

# **Тема 16. Основы работы с MySQL в Windows Forms.**

# **Цель занятия:**

**Получить навыки взаимодействия приложения с базой данных, используя прямые SQL-запросы.**

# **Учебные вопросы:**

- 1. Подготовка к работе.**
- 2. Подключение Windows Forms к MySQL.**
- 3. Выполнение основных SQL-запросов в Windows Forms.**
- 4. Асинхронные запросы к БД.**

# 1. Подготовка к работе.

Каждое приложение, взаимодействующее с базой данных, должно установить соединение с сервером базы данных.

Это осуществляется через драйверы или библиотеки, которые обеспечивают возможность подключения к различным системам управления базами данных (СУБД), включая MySQL.

**MySql.Data** — это официальный драйвер MySQL для .NET, который позволяет взаимодействовать с базами данных MySQL из приложений на C#.

Этот драйвер предоставляет классы и методы для выполнения SQL-запросов, управления соединениями и работы с данными.

## Установка MySql.Data.

Перед установкой библиотеки убедитесь, что:

- У вас установлен Visual Studio (Community, Professional или Enterprise).
- Создан новый проект Windows Forms (.NET Framework или .NET Core/5+).

## Установка MySql.Data через NuGet Package Manager

**NuGet** — это менеджер пакетов для .NET, который позволяет легко добавлять сторонние библиотеки в проект.

### **Шаг 1:** Откройте NuGet Package Manager

В Visual Studio откройте ваш проект.

Перейдите в меню Tools → NuGet Package Manager → Manage NuGet Packages for Solution .

### **Шаг 2:** Поиск и установка MySql.Data

В открывшемся окне перейдите на вкладку Browse .

В поле поиска введите MySql.Data.

Найдите пакет MySql.Data, разработанный Oracle.

Выберите ваш проект в правой части окна.

Нажмите кнопку Install , чтобы установить пакет.

### **Шаг 3:** Подтверждение установки

После нажатия Install появится диалоговое окно с предложением подтвердить изменения.

Нажмите ОК для завершения установки.

Обзор

Установлено

Обновления

MySQL.Data x ↺

☐ Включить предварительные версии

Источник пакета: nuget.org ⚙

## Диспетчер пакетов NuGet: lect12

**MySQL.Data** ✓ автор: Oracle Corporation, Скачиваний: 1 9.2.0  
MySQL.Data.MySqlClient .Net Core Class Library**MySQL.Data.EntityFrameworkCore** ✓ автор: C ⚠ 8.0.22  
MySQL.Data.EntityFrameworkCore for Entity Framework.  
Эта версия пакета является нерекомендуемой.**Pomelo.EntityFrameworkCore.MySql** автор: Laure 8.0.2  
Pomelo's MySQL database provider for Entity Framework Core.

Все пакеты лицензируются их владельцами. NuGet не несет ответственности за пакеты сторонних производителей и не предоставляет лицензии на такие пакеты.

☐ Больше не показывать**MySQL.Data** ✓

nuget.org

Версия: Последняя стабильная 9.2.0

Установить

ⓘ Сопоставление источника пакета отключено. Настроить

⌵ Параметры

## Описание

MySQL.Data.MySqlClient .Net Core Class Library

Версия: 9.2.0

Авторы: Oracle Corporation

Лицензия: GPL-2.0-only WITH  
Universal-FOSS-exception-1.0

Файл сведений: Просмотреть файл сведений

Обозреватель решений

Обозреватель решений

Обозреватель решений

Решение "lect12"

C# lect12

Properties

Ссылки

App.config

Form1.cs

C# Program.cs

Обозревате... Измене

Свойства



## 2. Подключение Windows Forms к MySQL.

### Подключение к базе данных

Подключение обычно требует указания следующих параметров:

- **Хост** — адрес сервера базы данных.
- **Порт** — порт для подключения (по умолчанию для MySQL — 3306).
- **Имя пользователя и пароль** для авторизации.
- **Имя базы данных** — указание конкретной базы данных, с которой планируется работа.



Перед началом убедитесь, что:

- Установлен MySQL Server .
- Установлен MySQL Workbench (для управления базой данных).
- Установлена библиотека MySql.Data через NuGet.
- Создана тестовая база данных и таблица.

