения оперативных данных с высокой скоростью записи данных и фиксации изменений, ввиду того, что данные хранятся классически по записям, то есть по строкам плоской таблицы.

Их основная цель - ввод/редактирование/удаление данных в режиме онлайн. Примеры использования: мессенджеры, социальные сети, 1С: Бухгалтерия и т.д.

Из объективно популярных представителей можно перечислить PostgreSQL и MySQL СУБД.

OLTP БД придерживаются принципа нормализации данных

OLAP (англ. Online Analytical Processing) или многомерные БД - это базы данных, которые служат непосредственно для проведения быстрого анализа больших объемов данных.

Обычно такие БД используются на больших предприятиях для построения аналитической отчетности за большой промежуток времени (месяц, квартал, год).

Такая информация в основном используется для анализа прошедшего периода и планирования будущего.

Основные пользователи аналитических данных – руководители, аналитики, инженеры машинного обучения.

OLAP системы по принципу организации витрин и хранилищ бывают разных видов:

ROLAP – реляционный OLAP который строится на основе обычных OLTP баз данных.

В реляционных OLAP-системах структура куба данных [4] хранится в реляционной базе данных.

Меры самого нижнего уровня остаются в реляционной витрине данных, служащей источником данных для куба.

Предварительно обработанные агрегаты также хранятся в реляционной таблице [3,5-9].

Когда человек, принимающий решение, запрашивает значение меры для определенного набора элементов измерения,

ROLAP-система проверяет, указывают ли эти элементы на агрегат или на значение самого нижнего уровня иерархии (листовое значение).

Если указан агрегат, то значение выбирается из реляционной таблицы. Если выбрано листовое значение, то значение берется из витрины данных.

Благодаря реляционным таблицам, архитектура ROLAP позволяет хранить большие объемы данных.

Поскольку в архитектуре ROLAP листовые значения берутся непосредственно из витрины данных,

то возвращаемые ROLAP-системой листовые значения всегда будут соответствовать актуальному на данный момент положению дел.

Другими словами, ROLAP-системы лишены запаздывания в части листовых данных.

Достоинства этого класса систем:

возможность использования ROLAP с хранилищами данных и различными OLTP-системами;

возможность манипулирования большими объемами данных; объем данных могут ограничивать только лежащие в основе ROLAP системы реляционных баз данных, подход ROLAP сам по себе не ограничивает объем данных;

безопасность и администрирование обеспечивается реляционными СУБД.

Недостатки:

* олучение агрегатов и листовых данных происходит медленнее, чем, например, в MOLAP и HOLAP (см. ниже);

функциональность систем ограничивается возможностями SQL, так как аналитические запросы пользователя транслируются в SQL-операторы выборки;

сложно пересчитывать агрегированные значения при изменениях начальных данных;

сложно поддерживать таблицы агрегатов.

MOLAP – строится в основном на OLAP движках, которые готовят данные специальным образом в форму денормализованных таблиц заранее

**MOLAP, Multidimensional OLAP – многомерный OLAP**

В многомерных OLAP-системах структура куба хранится в многомерной базе данных.

В той же базе данных хранятся предварительно обработанные агрегаты и копии табличных значений.

В связи с этим все запросы к данным удовлетворяются многомерной системой баз данных, что делает MOLAP-системы исключительно быстрыми.

Для загрузки MOLAP-системы требуется дополнительное время на копирование в многомерную базу всех листовых данных.

Поэтому возникают ситуации, когда листовые данные MOLAP-системы оказываются рассинхронизированными с данными в витрине данных.

Таким образом, MOLAP-системы вносят запаздывание в данные нижнего уровня иерархии.

Архитектура MOLAP требует большего объема дискового пространства из-за хранения в многомерной базе копий листовых данных.

Но, несмотря на это, объем дополнительного пространства обычно не слишком велик, поскольку данные в MOLAP хранятся исключительно эффективно.

Достоинства MOLAP-систем:

все данные хранятся в многомерных структурах, что существенно повышает скорость обработки запросов;

доступны расширенные библиотеки для сложных функций оперативного анализа;

обработка разреженных данных выполняется лучше, чем в ROLAP.

Недостатки:

данные куба «оторваны» от базовой таблицы; необходимы специальные инструменты для формирования кубов и их пересчёта в случае изменения базовых значений;

сложно изменять измерения без повторной агрегации.

HOLAP – гибридный и самый оптимальный подход во множестве приложений

**HOLAP, Hybrid OLAP – гибридный OLAP**

В гибридных OLAP сочетаются черты ROLAP и MOLAP, отсюда и название – гибридный.

В моделях HOLAP используются преимущества и минимизируются недостатки обеих архитектур [3,5-8].

В HOLAP-системах структура куба и предварительно обработанные агрегаты хранятся в многомерной базе данных.

Это позволяет обеспечить быстрое извлечение агрегатов из структур MOLAP.

Значения нижнего уровня иерархии в HOLAP остаются в реляционной витрине данных, которая служит источником данных для куба.

HOLAP не требует копирования листовых данных из витрины, хотя это и ведет к увеличению времени доступа при обращении к листовым данным.

Данные в витрине доступны аналитику сразу после обновления.

Таким образом, HOLAP-системы не вносят запаздывания в работу с данными нижнего уровня иерархии.

По сути, HOLAP жертвует скоростью доступа к листовым данным ради устранения запаздывания при работе с ними и ускорения загрузки данных.

В связи с этим HOLAP проигрывает по скорости MOLAP.

К достоинствам подхода можно отнести комбинирование технологии ROLAP для разреженных данных и MOLAP для плотных областей, а к недостаткам – необходимость поддерживания MOLAP и ROLAP.

Таким образом, на стадии подготовки данных мы получили возможность провести статистическое моделирование на основе алгоритмов на основе данных, так как уже подготовили для этого витрину и консолидировали наше хранилище.

Рассмотрим следующий этап CRISP-DM

**Моделирование** (Modeling).

На данном этапе строятся и внедряются аналитические модели.

Моделирование обычно проводится в несколько итераций.

Сначала запускается несколько моделей с параметрами по умолчанию.

Затем параметры настраиваются таким образом, чтобы модель выполняла требуемую обработку данных.

Если это не удается, приходится возвращаться на этап подготовки данных и вносить изменения.

**Моделирование на основе данных** – это область, которая занимается созданием математических и статистических моделей, которые используются для прогнозирования или определения определенных результатов на основе имеющихся данных.

Это ключевой аспект в области анализа данных и машинного обучения, который позволяет нам делать обоснованные предположения или принимать решения на основе прошлых данных.

Цели и задачи моделирования на основе данных могут быть различными, в зависимости от конкретной области применения, но в целом они включают следующее:

Одной из основных целей моделирования на основе данных является предсказание будущих результатов на основе исторических данных.

Это может включать в себя предсказание продаж для определенного продукта, прогнозирование поведения клиентов или прогнозирование тенденций рынка.

Моделирование на основе данных также помогает организациям принимать более обоснованные решения.

Например, компания может использовать данные для определения,

какие продукты наиболее популярны среди клиентов, и затем использовать эту информацию для определения,

какие продукты следует производить больше.

Моделирование на основе данных может помочь организациям оптимизировать свои процессы и повысить эффективность.

Например, компания может использовать данные для анализа производительности своего производственного оборудования и затем использовать эту информацию для определения,

как можно улучшить процесс производства.

Другой важной задачей моделирования на основе данных является распознавание шаблонов и тенденций в данных.

Это может помочь организациям выявить новые возможности или проблемы, которые могут возникнуть в будущем.

Наконец, моделирование на основе данных может помочь организациям разрабатывать новые продукты или услуги.

Например, компания может использовать данные для определения, какие функции наиболее ценятся клиентами,

и затем использовать эту информацию для разработки новых продуктов, которые удовлетворяют этим потребностям.

Чисто статистические модели не решают задачи обучения моделей, принимающих решения на основе данных, а занимаются проектированием алгоритмов для генерации новых данных для систем поддержки принятия решения или рекомендаций, среди областей использующих такие модели выделяют:

Сегмент клиентской аналитики использует различные детерминированные алгоритмы вычисления метрик отношения клиента к сети магазинов или метрик товарной аналитики.

Часто применяемые модели:

- RFM (Recency, Frequency, Monetary) модель: Эта модель используется для анализа поведения клиента, основываясь на том, как недавно, как часто и сколько денег клиент тратит.

ABC анализ (Activity Based Costing) - метод классификации запасов на основе их значимости. Обычно используется для определения, какие товары приносят наибольшую прибыль. "A" - это товары с наибольшей прибылью, "B" - средней прибылью, "C" - наименьшей.

XYZ анализ - метод классификации запасов на основе их предсказуемости. "X" - это товары с предсказуемым спросом, "Y" - с частично предсказуемым спросом, "Z" - с непредсказуемым спросом.

VEN анализ (Vital Essential Non-essential) - метод классификации запасов на основе их важности для процесса производства или обслуживания. "V" (vital) - жизненно важные товары, без которых невозможно обслуживание или производство, "E" (essential) - важные товары, но их отсутствие не остановит процесс, "N" (non-essential) - неважные товары, которые не оказывают значительного влияния на процесс.

Эти методы помогают компаниям оптимизировать управление запасами и улучшить стратегию продаж

Банковский сектор (скоринговые модели)

Логистика (кратчайшие пути, динамическое программирование)

Модели машинного обучения используются для построения моделей, производящих новые знания из существующих табличных данных.

Такие модели обучаются самостоятельно

Модели глубокого обучения решают также задачи извлечения знаний на основе данных, не являющихся структурированными, например фото, видео, текст или аудио

Обучение с учителем и без учителя являются двумя основными типами алгоритмов машинного обучения.

Обучение с учителем (Supervised Learning):

В этом подходе, модель обучается на основе обучающего набора данных, который уже содержит ответы, то есть он имеет и входные данные и соответствующие им целевые значения (метки).

Задача модели - "выучить" зависимости между входными и выходными данными, чтобы затем использовать эту информацию для предсказания ответа на новые, ранее неизвестные данные. Примерами задач обучения с учителем являются классификация (где модель предсказывает категорию объекта) и регрессия (где модель предсказывает непрерывное значение).

Обучение без учителя (Unsupervised Learning):

В этом подходе, модель обучается на наборе данных без каких-либо предварительно известных ответов или меток.

Задача модели - найти скрытые структуры или зависимости в данных.

Обучение без учителя часто используется для задач кластеризации (где модель группирует данные по определенным признакам), снижения размерности (где модель упрощает данные без значительной потери информации) или ассоциации (где модель извлекает полезные признаки из данных).

Важно отметить, что выбор между обучением с учителем и без учителя зависит от конкретной задачи, доступности размеченных данных и целей исследования.

В некоторых случаях также используются комбинированные подходы, например, полу-надзорные обучение (где часть данных размечена, а часть - нет) или обучение с подкреплением (где модель учится на основе взаимодействия с окружающей средой).

**Обучение с учителем –** получение модели путем обучения на выборках с заранее заданными ответами.

Модель изменяет свои ответы в процессе тренировки на основе корректировки ответов по таблице известных ответов.

Модель изменяет свои ответы в процессе тренировки на основе корректировки ответов по таблице известных ответов.  
  
Обучение с учителем - это один из основных видов машинного обучения, где модель обучается на основе заранее подготовленного набора данных,

который содержит входные данные и соответствующие им правильные ответы (целевые значения).

Этот набор данных называется обучающей выборкой.

На слайде изображен пример по которому можно понять, что обучение происходит итерационно по таблице данных,

где в начале мы получаем ответы на некоторые известные входы из обучающей выборки,  
а затем на основе вычисления ошибки между верным ответом и ответом модели изменяем параметры модели.

Задача обучения с учителем состоит в том, чтобы построить модель, которая будет предсказывать целевые значения на новых, ранее неизвестных данных, опираясь на обученные в процессе обучения закономерности.

В зависимости от типа целевых значений, задачи обучения с учителем делятся на задачи классификации (когда целевое значение является категориальной переменной, например, "да" или "нет") и задачи регрессии (когда целевое значение - это непрерывная величина, например, цена дома).

Классификация: Компьютерная программа должна определить, к какой из k категорий относится некоторый входной сигнал.

Для решения этой задачи алгоритмы обучения обычно выводят функцию 𝑓: ℝ^𝑛→(1, 2, …𝑘).

Например, алгоритм классификации изображений в компьютерном зрении разработан для решения задач классификации.

Для решения задачи классификации применяются специальные алгоритмы среди которых:

- Метод К ближайших соседей

- Наивный байесовский метод

- Логистическая регрессия

- Метод опорных векторов

- Деревья решений

- Случайный лес

- Градиентный бустинг

Регрессия: Для этого типа задач компьютерная программа предсказывает выход для заданного входа.

Алгоритмы обучения обычно выдают функцию 𝑓: ℝ^𝑛→ℝ.

Примером такого типа задач является прогнозирование суммы страхового возмещения (для установления страховой премии) или прогнозирование цены ценной бумаги.

**Регрессия может быть решена одним из методов:**

**- Линейная регрессия**

**- Обобщенные регрессионные модели**

**- Полиномиальная регрессия**

**- Случайный лес**

**- Градиентный бустинг**

**Классификация** и **регрессия** - два основных вида прогнозирования, на долю которых приходится от 80% до 90%.

Результатом классификации являются дискретные значения категорий, а результатом регрессии - непрерывные числа.

**Среднеквадратичная ошибка (Mean Squared Error)** применяется в случаях, когда требуется подчеркнуть большие ошибки и выбрать модель, которая дает меньше именно больших ошибок. Большие значения ошибок становятся заметнее за счет квадратичной зависимости.

Действительно, допустим модель допустила на двух примерах ошибки 5 и 10. В абсолютном выражении они отличаются в два раза, но если их возвести в квадрат, получив 25 и 100 соответственно, то отличие будет уже в четыре раза.

Таким образом модель, которая обеспечивает меньшее значение *MSE* допускает меньше именно больших ошибок.

