1. Дайте определения следующим понятиям: «информационная система», «данное», «база данных», «предметная область». Приведите примеры каждого понятия. Классифицируйте СУБД по моделям данных и количеству пользователей

Информационная система — это система, предназначенная для сбора, хранения, обработки, передачи и использования информации для достижения определенных целей. Пример: система управления складом в компании, включающая в себя программное обеспечение для учета товаров, базу данных о наличии товаров и клиентов, а также оборудование для сканирования штрих-кодов.

Данное – факт, событие, объект или явление, представленное в виде символов, чисел, слов и т. д. Пример: 25 марта 2024 года, 1000 единиц товара, "Красная машина".

База данных — это организованная коллекция данных, обычно хранящихся и доступных для многократного использования. Пример: база данных сотрудников компании, содержащая информацию о их имени, должности, зарплате и т. д.

Предметная область — это область знаний или деятельности, в которой собрана информация, необходимая для решения определенных задач. Пример: предметная область управления проектами в IT-компании, включающая в себя информацию о задачах, сроках выполнения, ресурсах и т. д.

СУБД по моделям данных:

* Реляционные СУБД (SQL):

Примеры: MySQL, PostgreSQL, Oracle Database.

Основная модель данных: Реляционная модель данных, основанная на таблицах.

Характеристики: используют SQL для запросов и манипуляций данными.

* Документно-ориентированные СУБД:

Примеры: MongoDB, Couchbase.

Основная модель данных: Документно-ориентированная модель, данные хранятся в документах (например, JSON или BSON).

Характеристики: Гибкость схемы, масштабируемость, хорошо подходят для неструктурированных данных.

* Ключ-значение СУБД:

Примеры: Redis, DynamoDB.

Основная модель данных: Простая структура ключ-значение, где каждое значение связано с уникальным ключом.

Характеристики: Высокая производительность при доступе к данным по ключу, обычно используются для кэширования и сеансов.

СУБД по количеству пользователей:

* Однопользовательские СУБД:

Примеры: SQLite.

Характеристики: Предназначены для использования одним пользователем или в рамках одного процесса.

Область применения: Локальные приложения, мобильные приложения, небольшие веб-сайты.

* Многопользовательские СУБД:

Примеры: MySQL, PostgreSQL, Oracle Database.

Характеристики: Поддержка одновременного доступа нескольких пользователей к базе данных.

Область применения: Средние и крупные организации, веб-сайты с большим трафиком, приложения с распределенным доступом.

1. Дайте определение понятию «модель данных». Перечислите известные модели данных (не менее трех). Опишите назначение, отличительные признаки, достоинства и недостатки каждой из описанных моделей данных.

Модель данных — это абстрактное представление о структуре данных, которое определяет, как данные организованы, хранятся и обрабатываются в информационной системе.

Известные модели данных:

* Реляционная модель данных:

Назначение: Организация данных в виде таблиц, состоящих из строк и столбцов, связанных между собой по ключам.

Отличительные признаки: Четкая структура, использование SQL для запросов и манипуляций данными.

Достоинства: Простота запросов, надежность целостности данных, широкое распространение.

Недостатки: Неэффективность при работе с сложными иерархическими или связанными данными.

* Модель документа (неструктурированная):

Назначение: Хранение и обработка текстовых документов, изображений, видео и других форм неструктурированных данных.

Отличительные признаки: Отсутствие четкой структуры, данные могут быть представлены в свободной форме.

Достоинства: Гибкость, способность обрабатывать разнообразные данные.

Недостатки: Ограниченные возможности для поиска и анализа, сложность обработки.

* Модель XML (частично структурированная):

Назначение: Представление и обмен структурированными данными в виде текстовых документов.

Отличительные признаки: Использование тегов для определения структуры данных, возможность определения пользовательских схем данных.

Достоинства: Гибкость, поддержка схем для обеспечения целостности данных.

Недостатки: Больший объем данных из-за наличия тегов, сложность обработки.

1. Дайте определение понятию «база данных». Перечислите этапы проектирования базы данных. Опишите этап концептуального проектирования, укажите результат данного этапа.

База данных (БД) – это организованная коллекция данных, которая хранится и управляется с помощью компьютерной системы. В контексте баз данных, данные обычно организованы в виде таблиц, связанных друг с другом, чтобы обеспечить эффективный доступ, обновление и управление информацией.

Этапы проектирования базы данных:

Проектирование базы данных включает несколько этапов, в ходе которых определяется структура и организация данных:

* Концептуальное проектирование: на этом этапе определяются основные сущности и их атрибуты, а также связи между этими сущностями. Результатом этого этапа является ER-модель базы данных.
* Логическое проектирование: здесь происходит преобразование концептуальной модели в структуру данных, учитывая особенности конкретной СУБД. Создаются таблицы, определяются их поля и связи между таблицами. Результатом этого этапа является схема базы данных в виде таблиц и их отношений.
* Физическое проектирование: на этом этапе решаются вопросы организации данных на физическом уровне: выбор системы управления базами данных, оптимизация структуры таблиц, индексирование и размещение данных на диске.

Этап концептуального проектирования:

На этом этапе определяются ключевые сущности системы и их атрибуты, а также связи между ними. Это абстрактный уровень проектирования, который не зависит от конкретной СУБД и предназначен для понимания бизнес-логики системы. Основные задачи на этом этапе:

* Идентификация сущностей: Определение основных объектов, с которыми будет работать система (например, сущности "Клиент", "Товар", "Заказ" и т.д.).
* Определение атрибутов сущностей: Определение свойств каждой сущности, которые необходимы для описания ее характеристик.
* Определение связей между сущностями: Установление связей и зависимостей между различными сущностями, чтобы отразить взаимодействия и отношения между ними.

Результат концептуального проектирования: Результатом этапа концептуального проектирования является ER-модель базы данных, которая представляет собой диаграмму, отображающую основные сущности, их атрибуты и связи между ними. Эта модель обычно представляется в виде сущность-связь (ER-диаграмма) и служит основой для развития логической модели базы данных.

Начало формы

1. Дайте определение понятию «база данных». Перечислите этапы проектирования базы данных. Опишите этап логического (даталогического) проектирования: исходные данные и результаты этапа.

База данных (БД) – это организованная коллекция данных, которая хранится и управляется с помощью компьютерной системы. База данных обычно используется для хранения информации таким образом, чтобы она была легко доступна, управляема и обновляема.

Этапы проектирования базы данных:

* Концептуальное проектирование: определяются основные сущности, атрибуты и связи между ними. Создается ER-модель базы данных.
* Логическое (даталогическое) проектирование: преобразует концептуальную модель в структуру данных, учитывая особенности конкретной СУБД. Создается логическая модель данных.
* Физическое проектирование: организует данные на физическом уровне, оптимизирует структуру таблиц, выбирает индексы и т.д. Создается физическая модель данных.

Этап логического (даталогического) проектирования:

На этом этапе создается логическая модель данных, которая представляет собой абстрактное описание структуры данных на уровне конкретной СУБД.

Исходные данные этапа:

* ER-модель базы данных: Полученная на предыдущем этапе концептуальная модель, описывающая основные сущности, их атрибуты и связи между ними.

Результаты этапа:

* Логическая модель данных: это описание базы данных, которое учитывает особенности конкретной СУБД и является более конкретным и структурированным, чем концептуальная модель. Включает в себя:
* Определение таблиц и их столбцов.
* Определение первичных и внешних ключей.
* Определение связей между таблицами (один-к-одному, один-ко-многим, многие-ко-многим).
* Определение ограничений целостности данных.
* Определение правил нормализации данных.

Логическая модель данных не зависит от конкретной реализации в СУБД и может быть представлена в виде схемы базы данных или спецификации структуры данных в текстовом или графическом виде. Этот этап служит основой для создания физической модели данных на следующем этапе проектирования.

1. Дайте определение понятия «система управления базами данных (СУБД)». Перечислите этапы проектирования базы данных. Опишите этап физического проектирования и описания данных в базе данных: типы данных, определение размера данных, свойства (на примере MS SQL Server).

Система управления базами данных (СУБД) — это программное обеспечение, которое используется для создания, управления и обработки баз данных. Оно предоставляет средства для создания таблиц, индексов, хранимых процедур, а также для выполнения запросов к данным, их обновления и удаления.

Этапы проектирования базы данных:

* Концептуальное проектирование: Определение основных сущностей, атрибутов и связей между ними.
* Логическое (даталогическое) проектирование: Преобразование концептуальной модели в структуру данных, учитывая особенности конкретной СУБД.
* Физическое проектирование: Организация данных на физическом уровне, оптимизация структуры таблиц, индексирование и др.

Этап физического проектирования и описания данных в базе данных:

На этом этапе определяются физические аспекты базы данных, такие как типы данных, размеры данных, ограничения и другие свойства.

Типы данных: определяют формат и характеристики данных, которые могут храниться в каждом столбце таблицы. Например, целые числа, строки, даты и времена и т.д.

Определение размера данных: указывается максимальный размер данных, который может быть хранен в каждом столбце. Например, для строковых данных это может быть количество символов, для числовых данных - максимальное значение.

Свойства столбцов (на примере MS SQL Server):

* Типы данных: INT, VARCHAR, DATE, FLOAT и т.д.
* Определение размера данных: Например, VARCHAR(50) для строковых данных, указывает, что в этом столбце могут храниться строки длиной до 50 символов.
* Ограничения NULL и NOT NULL: указывает, может ли столбец содержать NULL значения или нет.
* Ограничения уникальности: гарантирует уникальность значений в столбце.
* Ограничения первичного ключа: указывает, что значения в столбце уникальны и не могут быть NULL.
* Ограничения внешнего ключа: связывает значения столбца с значениями в другой таблице.
* Индексы: позволяют ускорить поиск и сортировку данных по определенному столбцу.

Пример создания таблицы с использованием указанных свойств в MS SQL Server:

CREATE TABLE Employees (

EmployeeID INT PRIMARY KEY,

FirstName VARCHAR(50) NOT NULL,

LastName VARCHAR(50) NOT NULL,

BirthDate DATE,

Salary FLOAT,

DepartmentID INT,

CONSTRAINT FK\_Department FOREIGN KEY (DepartmentID) REFERENCES Departments(DepartmentID)

);

В этом примере EmployeeID - целочисленный столбец, являющийся первичным ключом, FirstName и LastName - строки с максимальной длиной 50 символов, ограниченные значением NOT NULL, BirthDate - столбец для хранения даты, Salary - числовой столбец для хранения зарплаты, DepartmentID - столбец для хранения идентификатора отдела, который является внешним ключом, связанным с другой таблицей Departments.

