[**1.** **Системные и локальные шины. Организация ввода/вывода в вычислительной системе.** 4](#_Toc168418026)

[**2.** **Иерархия запоминающих устройств. Характеристики, принципы построения и функционирования запоминающих устройств. Организация виртуальной памяти.** 5](#_Toc168418027)

[**3.** **Основные типы и характеристики вычислительных систем.** 7](#_Toc168418028)

[**4.** **Организация автоматической работы ЭВМ. Управляющие функции процессора. Общая организация выполнения программы на ЭВМ.** 8](#_Toc168418029)

[**5.** **Принципы фон-Неймана по построению вычислительных систем.** 9](#_Toc168418030)

[**6.** **Общая характеристика языка программирования Python. Схема выполнения. Система типов. Сравнение с другими языками.** 10](#_Toc168418031)

[**7.** **Списки в Python. Назначение, особенности, создание, изменение. Основные операции и методы. Способы итерации по спискам.** 11](#_Toc168418032)

[**8.** **Словари в Python. Назначение, особенности, создание, изменение. Основные операции и методы. Способы итерации по словарям.** 12](#_Toc168418033)

[**9.** **Связные списки: однонаправленные и двунаправленные – принцип реализации. Сравнение скорости выполнения основных операций в связных списках и в динамическом массиве.** 13](#_Toc168418034)

[**10.** **Основные алгоритмы сортировки. Их сложность, достоинства и недостатки. Алгоритм быстрого поиска в отсортированном массиве. Сложность поиска в отсортированном и не отсортированном массиве.** 15](#_Toc168418035)

[**11.** **Основные технологии, использующиеся для разработки веб-приложений. Языки разметки и программирования, технологии, библиотеки и фреймворки.** 16](#_Toc168418036)

[**12.** **Общая характеристика языка разметки HTML. Основные элементы - теги, атрибуты, значения. Структура веб-страницы. Единицы измерения и цвета в HTML.** 17](#_Toc168418037)

[**13.** **Общая характеристика языка CSS. Назначение, синтаксис, использование. Виды селекторов.** 18](#_Toc168418038)

[**14.** **Общая характеристика языка JavaScript. Синтаксис, назначение, система типов. Среда исполнения. Особенности выполнения на клиенте и на сервере.** 19](#_Toc168418039)

[**15.** **Серверное программирование. Особенности, обзор технологий и языков программирования. Распространенные серверные фреймворки.** 20](#_Toc168418040)

[**16.** **Технология взаимодействия клиентской и серверных частей веб-приложения. Протокол HTTP. Запросы и ответы.** 21](#_Toc168418041)

[**17.** **Адресация в интернете. Основные понятия о URI, URL, URN. Структура адресов, использование.** 22](#_Toc168418042)

[**18.** **Эволюция методологий моделирования. Моделирование деятельности и моделирование процессов. Использование методологии ARIS для моделирования деятельности предприятия. Модели, атрибуты моделей, действия над моделями, типы моделей. Объекты, свойства объектов Связи, свойства связей. Техническая реализация методологии моделирования.** 23](#_Toc168418043)

[**19.** **Разделение прав доступа и система безопасности современных многопользовательских операционных систем на примере Linux. Пользователи и группы, основные права. Основные команды для управления правами и пользователями.** 24](#_Toc168418044)

[**20.** **Работа в командной строке bash. Основные понятия и структура команды. Основные команды для перемещения по каталогам, просмотра файлов и информации о содержимом файлов.** 25](#_Toc168418045)

[**21.** **Настройка и диагностика сетевых подключений в современных операционных системах на примере Linux. Удаленный доступ к операционной системе по протоколу SSH.** 27](#_Toc168418046)

[**22.** **Основные сетевые службы. Веб-технологии - необходимое программное обеспечение, назначение схема работы, основные протоколы.** 27](#_Toc168418047)

[**23.** **Общая характеристика языка программирования Java. Схема выполнения, синтаксис, система типов.** 28](#_Toc168418048)

[**24.** **Объектно-ориентированное программирование в Java. Принципы, особенности реализации. Синтаксис.** 29](#_Toc168418049)

[**25.** **Фреймворк Spring. Назначение, основные понятия, применение. Объектно-реляционное отображение (ORM).** 30](#_Toc168418050)

[**26.** **Понятие, классификация и принципы построения банковских информационных систем (БИС).** 32](#_Toc168418051)

[**27.** **Реализация решения задач кредитного модуля банковских информационных систем.** 33](#_Toc168418052)

[**28.** **Применение технологии Big Data в банковской сфере.** 35](#_Toc168418053)

[**29.** **Системы дистанционного банковского обслуживания.** 36](#_Toc168418054)

[**30.** **Информационные системы бухгалтерского учета (ИС БУ), их структура и классификация.** 37](#_Toc168418055)

[**31.** **Подходы к классификации бухгалтерских информационных систем, их и различия.** 39](#_Toc168418056)

[**32.** **Модель системы счетов как основа любой бухгалтерской информационной системы. Модель организации синтетического учета. Модели организации аналитического учета.** 40](#_Toc168418057)

[**33.** **Понятие базы данных. Виды баз данных. Проектирование баз данных.** 41](#_Toc168418058)

[**34.** **Общая характеристика языка запросов SQL. Декларативная парадигма. Основные операторы, виды операторов.** 42](#_Toc168418059)

[**35.** **Оператор SELECT в SQL. Общий синтаксис, наиболее распространенные блоки. Условия, сортировки, ограничения. Примеры.** 44](#_Toc168418060)

[**36.** **Оператор JOIN в SQL. Общий синтаксис. Виды объединений, их различия и области применений. Примеры.** 45](#_Toc168418061)

[**37.** **Использование групповых функций в SQL. Синтаксис оператора SELECT при использовании группировок. Примеры использования.** 47](#_Toc168418062)

[**38.** **Реляционные базы данных. Нормализация отношений. Первая нормальная форма. Вторая нормальная форма. Третья нормальная форма. Другие нормальные формы.** 50](#_Toc168418063)

[**39.** **Транзакции в SQL. Понятие транзакции, организация транзакций, вложенные транзакции. Транзакции и управление параллельной работой.** 51](#_Toc168418064)

[**40.** **Типы задач машинного обучения: классификация, регрессия, кластеризация. Обучение с учителем и без учителя. Машинное обучение как оптимизационная задача.** 53](#_Toc168418065)

[**41.** **Регрессия как задача машинного обучения. Постановка задачи, математическая формализация. Целевая функция. Примеры.** 55](#_Toc168418066)

[**42.** **Классификация как задача машинного обучения. Постановка задачи, математическая формализация. Целевая функция. Примеры.** 56](#_Toc168418067)

[**43.** **Метрики качества результатов машинного обучения. Метрики качества моделей регрессии и классификации. MAE, MSE, Precision, Recall, F1, ROC-AUC.** 57](#_Toc168418068)

[**44.** **Метод ближайших соседей (kNN) и его обобщения. Подбор числа k. Методы метрической классификации в библиотеке Scikit-Learn.** 58](#_Toc168418069)

[**45.** **Кластеризация как задача машинного обучения. Постановка задачи, математическая формализация. Целевая функция. Примеры.** 60](#_Toc168418070)

[**46.** **Структура проекта нативного Android приложения. Основные файлы. Манифест приложения. Файлы с разметкой. Файлы ресурсов. Использование альтернативных ресурсов. Используемые языки для разработки и описания приложения.** 61](#_Toc168418071)

[**47.** **Компоненты Android приложений. Основные виды компонентов, их особенности и назначение. Жизненный цикл компонентов и использование методов жизненного цикла.** 62](#_Toc168418072)

[**48.** **Хранение данных в Android приложении. Виды хранилищ данных, их особенности и использование из приложения.** 64](#_Toc168418073)

[**49.** **Виды мобильных приложений - нативные, веб, гибридные. Сравнительная характеристика. Используемые технологии и языки программирования.** 65](#_Toc168418074)

[**50.** **Библиотеки Python для обработки больших данных. Обработка данных, превышающих объем доступной памяти. Библиотека Dask.** 66](#_Toc168418075)

[**51.** **Библиотеки Python для обработки данных. Numpy, pandas - основные понятия, возможности, назначение, примеры использования. Основные форматы структурированных данных.** 68](#_Toc168418076)

1. **Системные и локальные шины. Организация ввода/вывода в вычислительной системе.**

В вычислительной системе, состоящей из множества подсистем, необходим механизм для их взаимодействия. Эти подсистемы должны быстро и эффективно обмениваться данными. Например, процессор, с одной стороны, должен быть связан с памятью, с другой стороны, необходима связь процессора с устройствами ввода/вывода. Одним из простейших механизмов, позволяющих организовать взаимодействие различных подсистем, является единственная центральная шина, к которой подсоединяются все подсистемы.

Традиционно шины делятся на шины, обеспечивающие организацию связи процессора с памятью, и шины ввода/вывода. Шины ввода/вывода могут иметь большую протяженность, поддерживать подсоединение многих типов устройств и обычно следуют одному из шинных стандартов. Шины процессор-память, с другой стороны, сравнительно короткие, обычно высокоскоростные и соответствуют организации системы памяти для обеспечения максимальной пропускной способности канала память-процессор. На этапе разработки системы для шины процессор-память заранее известны все типы и параметры устройств, которые должны соединяться между собой, в то время как разработчик шины ввода/вывода должен иметь дело с устройствами, различающимися по задержке и пропускной способности.

С целью снижения стоимости некоторые компьютеры имеют единственную шину для памяти и устройств ввода/вывода. Такая шина часто называется системной. Персональные компьютеры, как правило, строятся на основе одной системной шины.

Необходимость сохранения баланса производительности по мере роста быстродействия микропроцессоров привела к двухуровневой организации шин в персональных компьютерах на основе локальной шины.

Локальной шиной называется шина, электрически выходящая непосредственно на контакты микропроцессора. Она обычно объединяет процессор, память, схемы буферизации для системной шины и ее контроллер, а также некоторые вспомогательные схемы.

Системные и локальные шины представляют собой коммуникационные каналы, используемые в вычислительных системах для передачи данных между различными компонентами.

Системная шина (например, шина PCI Express) обеспечивает связь между центральным процессором, оперативной памятью и другими устройствами в компьютере. Она обеспечивает высокую пропускную способность и низкую задержку передачи данных.

Локальные шины (например, шина USB, шина SATA) предназначены для подключения периферийных устройств к компьютеру. Они обеспечивают простой и универсальный способ подключения устройств, таких как клавиатура, мышь, принтеры, жесткие диски и другие внешние устройства.

Организация ввода/вывода в вычислительной системе означает способ, которым компьютер взаимодействует с внешними устройствами для передачи данных. Ввод данных происходит при получении информации от пользователя или другого источника, например клавиатуры или мыши. Вывод данных происходит при отправке информации на внешние устройства, например монитор или принтер.

1. **Иерархия запоминающих устройств. Характеристики, принципы построения и функционирования запоминающих устройств. Организация виртуальной памяти.**

Иерархия запоминающих устройств в компьютерных системах организована так, чтобы оптимально использовать каждый тип памяти в зависимости от его скорости, объема, стоимости и назначения. Иерархия состоит из нескольких уровней.

На самом верхнем уровне находятся регистры процессора. Они обладают самой высокой скоростью, но их объем очень небольшой, всего несколько десятков регистров. Регистры используются для хранения промежуточных данных, операндов и результатов операций, выполняемых процессором.

Следующий уровень — это кэш-память. Она делится на три уровня: L1, L2 и L3. L1 - самая быстрая и находится прямо в процессоре. L2 может быть как встроенной, так и внешней, и она медленнее, чем L1. L3 - самая медленная из кэшей, но ее объем больше, и она общая для нескольких ядер процессора. Кэш-память используется для ускорения доступа к часто используемым данным и инструкциям.

После кэш-памяти идет оперативная память (RAM). Она значительно больше по объему, обычно несколько гигабайт, и достаточно быстрая, хотя и медленнее кэша. Оперативная память используется для хранения данных и программ, которые активно используются процессором.

Виртуальная память является следующим уровнем. Она позволяет системе использовать больше памяти, чем имеется физически, за счет использования части дискового пространства. Виртуальная память работает за счет механизма таблиц страниц, которые отображают виртуальные адреса в физические. Это помогает расширить доступную память, защитить память каждого процесса и изолировать процессы друг от друга.

Далее идет внешняя память, такая как жесткие диски (HDD) и твердотельные накопители (SSD). Они имеют большие объемы (терабайты и больше), но их скорость значительно ниже по сравнению с оперативной памятью. Внешняя память используется для долговременного хранения данных.

На самом нижнем уровне находятся архивные устройства, такие как оптические диски и магнитные ленты. Они имеют самый большой объем и самую низкую скорость, предназначены для долговременного хранения данных.

Характеристики запоминающих устройств включают скорость доступа, объем памяти, стоимость на единицу объема, энергопотребление и надежность. Скорость доступа измеряется временем, необходимым для чтения или записи данных. Объем определяется количеством данных, которое может быть сохранено. Стоимость измеряется затратами на хранение определенного объема данных. Энергопотребление — это количество энергии, необходимое для работы памяти. Надежность показывает, как долго данные могут сохраняться без ошибок.

Основные принципы построения запоминающих устройств включают локальность ссылок, кэширование, пагинацию и сегментацию, а также параллелизм. Локальность ссылок подразумевает, что данные, расположенные близко друг к другу, вероятнее всего будут использоваться вместе (пространственная локальность) и что недавно использованные данные с высокой вероятностью будут использованы снова (временная локальность). Кэширование использует небольшую, но быструю память для хранения часто используемых данных, что ускоряет доступ к ним. Пагинация и сегментация помогают эффективно управлять и защищать память, разделяя ее на страницы и сегменты. Параллелизм позволяет использовать несколько модулей памяти для одновременного доступа и увеличения скорости работы.

Организация виртуальной памяти основывается на механизмах пагинации и сегментации. Пагинация делит память на блоки фиксированного размера, называемые страницами, и использует таблицы страниц для отображения виртуальных адресов в физические. Сегментация делит память на сегменты переменного размера и использует таблицы сегментов для управления ими. Комбинированный подход объединяет оба метода для большей гибкости. Преимущества виртуальной памяти включают расширение доступной памяти, защиту памяти и изоляцию процессов.

1. **Основные типы и характеристики вычислительных систем.**

Вычислительные системы разнообразны и классифицируются по различным признакам, таким как производительность, область применения и архитектура. Основные типы вычислительных систем включают суперкомпьютеры, мейнфреймы, серверы, персональные компьютеры (ПК), мобильные устройства и встроенные системы.

Суперкомпьютеры — это самые мощные вычислительные системы, предназначенные для выполнения сложных и ресурсоемких вычислений. Они характеризуются очень высокой производительностью, которая измеряется в петафлопсах, то есть квадриллионах операций с плавающей точкой в секунду. Суперкомпьютеры используют тысячи и даже миллионы процессоров, работающих параллельно, что позволяет им обрабатывать огромные объемы данных одновременно. Они применяются в научных исследованиях, моделировании климатических изменений, биомедицинских вычислениях и других областях, требующих интенсивных вычислительных ресурсов. Примером суперкомпьютера является Summit, разработанный IBM для Национальной лаборатории Ок-Ридж.

Мейнфреймы — это мощные серверные системы, используемые для обработки больших объемов данных и выполнения критически важных задач. Они обладают высокой производительностью для обработки транзакций и управления большими базами данных. Мейнфреймы отличаются высокой надежностью и отказоустойчивостью, способны работать без сбоев в течение длительного времени. Эти системы легко масштабируются для поддержки большего числа пользователей и объемов данных. Мейнфреймы широко применяются в банках, страховых компаниях и правительственных организациях. Примером мейнфрейма является IBM Z series.

Серверы — это вычислительные системы, которые предоставляют ресурсы и услуги другим компьютерам в сети. Производительность серверов варьируется от средних до высоких значений в зависимости от задач, которые они выполняют. Они обладают высокой масштабируемостью, позволяя увеличивать количество пользователей и нагрузку на систему. Серверы также характеризуются высокой надежностью и поддержкой отказоустойчивости. Они используются в качестве веб-серверов, почтовых серверов, файловых серверов и серверов баз данных. Примеры серверов включают Dell PowerEdge и HP ProLiant.

Персональные компьютеры — это универсальные вычислительные устройства, используемые индивидуальными пользователями для выполнения разнообразных задач. ПК обладают достаточной производительностью для повседневных задач, таких как офисная работа, интернет-серфинг, игры и мультимедиа. Они легко модернизируются и обновляются, что позволяет пользователям улучшать их характеристики по мере необходимости. ПК используются как в домашних условиях, так и в офисах, а также в качестве рабочих станций. Примеры персональных компьютеров включают Lenovo ThinkPad, Apple iMac и Dell Inspiron.

Мобильные устройства включают смартфоны и планшеты, предназначенные для портативного использования. Они обладают средней производительностью, достаточной для работы мобильных приложений, веб-серфинга и мультимедиа. Мобильные устройства характеризуются компактными размерами и легким весом, а также интегрированными беспроводными коммуникациями, такими как Wi-Fi, Bluetooth и мобильная связь. Они используются для повседневных задач, коммуникаций, мультимедиа и навигации. Примеры мобильных устройств включают iPhone, Samsung Galaxy и iPad.

