ORM and Dapper

Video 1: What is ORM and Dapper
Video 2: How to Install and Use Dapper + Understanding Parameterized Statements
Video 3: Demo

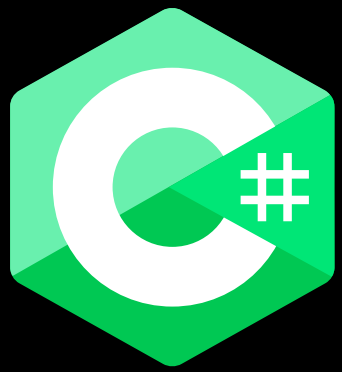
TRUE
CODERS

ORM and Dapper

Video 1: What is ORM and Dapper

TRUE
CODERS

We have learned



We have learned

Now we will combine them!

+ MySQL =

Some programs have databases

Websites that have databases?

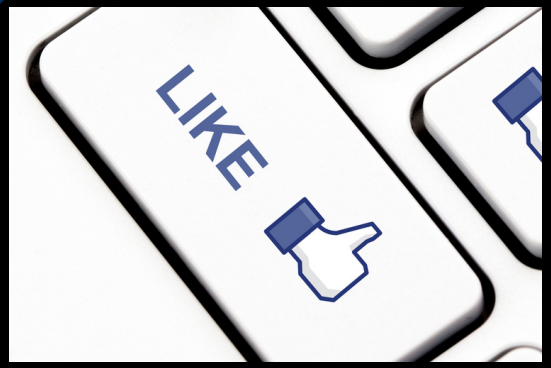
E-Commerce

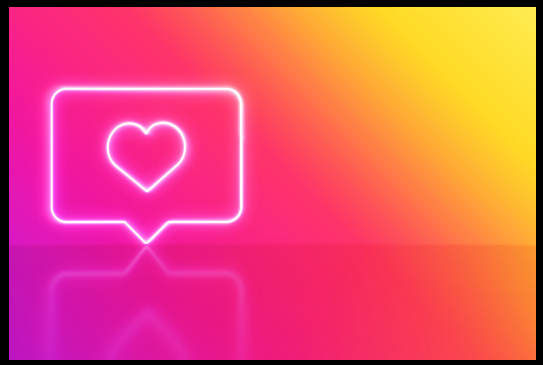
Banking and Financial Systems

CRM

Customer Relationship Manager System

Social Media





Combine C# and MySQL using an:

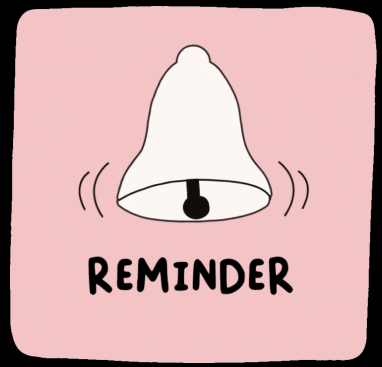
ORM

Interact with data from a database from a C# application

ORM

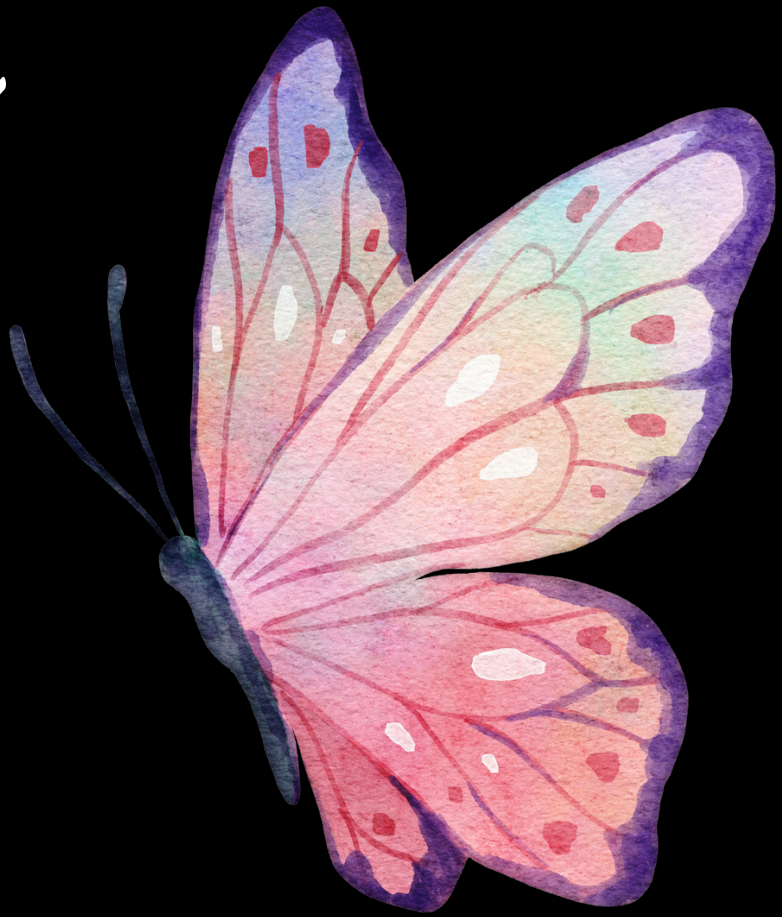
ORM

Object Relational Mapper



C# is an object oriented language

Transformation

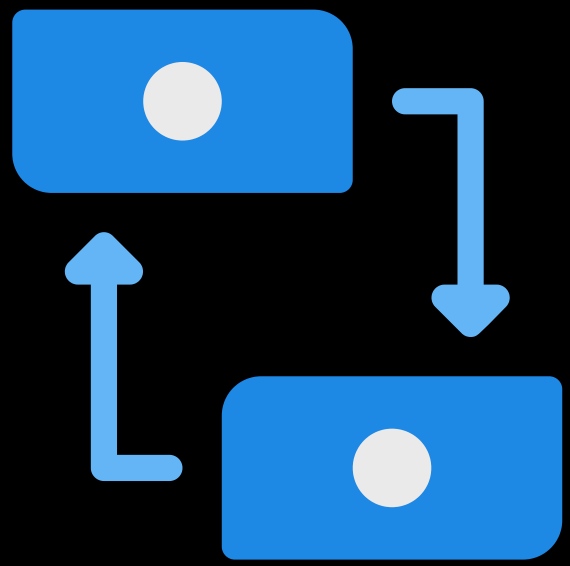


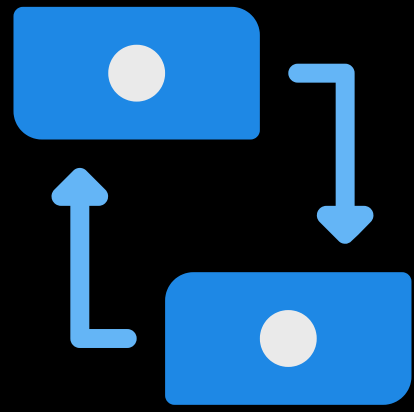
Data

Result Grid | Filter Rows: | Edit: | Export/Import:

ProductID	Name	Price	CategoryID	OnSale	StockLevel
1	Dell XPS 13	1400.00	1	0	1475
2	Lenovo Yoga	1600.00	1	0	245
4	Macbook	2200.00	1	0	903
8	Samsung Washer	450.00	2	1	880
16	Bose Wireless Headphones	450.00	4	0	192
17	Samsung Soundbar	300.00	5	1	1020
19	Samsung TV	600.00	5	0	222
23	HP Wireless Printer	400.00	6	0	751
34	Avril Lavigne: Let Go	11.99	7	0	443
37	Nelly Furtado: Loose	13.99	7	0	137
38	Evanescence: Fallen	11.99	7	0	658
41	Lady Gaga: The Fame	13.99	7	0	1376
46	Coldplay: A Rush of Blood t...	10.99	7	0	211
47	Jack Johnson: In Between ...	12.99	7	0	533
53	Eminem: Curtain Call: The ...	11.99	7	0	533
58	Celine Dion: A New Day Ha...	14.99	7	0	964
59	Dido: Life for Rent	11.99	7	0	323
61	50 Cent: Get Rich or Die Tr...	14.99	7	0	1422
62	Jack Johnson: Sleep Throu...	14.99	7	0	441
64	Madonna: Music	13.99	7	0	638
68	Imagine Dragons: Night Visi...	13.99	7	0	1216
71	Madonna: Confessions on ...	10.99	7	0	684
73	Nickelback: Silver Side Up	12.99	7	0	1070
74	Justin Timberlake: Justified	10.99	7	0	400
76	Justin Timberlake: FutureS...	10.99	7	0	1489
86	Various Artists: High School...	12.99	7	0	1223
89	Eminem: Recovery	12.99	7	0	964
92	Various Artists: Frozen	11.99	7	0	1050
94	Beyonce: Lemonade	11.99	7	0	859
97	Eagles: Hotel California	14.99	7	0	1046
101	Guns N' Roses: Appetite fo...	10.99	7	0	1152
103	Elton John: Greatest Hits	12.99	7	0	1121

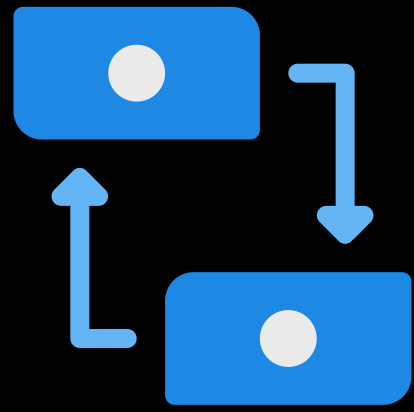
Convert Data to Objects





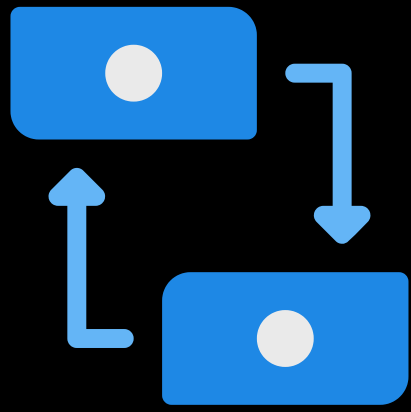
Result Grid | Filter Rows: | Edit: | Export/Import:

ProductID	Name	Price	CategoryID	OnSale	StockLevel
1	Dell XPS 13	1400.00	1	0	1475
2	Lenovo Yoga	1800.00	1	0	245
4	Macbook	2200.00	1	0	903
8	Samsung Washer	450.00	2	1	880
16	Bose Wireless Headphones	450.00	4	0	192
17	Samsung Soundbar	300.00	5	1	1020
19	Samsung TV	600.00	5	0	222
23	HP Wireless Printer	400.00	6	0	751
34	Avril Lavigne: Let Go	11.99	7	0	443
37	Nelly Furtado: Loose	13.99	7	0	137
38	Evanescence: Fallen	11.99	7	0	658
41	Lady Gaga: The Fame	13.99	7	0	1376
46	Coldplay: A Rush of Blood t...	10.99	7	0	211
47	Jack Johnson: In Between ...	12.99	7	0	533
53	Eminem: Curtain Call: The ...	11.99	7	0	533
58	Celine Dion: A New Day Ha...	14.99	7	0	964
59	Dido: Life for Rent	11.99	7	0	323
61	50 Cent: Get Rich or Die Tr...	14.99	7	0	1422
62	Jack Johnson: Sleep Throu...	14.99	7	0	441
64	Madonna: Music	13.99	7	0	638
68	Imagine Dragons: Night Visi...	13.99	7	0	1216
71	Madonna: Confessions on ...	10.99	7	0	684
73	Nickelback: Silver Side Up	12.99	7	0	1070
74	Justin Timberlake: Justified	10.99	7	0	400
76	Justin Timberlake: FutureS...	10.99	7	0	1489
86	Various Artists: High School...	12.99	7	0	1223
89	Eminem: Recovery	12.99	7	0	964
92	Various Artists: Frozen	11.99	7	0	1050
94	Beyonce: Lemonade	11.99	7	0	859
97	Eagles: Hotel California	14.99	7	0	1046
101	Guns N' Roses: Appetite fo...	10.99	7	0	1152
103	Elton John: Greatest Hits	12.99	7	0	1121



Result Grid | Filter Rows: | Edit: | Export/Import:

ProductID	Name	Price	CategoryID	OnSale	StockLevel
1	Dell XPS 13	1400.00	1	0	1475
2	Lenovo Yoga	1600.00	1	0	245
4	Macbook	2200.00	1	0	903
8	Samsung Washer	450.00	2	1	880
16	Bose Wireless Headphones	450.00	4	0	192
17	Samsung Soundbar	300.00	5	1	1020
19	Samsung TV	600.00	5	0	222
23	HP Wireless Printer	400.00	6	0	751
34	Avril Lavigne: Let Go	11.99	7	0	443
37	Nelly Furtado: Loose	13.99	7	0	137
38	Evanescence: Fallen	11.99	7	0	658
41	Lady Gaga: The Fame	13.99	7	0	1376
46	Coldplay: A Rush of Blood t...	10.99	7	0	211
47	Jack Johnson: In Between ...	12.99	7	0	533
53	Eminem: Curtain Call: The ...	11.99	7	0	533
58	Celine Dion: A New Day Ha...	14.99	7	0	964
59	Dido: Life for Rent	11.99	7	0	323
61	50 Cent: Get Rich or Die Tr...	14.99	7	0	1422
62	Jack Johnson: Sleep Throu...	14.99	7	0	441
64	Madonna: Music	13.99	7	0	638
68	Imagine Dragons: Night Visi...	13.99	7	0	1216
71	Madonna: Confessions on ...	10.99	7	0	684
73	Nickelback: Silver Side Up	12.99	7	0	1070
74	Justin Timberlake: Justified	10.99	7	0	400
76	Justin Timberlake: FutureS...	10.99	7	0	1489
86	Various Artists: High School...	12.99	7	0	1223
89	Eminem: Recovery	12.99	7	0	964
92	Various Artists: Frozen	11.99	7	0	1050
94	Beyonce: Lemonade	11.99	7	0	859
97	Eagles: Hotel California	14.99	7	0	1046
101	Guns N' Roses: Appetite fo...	10.99	7	0	1152
103	Elton John: Greatest Hits	12.99	7	0	1121

