CST-350 Activity #3 Database and Users

Table of Contents

Learning Objectives

1. Create a user's table to store usernames and passwords.
2. Build a page with restricted access

SQL Server on a Macintosh

This section applies only to students using Windows for ARM. This normally is seen on a Macintosh running Windows on a virtual machine. Windows "Copilot" is also an ARM-specific version of Windows that may not support MS SQL Server. Native x86 Intel Windows users can skip to the next section.

The problem

SQL Server requires an x64 CPU to run. If you are running Windows for ARM64 on Parallels on a Macintosh processor, the standard SQL Server will not run.

There are two good options for running a database with Visual Studio on a Macintosh:

1. MySQL on MAMP
2. Azure SQL Edge

Option 1 MySQL

GCU students who have taken CST-245 are already familiar with using MAMP and Visual Studio. A C# application requires a MySQL driver and MySQL connection string. These minor modifications will be needed in the instructions that follow.

Option 2 Azure SQL Edge

Azure SQL Edge is a variant of MS **SQL Server** designed specifically to run on containers. Azure SQL Edge does not have all the features of the standard MS SQL Server, but it will work fine for the small projects we are building in this course.

Understanding Containers and Docker

- A container is similar to a virtual machine in that applications can run on a platform that they were not originally designed for.
- macOS does not natively support Azure SQL Edge so Docker is necessary for running Azure SQL Edge on a Mac.
- When you pull a Docker image such as **mcr.microsoft.com/azure-sql-edge** you are pulling a pre-built container image that includes all the necessary dependencies and configurations to run Azure SQL Edge.
- Microsoft creates and maintains the mcr.microsoft.com/azure-sql-edge image.

1. Install Docker for Mac

https://www.docker.com/products/docker-desktop

2. Launch the Docker program

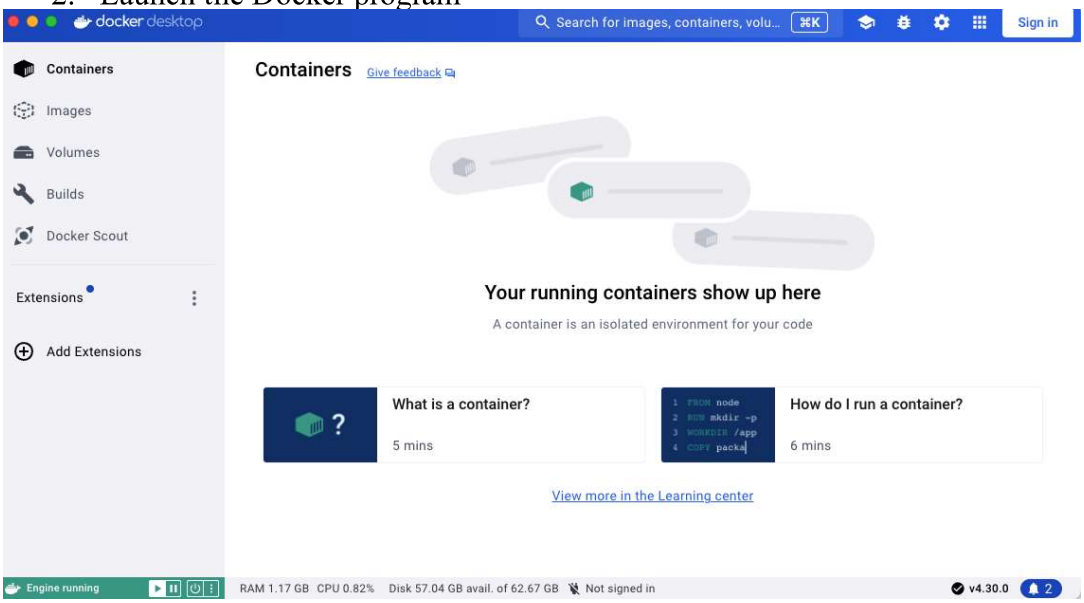


Figure 1 Docker for Mac is running. No containers have been installed yet.

3. Get the SQL Server Docker Image by searching for the Azure SQL Edge package using Google or DockerHub.

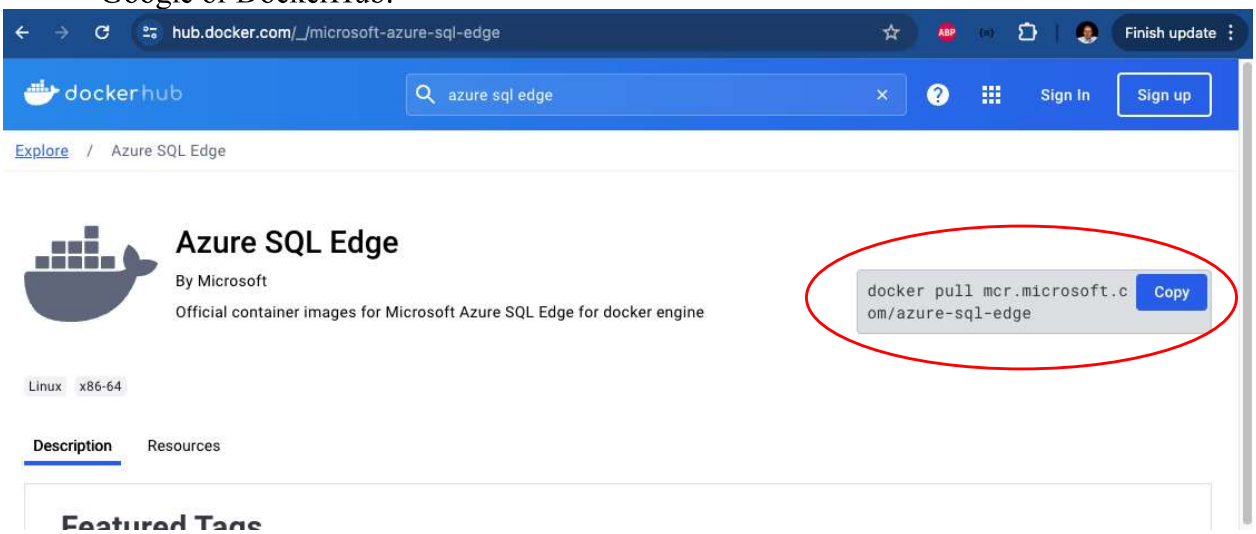


Figure 2 Azure SQL Edge at Docker Hub pages show the docker "pull" command.

Open a terminal and run the following command to pull the latest Azure Edge SQL Server image:

docker pull mcr.microsoft.com/azure-sql-edge

```
shadsluiter@shads-MBP ~ % docker pull mcr.microsoft.com/azure-sql-edge

Using default tag: latest
latest: Pulling from azure-sql-edge
c58359f0ed07: Pull complete
f9c126982b5c: Pull complete
589ba23f4d73: Pull complete
0c037bc6ac64: Pull complete
ce1f004ff642: Pull complete
4e0b1d630a9d: Pull complete
cf712679c0f8: Pull complete
7f5ed2ab3c5b: Pull complete
56e4c7793de3: Pull complete
89f8b7dcee44: Pull complete
82fa393cf611: Pull complete
Digest: sha256:902628a8be89e35dfb7895ca31d602974c7bafde4d583a0d0873844feb1c42cf
Status: Downloaded newer image for mcr.microsoft.com/azure-sql-edge:latest
mcr.microsoft.com/azure-sql-edge:latest

What's Next?
  1. Sign in to your Docker account → docker login
  2. View a summary of image vulnerabilities and recommendations → docker scout qu
ickview mcr.microsoft.com/azure-sql-edge
shadsluiter@shads-MBP ~ %
```

Figure 3 Terminal output after running the azure sql edge install command.

4. Run the following command to start a new SQL Server container. Feel free to change the password if you like.

```
docker run -e 'ACCEPT_EULA=Y' -e 'SA_PASSWORD=sqlserverpassword123!@#' -p
1433:1433 --name azuresql -d mcr.microsoft.com/azure-sql-edge
```

5. Return to the Docker app to view the status of the new container.

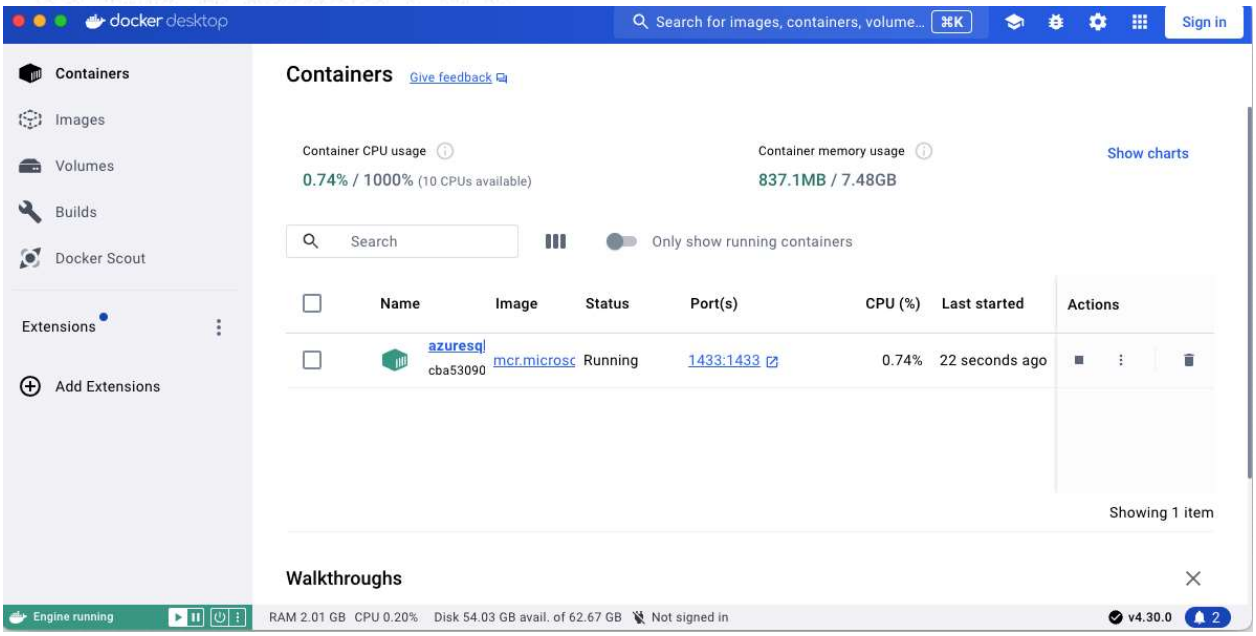


Figure 4 Docker application shows that "azuresql" is running on port 1433.

6. You can also verify the service is running from the terminal by typing “docker ps” at the command line.

```
shadsluiter@shads-MBP ~ % docker pull mcr.microsoft.com/azure-sql-edge

Using default tag: latest
latest: Pulling from azure-sql-edge
c58359f0ed07: Pull complete
f9c126982b5c: Pull complete
589ba23f4d73: Pull complete
0c037bc6ac64: Pull complete
ce1f004ff642: Pull complete
4e0b1d630a9d: Pull complete
cf712679c0f8: Pull complete
7f5ed2ab3c5b: Pull complete
56e4c7793de3: Pull complete
89f8b7dcee44: Pull complete
82fa393cf611: Pull complete
Digest: sha256:902628a8be89e35dfb7895ca31d602974c7bafde4d583a0d0873844feb1c42cf
Status: Downloaded newer image for mcr.microsoft.com/azure-sql-edge:latest
mcr.microsoft.com/azure-sql-edge:latest

What's Next?
  1. Sign in to your Docker account → docker login
  2. View a summary of image vulnerabilities and recommendations → docker scout qu
ickview mcr.microsoft.com/azure-sql-edge
shadsluiter@shads-MBP ~ % docker run -e 'ACCEPT_EULA=Y' -e 'SA_PASSWORD=sqlserverp
assword123!@#' -p 1433:1433 --name azuresql -d mcr.microsoft.com/azure-sql-edge

cba530909968c41b68506dc2566816d7cdac1d12de8418ba72c10f1acc6f68b7
shadsluiter@shads-MBP ~ % docker ps
CONTAINER ID   IMAGE                                  COMMAND                  CREATED
         STATUS         PORTS                                  NAMES
cba530909968   mcr.microsoft.com/azure-sql-edge   "/opt/mssql/bin/perm…"   2 minut
es ago   Up 2 minutes   1401/tcp, 0.0.0.0:1433->1433/tcp   azuresql
shadsluiter@shads-MBP ~ %
```

Figure 5 Terminal command 'docker ps' shows that a contains is running on port 1433.

7. Get the ip address from your Macintosh computer. You can find this in the settings of the WIFI area of network settings or run the following terminal command:

ipconfig getifaddr en0

```
shadsluiter@shads-MBP ~ % ipconfig getifaddr en0
192.168.0.52
shadsluiter@shads-MBP ~ %
```

Figure 6 Current IP address of the Macintosh OS.

