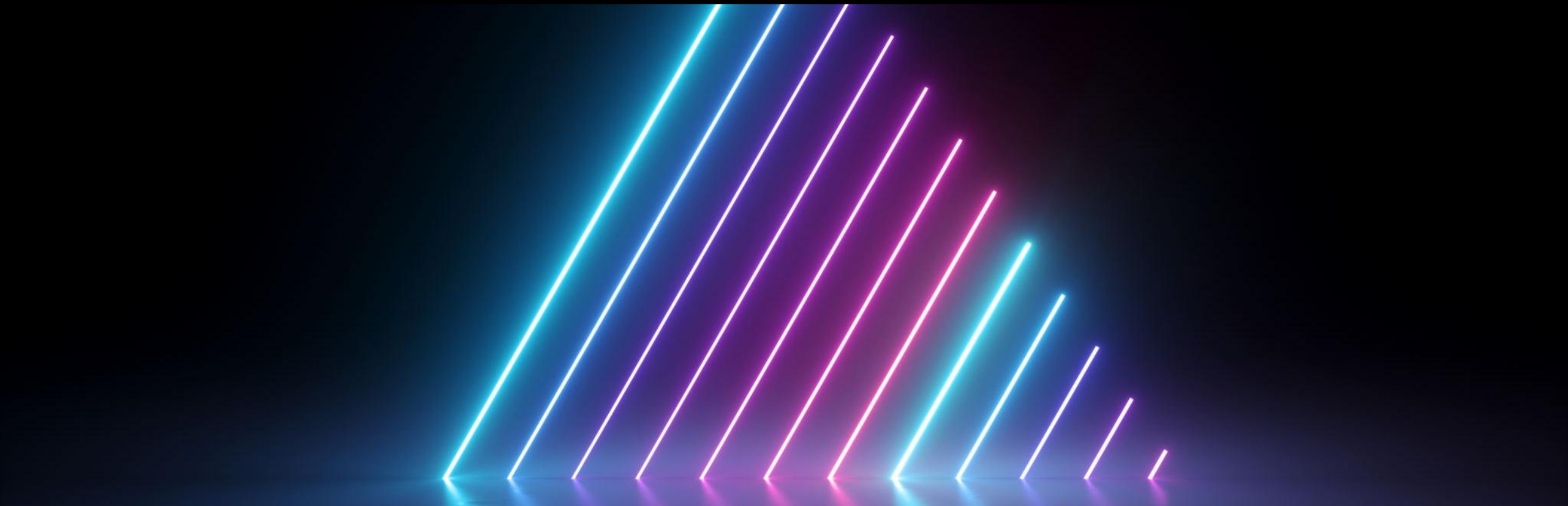
CSharp

08.04.2025



Plan für die Woche

SMART INDUSTRY CAMPUS

Montag	Dienstag	Mittwoch	Donnerstag	Freitag
• Wiederholung Java vs. C# • Übungen	• GUI-Programmierung in C# mit Hilfe der Toolbox in VS • Implementierung der GUI um Bücher hinzuzufügen	• Datenbankanbindung mit C# • Implementierung der DAO-Schicht in C#	• Spiele-Projekt	• Spiele-Projekt

Plan für heute

- ✓ Wie kann ich eine MySQL-Anbindung in VS herstellen?
- ✓ Wie erstelle ich das Schichtenmodell in VS?
- ✓ Übertrag Library-“Projekt“ von Java auf C# (ohne UI)

Wie baut man eine Verbindung auf?

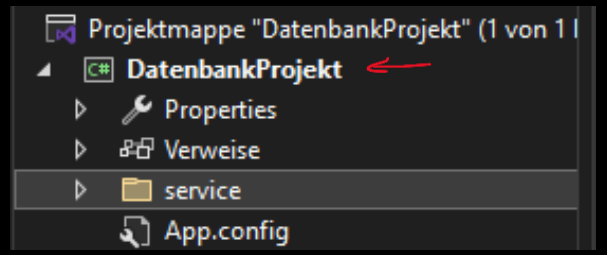
NuGet

→ In Visual Studio kann man mithilfe des NuGet Bibliotheken erstellen, teilen und verwenden

