1. Введение в базы данных

> База данных (БД) — это организованное хранилище информации, которое позволяет легко и эффективно хранить, извлекать и управлять данными. Базы данных могут хранить различные типы данных, такие как текст, цифры, изображения, звук и видео, и могут быть использованы для различных целей, таких как ведение учета, анализ данных, хранение мультимедийных файлов и многое другое. Базы данных обычно управляются с помощью специального программного обеспечения, называемого системой управления базами данных (СУБД), которое предоставляет инструменты для создания, модификации и запросов к данным в базе данных.

История и этапы развития

История баз данных начинается ещё в далёком прошлом, когда данные хранились вручную в бумажных карточных индексах. С развитием компьютерных технологий и необходимостью обработки больших объемов данных появились первые электронные системы хранения данных:

- **1960-е годы**: Появление иерархических и сетевых баз данных. Данные структурировались в виде древовидных структур или сложных сетей.
- **1970-е годы**: Разработка реляционной модели Эдгаром Коддом, которая позволяет хранить данные в таблицах, состоящих из строк и столбцов. Это стало революционным изменением, так как реляционная модель была более гибкой и понятной.
- **1980-е годы**: Распространение реляционных систем управления базами данных (СУБД), таких как Oracle, MySQL, Microsoft SQL Server, SQLite
- **1990-е годы**: Рост и развитие объектно-ориентированных и NoSQL баз данных, а также распространение интернет-технологий и облачных вычислений, что привело к созданию распределенных и масштабируемых систем хранения данных.

Типы баз данных

1. **Реляционные базы данных (РБД):**

РБД хранят данные в виде таблиц, состоящих из строк (записей) и столбцов (полей). Каждая таблица имеет первичный ключ — уникальный идентификатор, который используется для поиска и извлечения данных из таблицы. Связи между таблицами определяются с помощью внешних ключей — полей, содержащих первичные ключи из других таблиц.

Аналогии для РБД:

- Библиотека с полками и книгами: Каждая полка представляет таблицу, а книги на полке — записи в таблице. Первичный ключ — номер книги, а внешние ключи — ссылки на другие книги в библиотеке.
- Супермаркет: Товары в супермаркете разделены на отделы (таблицы), а товары в отделе — это записи. Первичный ключ — штрихкод товара, а внешние ключи — ссылки на другие товары, которые могут быть связаны (например, комплектующие для компьютера).

Исторические факты:

- РБД были разработаны в 1970-х годах Эдгаром Ф. Коддом на основе его работы по теории реляционных баз данных.
- Первая коммерческая РБД, Oracle, была выпущена в 1979 году.

Особенности РБД:

- Структурированность: РБД хранят данные в виде таблиц, состоящих из строк (записей) и столбцов (полей). Эта структурированность позволяет легко организовывать и находить данные.
- Связи между таблицами: РБД используют первичные и внешние ключи для создания связей между таблицами. Это позволяет избежать избыточности данных и упрощает процесс обновления и удаления данных.

- SQL: РБД поддерживают язык SQL (Structured Query Language), который является стандартом для работы с реляционными базами данных. SQL позволяет выполнять сложные запросы и операции с данными.
- Масштабируемость: РБД могут масштабироваться для хранения и обработки больших объемов данных, что делает их идеальными для корпоративных и веб-приложений.
- Нормализация: РБД поддерживают концепцию нормализации, которая позволяет избежать избыточности данных и упрощает процесс обновления и удаления данных.

2. **Объектно-ориентированные базы данных (ООБД):**

ООБД хранят данные в виде объектов, которые содержат как данные, так и методы для их обработки. ООБД используют объектно-ориентированную модель данных, основанную на концепциях классов, объектов, наследования и полиморфизма.