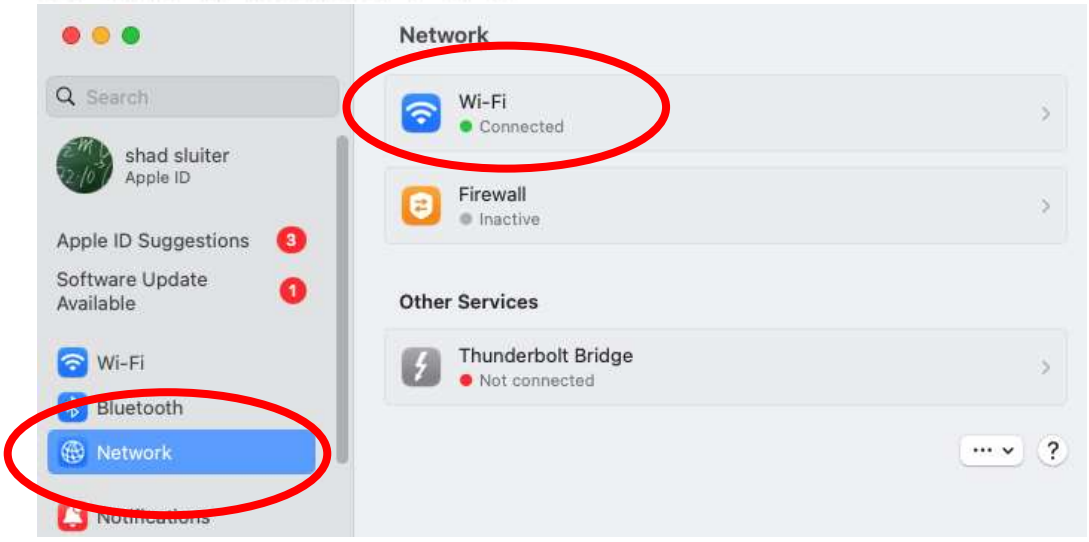


Figure 7 The location of the Network settings.

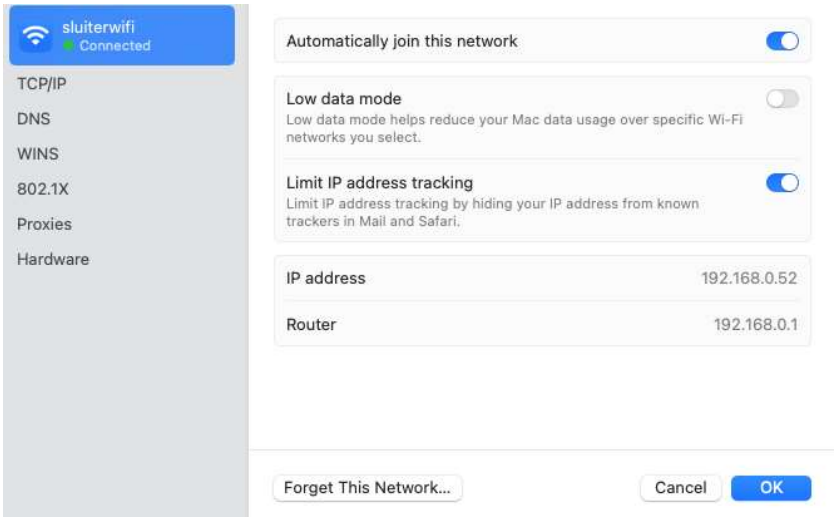


Figure 8 IP address listed under the WIFI connection.

8. In the Virtual Machine (Parallels), return to Visual Studio.
9. Under the View menu choose "SQL Server Object Explorer"

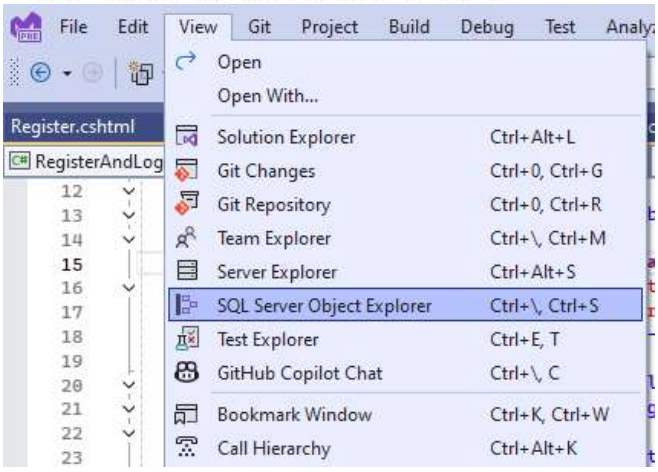


Figure 9 Opening the "SQL Server Object Explorer"

10. Right click on SQL Server and choose “Add SQL Server”

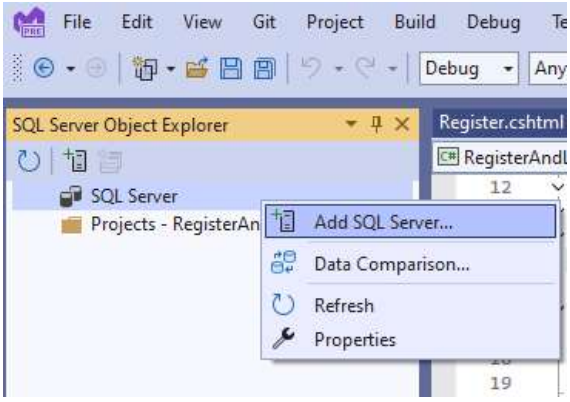


Figure 10 adding a new server

11. Configure the new connection by providing the following:

- Server name as the IP address of the Macintosh computer (192.168.0.52) followed by a comma and the port number 1433.
- Set the Authentication to “SQL Server Authentication”
- Use SA for the User Name
- Use the password you used when creating the container sqlserverpassword123!@#
- Set “Trust Server Certificate” to True.

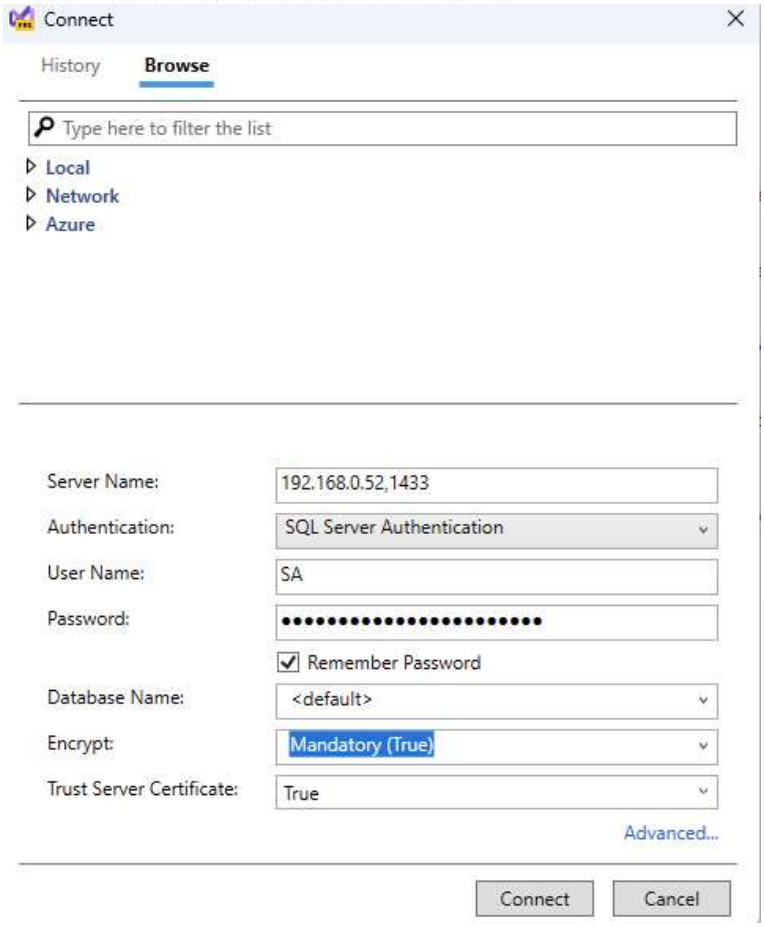


Figure 11 Configuring the new sql server connection.

12. The server should now be listed in the SQL Server Object Explorer if the connection was successful.

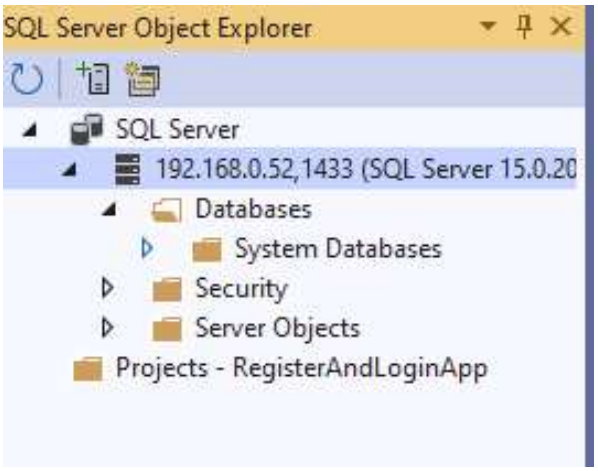
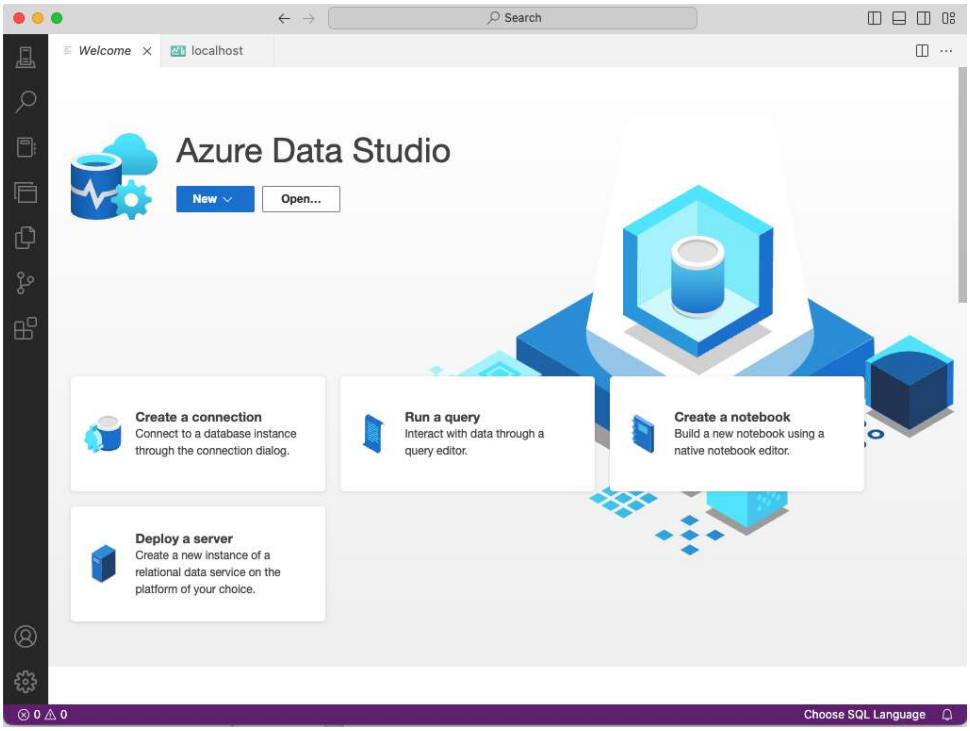


Figure 12 Server connected to 192.168.0.58

Additional SQL Manager Tools

Microsoft Provides a database manager tool called "Azure Data Studio" which allows you to connect and manage the databases on the Azure Edge server.



Choose "Create a Connection" and manage the settings as shown.

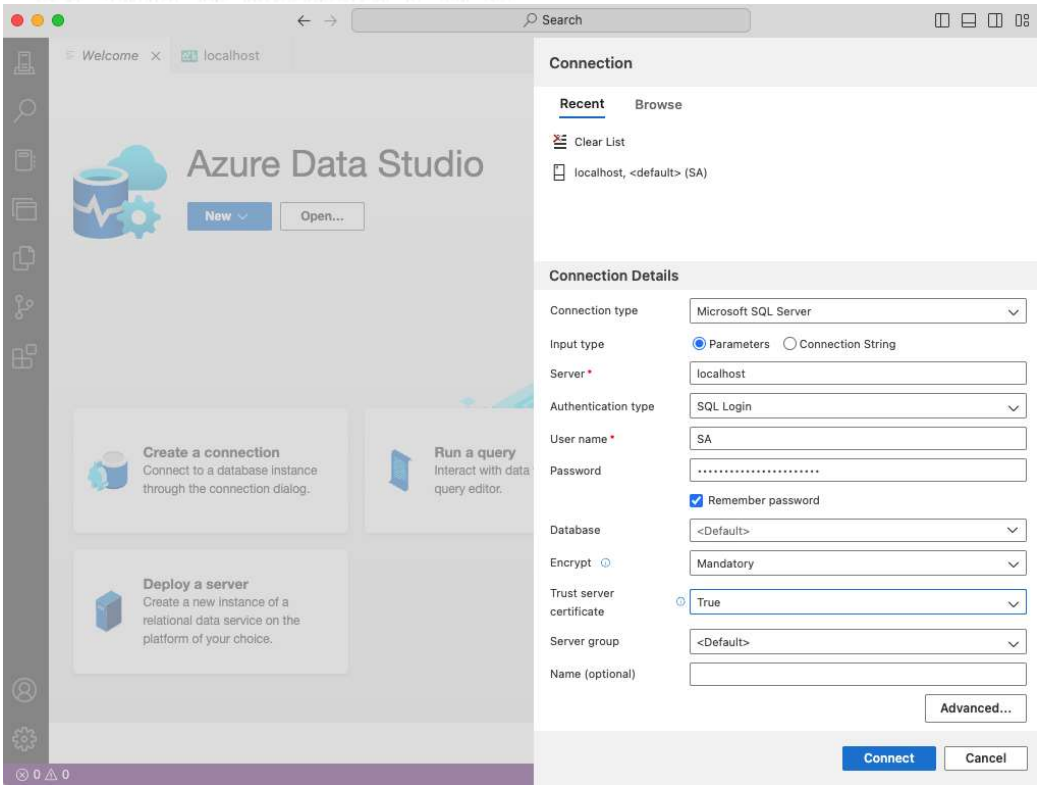


Figure 13 New connection uses "localhost".