## Создание строки подключения.

Строка подключения содержит информацию для подключения к базе данных MySQL. Она включает:

- Адрес сервера (server).
- Порт (port). Можно не указывать, если используется стандартный 3306.
- Имя пользователя (user).
- Пароль (password).
- Название базы данных (database).

Пример строки подключения:

```
string cs = "server=localhost;port=3306;user=root;database=testdb;password=yourpassword;"
```

Или:

```
string cs = "server=localhost;user=root;database=testdb;password=yourpassword;"
```

Для удаленного сервера:

```
string cs = "server=192.168.1.100;port=3307;user=root;database=testdb;password=yourpassword;"
```

Если вы не знаете, какой порт используется вашим сервером MySQL, выполните:

```
mysql> SHOW VARIABLES LIKE 'port';  
+-----+-----+  
| Variable_name | Value |  
+-----+-----+  
| port          | 3306  |  
+-----+-----+  
1 row in set, 1 warning (0.02 sec)  
  
mysql>
```

# Проверка соединения с базой данных MySQL

Проверка соединения с базой данных позволяет убедиться, что строка подключения настроена правильно, сервер доступен, и учётные данные верны.

**Шаг 1:** В файл Form1.cs нужно добавить:

```
using MySql.Data.MySqlClient;
```

## Шаг 2: Метод, который пытается открыть соединение с базой данных:

```
private string TestConnection(string connectionString)
{
    using (MySQLConnection connection = new MySQLConnection(connectionString))
    {
        try
        {
            connection.Open();
            return "Подключение успешно";
        }
        catch (Exception ex)
        {
            return ex.Message;
        }
    }
}
```

## **Пояснение к коду:**

### **Создание объекта MySqlConnection:**

- Метод принимает строку подключения (connectionString) как параметр.
- На основе этой строки создается объект MySqlConnection, который представляет соединение с базой данных MySQL.

### **Блок using:**

- Блок using гарантирует, что ресурсы (в данном случае соединение с базой данных) будут корректно освобождены, даже если произойдет исключение.
- После выхода из блока using метод Dispose() автоматически вызывается для объекта connection, что закрывает соединение.

### **Попытка открытия соединения:**

- Метод connection.Open() пытается установить соединение с базой данных.
- Если соединение успешно установлено, метод возвращает строку "Подключение успешно".

### **Обработка ошибок:**

- Если возникает исключение (например, неправильная строка подключения, сервер недоступен или проблемы с аутентификацией), оно перехватывается блоком catch.
- В этом случае метод возвращает текст ошибки из свойства Message объекта исключения (ex.Message).

### **Возврат результата:**

- Метод возвращает строку, которая содержит либо сообщение об успешном подключении, либо описание ошибки.

**Шаг 3:** Проверка соединения при загрузке формы.  
Добавим вызов метода в метод Load:

```
private void Form1_Load(object sender, EventArgs e)
{
    string msg = TestConnection(connString);
    string statusMsg = msg.Length > 50 ? msg.Substring(0, 50) + "... " : msg;
    toolStripStatusLabel1.Text = statusMsg;
}
```



## **Пояснение к коду:**

### **Событие Form1\_Load:**

- Метод Form1\_Load вызывается автоматически при загрузке формы (Form1).
- Это событие часто используется для выполнения начальных настроек, таких как проверка подключения к базе данных, загрузка данных или инициализация элементов управления.

### **Вызов метода TestConnection:**

- Метод TestConnection проверяет подключение к базе данных, используя строку подключения (connString).
- Предполагается, что этот метод возвращает строку:
- "Подключение успешно", если соединение установлено.
- Сообщение об ошибке, если подключение не удалось.

### **Обрезка сообщения:**

- Если сообщение слишком длинное (более 50 символов), оно обрезается до 50 символов, и добавляется многоточие (...).
- Это делается для того, чтобы текст помещался в элемент toolStripStatusLabel1, который обычно имеет ограниченную ширину.

### **Установка текста в toolStripStatusLabel1:**

- Элемент toolStripStatusLabel1 является частью компонента StatusStrip, который обычно находится в нижней части формы.
- Текст, содержащий результат проверки подключения, отображается в этом элементе.

# 3. Выполнение основных SQL-запросов в Windows Forms.

**Шаг 1.** Перед началом убедитесь, что:

- Установлен MySQL Server
- Создана тестовая база данных и таблица.