1. Дайте определения следующих понятий «сущность», «экземпляр сущности», «ключ», «потенциальный ключ», «искусственный ключ», «поле связи». Перечислите понятия, которыми характеризуется сущность. Приведите примеры сущностей и понятий, которыми характеризуется сущность.

Сущность — это объект или концепция, о которой хранится информация в базе данных. Каждая сущность обычно представляет реальный объект или событие, которое можно описать и идентифицировать.

Экземпляр сущности — это конкретный представитель сущности, который существует в базе данных. Например, если сущность "Клиент", то каждый отдельный клиент, записанный в базе данных, является экземпляром этой сущности.

Ключ — это атрибут или набор атрибутов, который однозначно идентифицирует каждую запись в таблице базы данных.

ключ — это атрибут или набор атрибутов, которые могут служить в качестве ключа, но не обязательно уникальны в каждой записи.

Искусственный ключ — это ключ, который создается специально для уникальной идентификации каждой записи в таблице. Обычно это целочисленный идентификатор, который автоматически генерируется базой данных.

Поле связи — это атрибут, который используется для создания связи между двумя таблицами в базе данных. Это обычно внешний ключ, который ссылается на первичный ключ другой таблицы.

Понятия, которыми характеризуется сущность:

* Имя сущности
* Атрибуты сущности
* Отношения с другими сущностями
* Ключи сущности
* Экземпляры сущности

Примеры сущностей и понятий:

Сущность "Студент":

* Имя сущности: "Студент"
* Атрибуты сущности: Имя, Фамилия, Дата рождения, Группа
* Ключи сущности: Идентификатор студента (студенческий билет)
* Экземпляры сущности: студенты с конкретными идентификаторами.

Понятие "Книга":

* Имя сущности: "Книга"
* Атрибуты сущности: Название, Автор, Год издания, ISBN
* Ключи сущности: ISBN (Международный стандартный книжный номер)
* Экземпляры сущности: Каждая отдельная книга в библиотеке.

Понятие "Заказ":

Имя сущности: "Заказ"

Атрибуты сущности: Номер заказа, Дата заказа, Сумма заказа

Ключи сущности: Номер заказа

Экземпляры сущности: Каждый отдельный заказ, оформленный клиентом.

1. Дайте определение понятия «связь между сущностями». Опишите основное их назначение и их типы, сопроводив примерами разных видов связей между сущностями. Разъясните поддержку связей «многие-ко-многим».

Связь между сущностями — это отношение между двумя или более сущностями в базе данных, которое определяет, как они связаны друг с другом или взаимодействуют между собой. Связи между сущностями позволяют моделировать различные виды отношений и структуры данных в базе данных.

Основное назначение связей между сущностями:

* Определение связей между различными сущностями в базе данных.
* Установление отношений и зависимостей между данными.
* Поддержка целостности данных и сохранение связности информации в базе данных.

Типы связей между сущностями:

* Один-ко-многим (One-to-Many): Один экземпляр одной сущности связан с несколькими экземплярами другой сущности. Например, один автор может написать несколько книг, но каждая книга имеет только одного автора.
* Многие-ко-многим (Many-to-Many): Множество экземпляров одной сущности связано с множеством экземпляров другой сущности. Например, каждый студент может выбрать несколько курсов, и каждый курс может быть выбран несколькими студентами.
* Один-к-одному (One-to-One): Каждый экземпляр одной сущности связан только с одним экземпляром другой сущности, и наоборот. Например, каждый человек имеет только один паспорт, и каждый паспорт принадлежит только одному человеку.

Примеры разных видов связей между сущностями:

Один-ко-многим:

* Один производитель может производить множество продуктов.
* Один автор может написать много книг.
* Один отдел может иметь много сотрудников.

Многие-ко-многим:

* Множество студентов может посещать множество курсов.
* Множество заказов может содержать множество товаров.
* Множество авторов может написать множество книг, а одна книга может иметь нескольких авторов.

Один-к-одному:

* Каждый пользователь может иметь только один профиль.
* Каждая компьютерная система может иметь только один уникальный серийный номер.

Поддержка связей "многие-ко-многим":

Связи "многие-ко-многим" обычно поддерживаются с использованием дополнительной таблицы-связи, которая связывает записи из двух других таблиц. Эта дополнительная таблица содержит в себе ключи из обеих таблиц, создавая таким образом связь между ними.

Например, если у нас есть таблица "Студенты" и таблица "Курсы", и мы хотим установить связь "многие-ко-многим" между ними, мы можем создать дополнительную таблицу "Регистрация на курсы", которая будет содержать в себе идентификаторы студентов и курсов, которые они посещают.

1. Дайте определение понятия «система управления базами данных (СУБД)». Опишите назначение и основные отличительные характеристики СУБД. Приведите примеры известных СУБД (не менее трех).

Система управления базами данных (СУБД) — это программное обеспечение, которое позволяет пользователям создавать, управлять и обрабатывать базы данных. Она предоставляет средства для создания и модификации структуры базы данных, выполнения запросов к данным, обеспечения безопасности данных и обеспечения целостности данных.

Назначение СУБД:

* Хранение и организация данных: СУБД обеспечивает удобное хранение и организацию данных, что позволяет эффективно управлять большими объемами информации.
* Управление данными: СУБД позволяет пользователям добавлять, изменять и удалять данные в базе данных с помощью удобных средств управления.
* Обработка данных: СУБД предоставляет средства для выполнения различных операций над данными, таких как фильтрация, сортировка, агрегация и т. д.
* Обеспечение безопасности данных: СУБД позволяет устанавливать права доступа к данным, шифровать конфиденциальные данные и реализовывать механизмы аутентификации и авторизации пользователей.
* Обеспечение целостности данных: СУБД гарантирует сохранность данных, контролируя их целостность и применяя соответствующие ограничения и правила.

Основные отличительные характеристики СУБД:

* Модель данных: СУБД может быть основана на различных моделях данных, таких как реляционная, объектно-ориентированная, иерархическая и др.
* Язык запросов: СУБД предоставляет язык запросов для выполнения операций с данными, такой как SQL (Structured Query Language).
* Многопользовательская поддержка: СУБД обеспечивает возможность одновременного доступа нескольких пользователей к базе данных.
* Масштабируемость: СУБД может масштабироваться от небольших локальных баз данных до крупных распределенных систем.
* Производительность: Хорошая производительность при выполнении запросов и обработке данных является важной характеристикой СУБД.

Примеры известных СУБД:

* MySQL: Открытая реляционная СУБД, широко используемая для веб-приложений.
* Oracle Database: Мощная реляционная СУБД, предназначенная для корпоративных приложений и баз данных большого масштаба.
* Microsoft SQL Server: Реляционная СУБД, разработанная Microsoft, которая обеспечивает широкий спектр возможностей для хранения и обработки данных в корпоративных средах.
* PostgreSQL: Мощная и расширяемая реляционная СУБД с открытым исходным кодом.
* MongoDB: Ориентированная на документы NoSQL СУБД, которая хранит данные в формате JSON-подобных документов.

1. Дайте определения следующих понятий: «нормализация», «декомпозиция отношений», «избыточность отношений». Поясните цель и основной метод. Поясните первую нормальную форму (1НФ). Опишите способы ее построения. Приведите примеры отношения, не находящегося в первой нормальной форме, и приведите его к первой нормальной форме.

* Нормализация в базах данных — это процесс организации структуры базы данных таким образом, чтобы уменьшить избыточность данных и обеспечить их целостность. Она включает в себя разделение таблиц на более мелкие и логически связанные части, чтобы уменьшить дублирование информации.
* Декомпозиция отношений – это процесс разделения отношений (таблиц) на более мелкие и более нормализованные отношения для улучшения структуры базы данных.
* Избыточность отношений возникает, когда в базе данных есть избыточная или повторяющаяся информация, что приводит к нежелательным дубликатам данных и увеличивает вероятность ошибок.

Цель нормализации:

* Уменьшение избыточности данных.
* Обеспечение целостности данных.
* Упрощение структуры базы данных.
* Увеличение эффективности операций с данными.

Основной метод нормализации: Разделение отношений на более мелкие и нормализованные части путем выделения функционально зависимых атрибутов.

Первая нормальная форма (1НФ): Отношение находится в первой нормальной форме, если все его атрибуты атомарны, то есть каждый атрибут содержит только одно значение, и отношение не содержит множественных значений или составных атрибутов.

Способы построения 1НФ:

Разделение составных атрибутов: если атрибут содержит несколько значений, его следует разделить на несколько атрибутов.

Разделение множественных значений: если в одной ячейке хранится несколько значений, каждое значение следует разместить в отдельной ячейке.

Пример отношения, не находящегося в 1НФ, и его приведение к 1НФ:

Предположим, у нас есть таблица "Заказы" с полями "Номер заказа", "Товары" и "Количество":

| Номер заказа | Товары | Количество |

|--------------|---------------- |------------|

| 001 | Хлеб, Молоко | 2, 3|

| 002 | Мясо, Яйца | 5, 6 |

Это отношение не находится в 1НФ, потому что атрибут "Товары" содержит множественные значения (Хлеб, Молоко, Мясо, Яйца), а атрибут "Количество" содержит несколько значений (2, 3, 5, 6) в виде списка.

Для приведения этого отношения к 1НФ мы можем разделить его на два отдельных отношения: "Заказы" и "Товары в заказах":

Таблица "Заказы":

| Номер заказа |

|--------------|

| 001 |

| 002 |

Таблица "Товары в заказах":

| Номер заказа | Товар | Количество |

|--------------|---------|------------|

| 001 | Хлеб | 2 |

| 001 | Молоко | 3 |

| 002 | Мясо | 5 |

| 002 | Яйца | 6 |

Теперь каждая запись содержит только одно значение для каждого атрибута, и отношение находится в первой нормальной форме.

1. Дайте определения следующих понятий: «нормализация», «функциональная зависимость». Поясните вторую нормальную форму (2НФ). Опишите способы ее построения. Приведите примеры.

Нормализация в базах данных — это процесс организации структуры базы данных для уменьшения избыточности данных и обеспечения их целостности. Этот процесс включает в себя разделение таблиц на более мелкие и логически связанные части.