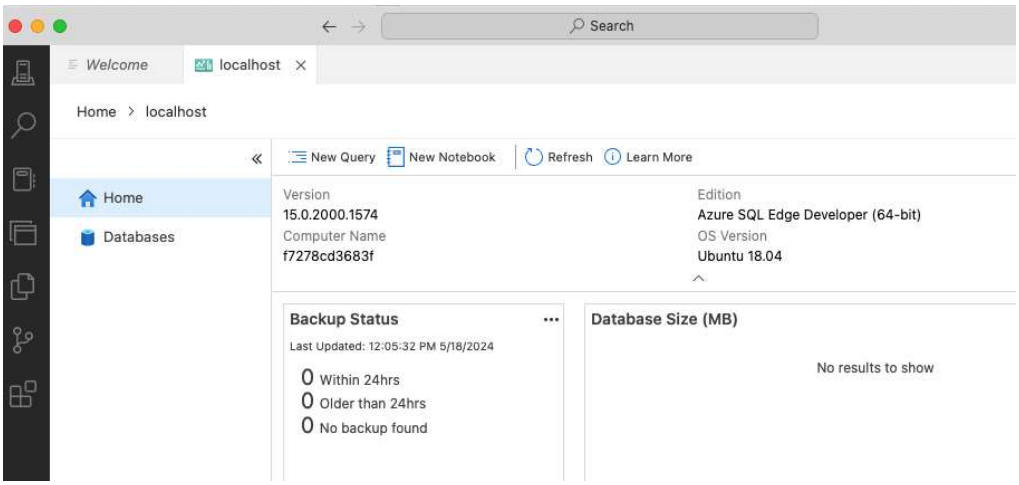


Figure 14 connection successful. Ready to create databases and tables.

This concludes the section for Macintosh users. Instructions for Windows should be valid from this point on.

GRAND CANYON UNIVERSITY

Part 1 Setting Up a User Database

Introduction

In this lesson, we will focus on setting up a user database for full-stack applications using SQL database servers. For students using Macintosh systems running Windows on a virtual machine, additional steps will guide you through setting up SQL Server on a Macintosh, utilizing either MySQL on MAMP or Azure SQL Edge.

Main Ideas

- Selecting and configuring a SQL database server for full-stack applications.
- Setting up and managing SQL Server databases on both Windows and Macintosh systems.
- Implementing a user database with CRUD operations.

By the end of this lesson, you will be able to:

1. Identify and select appropriate SQL database servers for different operating systems.
2. Set up and configure SQL Server databases using Visual Studio.
3. Implement a user database, create tables, and perform CRUD operations.
4. Connect C# applications to the SQL database using appropriate connection strings and manage user data.

Setup a Users Table:

In this section, we will add an actual database that will store usernames and password.

About ADO.NET

ADO.NET is a set of classes that exposes data access services for .NET Framework programmers. It provides a set of components for creating distributed, data-sharing applications. ADO stands for ActiveX Data Objects.

Key Components of ADO.NET

Data Providers: A set of components including **Connection**, **Command**, **DataReader**, and **DataAdapter** objects used to connect to a data source, execute commands, and retrieve results.

DataSet: A memory-resident representation of data that provides a consistent relational programming model regardless of the data source.

DataTable: Represents one table of in-memory data.

DataReader: Provides a fast, forward-only, read-only cursor for accessing data from a data source.

How ADO.NET is Used in This Lesson

In this lesson, ADO.NET is used to connect a C# application to a SQL Server database, perform CRUD operations, and manage user data.

Alternatives to ADO.NET

The approach shown in this lesson demonstrates the user of SQL statements being used to interact directly with tables and column names in the SQL server. Another popular option for SQL data access is the Entity Framework which is not shown in this lesson. Here is a comparison of the two approaches:

ADO.NET: ADO.NET is a lower-level data access technology that provides a set of classes for accessing and manipulating data from a variety of sources, such as databases and XML files. It requires you to write SQL queries to perform CRUD (Create, Read, Update, Delete) operations.

Entity Framework: Entity Framework is an Object-Relational Mapper (ORM) that sits on top of ADO.NET. It provides a higher level of abstraction by allowing developers to work with data as objects and properties, rather than directly dealing with database tables and columns. EF generates the SQL queries for you, based on the LINQ queries you write in your code.

1. Open SQL Server Object Explorer by selecting the **View → SQL Server Object Explorer** menu items.

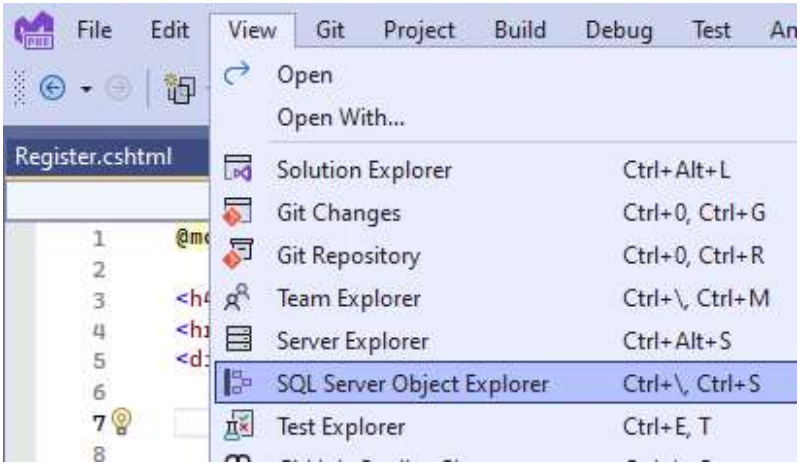


Figure 15 SQL Sever Object Explorer location

2. By default, under the SQL Servers you should see a local DB called MSSQLLocalDB listed. Expand this database server.

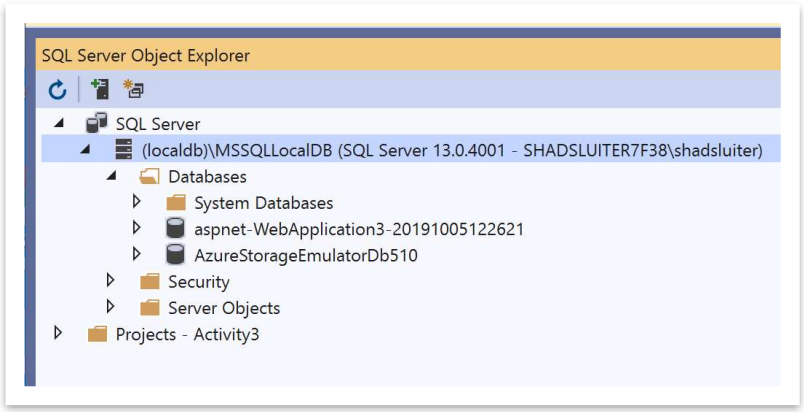


Figure 16 Default local SQL Server shown for Windows users.

3. If this server is not listed:
    a. Macintosh users should review the previous section for enabling Azure SQL Edge.
    b. Click the Add SQL Server icon.
    c. Expand the Local tree.
    1) Select the MSSQLLocalDB.
    2) Select Windows Authentication.
    3) Click the Connect button to add the local database server to the list of database servers in SQL Server Object Explorer.
4. Right-click on the Databases folder and select Add New Database.

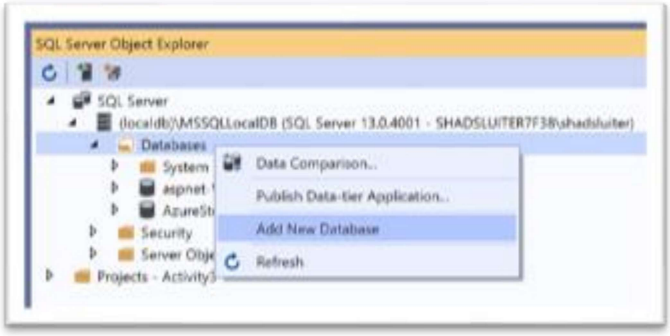


Figure 17 Adding a new database

5. Enter a Database Name of **Test**.

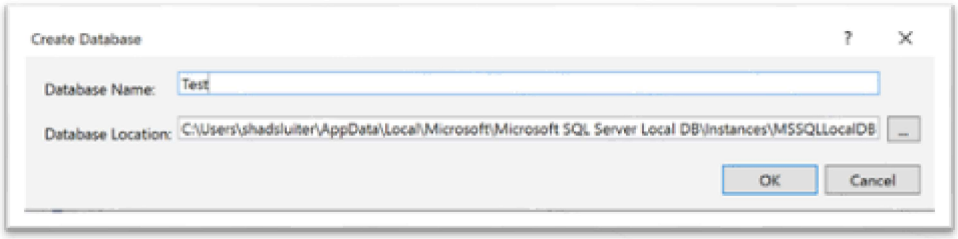


Figure 18 Adding a "Text" database to the sql server.

6. A new Database should be now available under the MSSQLLocalDB databases.

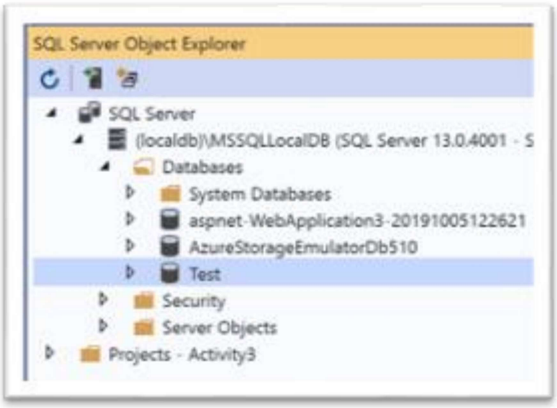


Figure 19 "Test" database has been added to the server.

7. Expand the Test database to display the Tables folder.
    a) Right-click on the **Tables** folder and select **Add New** table menu option.

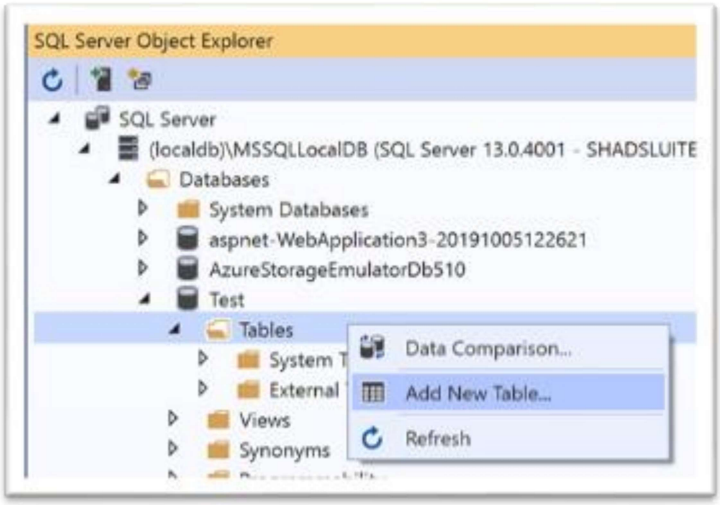


Figure 20 Adding a new table to the "Test" database.

You should see the **Table designer**.