Аналогии для ООБД:

- Коллекция игрушек: Каждая игрушка — это объект с определенными свойствами (цвет, размер, материал) и методами (движение, звук). Наследование позволяет создавать новые игрушки на основе существующих.
- Кулинарная книга: Рецепты — это объекты, которые содержат список ингредиентов (свойства) и инструкции по приготовлению (методы). Наследование позволяет создавать новые рецепты на основе существующих.

Исторические факты:

- ООБД были разработаны в 1980-х годах в ответ на рост популярности объектно-ориентированного программирования.
- Первая коммерческая ООБД, Ontos, была выпущена в 1989 году.

Особенности ООБД:

- Объекты: ООБД хранят данные в виде объектов, которые содержат как данные, так и методы для их обработки. Это позволяет легко моделировать и манипулировать сложными объектами и их отношениями.
- Наследование: ООБД поддерживают концепцию наследования, которая позволяет создавать новые объекты на основе существующих, что упрощает процесс разработки и сокращает объем кода.
- Полиморфизм: ООБД поддерживают концепцию полиморфизма, которая позволяет использовать один и тот же метод для различных объектов, что упрощает процесс разработки и сокращает объем кода.
- ООП: ООБД поддерживают концепции объектно-ориентированного программирования (ООП), такие как инкапсуляция, абстракция и модульность, что упрощает процесс разработки и сокращает объем кода.
- Сложность: ООБД могут быть сложнее в разработке и администрировании, чем другие типы баз данных, из-за их специализированной природы и необходимости обрабатывать сложные объекты и их отношения.

3. **Документно-ориентированные базы данных (ДОБД):**

ДОБД хранят данные в виде документов, которые представляют собой единицы информации, такие как файлы JSON или XML. ДОБД используют иерархическую модель данных, основанную на концепции вложенных элементов, для определения структуры и связей между данными.

Аналогии для ДОБД:

- Коллекция писем: Каждое письмо — это самостоятельный документ, содержащий всю необходимую информацию (адрес, дата, текст). Документы могут быть объединены в папки или альбомы для удобства хранения и поиска.
- Коллекция фотографий: Каждая фотография — это самостоятельный документ, содержащий всю необходимую информацию (дата, место, описание). Фотографии могут быть объединены в альбомы или коллекции для удобства хранения и поиска.

Исторические факты:

- ДОБД были разработаны в 2000-х годах в ответ на рост популярности веб-приложений и необходимость хранения полуструктурированных данных.
- Первая коммерческая ДОБД, MongoDB, была выпущена в 2009 году.

Особенности ДОБД:

- Документы: ДОБД хранят данные в виде документов, которые представляют собой самоописывающиеся единицы информации, такие как файлы JSON или XML. Это позволяет легко хранить и извлекать сложные иерархические

данные.

- Гибкость: ДОБД позволяют хранить данные с различной структурой в одной коллекции, что делает их гибкими и удобными для хранения неструктурированных данных.
- Индексирование: ДОБД поддерживают индексирование, которое позволяет быстро находить и извлекать данные из больших коллекций.
- Масштабируемость: ДОБД могут масштабироваться для хранения и обработки больших объемов данных, что делает их идеальными для веб-приложений и мобильных устройств.
- Простота: ДОБД могут быть проще в разработке и администрировании, чем другие типы баз данных, из-за их простой и гибкой природы.

4. **Графовые базы данных (ГБД):**

ГБД хранят данные в виде графов, которые состоят из узлов (вершин) и ребер, соединяющих эти узлы. ГБД используют модель данных, основанную на концепции графов, для определения структуры и связей между данными.

Аналогии для ГБД:

- Карта метро: Станции — это узлы, а пути между станциями — ребра. Связи между станциями могут быть использованы для нахождения кратчайшего пути или оптимального маршрута.
- Сеть дорог: Перекрестки — это узлы, а дороги между перекрестками — рёбра. Связи между перекрестками могут быть использованы для нахождения кратчайшего пути или оптимального маршрута.

