Database-First Web Application in ASP.NET Core MVC

using EntityFrameworkCore

including

4 ways to load CSV file into SQL Server table

AND

Calling Stored Procedures

(SELECT and INSERT - with parameters)

in website via Entity Framework

n.b. Depends on the data in the file and whether an Identity column is required (or not) in the database table.

OVERALL SUMMARY

*CSV LOADING*

(1) Load data into a table as is (i.e. no identify column) using SSMS. It assumes the table does not exist beforehand

(2) Load data into table (assuming you WANT to treat one column as identity column)

Problem is you can't change a column to an identity column once data loaded.

Hence, need to load a a normal table (set primary key) and then copy into table with identity column

(3) Load data via BULK INSERT (for identity, you'd need to prefix CSV with a comma)

That way, the identity column will be incremented automatically.

(4) load file via BCP (Bulk Copy Program)

*Web Application - Database First - using Scaffolding*

(1) Install 3 EntityFrameworkCore packages

(2) Use 'Scaffold' to create models based on tables in the database (n.b. procs do not get included)

This will allso create a ProjectDbContext with DbSet's mapped to your tables

(3) Create reference to your ProjectDbContext in controller (not best place, but quick and easy)

        private johntestContext \_context = new johntestContext();

Now, you can access the tables and data using code like the following:

var myList = new List<Test>();

            myList = \_context.Test.ToList();

            return View(myList);

(4) Call SELECT procedure using the following:

            var test = new List<Test>();

            string myName = "steve";

            test = \_context.Test

                    .FromSqlInterpolated($"sp\_getdetails @name={myName}")

                    .ToList();

            return View(test);

(5) Call INSERT procedure using the following:

using Microsoft.Data.SqlClient;

           var myName = new SqlParameter("@name", "Neil");

           var myPhone = new SqlParameter("@phone", 876);

           \_context.Database.ExecuteSqlRaw("dbo.sp\_insertdetails @name, @phone", myName, myPhone);

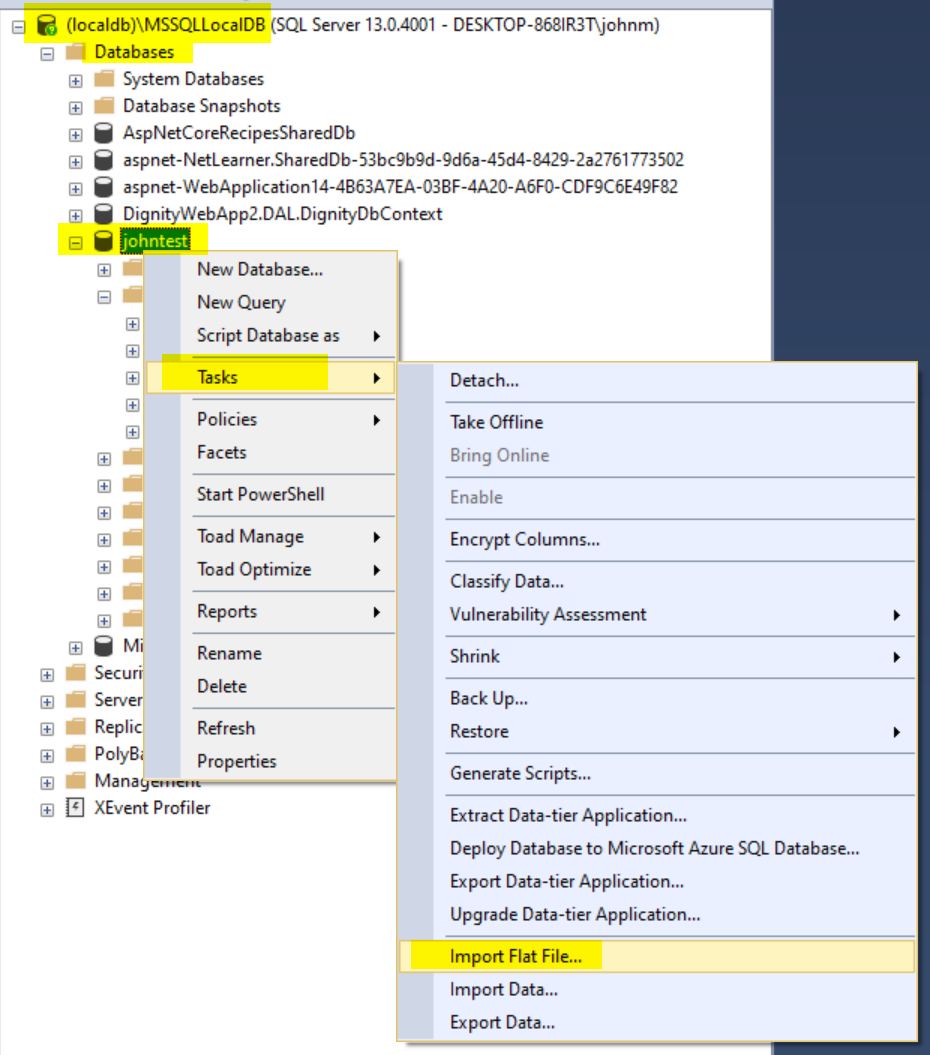
(1) Load data into a table as is (i.e. no identify column) using SSMS. It assumes the table does not exist beforehand. I named database 'johntest'.

