**Часть 1.**

**Ответы на вопросы экзамена по дисциплине "Базы данных"**

**1. Предпосылки появления баз данных и СУБД. Определение базы данных и СУБД. Общая характеристика моделей данных, лежащих в основе баз данных.**

**Предпосылки появления баз данных и СУБД**

До появления систем управления базами данных (СУБД) данные хранились в текстовых файлах или в специализированных файловых системах, где вся информация была организована в виде записей или строк. Однако такая организация данных не обеспечивала гибкости при их обработке и не позволяла эффективно управлять большими объемами информации.

Основные проблемы, которые привели к появлению СУБД:

* **Дублирование данных**: данные часто хранились в разных местах, что приводило к их избыточности и возможным противоречиям.
* **Сложность в организации доступа к данным**: старые системы не предоставляли удобных средств для поиска и извлечения данных.
* **Отсутствие целостности данных**: не существовало механизмов для обеспечения согласованности данных.
* **Медленная обработка информации**: в силу низкой эффективности старых систем обработки данных.

**Определение базы данных (БД) и Системы управления базами данных (СУБД)**

* **База данных (БД)** – это организованная совокупность данных, которые хранятся и обрабатываются с помощью СУБД. Базы данных могут содержать информацию, структурированную в виде таблиц, файлов, объектов или других моделей данных.
* **Система управления базами данных (СУБД)** – это программное обеспечение, которое позволяет создавать, хранить, управлять и использовать базы данных. СУБД обеспечивает средства для хранения данных, их обработки, выполнения запросов и поддержания целостности.

**Модели данных, лежащие в основе баз данных**

* **Иерархическая модель данных** – представляет данные в виде дерева, где каждый элемент имеет один родительский элемент, но может иметь несколько дочерних элементов.
* **Сетевая модель данных** – это расширение иерархической модели, где элементы могут быть связаны с несколькими родительскими и дочерними элементами.
* **Реляционная модель данных** – представляет данные в виде таблиц (отношений), где данные организованы по строкам и столбцам.
* **Объектно-ориентированная модель данных** – использует принципы объектно-ориентированного программирования для хранения и манипуляции данными.
* **Документно-ориентированная модель данных** – данные хранятся в виде документов (например, JSON, XML), которые могут иметь различные структуры.

**2. Базы данных и информационные системы. Банки данных.**

**Базы данных и информационные системы**

Информационная система – это совокупность аппаратных и программных средств, которые предназначены для сбора, хранения, обработки и передачи информации. Базы данных являются неотъемлемой частью большинства информационных систем, обеспечивая эффективное хранение и манипуляцию данными.

**Банки данных**

Банки данных (БД) – это специализированные базы данных, предназначенные для хранения больших объемов информации, которые могут быть использованы для аналитической обработки, извлечения данных, статистики и других целей. Банки данных часто используются в крупных организациях, научных учреждениях и правительственных органах.

Банки данных обеспечивают следующие функции:

* **Хранение больших объемов данных**: данные могут быть разнообразными, начиная от текстовых документов и заканчивая сложными графическими данными.
* **Поиск и извлечение информации**: в банке данных реализуются механизмы быстрого поиска и извлечения нужных данных.
* **Обеспечение целостности и безопасности данных**: банки данных обеспечивают механизмы контроля доступа и целостности данных.

**3. Модели данных. Характеристика иерархической модели данных. Достоинства и недостатки.**

**Иерархическая модель данных**

Иерархическая модель данных организует данные в виде дерева. Каждый элемент в такой модели (называемый узлом) имеет один родительский элемент, но может иметь несколько дочерних элементов. Это позволяет организовывать данные в виде структуры, подобной древу, где корень является начальной точкой, а листья — конечными элементами.

Пример иерархической модели: представьте организационную структуру компании, где директор является корнем дерева, а его подчиненные — дочерними элементами.

**Достоинства и недостатки иерархической модели**

**Достоинства**:

* **Простота представления данных**: структура данных легко воспринимается, особенно когда она естественным образом вписывается в иерархическую структуру.
* **Высокая производительность для простых запросов**: если запросы подразумевают последовательный доступ к данным, такая модель может быть эффективной.

**Недостатки**:

* **Ограниченность связей**: каждый элемент может иметь только один родительский элемент, что ограничивает гибкость структуры данных.
* **Сложность модификаций**: для изменения структуры иерархии (например, перемещения элемента в другое место) требуются значительные усилия, что может привести к трудностям при масштабировании и изменении данных.
* **Неэффективность при сложных запросах**: для сложных запросов, требующих доступа к данным на разных уровнях дерева, иерархическая модель может быть неэффективной.

**4. Модели данных. Характеристика сетевой модели данных. Достоинства и недостатки.**

Сетевая модель данных — это структура организации данных, в которой записи могут быть связаны между собой несколькими отношениями, образующими сеть. В отличие от иерархической модели, которая ограничивает связь между данными только в одном направлении (родитель — потомок), сетевая модель позволяет иметь более сложные связи, поддерживающие структуры "многие ко многим". Сетевая модель использует концепцию **графа**: записи данных представляют узлы, а связи между ними — рёбра графа.

**Принципы работы сетевой модели:**

1. **Записи (Records)**:
   * Запись — это основная единица данных в сетевой модели. Каждая запись может содержать различные поля, представляющие атрибуты сущности. Например, запись студента может включать такие поля, как имя, возраст, группа.
2. **Связи (Relationships)**:
   * Связи между записями создаются с помощью указателей (или ссылок). В сетевой модели связи могут быть **многие ко многим**, то есть одна запись может быть связана с несколькими другими записями и наоборот. Связи определяют, как данные в разных записях взаимодействуют друг с другом.
3. **Указатели (Pointers)**:
   * Сетевые модели используют указатели, которые служат для связи между записями. Это позволяет быстро перемещаться по записям, переходя от одной к другой, что является основным способом навигации.
4. **Навигация (Navigation)**:
   * Для получения информации из сетевой базы данных используется навигация по графу. Переход от одной записи к другой осуществляется по указателям, что требует знание структуры базы данных.

**Пример структуры базы данных в сетевой модели**

Предположим, у нас есть система, которая хранит информацию о **студентах**, **курсах** и **преподавателях**. Студент может быть записан на несколько курсов, а курс может быть связан с несколькими студентами. Преподаватель может вести несколько курсов.

* **Записи**:
  + Студент: СтудентID, Имя, Возраст, Группа.
  + Курс: КурсID, Название, Преподаватель.
  + Преподаватель: ПреподавательID, Имя.
* **Связи**:
  + Студент записан на курс.
  + Преподаватель ведет курс.

**Структура базы данных может быть представлена следующим образом:**

1. **Студенты** (Student):
   * StudentID → Идентификатор студента.
   * Name → Имя студента.
   * Age → Возраст студента.
   * Group → Группа, к которой принадлежит студент.
2. **Курсы** (Course):
   * CourseID → Идентификатор курса.
   * Name → Название курса.
   * TeacherID → Идентификатор преподавателя, который ведет курс.
3. **Преподаватели** (Teacher):
   * TeacherID → Идентификатор преподавателя.
   * Name → Имя преподавателя.
4. **Связи**:
   * Студент (Student) записан на курс (Course).
   * Преподаватель (Teacher) ведет курс (Course).

**Операции на данными в сетевой модели**

1. **Навигация (Navigation)**:
   * Операция перехода по графу от одного узла (записи) к другому. Например, можно перемещаться по указателям, чтобы найти все курсы, на которые записан студент, или все студенты, записанные на курс.
2. **Поиск (Search)**:
   * Операция поиска данных по заданному критерию. В сетевой модели поиск можно выполнить, следуя указателям и проверяя соответствующие атрибуты записей.

Пример: Найти всех студентов, записанных на курс "Математика".

1. **Добавление (Insert)**:
   * Операция добавления новой записи в базу данных или создания связи между записями. Например, можно добавить нового студента и создать связь между этим студентом и курсом, на который он записан.

Пример: Добавление нового студента и добавление связи между ним и курсом.

1. **Удаление (Delete)**:
   * Операция удаления записи или связи между записями. Например, если студент больше не записан на курс, удаляется связь между студентом и курсом.

Пример: Удаление записи о студенте, который больше не учится, и разрыв всех его связей с курсами.

1. **Обновление (Update)**:
   * Операция изменения данных в существующих записях или изменении связей между ними. Например, можно изменить имя студента или обновить информацию о курсе.

Пример: Обновление информации о преподавателе, который ведет курс.