Встроенные системы — это специализированные вычислительные системы, интегрированные в устройства для управления их работой. Они оптимизированы для выполнения конкретных задач и характеризуются высокой надежностью и устойчивостью к внешним условиям. Встроенные системы применяются в промышленной автоматике, бытовой электронике, автомобильных системах и медицинских устройствах. Примеры встроенных систем включают микроконтроллеры в бытовой технике и системы управления в автомобилях.

1. **Организация автоматической работы ЭВМ. Управляющие функции процессора. Общая организация выполнения программы на ЭВМ.**

Организация автоматической работы ЭВМ заключается в том, что компьютерная система выполняет запрограммированную последовательность операций без непосредственного участия пользователя. Основной компонент, ответственный за это, - процессор, который выполняет множество управляющих функций. Процессор, или центральный процессорный блок (ЦПУ), отвечает за извлечение, декодирование и выполнение команд из памяти, а также за управление переходами между этими командами.

Когда программа загружается в оперативную память компьютера, процессор начинает свою работу с извлечения первой инструкции из памяти. Эта инструкция попадает в специальный регистр, называемый счетчиком команд, который хранит адрес следующей инструкции, подлежащей выполнению. После извлечения процессор декодирует инструкцию, то есть определяет, какое действие необходимо выполнить. Например, это может быть арифметическая операция, передача данных или переход к другой части программы.

Выполнив декодирование, процессор приступает к выполнению инструкции. Это может включать выполнение математических операций, перемещение данных между регистрами или памятью, а также взаимодействие с периферийными устройствами. Процессор работает очень быстро, выполняя миллионы или даже миллиарды инструкций в секунду, в зависимости от его архитектуры и тактовой частоты.

Управление потоком выполнения программы - одна из ключевых функций процессора. Оно включает в себя управление условными и безусловными переходами, циклами и подпрограммами. Процессор использует специальные команды для изменения значения счетчика команд, что позволяет переходить к выполнению других частей программы в зависимости от условий, установленных предыдущими инструкциями.

Организация выполнения программы на ЭВМ также включает взаимодействие с оперативной памятью и различными периферийными устройствами. Процессор считывает данные из памяти, обрабатывает их и записывает результаты обратно в память или отправляет на внешние устройства. Этот процесс включает в себя эффективное управление ресурсами, такими как кэш-память, шины данных и системы ввода-вывода, что обеспечивает высокую производительность и надежность системы.

1. **Принципы фон-Неймана по построению вычислительных систем.**

Архитектура фон-Неймана стала основой для большинства современных вычислительных систем и описывает структуру компьютера и принципы его работы. Один из ключевых аспектов этой архитектуры заключается в том, что данные и программы хранятся в одной и той же памяти. Это значит, что компьютер может обрабатывать инструкции так же, как и данные, что делает возможным выполнение программ, которые могут изменять сами себя или другие программы.

В архитектуре фон-Неймана программы состоят из последовательности команд, которые выполняются процессором поочередно, одна за другой. Это обеспечивает предсказуемость и упрощает управление потоком выполнения программы. Память организована в виде адресуемого пространства, где каждая ячейка памяти имеет уникальный адрес. Процессор использует эти адреса для извлечения инструкций и данных из памяти.

Принцип хранимой программы является еще одним важным элементом. Программа, подлежащая выполнению, хранится в оперативной памяти вместе с данными. Это позволяет процессору быстро и эффективно извлекать и выполнять команды, что ускоряет выполнение программ и упрощает их изменение и обновление.

Центральный процессор, или ЦПУ, выполняет все арифметические и логические операции, извлекает инструкции из памяти, декодирует их и выполняет. ЦПУ состоит из нескольких ключевых компонентов, включая арифметико-логическое устройство (АЛУ), регистры и блок управления. Команды выполняются процессором последовательно, то есть одна команда за раз. Хотя современные процессоры могут выполнять несколько команд одновременно благодаря параллелизму, базовый принцип последовательного выполнения остается актуальным для понимания архитектуры фон-Неймана.

1. **Общая характеристика языка программирования Python. Схема выполнения. Система типов. Сравнение с другими языками.**

Python – это высокоуровневый язык программирования, который был разработан Гвидо ван Россумом и выпущен в 1991 году. Он стал популярен благодаря своей простоте и читабельности, что делает его отличным выбором как для начинающих, так и для опытных программистов. Python поддерживает несколько парадигм программирования, включая объектно-ориентированное, процедурное и функциональное программирование. Основные особенности Python включают в себя динамическую типизацию, автоматическое управление памятью и обширную стандартную библиотеку.

Программы на Python выполняются интерпретатором, что означает, что код Python исполняется построчно, без необходимости предварительной компиляции в машинный код. Когда вы запускаете скрипт на Python, интерпретатор сначала компилирует исходный код в байт-код, промежуточное представление, которое затем исполняется виртуальной машиной Python (PVM). Этот процесс делает разработку на Python быстрой и интерактивной, так как изменения в коде могут быть немедленно протестированы.

Python обладает динамической и сильной системой типов. Динамическая типизация означает, что тип переменной определяется во время выполнения, а не во время компиляции. Это позволяет писать более гибкий и лаконичный код. Сильная типизация означает, что Python не позволяет автоматически приводить один тип к другому, если это может привести к потере данных или неожиданному поведению. Например, попытка сложить строку и число вызовет ошибку, что помогает избежать множества ошибок на этапе выполнения программы.

В сравнении с другими языками программирования Python имеет несколько ключевых отличий:

* C++: C++ – это язык со статической типизацией и требует компиляции перед выполнением. В отличие от Python, он предоставляет более низкоуровневый контроль над памятью, что может привести к более эффективному коду, но также делает его более сложным в использовании и требует больше усилий для управления ресурсами.
* Java: Java также является языком со статической типизацией и требует компиляции. Как и Python, Java имеет богатую стандартную библиотеку и поддерживает объектно-ориентированное программирование. Однако синтаксис Java более сложный и многословный по сравнению с Python, что может усложнять разработку.
* JavaScript: JavaScript, как и Python, является интерпретируемым языком с динамической типизацией. Он в первую очередь используется для веб-разработки, тогда как Python находит применение в широком спектре областей, включая науку о данных, машинное обучение, веб-разработку и автоматизацию.

1. **Списки в Python. Назначение, особенности, создание, изменение. Основные операции и методы. Способы итерации по спискам.**

Списки в Python – это важная и часто используемая структура данных, которая позволяет хранить коллекции элементов. Их основное назначение заключается в том, чтобы хранить упорядоченные наборы данных, которые можно изменять по мере необходимости. Особенность списков в том, что они динамически изменяемы, то есть их размер может изменяться во время выполнения программы. В одном списке можно хранить элементы различных типов, например числа, строки и даже другие списки.

Создать список в Python очень просто. Самый распространенный способ – использовать квадратные скобки. Например, my\_list = [1, 2, 3, 'apple', [4, 5]] создаст список, содержащий числа, строку и даже вложенный список. Также можно использовать функцию list(), например, my\_list = list([1, 2, 3]).

Списки легко изменяются. Можно добавлять новые элементы с помощью метода append(), который добавляет элемент в конец списка. Если нужно добавить элемент на определенную позицию, используется метод insert(), например, my\_list.insert(1, 'orange'), что вставит 'orange' на вторую позицию. Для удаления элементов есть методы remove(), который удаляет первый найденный элемент, и pop(), который удаляет и возвращает последний элемент списка. Также можно изменять значения элементов с помощью индексации, например, my\_list[0] = 'grape' изменит первый элемент на 'grape'.

Списки поддерживают множество полезных операций и методов. Например, сложение списков позволяет объединить два списка, new\_list = my\_list + [6, 7, 8]. Операция умножения повторяет список несколько раз, repeated\_list = my\_list \* 2. Метод len() возвращает количество элементов в списке. Списки можно сортировать с помощью метода sort(), который сортирует элементы на месте, и функции sorted(), которая возвращает новый отсортированный список. Метод reverse() изменяет порядок элементов на противоположный.

Итерация по элементам списка может быть выполнена несколькими способами. Самый простой – это использование цикла for, например, for item in my\_list: print(item), что позволяет пройтись по всем элементам списка. Также можно использовать функцию enumerate(), которая возвращает индекс и значение элемента, for index, item in enumerate(my\_list): print(index, item). Генераторы списков позволяют создать новый список на основе существующего, например, [x\*\*2 for x in my\_list if isinstance(x, int)] создаст список квадратов всех чисел из my\_list. Функция map() применяется для применения функции ко всем элементам списка, squared\_list = list(map(lambda x: x\*x, my\_list)).

1. **Словари в Python. Назначение, особенности, создание, изменение. Основные операции и методы. Способы итерации по словарям.**

Словари в Python – это структура данных, предназначенная для хранения пар "ключ-значение". Они удобны, когда нужно быстро находить значение по известному ключу, что делает их незаменимыми для многих задач в программировании.

Словари позволяют хранить элементы, каждый из которых состоит из уникального ключа и соответствующего ему значения. Ключи должны быть уникальными и неизменяемыми, такими как строки, числа или кортежи, а значения могут быть любыми объектами Python.

Создать словарь можно с помощью фигурных скобок или функции `dict()`. Например,

my\_dict = {'apple': 3, 'banana': 5, 'orange': 2}

или

my\_dict = dict(apple=3, banana=5, orange=2)

Для добавления или изменения элемента словаря достаточно присвоить значение по ключу:

my\_dict['apple'] = 4 # Изменение значения существующего ключа

my\_dict['pear'] = 7 # Добавление нового ключа и значения

Чтобы удалить элемент из словаря, можно использовать оператор `del` или метод `pop()`. Например,

del my\_dict['banana'] # Удаление ключа 'banana'

или

value = my\_dict.pop('orange') # Удаление ключа 'orange' и возврат его значения

Для работы со словарями Python предлагает множество методов. Например, метод `keys()` возвращает все ключи словаря, `values()` – все значения, а `items()` – все пары "ключ-значение":

Проверить наличие ключа в словаре можно с помощью оператора `in`:

if 'apple' in my\_dict:

print("Apple is in the dictionary")

Итерация по словарю может быть выполнена несколькими способами. Самый простой способ – это использование цикла `for` для перебора ключей:

for key in my\_dict:

print(key, my\_dict[key])

Для итерации по парам "ключ-значение" можно использовать метод `items()`:

for key, value in my\_dict.items():

print(key, value)

Также можно итерироваться только по значениям с помощью метода `values()`:

for value in my\_dict.values():

print(value)

Или по ключам с помощью метода `keys()`:

for key in my\_dict.keys():

print(key)

1. **Связные списки: однонаправленные и двунаправленные – принцип реализации. Сравнение скорости выполнения основных операций в связных списках и в динамическом массиве.**

Связные списки – это структура данных, в которой элементы (узлы) связаны между собой с помощью указателей. Каждый узел содержит данные и ссылку на следующий узел. Связные списки бывают однонаправленные и двунаправленные. В однонаправленном списке каждый узел содержит данные и указатель на следующий узел. Начало списка называется головой. Например, можно создать класс узла и класс однонаправленного списка. Узел содержит данные и указатель на следующий узел, а список имеет метод для добавления новых узлов. При добавлении нового узла, если список пустой, новый узел становится головой списка. Если в списке уже есть элементы, нужно пройти по списку до последнего узла и добавить новый узел после него.

В двунаправленном списке каждый узел содержит данные, указатель на следующий узел и указатель на предыдущий узел. Это позволяет легко перемещаться в обе стороны. Можно создать класс узла и класс двунаправленного списка. Узел будет содержать данные, указатель на следующий и предыдущий узлы, а список – метод для добавления новых узлов. При добавлении нового узла, если список пустой, новый узел становится головой списка. Если в списке уже есть элементы, нужно пройти по списку до последнего узла и добавить новый узел после него, установив указатель на предыдущий узел.

Рассмотрим скорость выполнения основных операций в связных списках и динамических массивах (списках в Python). В связных списках доступ к элементу занимает линейное время, так как нужно пройти по всем узлам от головы до нужного узла. В динамическом массиве доступ к элементу занимает постоянное время, так как элементы хранятся в непрерывной области памяти и доступны по индексу.

Добавление элемента в связных списках в начало занимает постоянное время, так как достаточно изменить указатель на голову. Добавление в конец занимает линейное время для однонаправленного списка (нужно пройти до конца) и постоянное время для двунаправленного списка (если есть указатель на хвост). В динамическом массиве добавление элемента в конец занимает амортизированное постоянное время, если не нужно увеличивать размер массива, иначе линейное время.

Удаление элемента из начала связного списка занимает постоянное время, так как достаточно изменить указатель на голову. Удаление из конца занимает линейное время для однонаправленного списка и постоянное время для двунаправленного списка. Удаление по произвольному индексу занимает линейное время, так как нужно найти элемент. В динамическом массиве удаление элемента занимает линейное время, так как все последующие элементы нужно сдвинуть на одну позицию влево.

Вставка элемента в связных списках после заданного узла занимает постоянное время, так как достаточно изменить указатели. Вставка по произвольному индексу занимает линейное время, так как нужно найти узел. В динамическом массиве вставка элемента занимает линейное время, так как все последующие элементы нужно сдвинуть.

Таким образом, связные списки обеспечивают более эффективные операции добавления и удаления элементов в начале или в конце списка (особенно двунаправленные). Однако доступ к элементам и вставка/удаление в середине связного списка значительно медленнее, чем в динамическом массиве. Выбор структуры данных зависит от конкретных требований задачи.

1. **Основные алгоритмы сортировки. Их сложность, достоинства и недостатки. Алгоритм быстрого поиска в отсортированном массиве. Сложность поиска в отсортированном и не отсортированном массиве.**

Основные алгоритмы сортировки включают несколько методов, таких как сортировка пузырьком, сортировка вставками, сортировка выбором, быстрая сортировка, сортировка слиянием и сортировка кучей. Каждый из них имеет свои преимущества и недостатки.

Сортировка пузырьком работает, сравнивая пары соседних элементов и меняя их местами, если они стоят не в том порядке. Этот процесс повторяется до тех пор, пока массив не будет отсортирован. Сортировка пузырьком проста в реализации, но неэффективна на больших наборах данных, потому что для выполнения всех операций требуется много времени, особенно если данных много.

Сортировка вставками берет элементы по одному и вставляет каждый из них на нужное место среди ранее отсортированных элементов. Этот метод эффективен для небольших массивов или почти отсортированных данных, но медленный для больших массивов, так как требует много шагов для завершения.

Сортировка выбором на каждом шаге выбирает наименьший элемент из неотсортированной части массива и меняет его местами с первым элементом этой части. Этот процесс повторяется до тех пор, пока весь массив не будет отсортирован. Этот метод прост, но неэффективен для больших наборов данных, поскольку тоже требует много времени.

Быстрая сортировка делит массив на две части, используя элемент, выбранный как опорный, так что все элементы, меньшие опорного, перемещаются влево, а большие — вправо. Затем сортировка рекурсивно применяется к каждой из этих частей. Быстрая сортировка обычно считается одной из самых быстрых, так как ее выполнение требует гораздо меньше шагов по сравнению с другими методами, особенно если данных много.

Сортировка слиянием делит массив пополам, рекурсивно сортирует каждую половину, а затем сливает отсортированные половины в один массив. Этот метод эффективен и стабилен, так как требует меньше шагов для выполнения по сравнению с простыми методами.

Сортировка кучей использует структуру данных, называемую кучей, для создания отсортированного массива. Она повторно извлекает максимальный элемент из кучи и помещает его в конец массива. Этот метод также выполняется довольно быстро и может быть эффективным на больших наборах данных.

Алгоритм быстрого поиска в отсортированном массиве, известный как бинарный поиск, требует очень мало шагов по сравнению с линейным поиском. Этот алгоритм работает путем деления массива пополам и сравнения искомого значения со средним элементом. Если значение меньше среднего элемента, поиск продолжается в левой половине массива, если больше — в правой. Этот процесс повторяется до тех пор, пока искомый элемент не будет найден или не останется подмассива для поиска.

Для поиска в отсортированном массиве бинарный поиск выполняется гораздо быстрее и требует намного меньше шагов, чем линейный поиск, который перебирает все элементы последовательно. В неотсортированном массиве линейный поиск является единственным вариантом и требует много шагов для поиска нужного элемента, что делает его менее эффективным по сравнению с бинарным поиском в отсортированном массиве.

1. **Основные технологии, использующиеся для разработки веб-приложений. Языки разметки и программирования, технологии, библиотеки и фреймворки.**

Веб-приложения разрабатываются с использованием различных технологий, которые можно разделить на несколько категорий: языки разметки, языки программирования, технологии, библиотеки и фреймворки.

Начнем с языков разметки, которые являются основой веб-страниц. HTML (HyperText Markup Language) используется для создания структуры веб-страницы, описывая такие элементы как заголовки, абзацы, ссылки и другие блоки контента. CSS (Cascading Style Sheets) отвечает за внешний вид веб-страницы, позволяя разработчикам задавать стили для элементов HTML, такие как цвет, шрифт и расположение на странице. HTML и CSS являются фундаментальными технологиями для создания статических веб-страниц.