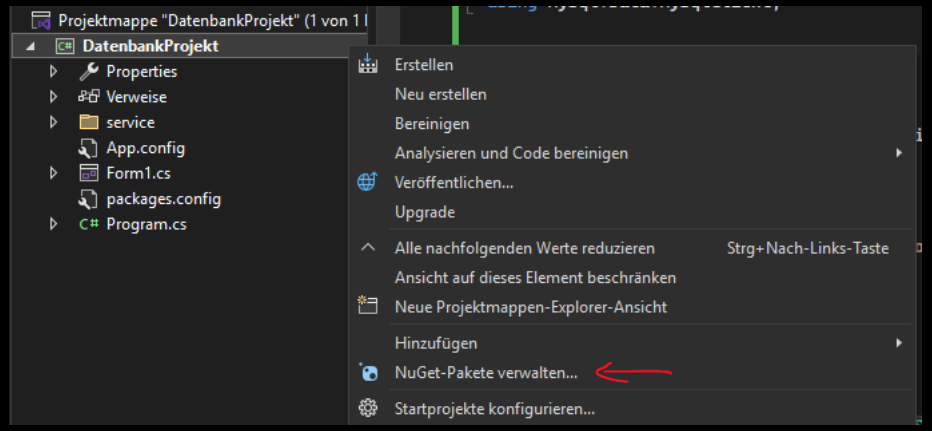
→ Mit Hilfe von NuGet ist es möglich, den MySQL-Connector zu holen, der benötigt wird, um mit Datenbanken zu arbeiten

Wie öffne ich NuGet?

→ In Visual Studio mit Rechtsklick auf das Projekt

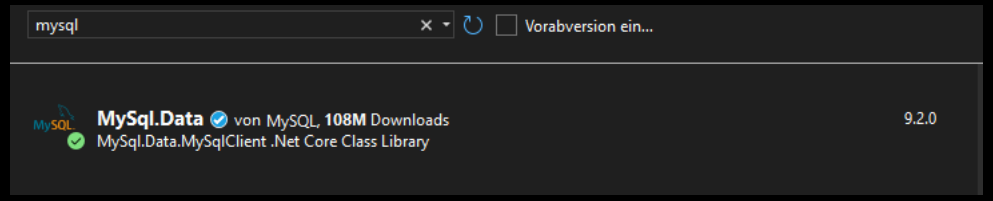


→ NuGet-Pakete verwalten drücken



→ MySQL suchen und MySql.Data auswählen

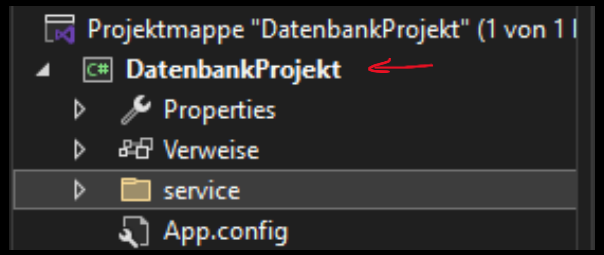
→ Installieren drücken



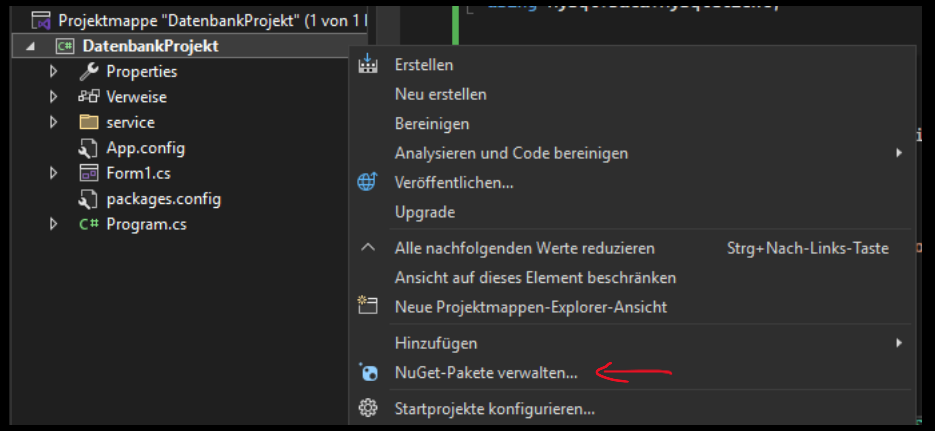
Wie öffne ich NuGet?