Функциональная зависимость между двумя атрибутами в базе данных означает, что значение одного атрибута определяет значение другого атрибута.

Вторая нормальная форма (2НФ):

Отношение находится во второй нормальной форме, если оно находится в первой нормальной форме и каждый его неключевой атрибут функционально зависит от всего первичного ключа, а не от его части.

Способы построения 2НФ:

* Выделение повторяющихся групп данных в отдельные таблицы: если в таблице есть группа атрибутов, которые зависят только от части первичного ключа, а не от всего ключа, эти атрибуты могут быть выделены в отдельную таблицу.
* Создание дополнительных таблиц для атрибутов, функционально зависящих от части ключа: если атрибуты зависят только от части ключа, то их можно выделить в отдельную таблицу и связать с основной таблицей через внешний ключ.

Пример:

Предположим, у нас есть таблица "Заказы" с атрибутами "Номер заказа", "Название товара", "Цена товара" и "Количество":

| Номер заказа | Название товара | Цена товара | Количество |

|--------------|-----------------|-------------|------------|

| 001 | Кофе | 100 | 2 |

| 001 | Чай | 50 | 3 |

| 002 | Кофе | 100 | 1 |

| 002 | Чай | 50 | 2 |

Здесь атрибуты "Название товара", "Цена товара" и "Количество" функционально зависят от "Номера заказа", но не зависят от части первичного ключа, который состоит из "Номера заказа" и "Название товара".

Чтобы привести эту таблицу к 2НФ, мы можем выделить повторяющиеся атрибуты в отдельную таблицу "Товары" с атрибутами "Название товара" и "Цена товара":  
Таблица "Заказы":

| Номер заказа |

|--------------|

| 001 |

| 002 |

Таблица "Товары":

| Название товара | Цена товара |

|-----------------|-------------|

| Кофе | 100 |

| Чай | 50 |

Таблица "Товары в заказах":

| Номер заказа | Название товара | Количество |

|--------------|-----------------|------------|

| 001 | Кофе | 2 |

| 001 | Чай | 3 |

| 002 | Кофе | 1 |

| 002 | Чай | 2 |

Теперь атрибуты "Название товара" и "Цена товара" зависят только от части первичного ключа, и отношение находится во второй нормальной форме.

1. Дайте определения следующих понятий «нормализация», «декомпозиция отношений», «избыточность отношений». Поясните третью нормальную форму (3НФ). Опишите способы ее построения. Приведите примеры.

Нормализация в базах данных — это процесс организации структуры базы данных для уменьшения избыточности данных и обеспечения их целостности. Этот процесс включает в себя разделение таблиц на более мелкие и логически связанные части.

Декомпозиция отношений — это процесс разбиения таблиц на более мелкие и более нормализованные отношения для улучшения структуры базы данных и уменьшения избыточности данных.

Избыточность отношений возникает, когда в базе данных есть избыточная или повторяющаяся информация, что приводит к нежелательным дубликатам данных и увеличивает вероятность ошибок.

Третья нормальная форма (3НФ):

Отношение находится в третьей нормальной форме, если оно находится во второй нормальной форме и каждый его неключевой атрибут не зависит от других неключевых атрибутов, то есть не имеет транзитивных функциональных зависимостей.

Способы построения 3НФ:

* Устранение транзитивных зависимостей: если в отношении есть транзитивные зависимости, они должны быть устранены путем выделения атрибутов, на которые они зависят, в отдельные таблицы.

Пример:

Предположим, у нас есть таблица "Заказы" с атрибутами "Номер заказа", "ID товара", "Название товара" и "Категория товара":

| Номер заказа | ID товара | Название товара | Категория товара |

|--------------|-----------|-----------------|------------------|

| 001 | 101 | Кофе | Горячие напитки |

| 001 | 102 | Чай | Горячие напитки |

| 002 | 101 | Кофе | Горячие напитки |

| 002 | 102 | Чай | Горячие напитки |

В этой таблице атрибут "Категория товара" зависит от "Название товара", а не от "ID товара" или "Номер заказа". Это приводит к транзитивной зависимости, так как "Название товара" определяет "Категорию товара".

Чтобы привести эту таблицу к 3НФ, мы можем выделить атрибуты "Название товара" и "Категория товара" в отдельную таблицу "Товары":

Таблица "Заказы":

| Номер заказа | ID товара |

|--------------|-----------|

| 001 | 101 |

| 001 | 102 |

| 002 | 101 |

| 002 | 102 |

Таблица "Товары":

| ID товара | Название товара | Категория товара |

|-----------|-----------------|------------------|

| 101 | Кофе | Горячие напитки |

| 102 | Чай | Горячие напитки |

Теперь "Категория товара" функционально зависит только от "Название товара", и отношение находится в третьей нормальной форме.

1. Дайте определение понятия «реляционная алгебра». Перечислите группы операций реляционной алгебры Кодда и специальные операции, приведите примеры каждой группы.

Реляционная алгебра — это формальный язык для манипулирования реляционными данными, который был предложен Эдгаром Коддом. Она представляет собой набор операций, которые могут быть применены к отношениям (таблицам) в реляционных базах данных для выполнения запросов и получения нужной информации.

Группы операций реляционной алгебры Кодда:

* Операции выбора (Selection): позволяют выбирать строки из таблицы, удовлетворяющие определенному условию.
* Операции проекции (Projection): позволяют выбирать определенные столбцы из таблицы.
* Операции объединения (Union): позволяют объединять два отношения (таблицы), которые имеют одинаковую структуру, чтобы получить новое отношение, содержащее все строки из обоих отношений.
* Операции пересечения (Intersection): позволяют получить общие строки из двух отношений.
* Операции разности (Difference): позволяют получить строки из одного отношения, которые отсутствуют в другом.

Специальные операции:

* Переименование (Renaming): позволяет переименовать атрибуты (столбцы) в отношении.
* Соединение (Join): позволяет объединить два отношения по определенному условию.

1. Перечислите команды в общем синтаксисе оператора SELECT. Охарактеризуйте следующие конструкции раздела SELECT: ключевое слово ALL, отличие от DISTINCT, арифметические операции, функции, оператор «\*» (звездочка), подписи полей, полную идентификацию поля, вычисляемые поля. Приведите примеры использования конструкций.

Общий синтаксис оператора SELECT в SQL:

SELECT [ALL | DISTINCT] выражение1 [, выражение2, ...]

FROM таблица1 [, таблица2, ...]

[WHERE условие]

[GROUP BY выражение [, ...]]

[HAVING условие]

[ORDER BY выражение [ASC | DESC] [, ...]];

Характеристика конструкций раздела SELECT:

* Ключевое слово ALL: Ключевое слово ALL используется для возврата всех строк, включая дубликаты. Это поведение по умолчанию.

Пример: SELECT ALL имя FROM сотрудники;

* Отличие от DISTINCT: Ключевое слово DISTINCT используется для удаления дубликатов строк из результата запроса.

Пример: SELECT DISTINCT имя FROM сотрудники;

* Арифметические операции: В операторе SELECT можно использовать арифметические операции для выполнения вычислений на значениях столбцов.

Пример: SELECT (зарплата + премия) AS общая\_зарплата FROM сотрудники;

* Функции: Оператор SELECT позволяет использовать различные встроенные функции для обработки данных, такие как функции агрегации (SUM, AVG, COUNT и т. д.), функции работы со строками, функции работы с датами и другие.

Пример: SELECT AVG(зарплата) AS средняя\_зарплата FROM сотрудники;

* Оператор «\*» (звездочка): Оператор «\*» используется для выбора всех столбцов из таблицы.

Пример: SELECT \* FROM сотрудники;

* Подписи полей: Подписи полей позволяют задать пользовательские имена для столбцов в результирующем наборе.

Пример: SELECT имя AS имя\_сотрудника, зарплата AS оклад FROM сотрудники;

* Полная идентификация поля: Если столбец имеет одинаковые имена в разных таблицах, полная идентификация поля указывает на конкретный столбец, добавляя имя таблицы или псевдоним перед именем столбца.

Пример:

SELECT сотрудники.имя, отделы.название AS название\_отдела

FROM сотрудники

JOIN отделы ON сотрудники.отдел\_id = отделы.id;

* Вычисляемые поля: Вычисляемые поля создаются в процессе выполнения запроса путем выполнения арифметических операций, конкатенации строк или применения функций к значениям столбцов.

Пример: SELECT имя, зарплата \* 1.1 AS повышенная\_зарплата FROM сотрудники;

1. Перечислите команды в общем синтаксисе оператора SELECT. Охарактеризуйте раздел WHERE (операторы BETWEEN, LIKE) в общем синтаксисе оператора SELECT. Приведите примеры использования операторов в разделе WHERE.

Общий синтаксис оператора SELECT в SQL:

SELECT [ALL | DISTINCT] выражение1 [, выражение2, ...]

FROM таблица1 [, таблица2, ...]

[WHERE условие]

[GROUP BY выражение [, ...]]

[HAVING условие]

[ORDER BY выражение [ASC | DESC] [, ...]];

Раздел WHERE в операторе SELECT используется для фильтрации строк, которые соответствуют определенному условию.

Операторы в разделе WHERE:

* Оператор BETWEEN: Оператор BETWEEN используется для выбора строк, значения которых находятся в заданном диапазоне.

Пример: SELECT \* FROM сотрудники WHERE возраст BETWEEN 25 AND 40;

* Оператор LIKE: Оператор LIKE используется для выполнения поиска строк, которые содержат определенный шаблон. Можно использовать специальные символы, такие как % (заменяет любую последовательность символов) и \_ (заменяет один символ).

Пример: SELECT \* FROM сотрудники WHERE имя LIKE 'Иван%';

1. Перечислите команды в общем синтаксисе оператора SELECT. Охарактеризуйте раздел сортировки данных в общем синтаксисе оператора SELECT. Перечислите команды различных вариантов сортировки. Приведите примеры использования раздела сортировки данных в общем синтаксисе оператора SELECT.

В общем синтаксисе оператора SELECT включаются следующие элементы:

* Ключевое слово SELECT.
* Список столбцов, которые требуется выбрать, разделенных запятыми. Может использоваться символ \* для выбора всех столбцов.
* Ключевое слово FROM, за которым следует имя таблицы или представления, откуда извлекаются данные.
* Необязательный раздел WHERE для фильтрации строк по определенному условию.
* Необязательный раздел ORDER BY для сортировки результатов запроса.