Языки программирования используются для создания динамических и интерактивных веб-приложений. JavaScript является основным языком программирования для клиентской части, который позволяет добавлять интерактивность на веб-страницы. На серверной стороне популярны такие языки, как Python, PHP, Ruby, Java и JavaScript (с использованием Node.js). Эти языки позволяют обрабатывать запросы от клиентов, взаимодействовать с базами данных и выполнять бизнес-логику.

Существует множество технологий и инструментов, которые помогают разработчикам создавать веб-приложения. Например, AJAX (Asynchronous JavaScript and XML) позволяет загружать данные с сервера без перезагрузки страницы, что улучшает пользовательский опыт. REST (Representational State Transfer) и GraphQL используются для создания API, через которые клиенты могут взаимодействовать с сервером. JSON (JavaScript Object Notation) является популярным форматом для обмена данными между клиентом и сервером.

Библиотеки и фреймворки значительно упрощают разработку веб-приложений. JQuery — это популярная библиотека для JavaScript, которая облегчает работу с DOM (Document Object Model), обработку событий и создание анимаций. Angular, React и Vue.js — это популярные фреймворки и библиотеки для создания сложных пользовательских интерфейсов на клиентской стороне. Эти инструменты предоставляют мощные возможности для создания реактивных и интерактивных приложений. На серверной стороне также существует множество фреймворков. Django и Flask (на Python), Ruby on Rails (на Ruby) и Spring (на Java) — это лишь некоторые из них. Эти фреймворки предоставляют разработчикам инструменты для быстрого создания безопасных и масштабируемых веб-приложений, включая средства для маршрутизации запросов, работы с базами данных и управления пользователями.

1. **Общая характеристика языка разметки HTML. Основные элементы - теги, атрибуты, значения. Структура веб-страницы. Единицы измерения и цвета в HTML.**

HTML (HyperText Markup Language) является основным языком разметки для создания структуры веб-страниц. Он определяет содержание и структуру документа с помощью различных элементов, таких как теги, атрибуты и значения.

Теги — это основные строительные блоки HTML. Они обрамляют содержимое и указывают браузеру, как его отображать. Теги обычно имеют открывающий и закрывающий элементы, например <tag> и </tag>, где tag — это название тега. Некоторые теги могут быть одиночными, без закрывающего элемента, например <img> или <br>.

Атрибуты предоставляют дополнительную информацию о содержимом тега. Они указываются в открывающем теге и обычно имеют формат name="value". Например, атрибут href в теге <a> определяет адрес ссылки, а атрибут src в теге <img> определяет путь к изображению.

Значения используются в атрибутах для определения конкретных параметров. Например, значение blue может быть использовано для атрибута color в CSS для задания цвета текста.

Структура веб-страницы включает в себя различные элементы, такие как заголовок, тело, разделы, параграфы, изображения и многие другие. В общем, веб-страница обычно начинается с объявления типа документа (<!DOCTYPE html>), затем следует элемент <html>, внутри которого располагаются <head> и <body>. В <head> обычно содержится информация о документе, такая как заголовок (<title>), мета-теги, подключение стилей и скриптов. В <body> находится основное содержимое страницы.

В HTML можно использовать различные единицы измерения для определения размеров элементов, такие как пиксели (px), проценты (%), абсолютные (em, rem) и другие. Цвета могут быть определены с помощью их названий (например, red, blue) или используя RGB (rgb(255, 0, 0)) или HEX (#ff0000) значения. Кроме того, можно использовать атрибуты class и id для применения стилей из CSS к определенным элементам на странице.

1. **Общая характеристика языка CSS. Назначение, синтаксис, использование. Виды селекторов.**

CSS (Cascading Style Sheets) — это язык стилей, который используется для задания внешнего вида и форматирования элементов на веб-страницах. Он позволяет определить такие стили, как цвет текста, фоновое изображение, размер и шрифт текста, расположение элементов на странице и многое другое. Основная идея CSS заключается в том, что он отделяет структуру HTML-документа от его стилизации, что делает разработку и поддержку веб-страниц более эффективной и удобной.

Синтаксис CSS состоит из правил, которые включают в себя селекторы и объявления стилей. Селекторы указывают на элементы HTML, к которым будут применены стили, а объявления стилей определяют, какие конкретные стили будут применены к выбранным элементам.

Селекторы в CSS бывают разные: теговые, классовые, идентификаторы, селекторы атрибутов, комбинированные и псевдоклассы. Каждый из них позволяет выбрать определенные элементы или группы элементов на веб-странице для применения стилей.

Теговые селекторы являются наиболее базовым типом и применяются ко всем элементам заданного типа, таким как параграфы `<p>` или заголовки `<h1>`. Классовые селекторы используются для выбора элементов с определенным классом, который указывается в атрибуте `class`, например `.classname`. Идентификаторы представлены с помощью символа `#` и позволяют выбрать элементы с определенным идентификатором, указанным в атрибуте `id`, например `#idname`. Селекторы атрибутов используются для выбора элементов на основе их атрибутов, например `[attribute="value"]`. Комбинированные селекторы позволяют комбинировать различные типы селекторов для более точного выбора элементов, например `tag.class` или `tag#id`. Псевдоклассы и псевдоэлементы используются для стилизации элементов в определенных состояниях или частях элементов, например `:hover` для применения стилей при наведении курсора или `::before` для добавления контента перед содержимым элемента.

CSS можно использовать в веб-странице разными способами: встроенным или внешним. Встроенные стили определяются непосредственно внутри тега <style> в разделе <head> HTML-документа, тогда как внешние стили определяются в отдельном файле CSS и подключаются к HTML-документу с помощью тега <link>.

1. **Общая характеристика языка JavaScript. Синтаксис, назначение, система типов. Среда исполнения. Особенности выполнения на клиенте и на сервере.**

JavaScript — это мощный язык программирования, который широко используется для создания интерактивных веб-страниц. Он позволяет добавлять динамическое поведение к веб-страницам, взаимодействовать с пользователем, изменять содержимое страницы в реальном времени и многое другое.

Синтаксис JavaScript схож с синтаксисом других языков программирования, таких как C++ и Java. Он использует различные конструкции, такие как переменные, операторы, условные операторы, циклы, функции и объекты. JavaScript чувствителен к регистру, что означает, что имена переменных и функций должны быть написаны точно так же, как их определение.

JavaScript применяется для создания интерактивных элементов на веб-страницах, таких как динамические формы, анимации, слайдеры, валидация форм, обработка событий (например, клики, наведения, отправка форм) и многое другое. Он также используется в разработке веб-приложений и игр.

JavaScript имеет динамическую систему типов, что означает, что переменные не обязательно должны быть объявлены с определенным типом данных, и их тип может изменяться в процессе выполнения программы. Он поддерживает различные типы данных, включая числа, строки, логические значения, объекты, массивы и функции.

JavaScript исполняется в среде браузера, что означает, что он выполняется на компьютере пользователя в его веб-браузере. Большинство современных браузеров, таких как Google Chrome, Mozilla Firefox, Safari и Microsoft Edge, включают встроенный движок JavaScript для интерпретации и выполнения кода JavaScript. Кроме того, JavaScript также может быть исполнен на сервере с использованием среды выполнения, такой как Node.js.

На клиентской стороне JavaScript выполняется в окружении браузера и используется для взаимодействия с пользователем и манипуляции содержимым веб-страницы. На серверной стороне JavaScript может выполняться с использованием среды выполнения Node.js и используется для создания веб-серверов, обработки запросов от клиентов, доступа к базам данных и других серверных операций.

1. **Серверное программирование. Особенности, обзор технологий и языков программирования. Распространенные серверные фреймворки.**

Серверное программирование — это процесс создания программного обеспечения, которое выполняется на сервере и обрабатывает запросы от клиентских приложений. Основная задача серверного программирования — это обеспечение доступа к данным, выполнение бизнес-логики, аутентификация и авторизация пользователей, а также обеспечение безопасности и масштабируемости приложений.

Для серверного программирования используются различные языки программирования и технологии, включая Python, Ruby, Java, JavaScript (с использованием Node.js), PHP и другие. Каждый из этих языков имеет свои особенности и преимущества, и выбор зависит от требований проекта и предпочтений разработчика.

Например, Python является популярным выбором для серверного программирования благодаря своей простоте, ясности синтаксиса и богатой экосистеме библиотек. Django и Flask - это два распространенных фреймворка для серверного программирования на Python, которые обеспечивают готовую инфраструктуру для создания веб-приложений.

Ruby также используется для серверного программирования, особенно с использованием фреймворка Ruby on Rails, который предлагает простоту и конвенции для быстрой разработки веб-приложений.

Java широко применяется в корпоративном окружении для создания масштабируемых и надежных серверных приложений. Spring Framework является одним из наиболее популярных фреймворков для серверного программирования на Java, предоставляя обширный набор инструментов для разработки веб-приложений.

JavaScript на сервере стал все более популярным благодаря Node.js, среде выполнения, которая позволяет исполнять JavaScript на сервере. С помощью Node.js разработчики могут создавать высокопроизводительные и масштабируемые веб-приложения, используя тот же язык программирования как на клиенте, что обеспечивает единый стек технологий и повышает эффективность разработки.

1. **Технология взаимодействия клиентской и серверных частей веб-приложения. Протокол HTTP. Запросы и ответы.**

Технология взаимодействия клиентской и серверной частей веб-приложения основана на протоколе HTTP (Hypertext Transfer Protocol). HTTP является протоколом передачи данных в Интернете и обеспечивает связь между клиентскими приложениями (браузерами) и серверами.

Процесс взаимодействия начинается с отправки HTTP-запроса от клиентского приложения к серверу. Этот запрос содержит информацию о том, что клиент хочет получить или сделать на сервере. Запрос может содержать различные параметры, заголовки и данные.

После получения запроса сервер обрабатывает его и отправляет HTTP-ответ обратно клиенту. Ответ содержит статус операции (например, успешно или неуспешно), заголовки ответа и данные, которые сервер возвращает клиенту. Эти данные могут быть в виде HTML-страницы, JSON-объекта, изображения и т.д., в зависимости от типа запроса и предназначения сервера.

HTTP-запросы и ответы могут быть различными по типу и содержанию. Например, GET-запрос используется для получения данных с сервера, POST-запрос - для отправки данных на сервер для обработки, а PUT-запрос - для обновления данных на сервере. Кроме того, ответы сервера могут содержать различные статусы, такие как 200 (OK), 404 (Not Found), 500 (Internal Server Error) и др., в зависимости от успешности выполнения запроса.

Таким образом, HTTP-протокол является основой для взаимодействия между клиентскими и серверными частями веб-приложений, обеспечивая передачу данных и управление операциями между ними. Он обеспечивает стандартизацию и надежность обмена информацией в Интернете, что позволяет создавать мощные и эффективные веб-приложения.

1. **Адресация в интернете. Основные понятия о URI, URL, URN. Структура адресов, использование.**

Адресация в интернете является важным аспектом взаимодействия между клиентскими и серверными приложениями. Основные понятия в адресации включают в себя URI (Uniform Resource Identifier), URL (Uniform Resource Locator) и URN (Uniform Resource Name).

URI (Uniform Resource Identifier) - это строка символов для идентификации ресурса в сети. URI является общим термином, который включает в себя два более конкретных понятия - URL и URN.

URL (Uniform Resource Locator) - это конкретный тип URI, который определяет местоположение ресурса в сети и способ доступа к нему. URL включает схему доступа (например, http, https, ftp), имя хоста (например, www.example.com), порт (необязательно), путь к ресурсу и дополнительные параметры (например, запросы GET).

URN (Uniform Resource Name) - это еще один тип URI, который представляет собой уникальное имя ресурса без указания его местоположения или способа доступа. URN обычно используется для идентификации ресурсов, независимо от их расположения в сети.

Структура адресов в интернете выглядит следующим образом:

схема://хост:порт/путь?запросы#фрагмент

Схема определяет протокол доступа к ресурсу (например, http, https, ftp). Хост указывает на имя сервера, на котором находится ресурс (например, [www.example.com](http://www.example.com)). Порт определяет номер порта, на котором работает сервер (обычно опускается для стандартных портов). Путь представляет собой путь к ресурсу на сервере. Запросы содержат дополнительные параметры запроса (например, данные формы в случае HTTP GET-запросов). Фрагмент указывает на конкретное место в ресурсе (например, якорь на веб-странице).

Использование адресации в интернете позволяет однозначно идентифицировать ресурсы и обеспечивает их доступность для клиентских приложений через стандартные протоколы и схемы. Она играет ключевую роль в веб-разработке, обеспечивая надежное взаимодействие между пользователями и ресурсами в сети.

1. **Эволюция методологий моделирования. Моделирование деятельности и моделирование процессов. Использование методологии ARIS для моделирования деятельности предприятия. Модели, атрибуты моделей, действия над моделями, типы моделей. Объекты, свойства объектов. Связи, свойства связей. Техническая реализация методологии моделирования.**

Эволюция методологий моделирования прошла долгий путь от простых диаграмм до сложных системных подходов, которые используются для моделирования и оптимизации бизнес-процессов. На протяжении времени появились различные методологии, которые помогают визуализировать, анализировать и улучшать деятельность и процессы предприятий. Эволюция методологий моделирования можно разделить на несколько ключевых этапов:

* Ранние методологии (до 1960-х): Графические методы (блок-схемы) и математические модели.
* Системный подход (1960-е - 1980-е): Системный анализ, динамическое моделирование, имитационные модели.
* Объектно- и процессно-ориентированные подходы (1980-е - 2000-е): UML, BPMN, CASE-технологии.
* Современные методологии (2000-е - настоящее время): Агентно-ориентированное моделирование, анализ больших данных, цифровые двойники, инструменты низкого кода.

Моделирование деятельности фокусируется на описании различных аспектов работы организации, включая ее структуру, функции и взаимодействия. Целью моделирования деятельности является понимание и улучшение работы всей организации. Моделирование процессов, с другой стороны, концентрируется на конкретных процессах внутри организации, описывая последовательность шагов и взаимодействие между участниками процесса. Основная цель моделирования процессов - оптимизация и повышение эффективности этих процессов.

Методология ARIS (Architecture of Integrated Information Systems) является одной из самых популярных и мощных методологий для моделирования деятельности предприятия. ARIS предоставляет набор инструментов и методов для создания моделей, которые помогают анализировать, оптимизировать и внедрять бизнес-процессы. Основные компоненты ARIS включают модели, атрибуты моделей, действия над моделями и типы моделей.

Модели в ARIS представляют собой абстрактные представления реальных бизнес-процессов и систем. Они могут включать различные аспекты, такие как организационная структура, процессы, данные и информационные системы. Атрибуты моделей содержат дополнительную информацию о моделях, такую как название, описание, владельцы и другие важные характеристики.

Действия над моделями включают создание, редактирование, анализ и оптимизацию моделей. С помощью этих действий можно уточнять и улучшать бизнес-процессы, выявлять узкие места и находить возможности для улучшений. Типы моделей в ARIS могут быть различными, включая функциональные модели, организационные модели, модели данных и модели процессов.

Объекты в ARIS представляют собой основные элементы модели, такие как процессы, функции, данные и роли. Свойства объектов содержат информацию о характеристиках этих элементов, таких как название, тип, владельцы и другие атрибуты. Связи в ARIS представляют собой отношения между объектами, такие как последовательность шагов в процессе или взаимодействие между различными функциями. Свойства связей включают информацию о типе связи, направлении и других важных характеристиках.

Техническая реализация методологии моделирования предполагает использование специализированных программных инструментов и платформ, которые поддерживают создание и управление моделями. Эти инструменты предоставляют удобные интерфейсы для создания диаграмм, анализа данных и оптимизации процессов. Примеры таких инструментов включают ARIS Toolset, Bizagi, IBM Blueworks Live и другие.

1. **Разделение прав доступа и система безопасности современных многопользовательских операционных систем на примере Linux. Пользователи и группы, основные права. Основные команды для управления правами и пользователями.**

Разделение прав доступа и система безопасности в современных многопользовательских операционных системах, таких как Linux, являются ключевыми аспектами, обеспечивающими защиту данных и стабильность системы. В Linux эта система основывается на концепции пользователей и групп, а также на системе прав доступа, которая контролирует, кто и что может делать с файлами и директориями.

В Linux каждый пользователь имеет уникальный идентификатор (UID), а каждая группа - групповой идентификатор (GID). Пользователи могут быть объединены в группы для упрощения управления правами доступа. Каждый файл и директория в системе принадлежат определенному пользователю и группе.

Права доступа в Linux делятся на три категории - для владельца файла, для группы и для всех остальных пользователей. Каждая из этих категорий может иметь три типа прав: чтение (r), запись (w) и выполнение (x). Эти права определяют, кто может читать, изменять или выполнять файл или заходить в директорию.

В Linux существует несколько команд для управления пользователями, группами и правами доступа.

* chmod (change mode) используется для изменения прав доступа к файлам и директориям. Например, команда chmod 755 filename установит права доступа на чтение, запись и выполнение для владельца, и права на чтение и выполнение для группы и остальных пользователей.
* chown (change owner) используется для изменения владельца файла или директории. Например, chown user:group filename изменит владельца файла на user и группу на group.
* usermod позволяет изменять параметры существующих пользователей, такие как добавление их в группы. Например, usermod -aG groupname username добавит пользователя username в группу groupname.
* useradd и userdel используются для создания и удаления пользователей соответственно.
* groupadd и groupdel предназначены для создания и удаления групп.