Создайте новый проект Windows Forms в Visual Studio.

Установите библиотеку MySql.Data через NuGet.

Добавьте элемент DataGridView на форму через конструктор.

Добавьте кнопку для загрузки данных.

В файле Form1.cs добавьте:

```
using MySql.Data.MySqlClient;  
using System.Data;
```



Table Name:

users

Schema:

students

Charset/Collation:

utf8mb4






utf8mb4\_0900\_ai\_ci



Engine:

InnoDB

Comments:

Column Name	Datatype	PK	NN	UQ	B	UN	ZF	AI	G	Default/Expression
 id	INT	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input type="checkbox"/>	<input checked="" type="checkbox"/>	<input type="checkbox"/>	<input checked="" type="checkbox"/>	<input type="checkbox"/>	
 name	VARCHAR(45)	<input type="checkbox"/>	<input checked="" type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	
 rating	DECIMAL(2,2)	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	NULL
		<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	

Column Name:

Data Type:

Charset/Collation:

Default Charset



Default Collation



Default:

Comments:

Storage:

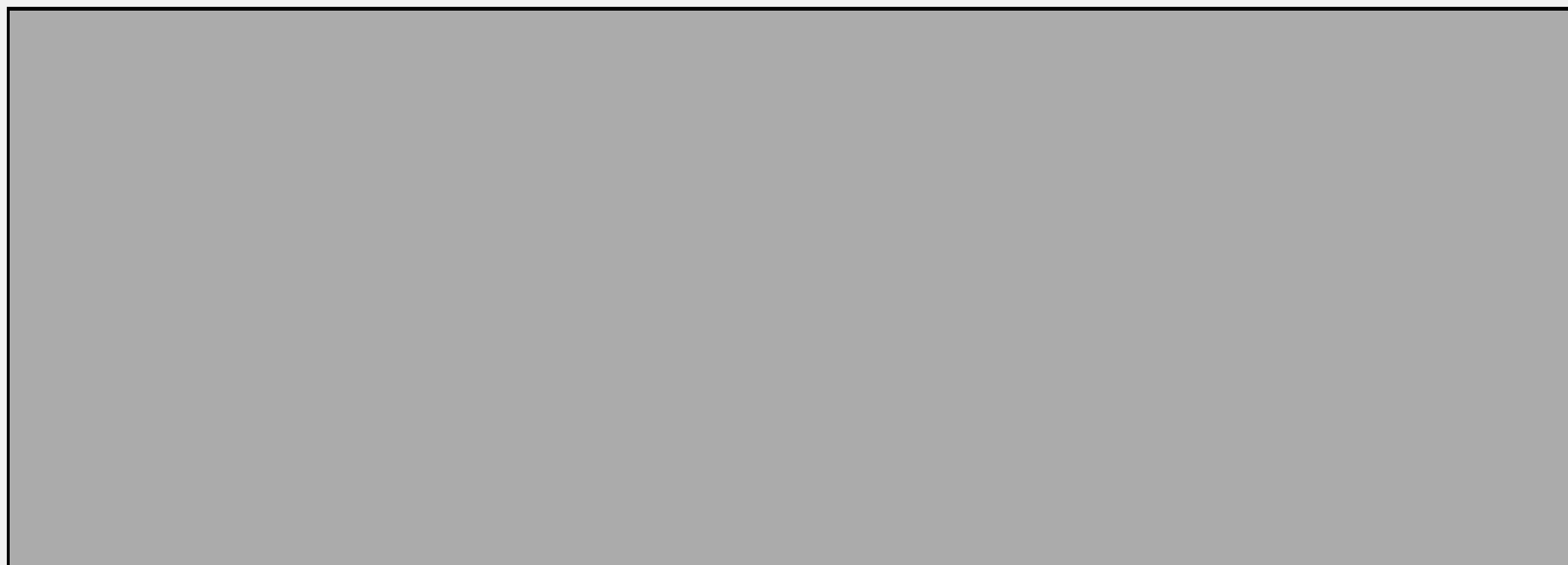
☐ Virtual

☐ S

☐ Primary Key

☐ N

Form1



Id

Name

Rating

SELECT

INSERT

UPDATE

DELETE

status



## Шаг 2: Строку подключения к базе данных:

```
public partial class Form1 : Form
{
    string connString =
        "server=localhost;user=root;database=students;password=1234;";
}
```