Раздел сортировки данных (ORDER BY) в общем синтаксисе оператора SELECT позволяет упорядочить результаты запроса по одному или нескольким столбцам. Синтаксис выглядит следующим образом:

SELECT column1, column2, ...

FROM table\_name

[WHERE condition]

ORDER BY column1 [ASC|DESC], column2 [ASC|DESC], ...

Команды различных вариантов сортировки включают в себя:

ASC: Сортировка по возрастанию (по умолчанию).

DESC: Сортировка по убыванию.

Примеры использования раздела сортировки данных:

Простая сортировка по одному столбцу:

SELECT first\_name, last\_name

FROM employees

ORDER BY last\_name ASC;

Сортировка по нескольким столбцам:  
SELECT product\_name, unit\_price

FROM products

ORDER BY unit\_price DESC, product\_name ASC;

Сортировка с использованием функции:

SELECT customer\_id, order\_date

FROM orders

ORDER BY DATE(order\_date) DESC;

1. Перечислите команды в общем синтаксисе оператора SELECT. Охарактеризуйте разделы группировки данных и отбора групп в общем синтаксисе оператора SELECT. Приведите примеры использования разделов группировки данных и отбора групп в общем синтаксисе оператора SELECT.

В общем синтаксисе оператора SELECT включаются следующие элементы:

* Ключевое слово SELECT.
* Список столбцов, которые требуется выбрать, разделенных запятыми. Может использоваться символ \* для выбора всех столбцов.
* Ключевое слово FROM, за которым следует имя таблицы или представления, откуда извлекаются данные.
* Необязательный раздел WHERE для фильтрации строк по определенному условию.
* Необязательный раздел GROUP BY для группировки строк по значениям определенных столбцов.
* Необязательный раздел HAVING для отбора групп на основе агрегированных данных.

Разделы группировки данных и отбора групп в общем синтаксисе оператора SELECT позволяют агрегировать данные по определенным критериям.

* GROUP BY используется для группировки строк по значениям определенных столбцов.
* HAVING используется для фильтрации групп, которые удовлетворяют определенному условию.

Синтаксис выглядит следующим образом:

SELECT column1, column2, ...

FROM table\_name

[WHERE condition]

GROUP BY column1, column2, ...

HAVING condition;

Примеры использования разделов группировки данных и отбора групп:

* Пример группировки данных по столбцу и отбора групп с помощью HAVING:

SELECT department\_id, COUNT(\*) AS employee\_count

FROM employees

GROUP BY department\_id

HAVING COUNT(\*) > 5;

Этот запрос выбирает department\_id и подсчитывает количество сотрудников в каждом отделе. Затем используется условие HAVING, чтобы выбрать только те группы, в которых количество сотрудников больше 5.

* Пример группировки данных по нескольким столбцам:

SELECT department\_id, job\_id, AVG(salary) AS avg\_salary

FROM employees

GROUP BY department\_id, job\_id;

Этот запрос группирует данные по department\_id и job\_id, а затем вычисляет среднюю зарплату (avg\_salary) для каждой уникальной комбинации department\_id и job\_id.

1. Перечислите команды в общем синтаксисе оператора SELECT. Охарактеризуйте раздел FROM (соединение таблиц): выборка из одной таблицы, выборка из двух таблиц, выборка из трех таблиц с использованием оператора JOIN. Перечислите и поясните виды оператора JOIN. Приведите примеры использования оператора JOIN в общем синтаксисе оператора SELECT.  
   В общем синтаксисе оператора SELECT включаются следующие элементы:

* Ключевое слово SELECT.
* Список столбцов, которые требуется выбрать, разделенных запятыми. Может использоваться символ \* для выбора всех столбцов.
* Ключевое слово FROM, за которым следует имя таблицы или представления, откуда извлекаются данные.
* Необязательный раздел WHERE для фильтрации строк по определенному условию.
* Необязательный раздел GROUP BY для группировки строк по значениям определенных столбцов.
* Необязательный раздел HAVING для отбора групп на основе агрегированных данных.
* Необязательный раздел ORDER BY для сортировки результатов запроса.

Раздел FROM (соединение таблиц) позволяет выбирать данные из одной, двух или более таблиц. Обычно это делается с помощью оператора JOIN.

* Выборка из одной таблицы: Простой случай, когда данные извлекаются только из одной таблицы без каких-либо дополнительных соединений.
* Выборка из двух таблиц: Данные извлекаются из двух таблиц с помощью оператора JOIN. Обычно это делается для объединения информации из разных таблиц на основе некоторого общего значения.
* Выборка из трех таблиц с использованием оператора JOIN: при необходимости выборки из трех таблиц используются соединения JOIN для связывания строк из всех трех таблиц по определенным условиям.

Виды оператора JOIN:

* INNER JOIN: возвращает только строки, имеющие соответствующие значения в обеих таблицах.
* LEFT JOIN (или LEFT OUTER JOIN): возвращает все строки из левой таблицы и соответствующие строки из правой таблицы. Если для строки из левой таблицы нет соответствующих строк в правой таблице, то для них будут использоваться NULL значения.
* RIGHT JOIN (или RIGHT OUTER JOIN): возвращает все строки из правой таблицы и соответствующие строки из левой таблицы. Аналогично LEFT JOIN, если для строки из правой таблицы нет соответствующих строк в левой таблице, то для них будут использоваться NULL значения.
* FULL JOIN (или FULL OUTER JOIN): возвращает строки, если они есть в любой из таблиц. Если для строки из одной таблицы нет соответствующей строки в другой таблице, то для них будут использоваться NULL значения.

Примеры использования оператора JOIN в общем синтаксисе оператора SELECT:

-- Пример INNER JOIN

SELECT orders.order\_id, customers.customer\_name

FROM orders

INNER JOIN customers ON orders.customer\_id = customers.customer\_id;

-- Пример LEFT JOIN

SELECT orders.order\_id, customers.customer\_name

FROM orders

LEFT JOIN customers ON orders.customer\_id = customers.customer\_id;

-- Пример RIGHT JOIN

SELECT orders.order\_id, customers.customer\_name

FROM orders

RIGHT JOIN customers ON orders.customer\_id = customers.customer\_id;

-- Пример FULL JOIN

SELECT orders.order\_id, customers.customer\_name

FROM orders

FULL JOIN customers ON orders.customer\_id = customers.customer\_id;

1. Перечислите команды в общем синтаксисе оператора SELECT. Охарактеризуйте подзапросы. Перечислите логические операторы, применяемые в подзапросах. Приведите примеры использования подзапросов и примеры применения логических операторов в подзапросах.

В общем синтаксисе оператора SELECT включаются следующие элементы:

* Ключевое слово SELECT.
* Список столбцов, которые требуется выбрать, разделенных запятыми. Может использоваться символ \* для выбора всех столбцов.
* Ключевое слово FROM, за которым следует имя таблицы или представления, откуда извлекаются данные.
* Необязательный раздел WHERE для фильтрации строк по определенному условию.
* Необязательный раздел GROUP BY для группировки строк по значениям определенных столбцов.
* Необязательный раздел HAVING для отбора групп на основе агрегированных данных.
* Необязательный раздел ORDER BY для сортировки результатов запроса.

Подзапросы – это запросы SQL, вложенные в основной запрос. Они могут использоваться в различных частях основного запроса, таких как WHERE, HAVING, SELECT или FROM, и позволяют выполнить более сложные операции обработки данных.

Логические операторы, применяемые в подзапросах, включают в себя:

* IN: проверяет, содержится ли значение в результирующем наборе подзапроса.
* NOT IN: проверяет, отсутствует ли значение в результирующем наборе подзапроса.
* ANY|SOME: проверяет, удовлетворяет ли значение хотя бы одному из значений, возвращенным подзапросом.
* ALL: проверяет, удовлетворяет ли значение всем значениям, возвращенным подзапросом.
* EXISTS: проверяет, существуют ли строки в результирующем наборе подзапроса.

Примеры использования подзапросов:

Пример использования подзапроса в разделе WHERE:

SELECT \*

FROM customers

WHERE customer\_id IN (SELECT customer\_id FROM orders WHERE order\_date > '2023-01-01');

Пример использования подзапроса в разделе SELECT:

SELECT product\_name,

(SELECT AVG(unit\_price) FROM order\_details WHERE order\_details.product\_id = products.product\_id) AS avg\_price

FROM products;

Примеры применения логических операторов в подзапросах:

-- Пример использования оператора IN

SELECT \* FROM employees WHERE department\_id IN (SELECT department\_id FROM departments WHERE location\_id = 1700);

-- Пример использования оператора EXISTS

SELECT \* FROM customers WHERE EXISTS (SELECT \* FROM orders WHERE orders.customer\_id = customers.customer\_id);

1. Перечислите команды в общем синтаксисе оператора SELECT. Опишите использование встроенных функций СУБД SQL Server в обработке данных: математические (не менее пяти), агрегатные (не менее пяти), функции работы с датами (не менее пяти), функции работы со строками (не менее трех). Приведите примеры использования математических функций, функций работы со строками, функций работы с датами в запросах.

В общем синтаксисе оператора SELECT в SQL Server включаются следующие элементы:

SELECT column1, column2, ...

FROM table\_name

[WHERE condition]

[GROUP BY columns]

[HAVING condition]

[ORDER BY columns];

Математические функции:

* ABS(): Возвращает абсолютное значение числа.
* ROUND(): Округляет число до заданного количества знаков.
* CEILING(): Округляет число до ближайшего большего целого.
* FLOOR(): Округляет число до ближайшего меньшего целого.
* POWER(): Возводит число в указанную степень.

Примеры использования математических функций:

SELECT ABS(-10) AS absolute\_value;

SELECT ROUND(3.14159, 2) AS rounded\_value;

SELECT CEILING(4.2) AS ceiling\_value;

SELECT FLOOR(4.8) AS floor\_value;

SELECT POWER(2, 3) AS power\_value;

Агрегатные функции:

* SUM(): Вычисляет сумму значений в столбце.
* AVG(): Вычисляет среднее значение в столбце.
* COUNT(): Подсчитывает количество строк.
* MIN(): Находит минимальное значение в столбце.
* MAX(): Находит максимальное значение в столбце.