**Достоинства и недостатки сетевой модели данных**

**Достоинства:**

1. **Гибкость в моделировании сложных связей**:
   * Сетевые модели идеально подходят для представления сложных взаимосвязанных данных, где требуется связь "многие ко многим" (например, студент может быть записан на несколько курсов, а курс может включать несколько студентов).
2. **Эффективность при больших объемах данных**:
   * Благодаря возможности переходить между записями через указатели, сетевые модели могут быть достаточно эффективными при работе с большими объемами данных, особенно когда нужно быстро находить связи.
3. **Поддержка сложных структур данных**:
   * Сетевые модели позволяют моделировать более сложные структуры данных, чем иерархическая модель. Например, одна запись может быть связана с несколькими другими записями в разных направлениях.

**Недостатки:**

1. **Сложность в реализации и поддержке**:
   * Сетевые модели требуют более сложной структуры данных и механизмов для навигации и манипуляции данными. Это делает их трудными для проектирования и поддержки по сравнению с реляционными моделями.
2. **Отсутствие стандартизации**:
   * В отличие от реляционной модели, в которой используется SQL, в сетевых моделях не существует общего стандарта для взаимодействия с данными. Это делает их использование менее гибким и универсальным.
3. **Трудности в изменении структуры данных**:
   * Изменение структуры сетевой базы данных (например, добавление новой связи или новой записи) может быть сложным и требовать изменений в множестве связанных записей и связей.
4. **Требование к навигации**:
   * Чтобы получить доступ к данным, необходимо знать структуру и связи между записями. Это может усложнить процесс работы с базой данных, особенно если она становится очень большой и сложной.

**Пример**

Предположим, у нас есть следующая информация:

* **Студенты**:
  + Иванов, 21, Группа A.
  + Петров, 22, Группа B.
* **Курсы**:
  + Курс "Математика", Преподаватель 1.
  + Курс "Программирование", Преподаватель 2.
* **Преподаватели**:
  + Преподаватель 1, Иванова.
  + Преподаватель 2, Петрова.

**Связи**:

1. Иванов и Петров записаны на курс "Математика".
2. Иванов записан на курс "Программирование".

**Заключение**

Сетевая модель данных была важным этапом в эволюции СУБД, предоставляя гибкость в описании сложных взаимосвязанных данных. Хотя эта модель имеет свои достоинства, такие как гибкость и эффективность при определенных сценариях, она уступила место реляционным моделям, которые обеспечивают более простое и стандартизированное представление данных.

**5. Модели данных. Характеристика объектно-ориентированной модели данных. Достоинства и недостатки.**

**Объектно-ориентированная модель данных**

Объектно-ориентированная модель данных основывается на принципах объектно-ориентированного программирования (ООП). В этой модели данные представляются как объекты, которые содержат как атрибуты (характеристики), так и методы (функции), которые могут манипулировать этими атрибутами.

Каждый объект в такой модели представляет собой экземпляр класса, где класс задает структуру данных и определяет доступные операции над ними. Это позволяет объединять данные и логику их обработки в одном месте.

**Достоинства объектно-ориентированной модели данных**

1. **Инкапсуляция**: Каждый объект изолирован от внешней среды, что позволяет скрыть детали реализации и облегчает управление данными.
2. **Наследование**: Возможность создания новых классов на основе существующих, что позволяет повторно использовать код и создавать гибкие структуры данных.
3. **Полиморфизм**: Позволяет объектам разного типа обрабатывать данные схожим образом, что упрощает проектирование системы.
4. **Моделирование сложных объектов**: Объектно-ориентированная модель идеально подходит для моделирования сложных и многосвязанных объектов, таких как графы, деревья и другие структуры.
5. **Управление большими объемами данных**: ООП позволяет легко управлять и обрабатывать большие объемы данных, поскольку данные и методы тесно связаны.

**Недостатки объектно-ориентированной модели данных**

1. **Сложность реализации**: В отличие от реляционных баз данных, объектно-ориентированная модель требует сложных механизмов для обеспечения устойчивости данных, что увеличивает сложность разработки СУБД.
2. **Отсутствие стандартов**: На момент появления объектно-ориентированных СУБД не существовало стандартов, что приводило к разнообразию реализаций и трудностям в совместимости данных.
3. **Программная сложность**: Разработка и поддержка объектно-ориентированных СУБД требует знания принципов ООП, что может быть трудным для разработчиков, не знакомых с этим подходом.

**6. Модели данных. Характеристика постреляционной модели данных. Достоинства и недостатки.**

**Постреляционная модель данных**

Постреляционная модель данных (Post-relational) представляет собой расширение традиционной реляционной модели. Эта модель была разработана для решения определенных ограничений реляционных СУБД, таких как работа с большими объемами данных или сложными структурами данных, которые не могут быть эффективно представлены в виде таблиц.

Постреляционные СУБД поддерживают новые типы данных, такие как мультимедийные файлы, документы или даже текстовые данные, что делает их более гибкими в работе с неструктурированной информацией. Сюда входят базы данных, поддерживающие графы, документы (например, XML, JSON), мультимедийные файлы и т. д.

**Достоинства постреляционной модели данных**

1. **Поддержка сложных типов данных**: Позволяет работать с данными, которые не могут быть легко представлены в реляционных таблицах (например, изображения, текст, видео).
2. **Гибкость**: Система может адаптироваться к различным типам данных и запросов, обеспечивая более универсальное представление.
3. **Меньше ограничений**: Модели, основанные на этой концепции, могут работать с данными, которые не требуют жесткой структуры и фиксированных схем.
4. **Простота использования для специализированных задач**: Особенно полезна для приложений, которые работают с большими объемами неструктурированных данных (например, в сфере мультимедиа или текстовых анализов).

**Недостатки постреляционной модели данных**

1. **Сложность в управлении**: Требует более сложных механизмов для хранения и обработки данных, чем традиционные реляционные модели.
2. **Отсутствие стандартов**: Постреляционные СУБД не всегда соответствуют стандартам, что делает их трудными для интеграции с другими системами.
3. **Производительность**: Для обработки сложных типов данных могут требоваться ресурсоемкие операции, что снижает производительность.

**7. Модели данных. Общая характеристика реляционной модели данных. Достоинства и недостатки.**

**Реляционная модель данных**

Реляционная модель данных — это одна из самых распространенных моделей, в основе которой лежит использование таблиц для хранения данных. В реляционной модели данные представляются в виде отношений (таблиц), которые состоят из строк (кортежей) и столбцов (атрибутов). Каждое отношение имеет уникальный идентификатор (первичный ключ) и может быть связано с другими отношениями с помощью внешних ключей.

Реляционные базы данных используют язык SQL (Structured Query Language) для выполнения запросов, обновлений и других операций с данными.

**Достоинства реляционной модели данных**

1. **Простота**: Реляционная модель представляет данные в виде таблиц, что легко воспринимается и подходит для большинства пользователей.
2. **Гибкость запросов**: Возможность выполнения сложных запросов с использованием языка SQL.
3. **Нормализация**: Возможность исключения избыточности данных и поддержания их целостности через нормализацию.
4. **Поддержка транзакций**: СУБД, основанные на реляционной модели, поддерживают транзакции, что гарантирует консистентность и целостность данных.
5. **Широкая поддержка**: Реляционные базы данных поддерживаются большинством СУБД и стандартов, что облегчает разработку приложений и их переносимость.

**Недостатки реляционной модели данных**

1. **Ограничения по структуре данных**: Реляционная модель подходит для работы с структурированными данными, но может быть неэффективной при работе с неструктурированными данными (например, текстами или мультимедийными файлами).
2. **Производительность на больших данных**: При очень больших объемах данных производительность может падать, особенно если данные плохо структурированы или требуется сложная обработка.
3. **Сложность для сложных объектов**: Моделирование сложных взаимосвязей, таких как графы или иерархии, может быть неудобным в реляционной модели.

**8. Реляционная модель данных. Понятия: отношения; заголовка, тела, схемы, степени и кардинального числа отношения; домена; атрибута; кортежа.**

**Определения в реляционной модели**

* **Отношение**: Таблица, состоящая из строк и столбцов, которая представляет собой структуру данных. Каждый элемент таблицы называется кортежем.
* **Заголовок отношения**: Множество всех атрибутов (столбцов), которые описывают структуру таблицы. Заголовок определяет, какие данные могут быть в каждом столбце.
* **Тело отношения**: Множество всех кортежей (строк) в таблице, представляющее конкретные данные.
* **Схема отношения**: Определение структуры отношения, включающее заголовок (список атрибутов) и их типы данных.
* **Степень отношения**: Количество атрибутов (столбцов) в отношении. Например, если таблица содержит 5 столбцов, ее степень равна 5.
* **Кардинальное число отношения**: Количество кортежей (строк) в таблице.
* **Домен**: Множество всех возможных значений для конкретного атрибута. Например, для атрибута "Возраст" домен может быть числовым типом данных, который ограничивает значения от 0 до 120.
* **Атрибут**: Столбец в таблице, который описывает характеристику данных (например, "Имя", "Возраст").
* **Кортеж**: Строка в таблице, которая представляет собой запись данных для одной единицы (например, одного студента или одного товара).