Эти команды помогают администраторам управлять правами доступа и обеспечивать безопасность системы. Важным аспектом является регулярный аудит прав доступа и контроль за тем, чтобы пользователи имели только необходимые им права, что минимизирует риск несанкционированного доступа к данным.

1. **Работа в командной строке bash. Основные понятия и структура команды. Основные команды для перемещения по каталогам, просмотра файлов и информации о содержимом файлов.**

Работа в командной строке bash является основой для эффективного взаимодействия с операционной системой Linux. Bash (Bourne Again Shell) - это один из самых популярных интерпретаторов командной строки в Linux, предоставляющий пользователям мощный и гибкий интерфейс для выполнения различных задач.

Команда в bash состоит из нескольких частей: имени команды, опций и аргументов. Имя команды указывает на выполняемую программу или скрипт. Опции (или флаги) изменяют поведение команды и обычно начинаются с одного или двух дефисов (например, -a или --all). Аргументы предоставляют команде необходимые данные или указывают на файлы и каталоги, с которыми она должна работать. Пример команды: ls -l /home/user, где ls - имя команды, -l - опция, а /home/user - аргумент.

Основные команды:

-Перемещение по каталогам:

* cd (change directory): Команда для смены текущего рабочего каталога.
* cd ..: Перемещает на один уровень вверх по каталогу.
* cd ~: Перемещает в домашний каталог текущего пользователя.
* cd -: Перемещает в предыдущий рабочий каталог.

-Просмотр файлов и каталогов:

* ls (list): Показывает список файлов и каталогов в текущем рабочем каталоге.
* ls -l: Показывает подробный список файлов и каталогов с их правами доступа, владельцами, размерами и датами изменения.
* ls -a: Показывает все файлы, включая скрытые файлы.

-Информация о содержимом файлов:

* cat (concatenate): Показывает содержимое файла.
* less и more: Команды для постраничного просмотра длинных файлов. Например, less filename открывает файл filename для просмотра с возможностью прокрутки.
* head: Показывает первые несколько строк файла.
* tail: Показывает последние несколько строк файла.
* grep: Ищет строки, соответствующие заданному шаблону. Например, grep "pattern" filename ищет строки, содержащие pattern в файле.

Эти команды и концепции являются основными для работы в командной строке bash и позволяют эффективно управлять файлами, каталогами и другими ресурсами системы.

1. **Настройка и диагностика сетевых подключений в современных операционных системах на примере Linux. Удаленный доступ к операционной системе по протоколу SSH.**

Настройка и диагностика сетевых подключений в Linux - важные аспекты, обеспечивающие корректное функционирование сети. В Linux настройка сетевых подключений может осуществляться с помощью командной строки, графических интерфейсов и текстовых конфигурационных файлов. Основной инструмент для управления сетевыми интерфейсами - команда ip. Например, чтобы настроить IP-адрес на интерфейсе eth0, используется команда sudo ip addr add 192.168.1.100/24 dev eth0. Для включения интерфейса применяется sudo ip link set eth0 up, а для настройки маршрута по умолчанию - sudo ip route add default via 192.168.1.1.

Для диагностики сетевых подключений используются различные команды. Команда ping проверяет доступность удаленного хоста, traceroute определяет маршрут до удаленного хоста, netstat и ss отображают состояние сетевых соединений и прослушивающих портов, а ifconfig и ip a показывают конфигурацию сетевых интерфейсов.

Удаленный доступ к операционной системе по протоколу SSH обеспечивает защищенное соединение. SSH используется для безопасного управления удаленными серверами. Для подключения по SSH используется команда ssh username@remote\_host, где username — это имя пользователя, а remote\_host - IP-адрес или доменное имя удаленного сервера. Конфигурация SSH-сервера осуществляется через файл /etc/ssh/sshd\_config. Например, для изменения порта SSH редактируется строка Port 2222, после чего сервер перезапускается командой sudo systemctl restart sshd. Для повышения безопасности можно использовать аутентификацию по ключам. Ключи создаются командой ssh-keygen, и публичный ключ добавляется на удаленный сервер в файл ~/.ssh/authorized\_keys.

1. **Основные сетевые службы. Веб-технологии - необходимое программное обеспечение, назначение схема работы, основные протоколы.**

Основные сетевые службы и веб-технологии играют ключевую роль в работе современных сетей и интернета. Они обеспечивают функциональность, взаимодействие и обмен данными между различными устройствами и приложениями.

Сетевые службы представляют собой программное обеспечение, которое предоставляет определенные функции и услуги в сети. К основным сетевым службам относятся DNS, DHCP, HTTP, FTP, SMTP и другие. Эти службы обеспечивают перевод доменных имен в IP-адреса, автоматическое назначение IP-адресов устройствам, передачу файлов и электронной почты, а также поддержку веб-сайтов.

Веб-технологии включают в себя множество компонентов, которые обеспечивают создание, развертывание и взаимодействие веб-приложений. Основное программное обеспечение для веб-технологий включает веб-серверы, базы данных и серверные скрипты. Веб-серверы, такие как Apache и Nginx, обрабатывают запросы от клиентов и передают им запрашиваемые веб-страницы. Базы данных, такие как MySQL и PostgreSQL, хранят данные, которые используются веб-приложениями. Серверные скрипты, написанные на языках программирования, таких как PHP, Python и JavaScript (Node.js), обрабатывают запросы пользователей и взаимодействуют с базами данных для получения и изменения данных.

Схема работы веб-технологий обычно включает взаимодействие клиента (браузера) и сервера через интернет. Клиент отправляет запрос на веб-сервер с использованием протокола HTTP или HTTPS. Веб-сервер обрабатывает запрос, взаимодействует с базой данных при необходимости, и возвращает клиенту ответ в виде HTML-страницы, JSON-данных или другого формата. Клиентский браузер отображает полученную информацию пользователю.

Основные протоколы, используемые в веб-технологиях, включают:

* HTTP (HyperText Transfer Protocol): Основной протокол для передачи данных в вебе. Он определяет правила обмена сообщениями между клиентом и сервером.
* HTTPS (HTTP Secure): Расширение HTTP с поддержкой шифрования через SSL/TLS, обеспечивающее безопасную передачу данных.
* FTP (File Transfer Protocol): Протокол для передачи файлов между клиентом и сервером. Используется для загрузки и скачивания файлов с удаленных серверов.
* DNS (Domain Name System): Служба, которая переводит доменные имена в IP-адреса, позволяя пользователям использовать удобные имена для доступа к веб-сайтам вместо числовых IP-адресов.

Эти протоколы и технологии работают вместе, обеспечивая функциональность современных веб-приложений и услуг.

1. **Общая характеристика языка программирования Java. Схема выполнения, синтаксис, система типов.**

Java — это высокоуровневый, объектно-ориентированный язык программирования, разработанный компанией Sun Microsystems в 1995 году, которая теперь является частью Oracle. Одной из ключевых особенностей Java является его платформонезависимость, что означает, что программы на Java можно писать один раз и запускать на любой платформе, поддерживающей Java Virtual Machine (JVM). Это достигается благодаря концепции "Write Once, Run Anywhere" (WORA).

Процесс выполнения программ на Java начинается с компиляции исходного кода в байт-код с помощью компилятора javac. Этот байт-код затем выполняется на JVM, которая интерпретирует его или компилирует в машинный код конкретного устройства с помощью JIT (Just-In-Time) компилятора. Это позволяет запускать Java-программы на любой платформе, где установлена JVM, обеспечивая их платформонезависимость.

Синтаксис Java схож с синтаксисом C и C++, что делает его знакомым и понятным для разработчиков, переходящих с этих языков. Программы на Java состоят из классов и методов. Главный метод, с которого начинается выполнение программы, имеет следующую сигнатуру: public static void main(String[] args). Основные элементы синтаксиса включают объявление переменных, управляющие конструкции, такие как if, for, while, классы и объекты, а также наследование и интерфейсы.

Система типов в Java строго статическая, что означает, что все типы переменных и выражений проверяются на этапе компиляции. Это обеспечивает высокую степень безопасности кода. В Java есть примитивные типы, такие как int, float, boolean, и ссылочные типы, такие как объекты и массивы. Java также управляет памятью автоматически с помощью сборщика мусора (Garbage Collector), что освобождает разработчиков от необходимости вручную управлять памятью.

В сравнении с другими языками программирования, Java выделяется своей платформонезависимостью, широкой стандартной библиотекой и сильной системой типов. Эти особенности делают его популярным для разработки корпоративных приложений, мобильных приложений для Android и других программных решений.

1. **Объектно-ориентированное программирование в Java. Принципы, особенности реализации. Синтаксис.**

Объектно-ориентированное программирование (ООП) в Java - это фундаментальный подход к разработке программ, который фокусируется на использовании объектов и классов. Основные принципы ООП включают инкапсуляцию, наследование, полиморфизм и абстракцию.

Инкапсуляция означает, что внутренние детали работы класса скрыты, а доступ к ним осуществляется только через публичные методы. В Java это реализуется с помощью модификаторов доступа, таких как `private`, `protected` и `public`. Например, поля класса обычно объявляются приватными, а доступ к ним осуществляется через публичные методы-геттеры и сеттеры. Это позволяет защитить данные и управлять доступом к ним.

Наследование позволяет создавать новый класс на основе существующего, унаследовав его поля и методы. В Java наследование реализуется с помощью ключевого слова `extends`. Это позволяет использовать повторно уже написанный код и расширять функциональность классов. Например, если у нас есть класс `Animal` с методом `eat`, мы можем создать класс `Dog`, который наследует `Animal` и добавляет новые методы, такие как `bark`.

Полиморфизм позволяет объектам разных классов обрабатывать данные через один и тот же интерфейс. Это достигается через переопределение методов и использование интерфейсов. Например, у нас может быть метод `makeSound`, который реализован по-разному в классах `Dog` и `Cat`. В результате мы можем вызвать этот метод на любом объекте типа `Animal`, и он будет выполнять соответствующее действие, в зависимости от того, какой конкретный объект используется.

Абстракция — это процесс выделения общих характеристик объектов с целью создания классов. В Java это реализуется с помощью абстрактных классов и интерфейсов. Абстрактные классы могут содержать как реализацию методов, так и абстрактные методы, которые должны быть реализованы в подклассах. Интерфейсы объявляют методы, которые должны быть реализованы классами, реализующими этот интерфейс.

Синтаксис ООП в Java включает классы, объекты, наследование, интерфейсы и абстрактные классы. Класс объявляется с помощью ключевого слова `class`, а объекты создаются с помощью оператора `new`. Наследование осуществляется с помощью ключевого слова `extends`, а интерфейсы реализуются с помощью ключевого слова `implements`.

1. **Фреймворк Spring. Назначение, основные понятия, применение. Объектно-реляционное отображение (ORM).**

Фреймворк Spring — это мощная платформа для разработки корпоративных приложений на языке Java. Его основное назначение - упростить создание масштабируемых, надежных и легко поддерживаемых приложений, предоставляя разработчикам обширный набор инструментов и библиотек.

Одним из ключевых понятий в Spring является инверсия управления (IoC) и внедрение зависимостей (DI). Вместо того, чтобы объекты сами создавали свои зависимости, контейнер Spring управляет жизненным циклом объектов и их зависимостями. Это позволяет сделать код более модульным, легко тестируемым и изменяемым. Например, в Spring можно аннотировать классы и их зависимости с помощью аннотаций `@Component` и `@Autowired`, что упростит создание и управление объектами.

Еще одно важное понятие в Spring - аспектно-ориентированное программирование (AOP). Это подход, который позволяет разделить кросс-срезные заботы, такие как логирование, транзакционное управление и безопасность, от бизнес-логики приложения. В Spring это реализуется с помощью аннотаций, таких как `@Aspect` и `@Before`, которые позволяют определять аспекты и их поведение.

Spring также предлагает мощный модуль Spring MVC для разработки веб-приложений. Этот модуль следит за моделью Model-View-Controller (MVC), что упрощает разделение представления, бизнес-логики и данных. Контроллеры обрабатывают HTTP-запросы, взаимодействуют с бизнес-логикой и возвращают модели данных для отображения во вьюхах. Аннотации, такие как `@Controller` и `@RequestMapping`, используются для определения контроллеров и их методов.

Еще одна важная часть Spring — это Spring Boot, который значительно упрощает настройку и развертывание приложений. Spring Boot предоставляет готовые настройки по умолчанию и встроенный сервер приложений, что позволяет быстро создавать работающие приложения с минимальными усилиями по настройке. Например, с помощью Spring Boot можно создать полноценное веб-приложение всего несколькими аннотациями и файлами конфигурации.

Объектно-реляционное отображение (ORM) — это техника, которая позволяет автоматически преобразовывать данные между объектами в Java и записями в реляционных базах данных. В контексте Spring, ORM часто реализуется с использованием Hibernate - популярного фреймворка для ORM. Spring предоставляет интеграцию с Hibernate через Spring Data JPA, что упрощает работу с базами данных. С помощью аннотаций, таких как `@Entity` и `@Table`, можно определить, как классы Java соответствуют таблицам в базе данных, а аннотации, такие как `@Id` и `@GeneratedValue`, используются для указания первичных ключей и их генерации.

Применение Spring разнообразно. Он используется для создания веб-приложений, RESTful сервисов, микросервисов и корпоративных приложений. Благодаря модульной архитектуре Spring позволяет разработчикам выбирать только те компоненты, которые им действительно нужны, и легко интегрировать их в свои проекты.

1. **Понятие, классификация и принципы построения банковских информационных систем (БИС).**

Банковские информационные системы, или БИС, представляют собой сложные программно-аппаратные комплексы, которые автоматизируют и управляют различными аспектами банковской деятельности. Основная цель БИС — это обеспечение эффективного и безопасного выполнения банковских операций, управление финансовыми потоками, учет клиентских данных и предоставление различных финансовых услуг.

БИС включают в себя программное обеспечение, аппаратное обеспечение и средства связи, которые вместе обеспечивают сбор, обработку, хранение и передачу данных, связанных с банковскими операциями. Эти системы используются как сотрудниками банков, так и клиентами, а также внешними пользователями, такими как регуляторные органы и партнеры.

Классификация БИС может осуществляться по разным критериям. Один из ключевых критериев - функциональное назначение. В зависимости от этого БИС можно разделить на несколько категорий. Операционные системы обеспечивают проведение банковских операций, таких как платежи, депозиты, кредиты и валютные операции. Управленческие системы поддерживают управленческую деятельность банка, включая планирование, контроль, анализ и отчетность. Информационные системы для клиентов предоставляют клиентам доступ к банковским услугам через интернет-банкинг, мобильные приложения и банкоматы. Системы риск-менеджмента предназначены для оценки и управления рисками, связанными с банковской деятельностью, а системы поддержки принятия решений помогают в принятии стратегических и оперативных решений на основе анализа данных.

Архитектура БИС также может быть различной. Одноуровневые системы представляют собой простые решения, где все компоненты работают на одном уровне. Двухуровневые системы разделяют функциональность между клиентской и серверной частями. Многоуровневые системы включают несколько уровней, таких как уровень данных, уровень бизнес-логики и уровень представления.

Принципы построения БИС включают модульность, интегрируемость, безопасность, надежность и отказоустойчивость, производительность и масштабируемость, а также удобство использования. Модульность означает, что система строится из отдельных модулей, каждый из которых отвечает за свою часть функциональности, что позволяет легко модернизировать и расширять систему. Интегрируемость подразумевает возможность взаимодействия с другими системами и приложениями через стандартизированные интерфейсы и протоколы. Безопасность заключается в защите данных и операций от несанкционированного доступа, мошенничества и других угроз, используя шифрование, аутентификацию и авторизацию. Надежность и отказоустойчивость обеспечивают стабильную работу системы и быстрое восстановление после сбоев через резервирование и дублирование компонентов. Производительность и масштабируемость позволяют системе обеспечивать высокую скорость обработки данных и возможность увеличения мощности по мере роста объемов операций и количества пользователей. Удобство использования означает, что интерфейсы системы должны быть интуитивно понятными и удобными для пользователей, что повышает эффективность работы и снижает вероятность ошибок.

1. **Реализация решения задач кредитного модуля банковских информационных систем.**

Реализация решения задач кредитного модуля в банковских информационных системах (БИС) - это процесс, включающий автоматизацию и оптимизацию всех этапов кредитного процесса, начиная от подачи заявки до полного погашения кредита. Кредитный модуль является одним из ключевых компонентов БИС и позволяет банкам эффективно управлять кредитными продуктами, оценивать риски и обслуживать клиентов.

Первый этап - это подача заявки на кредит. В современном кредитном модуле этот процесс полностью автоматизирован. Клиенты могут подавать заявки через интернет-банкинг, мобильные приложения или непосредственно в отделении банка. После подачи заявки система автоматически проверяет данные клиента, используя интеграцию с внешними базами данных, такими как кредитные бюро и государственные реестры, чтобы собрать необходимую информацию о кредитной истории и финансовом положении заявителя.