*MSE* рассчитывается по формуле:

где *n* — количество наблюдений по которым строится модель и количество прогнозов, *yi*​ — фактические значение зависимой переменной для *i*-го наблюдения, ^*y*​*i*​ — значение зависимой переменной, предсказанное моделью.

Таким образом, можно сделать вывод, что *MSE* настроена на отражение влияния именно больших ошибок на качество модели.

Недостатком использования *MSE* является то, что если на одном или нескольких неудачных примерах, возможно,

содержащих аномальные значения будет допущена значительная ошибка, то возведение в квадрат приведёт к ложному выводу, что вся модель работает плохо.

С другой стороны, если модель даст небольшие ошибки на большом числе примеров, то может возникнуть обратный эффект — недооценка слабости модели.

**MAE**

**Cредняя абсолютная ошибка (Mean Absolute Error)** вычисляется следующим образом:

Т.е.  *MAE* рассчитывается как среднее абсолютных разностей между наблюдаемым и предсказанным значениями.

В отличие от *MSE* она является линейной оценкой, а это значит, что все ошибки в среднем взвешены одинаково.

Например, разница между 0 и 10 будет вдвое больше разницы между 0 и 5. Для *MSE*, как отмечено выше, это не так.

Поэтому *MAE* широко используется, например, в финансовой сфере, где ошибка в 10 долларов должна интерпретироваться как в два раза худшая, чем ошибка в 5 долларов.

Перечисленные выше ошибки не так просто интерпретировать.

Действительно, просто зная значение средней абсолютной ошибки, скажем, равное 10, мы сразу не можем сказать хорошая это ошибка или плохая, и что нужно сделать чтобы улучшить модель.

В этой связи представляет интерес использование для оценки качества регрессионной модели не значения ошибок, а величину показывающую, насколько данная модель работает лучше, чем модель,

В которой присутствует только константа, а входные переменные отсутствуют или коэффициенты регрессии при них равны нулю.

Именно такой мерой и является **коэффициент детерминации (Coefficient of determination)**,

который показывает долю дисперсии зависимой переменной, объяснённой с помощью регрессионной модели.

Наиболее общей формулой для вычисления коэффициента детерминации является следующая:

Практически, в числителе данного выражения стоит среднеквадратическая ошибка оцениваемой модели, а в знаменателе — модели, в которой присутствует только константа.

Главным преимуществом коэффициента детерминации перед мерами, основанными на ошибках, является его инвариантность к масштабу данных.

Кроме того, он всегда изменяется в диапазоне от −∞ до 1.

При этом значения близкие к 1 указывают на высокую степень соответствия модели данным.

Очевидно, что это имеет место, когда отношение в формуле стремится к 0, т.е. ошибка модели с переменными намного меньше ошибки модели с константой.

*R*2=0 показывает,

что между независимой и зависимой переменными модели имеет место функциональная зависимость.

На практике используют следующую шкалу оценок.

Модель, для которой *R*2>0.5, является удовлетворительной.

Если *R*2>0.8, то модель рассматривается как очень хорошая.

Значения, меньшие 0.5 говорят о том, что модель плохая.

Прежде чем переходить к описанию собственно метрик качества бинарных классификаторов, рассмотрим методику описания этих метрик в терминах ошибок классификации.

Пусть заданы два класса *y*={0,1} и алгоритм, предсказывающий принадлежность каждого объекта одному из классов.

Эта задача анализа известна как бинарная классификация.

Любой реальный классификатор совершает ошибки. В нашем случае таких ошибок может быть две:

класс 0 распознается классификатором как класс 1, что можно интерпретировать как «ложную тревогу»;

класс 1 распознается как класс 0, что можно трактовать как «пропуск цели».

Тогда возможны следующие исходы классификации:

Объект, нежелательный для страхования, классифицирован как нежелательный, т.е. «положительный» класс распознан как положительный.

Такой исход классификации (а также пример, для которого он получен) называют истинноположительным или TP.

Объект, желательный для страхования, распознан как желательный, т.е. «отрицательный» класс распознан как отрицательный.

Такой исход классификации называют истинноотрицательными или TN.

Объект, желаемый для страхования, классифицирован как не желаемый, т.е. имела место ошибка, в результате которой отрицательный класс был распознан как положительный.

Данный исход классификации называют ложноположительным, а ошибка классификации называется ошибкой I рода или FP.

Нежелательный объект распознан как желательный, т.е. имела место ошибка, в результате которой положительный класс был распознан как отрицательный.

Такой исход классификации называется ложноотрицательным, а ошибка классификации — ошибкой II рода. FN

Идеальный классификатор, если бы он существовал, выдавал бы только истиннополо­жительные и истинноотрицательные классификации,

и его матрица ошибок содержала бы значения, отличные от нуля, только на главной диа­гонали.

Основные метрики качества классификации включают в себя следующие:

Меткость Accuracy: Представляет собой долю правильных классификаций модели:

Точность Prcision равна доле истинноположительных классификаций к общему числу положительных классификаций.

Данная величина часто упоминается как positive predictive value (PPV) или положительное прогностическое значение

Полнота Recall, известная еще как чувствительность или доля истинноположительных примеров (TPR — true positive rate),

определяется как число истинноположительных классификаций относительно общего числа положительных наблюдений

Чем выше точность и полнота, тем лучше модель.

Но на практике их максимальные значения одновременно недостижимы, поэтому приходится искать баланс между ними.

Для этого используется метрика, объединяющая в себе информацию о точности и полноте. Она называется F1-мера и вычисляется следующим образом

Также существуют и другие метрики классификации среди которых:

AUC-ROC

AUC-PR

P4

Обучение без учителя - это тип алгоритма машинного обучения, который используется для анализа и нахождения закономерностей в наборах данных, которые не имеют каких-либо конкретных целевых переменных или меток.

Иными словами, эти алгоритмы позволяют системе учиться и делать выводы из набора данных без какого-либо предварительного обучения.

Основная задача обучения без учителя - это нахождение структуры в данных.

Это может включать в себя группировку данных на основе сходства (кластеризация), снижение размерности на основе внутренних зависимостей в данных (уменьшение размерности), а также выявление пар или цепочек наиболее вероятных исходов в транзакционных данных (ассоциация)

Задача кластеризации данных - это процесс группировки набора объектов таким образом, чтобы объекты в одной группе (кластере) были более похожи друг на друга по определенным критериям, чем на объекты в других группах (кластерах). Это основная задача обучения без учителя в машинном обучении.

Кластеризация может быть использована для статистического анализа данных, где целью может быть идентификация шаблонов или трендов в данных.

Кроме того, она может быть использована для сегментации рынка в маркетинге, где целью может быть идентификация групп потребителей с похожими предпочтениями или характеристиками.

Ассоциация в транзакционных данных относится к процессу нахождения интересных паттернов, ассоциаций, корреляций или структур среди наборов элементов в транзакционных базах данных, реляционных базах данных или других информационных хранилищах.

Задача ассоциации включает в себя анализ транзакционных данных и выявление закономерностей и связей между различными элементами. Это может быть использовано для прогнозирования поведения покупателей, определения стратегий продаж, управления запасами и т.д.

Задача уменьшения размерности в машинном обучении и статистике – это процесс преобразования данных из высокоразмерного пространства в пространство меньшей размерности с целью упрощения анализа данных, ускорения обучения моделей, уменьшения шума и избыточности информации, улучшения интерпретируемости моделей и визуализации данных.

Примером может служить анализ большого набора данных с множеством переменных (признаков), где многие из этих переменных могут быть коррелированы.

Уменьшение размерности может позволить упростить данные, удалив избыточные или неинформативные признаки, при этом сохраняя основную структуру и информацию.

Существуют различные методы уменьшения размерности, включая линейные методы, такие как анализ главных компонент (PCA), и нелинейные методы, такие как t-SNE или UMAP.

Для решения задач извлечения информации из неструктурированных данных применяют методы глубокого обучения:

Компьютерное зрение (Computer Vision)

Глубокие модели обработки естественного языка (Natural Language Processing)

Задачи обработки голоса

**Задачи обработки текста** заключаются в преобразовании текстовой информации в удобный для анализа данных вид структурированной или векторизованной информации.

* Структуризация текстовых данных
* Векторизация текста
* Генерация признаков на основе текста
* Стемминг, лемматизация
* Очистка текста от ошибок

**Задачи анализа текста:**

* Сбор и классификация отзывов
* Анализ тональности текста
* Поиск смысла в тексте
* Кластеризация документов
* Генерация текста на заданную тему
* Автодополнение поисковых запросов
* Поиск именованных сущностей, ключевых слов
* Сегментация текста

**Популярные решения:**

GitHub Copilot,

Яндекс, Mail, DuckDuckGo

Алиса, Маруся, Салют

**Задачи для данных изображений и видео**

**Бинаризация**: преобразует изображение в серых тонах в бинарное (белые и черные пиксели);

**Сегментация**: используется для поиска и/или подсчета деталей;

**Чтение штрихкодов**: декодирование 1D и 2D кодов, разработанных для считывания или сканирования машинами;

**Оптическое распознавание символов**: автоматизированное чтение текста, например, серийных номеров;

**Обнаружение краев**: поиск краев объектов

**Сопоставление шаблонов**: поиск, подбор, и/или подсчет конкретных моделей.

Встречаются следующие задачи:

очистка от шумов;

выявление споттера (слово-триггер обычно для голосовых ассистентов);

разделение дорожек на речь и фоновый шум;

разделение дорожек на речи разных людей;

синтез голоса;

генерация новых аудиозаписей;

выявление акцента / языка.

 Если на этапе моделирования не удается построить требуемую по качеству модель, то приходится возвращаться на этап подготовки данных и вносить изменения.

**Оценка** (Evaluation). На этом этапе делается оценка того, соответствуют ли результаты проекта критериям успеха бизнеса.

Этот шаг требует четкого понимания заявленных бизнес-целей, поэтому нужно обязательно привлекать к нему ключевых лиц компании, принимающих решения.

Результатом предыдущего шага является построенная математическая модель (model), а также найденные закономерности (findings). На пятом шаге мы оцениваем результаты проекта.

На этапе оценки качества решений приводим аналитическую отчетность о качестве решения в формате пунктов бизнес-целей

Цель ведения аналитической отчетности — обнаружить проблему или возможность и объяснить, как построенная модель влияет на организацию, и как организация должна реагировать.

Во многих случаях также ожидают рекомендацию, основанную на проведенном анализе.

Примеры аналитической обработки данных:

* инвестиционный анализ;
* финансовый анализ показателей деятельности;
* анализ вероятности банкротства;
* ABC -анализ;
* сегментный анализ;
* факторный анализ.​

Чаще всего аналитическая отчетность строится на основе визуализаций в системах BI

На этапе оценки решения мы задаем вопросы самому первому этапу, сравниваем полученные результаты с эталонными.  
В случае, если построенная модель показала себя с неуспешной стороны, то мы повторяем пункты по мере сложности проблемы

**Внедрение** (Deployment). Внедрение — это процесс использования новых идей и знаний для повышения эффективности компании.

Приветствую вас уважаемые учащиеся и коллеги,

Мы продолжаем разбираться с основами больших данных на нашем курсе.

Наша сегодняшняя тема посвящена технологиям визуализации больших данных

В данной лекции мы рассмотрим виды визуализации, чем отличается базовая визуализация от специальной

Увидим возможности отображения аналитики на картах

Рассмотрим варианты построения дашбордов

Попробуем разобраться в необходимости применения BI систем

Прежде всего, нужно знать, что же такое визуализация данных и какие ее методы используются, в том числе и в повседневной жизни.

Визуализация данных — это наглядное представление массивов различной информации.

В аналитических технологиях визуализация — комплекс методов представления исходной информации и результатов анализа данных в наиболее удобной для восприятия и интерпретации форме.

Кроме этого, визуализация может применяться для мониторинга процесса построения и работы различных аналитических моделей, проверки гипотез и других целей, связанных с проведением анализа.

Визуальная информация лучше воспринимается и позволяет быстро и эффективно донести до зрителя собственные мысли и идеи.

Физиологически, восприятие визуальной информации является основной для человека.

Существует множество способов визуализации:

- Общая визуализация — обычное визуальное представление количественной информации в схематической форме. К этой группе можно отнести всем известные круговые и линейные диаграммы, гистограммы и спектрограммы, таблицы и различные точечные графики.

- Специальная визуализация — использует специфические формы представления информации — карты и полярные графики, графики с параллельными осями, диаграммы Эйлера и др.

- Концептуальная визуализация — позволяет разрабатывать сложные концепции, идеи и планы с помощью концептуальных карт, диаграмм Ганта, графов с минимальным путем и других подобных видов диаграмм.

- Стратегическая визуализация — переводит в визуальную форму различные данные об аспектах работы организаций. Это всевозможные диаграммы производительности, жизненного цикла и графики структур организаций.

- Метафорическая визуализация — используется для представления информации в виде геометрических фигур и их композиций (например, значения признака представляются кругами разного размера).

- Комбинированная визуализация позволяет объединить несколько сложных представлений в одну схему.

Разведочный анализ — это предварительный анализ данных с целью выявления наиболее общих зависимостей, закономерностей и тенденций, характера и свойств анализируемых данных, законов распределения анализируемых величин.

Применяется для нахождения связей между переменными в ситуациях, когда отсутствуют (или недостаточны) априорные представления о природе этих связей.

Как правило, при разведочном анализе учитывается и сравнивается большое число признаков, а для поиска закономерностей используются самые разные методы.

Термин «разведочный анализ» был впервые введен математиком из Принстонского университета Дж. Тьюки. Он также сформулировал основные **цели** данного анализа:

- Максимальное «проникновение» в данные.

- Выявление основных структур.

- Выбор наиболее важных переменных.

- Обнаружение отклонений и аномалий.

- Проверка основных гипотез (предположений).

- Разработка начальных моделей.

Результаты разведочного анализа не используются для выработки управленческих решений. Их назначение — помощь в разработке наилучшей стратегии углубленного анализа, выдвижение гипотез, уточнение особенностей применения тех или иных математических методов и моделей. Без разведочного анализа углубленный анализ данных будет производиться практически «вслепую».