**9. Реляционная модель данных. Понятия: потенциального, простого, составного, первичного, альтернативного и внешнего ключа отношения.**

**Ключи в реляционной модели данных**

* **Потенциальный ключ**: Это минимальный набор атрибутов, который может быть использован для однозначной идентификации каждой строки (кортежа) в таблице. В таблице может быть несколько потенциальных ключей. Например, если у нас есть таблица сотрудников, потенциальными ключами могут быть «номер паспорта» или «комбинация имени и даты рождения», если эти значения уникальны для каждого сотрудника.
* **Простой ключ**: Это ключ, который состоит из одного атрибута. Например, в таблице студентов ключом может быть уникальный «студенческий номер». Это простой ключ, потому что он состоит из одного атрибута.
* **Составной ключ**: Это ключ, который состоит из нескольких атрибутов. Например, если таблица заказов имеет атрибуты «номер заказа» и «номер клиента», для идентификации уникальной строки может использоваться составной ключ, состоящий из этих двух атрибутов.
* **Первичный ключ**: Это один из потенциальных ключей, который выбран для уникальной идентификации строк в таблице. Первичный ключ должен быть уникальным для каждой строки и не может содержать значений NULL. Например, в таблице сотрудников первичным ключом может быть «ID сотрудника».
* **Альтернативный ключ**: Это все остальные потенциальные ключи, кроме выбранного первичного. Если в таблице для уникальной идентификации строк существует несколько способов, то все кроме основного ключа называют альтернативными. Например, если «ID сотрудника» выбран в качестве первичного ключа, то «номер паспорта» будет альтернативным ключом.
* **Внешний ключ**: Это атрибут или группа атрибутов в одной таблице, которые ссылаются на первичный ключ другой таблицы. Внешний ключ обеспечивает ссылочную целостность данных между таблицами. Например, если есть таблица «Студенты» и таблица «Курсы», то внешний ключ «код курса» в таблице «Студенты» может ссылаться на первичный ключ «код курса» в таблице «Курсы».

**10. Реляционная модель данных. Свойства, которыми обладает реляционное отношение. Первая нормальная форма.**

**Свойства реляционного отношения**

Реляционные отношения должны соответствовать следующим свойствам:

1. **Каждый элемент отношения уникален**: Каждая строка (кортеж) в таблице должна быть уникальной, то есть не может быть одинаковых строк.
2. **Отсутствие повторяющихся атрибутов**: Каждый атрибут в таблице должен быть атомарным, то есть не может содержать несколько значений.
3. **Каждое поле содержит единственное значение**: В таблице не должно быть полей с множественными значениями (многозначность).
4. **Упорядочивание строк и столбцов не имеет значения**: Порядок строк и столбцов в таблице не имеет значения для логического представления данных.
5. **Каждое значение атрибута принадлежит одному и тому же домену**: Все значения в одном столбце должны быть одного типа данных.

**Первая нормальная форма (1НФ)**

Первая нормальная форма (1НФ) предполагает, что:

1. **Атомарность данных**: Все значения в таблице должны быть атомарными, то есть не делиться на более мелкие части. Например, если у нас есть таблица студентов и их адресов, то поле «Адрес» не должно содержать несколько значений (например, «ул. Ленина, 23, кв. 5»).
2. **Уникальность строк**: Каждая строка (кортеж) в таблице должна быть уникальной. Для этого обычно используется первичный ключ.

В первой нормальной форме таблица не должна содержать повторяющихся групп данных. Если столбцы содержат составные или многозначные значения, то таблица не находится в 1НФ.

Пример:

Если у нас есть таблица, в которой один студент может иметь несколько телефонов, то для приведения таблицы к 1НФ, нужно разделить информацию о телефонах в отдельные строки или создать отдельную таблицу для телефонов.

**11. Реляционная модель данных. Понятие целостности. Виды целостности. Механизмы обеспечения целостности.**

**Понятие целостности данных**

Целостность данных означает, что данные в базе данных остаются точными, полными и непротиворечивыми. Это важный аспект, который обеспечивает правильность, актуальность и непротиворечивость данных.

**Виды целостности данных**

1. **Целостность сущностей (Entity Integrity)**: Это правило, которое гласит, что каждая строка в таблице должна быть уникальной, а для каждой строки должен быть определен первичный ключ. Никакая строка не может иметь значение NULL в поле первичного ключа.
2. **Ссылочная целостность (Referential Integrity)**: Это правило, которое гарантирует, что ссылки между таблицами (внешние ключи) всегда корректны. Например, если внешний ключ в одной таблице ссылается на строку в другой таблице, то эта строка должна существовать. Нельзя создать запись, которая ссылается на несуществующую запись.
3. **Целостность доменов (Domain Integrity)**: Каждое поле в таблице должно содержать данные, соответствующие заранее определенному типу данных (домену). Например, если столбец содержит «возраст», то он должен содержать только числовые значения.

**Механизмы обеспечения целостности**

1. **Ограничения целостности (Constraints)**: Ограничения позволяют СУБД проверять целостность данных. Это могут быть ограничения на уникальность, наличие значения, допустимые диапазоны значений и т.д.

Примеры:

* + **NOT NULL** – столбец не может содержать значения NULL.
  + **UNIQUE** – значения в столбце должны быть уникальными.
  + **CHECK** – проверка значения столбца на соответствие заданному условию.

1. **Триггеры**: Триггеры — это механизмы, которые выполняются автоматически при определенных действиях (например, вставка, обновление, удаление данных). Триггеры могут проверять целостность данных и автоматически исправлять ошибки или предотвращать нарушение целостности.
2. **Процедуры обновления данных (cascade delete, cascade update)**: Эти механизмы поддерживают целостность данных, автоматически обновляя или удаляя связанные данные в других таблицах, когда данные в основной таблице изменяются или удаляются.

**12. Типы связей между реляционными таблицами. Обеспечение целостности данных при операциях с данными таблиц базы данных.**

**Типы связей между реляционными таблицами**

В реляционной базе данных между таблицами могут быть следующие типы связей:

1. **Один к одному (1:1)**: В этой связи каждая строка в одной таблице может быть связана только с одной строкой в другой таблице, и наоборот. Этот тип связи редко используется, но может применяться для разделения данных, которые не всегда должны быть одновременно видны в одной таблице.

Пример: Таблицы «Пользователь» и «Паспорт» могут иметь связь 1:1, если каждому пользователю соответствует только один паспорт.

1. **Один ко многим (1:N)**: В этой связи одна строка в первой таблице может быть связана с несколькими строками во второй таблице, но каждая строка во второй таблице может быть связана только с одной строкой в первой таблице.

Пример: Один заказ может содержать несколько товаров, но каждый товар может быть в одном заказе.

1. **Многие ко многим (M:N)**: В этой связи строки в обеих таблицах могут быть связаны между собой несколькими отношениями. Для реализации этой связи обычно создается третья таблица, которая будет хранить связи между первичными ключами двух других таблиц.

Пример: Студенты и курсы могут быть связаны через третью таблицу «Записи», где каждый студент может быть записан на несколько курсов, а каждый курс может содержать несколько студентов.

**Обеспечение целостности данных при операциях с данными**

Чтобы поддерживать целостность данных при операциях с таблицами, СУБД использует такие механизмы, как:

1. **Ограничения целостности (Integrity Constraints)**: Они проверяют, что изменения в таблицах (например, вставка, обновление или удаление строк) не нарушают целостность данных. Например, при удалении записи в родительской таблице может быть запрещено удаление записей в дочерней таблице, если на них ссылаются внешние ключи.
2. **Каскадное удаление и обновление (Cascade Delete/Update)**: Эти механизмы автоматически удаляют или обновляют связанные записи в других таблицах. Например, если удаляется запись в родительской таблице, то каскадное удаление также удаляет все связанные записи в дочерней таблице.
3. **Ограничения на внешние ключи**: СУБД может запрещать вставку или обновление данных в дочерней таблице, если соответствующая запись в родительской таблице не существует (обеспечивая тем самым ссылочную целостность).