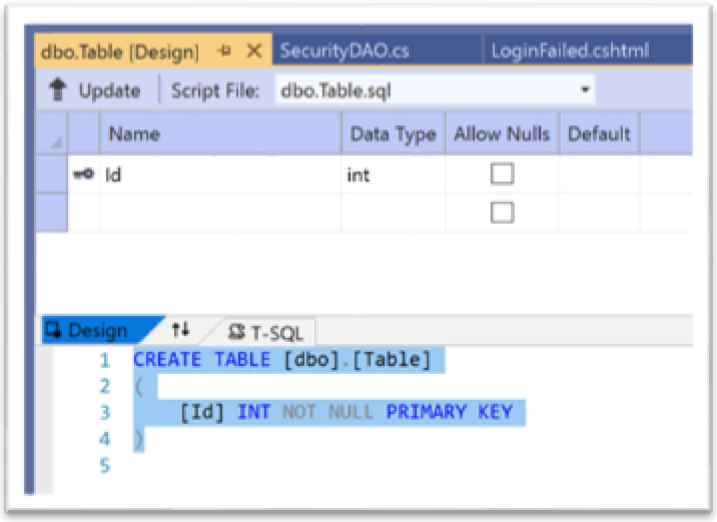


Figure 21 "Create table" command in the table designer.

b) In the T-SQL window, change the name of the Table in the Create statement to Users.

c) Add the following columns:

i. Id primary key, type int, no NULL's,

ii. auto increment in T-SQL is IDENTITY(1,1)

iii. Username, type nvarchar(50), no NULL's

iv. PasswordHash, type nvarchar(50), no NULL's

v. Salt varbinary(16), not NULL,

vi. Groups nvarchar (Max)

```
CREATE TABLE [dbo].[users] (
    [Id]           INT            IDENTITY (1, 1) NOT NULL,
    [Username]     NVARCHAR (50)  NOT NULL,
    [PasswordHash] NVARCHAR (50)  NOT NULL,
    [Salt]         VARBINARY (16) NOT NULL,
    [Groups]       NVARCHAR (MAX) NULL,
    PRIMARY KEY CLUSTERED ([Id] ASC)
);
```

Figure 22 SQL statement to create a table with three columns.

d) Click the **Update** button from the designer.

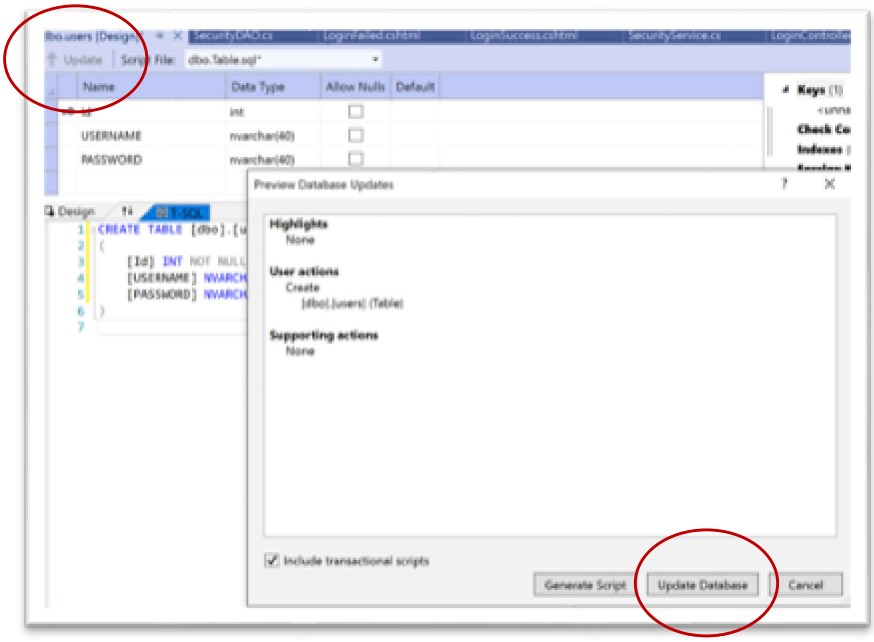


Figure 23 Update commands to run the "Create" query

e) In the Preview Database Updates dialog, click the **Update** Database button.
f) In the Output window, validate that the table and columns were created without errors.

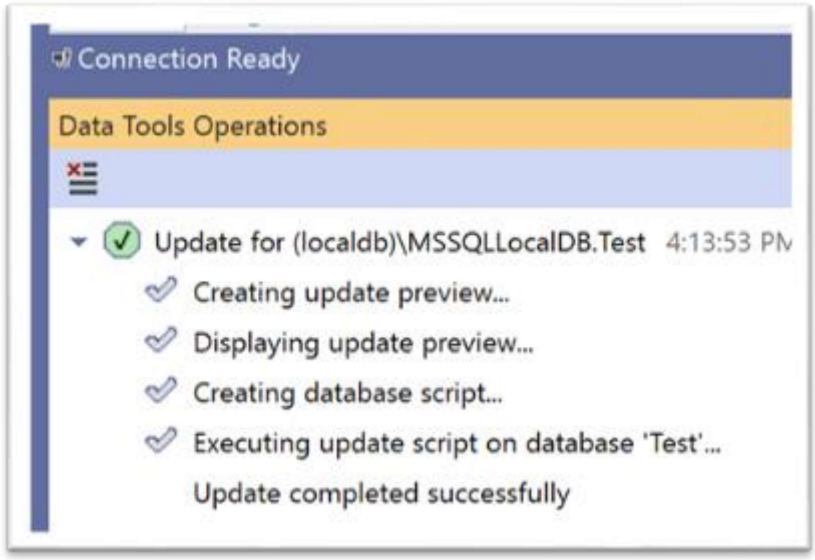


Figure 24 Success message from creating a new table.

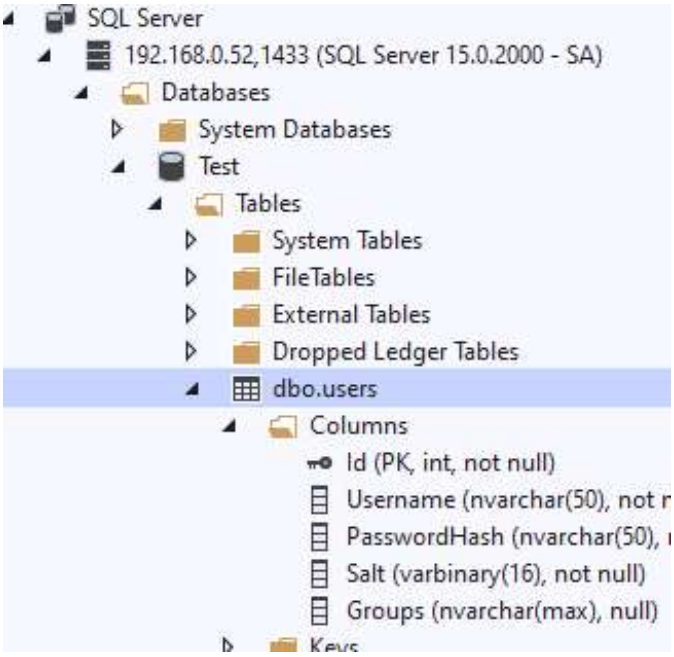


Figure 25 Expanded view of Test database and tables.

About T-SQL vs SQL

You may have noticed that "SQL" commands in the MS SQL server are referred to a "T-SQL" commands. What is the difference between these two?

T-SQL (Transact-SQL) is an extension of SQL created by Microsoft and runs only on SQL Server or Azure Edge.

Some standard SQL commands include SELECT, INSERT, UPDATE, DELETE, CREATE, ALTER and DROP.

T-SQL supports for procedural programming statements such as DECLARE and SET for defining variables. T-SQL can perform control flow with BEGIN...END, IF...ELSE, WHILE, TRY...CATCH.

Separation of Concerns
Separation of concerns (SoC) is a software design principle that promotes dividing a program into distinct sections, each addressing a separate concern of the application. "Separation of Concerns" means that a database system should be limited to do the work of storing and retrieving data. It should not venture into the job of business logic.

Reasons to avoid T-SQL commands

1. Business Logic Separation
- By using standard SQL, business logic can be kept within the application layer rather than embedded in the database. This makes the business logic database-agnostic and easier to modify or extend without being tied to a specific RDBMS.

2. Maintainability
Keeping SQL queries simple and focused on data retrieval and manipulation, while handling complex logic in the application layer, can make the codebase easier to maintain. Changes in business rules or logic only require updates in the application code.

3. Portability
Using standard SQL ensures that the database code is portable across different database systems.

4. Testing and Debugging
Business logic in the application layer can be tested using standard software testing tools and frameworks.

5. Scalability
By keeping the business logic in the application layer, you can scale the application and database independently. For example, you can add more application servers without worrying about the database becoming a bottleneck due to complex logic.

To apply SoC effectively while working with databases:

1. Database Layer (Data Access):
- Focus on data storage, retrieval, and basic validation.
- Use standard SQL for queries and data manipulation.
- Keep database schemas, constraints, and indexes well-defined.

2. Application Layer (Business Logic):
- Implement complex business rules, data processing, and transformations.

- Use application code to interact with the database via standard SQL queries.
- Ensure the application layer handles data validation, error handling, and business workflows.

3. Service Layer (Integration):
- Manage interactions between the application and external services.
- Abstract database access to provide a clear interface for the application layer.
- Use ORM (Object-Relational Mapping) tools if needed, which can help in maintaining SoC by abstracting direct SQL queries.

Reasons why a developer may intentionally violate the SoC principle in database design

Computing Performance – In a high-traffic application, performing operations in the database may be faster than running them inside an application class.
Network Traffic – If a database can perform multiple steps before returning data to an application, there will be fewer "round trip" network requests.
Database Developer Teams – Some organizations assign the duty of SQL statement creation to a specialized DBA team. This team creates a procedure and then exposes the results as a database "view" that application developers can consume.

Debatable
There is considerable debate among developers as to which approach is a better design: separation of concerns or database systems as application components. You will find heated arguments on Reddit or Stack Overflow often with neither side conceding ground to the other.

For this course, the examples will favor the "Separation of Concerns" principle.

Validate your Test Database:

1. Open SQL Server Object Explorer by selecting the View → SQL Server Object

2. Expand the MSSQLLocalDB database server and under the Databases folder select the Test database.

3. Right-click on the Test database and select the **New Query** menu option.

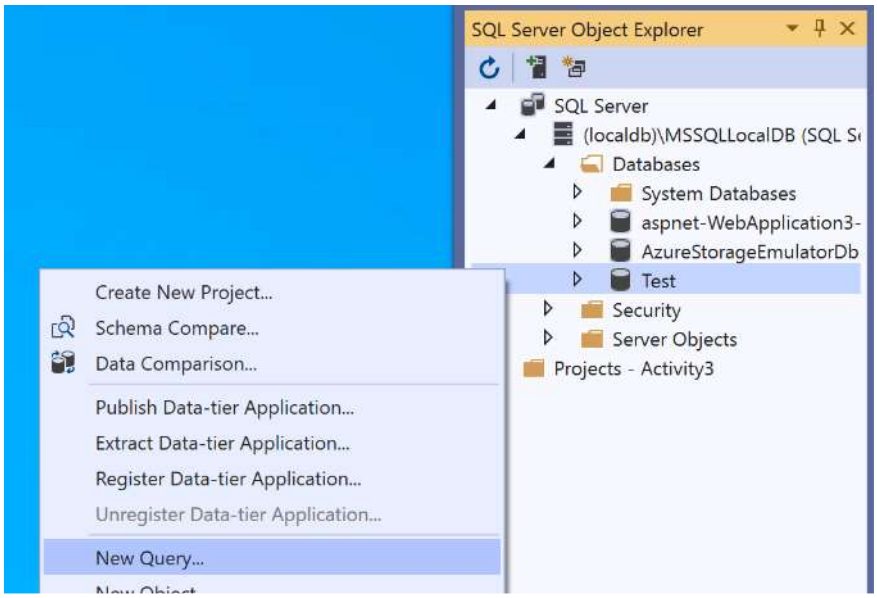


Figure 26 Creating a new query on the Test database.

4. In the query run execute the following **SQL Statement:**

SELECT * FROM dbo.Users

5. The Message window should display the columns of the Users table with no data.

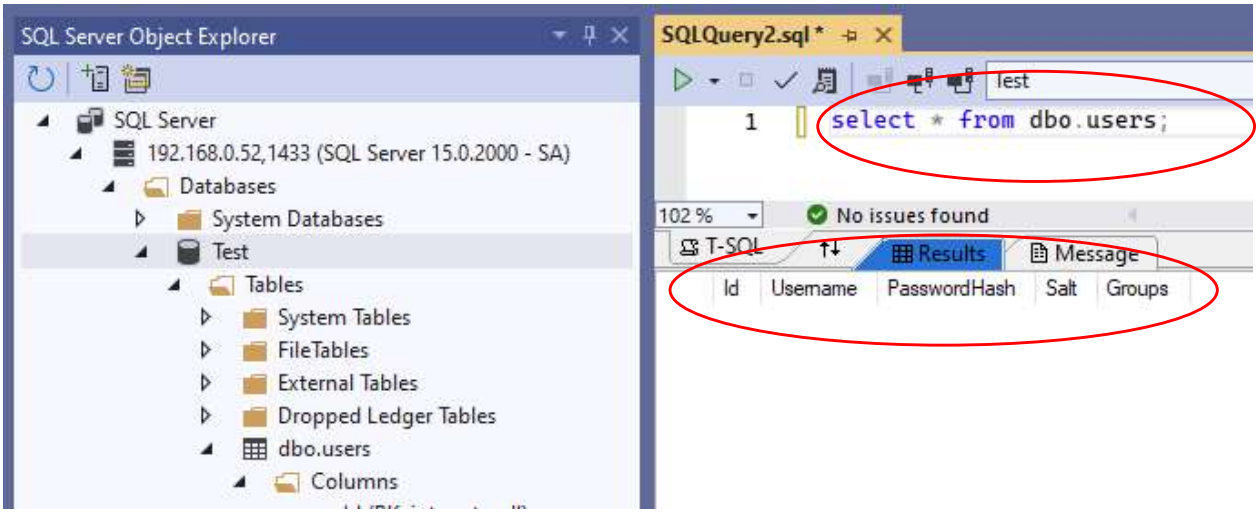


Figure 27 Select statement shows 3 columns of data but no content.