Примеры использования агрегатных функций:

SELECT SUM(salary) AS total\_salary FROM employees;

SELECT AVG(age) AS average\_age FROM students;

SELECT COUNT(\*) AS total\_records FROM orders;

SELECT MIN(price) AS min\_price FROM products;

SELECT MAX(quantity) AS max\_quantity FROM inventory;

Функции работы с датами:

* GETDATE(): Возвращает текущую дату и время.
* DATEADD(): Добавляет определенное количество времени к заданной дате.
* DATEDIFF(): Вычисляет разницу между двумя датами.
* YEAR(): Извлекает год из даты.
* MONTH(): Извлекает месяц из даты.

Примеры использования функций работы с датами:

SELECT GETDATE() AS current\_date\_time;

SELECT DATEADD(day, 7, '2024-03-20') AS future\_date;

SELECT DATEDIFF(day, '2023-01-01', '2023-12-31') AS days\_difference;

SELECT YEAR('2023-09-15') AS year\_value;

SELECT MONTH('2023-09-15') AS month\_value;

Функции работы со строками:

* LEN(): Возвращает длину строки.
* LOWER(): Преобразует все символы строки в нижний регистр.
* UPPER(): Преобразует все символы строки в верхний регистр.

Примеры использования функций работы со строками:

SELECT LEN('Hello') AS string\_length;

SELECT LOWER('HELLO') AS lowercase\_string;

SELECT UPPER('hello') AS uppercase\_string;

1. Дайте определение понятия «пакет», перечислите операторы языка SQL для обновления записей, удаление записей, вставка записей. Назовите команду, определяющую начало пакета. Опишите синтаксис операторов языка SQL для обновления записей, удаление записей, вставка записей, приведите примеры, демонстрирующие работу указанных инструкций. Поясните применение комментарий в SQL Server: однострочный и многострочный, их варианты записи.

В контексте баз данных и SQL, "пакет" обычно относится к группе SQL-инструкций, которые выполняются вместе как единое целое. Это может быть хранимая процедура, транзакция или другой блок кода, который обычно используется для выполнения определенной задачи или операции.

Операторы языка SQL для обновления, удаления и вставки записей:

* UPDATE: используется для обновления записей в таблице.
* DELETE: используется для удаления записей из таблицы.
* INSERT: используется для вставки новых записей в таблицу.

Команда, определяющая начало пакета: В SQL Server начало пакета обычно определяется ключевым словом BEGIN.

Синтаксис операторов языка SQL для обновления, удаления и вставки записей:

* UPDATE:

UPDATE table\_name

SET column1 = value1, column2 = value2, ...

WHERE condition;

* DELETE:

DELETE FROM table\_name

WHERE condition;

* INSERT:

INSERT INTO table\_name (column1, column2, ...)

VALUES (value1, value2, ...);

Применение комментариев в SQL Server:

Комментарии используются для добавления пояснений к коду SQL, чтобы облегчить его понимание и поддержку.

* Однострочный комментарий: В SQL Server однострочные комментарии начинаются с двух дефисов ‘--’ и применяются до конца строки.

-- Это однострочный комментарий

SELECT \* FROM employees; -- Этот комментарий относится к запросу SELECT

* Многострочный комментарий: Многострочные комментарии начинаются с /\* и заканчиваются на \*/. Они могут занимать несколько строк.

/\*

Это

многострочный

комментарий

\*/

SELECT \* FROM orders;

* Также можно использовать вариант однострочного комментария с ключевым словом REM:

REM Этот комментарий относится к запросу

SELECT \* FROM customers;

1. Дайте определения следующих понятий «владелец базы данных». Перечислите виды объектов базы данных в SQL Server. Приведите примеры. Поясните полный синтаксис инструкции CREATE DATABASE, опишите ее основные параметры.

Владелец базы данных – относится к пользователю или роли, который имеет наивысшие привилегии доступа к базе данных и ее объектам. Владелец базы данных обычно имеет полный контроль над базой данных, включая возможность создавать, изменять и удалять объекты, а также назначать права доступа другим пользователям.

Виды объектов базы данных в SQL Server:

* Таблицы
* Представления
* Индексы
* Хранимые процедуры
* Триггеры
* Функции
* Схемы
* Пользователи и роли
* Полнотекстовые каталоги
* И другие.

Примеры:

* Таблицы: employees, products, orders.
* Представления: sales\_report, customer\_details.
* Индексы: idx\_order\_date, idx\_customer\_id.
* Хранимые процедуры: usp\_update\_employee, usp\_insert\_order.
* Триггеры: trg\_employee\_audit, trg\_order\_log.
* Функции: fn\_get\_order\_total, fn\_format\_phone\_number.
* Схемы: dbo, sales, hr.
* Пользователи и роли: user\_sales, role\_admin.

Полный синтаксис инструкции CREATE DATABASE и его основные параметры:

CREATE DATABASE database\_name

[ON

[PRIMARY]

( NAME = logical\_file\_name,

FILENAME = 'path\_to\_data\_file',

SIZE = size,

MAXSIZE = max\_size,

FILEGROWTH = growth\_increment )

[, ...n ]

]

[LOG ON

( NAME = logical\_log\_file\_name,

FILENAME = 'path\_to\_log\_file',

SIZE = size,

MAXSIZE = max\_size,

FILEGROWTH = growth\_increment )

[, ...n ]

]

[ COLLATE collation\_name ]

[ ... ]

Основные параметры инструкции CREATE DATABASE:

* database\_name: Имя новой базы данных.
* ON: указывает местоположение и размер файлов данных.
* PRIMARY: определяет основной файл данных.
* NAME: Логическое имя файла данных.
* FILENAME: Путь к файлу данных на диске.
* SIZE: Начальный размер файла данных.
* MAXSIZE: Максимальный размер файла данных.
* FILEGROWTH: Увеличение файла данных при необходимости.
* LOG ON: указывает местоположение и размер файла журнала транзакций.
* NAME: Логическое имя файла журнала.
* FILENAME: Путь к файлу журнала на диске.
* SIZE: Начальный размер файла журнала.
* MAXSIZE: Максимальный размер файла журнала.
* FILEGROWTH: Увеличение файла журнала при необходимости.
* COLLATE: устанавливает порядок сортировки для базы данных.

Пример использования инструкции CREATE DATABASE:

CREATE DATABASE SalesDB

ON PRIMARY

(

NAME = SalesDB\_data,

FILENAME = 'C:\Data\SalesDB\_data.mdf',

SIZE = 100MB,

MAXSIZE = UNLIMITED,

FILEGROWTH = 20MB

)

LOG ON

(

NAME = SalesDB\_log,

FILENAME = 'C:\Data\SalesDB\_log.ldf',

SIZE = 50MB,

MAXSIZE = 100MB,

FILEGROWTH = 10MB

)

1. Назовите команду создания таблицы в базе данных. Поясните полный синтаксис инструкции создания базы данных, разъясните ее основные атрибуты в спецификации столбца. Приведите примеры, поясняющие работу с инструкцией создания таблиц.

Вот пример базового синтаксиса создания таблицы в SQL:

CREATE TABLE table\_name (

column1 datatype [constraint],

column2 datatype [constraint],

...

);

Где:

CREATE TABLE - ключевое слово для создания таблицы.

table\_name - имя новой таблицы.

column1, column2, ... - названия столбцов в таблице.

datatype - тип данных для каждого столбца.

constraint - ограничения или свойства столбца (например, PRIMARY KEY, FOREIGN KEY, NOT NULL и т. д.).

1. Дайте определения следующих понятий «проверочное ограничение», «потенциальный ключ», «первичный ключ», «внешний ключ», «дочерняя таблица», «родительская таблица». Назовите ограничения для обеспечения целостности в инструкции создания таблиц базы данных, перечислите их преимущества. Перечислите категории декларативных ограничений целостности, поясните их синтаксис. Поясните синтаксис создания внешнего ключа в инструкции создания таблицы.

* Проверочное ограничение (Check Constraint) – это ограничение, которое определяет условия, которым должны соответствовать данные в столбце.

Например, ограничение может быть наложено на столбец "возраст" таким образом, чтобы он не превышал определенное значение.

Преимущества: обеспечивает целостность данных, предотвращает вставку неправильных значений.

* Потенциальный ключ (Candidate Key) – это уникальный столбец или группа столбцов в таблице, которые могут однозначно идентифицировать каждую запись.

Каждый потенциальный ключ может быть потенциальным первичным ключом.

Например, в таблице пользователей, может быть, несколько потенциальных ключей, таких как "ID пользователя" или "Электронная почта".

* Первичный ключ (Primary Key) – это уникальный столбец или группа столбцов в таблице, который однозначно идентифицирует каждую запись.

Он гарантирует уникальность значений и предотвращает вставку дубликатов.

Преимущества: обеспечивает уникальность данных, ускоряет поиск записей.

* Внешний ключ (Foreign Key) – это столбец или группа столбцов в таблице, который связан с первичным ключом другой таблицы.

Он создает ссылочную целостность данных между таблицами.

Например, в таблице заказов столбец "ID пользователя" может быть внешним ключом, ссылаясь на первичный ключ таблицы пользователей.

Внешний ключ обеспечивает целостность связанных данных и предотвращает нарушения ссылочной целостности.

* Дочерняя таблица (Child Table) – это таблица, содержащая внешние ключи, ссылки на первичные ключи родительской таблицы.

Например, в таблице "заказы" "ID пользователя" является внешним ключом, ссылаясь на таблицу "пользователи".

Родительская таблица (Parent Table) – это таблица, содержащая первичные ключи, на которые ссылаются внешние ключи других таблиц.

Например, таблица "пользователи" может быть родительской таблицей для таблицы "заказы".

Ограничения для обеспечения целостности данных в инструкции создания таблиц базы данных включают:

* PRIMARY KEY
* FOREIGN KEY
* UNIQUE
* CHECK

Категории декларативных ограничений целостности включают:

* Ограничения уникальности (UNIQUE)
* Ограничения на внешние ключи (FOREIGN KEY)
* Ограничения на проверку (CHECK)

Синтаксис создания внешнего ключа в инструкции создания таблицы:

CREATE TABLE child\_table (

column1 datatype,

column2 datatype,

FOREIGN KEY (column1) REFERENCES parent\_table(primary\_key\_column)

);

1. Дайте определения следующих понятий «ссылочная целостность», «домен», «псевдоним типа данных». Назовите четыре правила, определяющие то, как SQL Server может реагировать на попытку обновления значений первичного ключа, вызывающие несогласованность в соответствующем внешнем ключе. Поясните синтаксис инструкции создания псевдонимного типа данных. Приведите примеры доменов, примеры создания псевдонимного типа данных.