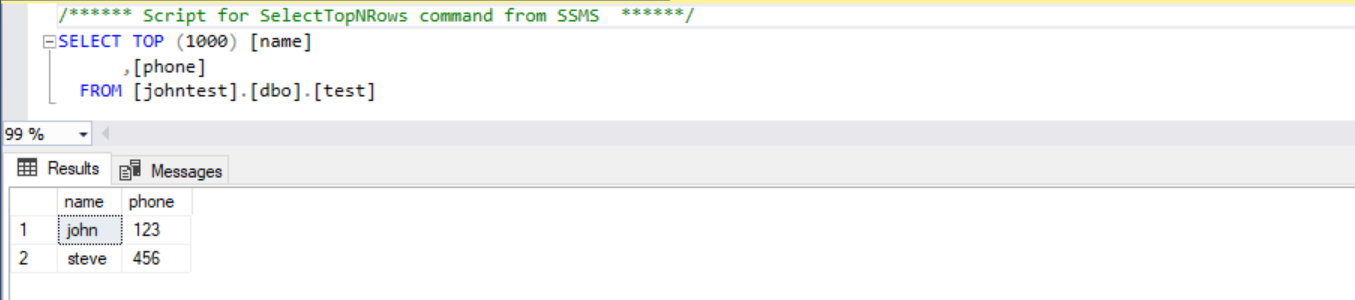
test.csv

name,phone

john,123

steve,456





now, create a table with an Identity column

create table test2

(Id int identity(1,1) NOT NULL,

name varchar(50),

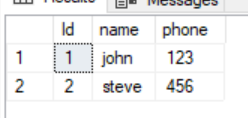
phone int );

and select the data into the table

insert into test2

(name, phone)

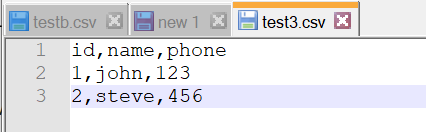
select name, phone from test;