- Take a screenshot of your application running at this point.
- Paste the image into a Word document.
- Put a caption below the image explaining what is being demonstrated.

6. Insert a new row in the table by supplying a set of user data. Use the following SQL Statement as an example.

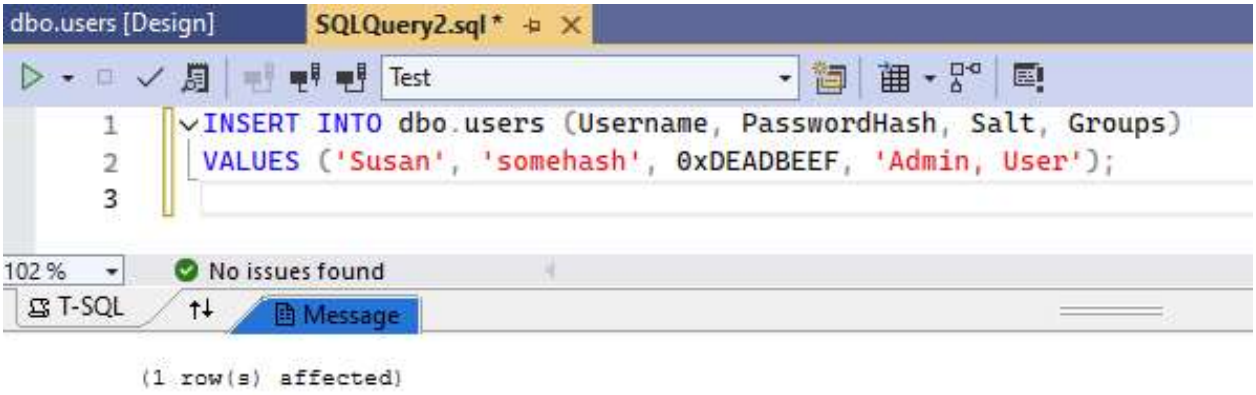


Figure 28 Inserting a new user and password into the dbo.users table

7. Confirm that the new data is saved in the table. Right-click on the dbo.users table and select View Data.

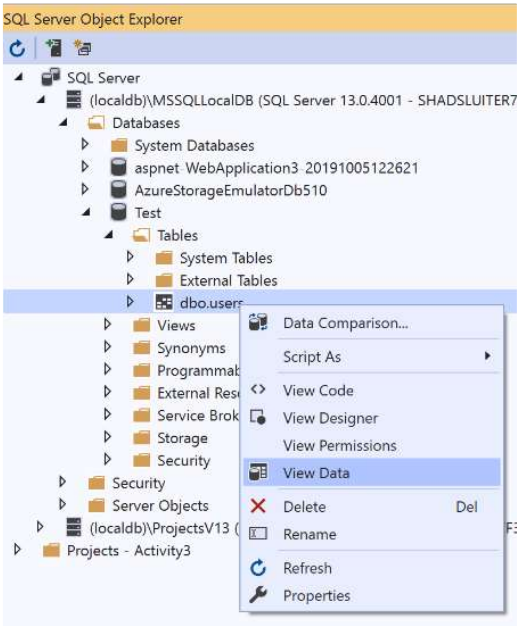


Figure 29 Running the "View Data" query.

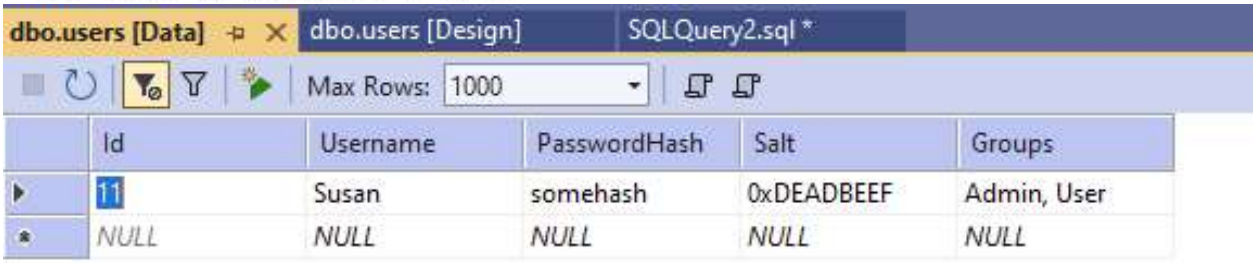


Figure 30 New user data is in the users table.

- Take a screenshot of your application running at this point.
- Paste the image into a Word document.
- Put a caption below the image explaining what is being demonstrated.

Add a User DAO using ADO.NET:

In this section, we will connect to the database from the C# code.

1) Create a new class in the Models folder. Name it UsersDAO.cs

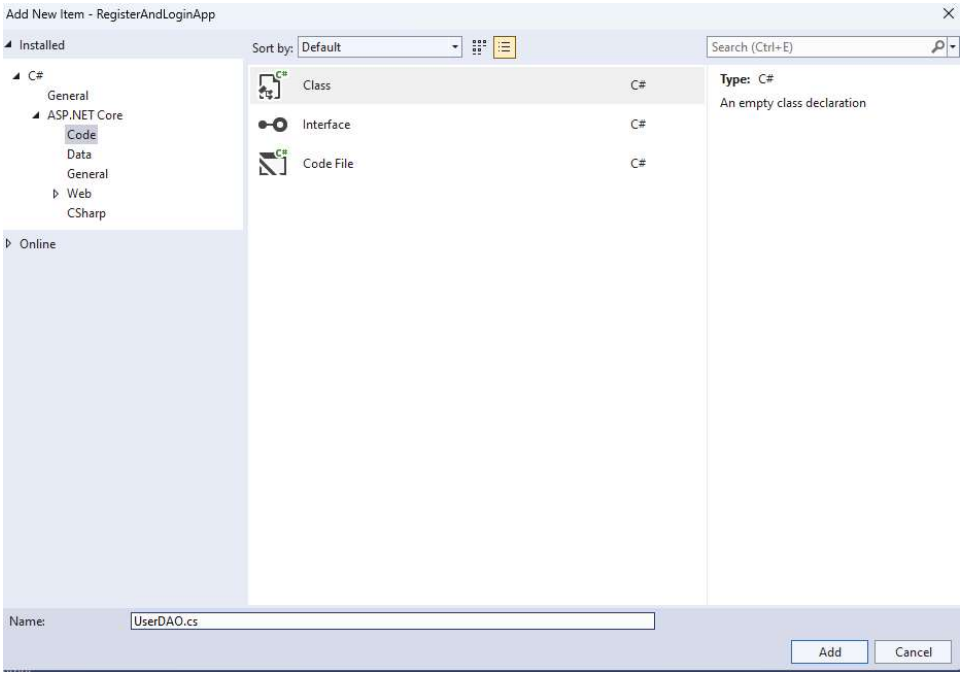


Figure 31 Adding a new class UserDAO

Figure 32 Implement the IUserManager interface in the new class.

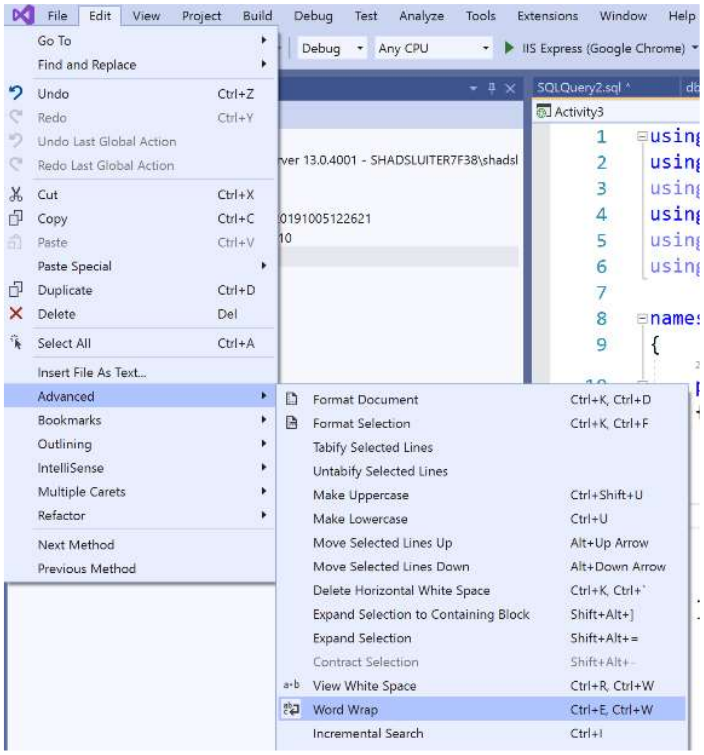


Figure 33 Enablign Word Wrap for showing long lines of code in the Visual Studio code editor.

You may wish to start by turning on **Word Wrap** in the code editor. The connection string and SQL queries can easily become wider than the width of the text editing window. Choose **Edit > Advanced > Word Wrap** to toggle the feature on or off.

2. Right-click on the red text error and choose implement the user interface.

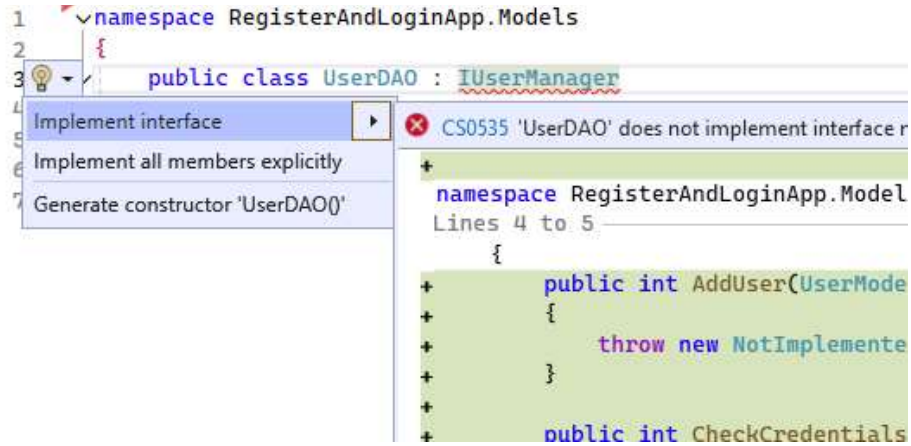


Figure 34 Implementing the interface

8. Get the properties for the Test database. Right-click the Test database and choose Properties.

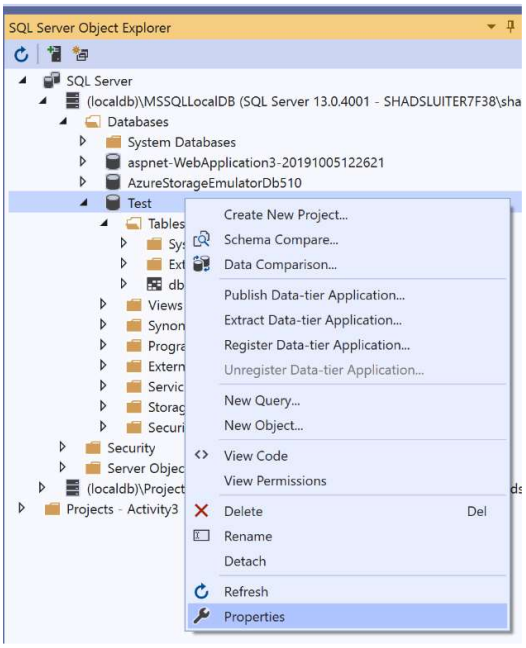


Figure 35 Getting the properties of the Test database

9. In the properties window you can right-click and copy the Connection String property.

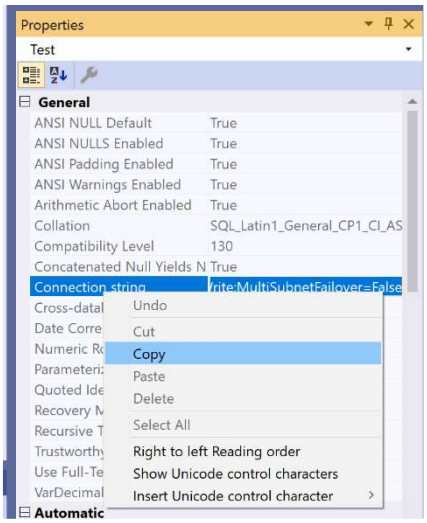


Figure 36 Copy the connection string property of the Test database.