* Ссылочная целостность (Referential Integrity) – это свойство базы данных, которое гарантирует согласованность данных между связанными таблицами через первичные и внешние ключи.

Она обеспечивает, что значение внешнего ключа всегда существует в связанной таблице или имеет значение NULL.

Если внешний ключ ссылается на первичный ключ, то его значение должно существовать в столбце первичного ключа родительской таблицы.

* Домен (Domain) – это множество допустимых значений для определенного атрибута в базе данных.

Он определяет тип данных, размер и другие ограничения для конкретного столбца в таблице.

Например, домен "электронной почты" может включать только строки, соответствующие стандарту электронной почты.

* Псевдоним типа данных (Data Type Alias) –это пользовательский определенный тип данных, который представляет собой альтернативное имя для существующего типа данных.

Он позволяет использовать более описательные и понятные имена для типов данных, что делает код более читаемым и понятным.

Псевдонимы типов данных могут использоваться для упрощения синтаксиса запросов и обеспечения стандартизации типов данных в базе данных.

Четыре правила, определяющие реакцию SQL Server на попытку обновления значений первичного ключа, вызывающие несогласованность в соответствующем внешнем ключе:

* CASCADE - Обновляет значения внешнего ключа, чтобы они соответствовали обновленным значениям первичного ключа.
* SET NULL - Устанавливает значения внешнего ключа в NULL, если значение первичного ключа обновляется на NULL.
* SET DEFAULT - Устанавливает значения внешнего ключа в значение по умолчанию, если значение первичного ключа обновляется на NULL.
* NO ACTION - Запрещает обновление значения первичного ключа, если это вызовет несогласованность с внешним ключом.

Синтаксис инструкции создания псевдонимного типа данных в SQL Server:

CREATE TYPE alias\_name FROM system\_data\_type;

Примеры доменов:

* Домен "телефонный номер": VARCHAR(15)
* Домен "адрес электронной почты": VARCHAR(255)

Примеры создания псевдонимного типа данных:

CREATE TYPE Phone AS VARCHAR(15);

CREATE TYPE Email AS VARCHAR(255);

1. Дайте определение понятия «индекс», назовите команду инструкцию создания индекса. Перечислите типы индексов, опишите их основное назначение. Опишите способы создания индекса в SQL Server: с использованием визуальной среды, с помощью инструкций языка SQL (пояснить полный синтаксис). Приведите примеры, демонстрирующие создание всех типов индексов с помощью инструкций языка SQL.

Индекс — это структура данных, используемая для ускорения поиска и извлечения записей из таблицы базы данных. Он содержит отсортированные значения столбца или набор столбцов таблицы, а также указатели на соответствующие строки таблицы. Индексы улучшают производительность запросов SELECT, но могут замедлить производительность запросов INSERT, UPDATE и DELETE, так как они должны быть обновлены при изменении данных.

Команда или инструкция создания индекса в SQL:

CREATE INDEX index\_name ON table\_name (column1, column2, ...);

Типы индексов:

* Кластеризованный индекс (Clustered Index):

Основное назначение - определение физического порядка хранения строк в таблице на основе значений индексируемого столбца.

В таблице может быть только один кластеризованный индекс.

Обычно используется для частого поиска и сортировки данных.

* Некластеризованный индекс (Non-Clustered Index):

Основное назначение - создание отдельной структуры данных, которая указывает на физическое расположение строк в таблице.

В таблице может быть несколько некластеризованных индексов.

Обычно используется для ускорения поиска данных по различным столбцам.

* Уникальный индекс (Unique Index):

Основное назначение - гарантировать уникальность значений в столбцах, на которые создан индекс.

Позволяет иметь только одно вхождение определенного значения в столбце.

* Полнотекстовый индекс (Full-Text Index):

Основное назначение - обеспечение полнотекстового поиска по текстовым столбцам в таблице.

Позволяет выполнить запросы, содержащие ключевые слова и фразы.

Способы создания индекса в SQL Server:

* С использованием визуальной среды:

В Microsoft SQL Server Management Studio (SSMS) можно создать индекс, выбрав таблицу, открыв контекстное меню и выбрав "New Index" (Новый индекс). Затем следует выбрать тип индекса и указать соответствующие столбцы.

* С помощью инструкций языка SQL:

Для создания индекса с помощью SQL инструкций необходимо использовать команду CREATE INDEX с указанием имени индекса, таблицы и столбца(ов), для которых создается индекс.

Примеры создания всех типов индексов с помощью инструкций языка SQL:

-- Кластеризованный индекс

CREATE CLUSTERED INDEX IX\_ClusteredIndex ON TableName (Column1);

-- Некластеризованный индекс

CREATE NONCLUSTERED INDEX IX\_NonClusteredIndex ON TableName (Column1);

-- Уникальный индекс

CREATE UNIQUE INDEX IX\_UniqueIndex ON TableName (Column1);

-- Полнотекстовый индекс

CREATE FULLTEXT INDEX ON TableName (Column1) KEY INDEX IX\_UniqueIndex;

1. Дайте определения следующих понятий «резервное копирование базы данных», «восстановление базы данных». Перечислите и поясните известные виды резервного копирования и восстановления базы данных. Назовите инструкции языка SQL для выполнения всех видов резервного копирования и восстановления базы данных.

Резервное копирование базы данных — это процесс создания копии всех или части данных базы данных с целью защиты от потери данных в случае аварии, сбоя оборудования, ошибок пользователя или других нежелательных событий. Резервное копирование обеспечивает возможность восстановления данных в случае их повреждения или утраты.

Восстановление базы данных — это процесс восстановления данных из резервной копии после их потери или повреждения. Восстановление может включать в себя восстановление всей базы данных или отдельных таблиц и объектов из копии.

Известные виды резервного копирования и восстановления базы данных:

* Полное резервное копирование (Full Backup):

Создает копию всех данных в базе данных.

Позволяет восстановить всю базу данных в случае ее полной потери или повреждения.

* Дифференциальное резервное копирование (Differential Backup):

Создает копию только тех данных, которые изменились с момента последнего полного резервного копирования.

Позволяет уменьшить время и объем копирования, по сравнению с полным копированием.

Позволяет восстановить базу данных до момента последнего полного резервного копирования, а затем применить дифференциальное копирование для восстановления измененных данных.

* Транзакционное резервное копирование (Transaction Log Backup):

Создает копию транзакционного журнала базы данных, который содержит информацию о всех транзакциях, выполняемых в базе данных.

Позволяет восстановить базу данных до определенного момента времени, включая изменения, сделанные после последнего полного или дифференциального резервного копирования.

Инструкции языка SQL для выполнения всех видов резервного копирования и восстановления базы данных:

* Полное резервное копирование:

BACKUP DATABASE database\_name TO disk = 'path\_to\_backup\_file';

Дифференциальное резервное копирование:

BACKUP DATABASE database\_name TO disk = 'path\_to\_backup\_file' WITH DIFFERENTIAL;

* Транзакционное резервное копирование:

BACKUP LOG database\_name TO disk = 'path\_to\_backup\_file';

* Восстановление базы данных из полного резервного копирования:

RESTORE DATABASE database\_name FROM disk = 'path\_to\_backup\_file' WITH REPLACE;

* Восстановление базы данных из дифференциального резервного копирования:

RESTORE DATABASE database\_name FROM disk = 'path\_to\_backup\_file' WITH NORECOVERY;

RESTORE DATABASE database\_name FROM disk = 'path\_to\_differential\_backup\_file' WITH RECOVERY;

* Восстановление базы данных с применением транзакционного журнала:

RESTORE DATABASE database\_name FROM disk = 'path\_to\_full\_backup\_file' WITH NORECOVERY;

RESTORE LOG database\_name FROM disk = 'path\_to\_transaction\_log\_backup\_file' WITH RECOVERY;

1. Дайте определение понятия «определенная пользователем функции (ОПФ)». Назовите инструкции создания, изменения, удаления, вызова определенной пользователем функции, поясните их основные параметры. Перечислите и опишите типы ОПФ.

Определенная пользователем функция (ОПФ) — это программный блок, который выполняет определенную операцию и возвращает результат. ОПФ позволяют абстрагировать повторяющиеся или сложные вычисления в базе данных, обеспечивая их повторное использование в различных запросах.

Инструкции для работы с определенными пользователем функциями в SQL:

* Создание ОПФ:

CREATE FUNCTION function\_name

( [parameter1, parameter2, ...] )

RETURNS return\_data\_type

AS

BEGIN

-- Тело функции

END;

Где:

function\_name - имя создаваемой функции.

parameter1, parameter2, ... - параметры функции (необязательно).

return\_data\_type - тип данных, возвращаемый функцией.

AS BEGIN ... END - тело функции, содержащее SQL-код для выполнения необходимых вычислений.

* Изменение ОПФ:

ALTER FUNCTION function\_name

( [parameter1, parameter2, ...] )

RETURNS return\_data\_type

AS

BEGIN

-- Измененное тело функции

END;

Удаление ОПФ:

DROP FUNCTION function\_name;

Вызов ОПФ:

SELECT function\_name(parameter1, parameter2, ...) AS result;

Типы ОПФ:

* Скалярные функции (Scalar Functions):

Возвращают одно значение на основе входных параметров.

Пример: функции, вычисляющие сумму, разность, произведение чисел и т. д.

* Табличные функции (Table-Valued Functions):

Возвращают таблицу результатов.

Могут быть встроенными функциями или функциями с параметрами.

Пример: функции, возвращающие список сотрудников, отфильтрованный по критериям.

* Многооператорные ОПФ (Multi-statement Functions):

Возвращают набор результатов, сгенерированных из последовательности операторов SQL.

Пример: функции, возвращающие список заказов для определенного клиента.

1. Дайте определение понятия «хранимая процедура». Назовите и опишите типы хранимых процедур. Поясните инструкции создания, удаления, изменения, вызова хранимой процедура, опишите их синтаксис. Приведите примеры создания, удаления, изменения, вызова хранимой процедуры.

Хранимая процедура — это набор предварительно скомпилированных инструкций SQL, которые хранятся в базе данных и могут быть вызваны для выполнения определенной операции или набора операций. Хранимые процедуры позволяют группировать и повторно использовать код, уменьшая повторяемость и упрощая обслуживание приложений.

Типы хранимых процедур:

* Процедуры без возвращаемого значения:

Выполняют набор операций без возврата результата.

Могут использоваться для изменения данных в базе данных или выполнения других операций.

* Функции:

Возвращают результат операции или набора операций.