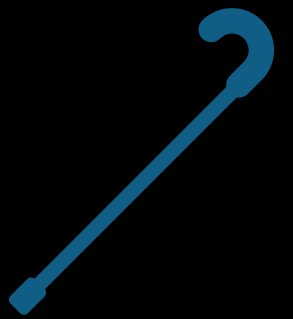


Result Grid | Filter Rows: | Edit: | Export/Import:

ProductID	Name	Price	CategoryID	OnSale	StockLevel
1	Dell XPS 13	1400.00	1	0	1475
4	Macbook	2200.00	1	0	903
8	Samsung Washer	450.00	2	1	880
16	Bose Wireless Headphones	450.00	4	0	192
17	Samsung Soundbar	300.00	5	1	1020
19	Samsung TV	600.00	5	0	222
23	HP Wireless Printer	400.00	6	0	751
34	Avril Lavigne: Let Go	11.99	7	0	443
37	Nelly Furtado: Loose	13.99	7	0	137
38	Evanescence: Fallen	11.99	7	0	658
41	Lady Gaga: The Fame	13.99	7	0	1376
46	Coldplay: A Rush of Blood t...	10.99	7	0	211
47	Jack Johnson: In Between ...	12.99	7	0	533
53	Eminem: Curtain Call: The ...	11.99	7	0	533
58	Celine Dion: A New Day Ha...	14.99	7	0	964
59	Dido: Life for Rent	11.99	7	0	323
61	50 Cent: Get Rich or Die Tr...	14.99	7	0	1422
62	Jack Johnson: Sleep Throu...	14.99	7	0	441
64	Madonna: Music	13.99	7	0	638
68	Imagine Dragons: Night Visi...	13.99	7	0	1216
71	Madonna: Confessions on ...	10.99	7	0	684
73	Nickelback: Silver Side Up	12.99	7	0	1070
74	Justin Timberlake: Justified	10.99	7	0	400
76	Justin Timberlake: FutureS...	10.99	7	0	1489
86	Various Artists: High School...	12.99	7	0	1223
89	Eminem: Recovery	12.99	7	0	964
92	Various Artists: Frozen	11.99	7	0	1050
94	Beyonce: Lemonade	11.99	7	0	859
97	Eagles: Hotel California	14.99	7	0	1046
101	Guns N' Roses: Appetite fo...	10.99	7	0	1152
103	Elton John: Greatest Hits	12.99	7	0	1121

Dapper

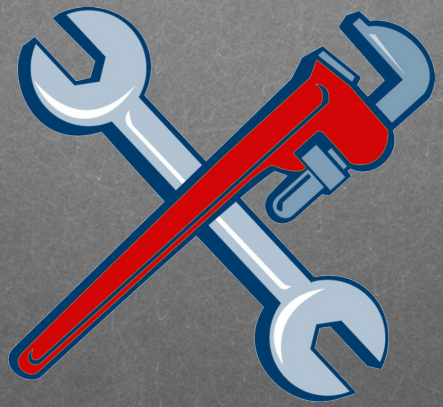
is an ORM



What is an ORM?

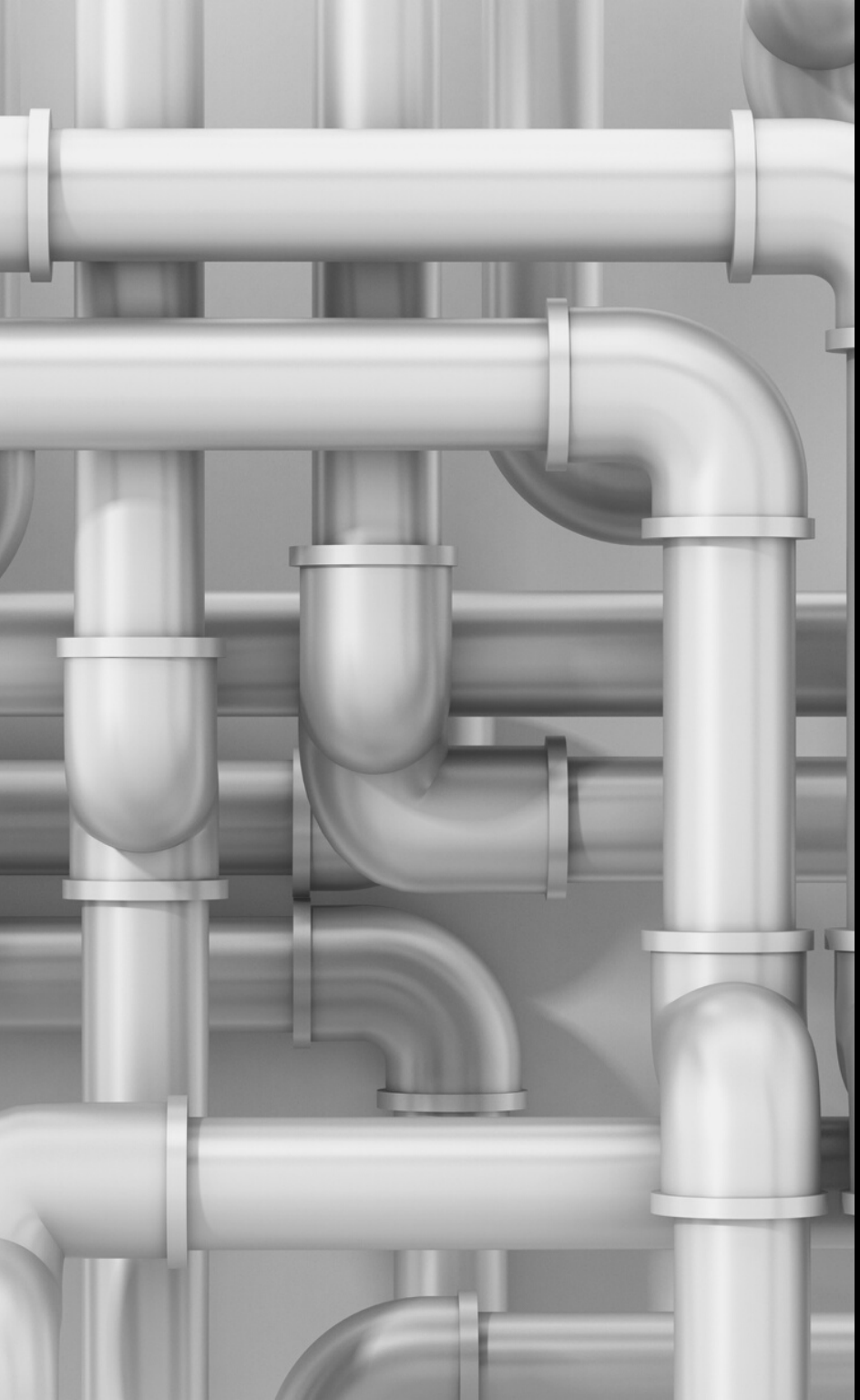
ORM

Object Relational Mapper



ORM

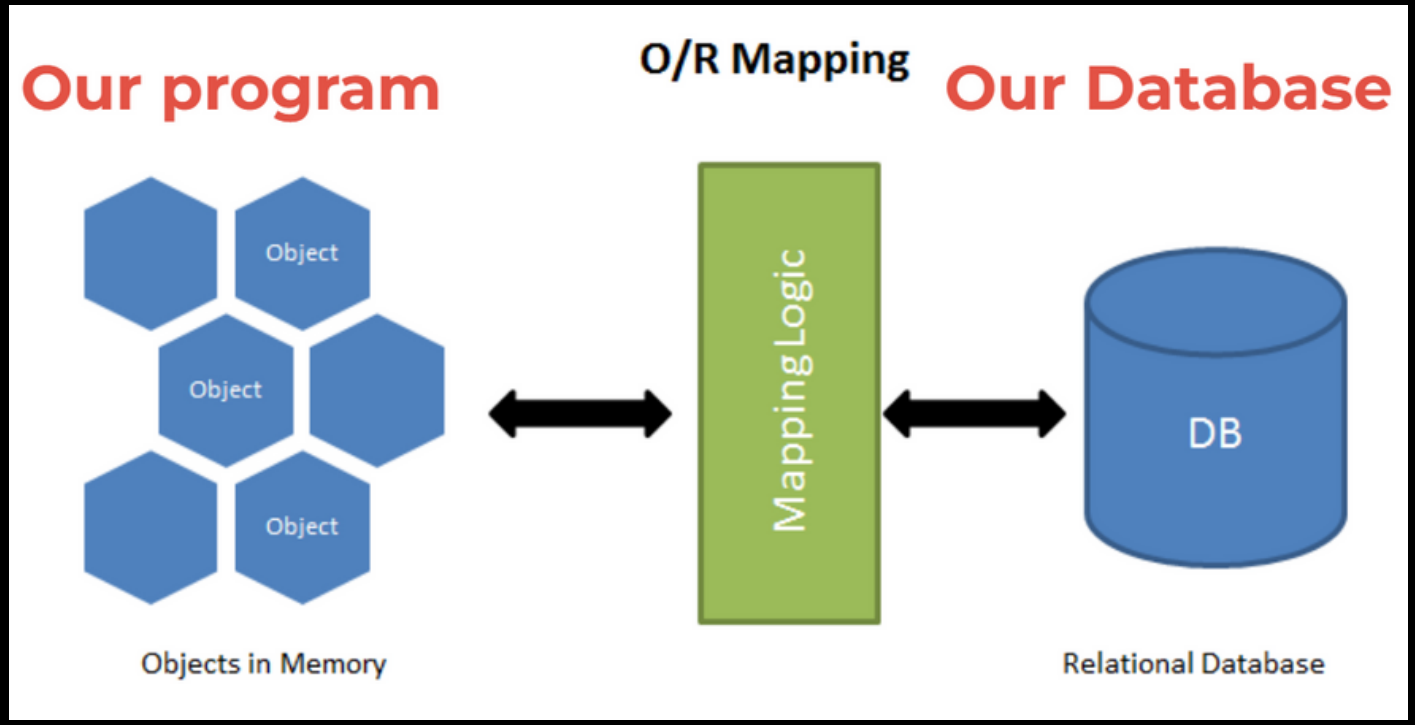
"Plumbers" of the programming world.



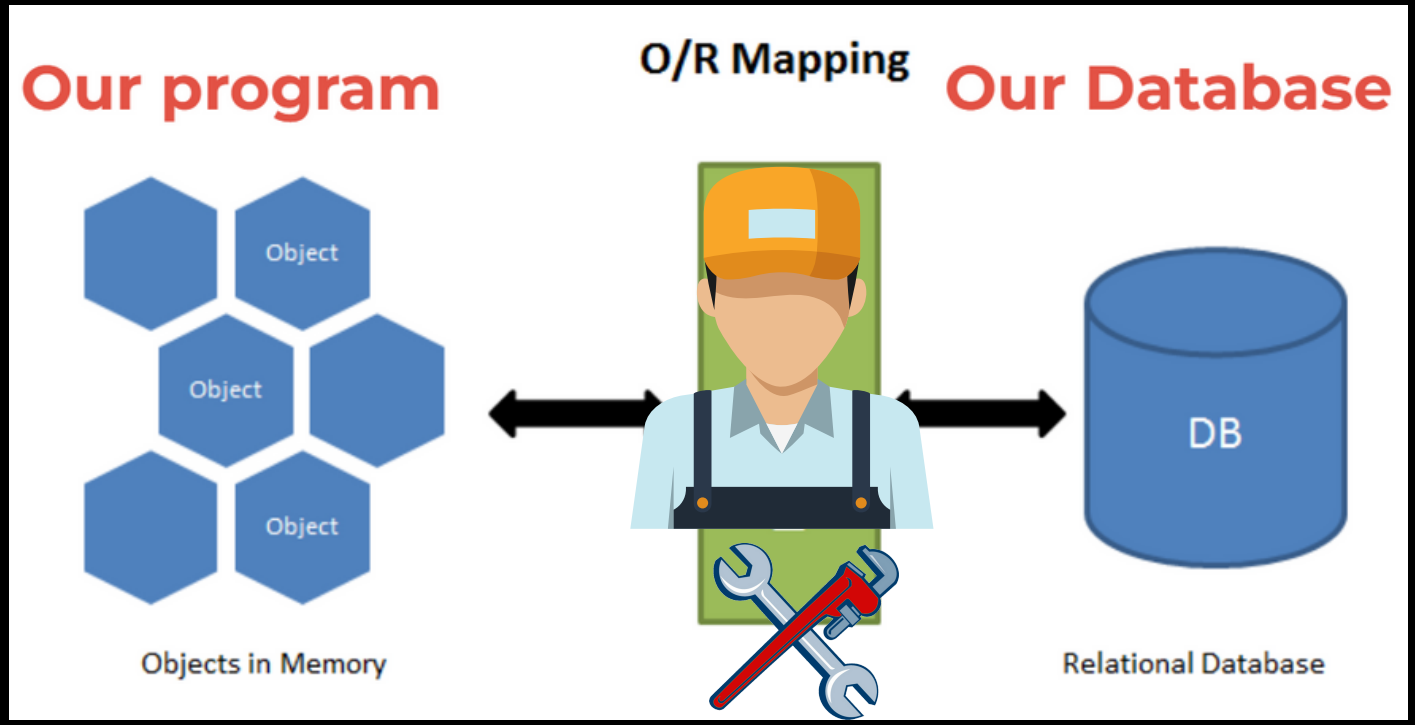
ORM

Get data out of and back into databases.