Исторические факты:

- ГБД были разработаны в 2000-х годах в ответ на необходимость хранения и анализа больших объемов связанных данных.
- Первая коммерческая графовая база данных, Neo4j, была выпущена в 2010 году.

Особенности графовых баз данных (ГБД):

- Графы: ГБД хранят данные в виде графов, которые состоят из узлов (вершин) и ребер, соединяющих эти узлы. Это позволяет легко моделировать и манипулировать сложными связями между данными.
- Связи: ГБД используют концепцию связей для создания и манипулирования отношениями между данными. Это позволяет легко находить и извлекать данные на основе их связей.
- Обход под дереву: ГБД поддерживают концепцию обходу по дереву, которая позволяет легко перемещаться по графу и находить данные на основе их связей.
- Масштабируемость: ГБД могут масштабироваться для хранения и обработки больших объемов данных, что делает их идеальными для социальных сетей, рекомендательных систем и других приложений, основанных на связях между данными.
- Сложность: ГБД могут быть сложнее в разработке и администрировании, чем другие типы баз данных, из-за их специализированной природы и необходимости обрабатывать сложные связи между данными.

5. **Векторные базы данных (ВБД):**

ВБД являются специализированным типом баз данных, предназначенным для хранения, поиска и управления векторными данными, такими как изображения, звук, видео и другие многомерные данные. ВБД используют векторную модель данных, основанную на концепции векторов и пространств, для определения структуры и связей между данными.

Аналогии для векторных БД:

- Коллекция фотографий: Каждая фотография представляет собой векторный объект, содержащий пиксели и цвета. Векторные БД могут использоваться для хранения и поиска фотографий по их визуальным характеристикам, таким как цвет, форма и текстура.
- Коллекция музыкальных треков: Каждый музыкальный трек представляет собой векторный объект, содержащий частоты и амплитуды звуковых волн. Векторные БД могут использоваться для хранения и поиска музыкальных треков по их звуковым характеристикам, таким как жанр, темп и настроение.

Исторические факты:

- Векторные базы данных были разработаны в 1990-х годах в ответ на рост популярности мультимедийных приложений и необходимость хранения и поиска больших объемов векторных данных.

- Первая коммерческая векторная база данных, FAST, была выпущена в 1997 году.

Особенности векторных баз данных:

- Многомерность: Векторные базы данных могут хранить и обрабатывать данные с высокой размерностью, такие как изображения, звук и видео.
- Подобие: Векторные базы данных могут использоваться для поиска подобных объектов на основе их векторных характеристик, таких как цвет, форма и текстура.
- Масштабируемость: Векторные базы данных могут масштабироваться для хранения и обработки больших объемов векторных данных, что делает их идеальными для мультимедийных и машинно-обучающихся приложений.
- Сложность: Векторные базы данных могут быть сложнее в разработке и администрировании, чем другие типы баз данных, из-за их специализированной природы и необходимости обрабатывать сложные векторные данные.

Понятие реляционной модели баз данных

Реляционная модель баз данных — это подход к управлению данными, который организует данные в структурированном формате, известном как таблицы. Эта модель была предложена Эдгаром Коддом в 1970 году и с тех пор стала одним из наиболее широко используемых подходов для управления базами данных. Вот основные аспекты реляционной модели:

Таблицы (отношения):

В реляционной базе данных, все данные хранятся в таблицах, которые также иногда называют "отношениями". Таблица состоит из рядов и столбцов, аналогично тому, как она представлена в электронной таблице.

- **Строки (кортежи)**: Каждая строка таблицы представляет собой запись, содержащую уникальные данные. В контексте базы данных, строки также часто называют "кортежами".
- **Столбцы (атрибуты)**: Каждый столбец таблицы представляет определенный тип данных (атрибут), который хранится в таблице. Например, столбец "Имя" в таблице "Сотрудники".

Ключи:

Ключи — это особые поля в таблице, предназначенные для идентификации и управления данными.