Могут использоваться для выполнения вычислений и возврата результата в запрос.

Инструкции для работы с хранимыми процедурами в SQL:

* Создание хранимой процедуры:

CREATE PROCEDURE procedure\_name

@parameter1 data\_type,

@parameter2 data\_type,

...

AS

BEGIN

-- Тело процедуры

END;

Где:

procedure\_name - имя создаваемой процедуры.

@parameter1, @parameter2, ... - параметры процедуры.

data\_type - тип данных параметра.

AS BEGIN ... END - тело процедуры, содержащее SQL-код для выполнения необходимых операций.

* Удаление хранимой процедуры:

DROP PROCEDURE procedure\_name;

Изменение хранимой процедуры:

ALTER PROCEDURE procedure\_name

@parameter1 new\_data\_type,

@parameter2 new\_data\_type,

...

AS

BEGIN

-- Измененное тело процедуры

END;

* Вызов хранимой процедуры:

EXEC procedure\_name @parameter1, @parameter2, ...;

Пример создания хранимой процедуры без возвращаемого значения:

CREATE PROCEDURE InsertEmployee

@name NVARCHAR(50),

@age INT,

@department NVARCHAR(50)

AS

BEGIN

INSERT INTO Employees (Name, Age, Department)

VALUES (@name, @age, @department);

END;

Пример удаления хранимой процедуры:

DROP PROCEDURE InsertEmployee;

Пример изменения хранимой процедуры:

ALTER PROCEDURE InsertEmployee

@name NVARCHAR(50),

@age INT,

@department NVARCHAR(50),

@position NVARCHAR(50)

AS

BEGIN

INSERT INTO Employees (Name, Age, Department, Position)

VALUES (@name, @age, @department, @position);

END;

Пример вызова хранимой процедуры:

EXEC InsertEmployee 'John Doe', 30, 'IT', 'Developer';

1. Дайте определение понятия «триггер». Назовите и поясните основные виртуальные таблицы, с которыми работает триггер. Назовите инструкции создания, удаления и изменения триггера, опишите их синтаксис. Приведите примеры создания, удаления, изменения триггера.

Триггер — это специальный тип хранимого объекта базы данных, который автоматически активируется при выполнении определенного события (например, вставка, обновление или удаление записи в таблице). Триггеры используются для автоматизации дополнительных операций или проверок, связанных с изменением данных в базе данных.

Основные виртуальные таблицы, с которыми работает триггер:

* Inserted - виртуальная таблица, доступная для триггера после выполнения операции вставки (INSERT). Она содержит новые строки, которые были вставлены в таблицу.
* Deleted - виртуальная таблица, доступная для триггера после выполнения операции удаления (DELETE). Она содержит удаленные строки из таблицы.
* Updated - виртуальная таблица, доступная для триггера после выполнения операции обновления (UPDATE). Она содержит старые и новые значения строк, которые были изменены.

Инструкции для работы с триггерами в SQL:

* Создание триггера:

CREATE TRIGGER trigger\_name

ON table\_name

FOR INSERT, UPDATE, DELETE

AS

BEGIN

-- Тело триггера

END;

Где:

trigger\_name - имя создаваемого триггера.

table\_name - имя таблицы, для которой создается триггер.

FOR INSERT, UPDATE, DELETE - указывает, на какие операции триггер должен реагировать.

AS BEGIN ... END - тело триггера, содержащее SQL-код для выполнения необходимых действий.

* Удаление триггера:

DROP TRIGGER trigger\_name;

* Изменение триггера:

Изменение триггера невозможно в большинстве систем управления базами данных (СУБД). Для внесения изменений, требуется удалить существующий триггер и создать новый с обновленным функционалом.

1. Дайте определение понятия «представление (виртуальная таблица)». Назовите инструкции создания, удаления и изменения представления, опишите их синтаксис. Поясните применение инструкций DML (SELECT, INSERT, UPDATE, DELETE) по отношению к представлениям, перечислите ограничения на их применение. Приведите примеры создания, удаления, изменения представления.

Представление (виртуальная таблица) — это набор данных, который выглядит и работает как обычная таблица в базе данных, но фактически представляет собой результат выполнения запроса к одной или нескольким реальным таблицам. Представления используются для упрощения работы с данными, скрытия сложных запросов или ограничения доступа к определенным данным для пользователей.

Инструкции для работы с представлениями в SQL:

* Создание представления:

CREATE VIEW view\_name AS

SELECT column1, column2, ...

FROM table\_name

WHERE condition;

Где:

view\_name - имя создаваемого представления.

column1, column2, ... - столбцы, которые будут включены в представление.

table\_name - имя таблицы или таблиц, к которым обращается представление.

condition - условие, определяющее фильтрацию данных.

* Удаление представления:

DROP VIEW view\_name;

* Изменение представления:

ALTER VIEW view\_name AS

SELECT new\_column1, new\_column2, ...

FROM new\_table\_name

WHERE new\_condition;

Применение инструкций DML (SELECT, INSERT, UPDATE, DELETE) по отношению к представлениям:

* SELECT: Представления можно использовать для выполнения запросов на выборку данных, как если бы они были обычными таблицами.
* INSERT: Некоторые представления можно использовать для вставки данных. Однако, в большинстве СУБД представления, основанные на сложных запросах, не могут быть обновлены через операцию INSERT.
* UPDATE: Представления, которые базируются на нескольких таблицах или имеют агрегирующие функции, обычно не могут быть обновлены через операцию UPDATE.
* DELETE: так же, как и операция UPDATE, представления, которые базируются на нескольких таблицах или имеют агрегирующие функции, обычно не могут быть обновлены через операцию DELETE.

1. Дайте определение понятия «транзакция». Перечислите свойства транзакции (ACID). Назовите и поясните режимы работы с транзакциями. Опишите инструкции языка T-SQL для работы с транзакциями в различных режимах. Приведите примеры.

Транзакция — это логическая единица работы, которая состоит из одного или нескольких SQL операторов, которые должны быть выполнены как единое целое. Транзакции в базах данных обеспечивают атомарность, согласованность, изолированность и долговечность операций (свойства ACID).

Свойства ACID транзакции:

* Атомарность (Atomicity): это означает, что транзакция является неделимой операцией. Все операции в транзакции либо выполняются полностью, либо не выполняются совсем. Не может быть частично выполненных операций.
* Согласованность (Consistency): гарантирует, что выполнение транзакции переводит базу данных из одного согласованного состояния в другое согласованное состояние. То есть все правила целостности данных должны быть соблюдены.
* Изолированность (Isolation): обеспечивает то, что выполнение транзакции не зависит от других одновременно выполняемых транзакций. Транзакция должна быть изолирована от других транзакций до того момента, пока она не завершится.
* Долговечность (Durability): гарантирует, что изменения, внесенные в базу данных в результате успешного завершения транзакции, сохранятся навсегда, даже в случае сбоя системы.

Режимы работы с транзакциями:

* Автоматический режим (Auto-commit mode): Каждая инструкция SQL автоматически считается отдельной транзакцией. Система автоматически фиксирует транзакцию после ее выполнения или откатывает изменения в случае возникновения ошибки.
* Ручной режим (Manual mode): Разработчик самостоятельно управляет началом, фиксацией и отменой транзакций.

Инструкции T-SQL для работы с транзакциями в различных режимах:

* BEGIN TRANSACTION: начинает новую транзакцию.
* COMMIT TRANSACTION: фиксирует изменения, выполненные в рамках текущей транзакции.
* ROLLBACK TRANSACTION: откатывает все изменения, выполненные в рамках текущей транзакции.
* SAVE TRANSACTION: создаёт точку сохранения, к которой можно вернуться позже с помощью команды ROLLBACK TRANSACTION.

Пример использования транзакций в T-SQL:

BEGIN TRANSACTION;

UPDATE Employees SET Salary = Salary \* 1.1 WHERE Department = 'IT';

INSERT INTO AuditLog (Action, TableName, DateTime) VALUES ('Salary update', 'Employees', GETDATE());

COMMIT TRANSACTION;

1. Дайте определения следующих понятий «архитектура клиент-сервер», «двух- и трехуровневая архитектура», «файл серверная архитектура». Опишите каждую архитектуру. Поясните достоинства и недостатки каждой архитектуры.

Архитектура клиент-сервер — это модель взаимодействия между клиентом и сервером в распределенной информационной системе. В этой архитектуре клиенты обращаются к серверу для получения доступа к ресурсам, обработки данных или выполнения операций.

Двухуровневая архитектура — это модель архитектуры клиент-сервер, в которой клиенты напрямую взаимодействуют с сервером баз данных. В этой модели клиенты отправляют запросы на сервер баз данных, который выполняет операции с данными и возвращает результаты клиентам.

Трехуровневая архитектура — это модель архитектуры клиент-сервер, в которой клиенты взаимодействуют с сервером приложений, который в свою очередь обращается к серверу баз данных. Таким образом, архитектура разделена на три уровня: уровень представления (клиентский), уровень логики приложения (сервер приложений) и уровень доступа к данным (сервер баз данных).

Файл-серверная архитектура — это модель архитектуры, в которой файлы и ресурсы хранятся на централизованном файл-сервере, к которому обращаются клиенты для доступа к данным. В этой модели клиенты получают доступ к файлам через сеть, используя протоколы файлового доступа, такие как SMB или NFS.

Достоинства и недостатки каждой архитектуры:

* Архитектура клиент-сервер:

Достоинства:

Распределение нагрузки между клиентами и сервером.

Централизованное управление данными и ресурсами.

Недостатки:

Возможны проблемы с масштабируемостью при большом числе клиентов.

Требуется постоянное подключение клиента к сети для доступа к ресурсам.

* Двухуровневая архитектура:

Достоинства:

Простота реализации и управления.

Прямое взаимодействие с базой данных без посредников.

Недостатки:

Ограниченная масштабируемость при увеличении числа клиентов.

Сложности в поддержке одновременной работы с несколькими клиентами.

* Трехуровневая архитектура:

Достоинства:

Распределение логики приложения и уровня доступа к данным, что улучшает масштабируемость и обеспечивает легкость разработки и поддержки.

Уменьшение зависимости клиента от структуры базы данных.

Недостатки:

Дополнительные слои и компоненты могут увеличить сложность архитектуры.

Необходимость в дополнительных серверах и инфраструктуре может увеличить стоимость развертывания.

* Файл-серверная архитектура:

Достоинства:

Централизованное хранение и управление файлами и ресурсами.