Следующий этап - оценка кредитоспособности клиента. Для этого используется система скоринга, которая на основе множества факторов, таких как кредитная история, доходы, возраст, семейное положение и другие параметры, рассчитывает кредитный рейтинг клиента. Алгоритмы скоринга могут быть реализованы с использованием методов машинного обучения и анализа больших данных, что позволяет повышать точность и объективность оценки. Если клиент проходит скоринговую оценку, его заявка передается на рассмотрение кредитного комитета или автоматически одобряется системой в зависимости от политики банка.

После одобрения заявки начинается процесс заключения кредитного договора и выдачи средств. Кредитный модуль автоматически генерирует кредитный договор, рассчитывает график погашения и отправляет клиенту на подпись через электронные каналы связи или в бумажном виде. В случае электронного документооборота используется электронная подпись для подтверждения договора. После подписания договора средства автоматически перечисляются на счет клиента или выдаются наличными.

Далее начинается этап сопровождения кредита. Кредитный модуль отслеживает выполнение условий кредитного договора, автоматически рассчитывает ежемесячные платежи и отправляет напоминания клиентам о предстоящих платежах. В случае просрочки платежей система автоматически начисляет штрафы и пени, а также может инициировать процесс реструктуризации долга или передачи дела в коллекторское агентство.

Кроме того, кредитный модуль предоставляет аналитические инструменты для мониторинга и управления кредитным портфелем банка. Система генерирует отчеты по различным параметрам, таким как объемы выданных кредитов, уровень просроченной задолженности, распределение кредитов по видам и срокам. Эти данные позволяют банковским аналитикам и менеджерам принимать обоснованные решения по управлению рисками и оптимизации кредитной политики.

Для реализации кредитного модуля используются различные технологии и подходы. Например, современные банковские системы часто строятся на основе микросервисной архитектуры, что позволяет разрабатывать и внедрять отдельные компоненты модуля независимо друг от друга. Использование облачных технологий позволяет масштабировать систему в зависимости от нагрузки и обеспечивать высокую доступность и отказоустойчивость.

Интеграция с внешними сервисами, такими как системы проверки кредитных историй, государственные реестры, платежные системы и другие, осуществляется через API (интерфейсы прикладного программирования). Это позволяет автоматизировать обмен данными и сократить время обработки заявок.

Важным аспектом является безопасность данных. Кредитный модуль должен обеспечивать защиту конфиденциальной информации клиентов и соблюдение требований законодательства в области защиты персональных данных. Для этого используются шифрование данных, многофакторная аутентификация, контроль доступа и регулярные аудиты безопасности.

1. **Применение технологии Big Data в банковской сфере.**

Технология Big Data, или большие данные, представляет собой метод обработки и анализа огромных объемов данных, которые не поддаются обработке традиционными методами. В банковской сфере применение Big Data открывает новые возможности для повышения эффективности, улучшения качества обслуживания клиентов и принятия обоснованных решений.

Одним из основных применений Big Data в банковском секторе является управление рисками и кредитоспособностью. Анализ больших объемов данных о транзакциях, поведении клиентов и внешних факторах позволяет более точно оценивать кредитоспособность клиентов. Традиционные методы оценки кредитных рисков могут быть дополнены данными из социальных сетей, онлайн-покупок и других источников, что позволяет формировать более полное представление о финансовом состоянии заемщика и снижать уровень дефолтов.

Персонализация услуг и продуктов - еще одна важная область применения Big Data в банках. Анализ данных о предпочтениях и поведении клиентов позволяет создавать персонализированные предложения, что увеличивает удовлетворенность клиентов и способствует росту продаж. Например, на основе анализа покупок и транзакций можно предлагать клиентам выгодные кредитные продукты или инвестиционные возможности, которые соответствуют их потребностям и финансовым целям.

Big Data также играют ключевую роль в борьбе с мошенничеством. Анализ больших объемов транзакционных данных в режиме реального времени позволяет выявлять подозрительные активности и предотвращать мошенничество. Например, если система обнаруживает аномальную активность, такую как одновременные попытки снятия денег в разных странах, она может автоматически заблокировать подозрительные транзакции и уведомить службу безопасности банка. Машинное обучение и аналитика больших данных позволяют постоянно совершенствовать алгоритмы обнаружения мошенничества, делая их более точными и эффективными.

В области маркетинга и продаж Big Data помогают банкам более точно сегментировать клиентов и разрабатывать целевые маркетинговые кампании. Анализ данных о поведении клиентов, их предпочтениях и откликах на предыдущие маркетинговые акции позволяет создавать более эффективные маркетинговые стратегии и повышать конверсию. Это также способствует более эффективному использованию маркетингового бюджета.

Операционная эффективность банков также значительно улучшается с использованием технологий Big Data. Анализ внутренних данных банка позволяет оптимизировать бизнес-процессы, улучшать управление ресурсами и снижать операционные затраты. Например, анализ данных о загрузке сотрудников и времени выполнения различных операций позволяет выявлять узкие места и внедрять меры по их устранению.

Наконец, Big Data играют важную роль в стратегическом планировании и принятии управленческих решений. Аналитика больших данных предоставляет банкам возможность проводить глубокий анализ рынка, оценивать тенденции и прогнозировать будущее развитие. Это позволяет принимать обоснованные стратегические решения, такие как запуск новых продуктов, выход на новые рынки или оптимизация текущей деятельности.

1. **Системы дистанционного банковского обслуживания.**

Системы дистанционного банковского обслуживания (ДБО) представляют собой набор технологий и сервисов, которые позволяют клиентам банка осуществлять различные финансовые операции удаленно, без необходимости посещения банковских отделений. ДБО включает в себя такие услуги, как интернет-банкинг, мобильный банкинг, телефонное банкинг и обслуживание через банкоматы.

Одним из основных компонентов ДБО является интернет-банкинг. Это веб-приложение, доступное через интернет, которое позволяет клиентам управлять своими банковскими счетами, оплачивать счета, переводить деньги, открывать депозиты, погашать кредиты и выполнять другие финансовые операции. Интернет-банкинг предоставляет круглосуточный доступ к банковским услугам и значительно экономит время клиентов.

Мобильный банкинг - еще один важный элемент ДБО, который стал особенно популярным с развитием смартфонов и мобильных приложений. Мобильные банковские приложения позволяют клиентам выполнять те же операции, что и интернет-банкинг, но с мобильного устройства. Это особенно удобно для людей, ведущих активный образ жизни и нуждающихся в быстром доступе к своим финансам. Мобильный банкинг также может включать дополнительные функции, такие как сканирование чеков для депозита и использование биометрических данных для аутентификации.

Телефонное банкинг позволяет клиентам выполнять банковские операции по телефону, используя автоматизированные системы или общаясь с оператором контактного центра. Этот способ особенно полезен для клиентов, не имеющих доступа к интернету или мобильным устройствам. Телефонное банкинг может использоваться для проверки баланса счета, оплаты счетов, перевода денег и других операций.

Банкоматы и киоски самообслуживания также являются частью системы ДБО. Они позволяют клиентам снимать наличные, вносить деньги на счет, оплачивать счета и выполнять другие операции без необходимости взаимодействия с банковским персоналом. Современные банкоматы могут предоставлять расширенные функции, такие как обмен валюты и выпуск банковских карт.

ДБО предоставляет множество преимуществ как для клиентов, так и для банков. Клиенты получают доступ к банковским услугам 24/7 из любой точки мира, что значительно повышает удобство использования банковских продуктов. Это особенно важно для занятых людей и тех, кто живет в удаленных районах. Для банков ДБО позволяет снизить затраты на обслуживание клиентов, уменьшить нагрузку на отделения и увеличить охват клиентов.

Однако системы ДБО также сталкиваются с рядом вызовов и рисков. Одним из основных рисков является безопасность. Поскольку ДБО предполагает удаленный доступ к финансовым данным и операциям, важно обеспечить защиту от киберугроз, таких как фишинг, взлом аккаунтов и мошенничество. Для этого используются различные методы, включая шифрование данных, двухфакторную аутентификацию, биометрические методы идентификации и мониторинг подозрительной активности.

Еще одной проблемой является доступность и удобство использования. Системы ДБО должны быть интуитивно понятными и простыми в использовании, чтобы удовлетворять потребности различных категорий клиентов, включая пожилых людей и тех, кто не обладает высоким уровнем технической грамотности. Банк должен обеспечивать техническую поддержку и помощь клиентам в использовании этих систем.

Кроме того, необходимо учитывать законодательные и регуляторные требования. Банки обязаны соблюдать законы о защите персональных данных, финансовом мониторинге и другие нормативные акты, что требует внедрения соответствующих процедур и технологий в системы ДБО.

1. **Информационные системы бухгалтерского учета (ИС БУ), их структура и классификация.**

Информационные системы бухгалтерского учета (ИС БУ) представляют собой совокупность программных и аппаратных средств, предназначенных для автоматизации бухгалтерских процессов в организациях. Эти системы помогают вести учет финансовых операций, контролировать движение денежных средств и материальных ценностей, а также формировать отчетность для внутренних и внешних пользователей.

Структура ИС БУ включает несколько основных компонентов. Во-первых, это модуль учета операций, который регистрирует все финансовые и хозяйственные операции компании. Этот модуль включает функции для ведения главной книги, учета дебиторской и кредиторской задолженности, доходов и расходов, а также движения денежных средств.

Во-вторых, это модуль формирования отчетности, который предназначен для подготовки различных форм отчетности, таких как бухгалтерский баланс, отчет о прибылях и убытках, и отчет о движении денежных средств. Этот модуль помогает соблюдать требования законодательства и внутренние стандарты компании.

В-третьих, модуль управления активами, который включает функции по учету основных средств, нематериальных активов и запасов. Этот модуль позволяет отслеживать амортизацию, инвентаризацию и переоценку активов.

В-четвертых, модуль налогового учета, который автоматизирует расчет и учет налогов, а также подготовку налоговой отчетности. Он помогает рассчитать налоговые обязательства и учитывать налоговые льготы.

В-пятых, модуль аналитического учета предоставляет инструменты для анализа финансово-хозяйственной деятельности предприятия, включая бюджетирование, планирование и прогнозирование, а также анализ финансовых показателей.

Наконец, модуль интеграции обеспечивает взаимодействие ИС БУ с другими информационными системами предприятия, такими как системы управления производством, управления персоналом и системы CRM.

Классификация ИС БУ может быть проведена по нескольким критериям. По масштабу применения выделяют корпоративные системы для крупных организаций, системы для средних и малых предприятий, а также системы для индивидуальных предпринимателей. По функциональности ИС БУ могут быть полнофункциональными, охватывающими все аспекты бухгалтерского учета, или специализированными, фокусирующимися на отдельных задачах, таких как расчет заработной платы или управление запасами. По способу реализации ИС БУ делятся на настольные системы, клиент-серверные системы и облачные системы, предоставляемые в виде онлайн-сервиса.

1. **Подходы к классификации бухгалтерских информационных систем, их и различия.**

Классификация бухгалтерских информационных систем (БИС) может быть осуществлена по различным критериям, в зависимости от целей и задач, которые они решают. Понимание этих классификаций важно для выбора наиболее подходящей системы для конкретной организации.

Один из основных подходов к классификации бухгалтерских информационных систем – по масштабу применения. Здесь выделяют системы для крупных организаций, для средних и малых предприятий, а также для индивидуальных предпринимателей. Крупные корпоративные системы, такие как SAP ERP и Oracle E-Business Suite, предназначены для комплексного управления финансовыми операциями в больших компаниях с множеством подразделений и сложной структурой учета. Они включают широкий спектр функций и модулей, обеспечивающих интеграцию всех бизнес-процессов. Системы для средних и малых предприятий, такие как 1С:Бухгалтерия и QuickBooks, обладают более ограниченным функционалом, но при этом удовлетворяют основные потребности бизнеса в учете и отчетности. Для индивидуальных предпринимателей и небольших компаний подходят упрощенные системы, такие как Wave и FreshBooks, которые ориентированы на базовые бухгалтерские операции.

Другой важный критерий классификации – по функциональности. Полнофункциональные системы охватывают весь спектр бухгалтерских задач: ведение главной книги, учет дебиторской и кредиторской задолженности, управление активами, налоговый учет и формирование отчетности. Специализированные системы, напротив, фокусируются на конкретных аспектах бухгалтерского учета, таких как расчет заработной платы, управление запасами или налоговый учет. Выбор между полнофункциональной и специализированной системой зависит от потребностей конкретной организации и ее бизнес-процессов.

Третий подход к классификации – по способу реализации. Настольные системы устанавливаются на локальные компьютеры и работают автономно, что подходит для небольших компаний или индивидуальных пользователей. Клиент-серверные системы, такие как Microsoft Dynamics GP, включают серверную часть для хранения данных и клиентские приложения для работы пользователей. Они обеспечивают централизованное управление данными и подходят для организаций со сложной структурой и распределенной системой доступа. Облачные системы, такие как Xero и NetSuite, предоставляются через интернет и доступны из любого места с доступом к сети. Они предлагают гибкость, простоту управления и возможность удаленного доступа, что особенно важно в современных условиях удаленной работы и мобильности.

Схожесть этих классификаций заключается в том, что все они направлены на организацию и автоматизацию бухгалтерского учета, обеспечивая точность, оперативность и соответствие законодательным требованиям. Основные различия заключаются в масштабе применения, функциональности и способе реализации, что позволяет выбирать систему в зависимости от специфики бизнеса, его размеров и потребностей.

1. **Модель системы счетов как основа любой бухгалтерской информационной системы. Модель организации синтетического учета. Модели организации аналитического учета.**

Модель системы счетов является основой любой бухгалтерской информационной системы (БИС). Она обеспечивает структурированное ведение бухгалтерского учета, позволяя регистрировать, классифицировать и анализировать финансовые операции предприятия. Система счетов включает синтетический и аналитический учет, которые совместно обеспечивают полноту и детализацию бухгалтерской информации.

Модель организации синтетического учета представляет собой обобщенный учет финансово-хозяйственных операций. Синтетические счета используются для учета обобщенной информации о состоянии и движении активов, обязательств и капитала предприятия. Они имеют высокий уровень обобщения и не предоставляют детализированных данных по отдельным элементам. Например, синтетический счет "Основные средства" будет содержать информацию о суммарной стоимости всех основных средств предприятия без детализации по каждому объекту. Синтетические счета необходимы для составления основного набора бухгалтерской отчетности, такой как баланс и отчет о прибылях и убытках.

Модель организации аналитического учета обеспечивает более детализированное ведение учета по каждому объекту, что позволяет проводить углубленный анализ финансово-хозяйственной деятельности. Аналитический учет ведется в разрезе аналитических счетов, которые детализируют данные синтетических счетов. Каждый синтетический счет может быть разделен на несколько аналитических, предоставляя подробную информацию по каждому элементу. Например, для синтетического счета "Основные средства" могут быть созданы аналитические счета по отдельным объектам, таким как здания, оборудование, транспортные средства и т.д. Аналитический учет позволяет отслеживать движение каждого объекта, учитывать его амортизацию, ремонты и модернизации.

Модель системы счетов в БИС обычно строится по принципу двойной записи, что означает, что каждая операция записывается на два счета – дебет одного счета и кредит другого. Это обеспечивает балансировку учета и позволяет контролировать правильность и полноту регистрации операций.

Бухгалтерская информационная система, основанная на модели системы счетов, обеспечивает автоматизацию всех этапов бухгалтерского учета. Сначала данные о финансово-хозяйственных операциях вводятся в систему, после чего происходит автоматическое распределение по синтетическим и аналитическим счетам. Система также позволяет формировать различные отчеты, как синтетические, так и аналитические, предоставляя пользователям необходимую информацию для принятия управленческих решений.

1. **Понятие базы данных. Виды баз данных. Проектирование баз данных.**

База данных – это организованная коллекция данных, предназначенная для эффективного хранения, управления и извлечения информации. Базы данных используются во многих сферах, включая бизнес, науку, медицину и другие области, где необходимо обрабатывать большие объемы данных. Существует несколько видов баз данных, которые классифицируются по структуре и способу организации данных.

Реляционные базы данных (РБД) являются наиболее распространенными. Они организованы в виде таблиц, где данные хранятся в строках и столбцах. Таблицы могут быть связаны между собой через ключи. Примеры таких систем включают MySQL, PostgreSQL и Oracle Database. РБД используют язык SQL (Structured Query Language) для управления данными.

Иерархические базы данных организованы в виде дерева, где каждая запись имеет одну родительскую и множество дочерних записей. Это подходит для структурированных данных с иерархическими отношениями, например, в каталогах товаров. Примером является IBM Information Management System (IMS).

Сетевые базы данных похожи на иерархические, но каждая запись может иметь несколько родительских записей. Это позволяет моделировать более сложные взаимосвязи. Примером сетевой базы данных является Integrated Data Store (IDS).

Объектно-ориентированные базы данных (ООБД) хранят данные в виде объектов, как в объектно-ориентированном программировании. Это позволяет интегрировать базы данных с объектно-ориентированными языками программирования, такими как Java и C++. Примером является db4o.