### Шаг 3: Метод который выполняет запрос SELECT и возвращает данные в виде DataTable:

```
private DataTable SelectQuery(string query)
{
    DataTable dataTable = new DataTable();

    using (MySqlConnection connection = new MySqlConnection(connString))
    {
        try
        {
            connection.Open();
            MySqlCommand command = new MySqlCommand(query, connection);
            MySqlDataAdapter adapter = new MySqlDataAdapter(command);
            adapter.Fill(dataTable); // Заполняем DataTable данными
        }
        catch (Exception ex)
        {
            MessageBox.Show("Ошибка: " + ex.Message);
        }
    }
    return dataTable;
}
```

## Объяснение работы метода

### Создание объекта DataTable:

- Объект `DataTable` используется для хранения данных, полученных из базы данных. Это удобная структура данных для работы с табличными данными.

### Создание объекта MySqlConnection:

- Метод принимает строку подключения (`connString`) как параметр.
- На основе этой строки создается объект `MySqlConnection`, который представляет соединение с базой данных MySQL.

### Блок using:

- Блок `using` гарантирует, что ресурсы (в данном случае соединение с базой данных) будут корректно освобождены, даже если произойдет исключение.
- После выхода из блока `using` метод `Dispose()` автоматически вызывается для объекта `connection`, что закрывает соединение.

### Попытка открытия соединения:

- Метод `connection.Open()` пытается установить соединение с базой данных.
- Если соединение успешно установлено, выполняется SQL-запрос.

## **Выполнение SQL-запроса:**

- Создается объект MySqlCommand, который представляет SQL-запрос.
- Затем создается объект MySqlDataAdapter, который используется для выполнения запроса и заполнения DataTable данными из базы данных.
- Метод adapter.Fill(dataTable) выполняет запрос и заполняет DataTable результатами.

## **Обработка ошибок:**

- Если возникает исключение (например, неправильный SQL-запрос, проблемы с подключением или отсутствие данных), оно перехватывается блоком catch.
- В этом случае выводится сообщение об ошибке через MessageBox.

## **Возврат результата:**

- Метод возвращает объект DataTable, содержащий данные из базы данных. Если произошла ошибка, DataTable будет пустым.



**Шаг 4:** Обработчик события для кнопки, который загружает данные и отображает их в DataGridView:

```
private void button1_Click(object sender, EventArgs e)
{
    string query = "SELECT * FROM users"; // SQL-запрос
    DataTable dataTable = SelectQuery(query);

    if (dataTable.Rows.Count > 0)
    {
        // Привязка данных к DataGridView
        dataGridView1.DataSource = dataTable;
    }
    else
    {
        MessageBox.Show("Данные не найдены.");
    }
}
```

## Объяснение работы метода

### SQL-запрос:

- Строка `string query = "SELECT * FROM users";` определяет SQL-запрос.
- Запрос выбирает все записи из таблицы `users`. Звездочка (\*) означает, что выбираются все столбцы таблицы.

### Выполнение запроса:

- Метод `SelectQuery(query)` выполняет SQL-запрос и возвращает результат в виде объекта `DataTable`.
- Предполагается, что метод `SelectQuery` уже реализован и обрабатывает подключение к базе данных, выполнение запроса и возможные ошибки.

### Проверка наличия данных:

- Свойство `dataTable.Rows.Count` возвращает количество строк в таблице.
- Если `dataTable.Rows.Count > 0`, это означает, что запрос вернул данные.

### Привязка данных к `DataGridView`:

- Если данные найдены, они привязываются к элементу управления `dataGridView1` через свойство `DataSource`.
- Это автоматически отображает данные в таблице на форме.

### Обработка случая отсутствия данных:

- Если `dataTable.Rows.Count == 0`, это означает, что запрос не вернул ни одной строки.
- В этом случае выводится сообщение пользователю через `MessageBox`.