К основным методам разведочного анализа относится процедура анализа распределений переменных, корреляционный анализ c целью поиска коэффициентов,

превосходящих по величине определенные пороговые значения, факторный анализ, дискриминантный анализ, многомерное шкалирование, визуальный анализ гистограмм и т.д.

Предварительное исследование данных может служить лишь первым этапом в процессе их анализа, и пока результаты не подтверждены на других выборках или на независимом множестве данных, их следует воспринимать самое большее как гипотезу.

Если результаты разведочного анализа говорят в пользу некоторой модели, то ее правильность можно затем проверить, применив к новым данным.

Столбчатая диаграмма - это вид диаграммы, который используется для представления сравнительных данных в визуально удобочитаемом формате.

Она состоит из вертикальных или горизонтальных столбцов (или баров), длина которых пропорциональна представляемым значениям.

Каждый столбец в диаграмме представляет определенную категорию данных, и высота или длина столбца соответствует значению этой категории.

Столбцы могут быть цветными для лучшего различия между разными категориями.

Основная концепция столбчатой диаграммы заключается в том, чтобы предоставить четкое и наглядное представление о сравнении между различными пунктами данных или переменными.

Это помогает в анализе и интерпретации данных, облегчает их понимание и делает информацию более доступной.

Столбчатые диаграммы часто используются в статистике, научных исследованиях, бизнесе и экономике для визуализации данных и сравнения показателей.

Они могут быть особенно полезны при работе с большими объемами данных или при необходимости сравнения нескольких наборов данных.

Столбчатые диаграммы могут быть одиночными (с одним набором данных), групповыми (с несколькими наборами данных, представленными в виде групп столбцов) или сложенными (где столбцы разделены на сегменты, представляющие подкатегории данных).

Важно убедиться, что данные корректно представлены на столбчатой диаграмме, чтобы избежать искажения информации. Это включает в себя правильное масштабирование осей, выбор подходящего типа диаграммы и обеспечение точности и полноты данных.

Круговая диаграмма, также известная как круговой график или pie chart, это тип графического представления, которое используется для иллюстрации числовых пропорций в наборе данных. Круговая диаграмма представляет собой круг, разделенный на секторы, каждый из которых представляет процентное соотношение общего количества.

Концепция круговой диаграммы основана на визуализации данных в виде "кусочков пирога", где размер каждого "кусочка" соответствует его доле от общего значения.

Вот несколько ключевых аспектов концепции круговой диаграммы:

1. Процентное представление: Круговые диаграммы обычно используются для представления процентного распределения различных категорий в общем количестве.

2. Визуализация данных: Они предлагают визуальный способ сравнения пропорций и понимания данных в целом.

3. Размер секторов: Размер каждого сектора прямо пропорционален величине данных, которые он представляет.

4. Использование цвета: Цвета часто используются для различения между разными секторами и помогают в интерпретации диаграммы.

5. Сумма всех секторов: Сумма процентов всех секторов на круговой диаграмме всегда равна 100%.

Круговые диаграммы широко используются в бизнесе, статистике, исследованиях и образовании для представления долей или процентов, таких как распределение доходов, демографические данные или результаты опроса. Однако они могут быть менее эффективными при сравнении относительных размеров разных категорий, особенно когда количество категорий увеличивается или когда значения очень близки друг к другу.

Этот вид древовидной диаграммы, описанный вами, называется "диаграмма дерева" или "треемап".

Это эффективный метод визуализации иерархических данных, где каждое измерение представлено в виде прямоугольника, а размер каждого прямоугольника определяется значением показателя.

В отличие от традиционных деревьев, которые растут вертикально, диаграмма дерева растет горизонтально.

Это позволяет легко визуализировать большие объемы данных без потери деталей и сохраняя пространство на дашборде.

При этом каждый уровень иерархии представлен цветным прямоугольником, что облегчает восприятие информации.

Вложенные прямоугольники представляют подкатегории данных. Чем больше значение показателя, тем больше размер соответствующего прямоугольника.

Таким образом, треемап позволяет быстро и наглядно увидеть распределение данных по категориям и подкатегориям, выделить наиболее значимые элементы и увидеть общую картину.

Например, с помощью древовидной диаграммы можно показать продажи в разрезе брендов продукта и способа доставки товаров, как это показано на слайде.

Размером в данной древовидной диаграмме служит количество клиентов магазина за весь период наблюдения, а цветом указаны суммарные продажи в рублях от синего (маленькие значения) к красному (большие).

На основе данной древовидной диаграммы можно сделать вывод о том, что бренд разумный дом, например, является одним из самых результативных.

Гистограмма представляет собой диаграмму (обычно столбиковую), которая используется в статистике для графического представления распределения вероятностей значений случайной величины.

По горизонтальной оси гистограммы откладывается диапазон наблюдаемых значений величины, разбитый на определенное число (обычно 10-15) интервалов, а по вертикальной — вероятность или частота ее попадания в каждый интервал. Тогда столбик будет отражать значения этих показателей для интервала, на который он опирается.

Чтобы построить гистограмму частот, просто подсчитывают, сколько раз значение случайной величины попало в каждый интервал.

Для перехода к относительным частотам достаточно разделить количество значений в каждом интервале на общее число наблюдений.

Очевидно, что в этом случае сумма всех столбцов гистограммы будет равна 1.

На графике же на слайде изображена гистограмма плотности вероятности появления определенного значения кредитного скора.

Оно получено в результате деления относительных частот еще и на ширину столбца столбчатой диаграммы для того, чтобы получить площадь под графиком, равную единице.

Такое построение гистограммы позволяет нам судить об отношении распределения одномерной величины к одному из известных дискретных или непрерывных распределений и возможно визуально сравнив с ними,

оценив их параметры на основе данной выборки.   
Например, если все столбцы гистограммы примерно одинаковы, то равномерному, если в виде симметричного «холма», то нормальному, и т.д.

По гистограмме частот на рисунке можно предположить, что случайная величина x описывается законом, близким к нормальному, и имеет наиболее вероятное значение, лежащее в пределах 600-700.

Достаточно вероятными будут значения из интервала 550-750 и очень маловероятными — меньше 500 и больше 800.

Иногда применяют так называемую **кумулятивную гистограмму**, или **гистограмму с накоплением**, в которой каждый столбец содержит в себе сумму частот предыдущих.

График «ящик с усами» позволяет очень компактно и наглядно представлять порядковые статистики одномерного закона распределения: квартили, медиану, наблюдаемые минимальное и максимальное значение выборки, а также отображать выбросы.

При использовании «ящика с усами» не делается никаких предположений относительно закона распределения выборки, поэтому его можно рассматривать как инструмент непараметрической статистики.

График «ящик с усами» был предложен Джоном Тьюки (поэтому в литературе её иногда называют ещё диаграммой Тьюки) в 1969 году.

Хотя «ящик с усами» считается менее информативным, чем гистограмма, но при этом является более компактным и простым в построении.

В основе графической структуры «ящика с усами» лежит прямоугольник (ящик) из боковых сторон которого отходят отрезки (усы).

Если с помощью графика исследуется одна выборка, то «ящик» обычно изображается горизонтально.

Если требуется визуализировать статистики для нескольких выборок с целью их сравнения, то «ящики» для каждого из них изображаются вертикально рядом друг с другом.

Изначально «ящик с усами» задумывался как способ представления так называемой **сводки пяти чисел** (five-number summary) — набора описательных статистик, характеризующих распределение исследуемой выборки.

Сводка включает следующие элементы, отображаемые с помощью «ящика с усами»:

- минимальное наблюдаемое значение (0-й квартиль или 0-й процентиль) — минимальная точка данных выборки, отображается началом левого или нижнего «уса»;

- максимальное наблюдаемое значение выборки (4-й квартиль или 100-й процентиль) — максимальная точка данных выборки, отображается концом правого или верхнего «уса»;

- медиана (2-й квартиль или 50-процентиль) — отображается чертой, разделяющей «ящик» на две части;

- 1-й квартиль или 25-й процентиль — представляется левой или нижней стороной «ящика»;

- 3-й квартиль или 75-й процентиль — представляется правой или верхней стороной «ящика»;

Таким образом длина «ящика» представляет собой интерквартильный размах, который составляет величину разницы между 1 и 3 квартилями, 25 и 75 процентилями.

График ящика с усами является специализированной визуализацией, которую необходимо строить в программных средствах.

Длина «усов» на диаграмме характеризует разброс (вариацию) значений выборки.

Расстояние между концами «усов», таким образом, представляет собой размах вариации.

Если длина «усов» одинаковая, это говорит в пользу того, что распределение выборки симметричное.

Дисбаланс между длинами «усов» говорит об асимметрии распределения.

Отметки, отображаемые на диаграмме кружками или звёздочками, представляют собой выбросы данных.

Диаграмму гистограммы и ящика с усами для одномерных данных можно хорошо сопоставить, на основе чего сравнить показатели основных пяти характеристик ящика с визуализацией распределения величины.

На основе данного сопоставления можно сделать более качественный вывод о предполагаемом распределении данных

Часто диаграмму «ящик с усами» строят вертикально, представляя несколько «ящиков» рядом для разных выборок.

Это очень удобно для сравнения статистических характеристик нескольких выборок.

Можно, например, сравнить медианы и квартили на визуально значимые отличия

Следует отменить, что каких-либо строгих правил или стандартов, регламентирующих построение диаграммы «ящик с усами», не существует.

Например, в некоторых случаях могут не изображаться «усы» или выбросы, если представляемая ими информация о распределении не существенна для решаемой задачи.

Диаграмма рассеяния — это графическое представление данных в двумерном пространстве, которое используется для иллюстрации взаимосвязи между двумя переменными.

Каждая точка на диаграмме представляет отдельное наблюдение и определяется значениями двух переменных: одной по горизонтальной оси (ось X) и другой по вертикальной оси (ось Y).

Диаграмма рассеяния позволяет наглядно увидеть распределение данных, выявить выбросы, оценить характер связи между переменными (линейную, нелинейную, отсутствие связи) и сделать предварительные выводы о возможности построения регрессионной модели.

На диаграмме рассеяния можно увидеть различные тренды и шаблоны.

Если точки располагаются вдоль прямой линии, это указывает на сильную корреляцию между переменными.

Если точки распределены хаотично, это указывает на отсутствие или слабую корреляцию.

Диаграммы рассеяния широко используются в статистике, анализе данных, машинном обучении и других областях, где требуется визуализация данных и анализ взаимосвязей между переменными.

По степени разброса точек от, например модели регрессии, можно оценить её точность.

Если большая часть точек сосредоточена вдоль идеальной линии, а значительные отклонения редки или совсем отсутствуют, то модель работает хорошо.

Если разброс точек велик, то ее точность низкая.

График разброса может быть с большим успехом применен в визуализации данных кластеров текстов или векторных представлений слов в пространстве сниженной размерности в задачах обработки текста.

Векторные языковые модели, такие как Word2Vec или GloVe, используются для преобразования слов в численные векторы.

Эти векторы затем можно использовать для анализа и визуализации отношений между словами в тексте.

График разброса может быть полезен при визуализации этих векторов. Вот как это может быть сделано:

1. Обучение модели: Сначала требуется обучить векторную языковую модель на вашем текстовом корпусе или взять готовую языковую модель.

По результатам этого этапа становится возможным преобразование каждого слова в набор чисел, представляющих его "смысл" в контексте других слов.

2. Проекция векторов:

Поскольку каждое слово теперь представлено вектором в многомерном пространстве, мы можем использовать методы уменьшения размерности, такие как t-SNE или PCA, чтобы проецировать эти векторы на двумерное пространство.

Это делает возможным их отображение на графике разброса.

3. Построение графика:

Теперь каждое слово можно представить точкой на графике разброса, где координаты точки соответствуют двум главным компонентам вектора слова.

4. Интерпретация графика:

Слова, которые имеют схожий смысл или используются в схожем контексте, будут ближе друг к другу на графике.

Это позволяет нам увидеть отношения между словами и может помочь в анализе текста.

Данный график позволяет визуально анализировать кластеры слов или оценивать взаимное расположение слов друг относительно друга.

Таким образом, график разброса может быть очень полезен при визуализации векторного представления слов.

Он позволяет нам увидеть отношения между словами и может помочь в анализе и интерпретации текста.

График шестиугольника, также известный как hexplot, это двумерная гистограмма или двумерное распределение, где вместо столбцов используются шестиугольники.

Это форма визуализации данных, которая используется для представления плотности точек между двумя переменными, обычно коррелирующими.

Основной концепт графика шестиугольника заключается в разделении пространства на шестиугольники, подсчитывающих количество наблюдений в каждом из этих шестиугольников и затем отображающих их с использованием цветовой шкалы.

Чем темнее шестиугольник, тем больше точек попадает в эту область. В результате получается визуализация плотности точек.

Hexplot обычно используется, когда точки данных слишком плотно упакованы, чтобы быть отображенными как scatter plot.

Он может быть полезен для изучения взаимосвязи между двуми числовыми переменными, особенно когда есть много перекрывающихся точек.

Hexplot также может быть полезен для представления больших наборов данных, поскольку он может отображать плотность точек в определенных областях, что может быть сложно увидеть на других типах графиков. Он может быть использован для исследования данных, поиска шаблонов, выявления аномалий и т.д.

График линий - это тип графика, который отображает информацию в виде серии точек данных, соединенных прямыми линиями.

Это один из самых простых и наиболее часто используемых типов диаграмм, особенно в области науки и инженерии.

Концепт графика линий основан на двух ключевых элементах - оси X и Y.

Ось X (горизонтальная) обычно представляет временной период, хотя она может также отображать любую категорию данных.

Ось Y (вертикальная) обычно отображает числовые значения, которые соответствуют данным на оси X.

Точки данных на графике представляют собой конкретные значения в определенный момент времени или для определенной категории.

Линия, которая соединяет эти точки, помогает визуализировать тренды или изменения в данных за определенный период времени или между категориями.

Графики линий часто используются для отслеживания изменений во времени и могут быть полезны для идентификации пиков, впадин, трендов или паттернов в данных.