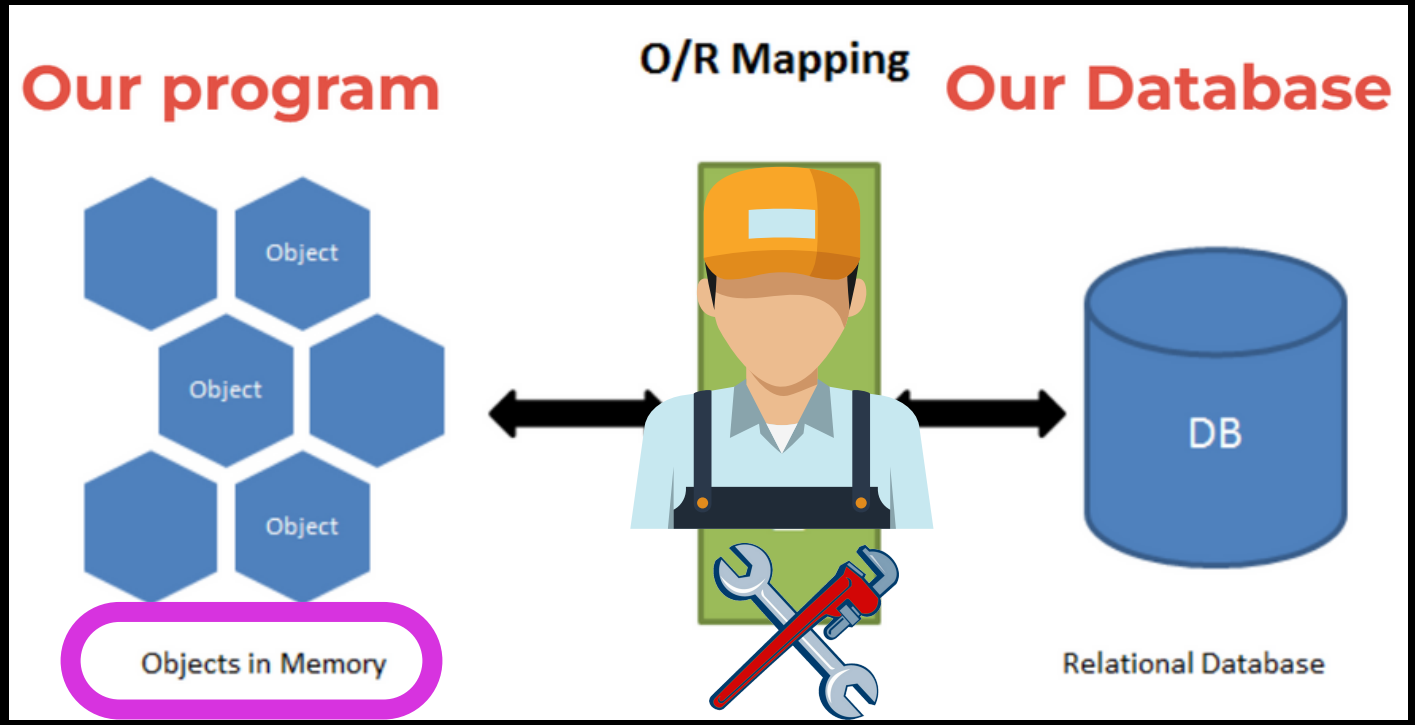
Middleman

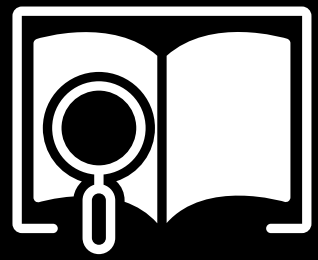

Our program
O/R Mapping
Our Database
Object
Object
Object
Mapping Logic
DB
Objects in Memory
Relational Database

Map Data


Our program
O/R Mapping
Our Database
Object
Object
Object
DB
Objects in Memory
Relational Database

Turns data into objects





ORM - Object Relational Mapper

An Object Relational Mapper is a software abstractor that is used to access a relational database from an object-oriented language.

Picking out a Plumber

Hibernate

Django

Dapper

Entity

Doctrine

Picking out a Plumber

Why Dapper

Dapper has no database specific implementation details.

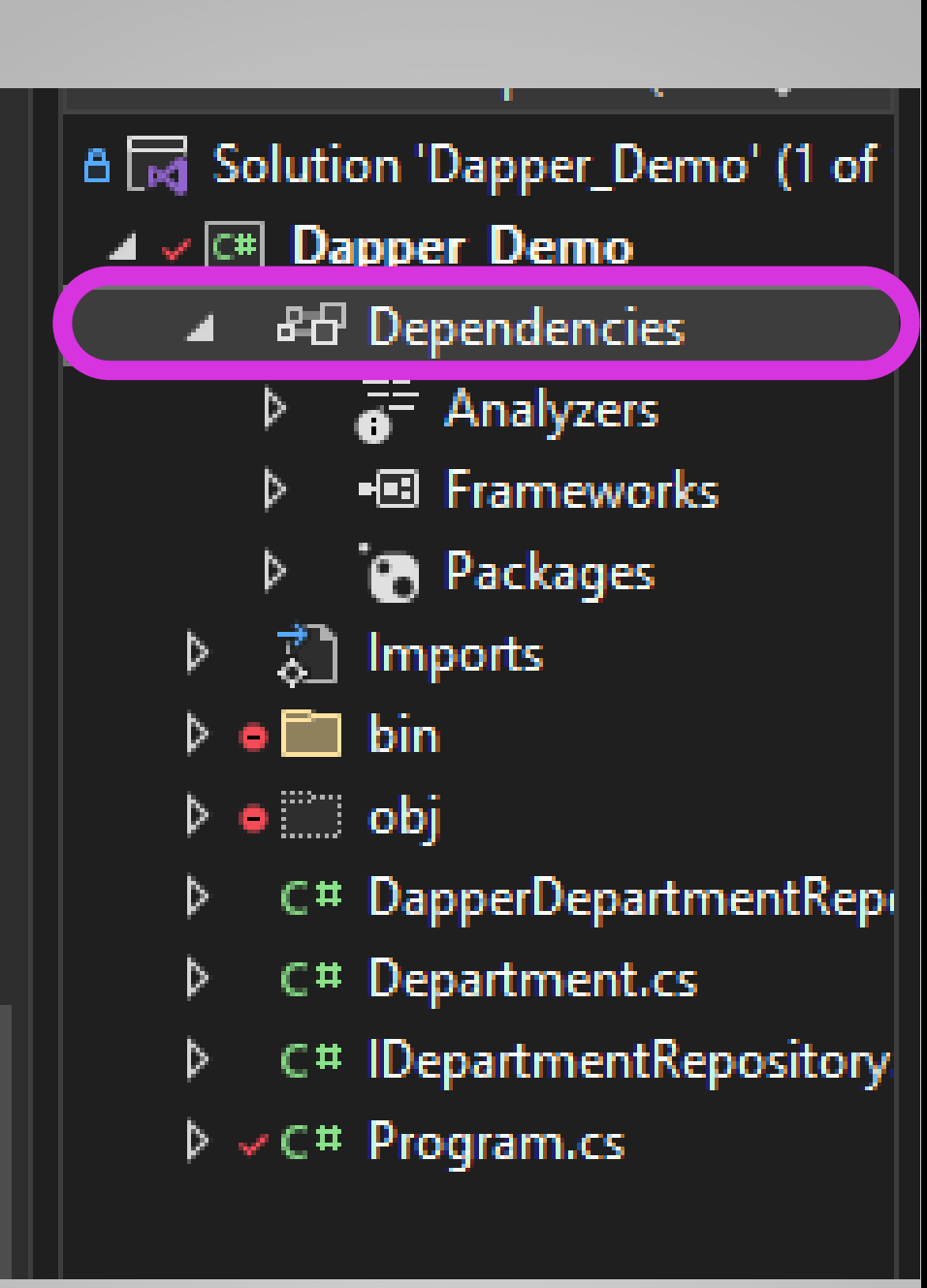
- It works across SQLite, Oracle, MySQL, PostgreSQL, and SQL Server, to name a few.

ORM and Dapper

Video 2: How to use Dapper and Parameterized Statements

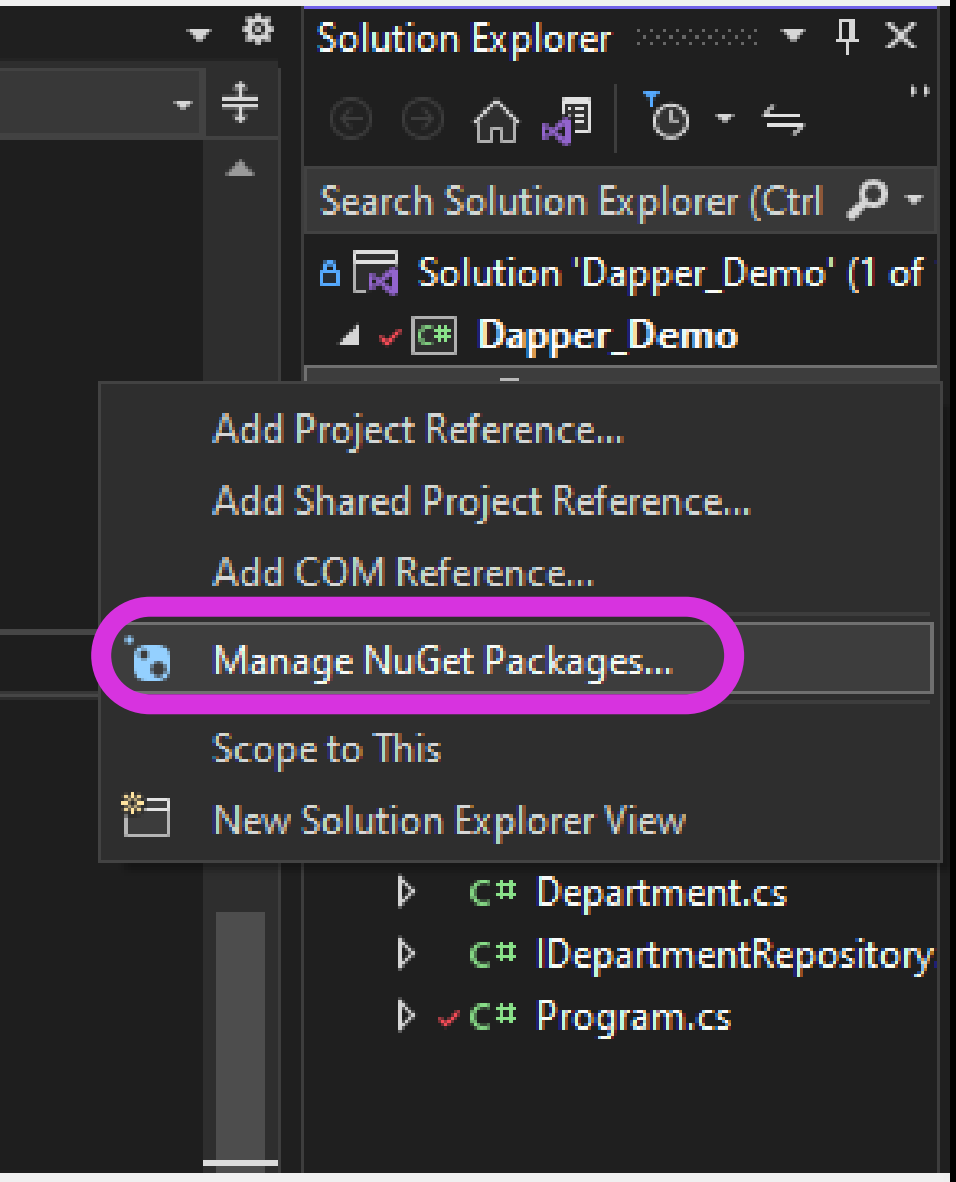
Dapper is a Library

- It's NuGet library ORM that you can add to your project that will extend your IDbConnection interface.
- IDbConnection interface is part of the .NET Framework used for interacting with databases



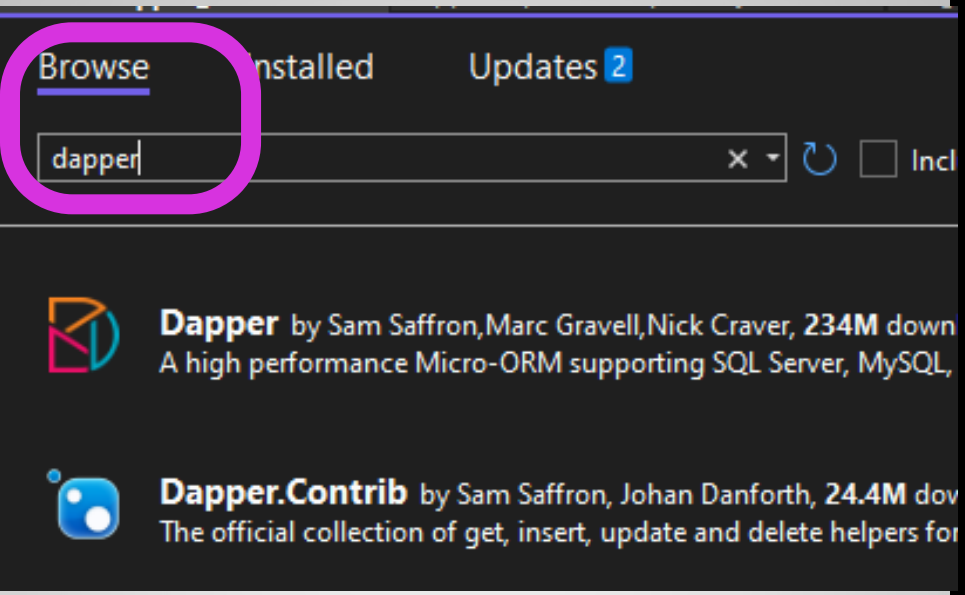
Installing Dapper

Right click on "Dependencies"