**13. Общая характеристика языков для выполнения операций над реляционными отношениями. Реляционная алгебра.**

**Языки для выполнения операций над реляционными отношениями**

В реляционных базах данных для выполнения операций над данными используются два основных типа языков:

1. **Язык запросов (например, SQL)**: Это декларативный язык, который позволяет пользователю запросить данные без указания, как именно они должны быть получены. В SQL можно выполнять операции выбора, вставки, обновления и удаления данных, а также создавать и изменять структуру таблиц. Язык SQL является основным для работы с реляционными базами данных.
2. **Реляционная алгебра**: Это формальный язык, который описывает операции над реляционными отношениями с помощью математических операций. В отличие от SQL, реляционная алгебра больше ориентирована на математические теории и служит для описания и выполнения запросов в реляционных СУБД.

Реляционная алгебра состоит из набора операторов, которые выполняются над отношениями, и возвращают новые отношения.

**Реляционная алгебра**

Реляционная алгебра включает в себя несколько базовых операций, которые можно комбинировать для выполнения более сложных запросов:

1. **Селекция (σ)**: Операция, которая выбирает строки (кортежи) из отношения, удовлетворяющие заданному условию. Это аналог SQL-оператора SELECT с условием WHERE.

Пример: σage>18(students)\sigma\_{\text{age} > 18}(\text{students})σage>18​(students) — выбирает всех студентов старше 18 лет.

1. **Проекция (π)**: Операция, которая выбирает столбцы (атрибуты) из отношения, создавая новое отношение, которое содержит только указанные атрибуты.

Пример: πname, age(students)\pi\_{\text{name, age}}(\text{students})πname, age​(students) — выбирает только имена и возраст студентов.

1. **Объединение (∪)**: Операция, которая объединяет два отношения, при этом исключает дубликаты. Операция объединения требует, чтобы два отношения имели одинаковую схему.

Пример: students∪graduates\text{students} \cup \text{graduates}students∪graduates — объединяет таблицы студентов и выпускников.

1. **Разность (−)**: Операция, которая возвращает строки из первого отношения, которых нет во втором отношении.

Пример: students−graduates\text{students} - \text{graduates}students−graduates — выбирает студентов, которые не являются выпускниками.

1. **Пересечение (∩)**: Операция, которая возвращает только те строки, которые присутствуют в обоих отношениях.

Пример: students∩graduates\text{students} \cap \text{graduates}students∩graduates — выбирает тех студентов, которые также являются выпускниками.

1. **Декартово произведение (×)**: Операция, которая создает новое отношение путем комбинирования каждой строки одного отношения с каждой строкой другого отношения. Это аналог SQL-операции JOIN, но без условия соединения.

Пример: students×courses\text{students} \times \text{courses}students×courses — возвращает все возможные комбинации студентов и курсов.

1. **Соединение (⨝)**: Операция, которая соединяет два отношения по общему атрибуту, аналог SQL-операции JOIN. Есть несколько типов соединений: внутреннее соединение, внешнее соединение и т.д.

Пример: \text{students} \bowtie\_{\text{student\_id} = \text{id}} \text{courses} — соединяет таблицы студентов и курсов по атрибуту student\_id.

**14. Унарные и бинарные операции реляционной алгебры. Характеристика и примеры операций объединения, разности (вычитания), пересечения, декартового (прямого) произведения.**

**Унарные операции**

Унарные операции в реляционной алгебре выполняются над одним отношением. Основными унарными операциями являются:

1. **Селекция (σ)**: Выбор строк, которые соответствуют определенному условию.

Пример: σage>18(students)\sigma\_{\text{age} > 18}(\text{students})σage>18​(students) — выбор студентов старше 18 лет.

1. **Проекция (π)**: Выбор столбцов из отношения.

Пример: πname, age(students)\pi\_{\text{name, age}}(\text{students})πname, age​(students) — выбор только столбцов name и age из таблицы студентов.

**Бинарные операции**

Бинарные операции в реляционной алгебре выполняются между двумя отношениями. К ним относятся:

1. **Объединение (∪)**: Операция, которая возвращает объединение двух отношений, исключая дубликаты. Для выполнения объединения отношения должны иметь одинаковую схему (структуру).

Пример: Если есть два отношения студентов из разных групп, то объединение двух таких таблиц вернет всех студентов без повторений.

1. **Разность (−)**: Операция, которая возвращает строки из первого отношения, которые отсутствуют во втором.

Пример: students−graduates\text{students} - \text{graduates}students−graduates — возвращает студентов, которые не являются выпускниками.

1. **Пересечение (∩)**: Операция, которая возвращает строки, которые присутствуют и в первом, и во втором отношении.

Пример: students∩graduates\text{students} \cap \text{graduates}students∩graduates — возвращает студентов, которые одновременно являются выпускниками.

1. **Декартово произведение (×)**: Операция, которая создает все возможные комбинации строк из двух отношений. Это создаст новые строки, объединяя все данные из обеих таблиц.

Пример: students×courses\text{students} \times \text{courses}students×courses — создает новое отношение, где каждой строке студентов соответствует каждая строка курсов.

**15. Унарные и бинарные операции реляционной алгебры. Характеристика и примеры операций выборки (селекции), проекции, деления, соединения.**

**Операции выборки и проекции**

1. **Селекция (σ)**: Эта операция используется для выбора строк, которые удовлетворяют заданному условию.

Пример: σage>18(students)\sigma\_{\text{age} > 18}(\text{students})σage>18​(students) — выберет всех студентов старше 18 лет.

1. **Проекция (π)**: Операция, которая выбирает только определенные столбцы из отношения.

Пример: πname, age(students)\pi\_{\text{name, age}}(\text{students})πname, age​(students) — вернет только имена и возраст студентов.

**Операции деления и соединения**

1. **Деление (÷)**: Операция деления используется для нахождения всех строк из одного отношения, которые связаны с каждой строкой из другого отношения. Она часто используется для задач, связанных с отношением «каждый».

Пример: Если таблица студентов записана на курсы, а другая таблица содержит курсы, деление вернет всех студентов, которые записаны на все курсы.

1. **Соединение (⨝)**: Эта операция используется для соединения двух отношений по общему атрибуту. Она возвращает новое отношение, объединяя столбцы двух таблиц, где значения общих атрибутов совпадают.

Пример: \text{students} \bowtie\_{\text{student\_id} = \text{id}} \text{courses} — соединяет таблицы студентов и курсов по идентификатору студента.

**16. Общая характеристика реляционного исчисления. Реляционное исчисление доменов и реляционное исчисление кортежей.**

Реляционное исчисление — это формальная система для описания запросов к данным в реляционных базах данных, основанная на логике предикатов. Оно используется для выражения запросов в реляционных системах управления базами данных (СУБД), таких как SQL. Реляционное исчисление работает с логическими выражениями и предикатами, а его цель — извлечение данных, удовлетворяющих определенным условиям.

**Логика предикатов** является основой для реляционного исчисления. Это расширение логики высказываний, где переменные могут принимать значения, и логика работает с предикатами (утверждениями), которые могут быть истинными или ложными в зависимости от значений переменных. Предикаты выражают отношения между объектами (например, "студент является отличником", "работник работает в отделе"), а также могут быть использованы для выборки данных из базы.

**Основные определения логики предикатов:**

1. **Предикат** — это логическое выражение, которое может быть истинным или ложным в зависимости от значений его аргументов. Например, предикат "клиент имеет заказ" может быть истинным для некоторых клиентов и ложным для других.
   * Пример: P(x):P(x):P(x): "x является студентом". Это предикат, который может быть истинным или ложным для конкретных значений переменной xxx.
2. **Переменная** — это символ, который принимает значение в некотором домене. В контексте реляционных баз данных переменные часто представляют собой строки или столбцы в таблицах.
3. **Квантор** — символ, который указывает на количество экземпляров переменной:
   * **Квантор существования (∃\exists∃)**: "Существует хотя бы одно значение переменной, для которого выражение истинно."
   * **Квантор всеобщности (∀\forall∀)**: "Для всех значений переменной выражение истинно."
4. **Логические операторы**:
   * **Конъюнкция (∧\land∧)** — логическое "И".
   * **Дизъюнкция (∨\lor∨)** — логическое "ИЛИ".
   * **Отрицание (¬\neg¬)** — логическое "НЕ".
   * **Импликация (→\rightarrow→)** — логическое "ЕСЛИ, ТО".
   * **Эквиваленция (↔\leftrightarrow↔)** — логическое "Тогда и только тогда".
5. **Формула** — это логическое выражение, которое включает предикаты, переменные, кванторы и логические операторы. Формулы используются для выражения запросов и условий на данные.