- **Первичный ключ (Primary Key)**: Уникально идентифицирует каждую строку в таблице. Ни один из компонентов первичного ключа не может быть null.
- **Внешний ключ (Foreign Key)**: Ссылка на первичный ключ другой таблицы, используется для установления связи между таблицами.

Нормализация:

Нормализация — это процесс организации данных в таблицах таким образом, чтобы минимизировать избыточность и зависимость данных. Это достигается путем разделения данных на несколько связанных таблиц и установления связей между ними с помощью внешних ключей. Целью нормализации является повышение целостности и оптимизация хранения данных.

Целостность данных:

Реляционная модель предоставляет строгие правила, которые помогают поддерживать точность и надежность данных, известные как ограничения целостности. Эти ограничения включают:

- **Ограничения первичного ключа**: Обеспечивают уникальность каждой строки в таблице.
- **Ограничения внешнего ключа**: Гарантируют, что связи между таблицами остаются действительными.
- **Ограничения целостности домена**: Устанавливают правила для допустимых значений в столбцах.

Язык SQL

SQL (Structured Query Language) — это специальный язык, разработанный для работы с реляционными базами данных. SQL появился в 1970-х годах в лабораториях IBM и был стандартизирован в 1986 году. SQL используется для создания, модификации и удаления таблиц и данных в базе данных, а также для выполнения запросов — инструкций, которые извлекают и обрабатывают данные из одной или нескольких таблиц.

Исторические факты:

- SQL был разработан в 1970-х годах Дональдом Чемберленом и Рэем Бойсом в лабораториях IBM.
- Первоначально SQL назывался SEQUEL (Structured English Query Language) и был создан для работы с экспериментальной реляционной базой данных System R.
- В 1979 году название было сокращено до SQL, и в 1986 году SQL стал стандартом ANSI.

Диалекты и базы данных SQL:
SQL является стандартизированным языком, но существуют различные диалекты SQL, которые отличаются друг от друга синтаксисом и функциями. Некоторые из наиболее распространенных диалектов SQL включают:

1. **MySQL** — популярная открытая СУБД, которая используется во многих веб-приложениях.
2. **PostgreSQL** — открытая СУБД, которая известна своей надежностью и масштабируемостью.
3. **Microsoft SQL Server** — коммерческая СУБД от Microsoft, которая широко используется в корпоративных приложениях.
4. **Oracle Database** — коммерческая СУБД от Oracle Corporation, которая известна своей высокой производительностью и безопасностью.
5. **SQLite** — легковесная реляционная база данных, которая хранится в одном файле и не требует установки сервера.
6. **MariaDB** — ответвление открытой реляционной базы данных MySQL, созданное разработчиками MySQL после её поглощения компанией Oracle.

Отличия между диалектами и базами данных SQL заключаются в синтаксисе, функциях, производительности, масштабируемости и безопасности. Выбор конкретного диалекта или базы данных SQL зависит от требований к приложению, бюджета и предпочтений разработчика.

Особенности разных СУБД:

1. **Oracle:**

- Коммерческая реляционная база данных, разработанная компанией Oracle Corporation.
- Известна своей высокой производительностью, масштабируемостью и безопасностью.
- Поддерживает широкий спектр функций и инструментов для разработки, администрирования и анализа данных.
- Используется во многих крупных корпоративных системах, таких как финансовые, телекоммуникационные и государственные учреждения.
- Дороже, чем другие решения, и требует квалифицированных специалистов для администрирования и поддержки.

2. **PostgreSQL:**

- Открытая реляционная база данных, разработанная сообществом разработчиков.
- Известна своей надежностью, масштабируемостью и гибкостью.
- Поддерживает широкий спектр функций и инструментов для разработки, администрирования и анализа данных.
- Используется во многих веб-приложениях, геоинформационных системах и научных исследованиях.
- Бесплатна и имеет большое сообщество пользователей и разработчиков, которые предоставляют поддержку и обновления.