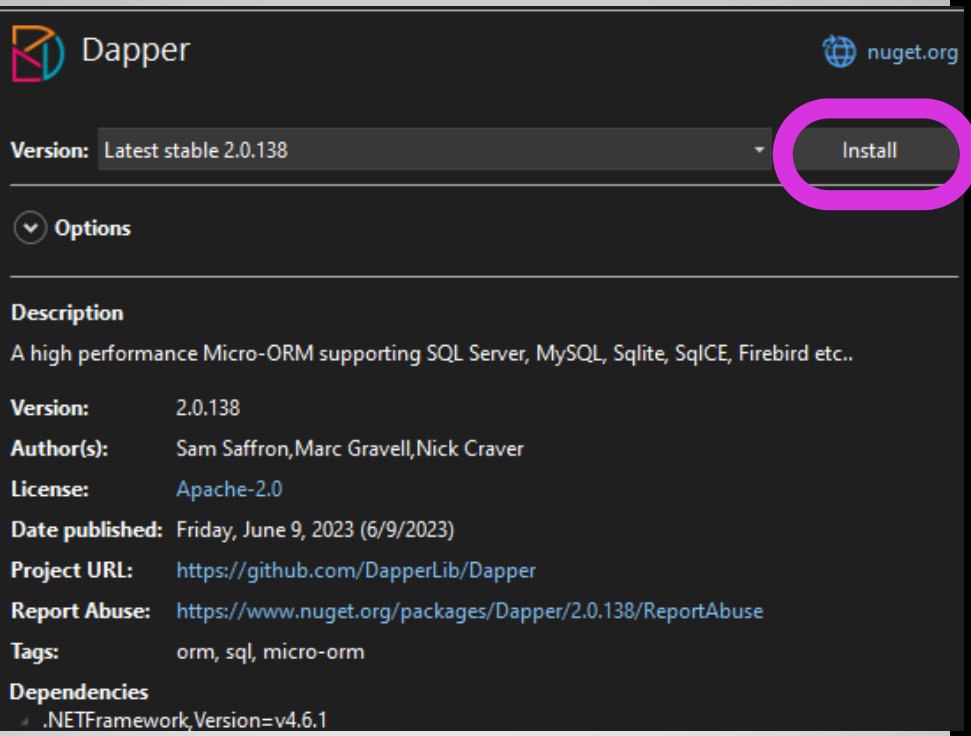
Installing Dapper

Click "Manage NuGet Packages"



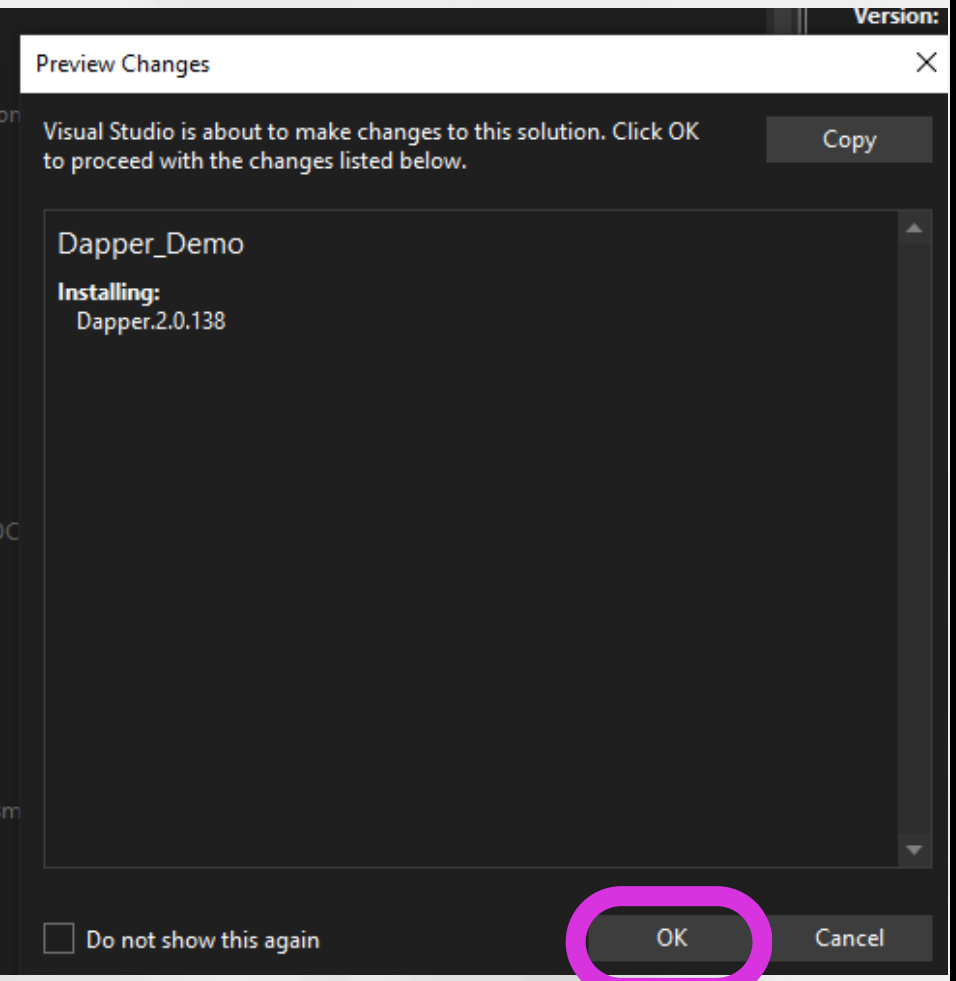
Installing Dapper

Search in Browse tab for "Dapper"



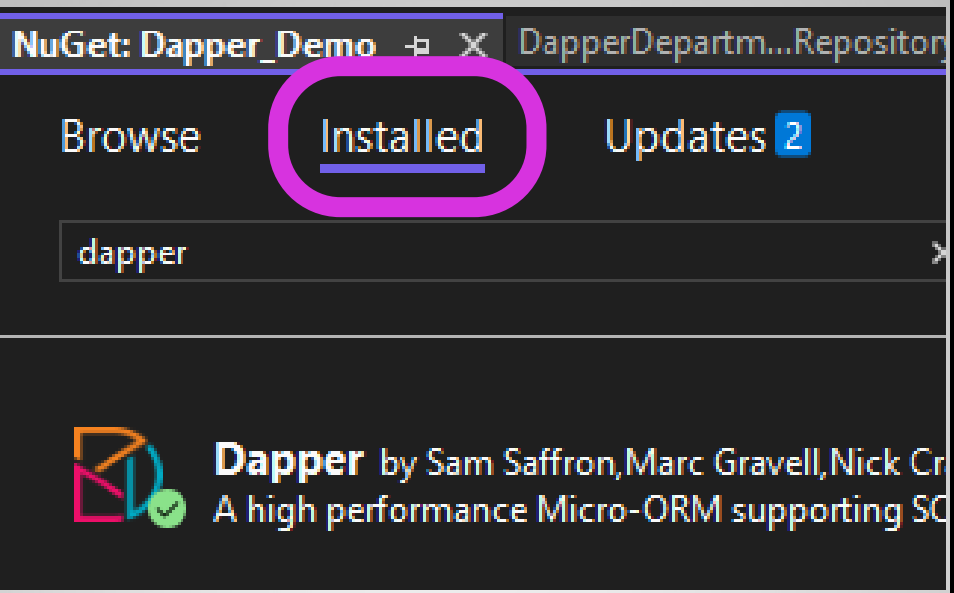
Installing Dapper

Click "Install"



Installing Dapper

Click "OK"



Installing Dapper

Success!

Dapper Methods

- After you install Dapper, you will have access to its library which includes these methods:
  - **Query()**
  - **Execute()**

Do we have to use Dapper?


NO NOT REALLY NO

Without Dapper

```
15
16      public IEnumerable<Department> GetAllDepartments()
17      {
18          using (var conn = new MySqlConnection(_connectionString))
19          {
20              conn.Open();
21
22              MySqlCommand cmd = conn.CreateCommand();
23              cmd.CommandText = "SELECT * FROM Departments;";
24
25              MySqlDataReader reader = cmd.ExecuteReader();
26
27              List<Department> allDepartments = new List<Department>();
28
29              while (reader.Read() == true)
30              {
31                  var currentDepartment = new Department();
32                  currentDepartment.DepartmentID = (int)reader["DepartmentID"];
33                  currentDepartment.Name = (string)reader["Name"];
34
35                  allDepartments.Add(currentDepartment);
36              }
37
38              return allDepartments;
39          }
40      }
```

With Dapper

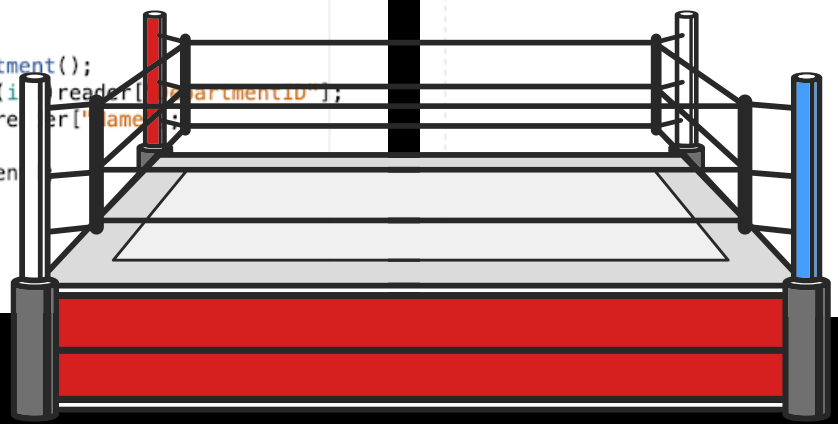
```
18
19          public IEnumerable<Department> GetAllDepartments()
20          {
21              return _connection.Query<Department>("SELECT * FROM Departments;");
22          }
23
24
25
26
27
28
29
30
31
32
33
34
35
36
```

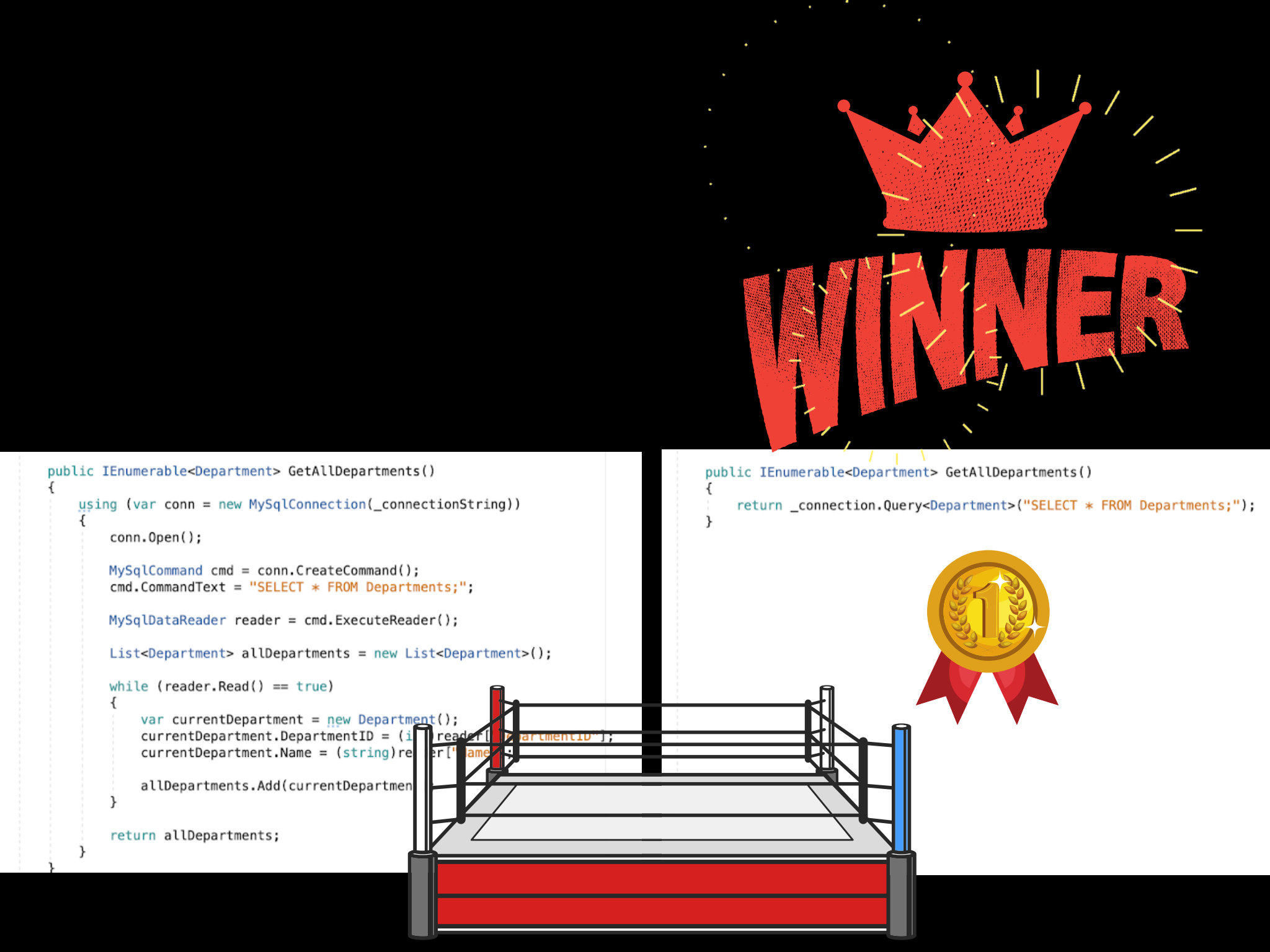
Comparing

Vs

```
public IEnumerable<Department> GetAllDepartments()
{
    using (var conn = new MySqlConnection(_connectionString))
    {
        conn.Open();

        MySqlCommand cmd = conn.CreateCommand();
        cmd.CommandText = "SELECT * FROM Departments;";

        MySqlDataReader reader = cmd.ExecuteReader();

        List<Department> allDepartments = new List<Department>();

        while (reader.Read() == true)
        {
            var currentDepartment = new Department();
            currentDepartment.DepartmentID = (i
            currentDepartment.Name = (string)re

            allDepartments.Add(currentDepartmen
        }

        return allDepartments;
    }
}
```

```
public IEnumerable<Department> GetAllDepartments()
{
    return _connection.Query<Department>("SELECT * FROM Departments;");
}
```