Они также могут быть использованы для сравнения нескольких наборов данных, путем отображения нескольких линий на одном графике.

Важно помнить, что, хотя графики линий могут быть очень полезными для визуализации данных, они также могут быть вводящими в заблуждение,

если оси не масштабированы должным образом или если данные представлены без контекста.

Географический анализ данных - это метод анализа, который позволяет оценить распределение и взаимосвязи между географическими объектами и явлениями.

Он может быть использован для изучения широкого спектра вопросов, от определения местоположения нового магазина до прогнозирования изменений климата.

Географический анализ данных включает в себя сбор, обработку и интерпретацию географических данных.

Географические карты являются основным инструментом для визуализации географических данных.

Они представляют собой графическое изображение поверхности Земли или ее части, позволяющее увидеть распределение объектов и явлений в пространстве.

Карты со слоем точек:

Это наиболее простой тип карты, который используется для отображения точечных данных.

Каждая точка на карте представляет конкретное местоположение или событие.

Точки могут быть разного размера и цвета, чтобы отразить различные характеристики данных.

Например, на карте мира точки могут представлять города, размер и цвет которых соответствуют их населению.

На примере на слайде видим визуализацию продаж товаров по различным районам Москвы с размером точки, обозначающим число клиентов, заказавших товар,

а цвет обозначает число заявок на доставку.

Карты со слоем кластеров:   
Этот тип карты используется для отображения группы точек, которые были объединены вместе на основе определенных критериев.

Кластеризация обычно используется для упрощения данных и улучшения читаемости карты.

Например, на карте могут быть показаны кластеры землетрясений, где каждый кластер представляет собой группу землетрясений, произошедших в определенном районе.

На слайде визуализация данных показывает кластеры количества продаж товаров по количеству последних доставок, цветом обозначены категории клиентов.

Карты со слоем полигонов:

Этот тип карты используется для отображения данных, которые связаны с определенными географическими областями, такими как страны, штаты или районы.

Каждый полигон может быть окрашен в разный цвет или оттенок, чтобы отразить значение определенного показателя.

Например, на карте могут быть показаны уровни безработицы в разных странах, где каждая страна представляет собой полигон, цвет которого соответствует уровню безработицы.

На данном слайде цветом выделена информация о сумме продаж по районам Москвы

Все эти типы карт могут быть использованы вместе для создания сложных и информативных визуализаций географических данных.

Дашборд (от англ. dashboard – инструментальная панель) – планка, доска с одним или несколькими измерительными приборами, содержащая некую информацию.

Приборная доска автомобиля, любой градусник, настенные часы, расписание автобусов на остановке – это вариации дашбордов.

Сегодня так называют аналитическую панель, на которой собрана информация из разных источников.

Дашборд — это интерактивная аналитическая панель, графический интерфейс.

Смысл в том, что на одном экране расположены все ключевые метрики, показатели цели или процессов.

С помощью этих метрик можно выявить и проанализировать тренды и изменения.

Для визуального анализа хорошо использовать визуализацию данных, но еще лучше применять динамические и интерактивные диаграммы и графики, которые предоставляет дашборд.

Дашборды обеспечивают быстрый доступ к ключевым метрикам и показателям, а также позволяют отслеживать изменение данных в реальном времени.

Концепция построения дашбордов базируется на следующих принципах:

* Целевая аудитория: Дашборды должны быть разработаны с учетом потребностей конкретной аудитории.

Например, дашборд для руководителя будет отличаться от дашборда для аналитика или менеджера проекта.

2. Важность данных: На дашборде должны отображаться только самые важные и актуальные данные. Информация должна быть представлена таким образом, чтобы пользователь мог быстро увидеть ключевые показатели и тенденции.

3. Визуализация данных: Важно использовать правильные типы графиков и диаграмм для представления данных. Это помогает пользователю быстрее и легче интерпретировать информацию.

4. Интерактивность: Дашборды часто включают интерактивные элементы, такие как фильтры и переключатели, которые позволяют пользователям настраивать отображение данных.

5. Обновление данных: Дашборды должны обновляться в реальном времени или с определенной периодичностью, чтобы предоставлять актуальную информацию.

На дашбордах часто используются следующие типы графиков:

- Столбчатые и гистограммы: Используются для сравнения данных в разных категориях.

- Линейные графики: Помогают отслеживать изменения данных во времени.

- Круговые диаграммы: Подходят для отображения пропорций между категориями.

- Тепловые карты: Используются для визуализации матриц данных.

- Диаграммы рассеяния: Помогают исследовать корреляцию между двумя переменными.

В зависимости от конкретных целей и задач, могут использоваться и другие типы графиков и диаграмм.

Ключевой особенностью дашбордов является динамическое представление информации в зависимости от выбранных фильтров категорий, даты и времени, настроенных на холсте

Дашборды применяются на стадии оценки результатов обучения модели машинного обучения, стадии анализа имеющихся данных в преддверие обучения модели и на этапе внедрения, когда результаты работы модели машинного обучения или статистической модели имеется возможность визуализировать на панели.

Дашборды являются лучшим способом представить информацию о данных и проделанной работе на основе данных в подробном и простом виде.

Чем проще будет визуализация и понятнее предоставляемая зависимость, тем лучше ощутить положительный эффект от внедрения дашборда.

BI (Business Intelligence) система - это комплекс программных решений, которые позволяют собирать, обрабатывать, анализировать и визуализировать данные для поддержки принятия бизнес-решений.

BI системы применяются практически во всех аналитических компаниях, занимающихся разработкой аналитических решений.

Такое ПО нужно не только финансовым компаниям и большим компаниям, но всем,

кто хочет свести воедино информацию от отделов закупок и продаж, с сайта и маркетплейсов, из рекламного кабинета и CRM.

BI-системы бывают широкого назначения – для любого бизнеса, и специализированные – для конкретной сферы (медицина, бухгалтерия).  
  
Основные функции BI системы:

1. Сбор данных: BI системы могут собирать данные из различных источников, включая базы данных, файлы, документы, электронные таблицы, веб-сайты и даже социальные сети.

2. Обработка данных: После сбора данных, BI системы обрабатывают их, преобразуя сырые данные в структурированную форму, которую можно анализировать.

3. Визуализация данных: BI системы предоставляют инструменты для создания диаграмм, графиков и других визуальных представлений данных, которые помогают пользователям лучше понять информацию и увидеть скрытые закономерности и тренды.

4. Отчетность: BI системы позволяют создавать различные отчеты, включая оперативные отчеты, аналитические отчеты, дашборды и другие.

5. Поддержка принятия решений: Наконец, BI системы помогают пользователям принимать обоснованные бизнес-решения на основе анализа данных.

BI системы могут быть использованы в различных областях, включая финансы, маркетинг, продажи, производство, логистику, управление персоналом и другие.

Они помогают компаниям улучшить эффективность, повысить производительность, снизить риски и получить конкурентное преимущество.

Чтобы повысить эффективность и оптимизировать расходы, российский бизнес инвестирует в бизнес-аналитику.

Еще несколько лет назад задачу решали с помощью зарубежных BI-решений — как правило, Power BI или Tableau.

Однако из-за массового оттока иностранных BI-вендоров в 2022 году и прогрессирующего тренда на импортозамещение российские компании стали сотрудничать с отечественными производителями BI-платформ.

**Visiology, Alpha BI, Analytic Workspace (AW BI), Modus BI, Luxms BI, «Форсайт», «Триафлай», PolyAnalyst, PIX BI, Yandex DataLens** — популярные российские BI-системы.

Они предлагают широкий спектр решений на основе лучших практик западных BI-платформ Power BI, Tableau и Metabase.

Это позволяет подобрать подходящую функциональность с учетом уникальных BI-задач  и размеров бизнеса в разных технологических и ценовых категориях.

Платформа для бизнес-аналитики.

Позволяет визуализировать показатели, определять тренды и находить зависимости, создавать прогнозы на основе ваших данных.

Эффективный инструмент для управления всеми сферами деятельности.

**Адаптивный интерфейс**

Отображение отчетов на любых устройствах.

Верстка под разные разрешения экрана

**Формы ввода данных**

Возможность ручного ввода данных с настраиваемым контролем вводимых значений

**Работа в браузере**

Не требует установки программного обеспечения на компьютер

**Безопасность и настройка доступа**

Настраиваемые роли пользователей и профили доступа к отчетам.

**Множество источников данных**

Большая библиотека настроенных интеграций для быстрого подключения источников данных

- выдает результат в виде интерактивных отчетов;

- отчеты отображаются на любых устройствах;

- данные можно изменять вручную с контролем значений и аудиторским следом (можно исправить, но нельзя подделать);

- ПО работает в браузере, устанавливать не нужно;

- для нескольких пользователей можно настроить разные права доступа;

- рассчитана на разное количество пользователей (зависит от цены);

- большая библиотека встроенных интеграций для получения данных из других программ, серверов, сайтов.

Yandex DataLens

BI‑инструмент, c помощью которого можно подключиться к источнику, описать модель данных, создать визуализации, собрать дашборд и поделиться результатом с командой.

**Возможности платформы**

**Единая экосистема** работа с системой происходит в едином контуре безопасности YandexCloud;

**Гибкость** объединить разные источники данных можно в облаке или на собственном сервере;

**Совместный доступ** построенными дашбордами можно делиться с коллегами или встроить в код сайта;

**Безопасность** Datalens входит в реестр отечественного ПО и соответствует российским и международным стандартам безопасности

**Бесплатный доступ** нет ограничений по количеству пользователей, объему хранилища или вариантов визуализаций.

- Полностью бесплатный сервис визуализации и анализа данных, можно подключить любое количество пользователей.

- Платформа берет данные с сайтов, сервисов аналитики, бизнес-аналитику Битрикс24, баз данных.

На сайте сервиса есть подробные сценарии, по которым можно научиться пользоваться аналитикой.

- Есть недостаток – все данные находятся в облаке, поэтому есть риск утечки данных.

Последнее обновление DataLens позволяет развернуть систему на локальном устройстве или локальном сервере, что позволяет проводить аналитику с критическими конфиденциальными данными.

Открытый исходный код гарантирует безопасность такого решения

Современная BI-платформа для бизнес-анализа и планирования. Платформа позволяет быстро принимать грамотные управленческие решения на основе достоверной информации за счет удобства представления данных.

**Возможности платформы**

**Поддержка языка DAX**

Возможность писать запросы на языке DAX, к которому привыкли специалисты по Microsoft Power BI.

**Большие данные**

Visiology расширяет возможности по работе с BigData за счет новой архитектуры хранения и обработки данных.

**Скорость**

Новое вычислительный движок на базе Clickhouse который обеспечивает кратный рост скорости обработки данных.

**Визуальная модель данных**

Упрощенная схема работы с объединением таблиц простым перетаскиванием объектов (Drag&Drop)

**Архитектура**

Соответствие архитектуры ведущим платформам подходит для миграции решений Qlik, Tableau, Oracle BI, SAP BI.

ПО можно установить на компьютер или пользоваться в облаке.

Разработчик – компания Visiology. Сервис платный, предоставляют бесплатную демоверсию.

примеры кейсов использования BI систем для построения дашбордов

* Оптимизация бизнес-процессов в розничной торговле: BI система может собирать данные о продажах, покупателях, товарах и т.д.

Дашборды могут отображать информацию о самых продаваемых товарах, доходности различных категорий товаров, сезонности продаж и другую полезную информацию, которая поможет управлять ассортиментом и ценообразованием.

2. Управление проектами: дашборды могут отображать информацию о статусе проектов, выполнении задач, затраченном времени и ресурсах, эффективности команды и т.д. Это поможет менеджерам проектов принимать обоснованные решения и своевременно корректировать курс действий.

3. Финансовый анализ: BI система может собирать данные из различных источников, таких как бухгалтерский учет, банковские транзакции, отчеты о продажах и т.д. Дашборды могут отображать информацию о доходах и расходах, финансовом состоянии компании, эффективности инвестиций и т.д.

4. Анализ работы отдела маркетинга: BI система может собирать данные о рекламных кампаниях, кликах, конверсиях, продажах и т.д. Дашборды могут отображать информацию о эффективности различных маркетинговых каналов, ROI рекламных кампаний, поведении пользователей на сайте и т.д.

5. HR аналитика: дашборды могут отображать информацию о численности персонала, текучке кадров, затратах на зарплаты и прочие HR-метрики. Это поможет HR-менеджерам оптимизировать процессы найма и увольнения, а также управление персоналом в целом.

Приветствую вас уважаемые учащиеся и коллеги,

Мы продолжаем разбираться с основами больших данных на нашем курсе.

Технологии обработки больших объемов данных

- Распределенные системы

- NoSQL

- Озера данных

- Hadoop, HDFS, Spark, Yarn

- Потоки обработки данных

- Виды облачных сервисов

Все вычислительные задачи, решаемые на компьютерных системах, опираются на **процессорные вычисления** и **локальные хранилища данных**.

Однопроцессорные вычисления способны обработать значительное число информации за короткое время в масштабах локальных персональных исследований (гигабайты данных).

Однако для обработки больших массивов информации необходимы новые подходы к проектированию систем хранения и обработки данных.

Для обработки современных больших данных (петабайты) каждый день недостаточно одной серверной единицы (одного серверного компьютера).

Требуется внедрять вычисления на нескольких процессорах и запись на несколько дисков одновременно.

Представим идеализированную ситуацию в отрыве от оптимизаций и реальной физической работы записи файлов на носитель.

На диск необходимо записать 4 одинаковых файла общим размером в 100 гигабайт.

При примерной средней скорости записи файлов на жесткий диск в 50 мегабит (примерная скорость записи данных на жесткий диск) в секунду мы получим примерное время записи в 4 часа реального времени.

Это довольно долго для столь малого объема данных по сегодняшним меркам, поэтому

в дело вступает параллельная обработка данных.

**Последовательная:** один процессор выполняет одну задачу в заданное время, а другие задачи ждут в очереди.

В операционной системе может быть запущено несколько программ, и каждая программа имеет несколько запущенных задач.

