**АРХАНГЕЛЬСКИЙ КОЛЛЕДЖ ТЕЛЕКОММУНИКАЦИЙ**

**ИМ. Б.Л. РОЗИНГА (ФИЛИАЛ) СПбГУТ**

**(АКТ (ф) СПбГУТ)**

**Отчеты по лабораторным и практическим работам**

**МДК 11.01**

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Студент | ИСПП-35 | |  |  | *Волков Н.А.* |
|  | (Группа) | | (Подпись) | (Дата) | (И.О. Фамилия) |
| Преподаватель | | |  |  | *Маломан Ю. С.* |
|  | |  | (Подпись) | (Дата) | (И.О. Фамилия) |

Архангельск 2024

# **Лабораторная работа №1**

**Сбор и анализ требований методом use-case**

1. **Цель работы** 
   1. Изучить процесс описания требований к системе методом use-case.
   2. Изучить процесс создания диаграммы вариантов использования.
2. **Контрольные вопросы**
   1. Для чего используется диаграмма вариантов использования?

Ответ: отражает отношения между “акторами” и “прецедентами” и является составной частью модели прецедентов, позволяющей описать систему на концептуальном уровне.

* 1. Что такое «актор» и как он обозначается на диаграмме вариантов использования?

Ответ: Актор – внешняя сущность, взаимодействующая с системой (пользователь, устройство, другая система).

На диаграмме обозначается палочкой.

* 1. Что такое «прецедент» и как он обозначается на диаграмме вариантов использования?

Ответ: Прецедент – конкретный вариант использования системы, описывающий её функциональность. На диаграмме обозначается овалом.

* 1. Что обозначает «отношение ассоциации»?

Ответ: Указывает на связь между актором и прецедентом.

* 1. Что обозначает «отношение обобщения»?

Ответ: Означает наследование свойства и поведение одного элемента другим.

* 1. Что обозначает «отношение включения»?

Ответ: Означает, что один прецедент всегда включает в себя выполнение другого прецеданта.

* 1. Что обозначает «отношение расширения»?

Ответ: Означает, что дополнительный прецедент может быть выполнен в рамках основного, если выполняются определенные условия.

1. Вывод
   1. В ходе лабораторной работы был изучен процесс описания требований к системе методом use-case, а также процесс создания диаграммы вариантов использования.

# **Лабораторная работа №2**

**Проектирование реляционной схемы базы данных в среде СУБД**

1. **Цель работы** 
   1. Научиться применять MySQL Workbench в процессе создания схем моделей БД;
   2. Научиться представлять логическую модель данных согласно нотациям ERD и IDEF1X.
2. **Контрольные вопросы**

Вопрос: Что такое "сущность"?

Сущностью называется объект реального мира, который имеет независимое существование и может быть различен от других объектов. Это могут быть как физические объекты (например, человек, автомобиль), так и абстрактные концепции (например, должность, курс).

Вопрос: Что такое "атрибут"?

Атрибутом называется характеристика или свойство, которое описывает сущность. Атрибуты представляют собой данные о сущности. Например, для сущности "работник" атрибутами могут быть имя, возраст, зарплата и т.д.

Вопрос: Что такое "ключевое поле"?

Ключевым полем (или ключом) называется атрибут или набор атрибутов, которые уникально идентифицируют каждую сущность в наборе сущностей. Ключевое поле используется для однозначного опознавания записей в таблице.

Вопрос: Назначение первичных и внешних ключей:

Первичный ключ - это основной способ уникального идентификации записи в таблице сущности. Внешний ключ используется для установления связи между таблицами и указывает на связанную сущность.

Вопрос: Что такое "связь"?

Связью называется отношение между сущностями, которое отражает взаимосвязи в реальном мире. В модели ER связи представлены линиями между сущностями в диаграмме.

Вопрос: Виды связей между сущностями:

Основные виды связей - это одна ко многим (1:N), многие ко многим (M:N) и одна ко одной (1:1). Также существует рекурсивная связь (одна к самой себе).

Вопрос: Элементы, входящие в ER-диаграммы:

- Символы для обозначения сущностей (прямоугольники)

- Символы для атрибутов (овалы)

- Символы для связи (линии)

- Символы для агрегации (круглые скобки)

Вопрос: Для чего применяются ER-диаграммы?

ER-диаграммы используются для графического представления структуры базы данных, моделирования данных, проектирования схемы базы данных и визуализации отношений между сущностями.

1. **Вывод**
   1. В ходе лабораторной работы мы научились применять MySQL Workbench в процессе создания схем моделей БД;
   2. Научились представлять логическую модель данных согласно нотациям ERD и IDEF1X.

# **Лабораторная работа №3**

**Приведение БД к нормальной форме**

1. **Цель работы** 
   1. Изучить процесс приведения отношений от ненормализованного вида к четвертой нормальной форме;
   2. Изучить процесс декомпозии отношений.
2. **Контрольные вопросы**

Вопрос: что называется первичным ключом отношения?

Первичный ключ - это уникальный идентификатор записи в таблице, который позволяет однозначно определить каждую запись в таблице [1].

Вопрос: Что называется внешним ключом отношения?

Внешний ключ - это поле в одной таблице, которое содержит значение из поля первичного ключа другой таблицы, устанавливая связь между таблицами [1].

Вопрос: В чем заключается процесс нормализации отношений?

Процесс нормализации отношений включает следующие этапы:

1. Первая форма нормальности (1НФ): каждый атрибут должен иметь только один значение и быть уникальным для каждой записи [1].

2. Вторая форма нормальности (2НФ): кроме этого, все неключевые атрибуты должны зависеть от составного ключа [1].

3. Третья форма нормальности (3НФ): нет транзитивных зависимостей между атрибутами [1].

Вопрос: В каком случае атрибут А функционально зависит от атрибута В?

Атрибут А функционально зависит от атрибута В, если значение А можно однозначно определить по значению Б [1].

Вопрос: В каком случае атрибут А транзитивно зависит от атрибута В?

Атрибут А транзитивно зависит от атрибута В, если значение А можно однозначно определить по значению Б через другие атрибуты [1].

Вопрос: Каким требованиям должно отвечать отношение, находящееся в 1НФ?

Отношение в 1НФ должно удовлетворять следующим условиям:

- Каждая ячейка должна содержать только один значение (атомарность)

- Должно быть уникальный идентификатор записи (первичный ключ)

- Нет дубликатов строк или столбцов [1]

Вопрос: Каким требованиям должно отвечать отношение, находящееся во 2НФ?

Отношение во 2НФ должно удовлетворять условиям 1НФ и дополнительно:

- Все неключевые атрибуты должны полностью зависеть от составного ключа [1].

Вопрос: Каким требованиям должно отвечать отношение, находящееся в 3НФ?

Отношение в 3НФ должно удовлетворять условиям 2НФ и дополнительно:

- Нет транзитивных зависимостей между неключевыми атрибутами [1].

1. **Вывод**

Мы научились изучать процесс приведения отношений от ненорамлизованного вида к четвертой нормальной форме

# **Лабораторная работа №4**

**Установка SQL сервера**

1. **Цель работы** 
   1. Научиться устанавливать разные СУБД, используя docker.
2. **Контрольные вопросы**

Вопрос: Какие стандартные порты для подключения к СУБД MSSQL, MySQL, Postgres?

Для каждого из этих СУБД есть стандартные порты, которые используются для подключения:

- MSSQL (Microsoft SQL Server):

- Основной порт: 1433

- Альтернативный порт: 5043

- MySQL:

- Основной порт: 3306

- PostgreSQL:

- Основной порт: 5432

Вопрос: Какие стандартные папки для хранения данных СУБД MSSQL, MySQL, Postgres?

Стандартные пути для хранения данных различаются в зависимости от операционной системы и версии СУБД:

- MSSQL:

- На Windows: C:\Program Files\Microsoft SQL Server\MSSQL14.MSSQLSERVER\MSSQL\DATA\

- На Linux: /var/mssql/data/

- MySQL:

- На Windows: C:\ProgramData\MySQL\MySQL Server X.X\data\

- На Linux: /var/lib/mysql/

- PostgreSQL:

- На Windows: C:\Program Files\PostgreSQL X.X\data\

- На Linux: /var/lib/postgresql/X.X/main/

Где X.X - это номер версии СУБД.

Вопрос: Для чего используется Docker?

Docker используется для следующих целей:

1. Изоляция приложений: Docker создает контейнеры, которые изолируют приложения друг от друга и от окружения хост-системы [2].

2. Обеспечение переносимости: Контейнеры с Docker могут легко перенестись между разными средами без изменения кода [2].

3. Управление версиями: Docker позволяет легко управлять версиями зависимостей и библиотек [2].

4. Масштабирование: Docker упрощает горизонтальное масштабирование приложений [2].

5. Разработка: Docker предоставляет инструменты для разработчиков, таких как Docker Compose, которые облегчают процесс разработки и тестирования [2].

процесс разработки и развертывания приложений.

1. **Вывод**

Мы научились устанавливать разные СУБД используя docker

# **Лабораторная работа №5**

**Создание базы данных в среде разработки**

1. **Цель работы** 
   1. Изучить способы обеспечения целостности данных в MS SQL Server;
   2. Научиться работать в среде SQL Server Management Studio (SSMS).
2. **Контрольные вопросы**

Вопрос: Что такое SQL Server Management Studio?

SQL Server Management Studio (SSMS) - это графический интерфейс для управления Microsoft SQL Server [1]. Он предоставляет широкий спектр возможностей для работы с базами данных, включая:

- Создание и управление базами данных

- Выполнение запросов и скриптов

- Мониторинг производительности

- Настройка безопасности

- Отладку

SSMS является основным инструментом для взаимодействия с SQL Server и предоставляет удобную визуальную оболочку для выполнения большинства задач администрирования и разработки.

Вопрос: Какие виды авторизации поддерживаются в MS SQL Server?

MS SQL Server поддерживает несколько видов авторизации:

1. Windows Authentication: Использование учетных записей Windows для аутентификации [1].

2. SQL Server Authentication: Пользователи могут аутентифицироваться с помощью имени пользователя и пароля [1].

Вопрос: 8.3 Что такое первичный ключ?

Первичный ключ - это уникальный идентификатор записи в таблице, который позволяет однозначно определять каждую запись в таблице [1]. Он обеспечивает уникальность и целостность данных.

Основные характеристики первичного ключа:

- Не может быть NULL

- Уникален в пределах таблицы

- Обычно состоит из одного или нескольких столбцов

Первичный ключ играет важную роль в организации и структурировании данных в базе данных.

Вопрос: 8.4 Как указать заполнение столбца автоинкрементными значениями?

Для создания столбца с автоматически увеличивающимися значениями используйте следующую синтаксис:

CREATE TABLE TableName (

ColumnName INT IDENTITY(1,1) PRIMARY KEY

);

Здесь `IDENTITY(1,1)` означает, что значения будут начинаться с 1 и увеличиваться на 1 для каждой новой записи [1].