WINNER
public IEnumerable<Department> GetAllDepartments()
{
using (var conn = new MySqlConnection(_connectionString))
{
conn.Open();
MySqlCommand cmd = conn.CreateCommand();
cmd.CommandText = "SELECT * FROM Departments;";
MySqlDataReader reader = cmd.ExecuteReader();
List<Department> allDepartments = new List<Department>();
while (reader.Read() == true)
{
var currentDepartment = new Department();
currentDepartment.DepartmentID = (i
currentDepartment.Name = (string)re
allDepartments.Add(currentDepartmen
}
return allDepartments;
}
}
public IEnumerable<Department> GetAllDepartments()
{
return _connection.Query<Department>("SELECT * FROM Departments;");
}
1

What's inside the parenthesis?

```
public IEnumerable<Department> GetAllDepartments()
{
    return _connection.Query<Department>("SELECT * FROM Departments;");
}
```

What's inside the parenthesis?

```
public IEnumerable<Department> GetAllDepartments()
{
    return _connection.Query<Department>("SELECT * FROM Departments;");
}
```

Query Method

- The Dapper Query is designed for any database reads, like SELECT.

```
public IEnumerable<Department> GetAllDepartments()
{
    return _connectio.Query<Department>("SELECT * FROM Departments;");
}
```

Query Method

- Query returns an IEnumerable<T>

```
public IEnumerable<Department> GetAllDepartments()
{
    return _connection.Query<Department>("SELECT * FROM Departments;");
}
```

Query Method

Return Type

```
public IEnumerable<Department> GetAllDepartments()
{
    return _connection.Query<Department>("SELECT * FROM Departments;");
}
```

Execute Method

- The Dapper Execute is designed for any database writes, like INSERT, UPDATE, and DELETE.

```
2 references
public void InsertDepartment(string newDepartmentName)
{
    _connection.Execute("INSERT INTO DEPARTMENTS (Name)
     new { departmentName = newDepartmentName });
}
```

Execute Method

- The Dapper Execute is designed for any database writes, like INSERT, UPDATE, and DELETE.

```
2 references
public void InsertDepartment(string newDepartmentName)
{
    _connection.Execute("INSERT INTO DEPARTMENTS (Name)
     new { departmentName = newDepartmentName });
}
```

Execute Method

- The Dapper Execute is designed for any database writes, like INSERT, UPDATE, and DELETE.

Return Type

```
2 references
public void InsertDepartment(string newDepartmentName)
{
    _connection.Execute("INSERT INTO DEPARTMENTS (Name)
     new { departmentName = newDepartmentName });
}
```

Query Method

has no parameters

```
3 references
public IEnumerable<Department> GetAllDepartments()   //R in Crud for Read
{
    return _connection.Query<Department>("SELECT * FROM Departments;");
}
```

Execute Method

has a parameter

```
2 references
public void InsertDepartment(string newDepartmentName)  //C in Crud for Create
{
    _connection.Execute("INSERT INTO DEPARTMENTS (Name) VALUES (@departmentName);",
     new { departmentName = newDepartmentName });
}
```

With Parameterized Statement

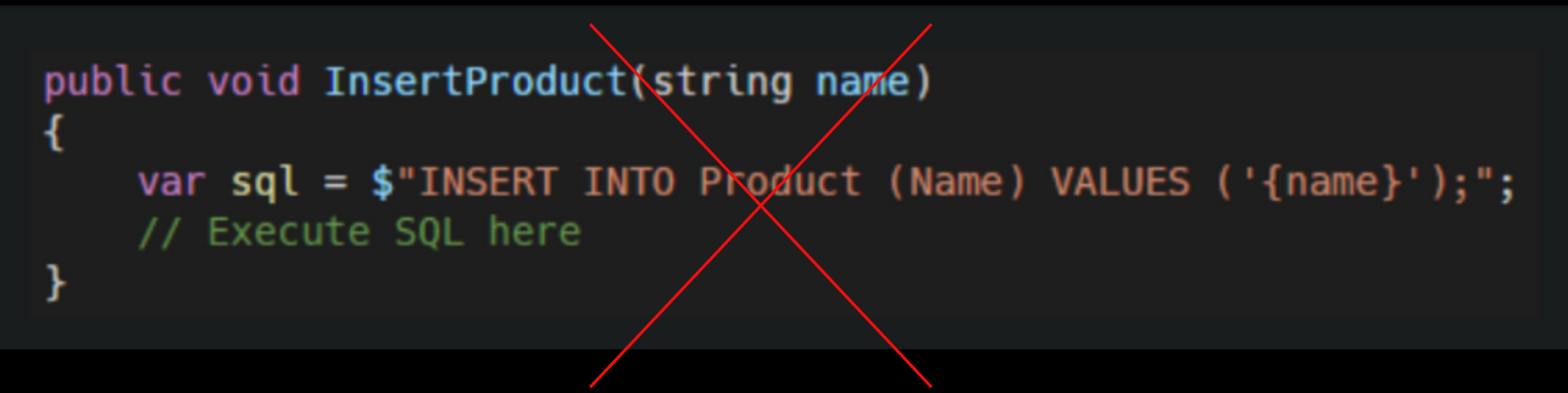
```
2 references
public void InsertDepartment(string newDepartmentName)  //C in Crud for Create
{
    _connection.Execute("INSERT INTO DEPARTMENTS (Name) VALUES (@departmentName);",
     new { departmentName = newDepartmentName });
}
```

Without Parameterized Statement

```
public void InsertProduct(string name)
{
    var sql = $"INSERT INTO Product (Name) VALUES ('{name}');";
    // Execute SQL here
}
```

Without Parameterized Statement

- Potential **SQL Injection**



```
public void InsertProduct(string name)
{
    var sql = $"INSERT INTO Product (Name) VALUES ('{name}');";
    // Execute SQL here
}
```

Without Parameterized Statement

- SQL Injection == attack on your code

```
public void InsertProduct(string name)
{
    var sql = $"INSERT INTO Product (Name) VALUES ('{name}');";
    // Execute SQL here
}
```

Without Parameterized Statement

- **Calling your method:**

```
InsertProduct("Test'); DROP TABLE Product; --");
```

- **Method definition:**

```
public void InsertProduct(string name)
{
    var sql = $"INSERT INTO Product (Name) VALUES ('{name}');";
    // Execute SQL here
}
```

Without Parameterized Statement

- **Calling your method:**

```
InsertProduct("Test'); DROP TABLE Product; --" ;
```

1

2

- **Method definition:**

```
public void InsertProduct(string name)
{
    var sql = $"INSERT INTO Product (Name) VALUES ('{name}');";
    // Execute SQL here
}
```

End Result

- **Malicious**

```
INSERT INTO Product (Name) VALUES ('Test'); DROP TABLE Product; --');
```

This is a valid SQL statement that will add a product and then delete the Product table.

Without Parameterized Statement

- **Calling your method:**

```
InsertProduct("O'Henry");
```

- **Method definition:**

```
public void InsertProduct(string name)
{
    var sql = $"INSERT INTO Product (Name) VALUES ('{name}');";
    // Execute SQL here
}
```

Without Parameterized Statement

- **Calling your method:**

```
InsertProduct("O'Henry");
```

- **Method definition:**

```
public void InsertProduct(string name)
{
    var sql = $"INSERT INTO Product (Name) VALUES ('{name}');";
    // Execute SQL here
}
```

End Result

- **Accidental** 

```
INSERT INTO Product (Name) VALUES ('O'Henry');
```

This is an invalid SQL statement that will fail because we have an open string delimiter, but no closed string delimiter. Not malicious code, but it will still fail.

Parameterized Statements

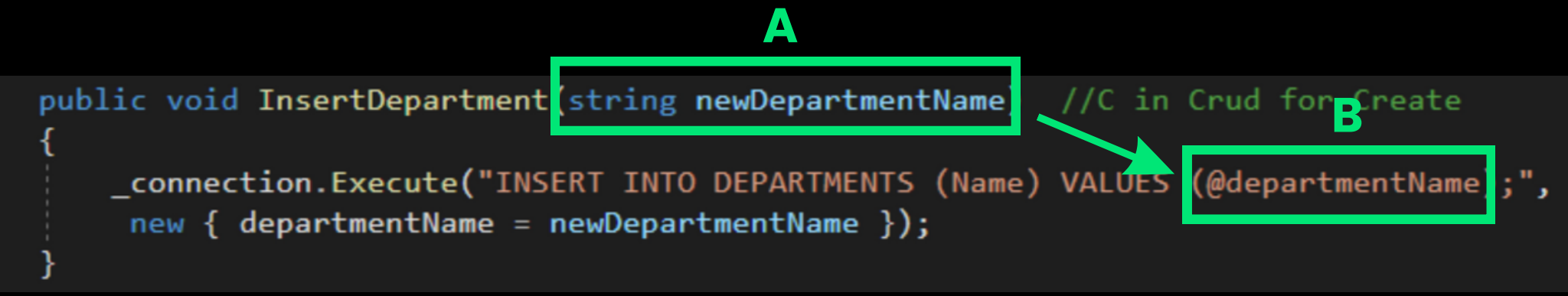
- Protect our code!

Parameterized Statements

- Safely pass values

Parameterized Statements

- We need to safely pass in our parameter and set the value of departmentName



```
public void InsertDepartment(string newDepartmentName)  //C in Crud for Create
{
    _connection.Execute("INSERT INTO DEPARTMENTS (Name) VALUES (@departmentName);",
    new { departmentName = newDepartmentName });
}
```

```
public void InsertDepartment(string newDepartmentName)  //C in Crud for Create
{
    _connection.Execute("INSERT INTO DEPARTMENTS (Name) VALUES (@departmentName);",
    new { departmentName = newDepartmentName });
}
```

Anonymous Type -> Temporary Object

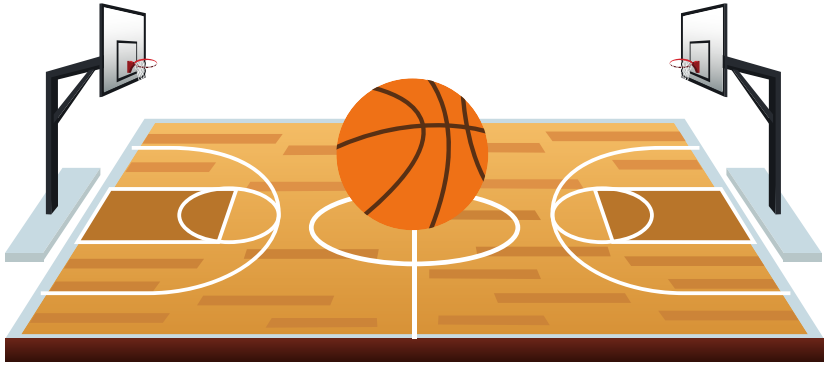
```
public void InsertDepartment(string newDepartmentName)  //C in Crud for Create
{
    _connection.Execute("INSERT INTO DEPARTMENTS (Name) VALUES (@departmentName);",
    new { departmentName = newDepartmentName });
}
```

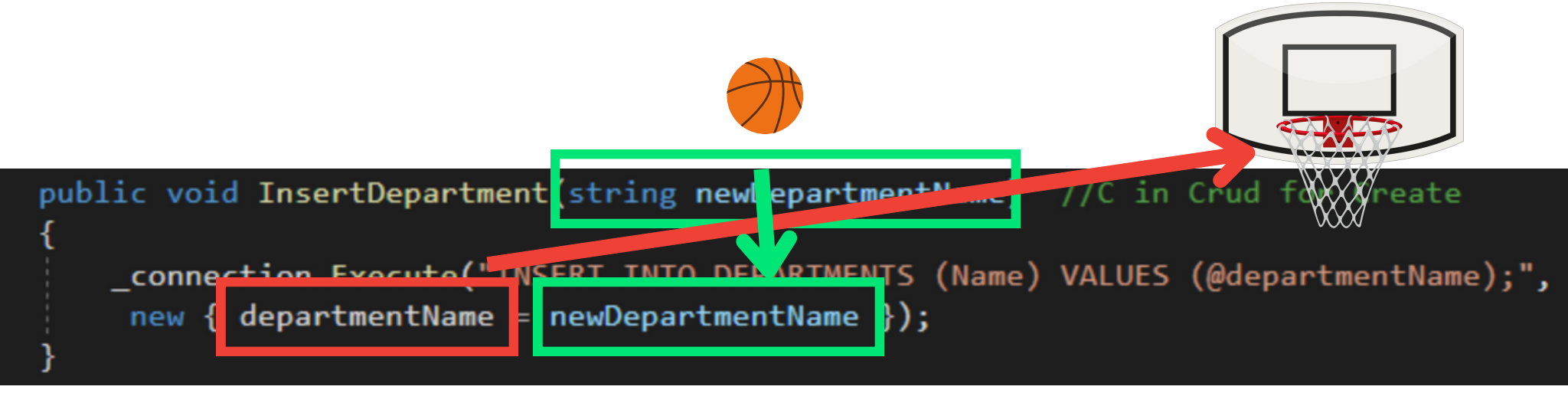
Anonymous Type -> This will allow us to safely pass in our parameter value

```
public void InsertDepartment(string newDepartmentName)  //C in Crud for Create
{
    _connection.Execute("INSERT INTO DEPARTMENTS (Name) VALUES (@departmentName);",
     new { departmentName = newDepartmentName });
}
```