NoSQL базы данных предназначены для работы с большими объемами неструктурированных или полуструктурированных данных. Они не используют табличную модель и включают несколько типов: документо-ориентированные (например, MongoDB), колоночные (например, Cassandra), графовые (например, Neo4j) и ключ-значение (например, Redis). NoSQL базы данных обеспечивают высокую масштабируемость и гибкость в управлении данными.

Проектирование баз данных – это процесс создания логической и физической структуры базы данных, который включает несколько этапов.

Сначала происходит сбор и анализ требований, где определяются цели и задачи базы данных, потребности пользователей и бизнес-процессы, которые будут поддерживаться. Затем создается концептуальная модель данных с использованием модели «сущность-связь» (ER-модель), которая описывает сущности, атрибуты и их взаимосвязи. После этого концептуальная модель преобразуется в логическую, подходящую для конкретной системы управления базами данных (СУБД). В случае реляционной базы данных это означает создание схемы таблиц, ключей и отношений.

Далее следует физическое проектирование, где определяется физическая структура базы данных на уровне хранения, включая выбор индексов, оптимизацию запросов и определение требований к оборудованию. На этапе имплементации проектированная структура базы данных реализуется в выбранной СУБД, создаются таблицы, связи, индексы и другие объекты.

Заключительным этапом является тестирование и оптимизация базы данных. Проверяется работоспособность, тестируется производительность, оптимизируются запросы и структура для обеспечения эффективности и быстродействия.

1. **Общая характеристика языка запросов SQL. Декларативная парадигма. Основные операторы, виды операторов.**

SQL, или Structured Query Language, — это язык запросов, который используется для управления и манипулирования данными в реляционных базах данных. Основная задача SQL заключается в том, чтобы предоставлять удобный и мощный инструмент для работы с данными, находящимися в таблицах базы данных.

Одной из ключевых особенностей SQL является его декларативная парадигма. Это означает, что при написании SQL-запросов мы описываем, какие данные нам нужны, а не как их получить. Другими словами, мы указываем, что мы хотим сделать, а не какие шаги необходимо предпринять для выполнения этой задачи. Это делает SQL интуитивно понятным и позволяет пользователям сосредоточиться на логике своих запросов, а не на деталях их выполнения.

Основные операторы SQL можно разделить на несколько категорий:

DML (Data Manipulation Language) — операторы, предназначенные для манипулирования данными. К ним относятся:

* SELECT, который используется для извлечения данных из таблиц. Например, SELECT \* FROM employees вернет все данные из таблицы "employees".
* INSERT, который позволяет добавлять новые строки в таблицы. Например, INSERT INTO employees (name, position) VALUES ('John Doe', 'Manager').
* UPDATE, который используется для изменения существующих данных. Например, UPDATE employees SET position = 'Senior Manager' WHERE name = 'John Doe'.
* DELETE, который удаляет строки из таблиц. Например, DELETE FROM employees WHERE name = 'John Doe'.

DDL (Data Definition Language) — операторы, которые используются для определения и изменения структуры базы данных. К ним относятся:

* CREATE, который создает новые таблицы, базы данных или другие объекты. Например, CREATE TABLE employees (id INT PRIMARY KEY, name VARCHAR(50), position VARCHAR(50)).
* ALTER, который изменяет структуру существующих объектов. Например, ALTER TABLE employees ADD COLUMN salary DECIMAL(10, 2).
* DROP, который удаляет объекты из базы данных. Например, DROP TABLE employees.

DCL (Data Control Language) — операторы, которые управляют доступом к данным и привилегиями пользователей. К ним относятся:

* GRANT, который предоставляет пользователям определенные права. Например, GRANT SELECT ON employees TO user1.
* REVOKE, который отменяет ранее предоставленные права. Например, REVOKE SELECT ON employees FROM user1.

TCL (Transaction Control Language) — операторы, которые управляют транзакциями в базе данных. К ним относятся:

* COMMIT, который фиксирует все изменения, сделанные в рамках текущей транзакции. Например, после выполнения COMMIT все изменения становятся постоянными.
* ROLLBACK, который отменяет изменения, сделанные в рамках текущей транзакции. Например, ROLLBACK вернет базу данных в состояние до начала транзакции.
* SAVEPOINT, который создает точку сохранения внутри транзакции, к которой можно откатиться. Например, SAVEPOINT sp1.

1. **Оператор SELECT в SQL. Общий синтаксис, наиболее распространенные блоки. Условия, сортировки, ограничения. Примеры.**

Оператор `SELECT` в SQL используется для извлечения данных из одной или нескольких таблиц базы данных. Он является одним из наиболее часто используемых и мощных инструментов в SQL. Общий синтаксис оператора `SELECT` включает несколько основных блоков, которые позволяют гибко настраивать запросы.

Общий синтаксис оператора `SELECT` выглядит следующим образом:

SELECT [DISTINCT] столбец1, столбец2, ...

FROM таблица

[WHERE условия]

[GROUP BY столбец1, столбец2, ...]

[HAVING условия\_группировки]

[ORDER BY столбец1 [ASC|DESC], столбец2 [ASC|DESC], ...]

[LIMIT количество\_строк];

Рассмотрим наиболее распространенные блоки более подробно:

1. SELECT: Указывает столбцы, которые необходимо выбрать. Можно использовать `\*`, чтобы выбрать все столбцы. Выбор всех столбцов из таблицы `employees`:

SELECT name, age FROM employees;

2. DISTINCT: Убирает дубликаты из результатов. Выбор уникальных позиций из таблицы `employees`:

SELECT DISTINCT position FROM employees;

3. FROM: Указывает таблицу, из которой будут извлекаться данные.

SELECT \* FROM employees;

4. WHERE: Используется для фильтрации строк на основе определенных условий. Выбор имен и возрастов сотрудников старше 30 лет:

SELECT name, age FROM employees WHERE age > 30;

5. GROUP BY: Группирует строки, имеющие одинаковые значения в указанных столбцах. Часто используется вместе с агрегатными функциями (например, `COUNT`, `SUM`, `AVG`). Группировка сотрудников по позициям и подсчет количества сотрудников на каждой позиции:

SELECT position, COUNT(\*) FROM employees GROUP BY position;

6. HAVING: Фильтрует группы, определенные оператором `GROUP BY`, на основе определенных условий. Группировка сотрудников по позициям и подсчет количества сотрудников на каждой позиции, где количество сотрудников больше 1:

SELECT position, COUNT(\*) FROM employees GROUP BY position HAVING COUNT(\*) > 1;

7. ORDER BY: Сортирует результаты по указанным столбцам. По умолчанию сортировка производится по возрастанию (`ASC`), но можно указать сортировку по убыванию (`DESC`). Сортировка сотрудников по возрасту в порядке убывания:

SELECT name, age FROM employees ORDER BY age DESC;

8. LIMIT: Ограничивает количество возвращаемых строк. Ограничение результата десятью первыми строками:

SELECT \* FROM employees LIMIT 10;

1. **Оператор JOIN в SQL. Общий синтаксис. Виды объединений, их различия и области применений. Примеры.**

Оператор `JOIN` в SQL используется для объединения строк из двух или более таблиц на основе логического условия. Это позволяет извлекать связанные данные, хранящиеся в разных таблицах. Общий синтаксис оператора `JOIN` выглядит следующим образом:

SELECT столбцы

FROM таблица1

JOIN тип JOIN таблица2

ON условие\_соединения;

В SQL существует несколько видов объединений (`JOIN`), каждое из которых имеет свои особенности и области применения. Рассмотрим основные виды объединений и их различия.

1. \*\*INNER JOIN\*\*: Возвращает только те строки, которые имеют соответствующие значения в обеих таблицах. Если совпадений нет, строки не включаются в результат.

SELECT employees.name, departments.department\_name

FROM employees

INNER JOIN departments ON employees.department\_id = departments.id;

Этот запрос возвращает имена сотрудников и названия их отделов, только если сотрудник имеет соответствующий отдел.

2. \*\*LEFT JOIN (или LEFT OUTER JOIN)\*\*: Возвращает все строки из левой таблицы и соответствующие строки из правой таблицы. Если соответствующих строк нет, возвращаются `NULL` для столбцов правой таблицы.

SELECT employees.name, departments.department\_name

FROM employees

LEFT JOIN departments ON employees.department\_id = departments.id;

Этот запрос возвращает все имена сотрудников и названия их отделов. Если сотрудник не имеет отдела, будет возвращено `NULL` в колонке названия отдела.

3. \*\*RIGHT JOIN (или RIGHT OUTER JOIN)\*\*: Возвращает все строки из правой таблицы и соответствующие строки из левой таблицы. Если соответствующих строк нет, возвращаются `NULL` для столбцов левой таблицы.

SELECT employees.name, departments.department\_name

FROM employees

RIGHT JOIN departments ON employees.department\_id = departments.id;

Этот запрос возвращает все названия отделов и имена сотрудников. Если отдел не имеет сотрудников, будет возвращено `NULL` в колонке имени сотрудника.

4. \*\*FULL JOIN (или FULL OUTER JOIN)\*\*: Возвращает все строки, когда есть совпадения в одной из таблиц. Если соответствующих строк нет, возвращаются `NULL` для столбцов, где нет совпадений.

SELECT employees.name, departments.department\_name

FROM employees

FULL JOIN departments ON employees.department\_id = departments.id;

Этот запрос возвращает все имена сотрудников и названия отделов, включая строки, где нет совпадений (возвращаются `NULL`).

5. \*\*CROSS JOIN\*\*: Возвращает декартово произведение двух таблиц, то есть каждая строка из первой таблицы сочетается с каждой строкой из второй таблицы.

SELECT employees.name, departments.department\_name

FROM employees

CROSS JOIN departments;

Этот запрос возвращает все возможные комбинации сотрудников и отделов, то есть каждая строка из таблицы сотрудников соединяется с каждой строкой из таблицы отделов.

6. \*\*SELF JOIN\*\*: Используется для соединения таблицы с самой собой. Это полезно, когда необходимо сравнить строки одной таблицы.

SELECT e1.name AS Employee1, e2.name AS Employee2

FROM employees e1

INNER JOIN employees e2 ON e1.manager\_id = e2.id;

Этот запрос возвращает пары сотрудников, где один сотрудник является менеджером другого.

1. **Использование групповых функций в SQL. Синтаксис оператора SELECT при использовании группировок. Примеры использования.**

Групповые функции в SQL используются для выполнения операций над наборами строк и возвращают одно значение для каждой группы строк. Эти функции включают такие функции, как `COUNT`, `SUM`, `AVG`, `MIN`, и `MAX`. Они позволяют агрегировать данные и получать сводную информацию.

Когда используются групповые функции, часто необходимо использовать оператор `GROUP BY`, чтобы указать, по каким столбцам группировать данные. Общий синтаксис оператора `SELECT` при использовании групповых функций выглядит следующим образом:

SELECT столбец1, групповые\_функции

FROM таблица

[WHERE условия]

GROUP BY столбец1, столбец2, ...

[HAVING условия\_группировки]

[ORDER BY столбец1, столбец2, ...];

Теперь рассмотрим основные групповые функции и их применение:

- \*\*COUNT()\*\*: Возвращает количество строк в группе.

SELECT department\_id, COUNT(\*)

FROM employees

GROUP BY department\_id;

Этот запрос возвращает количество сотрудников в каждом отделе.

- \*\*SUM()\*\*: Возвращает сумму значений в группе.

SELECT department\_id, SUM(salary)

FROM employees

GROUP BY department\_id;

Этот запрос возвращает общую сумму зарплат в каждом отделе.

- \*\*AVG()\*\*: Возвращает среднее значение в группе.

SELECT department\_id, AVG(salary)

FROM employees

GROUP BY department\_id;

Этот запрос возвращает среднюю зарплату в каждом отделе.

- \*\*MIN()\*\*: Возвращает минимальное значение в группе.

SELECT department\_id, MIN(salary)

FROM employees

GROUP BY department\_id;

Этот запрос возвращает минимальную зарплату в каждом отделе.

- \*\*MAX()\*\*: Возвращает максимальное значение в группе.

SELECT department\_id, MAX(salary)

FROM employees

GROUP BY department\_id;

Этот запрос возвращает максимальную зарплату в каждом отделе.

Когда мы используем групповые функции, иногда возникает необходимость фильтровать группы на основе агрегированных значений. Для этого используется предложение `HAVING`, которое применяется после группировки данных.

Пример с `HAVING`:

SELECT department\_id, COUNT(\*)

FROM employees

GROUP BY department\_id

HAVING COUNT(\*) > 5;

Этот запрос возвращает идентификаторы отделов и количество сотрудников в них, но только для тех отделов, где количество сотрудников больше 5.

Важные моменты при использовании групповых функций и `GROUP BY`:

- Все столбцы, которые не входят в агрегатные функции, должны быть включены в предложение `GROUP BY`.

- `HAVING` используется для фильтрации групп после применения групповых функций, тогда как `WHERE` используется для фильтрации строк до группировки.

Рассмотрим более сложный пример:

SELECT department\_id, AVG(salary) AS avg\_salary, COUNT(\*) AS num\_employees

FROM employees

WHERE hire\_date > '2024-06-05'

GROUP BY department\_id

HAVING AVG(salary) > 50000

ORDER BY avg\_salary DESC;

Этот запрос выполняет следующие шаги:

1. Фильтрует сотрудников, принятых на работу после 5 июня 2024 года.

2. Группирует оставшиеся строки по идентификаторам отделов.

3. Вычисляет среднюю зарплату и количество сотрудников в каждом отделе.

4. Фильтрует группы, где средняя зарплата превышает 50 000.

5. Сортирует результаты по средней зарплате в порядке убывания.

1. **Реляционные базы данных. Нормализация отношений. Первая нормальная форма. Вторая нормальная форма. Третья нормальная форма. Другие нормальные формы.**

Реляционные базы данных (РБД) — это тип баз данных, где данные хранятся в виде таблиц (или отношений). Каждая таблица состоит из строк (записей) и столбцов (полей). Основное преимущество реляционных баз данных заключается в их способности легко управлять и манипулировать данными с помощью структурированных запросов на языке SQL.

Нормализация — это процесс организации данных в базе данных таким образом, чтобы уменьшить избыточность данных и улучшить целостность данных. Нормализация включает несколько шагов, известных как нормальные формы. Основные нормальные формы включают первую, вторую и третью нормальные формы.

Первая нормальная форма (1НФ) требует, чтобы все значения атрибутов (столбцов) были атомарными, то есть неделимыми. Это означает, что каждая ячейка таблицы должна содержать одно значение, а не набор значений или подмножества данных. Например, если у нас есть таблица с информацией о студентах и курсах, каждый студент должен иметь отдельную запись для каждого курса, а не одну запись с несколькими курсами.

Вторая нормальная форма (2НФ) требует выполнения всех условий первой нормальной формы и наличия полной функциональной зависимости атрибутов от первичного ключа. Это означает, что каждый неключевой атрибут должен зависеть от всего первичного ключа, а не только от его части. В таблицах, где первичный ключ составной (состоит из нескольких столбцов), необходимо убедиться, что каждый неключевой столбец зависит от всей комбинации ключевых столбцов, а не от их части.

Третья нормальная форма (3НФ) требует выполнения всех условий второй нормальной формы и отсутствия транзитивной зависимости между неключевыми атрибутами. Это означает, что неключевые атрибуты должны зависеть только от первичного ключа и не должны зависеть друг от друга. Например, если у нас есть таблица с информацией о заказах, включающая атрибуты "номер заказа", "код клиента" и "имя клиента", мы должны разделить эту таблицу на две: одна с "номер заказа" и "код клиента", другая с "код клиента" и "имя клиента".

Существует также несколько других нормальных форм, таких как Бойс-Кодд нормальная форма (BCNF), четвертая нормальная форма (4НФ) и пятая нормальная форма (5НФ). BCNF является усилением третьей нормальной формы и требует, чтобы в любой неключевой функциональной зависимости слева находился суперключ. Четвертая нормальная форма устраняет многозначные зависимости, а пятая нормальная форма направлена на устранение аномалий при соединении данных.

Процесс нормализации важен для поддержания целостности данных и обеспечения эффективности работы базы данных. Он помогает избежать проблем, связанных с избыточностью данных и аномалиями при вставке, обновлении и удалении данных. Например, в ненормализованной базе данных может возникнуть ситуация, когда одно и то же значение хранится в нескольких местах, что приводит к необходимости синхронизации данных и возможным ошибкам.

На практике баланс между нормализацией и производительностью базы данных часто диктуется конкретными требованиями проекта. В некоторых случаях может быть предпочтительно использовать денормализацию, чтобы улучшить скорость выполнения запросов за счет некоторой избыточности данных. Однако знание и понимание нормальных форм остаются основой для проектирования эффективных и надежных реляционных баз данных.

1. **Транзакции в SQL. Понятие транзакции, организация транзакций, вложенные транзакции. Транзакции и управление параллельной работой.**

Транзакции в SQL являются основным механизмом для обеспечения целостности данных в реляционных базах данных. Транзакция представляет собой последовательность одной или нескольких операций SQL, выполняемых как единое целое. Транзакция имеет четыре основных свойства, известные как ACID: атомарность, согласованность, изоляция и долговечность.

Атомарность (Atomicity) означает, что все операции внутри транзакции выполняются полностью или не выполняются вообще. Если хотя бы одна операция не может быть выполнена, все изменения, внесенные другими операциями транзакции, откатываются (rollback).