Простота в реализации и использовании.

Недостатки:

Ограниченная производительность при большом количестве одновременных запросов.

Риск единой точки отказа (SPOF) при сбое сервера или сети.

1. Расшифруйте аббревиатуру ADO.Net, перечислите основные провайдеры данных технологии ADO.Net. Приведите общую характеристику подсоединенных объектов технологии ADO.Net. Поясните на примерах.
2. Расшифруйте аббревиатуру ADO.Net. Назовите отсоединенные (автономные) объекты. Приведите характеристику свойств описанных объектов. Поясните на примерах.

ADO.NET расшифровывается как ActiveX Data Objects .NET.

Отсоединенные (автономные) объекты в ADO.NET относятся к объектам, которые могут содержать данные из базы данных или другого источника данных, но не имеют непосредственного подключения к источнику данных. Эти объекты могут быть изменены, добавлены или удалены независимо от источника данных, а затем синхронизированы с источником данных при необходимости.

Основными отсоединенными (автономными) объектами в ADO.NET являются DataSet и DataTable.

DataSet — это контейнер для одного или нескольких объектов DataTable, а также информации о отношениях между ними. Он может быть использован для хранения данных из базы данных в виде таблиц и работает в автономном режиме, то есть не требует постоянного подключения к источнику данных.

DataTable — это объект, представляющий таблицу данных в DataSet. Он содержит строки и столбцы данных, а также связанную с ними метаинформацию. DataTable также может работать в автономном режиме.

Характеристики свойств объектов DataSet и DataTable:

* Rows: представляет коллекцию строк данных в таблице.
* Columns: представляет коллекцию столбцов данных в таблице.
* Constraints: представляет ограничения данных, такие как уникальность и внешние ключи.
* Relations: представляет отношения между таблицами в DataSet.
* DataSet: для DataTable - ссылка на родительский DataSet.

Пример использования DataSet и DataTable:

using System;

using System.Data;

class Program

{

static void Main(string[] args)

{

// Создание DataSet

DataSet dataSet = new DataSet();

// Создание DataTable

DataTable dataTable = new DataTable("Employees");

// Добавление столбцов в DataTable

dataTable.Columns.Add("ID", typeof(int));

dataTable.Columns.Add("Name", typeof(string));

dataTable.Columns.Add("Department", typeof(string));

// Добавление DataTable в DataSet

dataSet.Tables.Add(dataTable);

// Добавление данных в DataTable

dataTable.Rows.Add(1, "John", "IT");

dataTable.Rows.Add(2, "Alice", "HR");

dataTable.Rows.Add(3, "Bob", "Finance");

// Изменение данных в DataTable

dataTable.Rows[0]["Department"] = "Engineering";

// Удаление строки из DataTable

dataTable.Rows[1].Delete();

// Синхронизация данных с источником данных

// (например, с базой данных) при необходимости

}

}

1. (35-38). Перечислите подключенные классы технологии ADO.Net. Охарактеризуйте класс Connection: защита сведений о соединении с помощью построителей строк подключения. Приведите примеры, демонстрирующие работу описанного класса.

В ADO.NET есть несколько основных классов, однако наиболее важными из них являются:

* Connection: представляет соединение с источником данных. Этот класс используется для открытия, закрытия и управления соединением с базой данных.
* Command: представляет SQL-запрос или хранимую процедуру, которая должна быть выполнена в базе данных.
* Parameters: коллекция параметров команды.
* DataAdapter: позволяет заполнить DataSet данными из источника данных и обновить источник данных с изменениями из DataSet.
* DataReader: позволяет только для чтения последовательно извлечь данные из источника данных.
* DataSet: представляет локальное кэшированное представление данных.

Описание класса Connection:

Класс Connection в ADO.NET используется для установки и управления соединением с базой данных. Он предоставляет методы для открытия, закрытия и управления соединением.

Защита сведений о соединении с помощью построителей строк подключения позволяет хранить конфиденциальную информацию о подключении (такую как имя сервера, имя пользователя, пароль и т. д.) в зашифрованном виде в конфигурационных файлах приложения.

Примеры использования класса Connection:

using System.Data.SqlClient;

class Program

{

static void Main(string[] args)

{

// Создание строки подключения

SqlConnectionStringBuilder builder = new SqlConnectionStringBuilder();

builder.DataSource = "localhost";

builder.InitialCatalog = "MyDatabase";

builder.UserID = "myUsername";

builder.Password = "myPassword";

// Использование строки подключения для создания объекта Connection

using (SqlConnection connection = new SqlConnection(builder.ConnectionString))

{

try

{

// Открытие соединения

connection.Open();

Console.WriteLine("Подключение открыто");

// Делать что-то с подключением...

// Закрытие соединения

connection.Close();

Console.WriteLine("Подключение закрыто");

}

catch (SqlException ex)

{

Console.WriteLine(ex.Message);

}

}

}

}

Пример использования класса Command и DataReader:

using System;

using System.Data.SqlClient;

class Program

{

static void Main(string[] args)

{

string connectionString = "Data Source=localhost;Initial Catalog=MyDatabase;Integrated Security=True";

// Создание объекта Connection

using (SqlConnection connection = new SqlConnection(connectionString))

{

// Создание объекта Command

SqlCommand command = new SqlCommand();

command.Connection = connection;

command.CommandText = "SELECT \* FROM Employees WHERE Department = @department";

// Добавление параметра к команде

command.Parameters.AddWithValue("@department", "IT");

try

{

// Открытие соединения

connection.Open();

// Выполнение команды и получение результата

SqlDataReader reader = command.ExecuteReader();

// Обработка результатов

while (reader.Read())

{

Console.WriteLine("Employee Name: {0}", reader["Name"]);

}

}

catch (SqlException ex)

{

Console.WriteLine("Error: " + ex.Message);

}

}

}

}

Пример использования класса Parameter:

using System;

using System.Data.SqlClient;

class Program

{

static void Main(string[] args)

{

string connectionString = "Data Source=localhost;Initial Catalog=MyDatabase;Integrated Security=True";

using (SqlConnection connection = new SqlConnection(connectionString))

{

SqlCommand command = new SqlCommand("SELECT \* FROM Employees WHERE Department = @department", connection);

// Создание и добавление параметра к команде

SqlParameter parameter = new SqlParameter("@department", "IT");

command.Parameters.Add(parameter);

try

{

connection.Open();

SqlDataReader reader = command.ExecuteReader();

while (reader.Read())

{

Console.WriteLine("Employee Name: {0}, Department: {1}", reader["Name"], reader["Department"]);

}

reader.Close();

}

catch (SqlException ex)

{

Console.WriteLine("Error: " + ex.Message);

}

}

}

}

1. Назовите объект, посредством которого происходит связывание данных при создании пользовательского интерфейса. Поясните суть простой и сложной привязки данных.

Объект, который обеспечивает связывание данных при создании пользовательского интерфейса в технологии .NET, называется DataBinding или просто привязка данных.

Простая привязка данных предполагает привязку одного свойства элемента управления (например, текстового поля или метки) к одному свойству объекта данных (например, свойству класса или значению в таблице базы данных). Например, привязка текстового поля к свойству Name объекта типа Person:

textBox1.DataBindings.Add("Text", person, "Name");

В этом примере текстовое поле textBox1 связано с объектом person через свойство Name. При изменении значения свойства Name в объекте person, значение в текстовом поле автоматически обновляется, и наоборот.

Сложная привязка данных подразумевает более сложные сценарии, такие как привязка элемента управления к коллекции объектов, использование форматирования данных, фильтрация, сортировка и т. д. Пример сложной привязки данных может включать привязку элемента управления DataGridView к коллекции объектов:

dataGridView1.DataSource = list;

Здесь list — это коллекция объектов, которая является источником данных для элемента управления DataGridView. DataGridView будет автоматически отображать данные из коллекции и обновляться при изменениях в коллекции.

Привязка данных позволяет с легкостью обеспечивать синхронизацию между данными в приложении и их отображением в пользовательском интерфейсе, уменьшая объем кода и повышая поддерживаемость приложения.

1. Назовите объект, посредством которого происходит навигация по записям при создании пользовательского интерфейса. Охарактеризуйте основные его свойства и методы. Приведите примеры.

Объект, посредством которого происходит навигация по записям при создании пользовательского интерфейса в технологии .NET, называется BindingNavigator.

BindingNavigator представляет собой элемент управления, который обеспечивает удобный интерфейс для навигации по записям и выполнения операций CRUD (Create, Read, Update, Delete) в связанных источниках данных, таких как базы данных, списки или коллекции объектов.

Основные свойства и методы BindingNavigator:

* AddNewItem: Метод для добавления новой записи в источник данных.
* DeleteItem: Метод для удаления текущей записи из источника данных.
* MoveFirstItem, MovePreviousItem, MoveNextItem, MoveLastItem: Методы для навигации к первой, предыдущей, следующей и последней записи соответственно.
* PositionItem: Свойство, отображающее текущую позицию пользователя в источнике данных.
* CountItem: Свойство, отображающее общее количество записей в источнике данных.

Пример использования BindingNavigator:

// Создание BindingSource и привязка его к источнику данных (например, таблице базы данных)

BindingSource bindingSource = new BindingSource();

bindingSource.DataSource = myDataTable;

// Создание BindingNavigator и привязка его к BindingSource

BindingNavigator bindingNavigator = new BindingNavigator();

bindingNavigator.BindingSource = bindingSource;

// Добавление элементов управления на форму

this.Controls.Add(bindingNavigator);

// Пример использования методов и свойств BindingNavigator для навигации по записям

bindingNavigator.MoveFirstItem.Click += (sender, e) => bindingSource.MoveFirst();

bindingNavigator.MovePreviousItem.Click += (sender, e) => bindingSource.MovePrevious();

bindingNavigator.MoveNextItem.Click += (sender, e) => bindingSource.MoveNext();

bindingNavigator.MoveLastItem.Click += (sender, e) => bindingSource.MoveLast();

// Пример использования методов BindingNavigator для выполнения операций CRUD

bindingNavigator.AddNewItem.Click += (sender, e) =>

{

// Добавление новой записи

DataRow newRow = myDataTable.NewRow();

myDataTable.Rows.Add(newRow);

};

bindingNavigator.DeleteItem.Click += (sender, e) =>

{

// Удаление текущей записи

if (bindingSource.Current != null)

bindingSource.RemoveCurrent();

};