3. **Microsoft SQL Server:**

- Коммерческая реляционная база данных, разработанная компанией Microsoft.
- Известна своей тесной интеграцией с другими продуктами Microsoft, такими как Windows, Office и SharePoint.
- Поддерживает широкий спектр функций и инструментов для разработки, администрирования и анализа данных.
- Используется во многих корпоративных системах, таких как финансовые, здравоохранение и розничная торговля.
- Имеет несколько редакций, от базовой до предприятийной, с различными функциями и ценами.
- Требует квалифицированных специалистов для администрирования и поддержки, но имеет большое сообщество пользователей и разработчиков, которые предоставляют поддержку и обновления.

4. **MariaDB:**

- Ответвление открытой реляционной базы данных MySQL, созданное разработчиками MySQL после её поглощения компанией Oracle.
- Совместима с MySQL и поддерживает большинство ее функций и синтаксиса.
- Имеет ряд улучшений по сравнению с MySQL, таких как улучшенная производительность, безопасность и масштабируемость.
- Используется во многих веб-приложениях и корпоративных системах.

5. **SQLite:**

- Автономность: SQLite не требует установки сервера и может быть использована в автономном режиме.
- Портируемость: SQLite может быть легко перенесена между различными платформами и системами.
- Простота: SQLite имеет простой и понятный синтаксис, что делает его идеальным для обучения и разработки небольших приложений.
- Надежность: SQLite использует транзакционную модель, которая обеспечивает надежность и целостность данных.
- Масштабируемость: SQLite может обрабатывать большие объемы данных, но не подходит для высоконагруженных многопользовательских систем.

В зависимости от требований к приложению, бюджета и предпочтений разработчика, выбор конкретной базы данных может существенно отличаться. Важно тщательно изучить особенности, преимущества и недостатки каждой базы данных, прежде чем принимать решение.

Транзакции:

Транзакции в реляционной модели обеспечивают надежное выполнение группы операций. Они гарантируют, что все операции внутри транзакции либо полностью выполняются, либо полностью отменяются, что помогает поддерживать целостность данных даже в случае сбоев системы или ошибок.

Реляционная модель баз данных обеспечивает мощный, гибкий и интуитивно понятный способ организации и управления данными. Ее широкое применение и поддержка делают ее основным выбором для множества приложений и систем, от малых проектов до крупных корпоративных решений.

Почему на старте лучше начать с SQLite?

SQLite — это уникальная система управления базами данных (СУБД), которая существенно отличается от таких серверных СУБД, как MySQL или PostgreSQL. Основные особенности работы SQLite заключаются в следующем:

Встроенная СУБД

1. **Без Сервера:** В отличие от серверных СУБД, таких как MySQL или PostgreSQL, SQLite не использует клиент-серверную архитектуру. Это означает, что нет отдельного серверного процесса, который бы обрабатывал запросы к базе данных. Вместо этого SQLite работает как встраиваемая библиотека, непосредственно интегрируясь в приложения.
2. **Прямой Доступ к Файлам:** SQLite читает и записывает данные непосредственно в файлы на диске. Каждая база данных SQLite представляет собой отдельный файл, который можно перемещать, копировать или делиться им так же, как и с любым другим файлом.

Работа с SQLite через Программы и Библиотеки

1. **Графические Оболочки (например, SQLite Studio):**
   - Когда вы используете программу с графическим интерфейсом для работы с SQLite, такую как SQLite Studio, она встраивает в себя движок SQLite.
   - При выполнении запросов программа напрямую взаимодействует с файлом базы данных SQLite, используя встроенные функции SQLite для чтения и модификации данных.
   - Пользовательский интерфейс предоставляет удобную оболочку для формирования запросов SQL и визуализации данных.
2. **Использование в Программировании (например, библиотека sqlite3 в Python):**

- При использовании библиотеки, такой как `sqlite3` в Python, вы интегрируете SQLite непосредственно в свое приложение.
- Когда приложение запускается, оно может открывать, читать и писать в файл базы данных SQLite напрямую, используя функции библиотеки.
- Все операции с базой данных, включая создание таблиц, вставку, обновление, удаление и выборку данных, осуществляются через вызовы функций библиотеки.