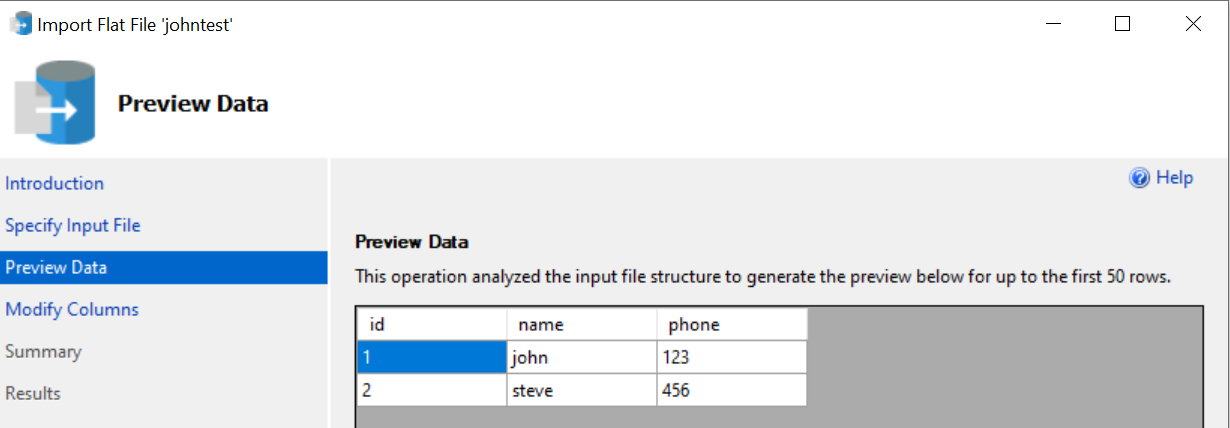


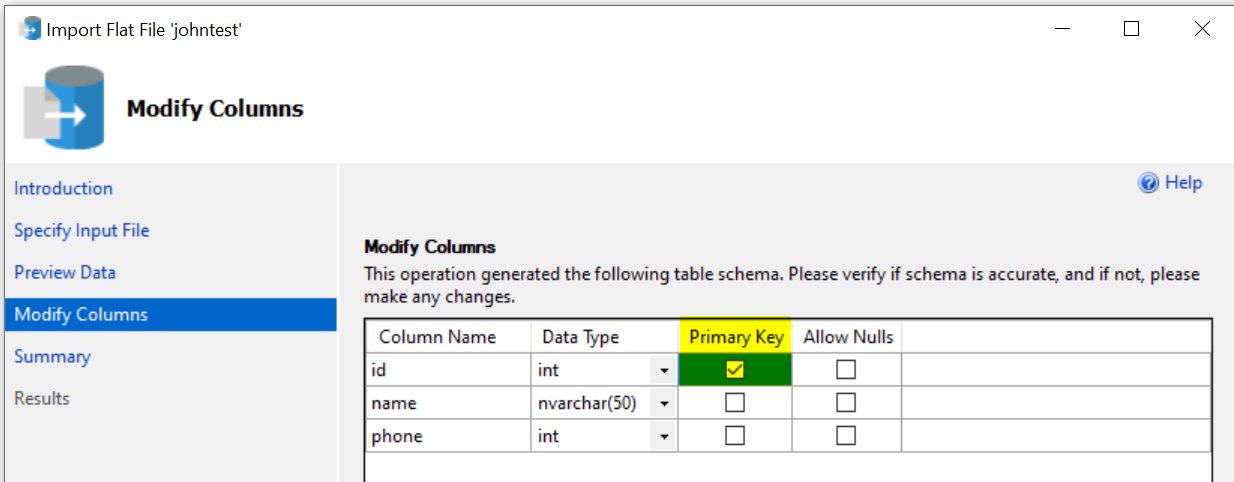
(2) Load data into table (assuming you WANT to treat one column as identity column)

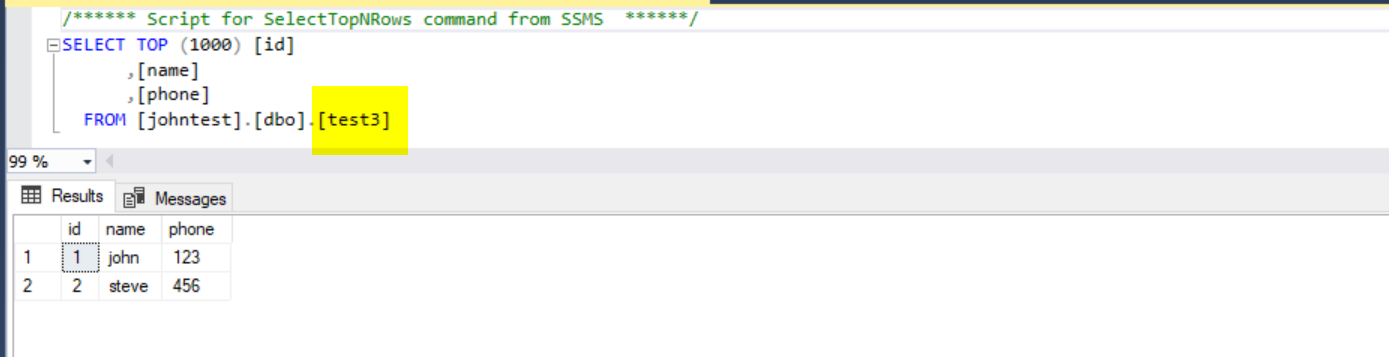
Problem is you can't change a column to an identity column once data loaded.