Вопрос: 8.5 Как указать значение по умолчанию?

Чтобы установить значение по умолчанию для столбца, используйте следующий синтаксис:

ALTER TABLE TableName ADD CONSTRAINT DF\_ColumnName DEFAULT 'Default Value' FOR ColumnName;

Это создаст ограничение, которое будет применяться ко всем новым записям, если значение не будет явно указано [1].

Вопрос: 8.6 Как задать проверочное ограничение?

Проверочное ограничение позволяет проверять значения перед их сохранением в базе данных. Вот пример его использования:

ALTER TABLE TableName ADD CHECK (ColumnName > 10 AND ColumnName < 100);

Это ограничение будет блокировать любые попытки сохранения записей, где значение ColumnName не находится в диапазоне от 11 до 99 [1].

Вопрос: Как обеспечить уникальность значений в столбце или наборе столбцов?

Для обеспечения уникальности значений в столбце или наборе столбцов используйте UNIQUE constraint:

ALTER TABLE TableName ADD CONSTRAINT UC\_UniqueConstraintName UNIQUE (Column1, Column2);

Это ограничение гарантирует, что комбинация значений в указанных столбцах будет уникальной для каждой записи [1].

Вопрос: Что такое внешний ключ?

Внешний ключ - это ссылка на первичный ключ в другой таблице, которая устанавливает связь между двумя таблицами [1]. Это позволяет поддерживать целостность данных и обеспечивает возможность создания сложных отношений между таблицами.

Вопрос: 8.9 Какие значения может принимать внешний ключ?

Внешний ключ может принимать следующие значения:

1. CASCADE: При удалении родительской записи, дочерние записи также удаляются.

2. SET NULL: При удалении родительской записи, дочерние записи заполняются NULL.

3. SET DEFAULT: При удалении родительской записи, дочерние записи устанавливаются в значение по умолчанию.

4. NO ACTION: Запрещает удаление родительской записи, если существуют связанные дочерние записи.

5. RESTRICT: Предотвращает удаление родительской записи, если она используется в качестве внешнего ключа [1].

1. Вывод

Мы изучили способы обеспечения целостности данных в MS SQL Server и научились работать в среде SQL Server Management Studio (SSMS).

# **Лабораторная работа №6**

**Создание представлений в СУБД**

1. **Цель работы** 
   1. Научиться создавать и использовать представления в MS SQL Server;
   2. Закрепить навык создания запросов на выборку в MS SQL Server.
2. **Контрольные вопросы**

Вопрос: Что такое представления и чем они отличаются от таблиц?

Представления - это виртуальные таблицы, созданные на основе одного или нескольких существующих таблиц, представлений или других представлений [1]. Основные отличия представлений от обычных таблиц:

1. Физическое хранение: Представления не хранят данные сами по себе, а только содержат запросы для получения данных из исходных источников [1].

2. Вычисление: При обращении к представлению выполняется соответствующий SQL-запрос, который может включать сложные вычисления и соединения таблиц [1].

3. Динамичность: Содержимое представления может меняться при изменении исходных данных [1].

4. Производительность: Использование представлений может влиять на производительность, особенно если они часто обновляются [1].

5. Индексация: Представления обычно не индексируются автоматически, что может повлиять на их производительность [1].

Вопрос: Для чего применяются представления?

Представления используются для следующих целей:

1. Упрощения доступа к данным: Создание удобного интерфейса для сложных запросов [1].

2. Повторяющихся запросов: Объединение частых запросов в одно представление для ускорения выполнения [1].

3. Изоляции бизнес-логики: Хранение сложной логики в представлениях для упрощения управления кодом [1].

4. Обеспечения безопасности: Ограничение доступа к данным через представления вместо прямого обращения к таблицам [1].

5. Анализа данных: Создание отображений сложных запросов для аналитических целей [1].

Вопрос: Возможно ли создание представления, включающего информацию из нескольких таблиц одновременно?

Да, это возможно и часто используется в практике. Представления могут включать информацию из нескольких таблиц одновременно [1]. Это позволяет создавать сложные виртуальные таблицы на основе соединений и агрегации данных из разных источников.

Пример SQL-запроса для создания представления с несколькими таблицами:

CREATE VIEW MyView AS

SELECT t1.ColumnA, t2.ColumnB, t3.ColumnC

FROM Table1 t1

JOIN Table2 t2 ON t1.ID = t2.MainID

JOIN Table3 t3 ON t2.SecondaryID = t3.PrimaryID;

Вопрос: Какие требования предъявляются к обновляемым представлениям?

Для обновляемых представлений (также известных как материализованных представлений) предъявляются следующие требования:

1. Стабильность данных: Содержимое представления должно оставаться стабильным между запросами [1].

2. Оптимизация производительности: Материализованное представление должно быть оптимизировано для быстрого доступа [1].

3. Обновление: Представление должно автоматически обновляться при изменении исходных данных [1].

4. Согласованность: Материализованное представление должно всегда отражать последнее состояние исходных данных [1].

5. Хранение: Представление хранит данные отдельно от исходных таблиц [1].

Вопрос: Как создать обновляемое представление?

Создание обновляемого представления (материализованного представления) зависит от конкретной СУБД. Вот пример SQL-запроса для PostgreSQL:

CREATE MATERIALIZED VIEW my\_materialized\_view AS

SELECT column1, column2, ...

FROM table1

JOIN table2 ON condition;

-- Для заполнения данных:

REFRESH MATERIALIZED VIEW my\_materialized\_view;

-- Для автоматического обновления при изменении исходных данных:

CREATE OR REPLACE MATERIALIZED VIEW my\_materialized\_view AS ...;

1. Вывод
   1. В ходе лабораторной работы Научились создавать и использовать представления в MS SQL Server; Закреплять навык создания запросов на выборку в MS SQL Server.

# **Лабораторная работа №7**

**Создание функций пользователя в СУБД**

1. **Цель работы** 
   1. Научиться создавать и использовать скалярные и табличные функции пользователя в MS SQL Server;
2. **Контрольные вопросы**

Вопрос: Что такое функции пользователя и для чего они применяются?

Функции пользователя - это пользовательские определения SQL-функций, создаваемые в рамках конкретной базы данных [1]. Они используются для:

1. Упрощения сложных вычислений

2. Повторяющихся запросов

3. Изоляции бизнес-логики

4. Обеспечения согласованности данных

5. Улучшения производительности запросов

Функции пользователя могут быть определены как скалярные (возвращают одно значение) или табличные (возвращают набор значений).

Вопрос: Чем отличается скалярная функция от табличной?

Основные отличия скалярных и табличных функций:

1. Возвращаемый тип результата:

- Скалярная функция возвращает одно значение

- Табличная функция возвращает набор значений (обычно таблицу)

2. Использование:

- Скалярные функции используются в выражениях и сравнениях

- Табличные функции используются в подзапросах и JOIN'ах

3. Применимость:

- Скалярные функции могут быть использованы в WHERE, SELECT, FROM, GROUP BY, HAVING

- Табличные функции обычно используются в FROM или WHERE

4. Выполнение:

- Скалярные функции выполняются для каждого элемента набора

- Табличные функции выполняются один раз и возвращают результат

Вопрос: Как объявить переменную и присвоить ей значение?

Объявление переменной и присвоение значения зависит от языка программирования, используемого для работы с базой данных. Вот примеры для некоторых популярных языков:

1. T-SQL (MSSQL):

DECLARE @variable\_name DATATYPE

SET @variable\_name = value

2. PL/SQL (Oracle):

DECLARE

variable\_name datatype;

BEGIN

variable\_name := value;

END;

3. PostgreSQL:

DECLARE variable\_name datatype;

SET variable\_name = value;

Вопрос: Как указать в функции параметр по умолчанию?

Указание параметра по умолчанию зависит от языка SQL и типа функции. Вот несколько примеров:

1. T-SQL (MSSQL):

CREATE FUNCTION MyFunction (@param1 INT = 10) RETURNS INT AS

2. PostgreSQL:

CREATE OR REPLACE FUNCTION my\_function(param1 INT DEFAULT 10)

RETURNS integer AS $$

BEGIN

RETURN param1 \* 2;

END;

$$ LANGUAGE plpgsql;

3. PL/SQL (Oracle):

FUNCTION my\_function(p\_param1 IN NUMBER := 10) RETURN NUMBER IS

BEGIN

RETURN p\_param1 \* 2;

END my\_function;

Вопрос: Как вызвать скалярную функцию?

Вызов скалярной функции зависит от языка SQL и типа функции. Вот несколько примеров:

1. T-SQL (MSSQL):

SELECT dbo.MyScalarFunction(10)

2. PostgreSQL:

SELECT my\_scalar\_function(10);

3. PL/SQL (Oracle):

DECLARE

result NUMBER;

BEGIN

result := my\_scalar\_function(10);

DBMS\_OUTPUT.PUT\_LINE(result);

END;

Вопрос: Как вызвать табличную функцию?

Вызов табличной функции также зависит от языка SQL и типа функции. Вот несколько примеров:

1. T-SQL (MSSQL):

SELECT \* FROM dbo.MyTableFunction()

2. PostgreSQL:

SELECT \* FROM my\_table\_function();

3. PL/SQL (Oracle):

DECLARE

result SYS\_REFCURSOR;

BEGIN

OPEN result FOR my\_table\_function();

LOOP

FETCH result INTO v\_var1, v\_var2;

EXIT WHEN result%NOTFOUND;

-- Обработка результатов

END LOOP;

CLOSE result;

END;

1. **Вывод**
   1. В ходе лабораторной работы мы научились создавать и использовать скалярные и табличные функции пользователя в MS SQL Server;

# **Лабораторная работа №8**

**Создание хранимых процедур в СУБД**

1. **Цель работы** 
   1. Научиться создавать и использовать хранимые процедуры в MS SQL Server;
   2. Закрепить навык создания запросов на выборку и модификацию данных в MS SQL Server.
2. **Контрольные вопросы**

Вопрос: Что такое хранимые процедуры и для чего они применяются?

Хранимые процедуры - это наборы инструкций SQL, сохраняемые в базе данных и выполняемые как единое целое [1]. Они используются для следующих целей:

1. Упрощения сложных операций

2. Изоляции бизнес-логики от остальной части кода

3. Повышения производительности за счет оптимизации запросов

4. Обеспечения согласованности данных

5. Реализации сложных бизнес-процессов

6. Управления транзакциями

7. Выполнения повторяющихся задач

Хранимые процедуры особенно полезны при работе с транзакциями, сложными вычислениях и операциях, требующих изменения нескольких таблиц одновременно.

Вопрос: Чем отличается функция пользователя от хранимой процедуры?