Файлы баз данных SQLite обычно имеют следующие расширения:

- **.db**: Это наиболее общее расширение для файлов баз данных SQLite.
- **.sqlite**: Это другое популярное расширение, которое часто используется для уточнения, что файл является базой данных SQLite.
- **.sqlite3**: Это расширение указывает на то, что файл базы данных соответствует версии SQLite 3, которая является наиболее широко используемой версией на сегодняшний день.
- **.db3**: Менее распространенное, но также используемое расширение для файлов баз данных SQLite.

Преимущества и Особенности SQLite

1. **Простота и Легкость:** Одним из основных преимуществ SQLite является его простота. Нет необходимости в установке и настройке сервера баз данных, что делает его идеальным для легких приложений, встраиваемых систем и разработки прототипов.
2. **Портативность:** Файлы базы данных SQLite легко переносить и использовать на различных платформах и системах, что обеспечивает высокий уровень портативности данных.
3. **Низкие Затраты на Поддержку:** SQLite не требует регулярного обслуживания, такого как резервное копирование или мониторинг, что характерно для серверных СУБД.
4. **Мультиплатформенность:** SQLite доступен на множестве операционных систем и может быть встроен в разнообразные программные приложения, от мобильных приложений до крупных серверных систем.
5. **Транзакционность:** SQLite поддерживает транзакции, что гарантирует целостность данных даже при возникновении ошибок или сбоев системы.

Ограничения SQLite

1. **Масштабируемость:** SQLite не лучшим образом подходит для высоконагруженных приложений или приложений с высокой степенью параллелизма доступа к данным, так как он предназначен для локального доступа и не оптимизирован для большого количества параллельных операций записи.
2. **Функциональность:** В некоторых аспектах SQLite менее функционален, чем полноценные серверные СУБД. Например, он поддерживает ограниченный набор типов данных и не имеет встроенной поддержки таких функций, как хранимые процедуры или триггеры, на том же уровне, что и более крупные СУБД.

Где используется SQLite?

Ниже приведены несколько примеров, где SQLite находит свое применение:

1. **Мобильные Приложения:** Одно из самых распространенных применений SQLite - мобильные приложения. Из-за своего небольшого размера и легкости интеграции SQLite часто используется в качестве СУБД для iOS и Android приложений. Он идеален для хранения данных на устройствах с ограниченными ресурсами.
2. **Веб-браузеры:** Популярные веб-браузеры, такие как Google Chrome и Mozilla Firefox, используют SQLite для хранения данных о закладках, истории просмотров, кэша и других пользовательских данных.
3. **Компьютерные Программы:** Многие настольные приложения, включая программы для обработки текста, электронные таблицы и инструменты для управления данными, используют SQLite для хранения конфигураций, настроек, пользовательских данных и другой информации.
4. **Встроенные Системы и Интернет Вещей (IoT):** SQLite широко применяется во встроенных системах и устройствах Интернета вещей для локального хранения данных. Его легковесность и способность работать без необходимости серверной инфраструктуры делают его идеальным выбором для таких устройств.
5. **Файловые Форматы:** Некоторые программные приложения используют SQLite в качестве формата файлов. Например, приложение может сохранять свои данные в файле базы данных SQLite, что облегчает управление данными и обеспечивает их целостность.
6. **Аналитика и Журналирование:** SQLite может использоваться для журналирования и аналитики в приложениях, где

не требуется сложная обработка или масштабирование данных, но важна скорость и простота доступа к логам.