Реляционное исчисление делится на два вида: **реляционное исчисление доменов** и **реляционное исчисление кортежей**.

**Реляционное исчисление доменов (Domain Relational Calculus, DRC)**

Реляционное исчисление доменов ориентировано на работу с отдельными значениями (доменами) данных. В нем запросы формулируются в виде формул, где переменные представляют собой домены значений (например, строки или числа в таблице).

**Основные характеристики**:

* В реляционном исчислении доменов переменные представляют отдельные значения, такие как конкретные атрибуты (столбцы).
* Формулируются запросы с использованием кванторов существования и всеобщности.
* Запрос состоит из предикатов, которые должны быть истинными для значений в таблицах.

**Пример**:

Предположим, у нас есть таблица студентов:

| **СтудентID** | **Имя** | **Возраст** | **Группа** |
| --- | --- | --- | --- |
| 1 | Иванов | 20 | A |
| 2 | Петров | 21 | B |
| 3 | Сидоров | 22 | A |

Задание: "Найти студентов, возраст которых больше 20".

Запрос в реляционном исчислении доменов:

{s.Имя∣∃s.(Студент(s)∧s.Возраст>20)}\{ s.\text{Имя} \mid \exists s. (\text{Студент}(s) \land s.\text{Возраст} > 20) \}{s.Имя∣∃s.(Студент(s)∧s.Возраст>20)}

Здесь:

* s.Имяs.\text{Имя}s.Имя — переменная, которая принимает значения из столбца "Имя".
* Студент(s)\text{Студент}(s)Студент(s) — предикат, который указывает, что переменная sss является строкой в таблице "Студенты".
* s.Возраст>20s.\text{Возраст} > 20s.Возраст>20 — условие, которое проверяет возраст студентов.

Этот запрос возвращает имена студентов, чей возраст больше 20.

**Реляционное исчисление кортежей (Tuple Relational Calculus, TRC)**

Реляционное исчисление кортежей ориентировано на работу с кортежами (строками) в таблицах. В нем переменные представляют собой кортежи, которые соответствуют строкам в таблицах базы данных.

**Основные характеристики**:

* В реляционном исчислении кортежей переменные представляют собой кортежи, то есть строки в таблицах.
* Запросы формулируются с использованием кванторов существования и всеобщности, а также предикатов, которые проверяют свойства целых кортежей.
* Запросы могут возвращать целые строки (кортежи).

**Пример**:

Используем ту же таблицу студентов:

| **СтудентID** | **Имя** | **Возраст** | **Группа** |
| --- | --- | --- | --- |
| 1 | Иванов | 20 | A |
| 2 | Петров | 21 | B |
| 3 | Сидоров | 22 | A |

Задание: "Найти всех студентов из группы A".

Запрос в реляционном исчислении кортежей:

{t∣Студент(t)∧t.Группа=′A′}\{ t \mid \text{Студент}(t) \land t.\text{Группа} = 'A' \}{t∣Студент(t)∧t.Группа=′A′}

Здесь:

* ttt — переменная, представляющая кортеж.
* Студент(t)\text{Студент}(t)Студент(t) — предикат, который указывает, что переменная ttt является строкой в таблице "Студенты".
* t.Группа=′A′t.\text{Группа} = 'A't.Группа=′A′ — условие, проверяющее, что значение в столбце "Группа" равно 'A'.

Этот запрос вернет все строки (кортежи) из таблицы студентов, где группа равна 'A'.

**Сравнение реляционного исчисления доменов и реляционного исчисления кортежей**

| **Характеристика** | **Реляционное исчисление доменов (DRC)** | **Реляционное исчисление кортежей (TRC)** |
| --- | --- | --- |
| **Представление данных** | Работает с отдельными значениями (доменами) данных. | Работает с кортежами (строками) в таблице. |
| **Переменные** | Переменные представляют отдельные значения атрибутов. | Переменные представляют собой кортежи (строки). |
| **Тип запроса** | Запросы формулируются через домены значений. | Запросы формулируются через строки (кортежи) таблиц. |
| **Пример** | Найти все имена студентов старше 20 лет. | Найти все строки студентов из группы A. |
| **Использование в практике** | Реже используется в практических СУБД. | Чаще используется в СУБД, поскольку запросы с кортежами более естественны для реляционных БД. |

**Заключение**

* **Реляционное исчисление доменов** ориентировано на работу с отдельными значениями атрибутов, в то время как **реляционное исчисление кортежей** работает с целыми строками в таблицах.
* Оба типа исчислений используют логику предикатов для формулирования запросов и включают кванторы существования и всеобщности для ограничения результатов.
* В реляционных системах чаще используется реляционное исчисление кортежей, так как оно напрямую работает с кортежами (строками) таблиц, что более интуитивно при выполнении запросов.

Эти формы логики используются для выражения запросов, аналогичных SQL-запросам, и важны для понимания теоретической основы работы реляционных баз данных.

**17. Процесс проектирования базы данных. Общая характеристика этапов проектирования.**  
Проектирование базы данных — это процесс создания эффективной и надежной системы для хранения и управления данными. Процесс включает несколько этапов:

1. **Анализ требований**: на этом этапе анализируются потребности пользователей и цели системы. Определяется, какие данные будут храниться, как они будут использоваться, кто будет работать с базой данных.
2. **Концептуальное проектирование**: создается концептуальная модель данных, которая описывает структуру данных без привязки к конкретной СУБД. В этом процессе часто используется метод ER-диаграмм.
3. **Логическое проектирование**: на этом этапе концептуальная модель преобразуется в логическую модель, которая более детализирована, но все еще независима от конкретной СУБД. Логическая модель обычно представляется в виде реляционных таблиц.
4. **Физическое проектирование**: определяется, как данные будут храниться физически на диске, выбираются индексы, оптимизируются запросы, планируется распределение данных.
5. **Тестирование и оптимизация**: после реализации базы данных проверяется, насколько она эффективна и соответствует требованиям, после чего выполняется оптимизация для повышения производительности.

**18. Концептуальное проектирование баз данных. Метод ER диаграмм.**  
Концептуальное проектирование — это процесс создания модели данных, которая отображает структуру информации и связи между сущностями в предметной области.  
Метод ER-диаграмм (Entity-Relationship Diagram) используется для визуального представления концептуальной модели. ER-диаграмма включает следующие элементы:

* **Сущности** (Entities): объекты, для которых нужно хранить информацию (например, "Студент", "Курс").
* **Атрибуты** (Attributes): характеристики сущностей (например, "Имя", "Дата рождения").
* **Связи** (Relationships): отношения между сущностями (например, "Записывается на", связывающая "Студента" и "Курс").
* **Кардинальность** связей: определяет, сколько экземпляров одной сущности может быть связано с экземплярами другой сущности (например, один студент может записаться на несколько курсов).

**19. Правила преобразования ER-модели в логическую реляционную модель.**  
Правила преобразования ER-модели в логическую реляционную модель

В процессе преобразования ER-модели в реляционную модель важно соблюдать несколько правил для корректного отображения связей, атрибутов и сущностей. Основные этапы преобразования описаны ниже:

1. Сущности в ER-модели преобразуются в таблицы.

Каждая сущность, представляемая прямоугольником в ER-диаграмме, становится таблицей в реляционной модели. Например, сущность "Студент" с атрибутами "Имя", "Дата рождения" и "Номер зачетки" будет преобразована в таблицу, где каждый атрибут становится столбцом.

2. Атрибуты сущностей становятся столбцами таблиц.

Каждый атрибут, связанный с сущностью, становится столбцом в таблице, которая отображает эту сущность. Для атрибутов с несколькими значениями создается отдельная таблица (например, если у студента несколько телефонов, то для них создается отдельная таблица, связанная с основной таблицей "Студент").

3. Связи между сущностями преобразуются в связи между таблицами.

- Связь один-к-одному (1:1): если между двумя сущностями существует связь один к одному, то в реляционной модели одна таблица будет содержать внешний ключ, ссылающийся на другую таблицу. Можно также создать отдельную таблицу для связи, если она имеет атрибуты.

- Связь один ко многим (1:N): если между сущностями существует связь один ко многим, то в таблице, представляющей сущность "многие", добавляется внешний ключ, ссылающийся на таблицу "один".

- Связь многие ко многим (M:N): для связи многие ко многим создается отдельная промежуточная таблица, которая содержит два внешних ключа — по одному от каждой из таблиц, участвующих в связи.

4. Кардинальность связей.