Основные различия между пользовательскими функциями и хранимыми процедурами:

1. Возвращаемое значение:

- Функции возвращают одно значение (обычно через RETURN)

- Процедуры не возвращают явное значение

2. Структура:

- Функции имеют точку входа и выхода

- Процедуры могут иметь несколько точек входа и выходов

3. Использование:

- Функции часто используются в выражениях и сравнениях

- Процедуры обычно вызываются отдельно для выполнения задачи

4. Применение:

- Функции хорошо подходят для вычислений и преобразований данных

- Процедуры лучше всего подходит для выполнения операций, изменяющих данные или выполняющих бизнес-процессы

5. Повторяемость:

- Функции часто повторяются в коде

- Процедуры обычно выполняются один раз за время работы программы

Выбор между функцией и процедурой зависит от конкретной задачи и архитектуры приложения.

Вопрос: Как задать входные параметры хранимой процедуры?

Входные параметры в хранимых процедурах определяются как переменные, которые передаются при вызове процедуры [1]. Вот пример определения процедуры с входными параметрами:

CREATE PROCEDURE MyProcedure (@Param1 INT, @Param2 VARCHAR(50))

AS

BEGIN

-- Тело процедуры

END

При вызове процедуры эти параметры передаются:

EXEC MyProcedure @Param1 = 10, @Param2 = 'SomeValue'

Вопрос: Как задать выходные параметры хранимой процедуры?

Выходные параметры в хранимых процедурах также определяются как переменные, но они должны быть объявлены как OUTPUT [1]:

CREATE PROCEDURE MyProcedure (@Param1 INT, @Param2 VARCHAR(50), @OutputParam INT OUTPUT)

AS

BEGIN

-- Тело процедуры

SET @OutputParam = @Param1 \* 2

END

При вызове процедуры выходные параметры должны быть инициализированы:

DECLARE @OutputValue INT

EXEC MyProcedure @Param1 = 10, @Param2 = 'SomeValue', @OutputParam = @OutputValue OUTPUT

SELECT @OutputValue -- Выводит 20

Вопрос: Как вызвать выполнение хранимой процедуры?

Вызов хранимой процедуры зависит от языка SQL и типа базы данных [1]. Вот несколько примеров:

1. T-SQL (MSSQL):

EXEC MyProcedure @Param1 = 10, @Param2 = 'SomeValue'

2. PostgreSQL:

CALL MyProcedure(10, 'SomeValue');

3. PL/SQL (Oracle):

DECLARE

result NUMBER;

BEGIN

MyProcedure(10, 'SomeValue', result);

DBMS\_OUTPUT.PUT\_LINE(result);

END;

4. MySQL:

CALL MyProcedure(10, 'SomeValue');

1. **Вывод**
   1. В ходе лабораторной работы мы Научились создавать и использовать хранимые процедуры в MS SQL Server закреплять навык создания запросов на выборку и модификацию данных в MS SQL Server.

# **Лабораторная работа №9**

**Создание триггеров в СУБД**

1. **Цель работы** 
   1. Научиться создавать и использовать триггеры в MS SQL Server;
   2. Закрепить навык создания запросов на выборку и модификацию данных в MS SQLServer.
2. **Контрольные вопросы**

Вот ответы на ваши вопросы о триггерах в SQL:

Вопрос: Что такое триггер?

Триггер - это специальный тип хранимой процедуры, которая автоматически выполняется в ответ на определенные действия в базе данных [1]. Он действует как "слушатель" для определенных событий и выполняет определенную задачу при возникновении этих событий.

Основные характеристики триггера:

1. Автоматическое выполнение: Триггер активируется автоматически при определенных действиях над данными.

2. Связь с событием: Триггер привязан к конкретному типу операции (INSERT, UPDATE, DELETE).

3. Группировка: Может группироваться по таблицам и типам событий.

Вопрос: Чем триггер отличается от хранимой процедуры?

Основные различия между триггером и хранимой процедурой:

1. Автоматическое выполнение: Триггер выполняется автоматически, хранимая процедура вызывается вручную [1].

2. Связь с событиями: Триггеры привязаны к конкретным действиям над данными (INSERT, UPDATE, DELETE), хранимые процедуры - нет [1].

3. Точка входа: Триггер имеет четкую точку входа (BEFORE, AFTER, INSTEAD OF), хранимые процедуры - нет [1].

4. Целевая область: Триггеры фокусируются на обработке конкретных событий, хранимые процедуры могут выполнять более широкий спектр задач [1].

5. Синтаксис определения: Триггеры имеют специфический синтаксис определения, отличный от хранимых процедур [1].

Вопрос: Как запустить триггер на выполнение?

Триггеры автоматически активируются при соответствующих действиях над данными. Для их активации обычно требуется:

1. Вставка, обновление или удаление записей в связанной таблице.

2. Выполнение DML-запросов (INSERT, UPDATE, DELETE) над целевой таблицей.

Пример создания триггера:

CREATE TRIGGER MyTrigger

ON MyTable

AFTER INSERT

AS

BEGIN

-- Код триггера

END

Вопрос: Назначение триггеров

Основные назначения триггеров:

1. Управление целостностью данных: Обеспечение соблюдения правил и ограничений при изменении данных [1].

2. Автоматическое обновление связанных таблиц: Обновление зависимых таблиц при изменениях в основной таблице [1].

3. Логирование действий: Запись информации о внесенных изменениях [1].

4. Проверка бизнес-правил: Реализация сложных проверок и обработки событий [1].

5. Повторное выполнение операций: Повторная попытка выполнения операции после ошибки [1].

Вопрос: Чем отличаются триггеры INSTEAD OF и AFTER?

Основные различия между триггерами INSTEAD OF и AFTER:

1. Время выполнения:

- INSTEAD OF: Выполняется вместо основного оператора (заменяет его).

- AFTER: Выполняется после основного оператора [1].

2. Изменение исходного запроса:

- INSTEAD OF: Не изменяет исходный запрос, полностью заменяет его.

- AFTER: Не изменяет исходный запрос, выполняет дополнительные действия после него [1].

3. Поддержка операций:

- INSTEAD OF: Может заменять любые операции (INSERT, UPDATE, DELETE).

- AFTER: Обычно используется только для INSERT и UPDATE операций [1].

4. Производительность:

- INSTEAD OF: Может быть менее эффективным, так как заменяет весь запрос.

- AFTER: Обычно более эффективен, так как выполняется после основного запроса [1].

5. Применимость:

- INSTEAD OF: Лучше подходит для сложных логик, требующих полного контроля над операцией.

- AFTER: Лучше подходит для простых проверок и логирования после изменения данных [1].

Выбор между INSTEAD OF и AFTER зависит от конкретных требований к обработке событий и архитектуры приложения.

1. **Вывод**
   1. В ходе лабораторной работы мы научились создавать и использовать триггеры в MS SQL Server
   2. Закреплять навык создания запросов на выборку и модификацию данных в MS SQLServer.

# **Лабораторная работа №10**

**Реализация доступа пользователей к базе данных**

1. **Цель работы** 
   1. Научиться использовать системные хранимые процедуры и DDL-команды для управления именами входа и пользователями БД в СУБД;
   2. Научиться назначать привилегии пользователю БД;
   3. Закрепить навык создания объектов БД.
2. **Контрольные вопросы**

Вопрос: В чем отличие между именами входа и пользователями БД?

Отличия между именами входа и пользователями базы данных в MS SQL Server:

1. Имя входа:

- Это имя, которое используется для аутентификации при подключении к серверу.

- Оно не связано напрямую с конкретной базой данных.

2. Пользователь БД:

- Это отдельный объект внутри конкретной базы данных.

- Может иметь свои права доступа и разрешения на уровне базы данных.

Основные различия:

- Имя входа применяется на уровне сервера, пользователь БД - на уровне базы данных.

- Имя входа может быть использовано для доступа ко всем базам данных на сервере, пользователь БД ограничен одной базой данных.

- Права доступа могут быть назначены отдельно для имени входа и пользователя БД.

Вопрос: 8.2 Как идентифицируются пользователи в MS SQL Server?

В MS SQL Server пользователи могут быть идентифицированы следующими способами:

1. Имя входа (Login):

- Это уникальное имя пользователя, которое используется для аутентификации.

- Пример: 'john\_doe'

2. Пароль:

- Секретный код, который пользователь должен знать для аутентификации.

3. Учетная запись Windows:

- Возможность использования учетных данных из Active Directory.

- Пользователи могут аутентифицироваться с помощью имени пользователя и пароля Windows.

4. Сертификаты:

- Возможность использования цифровых сертификатов для шифрования паролей.

5. Azure AD:

- Поддержка аутентификации через Azure Active Directory.

Вопрос: 8.3 На какие уровни разделяется система безопасности MS SQL Server?

Система безопасности в MS SQL Server разделена на несколько уровней:

1. Сетевой уровень:

- Определяет, может ли клиент подключиться к серверу.

- Контролируется через сетевые протоколы и файрволы.

2. Серверный уровень:

- Управление правами доступа к серверу.

- Контролируется через SQL Server Configuration Manager.

3. Базовый уровень:

- Управление правами доступа в конкретных базах данных.

- Контролируется через объекты базы данных (пользователи, роли).

4. Объектный уровень:

- Права доступа к отдельным объектам в базе данных (таблицы, процедуры и т.д.).

5. Программный уровень:

- Безопасность приложений, использующих SQL Server.

Вопрос: 8.4 Каково назначение ролей сервера?

Роли сервера в MS SQL Server используются для:

1. Управления правами доступа к серверу.

2. Контроля над ресурсами сервера.

3. Определения уровня привилегий пользователей.

4. Управления доступом к системным объектам и функциям.

5. Контроля над выполнением определенных операций на сервере.

Примеры ролей сервера:

- sysadmin: полные права управления сервером

- serveradmin: управление конфигурацией сервера

- setupadmin: управление установкой и настройкой сервера

- securityadmin: управление правами безопасности

- processadmin: управление процессами SQL Server

Вопрос: Каково назначение ролей БД?

Роли базы данных в MS SQL Server используются для:

1. Управления правами доступа к объектам базы данных.

2. Контроля над выполнением определенных операций в рамках базы данных.

3. Определения уровня привилегий пользователей.

4. Группировки пользователей с схожими потребностями в правах доступа.

5. Упрощения управления правами доступа на уровне базы данных.

Примеры ролей базы данных:

- db\_owner: полные права доступа ко всем объектам базы данных

- db\_datareader: чтение данных во всех таблицах

- db\_datawriter: запись данных во все таблицы

- db\_ddladmin: выполнение DDL-запросов

- db\_securityadmin: управление правами безопасности

1. **Вывод**
   1. В ходе лабораторной работы мы научились использовать системные хранимые процедуры и DDL-команды для управления именами входа и пользователями БД в СУБД;
   2. Научились назначать привилегии пользователю БД;
   3. Закрепили навык создания объектов БД.