Form1



	id	name	rating
▶	3	Иванов	70,00
	4	Степанов	80,00
	5	Егоров	95,00
✱			

Id

Name

Rating

SELECT

INSERT

UPDATE

DELETE

Подключение успешно



Выполнение INSERT, UPDATE, DELETE через элементы управления (TextBox, Button)

## Настройка формы

Создайте форму с элементами управления:

Добавьте текстовые поля (TextBox) для ввода данных:

- **textName** (имя).
- **textRating** (средний балл)

Добавьте кнопки:

- **btnInsert** (для добавления данных).
- **btnUpdate** (для обновления данных).
- **btnDelete** (для удаления данных).

## Универсальный метод для выполнения SQL-запросов:

```
private void ExecuteQuery(string query, params MySqlParameter[] parameters)
{
    using (MySqlConnection connection = new MySqlConnection(connString))
    {
        try
        {
            connection.Open();
            MySqlCommand command = new MySqlCommand(query, connection);
            command.Parameters.AddRange(parameters);
            command.ExecuteNonQuery(); // Выполняем запрос
            MessageBox.Show("Операция выполнена успешно!");
        }
        catch (Exception ex)
        {
            MessageBox.Show("Ошибка: " + ex.Message);
        }
    }
}
```

## Объяснение работы метода

### Параметры метода:

- `query`: Строка, содержащая SQL-запрос (например, `INSERT`, `UPDATE`, `DELETE`).
- `parameters`: Массив параметров (`MySqlParameter[]`), которые используются для безопасной передачи значений в запрос (защита от SQL-инъекций).

### Создание соединения:

- Объект `MySqlConnection` создается с использованием строки подключения `connString`.
- Блок `using` гарантирует, что соединение будет корректно закрыто после завершения работы, даже если произойдет исключение.

### Открытие соединения:

- Метод `connection.Open()` пытается установить соединение с базой данных.
- Если соединение не удастся открыть, выбрасывается исключение, которое перехватывается блоком `catch`.

## **Создание команды:**

- Объект MySqlCommand создается для выполнения SQL-запроса.
- Параметры добавляются к команде через метод `command.Parameters.AddRange(parameters)`.

## **Выполнение запроса:**

- Метод `command.ExecuteNonQuery()` выполняет запрос, который не возвращает данные (например, INSERT, UPDATE, DELETE).
- Этот метод возвращает количество затронутых строк, но в данном случае результат не используется.

## **Успешное выполнение:**

- Если запрос выполнен успешно, текст "Операция выполнена успешно!" устанавливается в элемент `toolStripStatusLabel1`.

## **Обработка ошибок:**

- Если возникает исключение (например, неправильный запрос, проблемы с подключением или некорректные параметры), оно перехватывается блоком `catch`.
- Сообщение об ошибке обрезается до 50 символов (если оно длиннее), чтобы оно поместилось в элементе `toolStripStatusLabel1`.

## Добавление данных (INSERT)

Обработчик события для кнопки **btnInsert**:

```
private void btnInsert_Click(object sender, EventArgs e)
{
    string query = "INSERT INTO users (name, rating) VALUES (@name, @rating)";
    MySqlParameter nameParam = new MySqlParameter("@name", txtName.Text);
    MySqlParameter ratingParam = new MySqlParameter("@rating", decimal.Parse(txtRating.Text));

    ExecuteQuery(query, nameParam, ratingParam);
}
```



## Объяснение работы метода

### Событие btnInsert\_Click:

- Этот метод вызывается при нажатии на кнопку (btnInsert) на форме.
- Он используется для выполнения операции вставки данных в таблицу users.

### SQL-запрос:

- Строка `string query = "INSERT INTO users (name, rating) VALUES (@name, @rating)";` определяет SQL-запрос для вставки данных.
- Параметры `@name` и `@rating` используются для безопасной передачи значений в запрос. Это защищает от SQL-инъекций.

### Создание параметров:

- `MySqlParameter nameParam = new SqlParameter("@name", txtName.Text);`
- Создается параметр `@name`, значение которого берется из текстового поля `txtName`.
- `MySqlParameter ratingParam = new SqlParameter("@rating", decimal.Parse(txtRating.Text));`
- Создается параметр `@rating`, значение которого преобразуется из текстового поля `txtRating` в тип `decimal`.
- Преобразование выполняется с помощью метода `decimal.Parse`. Если в поле `txtRating` введено некорректное значение, это вызовет исключение.

### Выполнение запроса:

- Метод `ExecuteQuery(query, nameParam, ratingParam)` выполняет SQL-запрос с использованием переданных параметров.