Anonymous Type -> Setting up the relationship

GAME PLAY




public void InsertDepartment(string newDepartmentName) //C in Crud for Create
{
_connection.Execute("INSERT INTO DEPARTMENTS (Name) VALUES (@departmentName);",
new { departmentName = newDepartmentName });
}

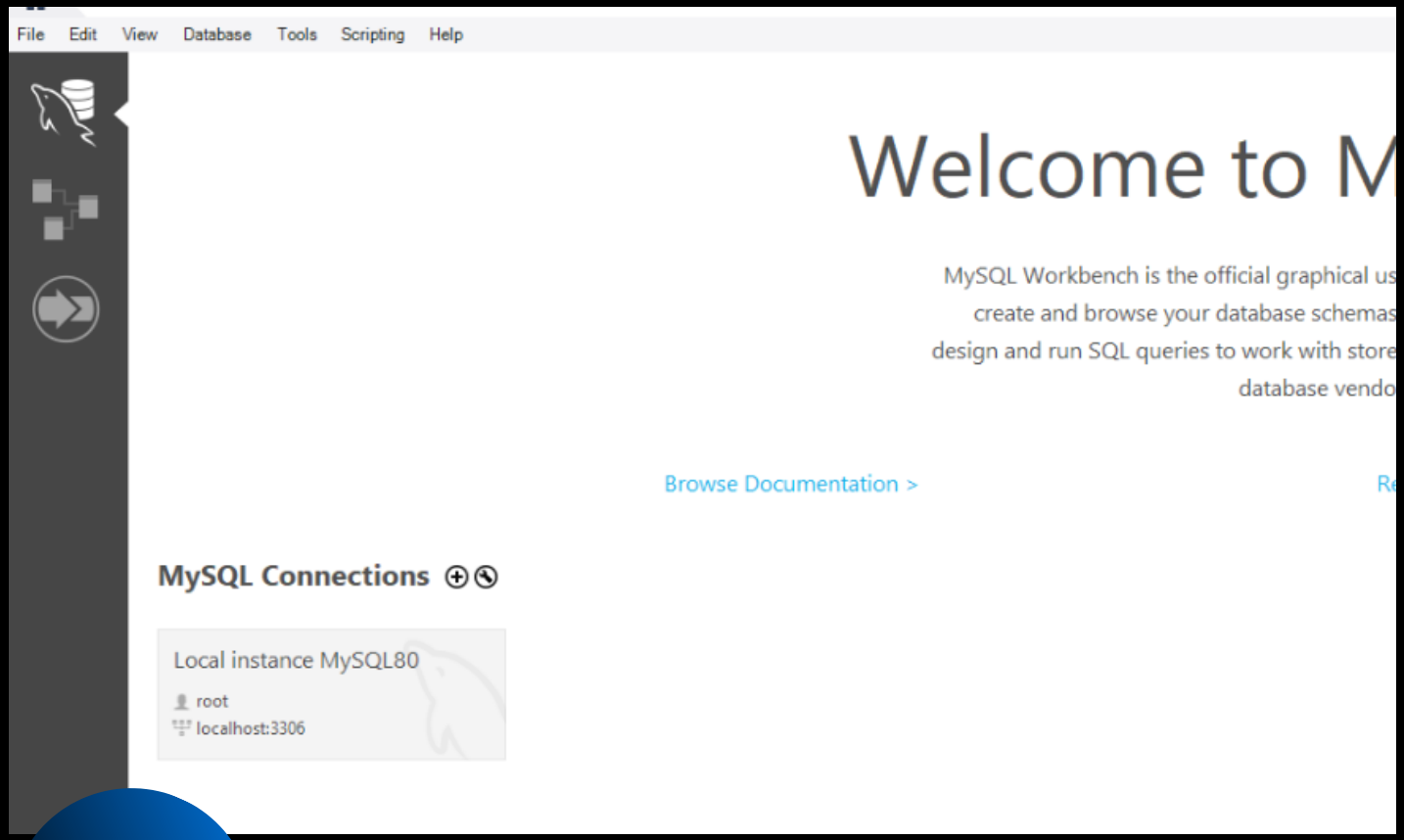
parameterized

statements

abstract your statement from your parameters

- sanitize your data

Prevents SQL Injection

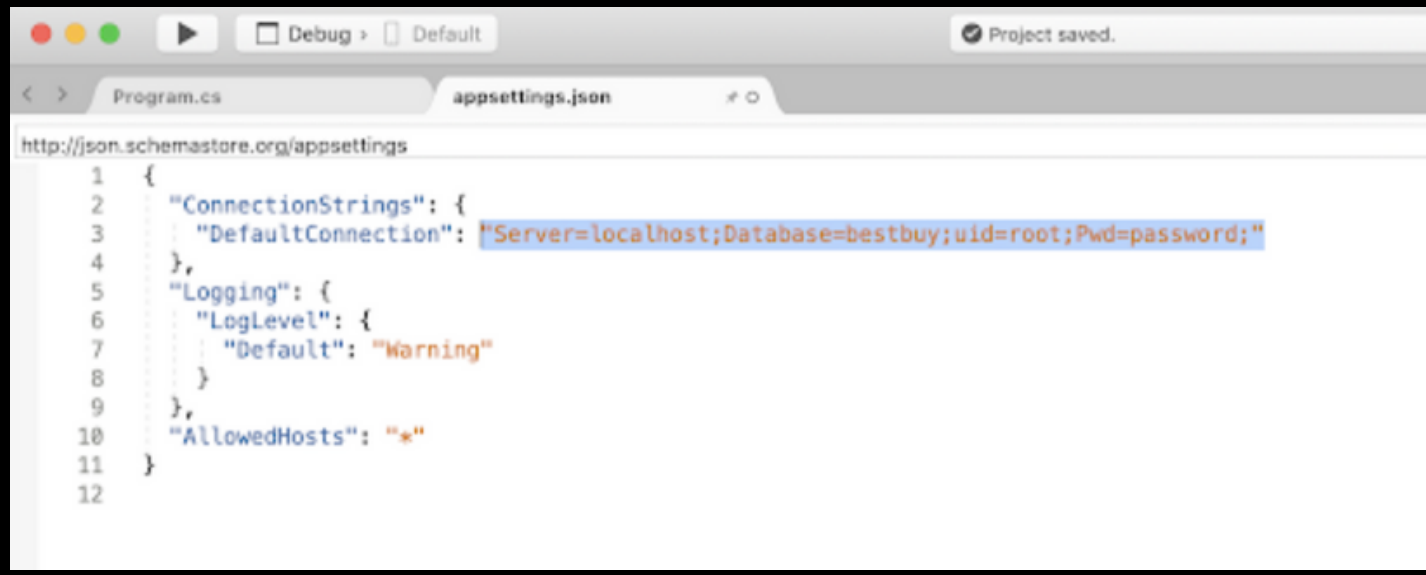


MySQL

Login with a Password

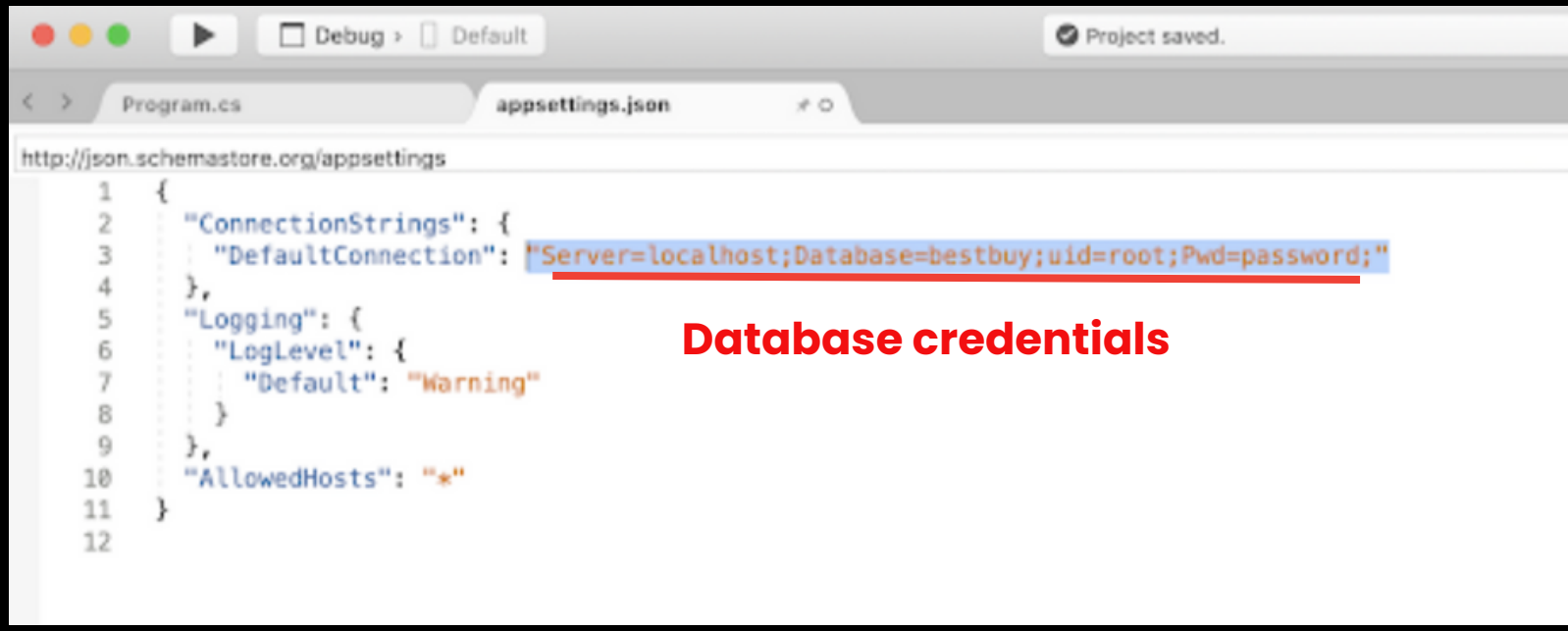
C# Application

Also needs a password to the database



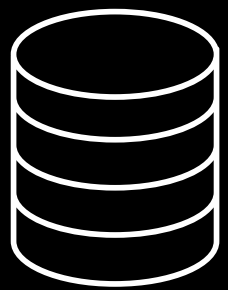
C# Application

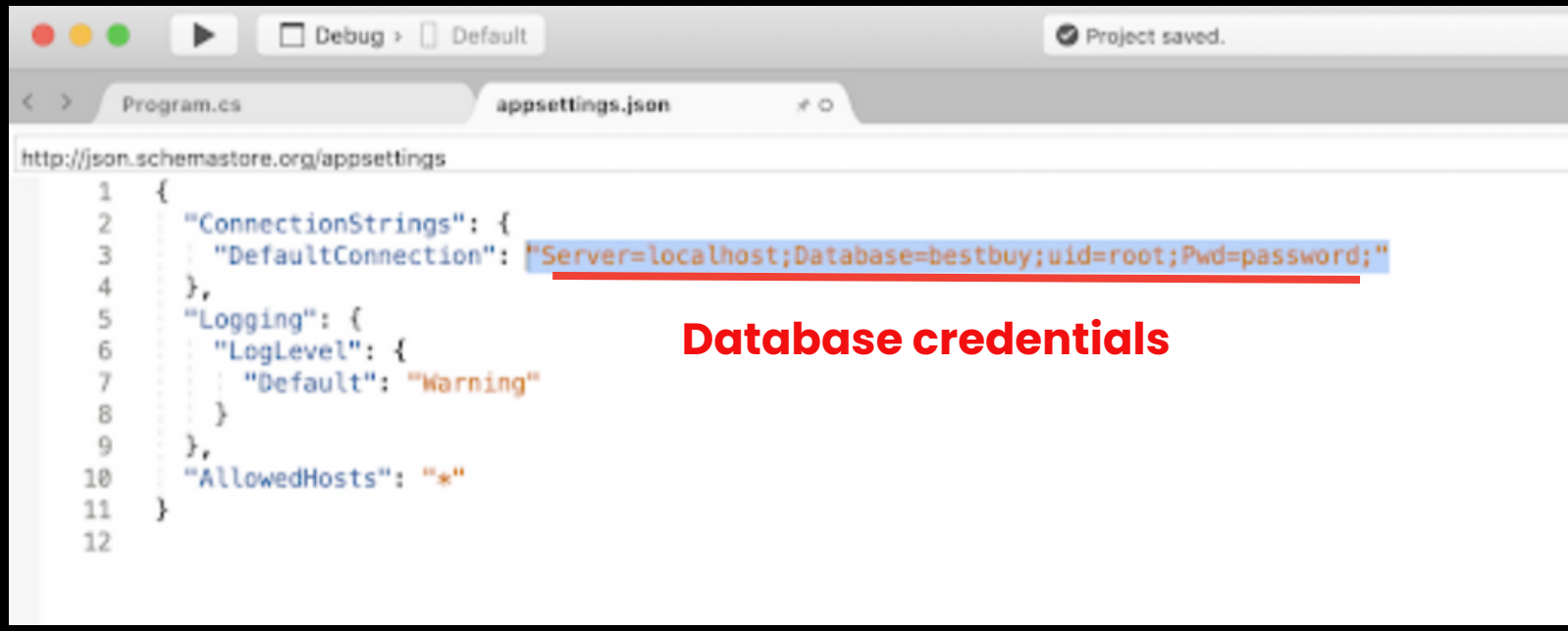
Set database credentials in appsettings.json file