# **Лабораторная работа №11**

**Выполнение резервного копирования и восстановления БД**

1. **Цель работы** 
   1. Научиться выполнять резервное копирование БД;
   2. Научиться сохранять во внешних файлах описание структуры и данные БД;
   3. Научиться выполнять восстановление БД
2. **Контрольные вопросы**

Вопрос: Для чего создаются резервные копии БД?

Резервные копии создаются для следующих целей:

1. Предотвращение потери данных в случае сбоя или физического повреждения оборудования.

2. Восстановление базы данных до определенного момента времени.

3. Миграция данных между различными средами (разработка, тестирование, продакшн).

4. Создание копий базы данных для других пользователей или организаций.

5. Обеспечение возможности быстрого восстановления после кибербезопасительных инцидентов.

Вопрос: В чем отличие между полным и разностным резервным копированием?

Основные отличия между полным и разностным резервным копированием:

1. Полное резервное копирование:

- Создает полную копию всей базы данных.

- Занимает больше места и времени.

- Позволяет восстановить всю базу данных с нуля.

- Часто выполняется реже (например, ежедневно или еженедельно).

2. Разностное резервное копирование:

- Создает только изменения с предыдущей полной копии.

- Занимает меньше места и времени.

- Позволяет восстановить базу данных до определенного момента.

- Часто выполняется чаще (например, ежечасно или ежедневно).

Вопрос: Как часто должен выполняться каждый из видов резервного копирования БД (привести пример расписания)?

Рекомендуемые частота выполнения:

1. Полное резервное копирование:

- Ежедневно или еженедельно в зависимости от критичности данных.

Пример расписания:

Понедельник: Полная резервная копия

Среда: Разностное резервное копирование

Четверг: Разностное резервное копирование

Пятница: Полная резервная копия

2. Разностное резервное копирование:

- Каждые несколько часов для критически важных систем.

- Ежедневно для менее критичных систем.

Вопрос: Какие скрипты можно сформировать для объектов БД?

Вот некоторые типы скриптов, которые можно создать для объектов базы данных:

1. CREATE скрипт: Создает объект из существующего.

2. ALTER скрипт: Изменяет существующий объект.

3. DROP скрипт: Удаляет объект из базы данных.

4. TRUNCATE скрипт: Очищает содержимое таблицы.

5. SELECT скрипт: Извлекает данные из объекта.

Вопрос: Как выполнить восстановление БД из резервной копии?

Восстановление базы данных обычно выполняется через специальные команды или инструменты управления базами данных.

Вопрос: Какая команда выполняет восстановление БД из резервной копии?

Команда для восстановления базы данных зависит от используемой системы управления базами данных:

1. SQL Server:

RESTORE DATABASE MyDatabase FROM DISK = 'C:\Backup\MyDatabase.bak'

2. Oracle:

RMAN > RUN {

RESTORE DATABASE mydb;

RECOVER DATABASE;

}

3. MySQL:

mysql -u root -p < dump.sql

Вопрос: В каком порядке надо восстанавливать резервные копии?

Обычно порядок восстановления выглядит так:

1. Полная резервная копия (если требуется).

2. Разностные резервные копии в обратном порядке времени создания.

Пример:

Понедельник: Полная копия (2023-12-18)

Вторник: Разностная копия (2023-12-19)

Среда: Разностная копия (2023-12-20)

Четверг: Разностная копия (2023-12-21)

Вопрос: Какой параметр у команды восстановления данных отключает/запускает восстановление БД?

В разных системах управления базами данных этот параметр может называться по-разному:

1. SQL Server:

RESTORE DATABASE MyDatabase WITH NORECOVERY

-- Для продолжения восстановления

RESTORE DATABASE MyDatabase WITH RECOVERY

-- Для завершения восстановления

2. Oracle:

ALTER DATABASE MyDB RECOVER USING BACKUP CONTROLFILE;

3. MySQL:

SET GLOBAL sql\_mode=(SELECT @@sql\_mode REGEXP 'only\_full\_group\_by');

Вопрос: 8.9 Для чего используется мастер импорта и экспорта?

Мастер импорта и экспорта используется для:

1. Экспорта данных из одной базы данных в другую.

2. Импорта данных из файла или другой базы данных.

3. Миграции данных между различными системами управления базами данных.

4. Создания резервных копий с помощью специальных форматов (например, BCP).

5. Обработки больших объемов данных без использования SQL-запросов.

1. **Вывод**
   1. В ходе лабораторной работы мы Научились выполнять резервное копирование БД;
   2. Научились сохранять во внешних файлах описание структуры и данные БД;
   3. Научились выполнять восстановление БД

# **Лабораторная работа №12**

**Экспорт данных базы в документы пользователя**

1. **Цель работы** 
   1. Научиться выполнять экспорт данных из БД.
2. **Контрольные вопросы**

Вопрос: Как выполнить экспорт csv-файла в Management Studio?

Для экспорта данных в CSV-файл в Microsoft Management Studio (MSSMS):

1. Откройте объект Explorer и выберите нужную таблицу или запрос.

2. Правый клик -> "Экспорт данных".

3. В диалоговом окне выберите "CSV" в качестве формата.

4. Выберите место сохранения файла и нажмите "ОК".

Альтернативный метод:

1. Выполните запрос, который вы хотите экспортировать.

2. Правый клик -> "Новая вкладка" -> "Экспорт данных".

3. Выберите "CSV" и следуйте инструкциям.

Вопрос: 6.2 Как выполнить экспорт xlsx-файла в Management Studio?

Для экспорта данных в XLSX-файл в MSSMS:

1. Откройте объект Explorer и выберите нужную таблицу или запрос.

2. Правый клик -> "Экспорт данных".

3. В диалоговом окне выберите "Excel" в качестве формата.

4. Выберите место сохранения файла и нажмите "ОК".

Важно отметить, что MSSMS не поддерживает прямой экспорт в формат XLSX. Он экспортирует в XLS, который затем можно конвертировать в XLSX.

Вопрос: Как в Excel открыть файл формата txt/csv и xml?

Для открытия файлов txt, csv и xml в Excel:

1. Нажмите "Файл" -> "Открыть" -> "Общий каталог".

2. Выберите нужный файл и нажмите "Открыть".

Для txt и csv файлов:

- Excel автоматически распознает формат и загрузит данные.

Для xml файлов:

- Выберите "XML-специфичный" в диалоговом окне.

- Выберите "Разделить на несколько листов".

Вопрос: Данные из сколько таблиц могут храниться в файле csv?

В одном CSV-файле может храниться информация из одной таблицы. Однако, если вам нужно экспортировать данные из нескольких таблиц, вы можете:

1. Создать временную таблицу, объединяющую необходимые данные.

2. Экспортировать эту временную таблицу в CSV.

Вопрос: Как экспортировать данные в файл формата JSON?

Для экспорта в JSON в MSSMS:

1. Выполните запрос, который вы хотите экспортировать.

2. Правый клик -> "Новая вкладка" -> "Экспорт данных".

3. Выберите "JSON" в качестве формата.

4. Выберите место сохранения файла и нажмите "ОК".

Если ваша версия MSSMS не поддерживает JSON, вы можете использовать сторонние инструменты или скрипты для конвертации данных в JSON.

Вопрос: Как экспортировать данные в файл формата XML?

Для экспорта в XML в MSSMS:

1. Выполните запрос, который вы хотите экспортировать.

2. Правый клик -> "Новая вкладка" -> "Экспорт данных".

3. Выберите "XML" в качестве формата.

4. Выберите место сохранения файла и нажмите "ОК".

Если вам нужно экспортировать структуру схемы базы данных в XML, вы можете использовать инструмент "Библиотека объектов" в MSSMS.

Важно помнить, что выбор формата экспорта зависит от конкретных требований проекта и дальнейшего использования данных. Каждый формат имеет свои особенности и преимущества в различных сценариях использования.

1. **Вывод**
   1. В ходе лабораторной работы мы научились выполнять экспорт данных из БД.

# **Лабораторная работа №13**

**Импорт данных пользователя в базу данных**

1. **Цель работы** 
   1. Научиться выполнять импорт данных в БД.
2. **Контрольные вопросы**

Вопрос: Как выполнить импорт xlsx-файла в Management Studio?

Для импорта xlsx-файла в Management Studio выполните следующие шаги:

1. Откройте MMS и перейдите в меню "Данные" -> "Импорт данных".

2. Выберите источник данных "Excel" и нажмите "Далее".

3. Найдите ваш xlsx-файл и выберите его.

4. Выберите схему данных, которую хотите использовать для импорта.

5. Настройте параметры импорта, такие как преобразование типов данных и обработку ошибок.

6. Нажмите "Далее" и затем "Завершить", чтобы начать импорт.

Вопрос: Как выполнить импорт csv-файла в Management Studio?

Для импорта csv-файла в Management Studio выполните следующие шаги:

1. Откройте MMS и перейдите в меню "Данные" -> "Импорт данных".

2. Выберите источник данных "CSV" и нажмите "Далее".

3. Найдите ваш csv-файл и выберите его.

4. Настройте параметры импорта, такие как разделитель полей и преобразование типов данных.

5. Выберите схему данных, которую хотите использовать для импорта.

6. Нажмите "Далее" и затем "Завершить", чтобы начать импорт.

Вопрос: Как выполнить импорт файла XML?

Для импорта файла XML в Management Studio выполните следующие шаги:

1. Откройте MMS и перейдите в меню "Данные" -> "Импорт данных".

2. Выберите источник данных "XML" и нажмите "Далее".

3. Найдите ваш XML-файл и выберите его.

4. Настройте параметры импорта, такие как преобразование типов данных и обработка ошибок.

5. Выберите схему данных, которую хотите использовать для импорта.

6. Нажмите "Далее" и затем "Завершить", чтобы начать импорт.

Вопрос: Как выполнить импорт файла JSON?

Для импорта файла JSON в Management Studio выполните следующие шаги:

1. Откройте MMS и перейдите в меню "Данные" -> "Импорт данных".

2. Выберите источник данных "JSON" и нажмите "Далее".

3. Найдите ваш JSON-файл и выберите его.

4. Настройте параметры импорта, такие как преобразование типов данных и обработка ошибок.

5. Выберите схему данных, которую хотите использовать для импорта.

6. Нажмите "Далее" и затем "Завершить", чтобы начать импорт.

При выполнении импорта будьте внимательны к настройкам преобразования типов данных и обработке ошибок. В некоторых случаях может потребоваться дополнительная настройка для корректного импорта данных из различных форматов файлов.

1. **Вывод**
   1. В ходе лабораторной работы мы научились выполнять импорт данных в БД.

# **Лабораторная работа №14**

**Создание слоя доступа к данным БД**

1. **Цель работы** 
   1. Научиться создавать приложение C# для организации доступа к БД.
   2. Изучить свойства и методы компонентов SqlConnection, SqlCommand, SqlDataReader, научиться их применять и настраивать;
2. **Контрольные вопросы**

Вопрос: 8.1 Какое назначение у элемента SqlConnection?