Form1



	id	name	rating
▶	3	Иванов	70,00
	4	Степанов	80,00
	5	Егоров	95,00
	6	Петров	55,00
✱			

Id

Name

Rating

SELECT

INSERT

UPDATE

DELETE

Операция выполнена успешно!



# Обновление данных (UPDATE)

Обработчик события для кнопки btnUpdate:

```
private void btnUpdate_Click(object sender, EventArgs e)
{
    string query = "UPDATE users SET " +
        "rating = @rating" +
        " WHERE id = @id";
    MySqlParameter idParam = new MySqlParameter("@id", int.Parse(txtId.Text));
    MySqlParameter ratingParam = new MySqlParameter("@rating", decimal.Parse(txtRating.Text));

    ExecuteQuery(query, idParam, ratingParam);
}
```

## Объяснение работы метода

### Событие btnUpdate\_Click:

- Этот метод вызывается при нажатии на кнопку (btnUpdate) на форме.
- Он используется для выполнения операции обновления данных в таблице users.

### SQL-запрос:

- Строка `string query = "UPDATE users SET rating = @rating WHERE id = @id";` определяет SQL-запрос для обновления данных.
- Параметры `@rating` и `@id` используются для безопасной передачи значений в запрос. Это защищает от SQL-инъекций.

### Создание параметров:

- `MySqlParameter idParam = new SqlParameter("@id", int.Parse(txtId.Text));`
- Создается параметр `@id`, значение которого преобразуется из текстового поля `txtId` в тип `int`.
- Если в поле `txtId` введено некорректное значение, это вызовет исключение.
- `MySqlParameter ratingParam = new SqlParameter("@rating", decimal.Parse(txtRating.Text));`
- Создается параметр `@rating`, значение которого преобразуется из текстового поля `txtRating` в тип `decimal`.
- Преобразование выполняется с помощью метода `decimal.Parse`. Если в поле `txtRating` введено некорректное значение, это вызовет исключение.

### Выполнение запроса:

- Метод `ExecuteQuery(query, idParam, ratingParam)` выполняет SQL-запрос с использованием переданных параметров.



Form1



	id	name	rating
▶	3	Иванов	70,00
	4	Степанов	80,00
	5	Егоров	80,00
✱			

Id

5

Name

Rating

80

SELECT

INSERT

UPDATE

DELETE

Операция выполнена успешно!



## Удаление данных (DELETE)

Обработчик события для кнопки btnDelete:

```
private void btnDelete_Click(object sender, EventArgs e)
{
    string query = "DELETE FROM users WHERE id = @id";
    MySqlParameter idParam = new MySqlParameter("@id", int.Parse(txtId.Text));

    ExecuteQuery(query, idParam);
}
```

## Объяснение работы метода

### Событие `btnDelete_Click`:

- Этот метод вызывается при нажатии на кнопку (`btnDelete`) на форме.
- Он используется для выполнения операции удаления записи из таблицы `users`.

### SQL-запрос:

- Строка `string query = "DELETE FROM users WHERE id = @id";` определяет SQL-запрос для удаления данных.
- Параметр `@id` используется для безопасной передачи значения идентификатора в запрос. Это защищает от SQL-инъекций.

### Создание параметра:

- `MySqlParameter idParam = new SqlParameter("@id", int.Parse(txtId.Text));`
- Создается параметр `@id`, значение которого преобразуется из текстового поля `txtId` в тип `int`.
- Если в поле `txtId` введено некорректное значение, это вызовет исключение.

### Выполнение запроса:

- Метод `ExecuteQuery(query, idParam)` выполняет SQL-запрос с использованием переданного параметра.

	id	name	rating
▶	3	Иванов	70,00
	4	Степанов	80,00
✱			

Id	Name	Rating
<input type="text" value="5"/>	<input type="text"/>	<input type="text"/>
<input type="button" value="SELECT"/>	<input type="button" value="INSERT"/>	<input type="button" value="UPDATE"/>
		<input type="button" value="DELETE"/>



## 4. Асинхронные запросы к БД.

Работа с базой данных в приложениях может занимать значительное время, особенно если запросы выполняются на удалённом сервере или обрабатывают большие объёмы данных. Если выполнение запроса происходит синхронно (в основном потоке), это может привести к "заморозке" пользовательского интерфейса (UI).