7. **Прототипирование и Тестирование:** Разработчики часто используют SQLite для быстрого прототипирования и тестирования приложений, так как это позволяет легко настраивать и управлять базами данных без сложной настройки.

Заключение

SQLite представляет собой уникальный тип СУБД, сочетающий в себе простоту и портативность с достаточным функционалом для многих приложений. Его встраиваемый характер делает его идеальным выбором для приложений, требующих легковесной, но надежной системы управления базами данных без сложностей, связанных с настройкой и поддержкой сервера.

SQLite широко используется в различных серьезных и крупных проектах благодаря своей надежности, простоте и портативности.

SQLite — это универсальное решение для множества приложений, от мобильных и настольных приложений до встроенных систем и IoT. Он особенно полезен в ситуациях, где требуется легковесная, но функциональная система управления базами данных, которая не требует сложной настройки и управления, характерных для серверных СУБД. Его надежность, простота в использовании и широкая поддержка различных платформ делают SQLite идеальным выбором для широкого спектра проектов, начиная от малых и заканчивая крупными и серьезными приложениями.

Запросы DDL, DML, DCL

Введение в SQL (Structured Query Language): SQL - это язык, который используется для работы с базами данных. С его помощью вы можете выполнять различные операции с данными, такие как создание, изменение, удаление и выборка данных из таблиц.

Стандарты языка SQL: SQL имеет различные стандарты, такие как SQL-92, SQL-99, SQL:2003 и другие. Эти стандарты определяют синтаксис и функциональность языка. В MySQL и SQLite используются разные версии стандарта SQL, но большинство основных команд и функций общие.

DDL, DML, DCL

- **DDL (Data Definition Language - язык определения данных):** Этот подмножество SQL используется для создания, изменения и удаления структуры базы данных, такой как таблицы, индексы и ключи. Примеры команд DDL включают `CREATE TABLE` (создание таблицы) и `ALTER TABLE` (изменение таблицы). В MySQL и SQLite поддерживаются команды DDL.
- **DML (Data Manipulation Language - язык манипуляции данными):** Этот подмножество SQL используется для выполнения операций с данными, таких как вставка, обновление и удаление записей из таблицы. Примеры команд DML включают `INSERT` (вставка данных), `UPDATE` (обновление данных) и `DELETE` (удаление данных). В MySQL и SQLite поддерживаются команды DML. Так же, сюда включены запросы выборки данных, всё что идет после `SELECT`
- **DCL (Data Control Language - язык управления данными):** Этот подмножество SQL используется для управления правами доступа к данным. Примеры команд DCL включают `GRANT` (разрешение доступа) и `REVOKE` (отзыв прав доступа). В MySQL и SQLite также поддерживаются команды DCL.

Синтаксис SQL

Основная структура запроса SQL выглядит следующим образом:

```
SELECT выражение1, выражение2, ...
FROM таблица1, таблица2, ...
WHERE условие1 AND/OR условие2 AND/OR ...
ORDER BY выражение1 ASC/DESC, выражение2 ASC/DESC, ...
LIMIT количество_строк OFFSET смещение
```

`SELECT`: Ключевое слово `SELECT` используется для выбора столбцов из таблицы или таблиц в реляционной базе данных. Это первое ключевое слово, которое пишется в инструкции SQL-запроса, и оно указывает, какие данные должны быть

извлечены из базы данных.

`FROM`: Ключевое слово `FROM` используется для указания таблицы или таблиц, из которых будут извлечены данные в реляционной базе данных. Это второе ключевое слово, которое пишется в инструкции SQL-запроса, и оно указывает, откуда должны быть получены данные.

`WHERE`: Ключевое слово `WHERE` используется для фильтрации данных, извлеченных из таблицы или таблиц в реляционной базе данных. Это ключевое слово указывает, какие конкретно строки должны быть возвращены в результате запроса, на основе указанных условий.