So, you can only load data as you did in (1) and set the primary key as you do it.









(3) Load data via BULK INSERT (for identity, you'd need to prefix CSV with a comma)

That way, the identity column will be incremented automatically.

create table test5

(Id int identity(1,1) NOT NULL,

name varchar(50),

phone int );

bulk insert dbo.test5

from 'C:\Users\johnm\Downloads\test.csv'

WITH

(

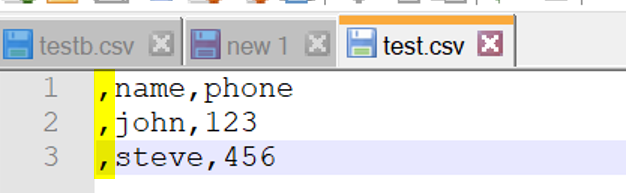
FIRSTROW = 2,

FIELDTERMINATOR = ',',

ROWTERMINATOR = '\n'

);

example file: test.csv (with leading comma - required beacuse of identity column in table)

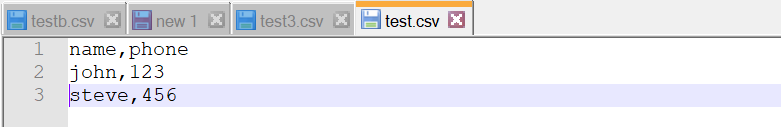


(4) load file via BCP (Bulk Copy Program)