10. Save the Connection string in a variable in the code for SecurityDAO as shown here.

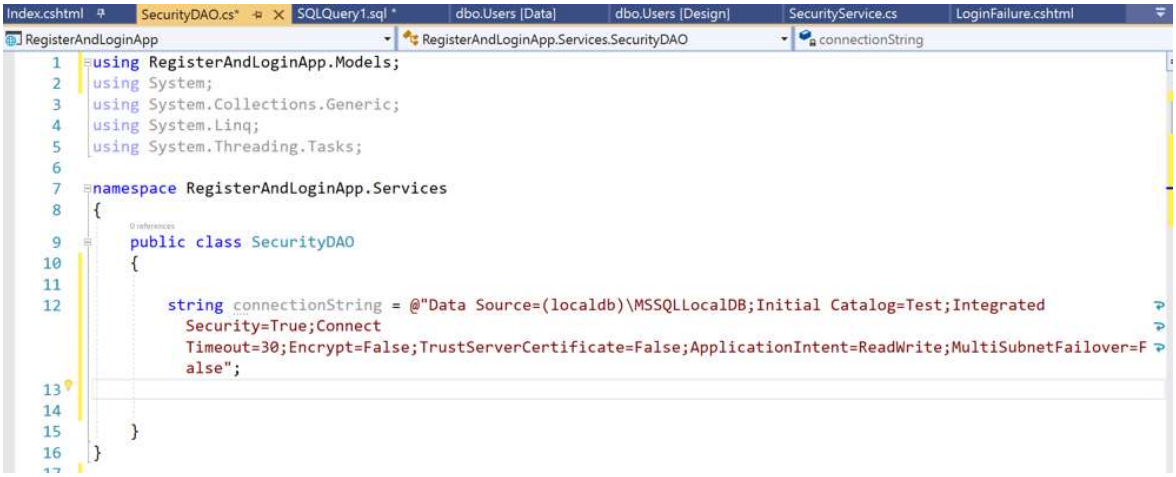


Figure 37 Connection string from SQL Server (Windows)

```
namespace RegisterAndLoginApp.Models
{
    public class UserDAO
    {
        // Azure SQL Edge.  removed  spaces from the following parameters
        // Trusted Server Certificate = True becomes TrustedServerCertificate=True
        // Application Intent = ReadWrite becomes ApplicationIntent=ReadWrite
        // Multi Subnet Failover = False becomes MultiSubnetFailover=False


        string connectionString = @"Data Source=192.168.0.52,1433;Initial Catalog=Test;User
          ID=SA;Password=sqlserverpassword123!@#;Connect
          Timeout=30;Encrypt=True;TrustServerCertificate=True;ApplicationIntent=ReadWrite;Multi
          SubnetFailover=False";
```

Figure 38 Connection string for Azure SQL Edge

```
UserDAO.cs*  dbo.users [Data]  dbo.users [Design]  UserCollection.cs  UserController.cs
RegisterAndLoginApp  RegisterAndLoginApp.Models.UserDAO  connectionString
1   using static System.Net.Mime.MediaTypeNames;
2   using static System.Runtime.InteropServices.JavaScript.JSType;
3   using System.Drawing;
4   using System;
5   using System.Data.SqlClient;
6   namespace RegisterAndLoginApp.Models
7   {
8       public class UserDAO
9       {
10
11          // mysql connection string
12          string connectionString =
              @"server=localhost;port=3306;database=Test;user=root;password=sqlserverpassword123!
              @#";
13
14
```

Figure 39 Example of a MySQL Connection string

11. Start implementing the AddUser method.

```
namespace RegisterAndLoginApp.Models
{
    public class UserDAO : IUserManager
    {

        // connection string for MS SQL, Azure SQL, MySQL, etc.

        string connectionString = @"192.168.0.52,1433; Initial Catalog = Test; User ID = SA;
          Password=sqlserverpassword123!@#;Connect Timeout = 30;
          Encrypt=True;TrustServerCertificate=True;ApplicationIntent = ReadWrite;
          MultiSubnetFailover=False";

        public int AddUser(UserModel user)
        {
            using (SqlConnection connection = new SqlConnection(ConnectionString))
            {

            }
        }
```

Figure 40 Initial code for the Adduser method. SqlConnection is not recognized.

12. You may need to install a package “System.Data.SqlClient”.

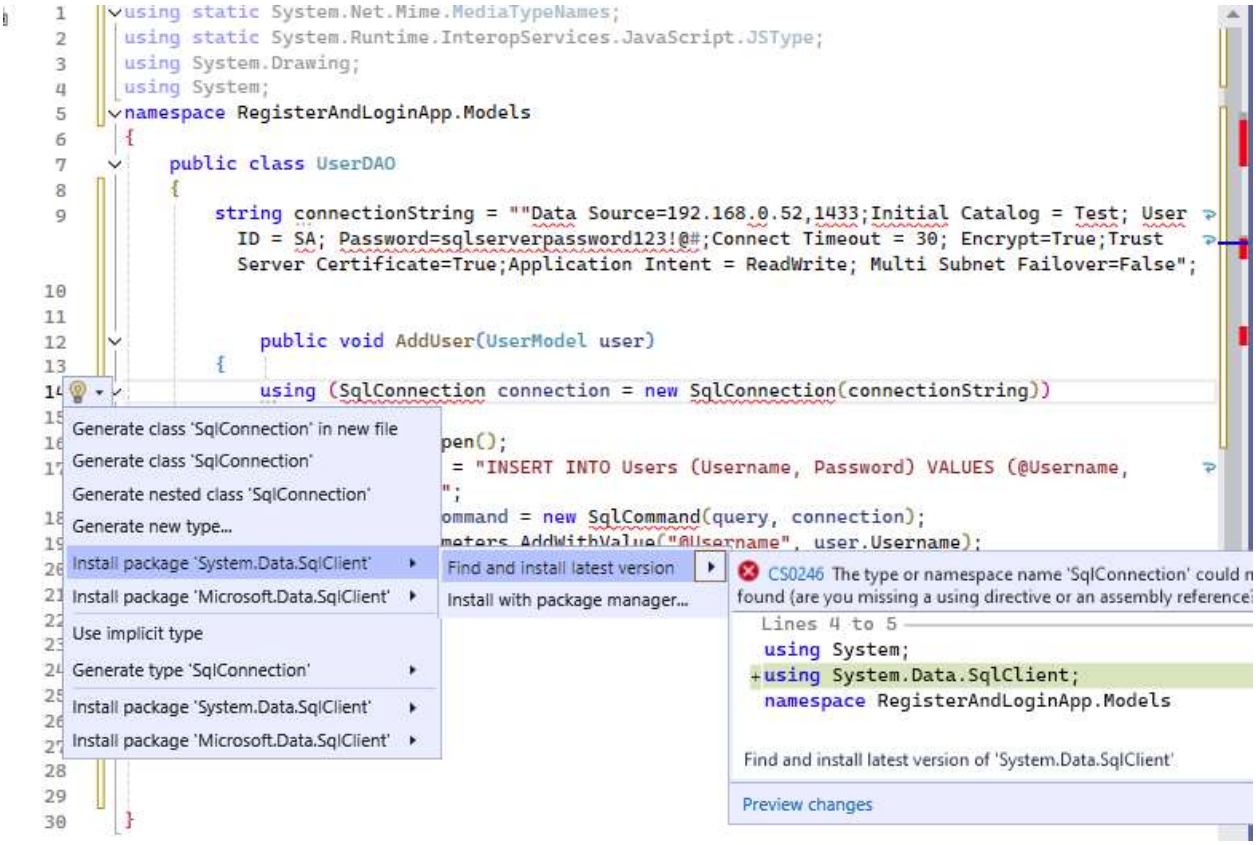


Figure 41 Installing the database sql client.

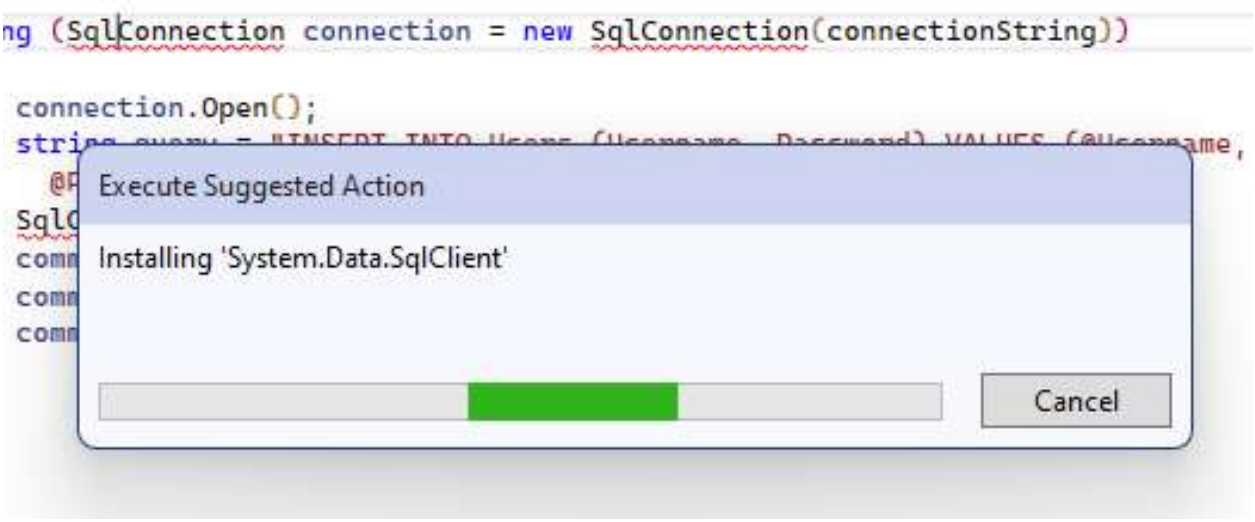


Figure 42 Waiting for the new dependency to be installed by NuGet.

13. Using the System.Data.SqlClient will resolve the SqlConnection error.

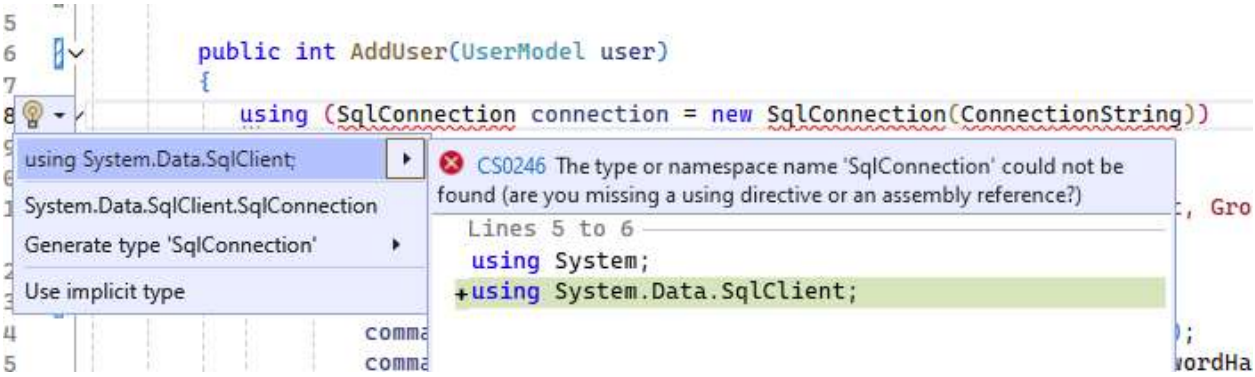


Figure 43 Using System.Data.SqlClient will resolve the sql connection error.

14. Complete the "AddUser" method.

```
17      public int AddUser(UserModel user)
18      {
19          using (SqlConnection connection = new SqlConnection(ConnectionString))
20          {
21              connection.Open();
22              string query = @"
23          INSERT INTO users (Username, PasswordHash, Salt, Groups)
24          VALUES (@Username, @PasswordHash, @Salt, @Groups);
25          SELECT SCOPE_IDENTITY();";
26
27              using (SqlCommand command = new SqlCommand(query, connection))
28              {
29                  command.Parameters.AddWithValue("@Username", user.Username);
30                  command.Parameters.AddWithValue("@PasswordHash", user.PasswordHash);
31                  command.Parameters.AddWithValue("@Salt", user.Salt);
32                  command.Parameters.AddWithValue("@Groups", user.Groups);
33
34                  // Execute the query and get the newly inserted ID
35                  object result = command.ExecuteScalar();
36                  return Convert.ToInt32(result);
37              }
38          }
39      }
```

Figure 44 SQL command to insert a new user

About "Execute" commands in ADO.NET

In ADO.NET, the SqlCommand class provides various methods to execute different types of SQL statements against a database. Each method is suited to a specific type of SQL operation including **ExecuteScalar**, **ExecuteReader**, and **ExecuteNonQuery**:

1. ExecuteScalar - Executes a query and returns a single value.

When to use it: When you need to retrieve a single value from the database, such as an aggregate value (e.g., COUNT, SUM, MAX).

```
string query = "SELECT COUNT(*) FROM Users"; SqlCommand command = new SqlCommand(query, connection); int userCount = (int)command.ExecuteScalar();
```

Return Type: Returns an object that you need to cast to the appropriate data type.

2. ExecuteReader - Executes a query and returns a SqlDataReader object to read the results row by row.

When to use it: When you need to retrieve multiple rows and columns from the database.

```
string query = "SELECT * FROM Users"; SqlCommand command = new SqlCommand(query, connection); SqlDataReader reader = command.ExecuteReader(); while (reader.Read()) { Console.WriteLine(reader["Username"].ToString()); } reader.Close();
```

Return Type: Returns a SqlDataReader object that provides a way to read the data sequentially.