`AND`: Ключевое слово `AND` используется для объединения двух или более условий в инструкции SQL-запроса. Это ключевое слово указывает, что все указанные условия должны быть выполнены, чтобы строка была возвращена в результате запроса.

`OR`: Ключевое слово `OR` используется для объединения двух или более условий в инструкции SQL-запроса. Это ключевое слово указывает, что любое из указанных условий должно быть выполнено, чтобы строка была возвращена в результате запроса.

`NOT`: Ключевое слово `NOT` используется для отрицания условия в инструкции SQL-запроса. Это ключевое слово указывает, что строки, которые выполняют указанное условие, не должны быть возвращены в результате запроса.

`IN`: Ключевое слово `IN` используется для проверки, принадлежит ли значение столбца набору значений, указанному в инструкции SQL-запроса. Это ключевое слово упрощает запись запросов, в которых требуется проверить принадлежность значения столбца к нескольким значениям.

`ORDER BY`: Ключевое слово `ORDER BY` используется для сортировки данных, извлеченных из таблицы или таблиц в реляционной базе данных. Это ключевое слово указывает, по какому столбцу или столбцам должны быть отсортированы данные, и в каком порядке (возрастающем или убывающем).

`LIMIT`: Ключевое слово `LIMIT` используется для ограничения количества строк, возвращаемых в результате запроса в реляционной базе данных. Это ключевое слово указывает, сколько строк должно быть возвращено в результате запроса, начиная с первой строки.

`OFFSET`: Ключевое слово `OFFSET` используется для пропуска определенного количества строк в результате запроса в реляционной базе данных. Это ключевое слово указывает, сколько строк должно быть пропущено в результате запроса, начиная с первой строки. Обычно используется вместе с `LIMIT` для реализации пагинации.

`DISTINCT`: Ключевое слово `DISTINCT` используется для удаления дубликатов из результата запроса в реляционной базе данных. Это ключевое слово указывает, что должны быть возвращены только уникальные значения столбца или столбцов, указанных в инструкции `SELECT`. Например, если в таблице есть несколько строк с одинаковым значением в столбце "город", то при использовании `DISTINCT` будут возвращены только уникальные значения столбца "город".

Примеры SQL-запросов:

1. Извлечение всех данных из таблицы "MarvelCharacters":

```sql
SELECT * FROM MarvelCharacters;
```

2. Извлечение данных о персонажах Marvel, у которых количество появлений превышает 1000:

```sql
SELECT * FROM MarvelCharacters WHERE appearances > 1000;
```

3. Извлечение уникальных значений мировоззрения (align) из таблицы "MarvelCharacters":

```sql
SELECT DISTINCT align FROM MarvelCharacters;
```

4. Извлечение данных о персонажах Marvel, мировоззрение которых "Good Characters" или "Bad Characters":

```sql
SELECT * FROM MarvelCharacters WHERE align = 'Good Characters' OR align = 'Bad Characters';
```

5. Извлечение данных о персонажах Marvel, выравнивание которых не "good":

```
SELECT * FROM MarvelCharacters WHERE align ≠ 'Good Characters';
```

6. Извлечение данных о персонажах Marvel, мировоззрение которых входит в список ('Good Characters', 'Neutral Characters'):

```
SELECT * FROM MarvelCharacters WHERE align IN ('Good Characters', 'Neutral Characters');
```

7. Извлечение данных о персонажах Marvel, отсортированных по количеству появлений в убывающем порядке:

```
SELECT * FROM MarvelCharacters ORDER BY appearances DESC;
```

8. Извлечение данных о персонажах Marvel, отсортированных по количеству появлений в убывающем порядке, с ограничением в 10 строк:

```
SELECT * FROM MarvelCharacters ORDER BY appearances DESC LIMIT 10;
```

9. Извлечение данных о персонажах Marvel, отсортированных по количеству появлений в убывающем порядке, начиная со строки 11:

```
SELECT * FROM MarvelCharacters ORDER BY appearances DESC OFFSET 10;
```