SqlConnection используется для установления соединения с сервером базы данных. Он представляет собой соединение с конкретной базой данных на сервере [1].

Вопрос: 8.2 Какое назначение у элемента SqlCommand?

SqlCommand используется для отправки SQL-запросов к серверу базы данных. Он позволяет выполнять различные операции с данными, такие как выборка, вставка, обновление и удаление [1].

Вопрос: 8.3 Какое назначение у элемента SqlDataReader?

SqlDataReader используется для чтения результатов SQL-запроса. Он предоставляет доступ к данным построчно, позволяя эффективно обрабатывать большие наборы данных [1].

Вопрос: 8.4 Какие пространства имен требуется подключить для реализации подключения к СУБД MS SQL Server, MySQL соответственно?

Для работы с MS SQL Server:

using System.Data.SqlClient;

Для работы с MySQL:

using MySql.Data.MySqlClient;

Вопрос: 8.5 Какие методы класса SqlCommand позволяют выполнить SQL-запрос?

Основные методы SqlCommand для выполнения SQL-запросов:

1. ExecuteReader() - для выполнения запросов, которые возвращают результаты (например, SELECT).

2. ExecuteNonQuery() - для выполнения запросов без возврата результата (например, INSERT, UPDATE, DELETE).

3. ExecuteScalar() - для выполнения запросов, которые возвращают только один значение.

4. ExecuteXmlReader() - для выполнения запросов, которые возвращают XML-данные [1].

Пример использования этих методов:

using (SqlConnection connection = new SqlConnection(connectionString))

{

connection.Open();

string sql = "SELECT \* FROM Customers WHERE Country='USA'";

// Выполнение запроса с чтением результатов

using (SqlCommand command = new SqlCommand(sql, connection))

{

using (SqlDataReader reader = command.ExecuteReader())

{

while (reader.Read())

{

Console.WriteLine(reader["CustomerID"] + " " + reader["CompanyName"]);

}

}

}

// Выполнение запроса без чтения результатов

int rowsAffected = new SqlCommand("UPDATE Orders SET Status='Shipped' WHERE OrderID=1", connection).ExecuteNonQuery();

}

Эти методы позволяют гибко работать с различными типами SQL-запросов в зависимости от ваших потребностей.

1. **Вывод**
   1. В ходе лабораторной работы мы Научились создавать приложение C# для организации доступа к БД.
   2. Изучили свойства и методы компонентов SqlConnection, SqlCommand, SqlDataReader, научиться их применять и настраивать;

# **Лабораторная работа №15**

**Применение ORM для доступа к данным БД**

1. **Цель работы** 
   1. Научиться создавать приложение C# для организации доступа к БД.
   2. Научиться проектировать приложение, использующее паттерн репозиторий и Dapper.
2. **Контрольные вопросы**

Вопрос: Что такое Dapper и для чего используется?

Dapper - это легковесное ORM (Object-Relational Mapping) для .NET Framework и .NET Core. Он используется для упрощения работы с базами данных, особенно при выполнении CRUD-операций (Create, Read, Update, Delete) [1][2].

Основные преимущества Dapper:

- Простота использования

- Высокая производительность

- Поддержка асинхронных операций

- Интеграция с Entity Framework

Вопрос: Какие методы Dapper позволяют извлечь данные из БД?

Основные методы Dapper для извлечения данных:

1. Query<T>(): выполняет SQL-запрос и возвращает список объектов типа T [2].

var users = db.Query<User>("SELECT \* FROM Users");

2. QueryFirstOrDefault<T>(): выполняет SQL-запрос и возвращает первый объект типа T или null, если не найдено [2].

User user = db.QueryFirstOrDefault<User>("SELECT \* FROM Users WHERE Id = @Id", new { Id = 1 });

3. Single<T>(): выполняет SQL-запрос и возвращает первый объект типа T или выбрасывает исключение, если больше одного объекта [2].

User user = db.Single<User>("SELECT \* FROM Users WHERE Id = @Id", new { Id = 1 });

4. QueryMultiple<T1, T2>(...): выполняет несколько SQL-запросов и возвращает списки разных типов [2].

Вопрос: Какие методы Dapper позволяют изменить данные в БД?

Основные методы Dapper для изменения данных:

1. Insert<T>(): вставляет новый объект в базу данных [2].

db.Insert(new User { Name = "John Doe", Age = 30 });

2. Update<T>(): обновляет существующий объект в базе данных [2].

db.Update(user);

3. Delete<T>(): удаляет объект из базы данных [2].

db.Delete(user);

4. BulkInsert<T>(): выполняет быструю вставку нескольких объектов [2].

Вопрос: Зачем используется паттерн «репозиторий»?

Паттерн "репозиторий" используется для абстрагирования доступа к хранилищам данных (например, базы данных) и предоставления единого интерфейса для операций CRUD [3].

Основные преимущества паттерна "репозиория":

- Разделение ответственности: бизнес-логика отделена от операций с данными.

- Повторное использование кода: единый интерфейс для разных типов хранилищ.

- Упрощение тестирования: легко создавать моки репозитория для unit-тестов.

- Гибкость: легко заменять реализацию репозитория без изменения остальной части приложения.

Пример использования паттерна "репозитория" с Dapper:

public interface IUserRepository

{

Task<User> GetByIdAsync(int id);

Task AddAsync(User user);

}

public class UserRepository : IUserRepository

{

private readonly IDbConnection \_db;

public UserRepository(IDbConnection db)

{

\_db = db;

}

public async Task<User> GetByIdAsync(int id)

{

return await \_db.QueryFirstOrDefaultAsync<User>(

"SELECT \* FROM Users WHERE Id = @Id",

new { Id = id });

}

public async Task AddAsync(User user)

{

await \_db.InsertAsync(user);

}

}

1. **Вывод**
   1. В ходе лабораторной работы мы научились создавать приложение C# для организации доступа к БД.
   2. Научились проектировать приложение, использующее паттерн репозиторий и Dapper.

# **Лабораторная работа №16**

**Применение ORM EF Core**

1. **Цель работы** 
   1. Научиться создавать приложение C# для организации доступа к БД.
   2. Научиться разрабатывать приложение, использующее EF Core.
2. **Контрольные вопросы**

Вопрос: Что такое «ORM»?

ORM (Object-Relational Mapping) - это техника программирования, которая автоматически переводит объекты языка программирования на объекты базы данных и обратно [1].

Основные принципы ORM:

- Абстракция: объекты представлены в виде абстракций, а не физических структур данных.

- Автоматическое преобразование: система автоматически преобразует объекты в строки запросов к базе данных и обратно.

- Объектно-ориентированное программирование: использует концепции ООП для работы с данными.

Вопрос: Что такое «EF Core» и для чего он предназначен?

Entity Framework Core (EF Core) - это легковесное и высокопроизводительное ORM для .NET Framework, .NET Core и Xamarin/macOS [1][2].

Основные цели EF Core:

- Упрощение работы с реляционными базами данных

- Автоматическое управление схемой базы данных

- Поддержка асинхронных операций

- Интеграция с различными базами данных (SQL Server, PostgreSQL, Azure SQL Database, SQLite)

Вопрос: Как получить данные из БД, используя EF Core?

Для получения данных из базы данных с помощью EF Core можно использовать следующие методы:

1. LINQ-запросы:

var users = dbContext.Users.Where(u => u.Age > 18).ToList();

LINQ-представленные запросы преобразуются в SQL-запросы.

2. Raw SQL-запросы:

var users = dbContext.Users.FromSqlRaw("SELECT \* FROM Users WHERE Age > 18").ToList();

3. Async-запросы:

var users = await dbContext.Users.ToListAsync();

4. Include для загрузки связанных данных:

var ordersWithDetails = await dbContext.Orders

.Include(o => o.OrderDetails)

.ToListAsync();

Вопрос: Как выполнить редактирование (вставку, обновление, удаление данных), используя EF Core?

Для редактирования данных в EF Core используются следующие методы:

1. Создание новых объектов:

var newUser = new User { Name = "John Doe", Age = 30 };

dbContext.Users.Add(newUser);

2. Сохранение изменений:

await dbContext.SaveChangesAsync();

3. Обновление существующих объектов:

var existingUser = dbContext.Users.Find(1);

existingUser.Name = "Jane Doe";

dbContext.Entry(existingUser).Property(e => e.Name).IsModified = true;

4. Удаление объектов:

dbContext.Users.Remove(userToDelete);

5. Использование DbSet-методов:

dbContext.Users.AddOrUpdate(u => u.Name, new User { Name = "Default User" });

dbContext.Users.RemoveRange(dbContext.Users.Where(u => u.Age < 18));

6. Использование TransactionScope для группировки операций:

using (var transaction = await dbContext.Database.BeginTransactionAsync())

{

dbContext.Users.Add(newUser);

dbContext.SaveChanges();

await transaction.CommitAsync();

}

1. **Вывод**
   1. В ходе лабораторной работы мы научились создавать приложение C# для организации доступа к БД.
   2. Научились разрабатывать приложение, использующее EF Core.

# **Лабораторная работа №17**

**Разработка приложения для фильтрации, поиска и сортировки данных**

1. **Цель работы** 
   1. Научиться создавать приложение для доступа к БД средствами EF Core.
   2. Научиться выполнять сортировку, фильтрацию и постраничный вывод данных, используя LINQ-запросы.
2. **Контрольные вопросы**

Вопрос: Для чего используются метод OrderBy, OrderByDescending, ThenBy, ThenByDescending и в чем их отличие?

Методы OrderBy, OrderByDescending, ThenBy, ThenByDescending используются для сортировки последовательностей элементов [1].

Основные отличия:

1. OrderBy и OrderByDescending:

- Сортируют по умолчанию по возрастанию (OrderBy) или убыванию (OrderByDescending).

- Если элементы равны, порядок не гарантируется.

2. ThenBy и ThenByDescending:

- Позволяют добавить дополнительные критерии сортировки после основного.

- Работают только если перед ними был вызван OrderBy или OrderByDescending.

Пример использования:

// Сортировка по возрастанию имени, затем по возрастанию возраста

var sortedUsers = users.OrderBy(u => u.Name).ThenBy(u => u.Age);

// Сортировка по убыванию возраста, затем по возрастанию имени

var sortedUsersDesc = users.OrderByDescending(u => u.Age).ThenBy(u => u.Name);

Вопрос: Для чего используются методы Take и Skip и как они применяются при пагинации?

Методы Take и Skip используются для ограничения количества элементов в результате запроса [1].

- Take(n): возвращает первые n элементов последовательности.

- Skip(n): пропускает первые n элементов последовательности.

Применение при пагинации:

int pageSize = 10;

int pageNumber = 2;

var paginatedItems = items.Skip((pageNumber - 1) \* pageSize).Take(pageSize);

Это позволяет реализовать страницную навигацию по большим наборам данных.