Кардинальность связи указывает, сколько экземпляров одной сущности может быть связано с экземплярами другой. Например, для связи "Многие ко многим" необходимо создать промежуточную таблицу, которая будет содержать два внешних ключа (например, "Студент ID" и "Курс ID").

5. Атрибуты, не относящиеся к сущности, преобразуются в отдельные таблицы.

Если атрибут является многозначным или связан с несколькими сущностями, то его необходимо преобразовать в отдельную таблицу с внешними ключами для связи с основными таблицами.

6. Композитные атрибуты.

Если атрибут состоит из нескольких податрибутов (например, адрес, состоящий из улицы, города, почтового индекса), то эти податрибуты становятся отдельными столбцами в таблице.

7. Идентификаторы (первичные ключи).

В реляционной модели необходимо определить первичный ключ для каждой таблицы. Он должен быть уникальным и использоваться для идентификации записей в таблице.

В дополнение к этим правилам важно учесть, что процесс преобразования должен учитывать также требования целостности данных и нормализации для минимизации избыточности и обеспечения согласованности данных в реляционной модели.

**20. Нормализация реляционных таблиц. Аномалии. Нормальные формы.**

Нормализация данных в реляционных базах данных — это процесс приведения данных в такую структуру, чтобы минимизировать избыточность и избежать различных аномалий. Аномалии могут возникать при работе с не нормализованными таблицами, и их можно разделить на три основные категории: **аномалии вставки**, **аномалии удаления** и **аномалии обновления**. Рассмотрим примеры каждой из этих аномалий.

**Примеры аномалий:**

1. **Аномалия вставки (Insert Anomaly)**

Эта аномалия возникает, когда невозможно добавить новые данные в таблицу без необходимости добавления избыточной информации.

**Пример:** Рассмотрим таблицу заказов:

| **Заказ ID** | **Клиент** | **Товар** | **Количество** | **Цена** |
| --- | --- | --- | --- | --- |
| 1 | Иванов | Товар 1 | 2 | 100 |
| 2 | Петров | Товар 2 | 1 | 200 |

Если новый клиент (например, "Сидоров") еще не сделал заказ, мы не можем записать информацию о нем в таблицу, так как для этого нужно указать хотя бы один товар. Это создает проблему вставки данных о клиенте без привязки к заказам, что приводит к избыточности.

1. **Аномалия удаления (Delete Anomaly)**

Аномалия удаления возникает, когда удаление данных приводит к потере других важных данных, которые не должны быть удалены.

**Пример:** В той же таблице заказов, если мы удалим заказ с ID = 1, то потеряем информацию о клиенте "Иванов", даже если этот клиент может сделать заказы в будущем.

| **Заказ ID** | **Клиент** | **Товар** | **Количество** | **Цена** |
| --- | --- | --- | --- | --- |
| 2 | Петров | Товар 2 | 1 | 200 |

В результате удаления данных о заказе клиента "Иванова", мы также теряем информацию о самом клиенте. Это создаёт проблему, если клиент делал несколько заказов.

1. **Аномалия обновления (Update Anomaly)**

Эта аномалия возникает, когда одно изменение требует обновления нескольких строк таблицы, что повышает вероятность ошибок и несогласованности данных.

**Пример:** В таблице заказов представлена информация о товаре и его цене. Если цена товара изменяется, например, для "Товар 1", то это изменение нужно выполнить в каждой строке, где упоминается этот товар. Это повышает вероятность ошибок.

| **Заказ ID** | **Клиент** | **Товар** | **Количество** | **Цена** |
| --- | --- | --- | --- | --- |
| 1 | Иванов | Товар 1 | 2 | 100 |
| 2 | Петров | Товар 1 | 1 | 100 |

Если цена товара изменится на 120, необходимо обновить каждую строку, что может привести к несогласованности данных, если обновление будет выполнено неполностью.

Нормализация данных и переход к нормальным формам (1НФ, 2НФ, 3НФ) помогает минимизировать эти аномалии и повысить целостность базы данных.

**21. Первая нормальная форма (1НФ)**

**Требование:** Все атрибуты должны содержать только атомарные (неделимые) значения. Это означает, что в одной ячейке не должно быть множества значений.

**Пример 1:**

**Нормализация таблицы:**

| **Заказ ID** | **Клиент** | **Товары** |
| --- | --- | --- |
| 1 | Иванов | Товар1, Товар2 |
| 2 | Петров | Товар3 |
| 3 | Сидоров | Товар4, Товар5, Товар6 |

Таблица нарушает 1НФ, так как в столбце "Товары" содержатся несколько значений для каждого заказа. Чтобы привести таблицу к первой нормальной форме, нужно разбить эти значения на отдельные строки.

**Таблица после нормализации:**

| **Заказ ID** | **Клиент** | **Товар** |
| --- | --- | --- |
| 1 | Иванов | Товар1 |
| 1 | Иванов | Товар2 |
| 2 | Петров | Товар3 |
| 3 | Сидоров | Товар4 |
| 3 | Сидоров | Товар5 |
| 3 | Сидоров | Товар6 |

Теперь каждый атрибут таблицы является атомарным (неделимым), и таблица приведена к 1НФ.

**22. Вторая нормальная форма (2НФ)**

**Требование:** Таблица должна находиться в 1НФ и не содержать частичных зависимостей, т.е. все атрибуты должны зависеть от всего составного первичного ключа.

**Пример 2:**

**Нормализация таблицы:**

| **Заказ ID** | **Товар ID** | **Название товара** | **Цена** |
| --- | --- | --- | --- |
| 1 | 101 | Товар A | 100 |
| 1 | 102 | Товар B | 150 |
| 2 | 101 | Товар A | 100 |
| 2 | 103 | Товар C | 200 |

В таблице первичный ключ состоит из двух атрибутов: "Заказ ID" и "Товар ID". Однако атрибуты "Название товара" и "Цена" зависят только от "Товар ID", а не от всего составного ключа. Это является частичной зависимостью и нарушает 2НФ.

**Таблица после нормализации:**

1. **Таблица заказов:**

| **Заказ ID** | **Товар ID** |
| --- | --- |
| 1 | 101 |
| 1 | 102 |
| 2 | 101 |
| 2 | 103 |

1. **Таблица товаров:**

| **Товар ID** | **Название товара** | **Цена** |
| --- | --- | --- |
| 101 | Товар A | 100 |
| 102 | Товар B | 150 |
| 103 | Товар C | 200 |

Теперь "Название товара" и "Цена" зависят только от "Товар ID", и таблицы приведены ко второй нормальной форме.

**23. Третья нормальная форма (3НФ)**

**Требование:** Таблица должна быть в 2НФ и не должна содержать транзитивных зависимостей. То есть, атрибуты должны зависеть только от первичного ключа, а не от других неключевых атрибутов.

**Пример 3:**

**Нормализация таблицы:**

| **Студент ID** | **Курс ID** | **Преподаватель** | **Факультет** |
| --- | --- | --- | --- |
| 1 | 101 | Иванов | Математика |
| 2 | 102 | Петров | Информатика |
| 3 | 103 | Сидоров | Физика |

В этой таблице мы видим, что информация о факультете зависит от "Преподавателя", а не от "Студент ID" или "Курс ID". Таким образом, существует транзитивная зависимость: "Студент ID" → "Курс ID" → "Преподаватель" → "Факультет". Эта транзитивная зависимость нарушает 3НФ.

**Таблица после нормализации:**

1. **Таблица студентов и курсов:**

| **Студент ID** | **Курс ID** |
| --- | --- |
| 1 | 101 |
| 2 | 102 |
| 3 | 103 |

1. **Таблица преподавателей:**

| **Преподаватель** | **Факультет** |
| --- | --- |
| Иванов | Математика |
| Петров | Информатика |
| Сидоров | Физика |

Теперь таблица не содержит транзитивных зависимостей и приведена к третьей нормальной форме.

**Дополнительные примеры нормализации**

**Пример 4 (1НФ → 2НФ → 3НФ):**

1. **Таблица до нормализации (смешанная):**

| **Заказ ID** | **Товар ID** | **Название товара** | **Цена** | **Количество** | **Дата** |
| --- | --- | --- | --- | --- | --- |
| 1 | 101 | Товар A | 100 | 2 | 2024-01-01 |
| 1 | 102 | Товар B | 150 | 1 | 2024-01-01 |
| 2 | 101 | Товар A | 100 | 3 | 2024-01-02 |

1. **Таблица после нормализации (1НФ → 2НФ → 3НФ):**
   * Преобразуем таблицу в 1НФ (разбиваем многозначные атрибуты).
   * Преобразуем таблицу в 2НФ, устранив частичные зависимости.
   * Преобразуем таблицу в 3НФ, устранив транзитивные зависимости.