Connecting the database

Will set password using Javascript Object Notation– uses name, value pairs



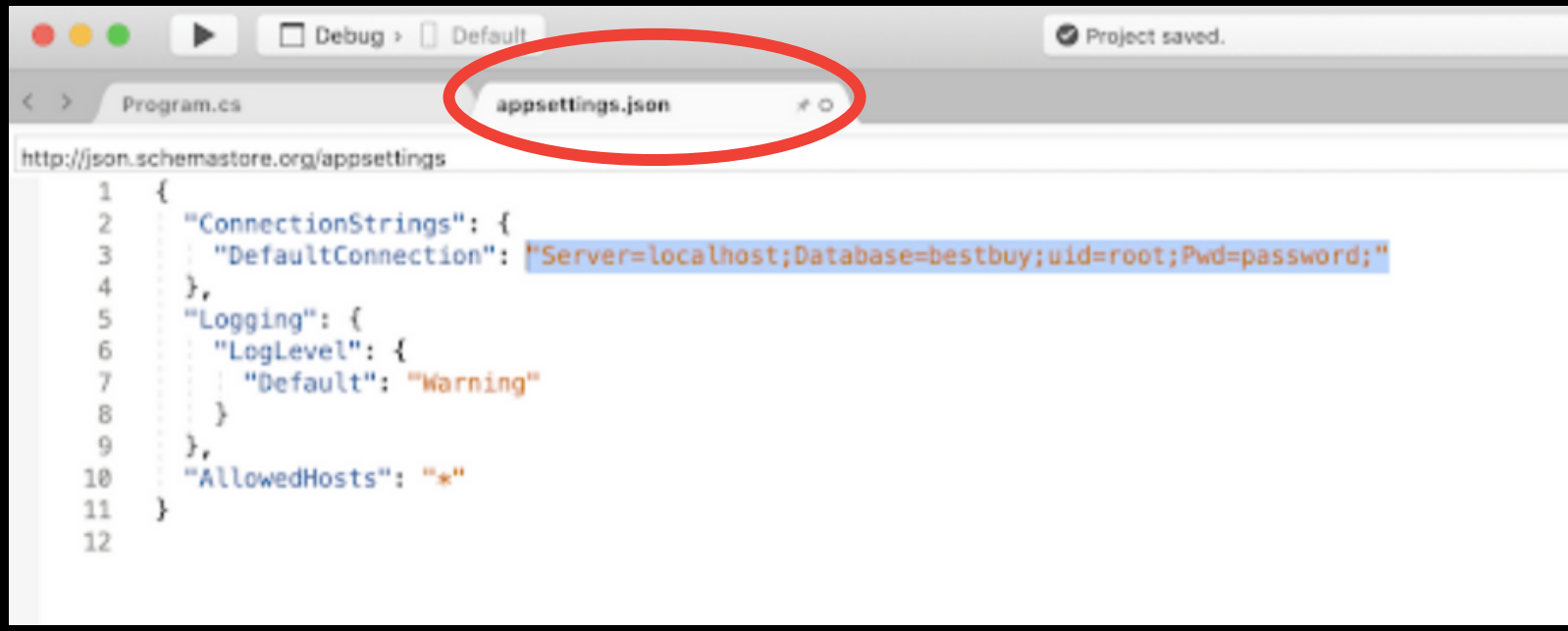


Push to GitHub

Small Problem

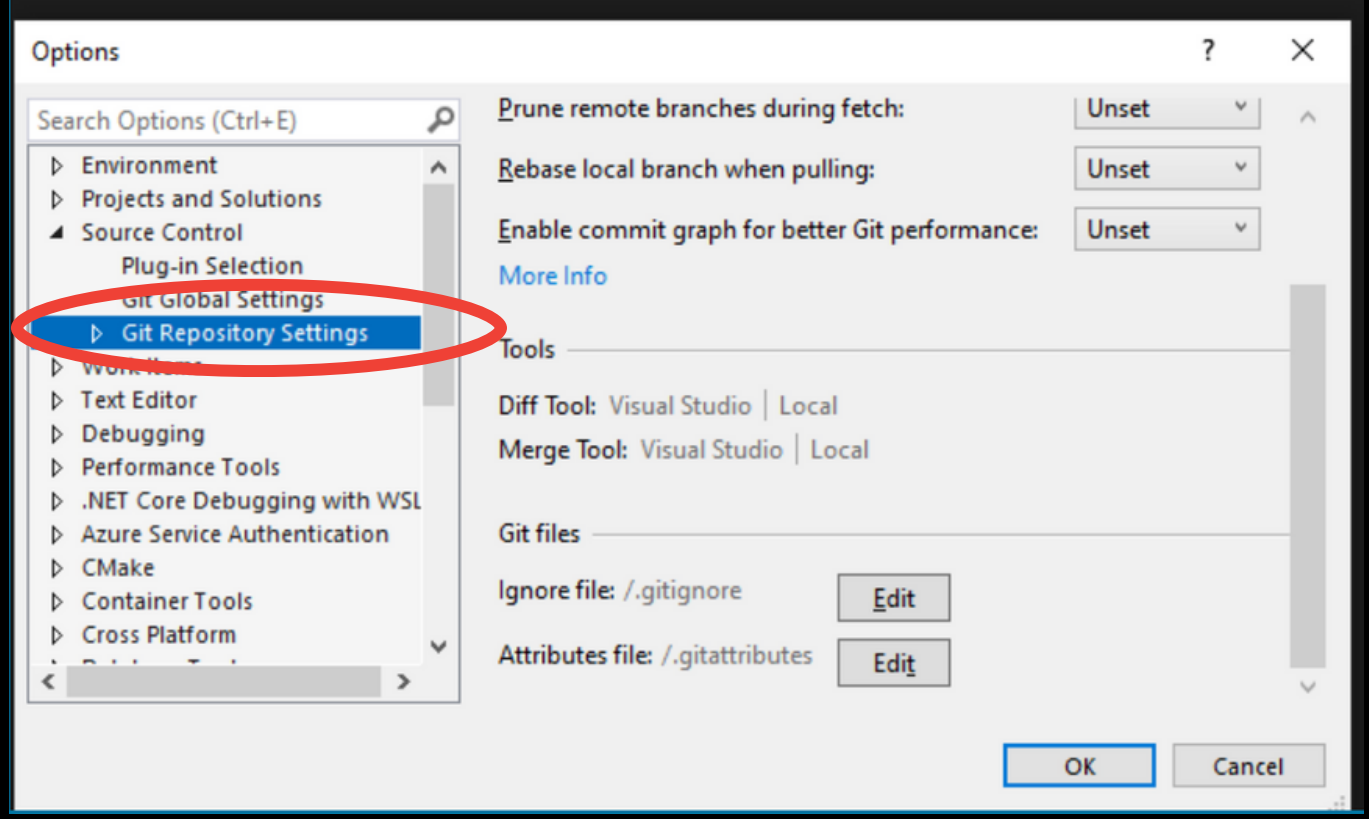
Internet Trolls

Hide password



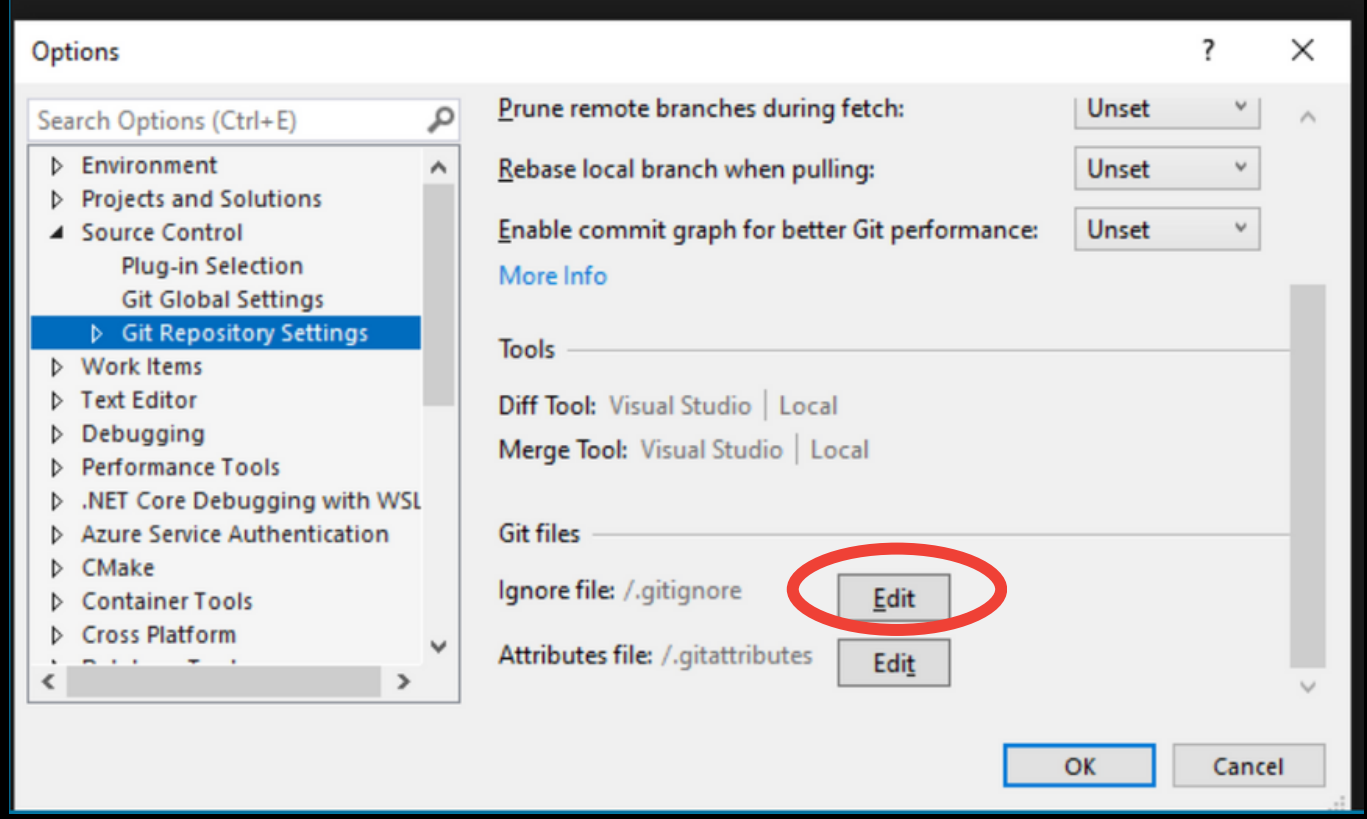
Hide your connection

Stop Git from tracking your appsettings.json file



Git Repository Settings

Stop Git from tracking your appsettings.json file



Edit

Stop Git from tracking your appsettings.json file

```
6   # User-specific files
7   *.rsuser
8   *.suo
9   *.user
10  *.userosscache
11  *.sln.docstates
12  appsettings.json
```

GitIgnore File

Add appsettings.json to gitignore file under user-specific files

```
using System;
using System.Data;  //IDbConnection Interface available in this namespace
using System.IO;
using MySql.Data.MySqlClient; //mysql.data nuget package
using Microsoft.Extensions.Configuration; //microsoft nuget package
using Dapper_Demo;
//^^^^MUST HAVE USING DIRECTIVES^^^^



var config = new ConfigurationBuilder() //creating an instance (object called config) of ConfigurationBuilder class
                .SetBasePath(Directory.GetCurrentDirectory())  //setting the base path to the current directory
                .AddJsonFile("appsettings.json") //adding the appsettings.json file to the current directory
                .Build();  //building it out
string connString = config.GetConnectionString("DefaultConnection");   //Set the connection string
IDbConnection conn = new MySqlConnection(connString);   //assigning the connection

```

Setting up the connection

In the Main method

ORM and Dapper

Video 3: Demo

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;

namespace Dapper_Demo;
3 references
internal class Department //class represents the table (we create objects using the class)
{
    //each column from the products table will be a property
    2 references
    public int DepartmentID { get; set; }
    2 references
    public string Name { get; set; }
}
```

Convert Data to Objects

Department class will represent our Departments Table

Note: We make objects from classes

```
1   using System;
2   using System.Collections.Generic;
3   using System.Linq;
4   using System.Text;
5   using System.Threading.Tasks;
6
7   namespace Dapper_Demo;
        1 reference
8   internal interface IDepartmentRepository
9   {
            3 references
10      public IEnumerable<Department> GetAllDepartments(); //Stubbed out method
            2 references
11      public void InsertDepartment(string newDepartmentName);
12  }
13
```

Interface

```
namespace Dapper_Demo;
2 references
internal class DapperDepartmentRepository : IDepartmentRepository
{
    private readonly IDbConnection _connection;
    //Constructor
    1 reference
    public DapperDepartmentRepository(IDbConnection connection)
    {
        _connection = connection;
    }

    3 references
    public IEnumerable<Department> GetAllDepartments()   //R in Crud for Read
    {
        return _connection.Query<Department>("SELECT * FROM Departments;");
    }

    2 references
    public void InsertDepartment(string newDepartmentName)  //C in Crud for Create
    {
        _connection.Execute("INSERT INTO DEPARTMENTS (Name) VALUES (@departmentName);",
         new { departmentName = newDepartmentName });
    }
}
```

Implement Interface

```
namespace Dapper_Demo;
2 references
internal class DapperDepartmentRepository : IDepartmentRepository
{
    private readonly IDbConnection _connection;
    //Constructor
    1 reference
    public DapperDepartmentRepository(IDbConnection connection)
    {
        _connection = connection;
    }

    public IEnumerable<Department> GetAllDepartments()   //R in Crud for Read
    {
        return _connection.Query<Department>("SELECT * FROM Departments;");
    }

    public void InsertDepartment(string newDepartmentName)  //C in Crud for Create
    {
        _connection.Execute("INSERT INTO DEPARTMENTS (Name) VALUES (@departmentName);",
         new { departmentName = newDepartmentName });
    }
}
```

Repository Class

Where Dapper methods will live

```
9   namespace Dapper_Demo;
    2 references
10  internal class DapperDepartmentRepository : IDepartmentRepository
11  {
12      private readonly IDbConnection _connection;
13      //Constructor
        1 reference
14      public DapperDepartmentRepository(IDbConnection connection)
15      {
16          _connection = connection;
17      }
18
        3 references
19      public IEnumerable<Department> GetAllDepartments()   //R in Crud for Read
20      {
21          return _connection.Query<Department>("SELECT * FROM Departments;");
22      }
23
        2 references
24      public void InsertDepartment(string newDepartmentName)  //C in Crud for Create
25      {
26          _connection.Execute("INSERT INTO DEPARTMENTS (Name) VALUES (@departmentName);",
27           new { departmentName = newDepartmentName });
28      }
29  }
30
```

Constructor

Which section is the constructor?