Согласованность (Consistency) гарантирует, что транзакция переводит базу данных из одного согласованного состояния в другое. Это означает, что любые правила, ограничения и инварианты базы данных остаются в силе до и после выполнения транзакции.

Изоляция (Isolation) обеспечивает, что параллельное выполнение транзакций не приводит к некорректным результатам. Каждая транзакция должна выполняться так, как будто она является единственной активной транзакцией в системе. Уровни изоляции включают чтение нефиксированных данных (read uncommitted), чтение фиксированных данных (read committed), повторяемое чтение (repeatable read) и сериализуемость (serializable).

Долговечность (Durability) гарантирует, что результаты завершенной транзакции будут сохранены и не потеряны, даже в случае сбоя системы.

Для управления транзакциями в SQL используются ключевые слова `BEGIN`, `COMMIT` и `ROLLBACK`. Транзакция начинается с `BEGIN`, все изменения фиксируются (commit) с помощью `COMMIT`, или откатываются (rollback) с помощью `ROLLBACK` в случае ошибки.

Пример организации транзакции:

BEGIN;

UPDATE accounts SET balance = balance - 100 WHERE account\_id = 1;

UPDATE accounts SET balance = balance + 100 WHERE account\_id = 2;

COMMIT;

В этом примере деньги переводятся с одного счета на другой. Если обе операции успешны, изменения фиксируются с помощью `COMMIT`. Если одна из операций не может быть выполнена, транзакция откатывается (rollback), и баланс остается неизменным.

Вложенные транзакции позволяют начать новую транзакцию внутри другой транзакции. Если внутренняя транзакция завершается с ошибкой, только она откатывается, в то время как внешняя транзакция может продолжаться. Однако не все системы управления базами данных (СУБД) поддерживают вложенные транзакции.

Управление параллельной работой транзакций связано с необходимостью обеспечения изоляции и предотвращения таких проблем, как потеря обновлений (lost updates), грязное чтение (dirty reads), неповторяющееся чтение (non-repeatable reads) и фантомные чтения (phantom reads). Для этого используются механизмы блокировок и уровни изоляции.

Проблема потери обновлений возникает, когда две транзакции одновременно пытаются изменить одно и то же значение. Грязное чтение происходит, когда одна транзакция читает данные, измененные другой транзакцией, которая еще не завершена. Неповторяющееся чтение возникает, когда одна транзакция читает данные дважды, и между этими чтениями другая транзакция изменяет эти данные. Фантомные чтения происходят, когда одна транзакция выполняет два одинаковых запроса, но другой запрос добавляет или удаляет строки между этими запросами, изменяя результат второго запроса.

Уровни изоляции помогают справляться с этими проблемами. Уровень `READ UNCOMMITTED` допускает все виды аномалий, `READ COMMITTED` предотвращает грязное чтение, `REPEATABLE READ` предотвращает грязное и неповторяющееся чтение, а `SERIALIZABLE` предотвращает все виды аномалий, но может значительно снижать производительность.

1. **Типы задач машинного обучения: классификация, регрессия, кластеризация. Обучение с учителем и без учителя. Машинное обучение как оптимизационная задача.**

Машинное обучение (ML) включает различные типы задач, каждая из которых решает специфические проблемы и используется в различных приложениях. Основные типы задач машинного обучения включают классификацию, регрессию и кластеризацию.

Классификация — это задача, в которой алгоритм машинного обучения обучается на наборе данных, где каждому примеру присвоена метка класса. Цель классификации — научить модель правильно определять класс новых, невиданных данных. Примеры задач классификации включают распознавание спама в электронной почте, диагностику заболеваний на основе симптомов и классификацию изображений. Модели классификации могут быть бинарными (два класса) или многоклассовыми (более двух классов).

Регрессия — это задача предсказания непрерывного значения на основе входных данных. Алгоритм регрессии обучается на данных с известными значениями и пытается предсказать выходное значение для новых данных. Примеры задач регрессии включают прогнозирование цен на недвижимость, предсказание температуры, оценку спроса на продукты и анализ временных рядов. Важной особенностью регрессии является то, что выходные значения являются числовыми и непрерывными.

Кластеризация — это задача группировки объектов в такие кластеры, что объекты внутри одного кластера более похожи друг на друга, чем на объекты из других кластеров. В отличие от классификации, кластеризация не требует наличия меток классов в данных, и её цель — выявить естественные группы или структуры в данных. Примеры применения кластеризации включают сегментацию клиентов в маркетинге, группировку документов по темам и анализ социальных сетей.

Машинное обучение можно разделить на обучение с учителем и обучение без учителя. Обучение с учителем (supervised learning) предполагает, что модель обучается на данных, которые имеют метки или известные выходные значения. Примеры задач обучения с учителем включают классификацию и регрессию, где модели обучаются на размеченных данных и затем применяются для предсказания меток или значений для новых данных.

Обучение без учителя (unsupervised learning) не требует меток или выходных значений в данных. Вместо этого модель пытается выявить скрытые структуры или закономерности в данных. Кластеризация является примером задачи обучения без учителя, где модель группирует данные по схожести без использования предварительно известных меток.

Машинное обучение можно также рассматривать как задачу оптимизации, где цель состоит в минимизации или максимизации определенной функции потерь (или стоимости). Эта функция потерь измеряет, насколько хорошо модель предсказывает выходные значения или выполняет заданную задачу. В процессе обучения модель находит такие параметры, которые минимизируют функцию потерь, что приводит к улучшению её производительности на новых данных. Примеры функций потерь включают среднеквадратичную ошибку (MSE) для задач регрессии и кросс-энтропию для задач классификации.

Оптимизация играет ключевую роль в обучении моделей машинного обучения. Алгоритмы оптимизации, такие как градиентный спуск и его вариации (например, стохастический градиентный спуск, Adam), используются для нахождения наилучших параметров модели. Эти алгоритмы итеративно обновляют параметры модели, чтобы минимизировать функцию потерь.

1. **Регрессия как задача машинного обучения. Постановка задачи, математическая формализация. Целевая функция. Примеры.**

Регрессия в контексте машинного обучения представляет собой задачу предсказания непрерывного числового значения на основе входных данных. Основная цель регрессии — установить зависимость между входными переменными (предикторами) и выходной переменной (целевым значением).

Постановка задачи регрессии заключается в том, чтобы найти такую функцию, которая наилучшим образом аппроксимирует зависимость между входными данными и целевым значением. Пусть у нас есть набор данных, состоящий из 𝑛 пар (𝑥𝑖,𝑦𝑖), где 𝑥𝑖 — это вектор входных признаков, а 𝑦𝑖 — это соответствующее выходное значение. Задача регрессии — обучить модель, которая может предсказывать значение 𝑦 для новых, невиданных данных 𝑥.

Математическая формализация задачи регрессии заключается в нахождении функции 𝑓(𝑥), которая минимизирует разницу между предсказанными значениями 𝑦^=𝑓(𝑥) и реальными значениями 𝑦. Обычно это достигается минимизацией целевой функции, которая измеряет ошибку модели. Одной из наиболее распространенных целевых функций для задач регрессии является среднеквадратичная ошибка (MSE), которая определяется как: MSE=*1/n*∑*i*=1*n*​(*yi*​−*y*^​*i*​)2

Где 𝑦𝑖 — это реальное значение, 𝑦^𝑖 — предсказанное значение, и 𝑛 — общее количество примеров в наборе данных.

Примером простой модели регрессии является линейная регрессия. В линейной регрессии предполагается, что зависимость между входными переменными и выходной переменной является линейной. Модель линейной регрессии описывается уравнением:

𝑦^=𝑤0+𝑤1𝑥1+𝑤2𝑥2+…+𝑤𝑚𝑥𝑚

​Где 𝑦^ — предсказанное значение, 𝑥1,𝑥2,…,𝑥𝑚 — входные переменные, а 𝑤0,𝑤1,…,𝑤𝑚 — коэффициенты модели, которые необходимо найти. Коэффициенты 𝑤 подбираются таким образом, чтобы минимизировать среднеквадратичную ошибку.

Пример задачи регрессии может включать предсказание цен на недвижимость на основе различных характеристик дома, таких как площадь, количество комнат, возраст здания и местоположение. В этом случае 𝑥 — это вектор характеристик дома, а 𝑦 — цена дома. Модель регрессии обучается на исторических данных о продаже домов и их характеристиках, чтобы предсказывать цену для новых объектов недвижимости.

Другим примером может быть прогнозирование температуры на основе данных о погодных условиях, таких как влажность, давление, скорость ветра и время суток. Здесь входные данные включают различные метеорологические параметры, а выходное значение — температура.

В дополнение к линейной регрессии существуют и другие методы регрессии, такие как полиномиальная регрессия, которая учитывает нелинейные зависимости, и методы на основе деревьев решений, такие как случайный лес и градиентный бустинг, которые могут моделировать более сложные зависимости между признаками и целевыми значениями.

1. **Классификация как задача машинного обучения. Постановка задачи, математическая формализация. Целевая функция. Примеры.**

Классификация в машинном обучении — это задача, в которой модель обучается на наборе данных, содержащих входные переменные и соответствующие метки классов, чтобы предсказать метку класса для новых, невиданных данных. Основная цель классификации — определить, к какому классу принадлежит данный объект на основе его характеристик.

Постановка задачи классификации включает в себя набор данных, состоящий из n примеров, где каждый пример представляет собой пару (x\_i, y\_i). Здесь x\_i — это вектор входных признаков, а y\_i — это метка класса, к которому принадлежит пример i. Модель классификации должна научиться предсказывать метку класса y для новых примеров x.

Математическая формализация задачи классификации включает поиск функции f(x), которая принимает вектор признаков x и возвращает метку класса y^. Для задач классификации часто используются вероятностные модели, которые предсказывают вероятность принадлежности объекта к каждому из возможных классов. В таких случаях выходная функция может быть вектором вероятностей, и финальное решение принимается на основе класса с наибольшей вероятностью.

Целевая функция в задачах классификации обычно направлена на минимизацию ошибок классификации. Одной из наиболее распространенных целевых функций является кросс-энтропийная ошибка (cross-entropy loss), которая измеряет разницу между предсказанными вероятностями и истинными метками классов. Формула кросс-энтропийной ошибки для бинарной классификации может быть записана как: Loss=−1/*n*​∑*i*=1*n*​[*yi*​log(*p*^​*i*​)+(1−*yi*​)log(1−*p*^​*i*​)]

Здесь y\_i — истинная метка класса (0 или 1), а p^\_i — предсказанная вероятность принадлежности к классу 1.

Пример задачи классификации — это определение, является ли электронное письмо спамом или нет. В этом случае каждый электронный письмо представляется в виде вектора признаков (например, частота встречаемости определенных слов), а метка класса указывает, является ли письмо спамом (1) или нет (0). Модель обучается на размеченных данных, чтобы затем классифицировать новые письма как спам или не спам.

Другой пример — классификация изображений, где каждый пиксель изображения представляет собой признак, а метка класса указывает, что изображено на картинке (например, кошка, собака, автомобиль и т.д.). Модель обучается на наборе размеченных изображений, чтобы затем предсказывать класс для новых изображений.

Различные алгоритмы классификации, такие как логистическая регрессия, деревья решений, случайные леса, метод опорных векторов (SVM), нейронные сети и другие, используются для решения задач классификации. Каждый из этих алгоритмов имеет свои преимущества и ограничения в зависимости от характера данных и конкретной задачи.

1. **Метрики качества результатов машинного обучения. Метрики качества моделей регрессии и классификации. MAE, MSE, Precision, Recall, F1, ROC-AUC.**

В машинном обучении для оценки качества моделей используются различные метрики. Эти метрики позволяют измерить, насколько хорошо модель предсказывает целевые значения для новых данных. Метрики делятся на две основные категории: для моделей регрессии и для моделей классификации.

Для моделей регрессии основными метриками качества являются средняя абсолютная ошибка (MAE) и среднеквадратичная ошибка (MSE).

MAE (Mean Absolute Error) — это среднее значение абсолютных ошибок между предсказанными и реальными значениями. Она вычисляется как среднее арифметическое абсолютных разностей между предсказанными значениями 𝑦^\_𝑖 и реальными значениями 𝑦\_𝑖:

MAE = 1/𝑛∑𝑖=1𝑛∣𝑦^\_𝑖−𝑦𝑖∣

MAE показывает, насколько предсказания модели в среднем отклоняются от истинных значений.

MSE (Mean Squared Error) — это среднее значение квадратов ошибок между предсказанными и реальными значениями. Она вычисляется как среднее арифметическое квадратов разностей между предсказанными значениями 𝑦^\_𝑖 и реальными значениями 𝑦𝑖:

MSE = 1/𝑛∑𝑖=1𝑛(𝑦^\_𝑖−𝑦𝑖)2

MSE более чувствительна к большим ошибкам, так как квадраты больших отклонений значительно увеличивают значение ошибки. Она подчеркивает важность точных предсказаний.

Для моделей классификации используются такие метрики, как точность (Precision), полнота (Recall), F1-Score и ROC-AUC.

Precision (Точность) — это доля правильных положительных предсказаний среди всех положительных предсказаний. Она вычисляется как отношение числа истинных положительных предсказаний к сумме истинных положительных и ложных положительных предсказаний:

Precision =𝑇𝑃/(𝑇𝑃+𝐹𝑃)

Recall (Полнота) — это доля правильных положительных предсказаний среди всех истинных положительных значений. Она вычисляется как отношение числа истинных положительных предсказаний к сумме истинных положительных и ложных отрицательных предсказаний:

Recall = 𝑇𝑃/(𝑇𝑃+𝐹𝑁)

F1-Score — это гармоническое среднее между точностью и полнотой. Она используется, когда необходимо найти баланс между точностью и полнотой. F1-Score вычисляется по формуле:

F1-Score = 2 × (Precision×Recall)/(Precision+Recall)

ROC-AUC (Receiver Operating Characteristic - Area Under Curve) — это метрика, которая оценивает качество классификации, независимо от порога отсечения. ROC-кривая строится путем отображения истинных положительных значений (TPR) против ложных положительных значений (FPR) при различных порогах классификации. Площадь под ROC-кривой (AUC) измеряет способность модели различать положительные и отрицательные классы. Значение AUC варьируется от 0.5 (случайное угадывание) до 1 (идеальная классификация).

1. **Метод ближайших соседей (kNN) и его обобщения. Подбор числа k. Методы метрической классификации в библиотеке Scikit-Learn. (с 45 вопросом)**

Метод ближайших соседей (k-Nearest Neighbors, kNN) — это алгоритм машинного обучения, который можно применять как для классификации, так и для регрессии. Основная идея kNN заключается в том, что для предсказания метки класса нового объекта нужно найти 𝑘 объектов в тренировочном наборе, которые находятся ближе всего к новому объекту, и использовать эти 𝑘 ближайших соседей для предсказания. В случае классификации предсказание метки класса происходит путем голосования среди 𝑘 ближайших соседей: метка, которая встречается чаще всего, назначается новому объекту. В случае регрессии, предсказанное значение является средним значением по 𝑘 ближайшим соседям.

Подбор числа 𝑘 в алгоритме kNN является важным шагом, поскольку выбор неправильного значения 𝑘 может сильно повлиять на качество модели. Малое значение 𝑘 (например, 𝑘 =1) может привести к переобучению, когда модель слишком чувствительна к шуму в данных. Большое значение 𝑘 может привести к недообучению, когда модель становится слишком общей и не учитывает особенности данных. Оптимальное значение 𝑘 часто подбирается с помощью методов кросс-валидации.

Методы метрической классификации в библиотеке Scikit-Learn включают реализацию алгоритма kNN и его обобщения. Scikit-Learn предоставляет класс KNeighborsClassifier для задач классификации и KNeighborsRegressor для задач регрессии. Эти классы позволяют легко настроить и обучить модели на основе метода ближайших соседей.

Пример использования метода kNN для классификации в Scikit-Learn:

from sklearn.datasets import load\_iris

from sklearn.model\_selection import train\_test\_split

from sklearn.neighbors import KNeighborsClassifier

from sklearn.metrics import accuracy\_score

# Загрузка данных

data = load\_iris()

X = data.data

y = data.target

# Разделение данных на обучающую и тестовую выборки

X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y, test\_size=0.3, random\_state=42)

# Создание и обучение модели kNN

k = 5

model = KNeighborsClassifier(n\_neighbors=k)

model.fit(X\_train, y\_train)

# Предсказание и оценка качества модели

y\_pred = model.predict(X\_test)

accuracy = accuracy\_score(y\_test, y\_pred)

print(f'Accuracy: {accuracy}')

Scikit-Learn также предоставляет множество других методов метрической классификации, таких как метод взвешенных ближайших соседей, где более близкие соседи имеют больший вес при принятии решения, и различные метрики расстояния, такие как евклидово расстояние, манхэттенское расстояние и расстояние Минковского.

1. **Кластеризация как задача машинного обучения. Постановка задачи, математическая формализация. Целевая функция. Примеры. (с 44 вопросом)**

Кластеризация — это метод машинного обучения без учителя, который используется для группировки объектов в кластеры таким образом, что объекты внутри одного кластера более похожи друг на друга, чем на объекты из других кластеров. Основная цель кластеризации — выявить скрытую структуру данных и обнаружить закономерности.