Вопрос: Для чего используется метод Where?

Метод Where используется для фильтрации элементов последовательности на основе заданных условий [1].

Пример:

var filteredUsers = users.Where(u => u.Age > 18 && u.Country == "USA");

Вопрос: Какие логические операторы могут применяться при составлении условий?

Основные логические операторы в LINQ:

- AND (&&): объединяет два условия

- OR (||): объединяет два условия

- NOT (!): инверсирует условие

Пример:

var result = users.Where(u => u.Age > 18 || (u.Age >= 65 && u.Country == "USA"));

Вопрос: Как проверить, что значение есть в списке?

Для проверки наличия значения в списке можно использовать метод Contains:

bool containsValue = list.Contains(valueToCheck);

Вопрос: Как проверить, что строка начинается с определенного текста?

Для проверки начала строки можно использовать метод StartsWith:

bool startsWithSpecificText = string.StartsWith(specificText);

Вопрос: Как проверить, что строка содержит определенный текст?

Для проверки наличия подстроки в строке используется метод Contains:

bool containsSpecificText = string.Contains(specificText);

Вопрос: Как указать в Select список требуемых данных?

Метод Select используется для проекции (выбора) нужных данных из исходной последовательности [1]:

var selectedData = source.Select(item => new { item.Id, item.Name, item.Price });

Этот метод позволяет выбрать только необходимые поля или вычислить новые значения на основе существующих.

LINQ предоставляет мощные инструменты для работы со структурированными данными, позволяя легко фильтровать, сортировать, агрегировать и трансформировать данные в цепочках запросов.

1. **Вывод**
   1. В ходе лабораторной работы мы научились создавать приложение для доступа к БД средствами EF Core.
   2. Научились выполнять сортировку, фильтрацию и постраничный вывод данных, используя LINQ-запросы.

# **Лабораторная работа №18**

**Разработка приложения для редактирования данных**

1. **Цель работы** 
   1. Научиться выполнять вставку, обновление и удаление записей средствами EF Core;
   2. Научиться обеспечивать обратную связь при редактировании данных.
2. **Контрольные вопросы**

Пример: Для чего используются методы Add() и AddRange() в EF Core?

Методы Add() и AddRange() используются для добавления новых сущностей в контекст базы данных [1]. Они начинают отслеживание сущностей, но еще не сохраняют их в базе данных.

- Add(): добавляет одну сущность

- AddRange(): добавляет коллекцию сущностей

Пример:

context.Users.Add(new User { Name = "John Doe" });

context.Users.AddRange(new List<User> { new User { Name = "Alice Smith" }, new User { Name = "Bob Johnson" } });

Пример: Для чего используются методы Update() в EF Core?

Метод Update() не существует в EF Core. Вместо этого используется изменение свойств сущности и пометка их как измененных [1].

Пример: Для чего используются методы Remove() и RemoveRange() в EF Core?

Методы Remove() и RemoveRange() используются для удаления сущностей из контекста базы данных [1].

- Remove(): удаляет одну сущность

- RemoveRange(): удаляет коллекцию сущностей

Пример:

context.Users.Remove(userToDelete);

context.Users.RemoveRange(usersToDelete);

Пример: Как сохранить изменения в БД, используя EF Core?

Изменения сохраняются с помощью метода SaveChanges() [1]:

await context.SaveChangesAsync();

Пример: Как изменить значения полей объекта?

Существующие объекты можно изменить путем прямого изменения их свойств [1]:

user.Name = "New Name";

user.Email = "new@email.com";

Пример: Какое значение по умолчанию присваивается идентификатору нового объекта?

По умолчанию, новым объектам присваивается значение null для идентификатора [1]. EF Core будет генерировать ID при сохранении.

Пример: Как передать объект с одной формы на другую?

Для преобразования объекта с одной формы на другую можно использовать методы AutoMapper или ручное копирование свойств [1].

Пример с AutoMapper:

cfg.CreateMap<SourceClass, DestinationClass>()

.ForMember(dest => dest.SomeProperty, opt => opt.MapFrom(src => src.AnotherProperty));

var destinationObject = mapper.Map<DestinationClass>(sourceObject);

1. **Вывод**
   1. Научиться выполнять вставку, обновление и удаление записей средствами EF Core;
   2. Научиться обеспечивать обратную связь при редактировании данных.

# **Лабораторная работа №19**

**Выполнение SQL-команд и SQL-подпрограмм**

1. **Цель работы** 
   1. Научиться выполнять SQL-команды и вызывать хранимые процедуры и функции пользователя средствами EF Core.
2. **Контрольные вопросы**

Вопрос: Как выполнить команду на выборку данных в EF Core?

Для выполнения команды на выборку данных в EF Core можно использовать следующие методы:

1. LINQ-запросы:

var users = dbContext.Users.Include(u => u.Address).Where(u => u.Age > 18);

2. Raw SQL-запросы:

var users = dbContext.Users.FromSqlRaw("SELECT \* FROM Users WHERE Age > 18");

3. Stored Procedure:

var users = dbContext.Users.FromSqlProcedure("GetUsersOver18");

4. Table-valued function:

var users = dbContext.Users.FromSqlFunction("GetActiveUsers");

Вопрос: Как выполнить команду на модификацию данных в EF Core?

Для модификации данных в EF Core используются следующие методы:

1. Add:

dbContext.Users.Add(new User { Name = "John Doe" });

2. Update:

var user = dbContext.Users.Find(1);

user.Name = "Jane Doe";

dbContext.Entry(user).Property(e => e.Name).IsModified = true;

3. Remove:

dbContext.Users.Remove(userToDelete);

4. Bulk operations:

dbContext.Users.AddRange(new List<User> { /\* ... \*/ });

5. Execute raw SQL:

dbContext.Database.ExecuteSqlRaw("UPDATE Users SET Name = 'Updated' WHERE Id = 1");

Вопрос: Как объявить и передать параметр в SQL-команду в EF Core?

Для передачи параметров в SQL-команду в EF Core можно использовать следующие способы:

1. Using parameterized queries:

var userId = 1;

var name = "John Doe";

var query = $"SELECT \* FROM Users WHERE Id = {userId} AND Name = '{name}'";

var result = dbContext.Users.FromSqlRaw(query);

2. Using DbParameter:

var userIdParam = new SqlParameter("@UserId", 1);

var nameParam = new SqlParameter("@Name", "'John Doe'");

var query = "SELECT \* FROM Users WHERE Id = @UserId AND Name = @Name";

var result = dbContext.Users.FromSqlRaw(query, userIdParam, nameParam);

3. Using FromSqlRaw with named parameters:

var query = "SELECT \* FROM Users WHERE Id = @UserId AND Name = @Name";

var result = dbContext.Users.FromSqlRaw(query, new { UserId = 1, Name = "John Doe" });

Вопрос: Как вызвать табличную функцию в EF Core?

Табличные функции в EF Core вызываются с помощью метода FromSqlFunction [1]:

var result = dbContext.Users.FromSqlFunction("dbo.GetActiveUsers");

Вопрос: Как вызвать хранимую процедуру в EF Core?

Хранимые процедуры в EF Core вызываются с помощью метода FromSqlProcedure [1]:

var result = dbContext.Users.FromSqlProcedure("sp\_GetUserDetails");

Вопрос: Как объявить и передать выходной параметр из хранимой процедуры в EF Core?

Для работы с выходными параметрами в хранившихся процедурах в EF Core можно использовать следующие подходы:

1. Using DbParameter:

var outputParam = new SqlParameter("@OutputParam", SqlDbType.NVarChar, 50);

outputParam.Direction = ParameterDirection.Output;

var result = dbContext.Database.ExecuteSqlRaw("EXEC sp\_GetUserDetails @OutputParam OUTPUT", outputParam);

var outputValue = outputParam.Value.ToString();

2. Using FromSqlRaw with named parameters:

var query = "EXEC sp\_GetUserDetails @OutputParam OUTPUT";

var result = dbContext.Database.ExecuteSqlRaw(query, new { OutputParam = "" });

var outputValue = result.OutputParam;

1. **Вывод**
   1. В ходе лабораторной работы мы научились выполнять SQL-команды и вызывать хранимые процедуры и функции пользователя средствами EF Core.

# **Лабораторная работа №20**

**Реализация разграничения прав доступа пользователей**

1. **Цель работы** 
   1. Научиться разграничивать права доступа пользователей на уровне интерфейса приложения;
   2. Научиться изменять настройки подключения к БД средствами Entity Framework Core.
2. **Контрольные вопросы**

Вопрос: 8.1 Как изменить настройки подключения к БД в клиентском приложении?

Настройки подключения к базе данных в клиентском приложении обычно определяются через конфигурационный файл или переменные окружения. Вот некоторые способы их изменения:

1. Через конфигурационный файл (например, appsettings.json):

{

"ConnectionStrings": {

"DefaultConnection": "Server=myserver;Database=MyDB;User Id=myuser;Password=mypassword;"

}

}

2. Через переменные окружения:

ConnectionStrings:DefaultConnection="Server=myserver;Database=MyDB;User Id=myuser;Password=mypassword;"

3. Через код:

Configuration.GetConnectionString("DefaultConnection") = "Server=newserver;Database=NewDB;User Id=newuser;Password=newpassword;";

```

4. Через пользовательский класс конфигурации:

public class ConnectionStringConfig

{

public string DefaultConnection { get; set; }

}

// В программе:

var connectionString = new ConnectionStringConfig();

connectionString.DefaultConnection = "Server=newserver;Database=NewDB;User Id=newuser;Password=newpassword;";

Вопрос: Какими способами можно обеспечить хранение пользователей и ролей пользователей в БД (отобразить в виде ERD)?

Вот пример ERD для хранения пользователей и их ролей:

1. Таблица Users:

- Id (PK)

- Username

- PasswordHash

- Email

- CreatedDate

2. Таблица Roles:

- Id (PK)

- Name

- Description

3. Таблица UserRoles:

- UserId (FK)

- RoleId (FK)

4. Таблица Permissions:

- Id (PK)

- Name

- Description

5. Таблица RolePermissions:

- RoleId (FK)

- PermissionId (FK)

ERD:

Users (1) ---\* UserRoles (2) ---\* Roles (3)

|---\* Permissions (4) ---\* RolePermissions (5)

|---\* Permissions (4) ---\* RolePermissions (5)

|

---\* UserRoles (2) ---\* Roles (3)

Этот подход позволяет:

- Хранить информацию о пользователях

- Связывать пользователей с ролями

- Связывать роли с разрешениями

- Динамически назначать разрешения пользователям через роли

Вопрос: Что такое «авторизация»?

Авторизация - это процесс проверки прав доступа пользователя к ресурсу или операции [1]. Она определяет, какие действия пользователь может выполнять в системе на основе его ролей или привилегий.

Основные аспекты авторизации:

- Проверка прав доступа пользователя к ресурсам

- Управление доступом к функциям и операциям

- Реализация механизмов безопасности для защиты данных

Вопрос: Что такое «регистрация»?

Регистрация - это процесс создания учетной записи пользователя в системе [1]. Она включает в себя:

- Создание нового пользователя

- Установка уникального идентификатора (например, username или email)

- Создание пароля или установка метода аутентификации

- Заполнение дополнительных полей профиля (если необходимо)

После регистрации пользователь получает возможность входа в систему и использования предоставленных ей возможностей.

Регистрация обычно включает следующие шаги:

1. Ввод персональной информации

2. Выбор пароля

3. Подтверждение регистрации (например, по электронной почте)

4. Активация аккаунта

1. **Вывод**
   1. В ходе лабораторной работы мы научились разграничивать права доступа пользователей на уровне интерфейса приложения;
   2. Научились изменять настройки подключения к БД средствами Entity Framework Core.

# **Лабораторная работа №21**

**Разработка приложения для импорта данных**

1. **Цель работы** 
   1. Импорт данных пользователя в базу данных средствами EF Core.
2. **Контрольные вопросы**

Вот ответы на вопросы о различных форматах файлов и их импорте в базу данных через приложение:

Вопрос: Как выполнить импорт из csv-файла в БД через приложение?

Импорт данных из CSV-файла в базу данных обычно выполняется следующим образом:

1. Чтение CSV-файла:

using (StreamReader reader = new StreamReader(filePath))

{

// Чтение содержимого файла

}

2. Парсинг данных:

string[] line = reader.ReadLine().Split(',');

int id = Convert.ToInt32(line[1]);

string name = line[2];

3. Преобразование данных в объекты:

var entity = new YourEntity { Id = id, Name = name };

4. Сохранение данных в базу:

context.YourEntities.Add(entity);

await context.SaveChangesAsync();

Пример полного кода:

public async Task ImportCsv(string filePath)

{

using (var reader = new StreamReader(filePath))

{

while (!reader.EndOfStream)

{

string line = reader.ReadLine();

string[] values = line.Split(',');

int id = Convert.ToInt32(values[1]);

string name = values[2];

var entity = new YourEntity { Id = id, Name = name };

context.YourEntities.Add(entity);

}

}

await context.SaveChangesAsync();

}

Вопрос: Как импортировать изображение в БД через приложение?

Импорт изображения в базу данных обычно выполняется следующим образом:

1. Чтение файла:

byte[] imageBytes = File.ReadAllBytes(imagePath);

2. Сохранение в базу данных:

entity.ImageData = imageBytes;

await context.SaveChangesAsync();

Пример полного кода:

public async Task ImportImage(string imagePath, int entityId)

{

using (var fileStream = new FileStream(imagePath, FileMode.Open))

{

byte[] imageBytes = await fileStream.ReadAsByteArrayAsync();

var entity = await context.Entities.FindAsync(entityId);

if (entity != null)

{

entity.ImageData = imageBytes;

await context.SaveChangesAsync();

}

}

}

Вопрос: Как выполнить импорт файла JSON в БД через приложение?

Импорт данных из JSON-файла в базу данных обычно выполняется следующим образом:

1. Чтение JSON-файла:

string jsonString = File.ReadAllText(filePath);

2. Десериализация JSON в объекты:

List<YourEntity> entities = JsonSerializer.Deserialize<List<YourEntity>>(jsonString);

3. Преобразование и сохранение в базу:

foreach (var entity in entities)

{

context.YourEntities.Add(entity);

}

await context.SaveChangesAsync();

Пример полного кода:

public async Task ImportJson(string filePath)

{

string jsonString = File.ReadAllText(filePath);

List<YourEntity> entities = JsonSerializer.Deserialize<List<YourEntity>>(jsonString);

foreach (var entity in entities)

{

context.YourEntities.Add(entity);

}

await context.SaveChangesAsync();

}

1. **Вывод**
   1. В ходе лабораторной работы мы научились импортировать данных пользователя в базу данных средствами EF Core.

# **Лабораторная работа №22**

**Разработка приложения для экспорта данных в текстовые файлы**

1. **Цель работы** 
   1. Научиться выполнять экспорт данных из БД в формате текстового файла.
   2. Научиться выполнять экспорт данных из БД с применением Microsoft Office Word.
2. **Контрольные вопросы**

Вопрос: Какое пространство имен требуется подключить для работы с Word?

Для работы с Word в C# обычно используется пространство имен `Microsoft.Office.Interop.Word` [1].

Пример подключения пространства имен:

using Microsoft.Office.Interop.Word;

Однако, стоит отметить, что использование COM-объектов Office может быть сложным и менее надежным, чем современные API.

Вопрос: Какие библиотеки позволяют сохранять данные в формате docx?

Для работы с форматом docx существует несколько библиотек:

1. Open XML SDK: официальная библиотека от Microsoft для работы с открытым форматом XML [2].

2. EPPlus: популярная библиотека для работы с Excel, но также поддерживает docx [2].

3. DocX: специализированная библиотека для работы с docx файлами [2].

4. Aspose.Words: коммерческая библиотека с широкими возможностями [2].

Пример использования EPPlus для сохранения документа:

using OfficeOpenXml;

// ... код для создания документа ...

using (var package = new ExcelPackage(filePath))

{

var worksheet = package.Workbook.Worksheets[1];

worksheet.Cells["A1"].Value = "Hello World";

package.Save();

}

Вопрос: Как выполнить экспорт данных в формат txt?

Для экспорта данных в формат txt можно использовать стандартный метод `File.WriteAllText()` или более продвинутый `StreamWriter` [1].

Пример с File.WriteAllText():

string data = "Line 1\nLine 2\nLine 3";

File.WriteAllText("output.txt", data);

Пример с StreamWriter:

using (StreamWriter writer = new StreamWriter("output.txt"))

{

writer.WriteLine("Line 1");

writer.WriteLine("Line 2");

writer.WriteLine("Line 3");

}

Если нужно экспортировать данные из коллекции, можно использовать циклы:

using (StreamWriter writer = new StreamWriter("output.txt"))

{

foreach (var item in collection)

{

writer.WriteLine($"{item.Id} - {item.Name}");

}

}

При экспорте больших объемов данных рекомендуется использовать буферизованное написание:

using (StreamWriter writer = new StreamWriter("output.txt", true))

{

foreach (var item in collection)

{

writer.Write(item.ToString());

writer.WriteLine();

}

}

1. **Вывод**
   1. В ходе лабораторной работы мы научились выполнять экспорт данных из БД в формате текстового файла.
   2. Научлись выполнять экспорт данных из БД с применением Microsoft Office Word.

# **Лабораторная работа №23**

**Разработка приложения для экспорта данных в табличные файлы**

1. **Цель работы** 
   1. Научиться выполнять экспорт данных из БД в формате табличного файла
   2. Научиться выполнять экспорт данных из БД с применением Microsoft Office Excel.
2. **Контрольные вопросы**

Вопрос: Какое пространство имен требуется подключить для работы с Word?

Для работы с Word в C# обычно используется пространство имен `Microsoft.Office.Interop.Word` [1].

Пример подключения пространства имен:

using Microsoft.Office.Interop.Word;

Однако, стоит отметить, что использование COM-объектов Office может быть сложным и менее надежным, чем современные API.

Вопрос: Какие библиотеки позволяют сохранять данные в формате docx?

Для работы с форматом docx существует несколько библиотек:

1. Open XML SDK: официальная библиотека от Microsoft для работы с открытым форматом XML [2].

2. EPPlus: популярная библиотека для работы с Excel, но также поддерживает docx [2].

3. DocX: специализированная библиотека для работы с docx файлами [2].

4. Aspose.Words: коммерческая библиотека с широкими возможностями [2].

Пример использования EPPlus для сохранения документа:

using OfficeOpenXml;

// ... код для создания документа ...

using (var package = new ExcelPackage(filePath))

{

var worksheet = package.Workbook.Worksheets[1];

worksheet.Cells["A1"].Value = "Hello World";

package.Save();

}

Вопрос: Как выполнить экспорт данных в формат txt?

Для экспорта данных в формат txt можно использовать стандартный метод `File.WriteAllText()` или более продвинутый `StreamWriter` [1].

Пример с File.WriteAllText():

string data = "Line 1\nLine 2\nLine 3";

File.WriteAllText("output.txt", data);

```

Пример с StreamWriter:

using (StreamWriter writer = new StreamWriter("output.txt"))

{

writer.WriteLine("Line 1");

writer.WriteLine("Line 2");

writer.WriteLine("Line 3");

}

Если нужно экспортировать данные из коллекции, можно использовать циклы:

using (StreamWriter writer = new StreamWriter("output.txt"))

{

foreach (var item in collection)

{

writer.WriteLine($"{item.Id} - {item.Name}");

}

}

При экспорте больших объемов данных рекомендуется использовать буферизованное написание:

using (StreamWriter writer = new StreamWriter("output.txt", true))

{

foreach (var item in collection)

{

writer.Write(item.ToString());

writer.WriteLine();

}

}

1. **Вывод**
   1. В ходе лабораторной работы мы научились выполнять экспорт данных из БД в формате табличного файла
   2. Научились выполнять экспорт данных из БД с применением Microsoft Office Excel.

# **Практическая работа №1**

**Разработка web-API для доступа к данным\**

1. **Цель работы** 
   1. Научиться выполнять разработку web-API для доступа к БД.
2. **Контрольные вопросы**

Вопрос: Что такое REST-запрос?

REST (Representational State Transfer) - это архитектурный стиль для распределенных систем [1]. REST-запрос - это HTTP-запрос, который соответствует принципам REST API для взаимодействия с ресурсами.

Ключевые характеристики REST-запроса:

- Использование стандартных HTTP-методов (GET, POST, PUT, DELETE, etc.)

- Stateless взаимодействие между клиентом и сервером

- Идемпотентность (повторные запросы не должны иметь побочных эффектов)

- Кэширование

Вопрос: Что такое RESTful?

RESTful - это описательное слово, которое используется для обозначения API, построенного в соответствии с принципами REST [1].

Характеристики RESTful API:

- Использование стандартных HTTP-методов

- Представление ресурсов через URI

- Адресация ресурсов

- Статус-коды HTTP

- Поддержка HATEOAS (Hypermedia as the Engine of Application State)

Вопрос: Для чего используется метод GET?

HTTP-метод GET используется для получения данных из сервера [1].

Основные применения:

- Получение информации о ресурсе

- Извлечение списка ресурсов

- Проверка состояния ресурса

Пример использования:

GET /api/users/123

Этот запрос получит информацию о пользователе с ID 123.

Вопрос: Для чего используется метод POST?

HTTP-метод POST используется для отправки данных на сервер с целью создания нового ресурса или выполнения операции [1].