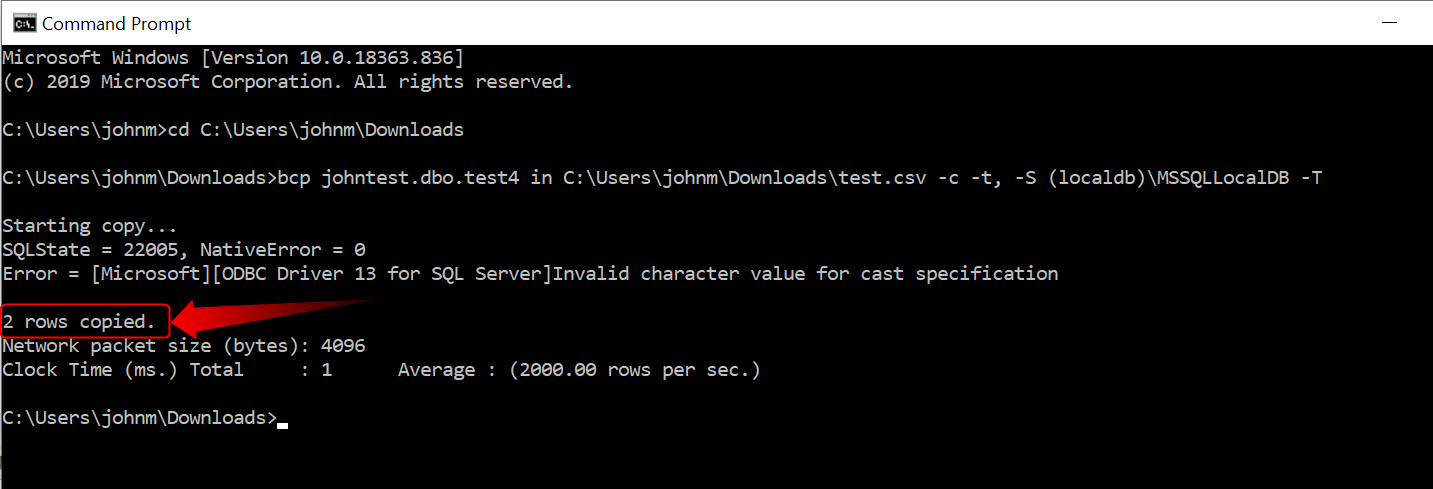
create table test4

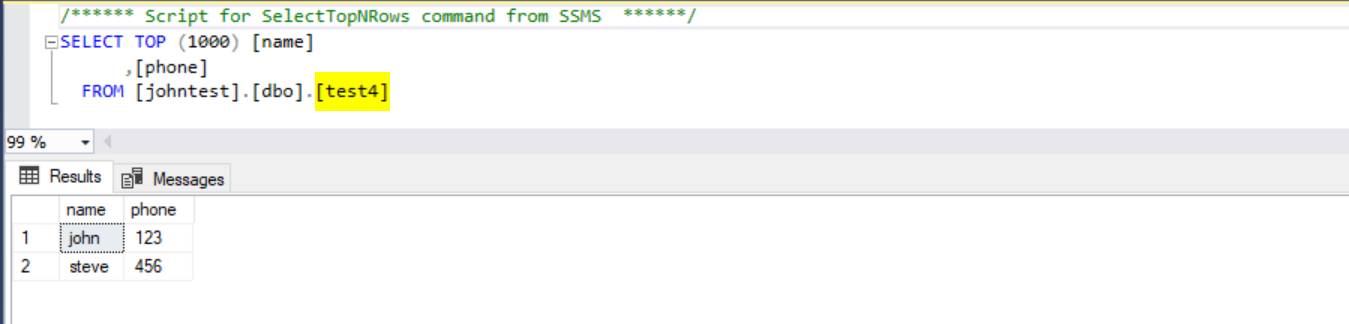
(name varchar(50),

phone int );

****

bcp johntest.dbo.test4 in C:\Users\johnm\Downloads\test.csv -c -t, -S (localdb)\MSSQLLocalDB -T





Further information about loading CSV files into table:

Here is the sample CSV with header:

Name,Class,Subject,ExamDate,Mark,Description

Prabhat,4,Math,2/10/2013,25,Test data for prabhat.

Murari,5,Science,2/11/2013,24,"Test data for his's test, where we can test 2nd ROW, Test."

sanjay,4,Science,,25,Test Only.

And SQL statement to import:

BULK INSERT SchoolsTemp

FROM 'C:\CSVData\Schools.csv'

WITH

(

FIRSTROW = 2,

FIELDTERMINATOR = ',', --CSV field delimiter

ROWTERMINATOR = '\n', --Use to shift the control to next row

TABLOCK

)

bulk insert dbo.test2

from 'C:\Users\johnm\Downloads\test.csv'

WITH

(

KEEPIDENTITY,

FIELDTERMINATOR = ',',

ROWTERMINATOR = '\n'

);

Create a stored procedure to get data from the table (SELECT)

CREATE PROCEDURE sp\_getdetails

(@name as varchar(50))

AS

BEGIN

SELECT name

,phone

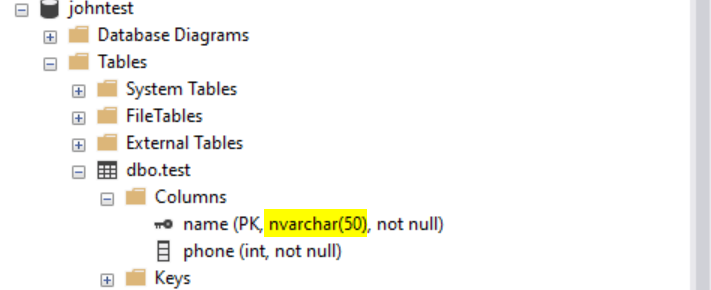
FROM test

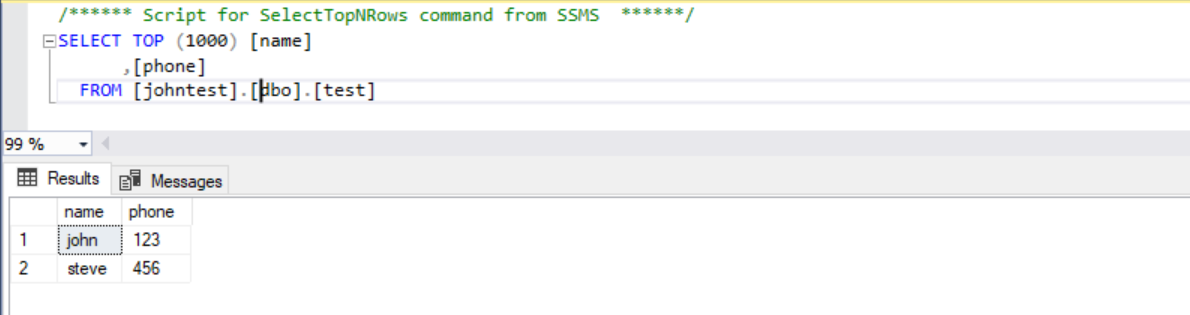
WHERE name = @name;