Чтобы избежать этого, используются асинхронные запросы.

В .NET для работы с асинхронными операциями используются методы с суффиксом `Async` (например, `OpenAsync`, `ExecuteNonQueryAsync`, `FillAsync`).

Эти методы позволяют выполнять операции в фоновом потоке, не блокируя основной поток UI.

## Преимущества асинхронных запросов

- Отсутствие блокировки UI. Пользовательский интерфейс остаётся отзывчивым, пока выполняется запрос.
- Улучшение производительности. Основной поток освобождается для выполнения других задач, что повышает общую производительность приложения.
- Масштабируемость. Асинхронные операции лучше подходят для многопользовательских приложений и высоконагруженных систем.

# Как сделать метод асинхронным?

Для создания асинхронного метода:

- Добавьте ключевое слово **async** перед определением метода.
- Используйте методы с суффиксом Async (например, OpenAsync).
- Возвращайте Task или Task<T> вместо обычного типа данных.

## Пример, TestConnection для работы асинхронно:

```
private async Task<string> TestConnectionAsync(string connectionString)
{
    // Создаём объект MySqlConnection.
    using (MySqlConnection connection = new MySqlConnection(connectionString))
    {
        try
        {
            // Открываем соединение асинхронно.
            await connection.OpenAsync();

            // Возвращаем сообщение об успешном подключении.
            return "Подключение успешно";
        }
        catch (Exception ex)
        {
            // Возвращаем сообщение об ошибке.
            return ex.Message;
        }
    }
}
```

## Объяснение изменений

### Ключевое слово `async`:

- Метод помечен как `async`, что позволяет использовать ключевое слово `await` внутри него.

### Метод `OpenAsync`:

- Вместо синхронного метода `Open()` используется асинхронный метод `OpenAsync()`.
- Ключевое слово `await` ожидает завершения асинхронной операции, не блокируя основной поток.

### Возвращаемый тип `Task<string>`:

- Метод теперь возвращает `Task<string>` вместо `string`. Это указывает, что метод выполняется асинхронно и возвращает результат типа `string`.

### Использование `await`:

- Ключевое слово `await` приостанавливает выполнение метода до завершения асинхронной операции, но не блокирует основной поток.

## Как вызвать асинхронный метод?

При вызове асинхронного метода важно учитывать, что он возвращает `Task`. Вы можете дождаться его завершения с помощью `await` или запустить его в фоновом потоке.

## Пример вызова в событии кнопки:

```
private async void btnTestConnection_Click(object sender, EventArgs e)
{
    // Запускаем асинхронный метод и ждём его завершения.
    string result = await TestConnectionAsync(connString);

    // Обновляем элемент управления с результатом.
    toolStripStatusLabel1.Text = result.Length > 50 ? result.Substring(0, 50) + "..." : result;
}
```

# Домашнее задание:

Задание 1: Подключение к базе данных MySQL

Создайте Windows Forms приложение, которое подключается к локальной базе данных MySQL

Реализуйте форму с DataGridView для отображения данных из выбранной таблицы

Добавьте индикатор статуса подключения к БД в StatusStrip



## Задание 2: CRUD операции:

Разработайте форму для управления данными студентов (Student) со следующими полями:

- ID (автоинкремент)
- Имя
- Фамилия
- Дата рождения
- Средний балл

Реализуйте все CRUD операции (Create, Read, Update, Delete) через отдельные кнопки.

Обеспечьте валидацию вводимых данных

Используйте параметризованные запросы для защиты от SQL-инъекций

Напишите код для выполнения SQL-запроса `SELECT * FROM users` и отобразите результаты в `DataGridView` на форме.

Добавьте кнопку на форму, которая будет выполнять запрос и обновлять данные в `DataGridView`.

### Задание 3: Асинхронная работа:

Модифицируйте предыдущее приложение, сделав все операции с БД асинхронными

Добавьте индикатор загрузки (ProgressBar или специальную анимацию), который будет отображаться во время выполнения запросов

Обеспечьте корректную обработку ошибок при асинхронных операциях

# Список литературы:

1. [MySQL и C# - работаем с базой из программы](#)
2. [Подключение к БД MySQL через C#](#)

# Материалы лекций:

<https://github.com/ShViktor72/Education2025>