MySql kann jetzt genutzt werden

→ In Visual Studio mit Rechtsklick auf das Projekt

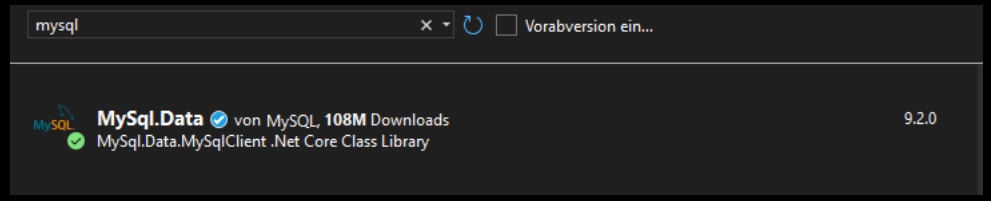


→ NuGet-Pakete verwalten drücken



→ MySQL suchen und MySql.Data auswählen

→ Installieren drücken



Grundlegende Sachen für C# und MySql

MySqlConnection

- → In Java wird der DriverManager für die Verbindungen genutzt
  Der DriverManager hat dann eine Connection zurückgegeben
- → In C# und der MySql.Data Bibliothek wird MySqlConnection als Klasse genutzt
- → Es werden nicht einzelne Parameter für URL, USER, PASSWORD übergeben, sondern:
- → ein String, der alles enthält:
  - → "Server=localhost;Database=library;User ID=root;Password=;"
- → Die MySqlConnection-Klasse hat zwei Methoden:
  - → Open() – zum Öffnen der Verbindung
  - → Close() – zum Schließen der Verbindung
- → Beim Öffnen der Verbindung sollte mit try/catch gearbeitet werden!

Aufgabe

- → Öffnet das Projekt von gestern, in der ihr eine Oberfläche zum Hinzufügen eines Buches gebaut hattet
- → Erstellt ein Package singleton
  - → Projekt – Rechtsklick – Hinzufügen – Neuer Ordner – Name: singleton
- → Erstellt eine **STATISCHE** Klasse DBConnector (kann somit nur statische Elemente beinhalten)
  - → Nutzt die Klasse MySqlConnection, um eine Singleton-Klasse zu schreiben
  - → Schreibe eine getter-Methode, die eine Connection zurückgeben soll.
  - → Überprüfe, bevor du eine Connection aufbaust, ob die Connection null oder closed ist
    - → ConnectionState.Closed
  - → Verbinde dich mit der Library-Datenbank
  - → Nutze dafür die Open()-Methode
  - → Nutze Exceptions
  - → Gebt mittels MessageBox.Show(„Connected Successfully“) aus, ob die Verbindung erfolgreich war. Wenn nicht, passt die MessageBox-Nachricht an.

"Server=localhost;Database=library;User ID=root;Password=;"

Lösung

```
public static class DBConnector
{

    private static MySqlConnection _conn;
    public static MySqlConnection Connection
    {
        get
        {
            if (_conn == null || _conn.State == ConnectionState.Closed)
            {
                string connectionString = "Server=localhost;Database=memorygame;User ID=root;Password=;";
                _conn = new MySqlConnection(connectionString);
            }

                try
                {
                    _conn.Open();

                }
                catch(MySqlException ex)
                {
                    MessageBox.Show(ex.ToString());
                }
            return _conn;
        }
    }
}
```

MySqlCommand

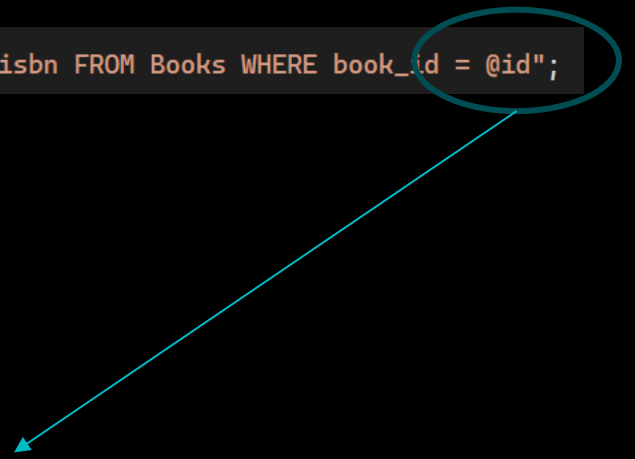
- → In Java nutzt man PreparedStatement
- → In C# mit MySql.Data nutzt man MySqlCommand
- → Wird genutzt, um Anfragen für die Datenbank vorzubereiten
- → Für alle Anfragen brauchbar
- → In den Konstruktor werden **query** und **connection** übergeben:

```
MySqlCommand cmd = new MySqlCommand(sql, conn);
```