3. ExecuteNonQuery - Executes a command that does not return any data.

When to use it: When performing operations such as INSERT, UPDATE, DELETE, or other DDL commands (e.g., CREATE TABLE).
string query = "INSERT INTO Users (Username, Password) VALUES ('user1', 'pass1')"; SqlCommand command = new SqlCommand(query, connection); int rowsAffected = command.ExecuteNonQuery();
Return Type: Returns an integer representing the number of rows affected by the command.

15. Complete the CheckCredentials method. Checking credentials requires the extra step of verifying the hash and salt values match the data stored in the database.

```
42      public int CheckCredentials(string username, string password)
43      {
44          using (SqlConnection connection = new SqlConnection(ConnectionString))
45          {
46              connection.Open();
47              // first find a user with the given username
48              string query = "SELECT * FROM Users WHERE Username = @Username";
49              SqlCommand command = new SqlCommand(query, connection);
50              command.Parameters.AddWithValue("@Username", username);
51              SqlDataReader reader = command.ExecuteReader();
52              // if the user is found, verify the password hash and return the user ID
53              if (reader.Read())
54              {
55                  UserModel user = new UserModel
56                  {
57                      Id = reader.GetInt32(reader.GetOrdinal("Id")),
58                      Username = reader.GetString(reader.GetOrdinal("Username")),
59                      PasswordHash = reader.GetString(reader.GetOrdinal("PasswordHash")),
60                      Salt = (byte[])reader["Salt"],
61                      Groups = reader.GetString(reader.GetOrdinal("Groups"))
62                  };
63                  bool valid = user.VerifyPassword(password);
64                  if (valid == true)
65                      return user.Id; // user found and password is correct
66                  else
67                      return 0;  // user found but password is incorrect
68              }
69              return 0; // user not found
70          }
71      }
```

Figure 45 CheckCredentials looks up a user record and verifies the password.

16. Complete the DeleteUser method.

```
74      public void DeleteUser(UserModel user)
75      {
76          // delete the matching user record
77          using (SqlConnection connection = new SqlConnection(ConnectionString))
78          {
79              connection.Open();
80              string query = "DELETE FROM Users WHERE Id = @Id";
81              SqlCommand command = new SqlCommand(query, connection);
82              command.Parameters.AddWithValue("@Id", user.Id);
83              command.ExecuteNonQuery();
84          }
85      }
```

Figure 46 Find the matching user record and execute a sql delete statement.

17. Complete the GetAllUsers method.

```
87      public List<UserModel> GetAllUsers()
88      {
89          // search for all users
90          List<UserModel> users = new List<UserModel>();
91          using (SqlConnection connection = new SqlConnection(ConnectionString))
92          {
93              connection.Open();
94              string query = "SELECT * FROM Users";
95              SqlCommand command = new SqlCommand(query, connection);
96              SqlDataReader reader = command.ExecuteReader();
97              while (reader.Read())
98              {
99                  UserModel user = new UserModel
100                 {
101                     Id = reader.GetInt32(reader.GetOrdinal("Id")),
102                     Username = reader.GetString(reader.GetOrdinal("Username")),
103                     PasswordHash = reader.GetString(reader.GetOrdinal("PasswordHash")),
104                     Salt = (byte[])reader["Salt"],
105                     Groups = reader.GetString(reader.GetOrdinal("Groups"))
106                 };
107                 users.Add(user);
108             }
109         }
110         return users;
111     }
```

Figure 47 GetAll users selects all records from the user table.

18. Complete the GetUserById method.

```
113         public UserModel GetUserById(int id)
114         {
115             // find the matching id number
116             using (SqlConnection connection = new SqlConnection(ConnectionString))
117             {
118                 connection.Open();
119                 string query = "SELECT * FROM Users WHERE Id = @Id";
120                 SqlCommand command = new SqlCommand(query, connection);
121                 command.Parameters.AddWithValue("@Id", id);
122                 SqlDataReader reader = command.ExecuteReader();
123                 while (reader.Read())
124                 {
125                     UserModel user = new UserModel
126                     {
127                         Id = reader.GetInt32(reader.GetOrdinal("Id")),
128                         Username = reader.GetString(reader.GetOrdinal("Username")),
129                         PasswordHash = reader.GetString(reader.GetOrdinal("PasswordHash")),
130                         Salt = (byte[])reader["Salt"],
131                         Groups = reader.GetString(reader.GetOrdinal("Groups"))
132                     };
133                     return user; // return the matching user
134                 }
135             }
136             return null; // no matching user found
137 
138         }
```

Figure 48 GetUserById returns a user if a matching id number is found.

19. Complete the UpdateUser method.

```
140         public void UpdateUser(UserModel user)
141         {
142             // find the matching user. if found, update the record using the new data
143             // id numbers do not change. all other fields can be updated.
144             int id = user.Id;
145             UserModel found = GetUserById(id);
146             if (found != null)
147             {
148                 using (SqlConnection connection = new SqlConnection(ConnectionString))
149                 {
150                     connection.Open();
151                     string query = @"UPDATE Users SET Username = @Username, PasswordHash =
                          @PasswordHash, Salt = @Salt, Groups = @Groups WHERE Id = @Id";
152                     SqlCommand command = new SqlCommand(query, connection);
153                     command.Parameters.AddWithValue("@Username", user.Username);
154                     command.Parameters.AddWithValue("@PasswordHash", user.PasswordHash);
155                     command.Parameters.AddWithValue("@Salt", user.Salt);
156                     command.Parameters.AddWithValue("@Id", user.Id);
157                     command.Parameters.AddWithValue("@Groups", user.Groups);
158                     command.ExecuteNonQuery();
159                 }
160             }
161         }
```

Figure 49 UpdateUser will replace the properties of a record. It will not replace the id number of a record.

20. In the UserController, replace the UserCollection class with the UsersDAO class. The instance name remains “users”.

```
using Microsoft.AspNetCore.Mvc;
using RegisterAndLoginApp.Models;
using RegisterAndLoginApp.Filters;

namespace RegisterAndLoginApp.Controllers
{
    public class UserController : Controller
    {
        // remove the UserCollection object and replace it with the UserDAO object
        //  static UserCollection users = new UserCollection();
        UserDAO users = new UserDAO();

          public IActionResult Index()
          {
              return View();
          }
```

Figure 50 Changing the UserManager class to the UserDAO class

21. Since the method names AddUser and CheckCredentials are the same in both the UsersDAO and the UserCollection classes, no further changes need to be made in the controller.

22. Run the application.

23. Register a new user.

24. Login with the new user credentials.

25. You should reach a login success screen.

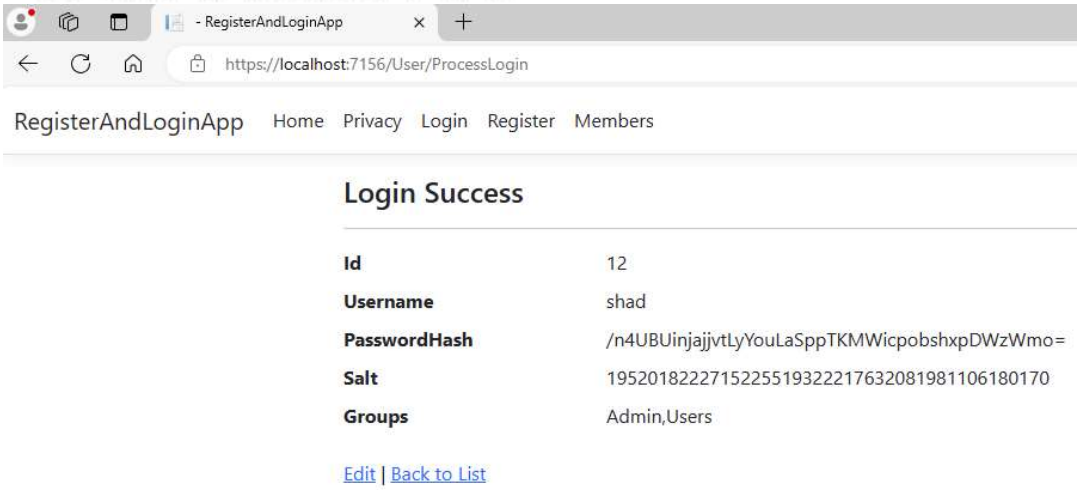


Figure 51 Login success with newly registered user.

- Take a screenshot of your application running at this point.
- Paste the image into a Word document.
- Put a caption below the image explaining what is being demonstrated.

26. You should be able to visit the Members Only page.

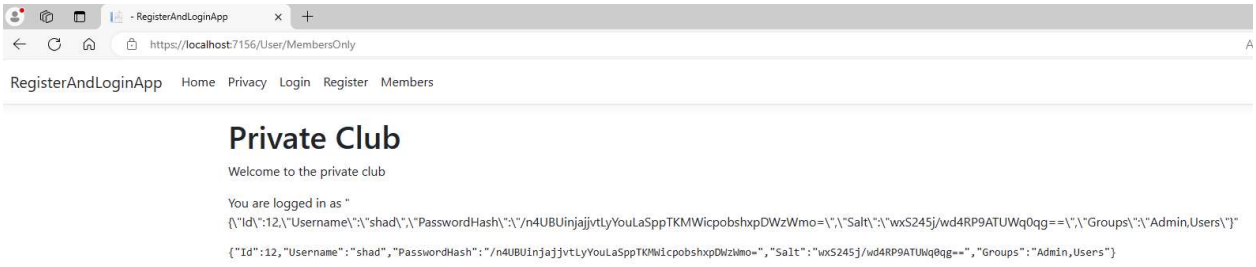


Figure 52 Members Only page shows the user information.

27. You should be able to see a new record in the users table.

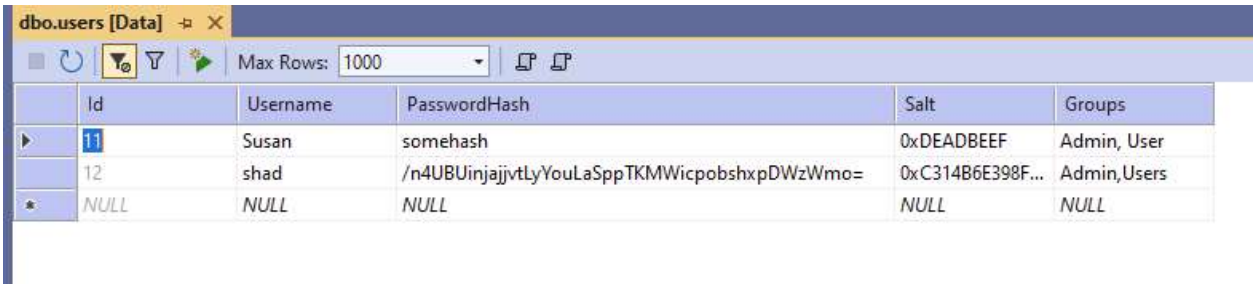
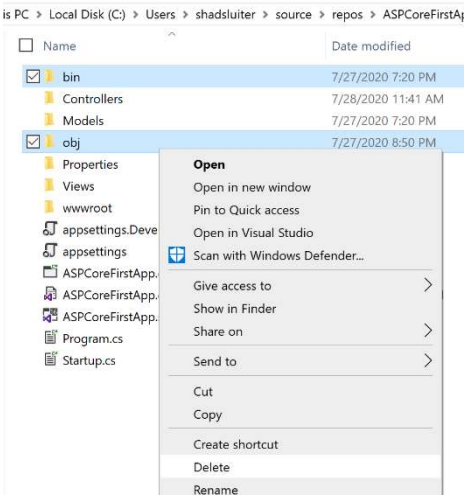


Figure 53 "shad" is a new record in the users table. Notice the hash value of Susan is clear text and would not be useful for authentication.

- Take a screenshot of your application running at this point.
- Paste the image into a Word document.
- Put a caption below the image explaining what is being demonstrated.

Deliverables:

1. This activity has multiple parts. Complete all parts before submitting.
2. Create a Microsoft Word document with screenshots of the application being run. Show each screen of the output and put a caption under each picture explaining what is being demonstrated.
3. In the same document, in one paragraph, write a summary of the key concepts that were demonstrated in this lesson. Be sure to explain the key words introduced in this lesson.
4. Turn the Word document into a PDF.



5. Submit a ZIP file of the project file. In order to save space, you can delete the bin and the obj folders of the project. These folders contain the compiled version of the application and are automatically regenerated each time the build or run commands are executed.
6. Attach the PDF separately from the zip file. Multiple files can be uploaded with an assignment.

Part 2 – User Group Access

Similar to the MembersOnly page, we are going to create a restriction on the Admin users. Only users who are members of the Admin group will be able to view this page.