END;

n.b. the procedure was setup as varchar(50), yet the underlying table was nvarchar

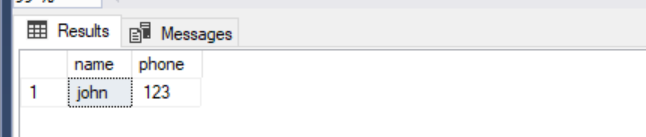
but doesn't seem to matter when checking data

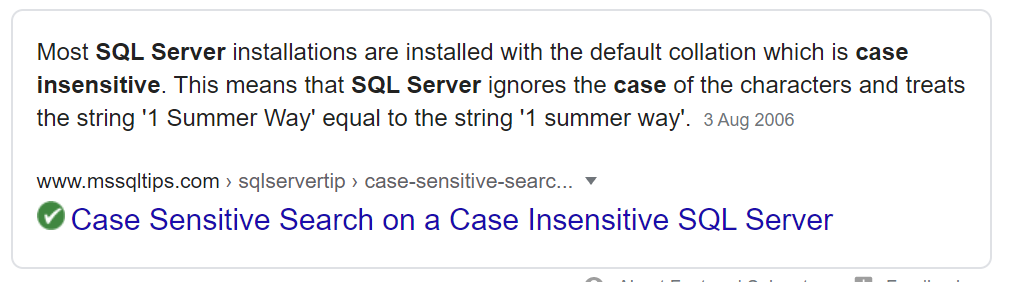




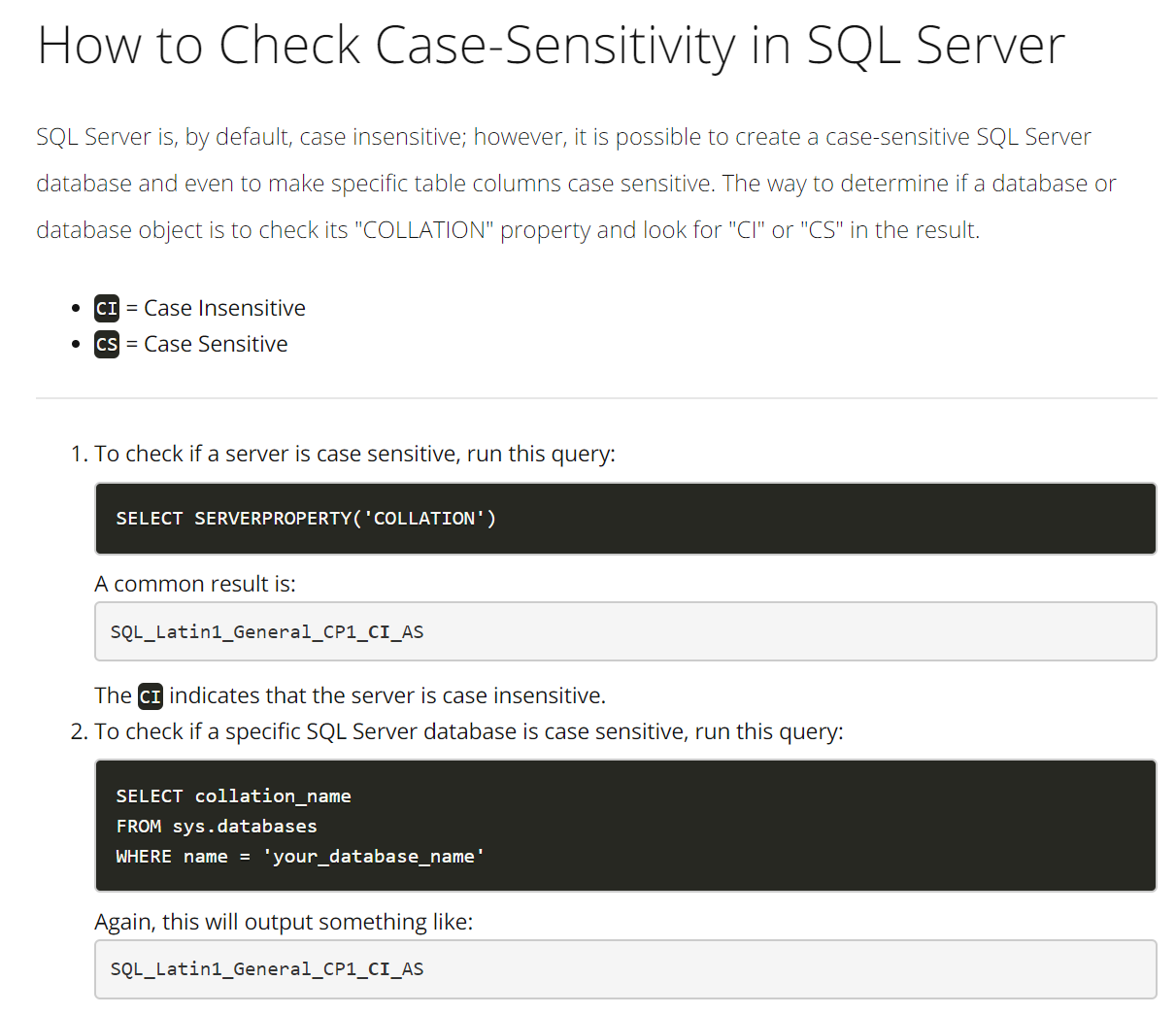
n.b the data is case-insensitive in SQL Server by default;