В этом случае все задачи разных программ передаются процессору через регистры и обрабатываются последовательно.

**Параллельная:** несколько задач выполняются одновременно разными исполнителями.

Операционная система, работающая на многоядерном процессоре, является примером параллельной операционной системы.

В такой системе задачи от разных программ распределяются между не занятыми процессорными ядрами, что увеличивает скорость обработки данных.

Предоставим нашему тестовому стенду общее хранилище с 4-мя физическими носителями.

Предположим, что наша программная система умеет записывать разные файлы с использованием разных ядер на одно устройство хранения.

Также предположим что в данный момент времени наша система не решает фоновых задач, 4 ядра всегда свободны и шина данных постоянно функционирует.

Одно ядро открывает для себя один поток с контроллера ввода-вывода, который содержит в себе одно дисковое пространство, представленное 4-мя различными устройствами хранения.

В такой идеализированной системе на диск необходимо записать 4 одинаковых файла общим размером в 100 гигабайт.

При примерной средней скорости записи файлов на жесткий диск в 50 мегабит в секунду мы получим примерное время записи в 1 час реального времени.

Это уже быстрее но все еще недостаточно при физическом ограничении на число процессоров.

И тогда в дело вступает распределенная обработка данных

**Распределённая обработка данных** - методика выполнения прикладных программ группой систем (узлов, компьютеров, устройств).

В данном понятии мы говорим о том, что разные системы находятся в едином сетевом окружении и прикладные программы рассчитаны на запуск и автоматический обмен данными между узлами.

Тогда пользователь получает возможность работать с сетевыми службами и прикладными процессами, расположенными в нескольких взаимосвязанных системах.

Для реализации такого паттерна чаще всего прибегают к виртуализации операционной системы, состоящей из нескольких модулей, отвечающих за автоматическую оркестровку узлов обработки информации.

**Распределенная система** — это совокупность компьютеров, которые обмениваются данными и синхронизируются в общей сети, образуя «единый компьютер» для конечного пользователя.

Ее также называют распределенными вычислениями или распределенной базой данных.

Система строится на базе различного оборудования, среди которых:

- серверное оборудование, отличающееся высокой долговечностью и пропускной способностью,

- пользовательское оборудование, менее надежное и более экономичное по затратам при прочих равных в производительности.

Тогда такая система физически (на крупных предприятиях или в ЦОДах) строится из так называемых «стоек».

«Стойка» в контексте организации вычислений на вычислительных устройствах представляет собой физический стенд с полками, на котором располагаются узлы – серверные компьютеры с горизонтальным расположением.

Узлы в стойке связаны между собой локальной сетью - скоростным каналом передачи данных.

На верхнем уровне стойки стоит коммутатор – сетевое устройство организующее передачу данных в одноранговой сети между узлами стойки.

Стойки друг с другом связаны также на уровне связи уже коммутаторов либо с помощью коммутаторов верхнего уровня для определения иерархии уровня передачи данных или для сегментации вычислительных устройств, либо с помощью перекрестного соединения всех-со-всеми для быстрой неорганизованной передачи пакетов в маленькой серверной.

Несколько узлов (как внутри одной «стойки», так и в разных стойках) на уровне виртуализации объединяют в одну сетевую инфраструктуру и развертывают на них единую высокоуровневую операционную систему, поддерживающую автоматическую синхронизацию и передачу данных между узлами.

Узлы – серверные компьютеры с серверным железом, которые физически не сильно отличаются от персональных компьютеров, за исключением наличия в них серверного процессора, нескольких дисковых хранилищ и возможных ускорителей.

Такого рода системы называются системами с разделенной или распределенной памятью, поскольку данные, как в хранилище так и в памяти, находятся на удалении друг от друга, хоть и в общем доступе.

Это может накладывать физические ограничения на сложность разрабатываемых алгоритмов обработки данных, а так же на их скорость.

Доступ к такой распределенной системе снаружи возможен посредством удаленного доступа или главного терминала в серверной.

Удаленный доступ может производиться различными способами вплоть до соединения через протокол безопасного сетевого взаимодействия SSH.

Свойства распределенной системы:

**Совместное использование ресурсов:**

Это свойство распределенной системы позволяет нескольким пользователям или процессам использовать общие ресурсы, такие как файлы, документы, базы данных и т.д.

Это обеспечивает эффективное использование ресурсов и облегчает доступ к ним.

Ресурсы могут быть распределены по всей сети и доступны для всех узлов в системе.

Данные ресурсы доступны по средством механизма сетевого взаимодействия, поэтому доступ к ним может быть не мгновенным.

**Параллельная обработка:**

В распределенной системе задачи или процессы могут быть разделены между несколькими узлами для параллельной обработки.

Это улучшает производительность и скорость обработки данных, поскольку задачи выполняются одновременно на нескольких машинах, а не последовательно на одном компьютере.

**Масштабируемость:**

Методики расширения ресурсов делятся на две большие категории: горизонтальные и вертикальные.

**Горизонтальное масштабирование** означает добавление дополнительных узлов в систему (или удаление узлов из) системы, например добавление нового компьютера в распределенное программное приложение.

**Вертикальное масштабирование** означает добавление ресурсов (или удаление ресурсов) к одному узлу, как правило, с добавлением ЦП, памяти или хранилища к одному компьютеру.

Распределенные системы могут легко масштабироваться путем добавления большего количества узлов или ресурсов, т.е. как горизонтально, так и вертикально.

В отличие от одноузловых систем, где доступно только вертикально масштабирование ресурсов.

Это означает, что система может расти и адаптироваться к увеличивающейся нагрузке, не теряя при этом производительности или эффективности.

**Обнаружение ошибок:**

Распределенные системы обычно имеют механизмы для обнаружения и восстановления от сбоев или отказов.

Это может включать в себя

- мониторинг состояния узлов,

- резервное копирование данных и процессов,

- а также автоматическое переключение на резервные узлы при обнаружении сбоев.

**Прозрачность:**

Прозрачность в распределенных системах означает, что детали реализации системы скрыты от пользователя.

Например, пользователь может не знать, на каком конкретном узле выполняется его задача или где физически хранятся его данные.

Это делает использование распределенной системы более простым и удобным для пользователя.

Также в процессе распределенной обработки данных стоит упомянуть способ управления узлами в кластерной распределенной системе.  
Одной из популярных моделей оркестровки рабочих узлов в распределенной системе является модель Master-Slave

Архитектура Master-Slave является одним из наиболее распространенных подходов в области распределенной обработки данных. В этой модели, один узел (Master) играет главную роль, а остальные узлы (Slaves) выполняют подчиненные функции.   
  
1. Master-узел: Это центральный узел, который контролирует и координирует работу всех Slave-узлов.

Он отвечает за распределение задач между Slave-узлами, мониторинг их состояния, восстановление после сбоев, а также сбор и агрегацию результатов.  
  
2. Slave-узлы: Это подчиненные узлы, которые выполняют задачи, полученные от Master-узла. Они выполняют обработку данных и отправляют результаты обратно на Master-узел.  
  
Взаимодействие между Master и Slave обычно происходит следующим образом:  
- Master-узел распределяет задачи между Slave-узлами.

- Slave-узлы выполняют полученные задачи, обрабатывая локально доступные данные.

- После завершения обработки, Slave-узлы отправляют результаты обратно на Master-узел.

- Master-узел агрегирует все полученные результаты для формирования окончательного вывода.

Однако ее реализация претерпевает ряд модификаций, среди которых:

- введение вспомогательного узла для управления во время сбоя,

- ведения журнала сбоев,

- ведение журнала журнала подчиненных узлов

На основе Master-Slave архитектуры построено множество решений для распределенной обработки данных, среди которых Hadoop и Spark

Maste-Slave архитектура является одной из популярных архитектур для обработки больших данных в облаке с применением распределенных вычислений.

Причины популярности такой архитектуры достаточно просты:

1. Простота реализации:

мастер контролирует все операции и обеспечивает координацию между рабочими узлами.

2. Эффективное использование ресурсов:

подчиненные узлы могут выполнять различные задачи параллельно, что позволяет увеличить производительность системы.

3. Масштабируемость:

можно легко добавить новые рабочие узлы в систему для обработки больших объемов данных.

4. Отказоустойчивость:

если один из рабочих узлов выходит из строя, мастер может перераспределить его задачи между другими узлами.

Из минусов можно выделить:

* Точка отказа:

Если мастер выходит из строя, вся система может остановиться, поскольку только мастер может распределять задачи.

2. Проблемы масштабирования:

Если количество узлов сильно увеличивается, мастер может стать узким местом, поскольку все запросы проходят через него.

3. Неравномерная нагрузка:

Мастер может быть перегружен, поскольку он обрабатывает все запросы на распределение задач, в то время как некоторые узлы могут быть недостаточно загружены.

4. Проблемы синхронизации: Если узлы работают с разной скоростью, мастеру может быть сложно эффективно координировать их работу.

Поскольку сеть компьютеров, объединенных в кластер, объединена не с помощью шины передачи данных, как компоненты компьютера на материнской плате, а с помощью сетевой инфраструктуры, а именно:

- коммутаторов

- интерфейсов

- проводов

Существует ряд ограничений на скорость обработки данных, которые являются естественным проложением недостатков параллельной обработки данных на системе с разделенной памятью.

В реальных задачах с распределением вычислений на N устройств не может быть и речи о повышении производительности в N раз в связи с законом **Амдала**.

Закон Адмала о параллельной обработке данных, также известный как закон Амдала, был сформулирован в 1967 году Джином Амдалом, компьютерным ученым из IBM.

Этот закон определяет теоретический максимум ускорения выполнения программы при использовании нескольких процессоров.   
  
Согласно закону Амдала, ускорение выполнения программы на параллельной системе ограничено долей времени, в течение которого задача может быть выполнена только последовательно.

Другими словами, если часть программы не может быть выполнена параллельно (например, из-за зависимостей между инструкциями), то это ограничивает общее ускорение, которое можно достичь при использовании нескольких процессоров.

Формула закона Адмала выглядит следующим образом:  
  
S(N) = 1 / ((1 - P) + P/N)  
  
где:  
  
- S(N) - ускорение работы программы  
- N - количество процессоров  
- P - доля программы, которую можно выполнить параллельно

Например, если 75% программы можно выполнить параллельно (P = 0.75), и у нас есть 4 процессора (N = 4), то максимальное ускорение, которое мы можем достичь, согласно закону Амдала, составляет 2.67 раза.  
  
Важно понимать, что закон Амдала является теоретическим идеалом, и на практике может быть трудно достичь ДАЖЕ этого уровня ускорения из-за различных факторов,

таких как затраты на коммуникацию между процессорами, задержки ввода-вывода и другие проблемы синхронизации.

На ускорение вычислений также влияют и другие факторы:

* Качество интерфейса соединения исполнителей в сети:

Относится к пропускной способности и надежности соединения между узлами в сети.

Если соединение медленное или нестабильное, это может замедлить обработку данных.

На слайде показан серверный соединительный кабель с самой низкой задержкой постановки данных на шину и оптимальной скоростью передачи данных.

Множество серверных решений прямо говоря держится на нем.

2. Программная балансировка нагрузки на исполнителях:

Это процесс распределения работы между различными узлами в сети для оптимизации использования ресурсов и минимизации времени ответа.

Если балансировка нагрузки неэффективна, это может привести к неравномерному использованию ресурсов и увеличению времени обработки.  
  
3. Помехи при передаче данных:

Относится к ошибкам, которые могут возникнуть во время передачи данных между узлами.

Это может быть вызвано шумом, потерей пакетов или другими проблемами сети, что может привести к повторной передаче данных и увеличению времени обработки.  
  
4. Скорость чтения и записи накопителей:

Это скорость, с которой данные могут быть прочитаны или записаны на физические накопители, такие как жесткие диски.

Если скорость чтения/записи накопителей медленная, это может стать узким местом в процессе обработки данных.

Уже само название заявляет, что управлять данными можно не только с помощью Structured Query Language (SQL), т. е. языка структурированных запросов.

Модель NoSQL появилась в ответ на необходимость оперативно обрабатывать действительно огромные объёмы данных. Поэтому NoSQL по большей части заточена под масштабирование по горизонтали и работу с недостаточно структурированными или постоянно меняющимися данными.

Эпоха реляционных баз данных, наступила в 1980-х годах.

В то время основным содержанием баз данных были текстовые документы и маленькие изображения.

Однако с технологическим прогрессом и увеличением объема обрабатываемой информации,

реляционные системы управления базами данных стали менее эффективными в решении некоторых задач.  
  
Термин NoSQL был впервые использован в 1998 году итальянским ученым Карло Строцци для описания его open source системы управления базами данных.

В процессе разработки Строцци отказался от SQL, а также от ключевого принципа реляционных систем управления базами данных - ACID (атомарность, согласованность, изолированность, долговечность).

Основные причины появления NoSQL:

Возникла необходимость в распределенных системах управления базами данных (СУБД).

С появлением IT-корпораций, глобальных приложений и социальных сетей, возникла потребность в масштабировании баз данных.

Вертикальное масштабирование оборудования является дорогостоящим решением, и реляционные БД плохо подходят для шардирования: чем больше серверов в системе, тем больше усилий необходимо для поддержания согласованности данных на узлах.  
  
Обработка данных стала быстрее.

В отличие от нереляционных БД, SQL извлекает данные из нескольких таблиц.

Однако, когда объем информации увеличивается, количество таблиц и связей становится слишком большим, что приводит к снижению скорости ответа на запрос.  
  
Разработчики стремились преодолеть ограничения реляционных схем.

Строгая реляционная модель не всегда подходит для всех предметных областей.

Иногда они бывают слишком сложными или требуют частых корректировок данных.

В результате получается либо излишне большое количество таблиц, либо плохо структурированная предметная область.

**SQL vs. NoSQL: в чём разница**

**Управление**

Работа с реляционной базой строится на общепринятом языке SQL.

У NoSQL СУБД нет единого стандарта: у каждой такой базы индивидуальный подход к записи, хранению и извлечению данных.