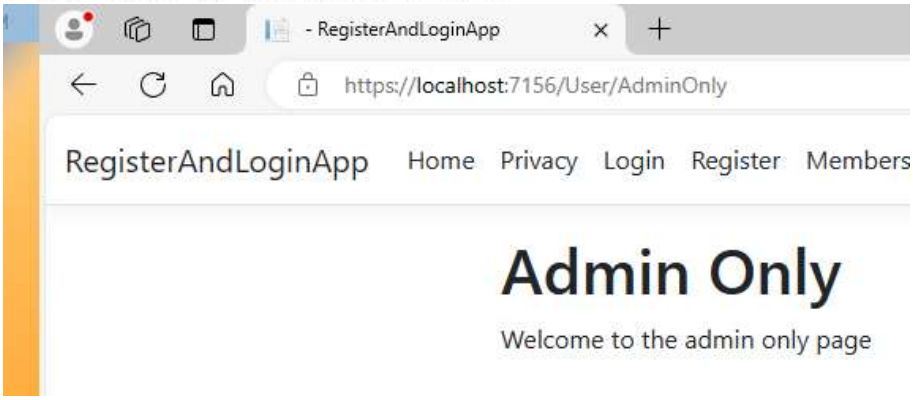


Figure 54 Only administrator accounts will be able to visit this page.

1. Create a new filter AdminCheckFilter

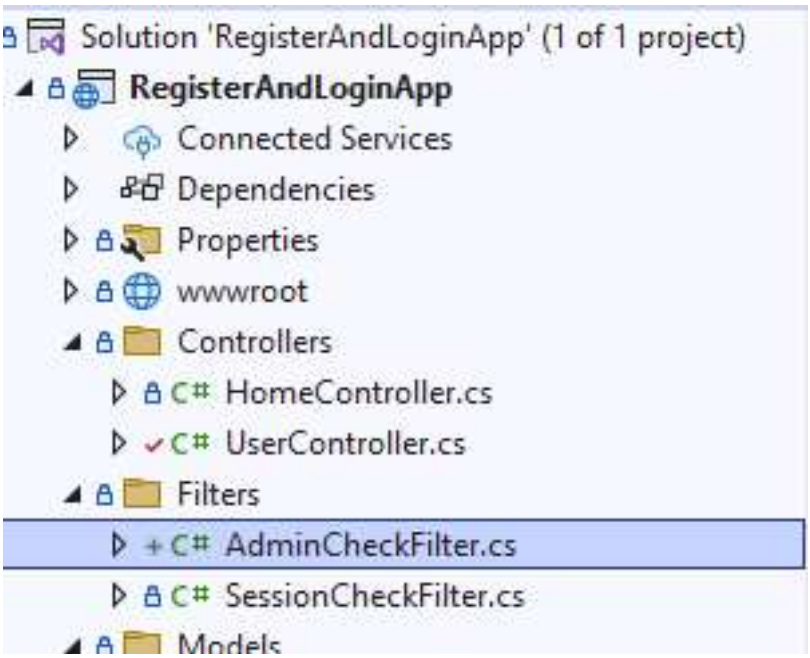


Figure 55 Location of the AdminCheckFilter

2. Check to see if the logged in user is part of the Admin team.

```
namespace RegisterAndLoginApp.Filters
{
    public class AdminCheckFilter : ActionFilterAttribute
    {
            public override void OnActionExecuting(ActionExecutingContext context)
            {
            string userInfo = context.HttpContext.Session.GetString("User");
            if (userInfo == null) {
                // no user info in session. Redirect to login page
                context.Result = new RedirectResult("/User/Index");
                return;
            }
            // part 2 - check to see if the user is an admin
            try {
                UserModel user = ServiceStack.Text.JsonSerializer.DeserializeFromString<UserModel>
                  (userInfo);
                if (user.Groups.Contains("Admin") == false)
                {
                    // user is not an admin. Redirect to login page
                    context.Result = new RedirectResult("/User/Index");
                    return;
                }
            } catch {
                // error in deserialization of the session information. Redirect to login page
                context.Result = new RedirectResult("/User/Index");
                return;
            }
        }
    }
}
```

Figure 56 Checking to see if valid user is logged in and if the user is admin

3. Create an AdminOnly.cshtml file

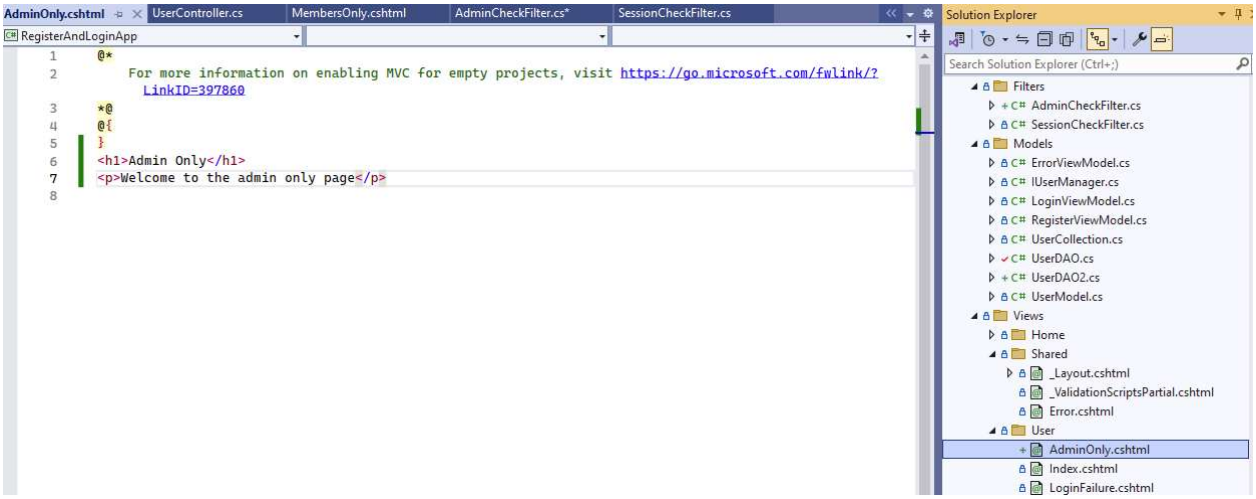


Figure 57 Content of AdminOnly page

4. In the UserController, create a new method to direct the users to the AdminOnly page using the filter.

```
35      [SessionCheckFilter]
36      public IActionResult MembersOnly()
37      {
38          return View();
39      }
40      [AdminCheckFilter]
41      public IActionResult AdminOnly()
42      {
43          return View();
44      }
45
46      public IActionResult Register()
47      {
48          return View(new RegisterViewModel());
49      }
50
```

Figure 58 AdminOnly method uses the AdminCheckFilter.

5. Run the program and verify that admin users can see the AdminOnly page, other groups cannot see the Admin page but can see the MembersOnly page.

- Take a screenshot of your application running at this point.
- Paste the image into a Word document.
- Put a caption below the image explaining what is being demonstrated.

Conclusions

What You Learned in This Lesson...

In this lesson, you learned the fundamental steps to set up and manage a user database for full-stack applications using SQL database servers. This lesson was designed to give you hands-on experience with configuring SQL Server on both Windows and Macintosh systems, and to guide you through creating and managing a user database.

Key Ideas

1. **Implementing a User Database**:
    - Creating tables and performing basic CRUD (Create, Read, Update, Delete) operations.
    - Writing SQL queries and understanding the differences between T-SQL and standard SQL.
2. **Connecting C# Applications to SQL Databases**:
    - Establishing database connections using appropriate connection strings.
    - Implementing ADO.NET to interact with the database from a C# application.
    - Managing user data effectively within the application.

Features of the Applications Created

1. **Database Setup**:
    - Configured a SQL Server database for both Windows and macOS environments.
    - Created a user table with fields for user ID, username, password hash, salt, and user groups.
2. **User Management**:
    - Implemented a UsersDAO class to handle database operations such as adding, deleting, updating, and fetching user records.
    - Key ADO.NET commands include **ExecuteScalar**, **ExecuteReader**, and **ExecuteNonQuery**.
    - Developed methods for user authentication, including password hashing and verification.
3. **Application Integration**:
    - Connected the database to a C# application.
    - Implemented user login, registration, and access control features.
    - Utilized session variables to maintain user login status.

Key Takeaways

- **SQL and T-SQL**: T-SQL (Transact-SQL) is an extension of SQL specific to Microsoft SQL Server. It includes procedural programming features and is used for managing and manipulating data in SQL Server.
- **Containers and Docker**: Containers allow applications to run on different platforms by encapsulating all necessary dependencies. Docker is a tool for managing containers, essential for running Azure SQL Edge on macOS.
- **Separation of Concerns (SoC)**: This principle promotes dividing an application into distinct sections, each addressing a specific concern. For databases, this means keeping data storage and retrieval separate from business logic, which resides in the application layer.
- **Session Management**: Using session variables in web applications to maintain user state across multiple requests. This is crucial for implementing login features and access control.

Terminology and Techniques to Remember

- **Docker**: A platform for developing, shipping, and running applications inside containers.
- **ADO.NET**: A data access technology used in .NET to interact with databases.

- **Connection String**: A string used to specify information about a data source and how to connect to it.
- **CRUD Operations**: Basic operations to Create, Read, Update, and Delete data in a database.
- **T-SQL**: Transact-SQL, an extension of SQL used in Microsoft SQL Server for procedural programming.
- **Salt and Hash**: Techniques used in cryptography to securely store passwords.

These concepts and tools are foundational for building robust and scalable web applications. Understanding and applying them will enhance your ability to develop full-stack applications effectively. Make sure to review these key points as they will likely feature in upcoming quizzes and assessments.

Check for Understanding

These questions are not graded but will give you a good idea of assessments to come.

1. **What is the primary focus of this lesson?**
   A. Setting up a user interface
   B. Setting up a user database for full-stack applications
   C. Learning advanced JavaScript
   D. Configuring network settings

2. **Which two options are suggested for running a database with Visual Studio on a Macintosh?**
   A. PostgreSQL and SQLite
   B. MySQL on MAMP and Azure SQL Edge
   C. Oracle and Microsoft Access
   D. MongoDB and Cassandra

3. **Why can't the standard SQL Server run on Windows for ARM64 on a Macintosh processor?**
   A. It requires an x64 CPU
   B. It is incompatible with macOS
   C. It requires specific network settings
   D. It needs additional RAM

4. **Which Docker command is used to pull the latest Azure SQL Edge image?**
   A. docker run mcr.microsoft.com/azure-sql-edge
   B. docker install azure-sql-edge
   C. docker pull mcr.microsoft.com/azure-sql-edge
   D. docker start azure-sql-edge

5. **What information is needed to configure a new SQL Server connection in Visual Studio?**
   A. Server name, authentication method, user name, password
   B. Database type, IP address, port number

C. Operating system version, network type
D. Application name, container ID

6. What is the purpose of the ExecuteNonQuery method in ADO.NET?
A. To execute a command that returns multiple rows of data
B. To execute a command that does not return any data
C. To execute a query and return a single value
D. To read data sequentially from the database

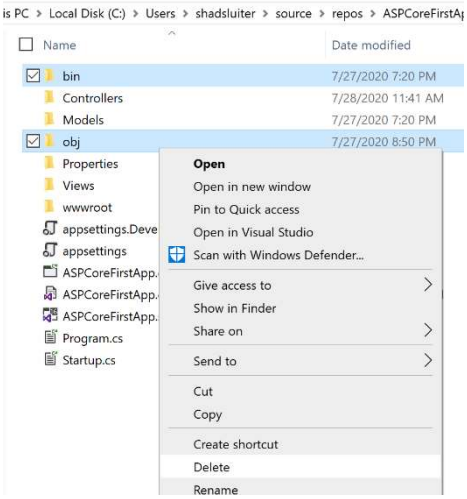
7. What does the term 'Salt' refer to in the context of password hashing?
A. A type of encryption algorithm
B. A random value added to the password before hashing
C. A database connection string
D. A hashing function

8. Which of the following statements about containers is true?
A. Containers can only run on Linux operating systems
B. Containers are similar to virtual machines but are more lightweight
C. Containers require extensive hardware resources
D. Containers cannot run applications that were not originally designed for them

9. What does 'CRUD' stand for in database operations?
A. Create, Read, Update, Delete
B. Connect, Retrieve, Update, Deploy
C. Configure, Run, Upload, Download
D. Compute, Render, Utilize, Delete

10. Why is the principle of Separation of Concerns important in database design?
A. It ensures that the database code is faster
B. It keeps the business logic within the database layer
C. It promotes dividing a program into distinct sections, each addressing a separate concern
D. It reduces the number of database queries

Deliverables:



1. This activity has multiple parts. Complete all parts before submitting.
2. Create a Microsoft Word document with screenshots of the application being run. Show each screen of the output and put a caption under each picture explaining what is being demonstrated.
3. In the same document, in one paragraph, write a summary of the key concepts that were demonstrated in this lesson. Be sure to explain the key words introduced in this lesson.
4. Turn the Word document into a PDF.
5. Submit a ZIP file of the project file. In order to save space, you can delete the bin and the obj folders of the project. These folders contain the compiled version of the application and are automatically regenerated each time the build or run commands are executed.
6. Attach the PDF separately from the zip file. Multiple files can be uploaded with an assignment.

Check for Understanding Answers:

1) B
2) B
3) A
4) C
5) A
6) B
7) B
8) B
9) A
10) C