**Таблица заказов (1НФ → 2НФ → 3НФ):**

| **Заказ ID** | **Товар ID** | **Количество** | **Дата** |
| --- | --- | --- | --- |
| 1 | 101 | 2 | 2024-01-01 |
| 1 | 102 | 1 | 2024-01-01 |
| 2 | 101 | 3 | 2024-01-02 |

**Таблица товаров (2НФ → 3НФ):**

| **Товар ID** | **Название товара** | **Цена** |
| --- | --- | --- |
| 101 | Товар A | 100 |
| 102 | Товар B | 150 |

**Таблица заказов:**

| **Заказ ID** | **Дата** |
| --- | --- |
| 1 | 2024-01-01 |
| 2 | 2024-01-02 |

**Итог**

* **1НФ** — удаление многозначных атрибутов.
* **2НФ** — устранение частичных зависимостей.
* **3НФ** — устранение транзитивных зависимостей.

Нормализация данных помогает улучшить структуру базы данных, избежать аномалий и поддерживать целостность данных, а также повысить эффективность работы с данными в долгосрочной перспективе.

**24. Виды и сравнительная характеристика архитектуры информационных систем на основе баз данных.**  
Архитектура информационных систем на основе баз данных может быть разделена на несколько типов:

1. **Клиент-сервер**: данные хранятся на сервере, а клиенты подключаются для работы с ними.
2. **Трехуровневая архитектура**: включает уровень клиента, уровня приложения и уровень базы данных.
3. **Облачные системы**: данные хранятся в облачных сервисах, и доступ к ним осуществляется через интернет. Сравнение:

* **Клиент-сервер**: быстрый доступ, но возможны ограничения по масштабируемости.
* **Трехуровневая архитектура**: гибкость, высокая производительность, но сложность в настройке.
* **Облачные системы**: высокая масштабируемость, простота развертывания, но зависимость от интернет-соединения.

**25. Типовая функциональность промышленных систем управления базами данных.**  
Промышленные СУБД (например, Oracle, MS SQL Server) обычно обеспечивают:

* Хранение данных с высокой производительностью.
* Поддержку транзакций и механизмов восстановления данных.
* Механизмы обеспечения целостности и безопасности данных.
* Инструменты для резервного копирования и восстановления.
* Средства для мониторинга и оптимизации работы базы данных.

**26. Типовой состав промышленных систем управления базами данных.**  
Промышленные системы управления базами данных (СУБД) — это сложные программные комплексы, предназначенные для хранения, управления и обработки данных в различных приложениях. Они обеспечивают организацию, целостность, безопасность данных и предоставляют интерфейсы для взаимодействия с пользователями и другими системами.

Типовой состав промышленных СУБД включает несколько ключевых компонентов, которые обеспечивают функционирование системы. Рассмотрим их более подробно, а также структуру типичной СУБД.

**Основные компоненты и типовой состав промышленных СУБД:**

1. **Ядро СУБД** (Database Engine):
   * Центральный компонент системы, который отвечает за основные операции над базой данных, такие как управление транзакциями, обработка запросов, взаимодействие с данными.
   * Сюда входят механизмы индексирования, планирования запросов, оптимизации, блокировки и восстановления данных.
2. **Система управления данными** (Data Management System):
   * Система хранения данных, которая управляет физическим представлением данных на диске.
   * Включает в себя управление пространством хранения, использование буферных кэш-памятей для ускорения доступа, и реализацию методов восстановления после сбоев.
3. **Язык запросов** (Query Language):
   * Основной язык для взаимодействия с СУБД, в промышленной практике чаще всего используется **SQL** (Structured Query Language).
   * Этот компонент включает парсер запросов, интерпретатор и компилятор, которые переводят запросы пользователя в операции, которые могут быть выполнены СУБД.
4. **Менеджер транзакций** (Transaction Manager):
   * Обеспечивает выполнение транзакций в соответствии с принципами ACID (атомарность, согласованность, изолированность, долговечность).
   * Отвечает за обеспечение целостности данных и восстановление базы данных после сбоев.
5. **Система управления метаданными** (Metadata Management):
   * Составляет описание структуры базы данных, определяет типы данных, связи между таблицами, ограничения, индексы и другие объекты.
   * Хранит информацию о данных, которую использует сама СУБД и прикладные программы.
6. **Менеджер доступа к данным** (Data Access Manager):
   * Обеспечивает пользователям и приложениям доступ к данным с соблюдением безопасности и прав доступа.
   * Включает механизмы аутентификации и авторизации, шифрования данных, а также управления ролями и правами.
7. **Интерфейсы для приложений** (Application Interfaces):
   * Предоставляют программы и пользователям возможность взаимодействовать с базой данных через **API** (Application Programming Interfaces), такие как ODBC (Open Database Connectivity), JDBC (Java Database Connectivity), .NET.
   * Включают драйверы и библиотеки для работы с данными из различных приложений.
8. **Система обеспечения безопасности** (Security System):
   * Управляет безопасностью данных и системы. Включает в себя механизмы аутентификации пользователей, разграничение прав доступа, шифрование данных на уровне базы данных.
   * Позволяет контролировать доступ к данным на основе ролей и прав.
9. **Система резервного копирования и восстановления** (Backup and Recovery System):
   * Обеспечивает защиту данных от потери и повреждения. Включает механизмы для создания резервных копий базы данных и восстановления данных после сбоев или утрат.
   * Управляет настройками автоматического резервного копирования и проведения восстановления.
10. **Система отчетности и анализа** (Reporting and Analytics):
    * Обеспечивает создание отчетов и выполнение аналитических запросов, часто включает в себя интеграцию с внешними BI-системами (Business Intelligence).
    * Могут быть реализованы средства визуализации данных и выполнения сложных вычислений.
11. **Инструменты администрирования и мониторинга** (Administration and Monitoring Tools):
    * Позволяют администраторам базы данных следить за состоянием системы, производительностью, а также настраивать параметры системы.
    * Включают средства для настройки индексов, мониторинга запросов, управления репликацией и распределенными базами данных.

**Примерная схема структуры промышленной СУБД**

Для лучшего понимания типового состава, рассмотрим схему структуры промышленной СУБД, которая может включать следующие компоненты:

+------------------------+

| Клиентские программы|

+------------------------+

|

v

+------------------------+ +------------------------+

| Интерфейсы приложений | <---> | Язык запросов (SQL) |

+------------------------+ +------------------------+

| |

v v

+------------------------+ +--------------------------+

| Менеджер транзакций | <-> | Ядро СУБД (Database Engine) |

+------------------------+ +--------------------------+

| |

v v

+------------------------+ +--------------------------+

| Менеджер доступа к | | Система хранения данных |

| данным (Data Access) | | (Data Management System) |

+------------------------+ +--------------------------+

| |

v v

+------------------------+ +--------------------------+

| Система безопасности| | Система резервного копирования|

| (Security) | | и восстановления (Backup & |

+------------------------+ | Recovery) |

| |

v v

+------------------------+ +--------------------------+

| Система метаданных | <-> | Система отчетности и |

| (Metadata Management) | | аналитики (Analytics) |

+------------------------+ +--------------------------+

**Описание компонентов схемы:**

1. **Клиентские программы**: Это приложения, которые используют базу данных для хранения и обработки данных. Например, это могут быть веб-приложения, мобильные приложения или настольные программы.
2. **Интерфейсы приложений (API)**: Связывают клиентские программы с СУБД. Они предоставляют стандартные интерфейсы для выполнения запросов и получения данных (например, ODBC, JDBC).
3. **Язык запросов (SQL)**: Язык, который используется для написания запросов к базе данных. SQL принимает запросы от пользователей или приложений и передает их ядру СУБД для обработки.
4. **Менеджер транзакций**: Обеспечивает атомарность и консистентность транзакций, а также управление их изоляцией и долговечностью.
5. **Ядро СУБД (Database Engine)**: Основная часть СУБД, которая отвечает за выполнение запросов, управление данными, выполнение операций над данными и взаимодействие с другими компонентами.
6. **Менеджер доступа к данным**: Обеспечивает контроль за правами доступа и защитой данных.
7. **Система хранения данных**: Управляет физическим хранением данных на дисках, управление пространством, кэшированием и оптимизацией хранения.
8. **Система безопасности**: Обеспечивает защиту данных от несанкционированного доступа и утечек.
9. **Система резервного копирования и восстановления**: Осуществляет регулярное создание резервных копий данных и позволяет восстанавливать их в случае сбоев.
10. **Система отчетности и аналитики**: Позволяет создавать отчеты и проводить анализ данных, предоставляя дополнительные инструменты для работы с бизнес-данными.
11. **Система администрирования и мониторинга**: Позволяет администрировать и следить за состоянием базы данных, а также управлять репликацией и производительностью.