```
9   namespace Dapper_Demo;
    2 references
10  internal class DapperDepartmentRepository : IDepartmentRepository
11  {
12      private readonly IDbConnection _connection;
13      //Constructor
        1 reference
14      public DapperDepartmentRepository(IDbConnection connection)
15      {
16          _connection = connection;
17      }
18
        3 references
19      public IEnumerable<Department> GetAllDepartments()   //R in Crud for Read
20      {
21          return _connection.Query<Department>("SELECT * FROM Departments;");
22      }
23
        2 references
24      public void InsertDepartment(string newDepartmentName)  //C in Crud for Create
25      {
26          _connection.Execute("INSERT INTO DEPARTMENTS (Name) VALUES (@departmentName);",
27           new { departmentName = newDepartmentName });
28      }
29  }
30
```

Constructor

Has the same name as the class, but does not have a return type

```
9   namespace Dapper_Demo;
    2 references
10  internal class DapperDepartmentRepository : IDepartmentRepository
11  {
12      private readonly IDbConnection _connection;
13      //Constructor
        1 reference
14      public DapperDepartmentRepository(IDbConnection connection)
15      {
16          _connection = connection;
17      }
18
        3 references
19      public IEnumerable<Department> GetAllDepartments()   //R in Crud for Read
20      {
21          return _connection.Query<Department>("SELECT * FROM Departments;");
22      }
23
        2 references
24      public void InsertDepartment(string newDepartmentName)  //C in Crud for Create
25      {
26          _connection.Execute("INSERT INTO DEPARTMENTS (Name) VALUES (@departmentName);",
27           new { departmentName = newDepartmentName });
28      }
29  }
30
```

Setting the connection to the private field

Constructor

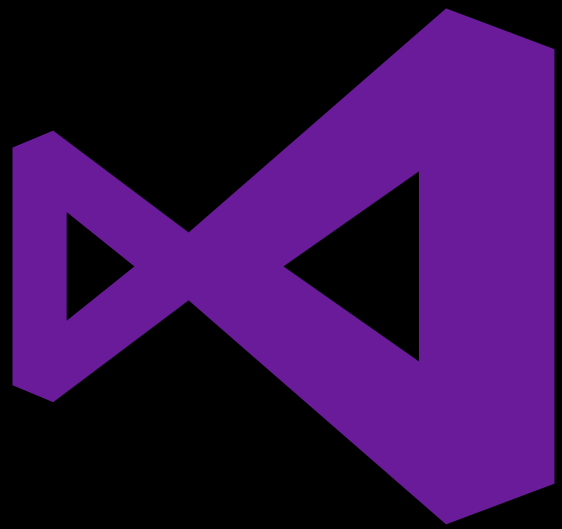
```
namespace Dapper_Demo;
2 references
internal class DapperDepartmentRepository : IDepartmentRepository
{
    private readonly IDbConnection _connection;
    //Constructor
    1 reference
    public DapperDepartmentRepository(IDbConnection connection)
    {
        _connection = connection;
    }
```

Constructor

The benefit of having connection **private and readonly** is that you can't inadvertently change it from another part of the DepartmentRepository class after it is initialized.
The readonly modifier ensures the **field can only be given a value during its initialization or in its class constructor**.

Steps:

Create a Console App



Create appsettings.json file (Make sure Git is not tracking it)

Steps:

****

3. ****
4. Enter database credentials (password) in appsettings.json
5. Verify git is not tracking appsetting.json
6. Add Nuget Packages
7. Configure connection to database in the Main method
8. Create classes/properties that represent data
9. Create an interface for repository
10. Create class that implements interface (where dapper methods live)
11. Call methods in Program.cs

See "How" Section of "lecture material" in Portal Lecture

1. Do not name your project file "Dapper"

ORM

Plumbers of the Programming World

About Us

Lorem ipsum dolor sit amet, consectetur adipiscing elit, sed do eiusmod tempor incididunt ut labore et dolore magna aliqua. Ut enim ad minim veniam, quis nostrud exercitation ullamco laboris nisi ut aliquip ex ea commodo consequat.

Our Team

Olivia Wilson
CEO

Aaron Loeb
Manager

Claudia Alves
Accounting

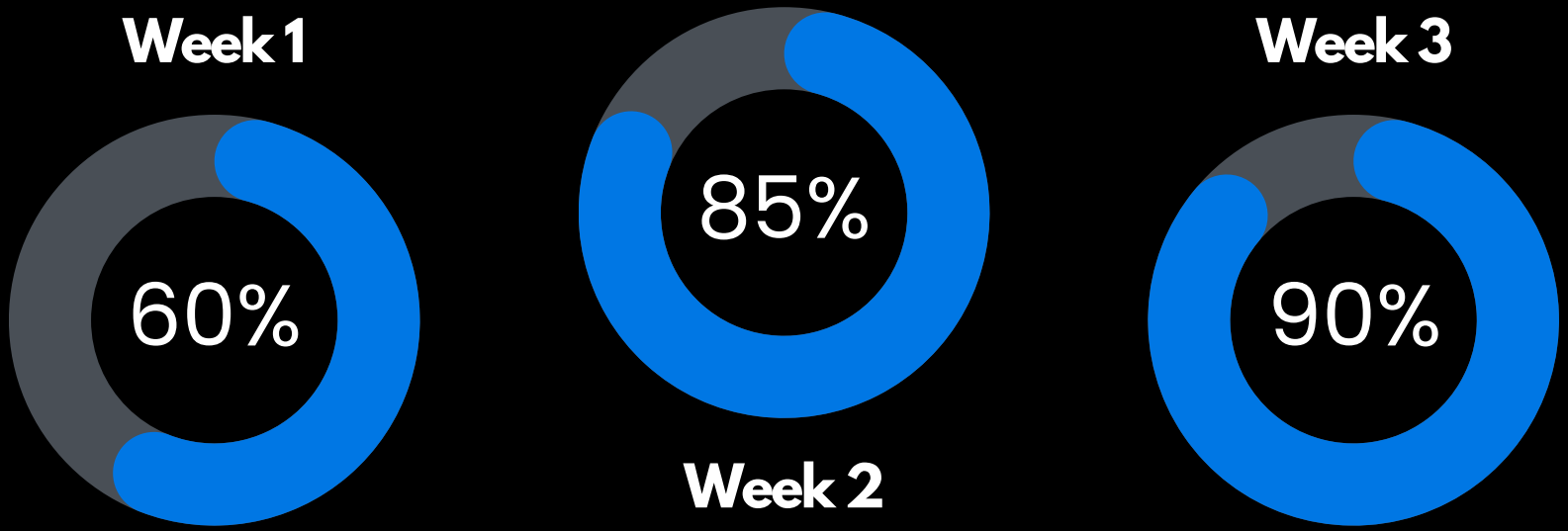
Experience

Lorem ipsum dolor sit amet, consectetur adipiscing elit, sed do eiusmod tempor incididunt ut labore et dolore magna aliqua. Ut enim ad minim veniam, quis nostrud exercitation ullamco laboris nisi ut aliquip ex ea commodo consequat.

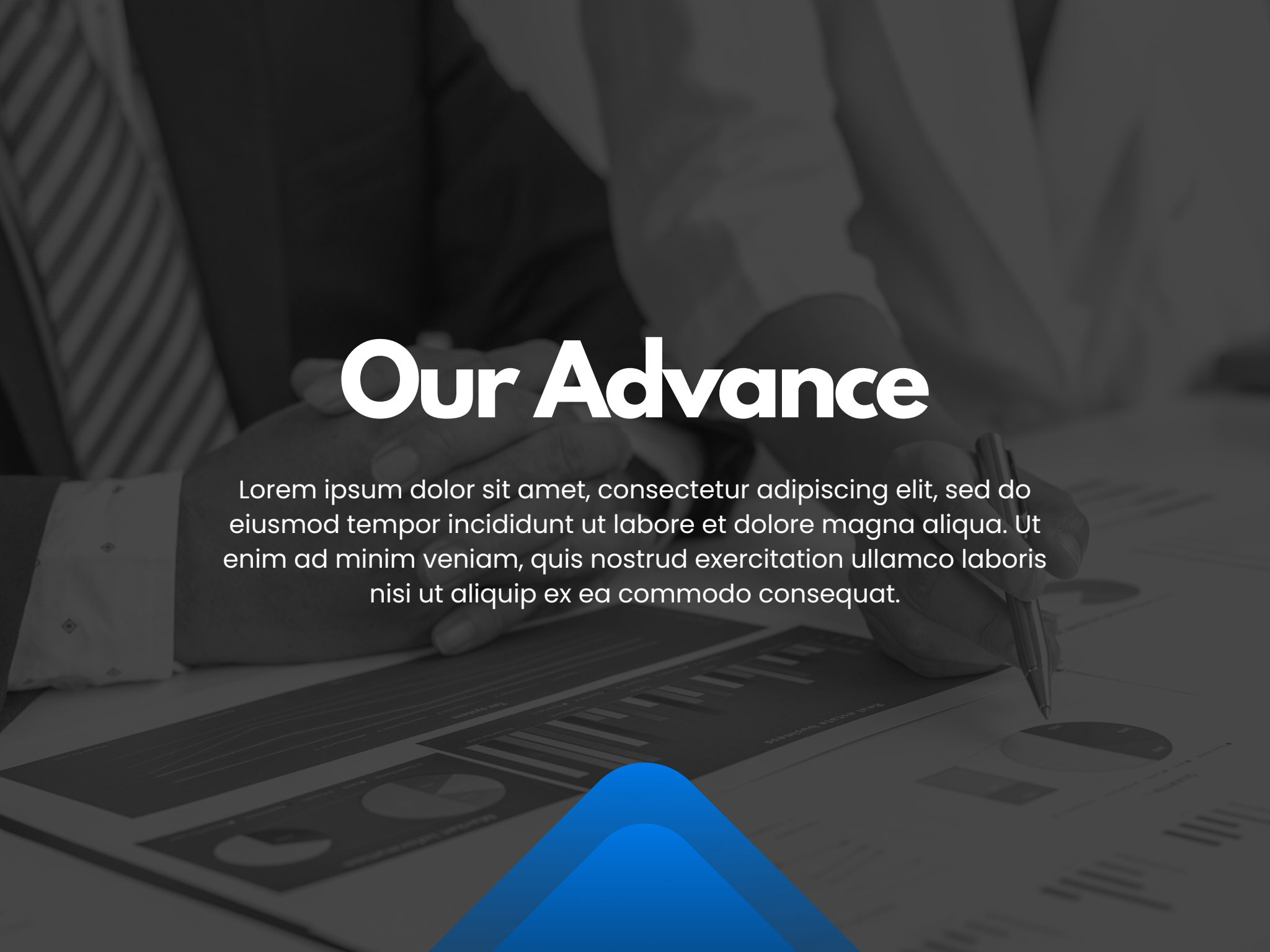
Services

Lorem ipsum dolor sit amet, consectetur adipiscing elit, sed do eiusmod tempor incididunt ut labore et dolore magna aliqua. Ut enim ad minim veniam, quis nostrud exercitation ullamco laboris nisi ut aliquip ex ea commodo consequat.

Infographics



Lorem ipsum dolor sit amet, consectetur adipiscing elit, sed do eiusmod tempor incididunt ut labore et dolore magna aliqua.



Our Advance

Lorem ipsum dolor sit amet, consectetur adipiscing elit, sed do eiusmod tempor incididunt ut labore et dolore magna aliqua. Ut enim ad minim veniam, quis nostrud exercitation ullamco laboris nisi ut aliquip ex ea commodo consequat.

Contact Us

Lorem ipsum dolor sit amet, consectetur adipiscing elit, sed do eiusmod tempor incididunt ut labore et dolore magna aliqua.

+123-456-7890

@reallygreatsite

123 Anywhere St., Any City, ST 12345

Thank You