MySqlCommand

Die Abfragen müssen ebenfalls angepasst werden:

```
string sql = "SELECT book_id, title, author_id, publication_year, genre, isbn FROM Books WHERE book_id = @id";
```



Parametername

MySqlCommand

Zum Setzen der Parameter muss man **cmd.Parameters.AddWithValue(param, value);** aufrufen.

```
string sql = "SELECT book_id, title, author_id, publication_year, genre, isbn FROM Books WHERE book_id = @id";

try
{
    MySqlCommand cmd = new MySqlCommand(sql, _connection);
    cmd.Parameters.AddWithValue("@id", id);
```

Parametername

Eigentlicher Wert

MySqlCommand

- → Wichtige Methoden:
  - → ExecuteNonQuery() – für alle Abfragen außer SELECT
    - → Gibt die Anzahl an modifizierten Zeilen zurück

```
public void AddBook(Book book)
{
    string sql = "INSERT INTO Books(title, author_id, publication_year, genre, isbn)" +
        " VALUES (@title, @author_id, @publication_year, @genre, @isbn);";

    try
    {
        MySqlCommand cmd = new MySqlCommand(sql, _connection);
        cmd.Parameters.AddWithValue("@title", book.Title);
        cmd.Parameters.AddWithValue("@author_id", book.AuthorId);
        cmd.Parameters.AddWithValue("@publication_year", book.PublicationYear);
        cmd.Parameters.AddWithValue("@genre", book.Genre);
        cmd.Parameters.AddWithValue("@isbn", book.ISBN);
        cmd.ExecuteNonQuery();
    }
    catch (Exception ex)
    {
        Console.WriteLine($"Error in AddBook: {ex.Message}");
    }

}
```

MySqlCommand

- → Wichtige Methoden:
  - → ExecuteReader() – für SELECT-Abfragen
    - → Gibt einen **MySqlDataReader** zurück
  - → MySqlDataReader hat eine **Read()**-Methode
    - → Muss in ein using gepackt werden, da pro Verbindung nur ein Reader offen sein darf
    - → Wird durch Nutzen von Using nach Gebrauch freigegeben

```
public List<Book> GetAllBooks()
{
    List<Book> books = new List<Book>();

    string sql = "SELECT book_id, title, author_id, publication_year, genre, isbn FROM Books";

    try
    {
        MySqlCommand cmd = new MySqlCommand(sql, _connection);
        using MySqlDataReader reader = cmd.ExecuteReader();

        while (reader.Read())
        {
            books.Add(new Book
            {
                BookId = reader.GetInt32("book_id"),
                Title = reader.GetString("title"),
                AuthorId = reader.GetInt32("author_id"),
                PublicationYear = reader.GetInt32("publication_year"),
                Genre = reader.GetString("genre"),
                ISBN = reader.GetString("isbn")
            });
        }
    }
    catch (Exception ex)
    {
        Console.WriteLine($"Error in GetAllBooks: {ex.Message}");
    }
    return books;
}
```

MySqlCommand

- → Wichtige Methoden:
  - → ExecuteScalar() – speziell für Abfragen mit Aggregatfunktionen
    - → Gibt ein object zurück, welches das Resultat der Aggregatfunktion hält

```
public void GetAveragePublicationYear()
{
    string sql = "SELECT AVG(publication_year) FROM Books";

    try
    {
        MySqlCommand cmd = new MySqlCommand(sql, _connection);
        Console.WriteLine(cmd.ExecuteScalar());
    }
    catch (Exception ex)
    {
        Console.WriteLine($"Error in GetAveragePublicationyear: {ex.Message}");
    }

}
```

Aufgabe

- → Erzeuge eine Modelklasse für die Book
- → Erzeuge eine DAO-Klasse für Book
  - → Schreibe eine Methode, die ein Buch anhand der Id holt
  - → Füge eine Methode hinzu, um ein Buch in die Datenbank zu speichern
  - → Verknüpfe das gestern erzeugte Fenster, sodass beim Klicken des Buttons ein Buch in die Datenbank gespeichert wird
- → Orientiert euch an der Klasse von Java