Основные применения:

- Создание нового ресурса

- Отправка формы

- Выполнение операции, не имеющей прямого сопоставления с существующим ресурсом

Пример использования:

POST /api/users

Content-Type: application/json

{

"name": "John Doe",

"email": "john@example.com"

}

Этот запрос создаст нового пользователя.

Вопрос: Для чего используется метод PUT?

HTTP-метод PUT используется для замены существующего ресурса на сервере [1].

Основные применения:

- Обновление существующего ресурса полностью

- Замена целевого ресурса новым ресурсом

Пример использования:

PUT /api/users/123

Content-Type: application/json

{

"name": "Jane Doe",

"email": "jane@example.com"

}

Этот запрос обновит информацию пользователя с ID 123.

Вопрос: Для чего используется метод DELETE?

HTTP-метод DELETE используется для удаления существующего ресурса с сервера [1].

Основные применения:

- Удаление ресурса полностью

- Логическое удаление (маркировка как удаленный)

Пример использования:

DELETE /api/users/123

Этот запрос удалит ресурс с ID 123.

1. **Вывод**
   1. В ходе лабораторной работы мы научились выполнять разработку web-API для доступа к БД.

# **Практическая работа №2**

**Вызов REST API**

1. **Цель работы** 
   1. Научиться проверять работоспособность RESTful API в клиентском приложении.
   2. Научиться выполнять тестирование RESTful API методом черного ящика.
2. **Контрольные вопросы**

Вопрос: Как указать у объекта HttpClient базовый адрес?

Для указания базового адреса можно использовать свойство `BaseAddress`:

HttpClient client = new HttpClient();

client.BaseAddress = new Uri("https://api.example.com");

Альтернативно, можно передать адрес при создании экземпляра:

HttpClient client = new HttpClient("https://api.example.com");

Вопрос: Какие коды ответа могут быть получены при Http-запросе (указать коды и значения)?

Вот основные коды ответа HTTP и их значения [1]:

1. 200 OK - запрос успешно обработан

2. 201 Created - ресурс создан

3. 202 Accepted - запрос принят для обработки

4. 204 No Content - запрос успешно обработан, но без содержимого

5. 301 Moved Permanently - ресурс перемещен permanently

6. 302 Found - временное перенаправление

7. 304 Not Modified - ресурс не изменился

8. 400 Bad Request - неправильный запрос

9. 401 Unauthorized - аутентификация требуется

10. 403 Forbidden - доступ запрещен

11. 404 Not Found - ресурс не найден

12. 405 Method Not Allowed - метод недопустим для этого ресурса

13. 500 Internal Server Error - внутренняя ошибка сервера

14. 501 Not Implemented - сервер не реализует функцию

15. 502 Bad Gateway - неверная внешняя связь

16. 503 Service Unavailable - сервис недоступен

17. 504 Gateway Timeout - ожидание ответа от интермедиарного сервера

Вопрос: Какой метод класса HttpClient используется для получения данных?

Метод `GetAsync()` используется для получения данных [1]:

var response = await client.GetAsync("api/resource");

Вопрос: Какой метод класса HttpClient используется для вставки данных?

Метод `PostAsync()` используется для отправки POST-запросов с данными [1]:

var content = new StringContent(JsonConvert.SerializeObject(data), Encoding.UTF8, "application/json");

var response = await client.PostAsync("api/resource", content);

Вопрос: Какой метод класса HttpClient используется для изменения данных?

Метод `PutAsync()` используется для обновления существующего ресурса [1]:

var content = new StringContent(JsonConvert.SerializeObject(data), Encoding.UTF8, "application/json");

var response = await client.PutAsync("api/resource", content);

Вопрос: Какой метод класса HttpClient используется для удаления данных?

Метод `DeleteAsync()` используется для удаления ресурса [1]:

var response = await client.DeleteAsync("api/resource");

При работе с асинхронными операциями важно использовать ключевое слово `async` и `await`:

try

{

var response = await client.GetAsync("api/resource");

response.EnsureSuccessStatusCode();

string responseBody = await response.Content.ReadAsStringAsync();

}

catch (HttpRequestException e)

{

Console.WriteLine($"Error: {e.Message}");

}

1. **Вывод**
   1. В ходе лабораторной работы мы научились проверять работоспособность RESTful API в клиентском приложении.
   2. Научились выполнять тестирование RESTful API методом черного ящика.

# **Практическая работа №3**

**Разграничение прав доступа на уровне REST API**

1. **Цель работы** 
   1. Научиться выполнять разработку web-API для доступа к БД.

Научиться работать с JWT.

1. **Контрольные вопросы**

Вопрос: Атрибуты методов REST для настройки доступа

При разработке REST API для управления доступом можно использовать следующие атрибуты:

- `@GetMapping`: Указывает HTTP-метод GET для получения ресурса.

- `@PostMapping`: Указывает HTTP-метод POST для отправки нового ресурса.

- `@PutMapping`: Указывает HTTP-метод PUT для обновления существующего ресурса.

- `@DeleteMapping`: Указывает HTTP-метод DELETE для удаления ресурса.

- `@PreAuthorize`: Позволяет применять аннотации Spring Security для проверки прав доступа.

- `@RolesAllowed`: Ограничивает доступ только определенным ролям.

- `@AuthenticationPrincipal`: Предоставляет текущего пользователя для проверки прав.

Для неавторизованных пользователей можно использовать:

- `@AnonymousAccessController`

- `@Secured({"ROLE\_ANONYMOUS"})`

Вопрос: Использование JWT

JWT (JSON Web Token) используется для передачи информации в безопасном и зашифрованном формате между клиентом и сервером. Он позволяет:

- Автоматически аутентифицировать пользователя

- Хранить информацию о состоянии сессии

- Обеспечить безопасное хранение чувствительных данных

- Улучшить производительность приложения за счет минимального количества запросов к серверу

Вопрос: Отличие между авторизацией с использованием cookie и JWT

Авторизация с использованием cookie:

- Более простая реализация

- Меньше нагрузки на сеть

- Потенциально более уязвима к атакам типа CSRF

Авторизация с использованием JWT:

- Более безопасна из-за отсутствия прямого обращения к серверу

- Не зависит от сессий на стороне сервера

- Легче масштабировать приложение

- Более сложная реализация

Вопрос: Настройки при создании токена

При создании токена JWT можно указать следующие настройки:

- Выбор алгоритма шифрования (HS256, RS256 и др.)

- Срок действия токена

- Ключи для шифрования и расшифровки

- Дополнительные claims (поля в токене)

- Параметры времени создания и истечения срока действия

Вопрос: Почему использовать метод POST для авторизации

Использование метода POST для авторизации рекомендуется по следующим причинам:

- Метод POST предназначен для отправки данных на сервер

- Он менее подвержен атакам типа CSRF по сравнению с GET-запросами

- Позволяет передавать чувствительные данные безопасно

- Улучшает защиту от повторных запросов (как физических, так и социальных)

- Поддерживает более сложную логику обработки входных данных

Использование POST обеспечивает более надежную и безопасную реализацию механизма авторизации.

1. **Вывод**
   1. В ходе лабораторной работы мы научились выполнять разработку web-API для доступа к БД.

Научились работать с JWT.

# **Практическая работа №4**

**Разработка веб-клиента**

1. **Цель работы** 
   1. Научиться разрабатывать клиентское веб-приложение для доступа к БД.
   2. Научиться проверять работоспособность RESTful API в клиентском приложении.
2. **Контрольные вопросы**

Вопрос: 4.1 Как выполнить HTTP-запрос к API с помощью Fetch API?

Для выполнения HTTP-запроса к API с помощью Fetch API можно использовать следующий код:

fetch('https://api.example.com/data', {

method: 'GET',

headers: {

'Content-Type': 'application/json',

'Authorization': 'Bearer token'

}

})

.then(response => response.json())

.then(data => console.log(data))

.catch(error => console.error('Ошибка:', error));

Ключевые моменты:

- Используется метод fetch()

- Указывается URL API и опции запроса в объекте второго аргумента

- Метод then() используется для обработки ответа

- Метод catch() для обработки ошибок

Вопрос: Как обрабатывать ошибки при вызове API в JavaScript?

Для обработки ошибок можно использовать следующий код:

fetch('https://api.example.com/data')

.then(response => {

if (!response.ok) {

throw new Error(`HTTP error! status: ${response.status}`);

}

return response.json();

})

.then(data => console.log(data))

.catch(error => console.error('Ошибка:', error));

Ключевые моменты:

- Проверка статуса ответа с помощью response.ok

- Выбрасывание исключения при неудачном статусе

- Обработка ошибок с помощью catch()

Вопрос: Что такое асинхронные функции и как они связаны с вызовом API?

Асинхронные функции позволяют выполнять операции без блокировки основного потока выполнения. Они связаны с вызовом API тем, что многие операции с API (например, сетевые запросы) являются асинхронными.

Пример асинхронной функции:

async function fetchData() {

try {

const response = await fetch('https://api.example.com/data');

const data = await response.json();

console.log(data);

} catch (error) {

console.error('Ошибка:', error);

}

}

fetchData();

Ключевые моменты:

- Использование ключевого слова async

- Оператор await для ожидания асинхронных операций

- Обработка ошибок с помощью try/catch

Вопрос: Что такое Razor Pages?

Razor Pages - это модель разработки в ASP.NET Core, которая позволяет создавать веб-страницы с использованием C# и HTML. Она предоставляет простой способ создания веб-страниц без необходимости использования контроллеров.

Основные характеристики:

- Использование C# для логики и HTML для представления

- Автоматическое маршрутизирование на основе URL

- Встроенная поддержка AJAX

Вопрос: Как создать и настроить новую страницу Razor в ASP.NET Core?

Для создания новой страницы Razor:

1. Создайте новый класс PageModel, наследующий от PageModel:

public class NewPageModel : PageModel

{

public void OnGet()

{

// Логика загрузки страницы

}

public IActionResult OnPost()

{

// Логика обработки POST-запроса

return Page();

}

}

2. Создайте файл Razor (.cshtml) в папке Pages с соответствующим именем:

@page

@model YourNamespace.NewPageModel

<h1>Новая страница</h1>

<form method="post">

<button type="submit">Отправить</button>

</form>

3. Настройте маршрутизацию в Startup.cs:

app.MapRazorPages();

Ключевые моменты:

- Использование @page директивы в файле Razor

- Наследование от PageModel

- Определение методов OnGet() и OnPost() для обработки запросов

- Простота создания и структурирования кода

Этот ответ охватывает основные аспекты работы с API в JavaScript, обработки ошибок, асинхронных функций, Razor Pages и создания новых страниц Razor в ASP.NET Core.

1. **Вывод**
   1. В ходе лабораторной работы мы научились разрабатывать клиентское веб-приложение для доступа к БД.
   2. Научились проверять работоспособность RESTful API в клиентском приложении.

.