Поиск информации может вестись, например, по парам «ключ — значение» или по наборам столбцов.

**Структура**

Для реляционных БД необходима фиксированная схема — точное описание структуры таблицы, её полей и ограничений.

Данные в строках таблиц должны быть нормализованы, а добавление нового свойства влечёт за собой изменение всей схемы.

Этот подход эффективен, когда необходима стабильность и чёткая структура.

Нереляционные СУБД существуют по другим принципам и законам, в них структура не регламентирована,

отсюда и принципиальное отличие NoSQL от реляционных БД — их гибкость.

СУБД без SQL позволяют записывать и получать неструктурированную информацию, дают возможность менять схемы и запросы в соответствии с требованиями к данным.

Когда речь идёт о работе с данными временных рядов, информацией без взаимосвязи, о хранении документов с разной структурой и высокой степенью вложенности, то именно NoSQL показывают большую скорость и производительность.

**ACID-транзакции**

При разработке NoSQL были смягчены жесткие требования к транзакциям.

Основные принципы ACID гарантируют, что целостность и согласованность данных в реляционных хранилищах будет обеспечена даже при сбоях.

Поскольку для ряда задач в строгом следовании ACID нет никакой необходимости, NoSQL-базы чаще всего предлагают компромисс и принципы BASE:

- система обеспечивает базовую доступность (Basic Availability), т. е. каждый запрос будет обязательно завершён, успешно или нет;

- система пребывает в гибком состоянии (Soft-state) — очерёдность записей соблюдать необязательно, реплики могут какое-то время находиться в несогласованном состоянии, а система может самостоятельно изменяться для достижения согласованности;

- все данные всё равно достигнут согласованности (Eventual Consistency).

**Плюсы NoSQL**

**Гибкость модели данных**

NoSQL-подход разрешает группировать любой набор данных и их связей.

Объект данных при этом может быть многосоставным: нормализация в NoSQL не требуется.

Это свойство будет полезно, например, стартапам, когда модель хранилища данных меняется во время разработки самого продукта и в начале работы не удаётся спрогнозировать финальную архитектуру БД.

**Доступность данных**

Благодаря механизмам отказоустойчивости секций, в том числе репликации и шардированию, NoSQL способна в любой момент обслужить входящий запрос и вернуть не ошибочный ответ.

Избыточность и отказоустойчивость достигаются благодаря репликации данных на узлах.

И даже если реплики окажутся недоступны, можно произвести запись в базу данных.

По мере доступности узлов каждая реплика будет обновлена.

**Лучшая масштабируемость**

Поскольку между записями в NoSQL нет жёсткой связи, данные можно дробить и хранить на нескольких независимых серверах.

Горизонтальное масштабирование легче и дешевле, чем вертикальное, присущее SQL-моделям.

Производительность реляционной системы БД увеличивается с помощью дополнительного дорогого оборудования (и то не бесконечно), а в нереляционных БД — с помощью добавления новых узлов.

Это делает NoSQL более удобными при взаимодействии с большими или меняющимися наборами данных.

**Высокая производительность**

Благодаря оптимизации баз под определённые виды моделей данных скорость представления информации часто превосходит скорость SQL-базы.

Например, если необходимые записи хранятся в одном документе, больше нет потребности в операции JOIN.

Это упрощает процесс, когда необходимо провести аналитику над данными, агрегацию или расчёты внутри сущности.

**Экономия ресурсов**

Горизонтальное масштабирование позволяет сократить количество дорогостоящих серверов.

А поскольку большинство NoSQL являются open source проектами, можно экономить на подписке и поддержке.

Или развернуть и эксплуатировать базы данных в облачном сервисе: NoSQL благодаря распределённости и горизонтальной масштабируемости прекрасно для этого подходят.

**Для каких задач подходят NoSQL базы данных**

Если реляционные СУБД на протяжении многих лет были универсальным решением, то с развитием NoSQL у разработчиков появился выбор: теперь базы данных реально подобрать под конкретные задачи.

NoSQL-СУБД различаются моделями данных, а также подходом к распределённости и репликации. Выделяют четыре основные категории нереляционных БД.

**Базы данных по принципу «ключ — значение» (key-value store)**

В этой БД записи хранятся в парах «ключ — значение», где ключ выступает уникальным идентификатором.

Ключи и значения фиксируются в виде простой или составной информации.

Эти хранилища максимально быстро реагируют на запросы информации и прекрасно масштабируются.

Key-value СУБД часто используется для систем, в которых скорость является приоритетом, а данные не слишком сложные.

Например, для хранения кеша данных, онлайн-списков, обработки истечения срока действия, разделения сеансов, построения рейтинга и прочих задач.

Яркий пример key-value store БД — Redis. Ей пользуются Airbnb, Slack, Twitter и Uber.

Система целиком работает в оперативной памяти, что позволяет информации считываться и записываться намного быстрее, чем даже на очень шустрые твердотельные накопители.

**Колоночные базы данных (column family store)**

**Мы уже разбирали колоночные СУБД в разрезе OLAP технологии хранения данных.  
Колоночные базы данных так же относятся к NoSQL решениям, поскольку находятся в денормализованных состояниях**

Эти БД имеют свои столбцы и строки, но информация записывается в колонки.

Колонки между собой не связаны, поэтому удаление или добавление новых свойств не затрагивает остальную систему.

Отсутствие заранее заданной схемы позволяет хранить в этих NoSQL-базах записи без чёткой структуры.

В традиционной СУБД при выполнении запроса сканируется вся таблица, а информация из всей строки извлекается целиком.

В колоночных БД выгружаются только необходимые значения, поскольку поиск ведётся по отдельным столбцам.

Такой подход колоночных NoSQL баз к хранению информации позволяет быстро получать данные из больших таблиц для анализа.

А возможность сильного сжатия данных экономит много места на диске.

**Документоориентированные базы данных (Document-oriented store)**

В БД этого типа данные записываются в документы и хранятся в формате, подобном JSON.

Таким хранилищам свойственны

- иерархичность (документы складываются в коллекции, а коллекции группируются логически) и

- гибкость (значения, свойства и их структура может меняться в процессе разработки).

Document-oriented-модель хороша в проектах, где нужно обрабатывать большой объём данных без четкой структуры, а также для работы со множеством уникальных документов, которые со временем требуют изменений.

Например, для каталогов товаров, соцсетей, платформ с блогами и видео, геоаналитики и в других сферах.

Представителем документной СУБД является MongoDB

Гибкость MongoDB даёт широкий выбор способов моделирования данных.

Такая модель облегчает хранение массивов и других сложных структур.

Ориентированная на документы база данных прекрасно показывает себя при большом потоке запросов в реальном времени, поэтому может стать частью социальной сети.

Также она подойдёт для кеширования данных и создания брокера очередей.

И всё это вкупе с отказоустойчивостью, простым масштабированием за несколько минут и высокими показателями безопасности данных.

**Графовые базы данных (graph store)**

Элементы базы данных хранятся в узлах (вершинах), между узлами существуют ребра, которые определяют отношения элементов друг к другу.

У ребра есть начальный и конечный узел, направление и тип (связи действия, права владения, «родитель — ребёнок» и пр.).

Главная особенность графовых БД — хранение не только сущностей данных, н

о и взаимосвязей, тогда как в реляционных БД соединения между элементами требует дополнительных вычислений.

Благодаря такой модели данных graph store NoSQL используются для выполнения задач, ориентированных на связи: для алгоритмов рекомендаций контента, социальных сетей, обнаружения случаев сетевого мошенничества.

Для доступа к таким БД необходим язык запросов, но, поскольку общепринятых стандартов у NoSQL нет, для разных типов графовых баз данных понадобится индивидуальный подход.

К графовым относятся базы данных Neo4j, OrientDB.

**Как выбрать базу данных**

С появлением такого разнообразия баз данных пропала необходимость останавливаться на каком-то одном варианте.

Теперь можно не создавать единое монолитное приложение, а решать разные задачи бизнеса с помощью подходящих для этого микросервисов.

Выбрать NoSQL стоит в тех случаях, когда:

- нужно хранить большие объёмы неструктурированных или быстро меняющихся данных;

- нет возможности описать схему СУБД до начала проектирования базы данных или предполагается, что в процессе работы архитектура хранилища будет меняться;

- необходимо быстро без большой предварительной подготовки запустить прототип продукта;

- требуется высокая масштабируемость системы без лишних ресурсозатрат;

- строгой согласованностью можно пожертвовать ради производительности и доступности.

Далее рассмотрим принципы работы DataLake а также поймем суть данной технологии в рамках пайплайна обработки данных

**Data Lake** — это репозиторий для хранения, который может вмещать большой объем данных в необработанном формате в виде файлов.

Хранилища Data Lake оптимизированы для масштабирования до **нескольких терабайт** и даже **петабайт данных**.

Данные обычно поступают из **нескольких разнородных источников** и могут быть **структурированными**, **частично структурированными** и **неструктурированными**.

DataLake выступают своего рода хранилищами неструктурированных данных для NoSQL варианта использования информации.

Идея, лежащая в основе Data Lake, — *хранение всех данных в исходном состоянии без каких-либо преобразований*.

Такой подход отличает Data Lake от традиционного хранилища данных, в котором данные преобразуются и обрабатываются во время приема.

Основные варианты использования озера данных.

- Перемещение данных в облаке и IoT

- Обработка больших данных

- Аналитика

- Отчеты

- Перемещение локальных данных

Преимущества DataLake обеспечены в основном положительными моментами использования NoSQL модели работы с данными,

Среди таких выделяем следующие

Данные никогда не отклоняются от загрузки в хранилище, так как хранятся в необработанном формате.

Это особенно полезно в окружении с большими данными, если заранее неизвестно, какие именно сведения будут получены в результате анализа данных.

Данные никогда не отклоняются от загрузки в хранилище, так как хранятся в необработанном формате.

Это особенно полезно в окружении с большими данными, если заранее неизвестно, какие именно сведения будут получены в результате анализа данных.

Пользователи могут просматривать данные и создавать собственные запросы.

Может работать быстрее, чем традиционные средства ETL за счет быстрой загрузки данных в хранилище и специализированной постановке задач обработки специализированной информации

Обладает большей гибкостью, чем хранилище данных, так как дает возможность хранить частично структурированные и неструктурированные данные.

Хранилище данных DataLake невозможно использовать в структурах, где используются стандарты хранения бизнес-информации, где поступление новых данных означает их своевременную регистрацию, организацию и учет, как например в бухгалтерии или в таможне.

В DataLake возможна потеря информации.

DataLake предназначено для потока сырой информации, среди которой выделяют:

- Данные потока кликов

- Логи сервера

- Социальные сети

- Координаты геолокаций

- Данные с датчиков и устройств

- Структурированные данные

- Электронная почта

Полное решение Data Lake состоит из компонентов хранения и обработки данных.

**Хранилище Data Lake**

создано для обеспечения отказоустойчивости, бесконечной масштабируемости и высокой пропускной способности при получении данных любых форм и размеров.

**Компоненты обработки Data Lake**

включают в себя один или несколько модулей обработки, созданных для целей обработки собираемых данных, и может работать с данными, хранящимися в Data Lake в нужном масштабе.

К наиболее распространенным сферам применения Data Lake относятся исследования данных, анализ данных и машинное обучение.

Data Lake также может служить источником данных для хранилища данных.

При таком подходе необработанные данные поступают в Data Lake, а затем преобразуются в структурированный формат, поддерживающий запросы.

Обычно для этого преобразования задействуется конвейер ELT (извлечение, загрузка и преобразование), в котором данные принимаются и преобразуются на месте.

Исходные данные, которые уже являются реляционными, могут передаваться непосредственно в хранилище данных с помощью процесса извлечения, преобразования и загрузки, минуя Data Lake.

Хранилища Data Lake часто используются при потоковой передаче событий или в сценариях Интернета вещей,

так как они могут хранить большие объемы реляционных и нереляционных данных без преобразования или определения схемы.

Они предназначены для обработки больших объемов мелких записей с низкой задержкой и оптимизированы для высокой пропускной способности.

В интеграции и работе c DataLake могут возникнуть следующие сложности:

Отсутствие схемы и описательных метаданных создает трудности при использовании данных и создании запросов.

- Отсутствие семантической согласованности между данными может затруднять анализ данных, если пользователи не обладают профессиональными навыками в этой области.

- Качество данных, поступающих в Data Lake, сложно гарантировать.

- Без надлежащего управления могут возникать проблемы с контролем доступа и конфиденциальностью. Какие данные поступают в Data Lake, кто может их использовать и с какой целью?

- Data Lake может оказаться не лучшим способом интеграции данных, которые уже являются реляционными.

- Само по себе хранилище Data Lake не поддерживает интегрированный или целостный просмотр данных для всей организации.

- Data Lake может превратиться в "свалку" данных, которые никогда не будут использоваться для изучения и анализа.

Бронзовая, серебряная и золотая стадии готовности данных в DataLake - это концепция, которая используется для описания различных этапов обработки и подготовки данных для анализа.

Эта система организации хранения данных помогает разделить процесс обработки данных на управляемые этапы, что обеспечивает более высокую эффективность и качество данных.  
  
1. Бронзовая стадия:

На этом этапе данные загружаются в DataLake в их первоначальном виде без какой-либо обработки или трансформации.

Это сырые данные, которые могут включать все, от лог-файлов и данных из IoT устройств до баз данных и файлов изображений.

Целью этого этапа является предоставление места для хранения большого объема необработанных данных, которые затем можно использовать для дальнейшего анализа.  
  
2. Серебряная стадия:

На этом этапе данные очищаются, нормализуются и трансформируются.

Это может включать удаление дубликатов, исправление ошибок, заполнение пропущенных значений и преобразование данных в более удобный для анализа формат.

Данные на этом этапе еще не готовы для анализа, но они более структурированы и организованы, чем на бронзовом этапе.  
  
3. Золотая стадия:

На этом этапе данные готовы к анализу.

Они были очищены, трансформированы и агрегированы в соответствии с потребностями бизнес-анализа.