**Заключение**

Типовой состав промышленных СУБД включает в себя несколько ключевых компонентов, которые обеспечивают функциональность, безопасность и удобство работы с данными. Каждый компонент выполняет определенные задачи и взаимодействует с другими компонентами для обеспечения надежной работы базы данных. Структура и состав СУБД зависят от конкретных требований, типа базы данных (реляционная, объектно-ориентированная и т. д.) и масштаба применения (например, корпоративные системы, облачные платформы).

**27. Характеристика распространённых систем управления базами данных.**  
**Сравнительная таблица СУБД**

| Характеристика | **PostgreSQL** | **MySQL** | **Microsoft SQL Server** | **Oracle Database** | **MongoDB** | **Redis** | **Cassandra** | **SQLite** |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **Логическая модель** | Реляционная | Реляционная | Реляционная | Реляционная | Документо-ориентированная | Ключ-значение | Широкостолбцовая | Реляционная |
| **Физическая модель** | Строчно-ориентированная | Строчно-ориентированная, с подключаемыми движками хранения | Странично-ориентированная | Таблично-пространственно-ориентированная | Документно-ориентированное хранилище | В оперативной памяти (с возможностью сохранения на диск) | Распределенная, на основе модели данных | Файловая |
| **Типы данных** | Обширные, включая массивы, JSON, XML | Стандартные SQL типы, JSON | Полный диапазон, включая типы CLR | Широкий диапазон, включая пользовательские типы | JSON-подобные документы, гибкие схемы | Строки, хэши, списки, множества и т. д. | Широко настраиваемые, столбцы могут хранить разные типы данных | Стандартные SQL типы |
| **Индексы** | B-дерево, хэш, GiST, GIN, BRIN | B-дерево, R-дерево, полнотекстовый, хэш | B-дерево, кластеризованные, некластеризованные, колоночные | B-дерево, битовые, на основе функций | Встроенные индексы, включая составные и геопространственные | Отсортированные множества, строковый индекс | Секционированные индексы, вторичные индексы | Индексы B-дерево |
| **Языки манипулирования** | SQL, Процедурный язык PL/pgSQL | SQL, расширения | T-SQL, CLR хранимые процедуры | SQL, PL/SQL | JavaScript, другие языки, специфичные для драйверов | Различные команды и функции | CQL (Язык запросов Cassandra) | SQL |
| **Встроенные языки прогр.** | PL/pgSQL, другие языки с расширениями | Хранимые процедуры, функции с расширениями | T-SQL, интеграция CLR | PL/SQL, Java, другие языки | JavaScript | Lua скриптинг | CQL, интеграция на стороне клиента | C, C++, Java bindings, Tcl |
| **Ген. форм/отчетов** | Ограниченные базовые возможности, сторонние инструменты | Ограниченные базовые возможности, сторонние инструменты | Службы отчетов SQL Server (SSRS), сторонние | Oracle Reports, сторонние инструменты | Обычно требуют внешних инструментов | Нет базовых возможностей | Обычно требуются внешние инструменты | Сторонние инструменты |
| **Поддержка транзакций** | Полное соответствие ACID | Полное соответствие ACID (для InnoDB) | Полное соответствие ACID | Полное соответствие ACID | Поддержка ACID в пределах одного документа | ACID (настраиваемый) | Легковесные транзакции, eventual consistency | Полное соответствие ACID |
| **Поддержка триггеров/хр. процедур** | PL/pgSQL, стандартные SQL триггеры | Хранимые процедуры, функции | T-SQL, CLR хранимые процедуры | PL/SQL, пакеты, триггеры | Функции, ограниченная поддержка | Lua скриптинг | Триггеры (менее традиционные, больше похоже на намеки) | Триггеры |
| **Платформы** | Linux, Windows, macOS, BSD | Linux, Windows, macOS, BSD, различные встроенные | Windows, Linux | Linux, Unix, Windows, другие | Linux, Windows, macOS, Docker, Kubernetes | Linux, Windows, macOS | Linux, Windows, macOS, другие облачные платформы | Windows, Linux, macOS, iOS, Android |
| **Область применения** | Общего назначения, геопространственная, хранилища данных | Веб-приложения, общего назначения, аналитика | Корпоративные приложения, бизнес-аналитика | Крупные предприятия, критически важные приложения | Большие данные, управление контентом, мобильные приложения | Кэширование, управление сессиями, обмен сообщениями | Высокая масштабируемость, быстрые операции чтения/записи | Мобильные, встроенные приложения, приложения, требующие встроенной БД |
| **Особенности и ограничения** | Открытый исходный код, расширяемая, сложные функции | Открытый исходный код, проста в использовании, ограничения для сложных операций | Хорошая интеграция с экосистемой Microsoft, более высокая стоимость для предприятий | Высокая стоимость, сложное администрирование, богатый набор функций | Масштабируемая, гибкие схемы, нет традиционного SQL | Быстрая работа в памяти, сохранение данных на диск настраивается | Высокомасштабируемая, но сложная, компромиссы в консистенции данных | Встроенная, простая, ограниченная масштабируемость для крупных приложений |

**Объяснение параметров:**

* **Логическая модель:** Относится к тому, как данные концептуально организованы (например, реляционные таблицы, документы, пары ключ-значение).
* **Физическая модель:** Описывает, как данные хранятся на диске (например, строчно-ориентированные, столбцово-ориентированные, в памяти).
* **Типы данных:** Виды данных, которые СУБД может хранить (например, целые числа, строки, даты, JSON и т. д.).
* **Индексы:** Структуры данных, которые улучшают скорость запросов (например, B-деревья, хэш-индексы, полнотекстовые).
* **Языки манипулирования:** Языки, используемые для запроса и изменения данных (например, SQL, T-SQL, JavaScript).
* **Встроенные языки программирования:** Языки, которые можно использовать для создания хранимых процедур или функций внутри СУБД.
* **Генераторы форм/отчетов:** Встроенные инструменты (или их отсутствие) для создания пользовательских интерфейсов и отчетов.
* **Поддержка транзакций:** Степень соответствия СУБД принципам ACID (атомарность, согласованность, изолированность, надежность).
* **Поддержка триггеров/хранимых процедур:** Возможности для создания повторно используемого кода, который выполняется автоматически при определенных событиях или по запросу.
* **Платформы:** Операционные системы или окружения, в которых может работать СУБД.
* **Область применения:** Типичные варианты использования и отрасли, в которых обычно используется каждая СУБД.
* **Особенности и ограничения:** Ключевые сильные и слабые стороны каждой СУБД.

**Дополнительные примечания:**

* **Открытый исходный код:** PostgreSQL, MySQL, MongoDB, Redis, Cassandra, SQLite — это системы с открытым исходным кодом.
* **Коммерческие:** Microsoft SQL Server и Oracle Database являются коммерческими продуктами, что означает, что у них обычно более высокая стоимость и больше вариантов поддержки.
* **NoSQL против реляционных:** MongoDB, Redis и Cassandra классифицируются как базы данных NoSQL и имеют другие шаблоны хранения данных и запросов, чем другие реляционные СУБД в списке.
* **Гибкость:** Документно-ориентированные и базы данных типа «ключ-значение» (такие как MongoDB и Redis соответственно) часто связаны с гибкостью схемы и более простым масштабированием за счет менее строгих отношений и целостности данных.
* **Сложность:** Некоторые системы очень сложны (Oracle) с богатым набором функций, в то время как другие (SQLite) предназначены для простоты и встроенного использования.

Эта таблица представляет собой сравнение высокого уровня. Конкретный выбор СУБД будет сильно зависеть от уникальных потребностей приложения или системы, которая разрабатывается. Каждая из этих баз данных имеет очень сложные и нюансированные конфигурации, варианты использования и реализации.

Это всеобъемлющее сравнение обеспечивает хорошую отправную точку для понимания различных доступных вариантов.

**28. СУБД MS SQL Server: системные БД и таблицы.**  
MS SQL Server включает несколько системных баз данных, таких как:

* **master**: хранит информацию о конфигурации.
* **model**: используется как шаблон для создания новых баз данных.
* **msdb**: используется для хранения информации о заданиях и операциях.
* **tempdb**: временные таблицы и хранимые процедуры.
* **resource**: системные объекты, требуемые для работы СУБД.