Задача кластеризации формулируется следующим образом: дано множество объектов, представленных в виде набора признаков. Необходимо разделить эти объекты на несколько групп или кластеров так, чтобы объекты внутри одного кластера были максимально похожи, а объекты из разных кластеров — максимально различны.

Математически это можно формализовать на примере метода k-средних (k-means). В этом методе мы пытаемся минимизировать сумму квадратов расстояний от каждого объекта до центра его кластера. Центр кластера вычисляется как среднее арифметическое всех объектов в этом кластере. Целевая функция, которую мы минимизируем, выглядит так: сумма квадратов расстояний между объектами и центрами их кластеров. Алгоритм итеративно распределяет объекты по кластерам и пересчитывает центры кластеров до тех пор, пока распределение объектов не стабилизируется.

Существуют различные методы кластеризации. Например, метод k-средних, который хорошо работает на числовых данных и используется благодаря своей простоте и эффективности. Иерархическая кластеризация строит дерево кластеров, где агломеративный подход начинается с каждого объекта в отдельном кластере и объединяет самые близкие кластеры, пока не останется один кластер или не будет достигнуто желаемое количество кластеров. Метод DBSCAN, основанный на плотности, выделяет кластеры на основе плотности объектов и хорошо справляется с данными, содержащими кластеры произвольной формы и шум.

Примеры применения кластеризации разнообразны. В маркетинге она используется для сегментации клиентов на группы с похожими покупательскими поведениями или предпочтениями, что помогает компаниям лучше понимать своих клиентов и предлагать им персонализированные предложения. В задаче выявления аномалий кластеризация помогает обнаруживать объекты, которые значительно отличаются от большинства, что может указывать на ошибки или мошеннические действия. В обработке изображений кластеризация применяется для сегментации изображений, где пиксели сгруппированы в области на основе их цветовых или текстурных характеристик.

1. **Структура проекта нативного Android приложения. Основные файлы. Манифест приложения. Файлы с разметкой. Файлы ресурсов. Использование альтернативных ресурсов. Используемые языки для разработки и описания приложения.**

Структура проекта нативного Android приложения организована таким образом, чтобы упростить управление различными компонентами приложения. Основными элементами структуры являются каталоги и файлы, которые содержат код, ресурсы и метаданные приложения. Рассмотрим основные файлы и каталоги в структуре проекта Android.

Во-первых, в корне проекта находится файл build.gradle, который содержит информацию о сборке приложения, зависимостях и настройках компиляции. В корне также может быть файл settings.gradle, который управляет конфигурацией проекта, если проект многомодульный.

В каталоге app находится основной код приложения и ресурсы. Внутри этого каталога важнейшие файлы и директории включают:

* src/main/java: содержит исходный код приложения на языке Java или Kotlin. Здесь размещены пакеты и классы приложения.
* src/main/res: содержит ресурсы приложения, такие как разметка интерфейсов, изображения, строки и стили.
* src/main/AndroidManifest.xml: это манифест приложения, который описывает основные компоненты приложения, разрешения, которые требуются приложению, и другую важную информацию.

Манифест приложения (AndroidManifest.xml) — это центральный файл конфигурации приложения. Он содержит информацию о пакете приложения, определяет компоненты (активности, сервисы, ресиверы и провайдеры контента), разрешения, необходимые приложению, и другие важные метаданные. Например, каждая активность, которая может быть запущена, должна быть объявлена в манифесте.

Файлы с разметкой находятся в каталоге res/layout. Эти файлы описывают пользовательский интерфейс приложения в формате XML. Каждый XML-файл представляет собой отдельный экран или часть экрана. Например, файл activity\_main.xml может описывать главный экран приложения. В этих файлах определяются виды и их свойства: кнопки, текстовые поля, изображения и т.д.

Файлы ресурсов включают изображения, строки, стили и другие данные, используемые в приложении. Каталог res/drawable содержит изображения (например, PNG, JPG) и другие графические ресурсы. Каталог res/values содержит XML-файлы с определениями строк (strings.xml), цветов (colors.xml), размеров (dimens.xml), стилей (styles.xml) и других значений.

Использование альтернативных ресурсов позволяет адаптировать приложение под различные устройства и конфигурации. Например, для поддержки разных экранных разрешений используются подкаталоги drawable-hdpi, drawable-mdpi, drawable-xhdpi и т.д. Для поддержки разных языков и регионов используются подкаталоги values-en, values-ru и т.д. Android автоматически выбирает подходящие ресурсы в зависимости от конфигурации устройства.

Для разработки нативных Android приложений используются языки Java и Kotlin. Kotlin является официально поддерживаемым языком для Android и рекомендуется Google. Java и Kotlin компилируются в байт-код, который выполняется виртуальной машиной Android (ART или Dalvik).

1. **Компоненты Android приложений. Основные виды компонентов, их особенности и назначение. Жизненный цикл компонентов и использование методов жизненного цикла.**

В Android-приложениях есть четыре основных вида компонентов: активности, службы, широковещательные ресиверы и провайдеры контента. Каждый из этих компонентов играет важную роль в функциональности приложения и имеет свои особенности и назначение.

Активности (Activities) представляют собой экран пользовательского интерфейса. Это основная точка взаимодействия пользователя с приложением. Каждая активность отвечает за отображение интерфейса и обработку взаимодействий с пользователем. Жизненный цикл активности включает несколько стадий: создание (onCreate), запуск (onStart), восстановление (onResume), приостановка (onPause), остановка (onStop) и уничтожение (onDestroy). Эти методы позволяют управлять состоянием активности и сохранять важные данные при изменении состояния.

Службы (Services) работают в фоновом режиме и не предоставляют интерфейс для пользователя. Они используются для выполнения длительных операций, таких как загрузка данных из сети или воспроизведение музыки. Жизненный цикл службы начинается с метода onCreate, который вызывается при создании службы. После этого вызывается onStartCommand для обработки команд, и onDestroy для остановки службы. В зависимости от типа службы она может быть привязанной (bound service), и тогда используются методы onBind и onUnbind для управления привязкой клиента к службе.

Широковещательные ресиверы (Broadcast Receivers) предназначены для получения и обработки системных и пользовательских широковещательных сообщений. Они могут реагировать на события, такие как разрядка батареи или получение SMS. Жизненный цикл широковещательного ресивера короткий: метод onReceive вызывается при получении сообщения, после чего ресивер завершается. Эти компоненты часто используются для выполнения краткосрочных задач, таких как запуск уведомления.

Провайдеры контента (Content Providers) обеспечивают доступ к данным между различными приложениями. Они управляют доступом к центральной базе данных и позволяют приложениям запрашивать и изменять данные. Жизненный цикл провайдера контента начинается с метода onCreate, который вызывается при первом запросе к провайдеру. Основные методы включают query для выборки данных, insert для добавления, update для изменения и delete для удаления данных. Эти методы обеспечивают стандартный интерфейс для работы с данными и поддерживают их целостность.

Каждый из этих компонентов имеет уникальный жизненный цикл и методы, которые обеспечивают его корректное функционирование в рамках приложения. Знание и правильное использование методов жизненного цикла позволяют разработчику создавать стабильные и эффективные приложения, которые правильно реагируют на изменения в состоянии и взаимодействии с пользователем.

1. **Хранение данных в Android приложении. Виды хранилищ данных, их особенности и использование из приложения.**

В Android-приложениях существуют различные виды хранилищ данных, которые используются в зависимости от типа данных и требований к их доступности и безопасности. Основные виды хранилищ данных включают внутреннее хранилище (Internal Storage), внешнее хранилище (External Storage), базу данных SQLite, SharedPreferences и сети. Каждый из этих видов хранилищ имеет свои особенности и сценарии использования.

Внутреннее хранилище (Internal Storage) предоставляет приватное пространство для хранения данных, доступное только данному приложению. Данные, сохраненные во внутреннем хранилище, остаются доступными до тех пор, пока приложение не будет удалено. Внутреннее хранилище используется для хранения конфиденциальных данных и данных, которые не должны быть доступны другим приложениям. Файлы во внутреннем хранилище можно записывать и читать с помощью стандартных методов Java для работы с файлами.

Внешнее хранилище (External Storage) предоставляет доступ к файлам, которые могут быть общими для всех приложений и пользователей устройства. Внешнее хранилище используется для хранения больших объемов данных, таких как фотографии, видео и документы, которые должны быть доступны вне приложения. При использовании внешнего хранилища необходимо запрашивать разрешения на чтение и запись данных. Доступ к файлам на внешнем хранилище также осуществляется с помощью стандартных методов Java для работы с файлами.

База данных SQLite — это встроенная реляционная база данных, которая позволяет приложениям хранить структурированные данные в виде таблиц. SQLite является мощным инструментом для работы с данными и поддерживает стандартный язык SQL для выполнения операций с данными. SQLite используется для хранения данных, которые требуют сложных запросов и структурированных отношений между сущностями. Для работы с SQLite в Android предоставляются классы SQLiteOpenHelper и SQLiteDatabase, которые облегчают создание и управление базой данных.

SharedPreferences предназначены для хранения небольших наборов данных в виде пар "ключ-значение". Этот метод хранения данных особенно полезен для сохранения настроек и конфигураций приложения, таких как параметры пользователя или состояние интерфейса. SharedPreferences обеспечивают простой способ записи и чтения данных с использованием методов put и get для различных типов данных, таких как строки, числа и булевы значения.

Сети (Network) используются для хранения и получения данных с удаленных серверов через интернет. Это подход особенно важен для приложений, которые взаимодействуют с веб-сервисами, получают данные в реальном времени или синхронизируют данные между устройствами. Для работы с сетью в Android можно использовать библиотеки, такие как Retrofit или Volley, которые упрощают выполнение сетевых запросов и обработку ответов.

Каждый из этих видов хранилищ данных в Android-приложениях предназначен для решения определенных задач и обеспечивает разработчикам гибкость в выборе подходящего метода хранения данных. Внутреннее хранилище защищает конфиденциальные данные, внешнее хранилище предоставляет доступ к общим файлам, SQLite обеспечивает мощные возможности для работы с реляционными данными, SharedPreferences удобны для хранения настроек, а сети позволяют взаимодействовать с удаленными серверами и веб-сервисами.

1. **Виды мобильных приложений - нативные, веб, гибридные. Сравнительная характеристика. Используемые технологии и языки программирования.**

Мобильные приложения можно классифицировать на три основные категории: нативные, веб и гибридные.

Нативные приложения разрабатываются специально для определенной платформы, такой как Android или iOS. Они пишутся с использованием языков и инструментов, предоставляемых платформой: для Android это Java или Kotlin с использованием Android Studio, а для iOS — Swift или Objective-C с использованием Xcode. Нативные приложения обладают наилучшей производительностью и доступом ко всем функциям устройства, таким как камера, GPS и другие. Они обеспечивают высокий уровень пользовательского опыта и возможность использования последних возможностей платформы. Однако разработка нативных приложений для каждой платформы требует отдельного кода и инструментов, что увеличивает время и стоимость разработки.

Веб-приложения — это приложения, которые работают в браузере и написаны с использованием веб-технологий, таких как HTML, CSS и JavaScript. Они могут быть доступны с любого устройства, имеющего веб-браузер, и не требуют установки. Веб-приложения легко обновляются и поддерживаются, поскольку обновления происходят на сервере, и пользователи сразу видят изменения. Основными преимуществами веб-приложений являются кроссплатформенность и простота в развертывании. Однако они ограничены в доступе к функциональности устройства и зависят от наличия интернет-соединения. Производительность и пользовательский опыт веб-приложений также могут быть ниже по сравнению с нативными приложениями.

Гибридные приложения сочетают в себе элементы нативных и веб-приложений. Они разрабатываются с использованием веб-технологий (HTML, CSS, JavaScript) и запускаются внутри нативного контейнера, который позволяет им работать на различных платформах. Гибридные приложения могут использовать фреймворки, такие как Apache Cordova, Ionic или React Native. Эти фреймворки позволяют разработчикам создавать приложения, которые выглядят и работают как нативные, при этом используя единый код для всех платформ. Гибридные приложения имеют доступ к функциям устройства через плагины и API, что делает их более функциональными, чем веб-приложения. Основными преимуществами гибридных приложений являются сокращение времени и затрат на разработку за счет использования общего кода для различных платформ. Однако производительность гибридных приложений может быть ниже, чем у нативных, и пользовательский опыт может немного уступать нативным приложениям.

1. **Библиотеки Python для обработки больших данных. Обработка данных, превышающих объем доступной памяти. Библиотека Dask.**

Python предоставляет несколько мощных библиотек для обработки больших данных, которые могут превышать объем доступной оперативной памяти. Основные библиотеки, которые часто используются для этой цели, включают Dask, Pandas, PySpark и другие.

Pandas — это одна из наиболее популярных библиотек для обработки данных в Python, предоставляющая высокоуровневые структуры данных и инструменты для анализа данных. Однако, Pandas работает в оперативной памяти, что делает ее неудобной для обработки больших данных, которые не помещаются в память.

Dask — это библиотека, предназначенная для обработки больших данных, которая расширяет возможности Pandas и других инструментов Python, позволяя работать с данными, превышающими объем доступной памяти. Dask позволяет выполнять параллельные вычисления на локальном компьютере или в распределенной системе. Основные концепции Dask включают использование Dask DataFrame и Dask Array, которые аналогичны структурам данных в Pandas и NumPy, но работают лениво и позволяют распределять вычисления на несколько ядер процессора или узлов в кластере.

Dask DataFrame работает подобно Pandas DataFrame, но разделяет данные на более мелкие части, которые могут обрабатываться независимо и параллельно. Это позволяет эффективно работать с данными, которые не помещаются в оперативную память целиком. Например, если у вас есть файл данных размером 100 ГБ, Dask может разделить его на более мелкие части и обрабатывать их параллельно, используя несколько процессоров или даже распределенный кластер.

Пример использования Dask DataFrame для работы с большими данными:

import dask.dataframe as dd

# Создание Dask DataFrame из большого CSV-файла

df = dd.read\_csv('large\_dataset.csv')

# Выполнение операций, аналогичных Pandas

df = df[df['column'] > 0] # Фильтрация данных

df = df.groupby('category').mean() # Группировка и агрегация

# Вычисление результатов

result = df.compute()

print(result)

Этот пример демонстрирует, как Dask позволяет работать с данными, превышающими объем памяти, и выполнять аналогичные Pandas операции.

Кроме Dask, PySpark также является мощным инструментом для обработки больших данных. PySpark предоставляет интерфейс к Apache Spark, который является распределенной вычислительной системой, способной обрабатывать большие объемы данных на кластере. PySpark использует ленивые вычисления и позволяет выполнять сложные операции, такие как фильтрация, группировка и агрегация, на больших наборах данных, распределенных по множеству узлов в кластере.

Обработка данных, превышающих объем доступной памяти, включает в себя несколько ключевых стратегий: использование ленивых вычислений, распределение данных и вычислений на несколько узлов, а также оптимизация использования памяти. Dask и PySpark предоставляют инструменты и методы для эффективного выполнения этих стратегий.

1. **Библиотеки Python для обработки данных. Numpy, pandas - основные понятия, возможности, назначение, примеры использования. Основные форматы структурированных данных.**

Python предоставляет мощные библиотеки для обработки данных, такие как Numpy и Pandas, которые широко используются для научных и аналитических задач.

Numpy, или Numerical Python, является фундаментальной библиотекой для численных вычислений в Python. Основным объектом в Numpy является многомерный массив, называемый ndarray. Numpy позволяет выполнять элементарные математические операции, линейную алгебру, преобразования Фурье и другие научные вычисления. Numpy используется для работы с числовыми данными, особенно когда требуется высокая производительность и работа с большими массивами данных. Пример использования Numpy включает создание массивов и выполнение операций над ними, таких как сложение, умножение и транспонирование матриц.

Pandas предоставляет высокоуровневые структуры данных для работы с табличными данными. Основными структурами данных в Pandas являются DataFrame и Series. DataFrame представляет собой двумерную таблицу данных с метками по строкам и столбцам, а Series — одномерный массив с метками. Pandas используется для манипуляции табличными данными, анализа данных, обработки отсутствующих значений, объединения и группировки данных, а также для выполнения статистических операций. Пример использования Pandas включает создание DataFrame из словаря, чтение данных из CSV-файла, фильтрацию данных, группировку данных и вычисление статистики.

Основные форматы структурированных данных включают CSV, Excel, JSON и SQL. CSV, или Comma-Separated Values, — один из самых распространенных форматов для хранения табличных данных. Он представляет собой текстовый файл, в котором значения отделены запятыми. Pandas легко читают и записывают данные в формате CSV. Excel — формат файлов, созданный для работы с электронными таблицами. Pandas поддерживает чтение и запись данных в формате Excel. JSON, или JavaScript Object Notation, — формат для обмена данными, который легко читается и пишется людьми и машинами. Pandas также поддерживают работу с JSON-файлами. SQL — это формат для хранения табличных данных в реляционных базах данных. Pandas предоставляет функции для чтения и записи данных из SQL баз данных с использованием библиотеки SQLAlchemy.