Это могут быть, например, сводные таблицы, которые позволяют быстро и легко получить ответы на бизнес-вопросы.

Данные на этом этапе имеют высокую ценность, поскольку они могут быть непосредственно использованы для принятия решений и получения инсайтов.  
  
Важно отметить, что не все данные проходят через все три стадии. Некоторые данные могут быть полезными в их сыром виде, в то время как другие требуют значительной обработки и трансформации перед тем, как они станут полезными.  
  
Система организации хранения данных в DataLake обычно включает разделение данных по этим стадиям в различных областях хранения или "бассейнах".

Чаще всего это один уровень иерархии по степени вложенности в иерархической системе хранения данных DataLake

Это может помочь облегчить управление данными и обеспечить, что каждый этап обработки данных может быть должным образом отслежен и управляем.

Инструменты DataLake

Hadoop: Это открытый фреймворк, который позволяет проводить распределенную обработку больших наборов данных на кластерах компьютеров с использованием простых программных моделей. Он разработан для масштабирования от отдельных серверов до тысяч машин, каждая из которых предлагает локальные вычисления и хранение.

Azure Data Lake Storage Gen2: Это масштабируемое, проприеритарное и экономически эффективное решение для больших данных. Он объединяет мощь высокопроизводительной файловой системы с огромным масштабом и экономичностью, чтобы помочь вам ускорить аналитику больших данных.

Delta Lake: Delta Lake - это открытый инструмент хранения, который привносит ACID (атомарность, согласованность, изоляцию, долговечность) транзакции в Apache Spark и большие рабочие нагрузки. Он обеспечивает соблюдение схемы и эволюцию для обеспечения качества данных.

На очереди раздел

Экосистема Hadoop.  
Файловая система HDFS.   
Обработка данных с применением MapReduce

В котором мы разберем принципы работы Hadoop как классического представителя NoSQL системы хранения и обработки данных

Множество продуктов является морально устаревшими.

Среди использующихся продуктов выделяются:

Hadoop Common: Это набор общих утилит и библиотек, которые поддерживают другие модули Hadoop.

Hadoop Distributed File System (HDFS): Это распределенная файловая система, которая обеспечивает высокую пропускную способность для доступа к данным.  
  
Hadoop MapReduce: Это модель программирования для больших объемов данных.  
  
Hadoop YARN: Это фреймворк для управления ресурсами и планирования задач.  
  
Hive: Это инструмент для анализа данных, который предоставляет язык запросов, похожий на SQL.  
  
Pig: Это платформа для обработки больших наборов данных, которая предоставляет свой собственный язык запросов.  
  
HBase: Это распределенная база данных, которая работает на основе Hadoop.  
  
Sqoop: Инструмент для переноса данных между Hadoop и реляционными базами данных.  
  
Flume: Сервис для сбора, агрегации и передачи больших объемов лог-данных.  
  
Oozie: Система для управления рабочими процессами и координации задач.  
  
Zookeeper: Сервис для координации процессов в распределенном приложении.  
  
Ambari: Инструмент для мониторинга и управления кластерами Hadoop.  
  
Mahout: Библиотека для машинного обучения и анализа данных.  
  
Spark: Быстрый движок обработки данных для Hadoop.

Hadoop – программный комплекс для хранения и обработки больших объемов слабоструктурированной информации, состоящий из:

- Подсистемы хранения распределенной файловой системы Hadoop – **HDFS**

- Подсистемы автоматического управления ресурсами кластера для балансировки нагрузки – **Yarn**

- Подсистемы пакетной обработки данных с применением дублирования (отображения) и агрегации – **MapReduce**

Hadoop обслуживает распределенный кластер на программном уровне, эмулируя файловую систему Linux.

Hadoop работает на основе Master/Slave архитектуры в двух возможных режимах, невысокой и высокой доступности.

В режиме высокой доступности Hadoop решает проблему единой точки отказа главного узла.

Работа кластера Hadoop организуется за счет служб NameNode, SecondaryNameNode, DataNode.

Подсистема хранения **HDFS** разбивает файлы на блоки фиксированного размера. Размер блока при хранении файлов 64/128 Мб.

Файл хранится в подчиненных узлах **DataNode**. Репликация блоков происходит на ближайших узлах.

Информация о блоках файлов хранится в главном узле **NameNode.**

Все управление распределенной файловой системой происходит через **NameNode.**

Демон (служба) NameNode объединяет виртуально в себе весь ресурс хранения данных, в то время как он связан по сети.

Вот основные правила репликации:  
  
1. Каждый блок данных по умолчанию реплицируется в HDFS три раза. Первая копия блока сохраняется на одном узле, вторая копия на другом узле в той же стойке, а третья копия на узле в другой стойке. Это делается для обеспечения отказоустойчивости.  
  
2. Параметр репликации можно изменить в зависимости от требований к доступности и надежности. Например, для очень важных данных можно установить более высокий уровень репликации.  
  
3. Репликация происходит после записи данных. Когда клиент загружает данные в HDFS, он сначала записывает данные на один из узлов. Затем этот узел передает данные другому узлу, который, в свою очередь, передает данные третьему узлу.  
  
4. Если один из узлов сбоит или теряет данные, HDFS автоматически создает новую копию блока на другом узле, чтобы поддерживать заданный уровень репликации.  
  
5. NameNode управляет всем процессом репликации. Он отслеживает расположение всех блоков и их реплик в системе. Если NameNode обнаруживает, что уровень репликации блока ниже требуемого, он инициирует процесс репликации этого блока.  
  
6. HDFS стремится минимизировать стоимость пропускной способности сети и максимизировать пропускную способность диска при репликации данных.

MapReduce обрабатывает данные ключ:значение по принципу:

1. Разбиение на задачи: Исходные данные разбиваются на более мелкие части, которые будут обрабатываться независимо друг от друга. Этот шаг выполняется в фазе "Map".  
  
2. Применение функции Map: Каждая часть данных обрабатывается функцией Map, которая преобразует входные данные в набор пар ключ-значение.  
  
3. Сортировка и объединение: Пары ключ-значение сортируются и объединяются по ключу, чтобы подготовить их к последующей обработке. Этот шаг выполняется в фазе "Shuffle and Sort".  
  
4. Применение функции Reduce: Пары ключ-значение обрабатываются функцией Reduce, которая выполняет необходимые операции над данными и генерирует выходные результаты.  
  
5. Сбор и вывод результатов: Результаты обработки собираются и выводятся в нужном формате.

Процедура основана на репликации блоков данных между узлами данных.

Данные на этапе Map обрабатываются на местах и происходит запись на диск.

После перемешивания агрегации происходят уже на других исполнителях.

Рассмотрим классический пример применения MapReduce для задачи подсчета количества слов в документах

Для задачи подсчета слов в наборе документов, MapReduce может быть применен следующим образом:  
  
1. Map: В первом этапе каждый документ разбивается на отдельные слова. Затем для каждого слова создается пара ключ-значение, где ключ - это слово, а значение - 1. Этот этап выполняется параллельно для каждого документа в наборе.  
  
2. Shuffle: Результаты Map-функции сортируются и группируются по ключу (слову). Это позволяет объединить все значения для одного слова вместе.  
  
3. Reduce: В этом этапе для каждого уникального слова выполняется суммирование всех значений, полученных на предыдущем этапе. Таким образом, мы получаем общее количество вхождений каждого слова в наборе документов.  
  
Эти три этапа выполняются параллельно на распределенных узлах кластера, что позволяет обрабатывать большие объемы данных эффективно и масштабируемо.  
  
Таким образом, с использованием MapReduce можно эффективно подсчитать количество вхождений каждого слова в наборе документов, даже если этот набор очень большой.

Apache Spark – система пакетной обработки данных в памяти.

В отличие от библиотек Hadoop, которые осуществляют обработку данных на основе чтения/записи в диск, обработка данных происходит в энергозависимой памяти.

Spark обладает большей производительностью в сравнении с Hadoop, но меньшую отказоустойчивость ввиду того, что данные обрабатываются в энергозависимых носителях.  
  
К примеру при работе MapReduce это может существенно повлиять на результат вычислений.

Spark также обеспечивает полный цикл обработки и анализа больших данных за счет обширной экосистемы и библиотек.

YARN – это система планирования заданий и управления кластером (Yet Another Resource Negotiator), которую также называют

MapReduce 2.0 – набор системных программ (демонов), обеспечивающих совместное использование, масштабирование и надежность работы распределенных приложений.

YARN является интерфейсом между аппаратными ресурсами кластера и приложениями, использующих его мощности для вычислений и аналитики больших данных.

***- ResourceManager (RM)***— менеджер ресурсов, которых отвечает за распределение ресурсов, необходимых для работы распределенных приложений, и наблюдение за узлами кластера, где эти приложения выполняются.

ResourceManager включает планировщик ресурсов (Scheduler) и диспетчер приложений (ApplicationsManager, AsM).

***- ApplicationMaster (AM)***– мастер приложения, ответственный за планирование его жизненного цикла, координацию и отслеживание статуса выполнения, включая динамическое масштабирование потребления ресурсов, управление потоком выполнения, обработку ошибок и искажений вычислений, выполнение локальных оптимизаций.

Каждое приложение имеет свой экземпляр ApplicationMaster.

ApplicationMaster выполняет произвольный пользовательский код и может быть написан на любом языке программирования благодаря расширяемым протоколам связи с менеджером ресурсов и менеджером узлов.

***- NodeManager (NM)***– менеджер узла – агент, запущенный на узле кластера, который отвечает за отслеживание используемых вычислительных ресурсов (CPU, RAM и пр.), управление логами и отправку отчетов об использовании ресурсов планировщику.

NodeManager управляет абстрактными контейнерами – ресурсами узла, доступными для конкретного приложения.

***- Контейнер (Container)***— набор физических ресурсов (ЦП, память, диск, сеть) в одном вычислительном узле кластера.

Принцип работы Hadoop YARN можно описать следующим образом:

- клиентское приложение отправляет запрос в кластер;

- менеджер ресурсов выделяет необходимые ресурсы для контейнера и запускает ApplicationMaster для обслуживания этого приложения;

- ApplicationMaster отправляет запрос менеджеру узла NodeManager, включая контекст запуска контейнера Container Launch Context (CLC);

- ApplicationMaster выделяет контейнеры для приложения в каждом узле и контролирует их работу до завершения работы приложения;

- Для запуска контейнера менеджер узла копирует в локальное хранилище все необходимые зависимости (данные, исполняемые файлы, архивы);

- по завершении задачи мастер приложения отменяет выделенный контейнер в диспетчере ресурсов, завершая жизненный цикл распределенного задания.

- клиент может отслеживать состояние распределенного приложения, обращаясь к менеджеру ресурсов или сразу к мастеру приложения.

HDFS, Spark Streaming, Spark, MapReduce и Yarn вместе образуют архитектуру больших данных, которая позволяет обрабатывать и анализировать огромные объемы данных в режиме реального времени. Кратко расскажем как они работают вместе  
  
1. HDFS (Hadoop Distributed File System):

Это основная система хранения данных в Hadoop.

Она разбивает большие объемы данных на блоки и распределяет их по узлам в кластере для параллельной обработки.   
  
2. MapReduce:

Это модель программирования, используемая для обработки и генерации больших наборов данных с параллельными, дистрибутивными алгоритмами в Hadoop.

Она состоит из двух фаз: Map (отображение) и Reduce (сокращение).

Map принимает набор входных данных и преобразует его в другой набор данных,

где отдельные элементы разбиваются на кортежи (ключ/значение), а затем Reduce берет выход Map и объединяет эти данные кортежи в меньший набор выходных данных.  
  
3. YARN (Yet Another Resource Negotiator):

Это фреймворк управления ресурсами в Hadoop, который обеспечивает управление задачами и управление ресурсами кластера.

YARN позволяет различным приложениям обработки данных (например, Spark и MapReduce) работать вместе в одной системе.  
  
4. Spark:

Это быстрый и общий движок обработки данных для больших данных.

Он предоставляет высокоуровневые API на Java, Scala, Python и R, и инструмент интерактивной оболочки для Scala и Python.

Spark может работать с Hadoop и его модулями.

Он использует HDFS для хранения данных и YARN для управления ресурсами.  
  
5. Spark Streaming:

Это компонент Spark, который обеспечивает обработку данных в режиме реального времени.

Он собирает входные данные в мини-пакеты и обрабатывает их с использованием функций Spark's RDD (Resilient Distributed Datasets).   
  
В целом, эти технологии работают вместе для обработки больших данных.

Данные хранятся в HDFS, YARN управляет ресурсами кластера,

MapReduce или Spark используются для обработки данных, а

Spark Streaming используется для обработки данных в реальном времени

Поговорим о потоках данных и обмене данными в системах BigData

**Система пакетной обработки данных –** конвейер обработки данных, состоящий из систем извлечения, обработки и хранения данных, где в качестве единицы обработки информации выделяют контейнер единообразной информации одной структуры.

Например, изменяют или отбирают нужные файлы по заданным критериям.

Выбранное действие обязательно применяется сразу ко всем файлам/байтам/записям в пакете.

Существует несколько основных методов группировки информации по разным контейнерам-пакетам:

- По времени создания. Например, мы кладем в пакет все файлы, поступившие на сервер компании за последние 30 минут. Или все сигналы с сенсоров в турбине самолета за последние четыре полета.

- По типу данных. Видеофайлы в одну кучку, текст — в другую. Тут все очевидно.

- По источнику. Записи о перемещениях грузов на складе кладем в один пакет, записи о перемещении грузов внутри магазина — в другой. Каждая строка в этих данных может выглядеть одинаково и представлять собой запись из базы данных, но источники происхождения у них разные.

- По содержимому. Фотографии котов — в одной папке, фотографии собак — в другой. Современные технологии уже позволяют раскидывать данные по категориям с помощью разных хитрых классификаторов.

- Вручную по различным критериям — самый трудный метод группировки. Его до сих пор применяют на практике, когда данных не так много, а автоматические критерии не подходят. К примеру, когда надо применить один и тот же метод цветокоррекции к фотографиям с фотосессии. Выбор фотографий — дело творческое, поэтому его делают вручную.

**При пакетной обработке данных** с данными в одном пакете может происходить:

**- Применение операций.**

Выбранная операция применяется к каждому элементу пакета.

**- Фильтрация.**

Можно фильтровать файлы внутри пакета — например, оставить в нем только картинки с котами и удалить все остальные.

Или фильтровать пакет в целом — пропускать данные на дальнейшую обработку тогда, когда в нем встретились фотографии только котов.

**Применение**

- Для разделения сложных процессов на мелкие, понятные и легко реализуемые операции.

Разбивка задач на мелкие подзадачи и применение этих подзадач к группам файлов отлично для этого подходит.

- Для того чтобы ускорить работу с данными.

Пакетную обработку данных можно параллелить и запускать в кластерах серверов, то есть сразу на нескольких серверах.

- Комбинация обеих причин — сложные многоступенчатые вычисления на больших объемах данных.

**Примеры применения пакетной обработки данных:**

- обработка данных с применением MapReduce

- стандартная аналитика данных с применением аналитических платформ и языков программирования

Минусы пакетной обработки (batch):

- данные доставляются с задержкой.

Поскольку у нас есть некий период вычислений, то на этот период мы всегда отстаём от реального времени.

И чем больше итераций, тем сильнее мы отстаём.

Таким образом, мы получаем задержку по времени, что в некоторых случаях критично;

- создаётся пиковая нагрузка на железо.

Если мы очень много вычисляем в пакетном режиме, у нас по окончании периода (дня, недели, месяца) наблюдается пик нагрузки, ведь посчитать нужно много всего.

К чему это приводит?

Но у пакетной обработки есть и плюсы:

- высокая эффективность.

Факт заключается в том, что пакетная обработка, если брать количество обработанных записей на единицу времени, будет эффективнее потоковой обработки данных по всем параметрам (среднее время на запись)

- простота разработки и поддержки.

Вы можете обрабатывать какую-нибудь часть данных, тестируя и пересчитывая по мере необходимости.

**Потоковая обработка** это однопроходная парадигма обработки данных,

которая всегда поддерживает данные в движении для достижения низкой задержки обработки.

Будучи более высокой абстракцией систем обмена сообщениями,

потоковая обработка поддерживает не только агрегацию и доставку сообщений,

но и способна выполнять асинхронные вычисления в реальном времени при передаче информации.

Потоковая обработка данных всецело базируется на системах обмена сообщениями, о которых мы говорили во второй лекции.

На слайде представлен пример с использованием брокера сообщений ApacheKafka.

Плюсы потоковой обработки данных (streaming):

- результат в режиме реального времени.

Мы не ждём конца каких-либо периодов: как только к нам приходят данные (пусть даже совсем небольшой объём), мы можем сразу их обработать и передать дальше.

То есть результат по определению стремится к реальному времени;

- равномерная нагрузка на железо.

Понятное дело, что есть суточные циклы и т. д.,

однако нагрузка всё равно распределяется на весь день и получается более равномерной и предсказуемой.

**Главный минус потоковой обработки:**

- сложность разработки и поддержки.

Во-первых, тестироваться, управлять и получать данные несколько тяжелее, если сравнивать с пакетной обработкой данных.

- Вторая сложность (на самом деле, это самая основная проблема) связана с откатами.

Если обработчики по приему и передаче сообщений не отработали, и произошёл сбой, то очень трудно уловить именно тот момент, где всё сломалось.

И решение проблемы потребует от вас больше усилий и ресурсов по сравнению с пакетной обработкой.

**Элементы системы потоковой обработки данных**

* **Загрузчик данных** (средство доставки данных до хранилища);

Apache Flume или Apache NIFI, StreamSets

**2. Шина обмена данными** (нужна не всегда, но в стримах без неё никак, т. к. вам потребуется система, через которую вы будете обмениваться данными в реал-тайме);

ApacheKafka, RabbitMQ, NATS

**3. Хранилище данных** (как же без него);

Apache HDFS+Hive, Apache Kudu+Impala, Yandex ClickHouse

**4.** **ETL-движок** (необходим, чтобы делать различные фильтрации, сортировки и прочие операции);

**5. BI** (чтобы выводить результаты);

**6. Оркестратор** (связывает весь процесс воедино, организовывая многоэтапную обработку данных).

Лямбда архитектура поддерживает параллельную обработку пакетных данных и потоковых данных на основе параллельного исполнения служб в слое пакетной и скоростной обработки.

**Преимущества:**

Скорость обработки

Простота построения ахритектуры

Отсутствие слияния двух потоков

**Недостатки:**

Высокая нагрузка на вычислительный кластер

Большой объем данных для хранения

Необходимость в запросах из двух потоков

Отличается от лямбда-архитектуры в слиянии направлений обработки данных: потоковой и пакетной.

Данные хранятся в одном хранилище данных.

Запросы можно производить к единому хранилищу данных.

Для каппа-архитектуры результаты потоковой обработки не нужны в оперативных отчетностях, а только в виде диагностической информации в случае проблем.

Теперь поговорим об облачных сервисах вычислений, то есть тех, кто предоставляет нам возможность работать с большими объемами данных в реальности.

Прежде чем облачные вычисления стали общедоступными, компаниям приходилось хранить свои данные и устанавливать программное обеспечение на собственных серверах. Больше всего это касалось крупных организаций, которым требовалась масса локальной инфраструктуры.

Хоть все казалось под контролем, но был один существенный недостаток - отсутствие возможности быстрого масштабирования при увеличении нагрузки.

Например, во время распродаж количество онлайн-заказов может резко увеличиться, и серверы могут не выдержать этого напора, что приводит к потере прибыли.

Решением может быть установка дополнительного оборудования, но это дорого и неэффективно, поскольку большую часть времени оно будет бездействовать.

Кроме того, покупка, установка и настройка оборудования занимают много времени, которого зачастую не хватает.

Облачные технологии предоставляют гибкость в управлении инфраструктурой.

Благодаря виртуализации, собрать необходимое количество серверов с требуемой мощностью становится быстрым и простым процессом.

Параметры, такие как GPU, RAM, объем диска и другие, могут быть настроены всего за несколько кликов.

Если текущая конфигурация не справляется с нагрузкой, дополнительные ресурсы могут быть добавлены через панель управления.

Функция автомасштабирования позволяет автоматизировать этот процесс.  
  
Таким образом, нет необходимости приобретать дорогостоящее оборудование, которое будет бездействовать большую часть времени.

Вместо этого, его можно арендовать у облачного провайдера и использовать по мере необходимости.

Использование облачных вычислений помогает минимизировать время простоя оборудования и более эффективно использовать ресурсы.

В связи с этим, многие организации различного типа, размера и отрасли переходят на облачную модель работы, так как это выгодно и удобно.  
  
К тому же, облачные технологии упрощают организацию IT-систем.

В любое время можно получить необходимое количество отлично функционирующих серверов, кластеров или СУБД.

Это действительно удобно, так как нет необходимости в их настройке и обновлении - об этом заботится провайдер облачных услуг.

Облачные технологии предоставляют возможности для решения различных проблем.

Благодаря виртуализации и облачным технологиям, можно, например,

- создать более надежную систему для резервного копирования и аварийного восстановления в случае непредвиденных обстоятельств,

- внедрить DevOps-методологии в процесс разработки,

- обучить искусственный интеллект,

- работать с Big Data и так далее.

Так, финансовые учреждения применяют облачные вычисления для формирования цифровых профилей пользователей и усовершенствования систем противодействия мошенничеству, обеспечивая защиту средств клиентов.

Разработчики видеоигр и медиа-компании могут использовать облачные платформы для рендеринга объектов и выполнения задач, связанных с доставкой контента (CDN).

Разница между облачными решениями и локальным хостингом становится более очевидной.

Традиционный хостинг предполагает аренду вычислительных ресурсов на определенный период.

Обычно объем предоставляемых ресурсов фиксирован, и быстро масштабировать арендованные мощности при изменении нагрузки не получится.

Приходится дополнительно привлекать ресурсы.

Использование облачных ресурсов дает организации масштабируемую и отказоустойчивую инфраструктуру,

где можно быстро запускать и останавливать виртуальные машины, изменять их конфигурацию и передавать данные между ними.

Какие еще преимущества предоставляют облачные технологии?

Гибкость.

Всего за несколько минут можно подключить больше ресурсов для выполнения "тяжелых" вычислительных задач,

развернуть несколько десятков виртуальных рабочих столов для новых сотрудников,

создать тестовую среду для нового приложения - облачные вычисления предоставляют компаниям мобильность и гибкость,

которые недоступны при использовании локальной инфраструктуры.

Эластичность.

Нет необходимости заранее покупать оборудование "на всякий случай".

Благодаря облакам, можно получить только те ресурсы, которые нужны для решения текущих задач.

Оборудование не остается без дела, а организация не зависит от закупок.

Экономия бюджета.

Облачные вычисления автоматизируют и удешевляют использование IT-инфраструктуры.

Например, облачный сервер можно включить на два часа и оплатить только это время, так как провайдеры предлагают оплату по факту использования.

При масштабировании IT-парка экономия становится значительной: нет лишних расходов.

Снижение издержек.

Поддержка собственной вычислительной инфраструктуры требует затрат на IT-специалистов, регулярные обновления программного обеспечения и другие процедуры.

Облака позволяют переложить эту ответственность на провайдера.

Организации получают производительное оборудование с актуальным ПО, обслуживаемое квалифицированными специалистами.

Стабильность.

Условия непрерывной работы виртуальной инфраструктуры указываются в договоре с провайдером (в рамках SLA).

Таким образом, организации получают гарантии стабильной работы своих сервисов и финансовую защиту в случае возникновения проблем.

Управляемость.

Организация сама может решить, какие ресурсы и в каком объеме она будет использовать.

Для этого не нужно обращаться к менеджерам провайдера или оставлять заявки в службу поддержки.

С помощью личного кабинета или терминала можно в любое время задать необходимые параметры.

Безопасность.

Уровень компетенции сотрудников облачного провайдера обычно выше, чем у сотрудников клиентских компаний.

Кроме того, провайдер использует оборудование и программное обеспечение промышленного уровня, что увеличивает безопасность и надежность IT-систем.

Традиционно выделяют три основные модели облачных вычислительных услуг, предоставляемые провайдерами: IaaS, PaaS и SaaS.

Они наиболее востребованы на рынке.

Однако существуют и другие модели оказания услуг, которые стали набирать популярность в последние годы.

Например, DBaaS, бессерверные и FaaS-вычисления.

Давайте посмотрим, чем они отличаются.

IaaS  
В рамках модели IaaS (Infrastructure as a Service), пользователи имеют доступ к виртуальным серверам, сетям, хранилищам и другим облачным вычислительным ресурсам на основном уровне. Это подобно традиционному подходу к работе с инфраструктурой, который знаком большинству IT-отделов.

Основное различие заключается в том, что оборудование располагается в дата-центре облачного сервис-провайдера.

Модель IaaS обеспечивает гибкость в управлении ресурсами, позволяя быстро адаптироваться к текущим потребностям.

PaaS  
Модель PaaS (Platform as a Service) предусматривает, что пользователь не управляет серверами, хранилищами и приложениями.

Вместо этого, он выбирает из доступного списка серверы и среды, необходимые для запуска, тестирования, развертывания, поддержки, обновления и масштабирования своих приложений.

Это увеличивает эффективность работы, позволяя сосредоточиться на развертывании и управлении приложениями.

Решения PaaS обычно основаны на технологии контейнеризации.

SaaS  
Модель SaaS (Software as a Service) предоставляет пользователям готовый продукт, который запускается и управляется облачным сервис-провайдером.

Доступ к приложению может быть осуществлен через браузер, API или клиентское приложение на устройстве пользователя.

Большинство платных программных продуктов предоставляются по этой модели, включая корпоративные базы данных, CRM, корпоративную почту, защиту от DDoS и Kubernetes.

Преимущества этой модели включают автоматическое обновление приложений и защиту от потери данных.

Существуют пять типов облаков: публичное, частное, мультиоблако, гибридное и Community Cloud. Каждый тип решает свои задачи.

**Публичное облако.**

Так называется один или несколько дата‑центров, управляемых провайдером облачных услуг.

Он обеспечивает работу облака и через интернет делает доступными все сервисы дата‑центра: от SaaS‑приложений до виртуальных машин.

Сервисы предоставляются по подписке или с платой за пользование.

Так как облачные сервисы обладают высокой эластичностью и масштабируемостью, многие организации переносят в них часть своей инфраструктуры.

Облачный провайдер берёт на себя ответственность за бесперебойную работу оборудования и инфраструктуры дата‑центра и обеспечивает быстрый доступ к приложениям и данным.

**Частное облако.**

Так называется среда, в которой вся инфраструктура и вычислительные ресурсы принадлежат и эксплуатируются одной организацией.

Частное облако обладает всеми названными преимуществами облачных вычислений, к которым добавляется контроль доступа, безопасность и настройка ресурсов под свои нужды.

Виртуальные машины, хранилища и базы данных в такой сети изолированы от трафика других клиентов и защищены от DDoS‑атак.

**Гибридное облако.**

Комбинация облачного и частного облака.

В этой модели организация использует ресурсы собственного дата‑центра, а в публичном облаке разворачивает отдельные сервисы, например платформу машинного обучения или систему оркестровки контейнеров.

Предназначение гибридного облака заключается в создании единой масштабируемой среды с сохранением контроля организации над критически важными данными.

**Мультиоблако.**

Подход, при котором используются решения нескольких облачных провайдеров.

Такая модель развёртывания позволяет переносить данные от одного облачного провайдера к другому или одновременно развёртывать приложения и платформы у нескольких облачных провайдеров.

Подход минимизирует вероятность сбоев, повышая надёжность и отказоустойчивость IT‑инфраструктуры.

**Community Cloud.**

Это облако, которым пользуется ограниченное число компаний с похожими ценностями, например финансовые организации.

При этом не важно, управляется ли оно сторонним облачным провайдером или принадлежит пользователям.