exec sp\_getdetails @name = 'john';





SELECT SERVERPROPERTY('COLLATION')



Create a stored procedure to perform an INSERT

CREATE PROCEDURE sp\_insertdetails

(@name as varchar(50)

,@phone as int)

AS

BEGIN

INSERT INTO test

(name, phone)

VALUES (@name, @phone);

--COMMIT; don't need a commit unless

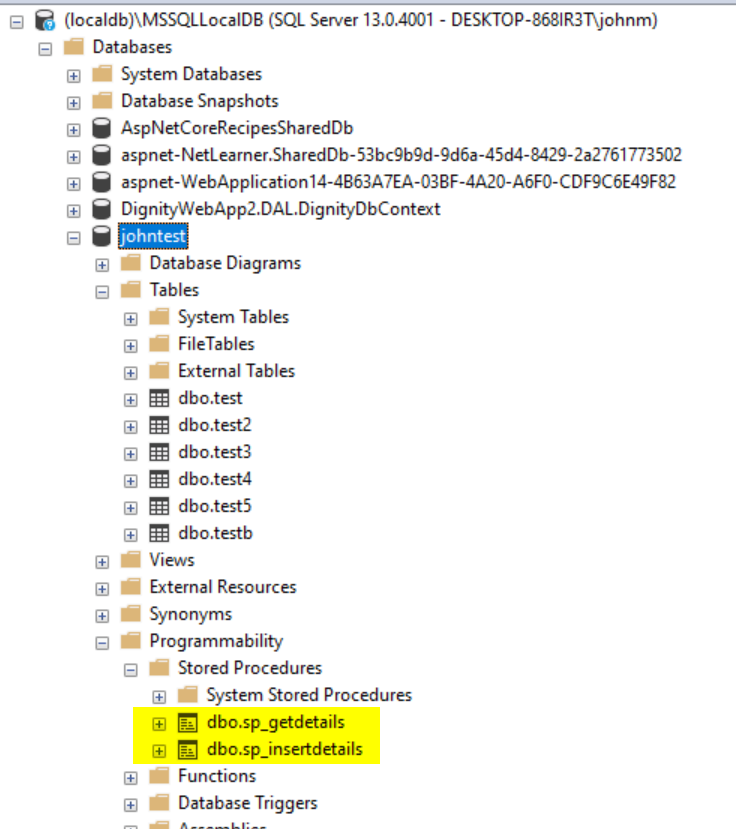
-- you use BEGIN TRANSACTION

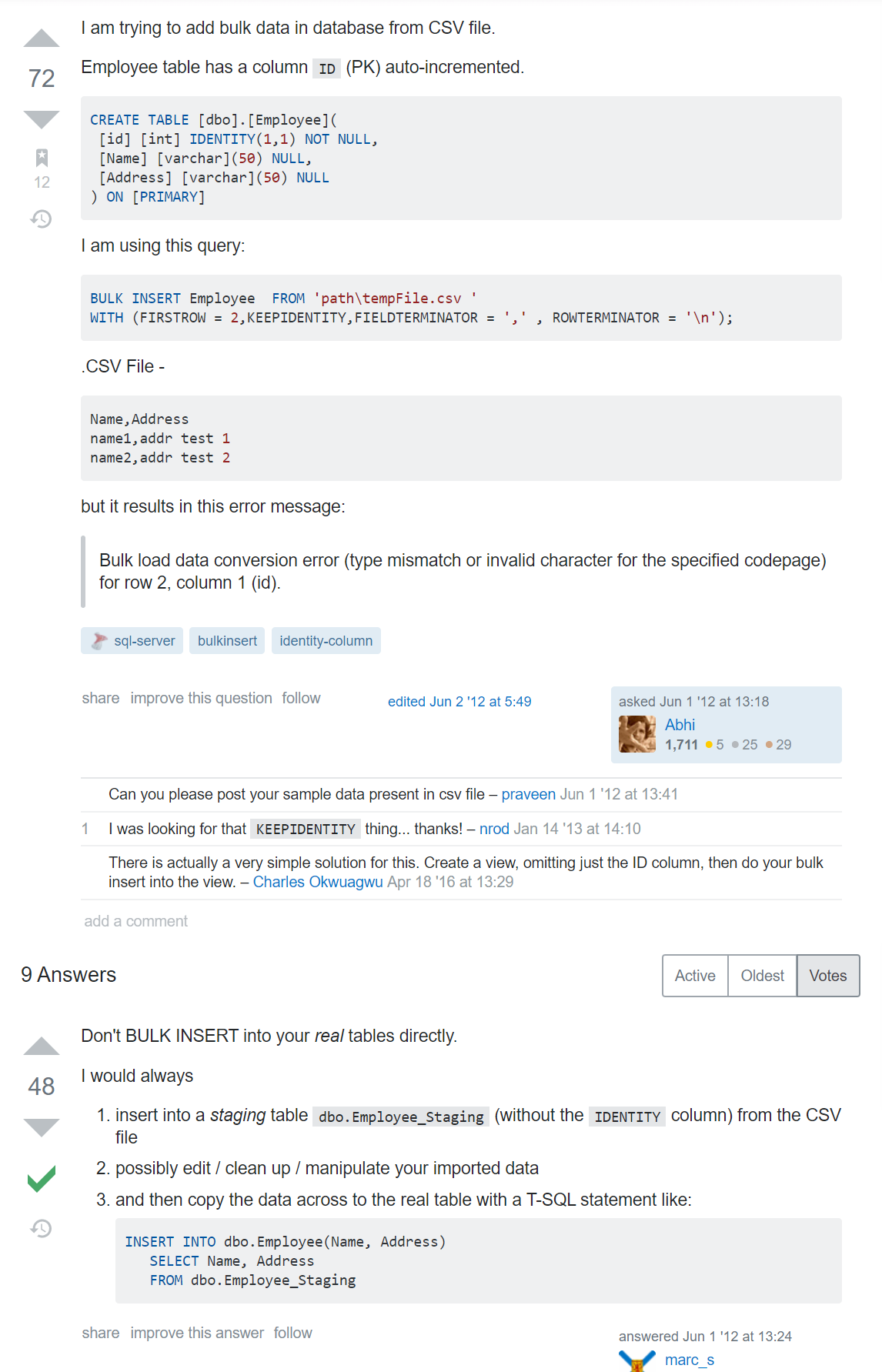
END;

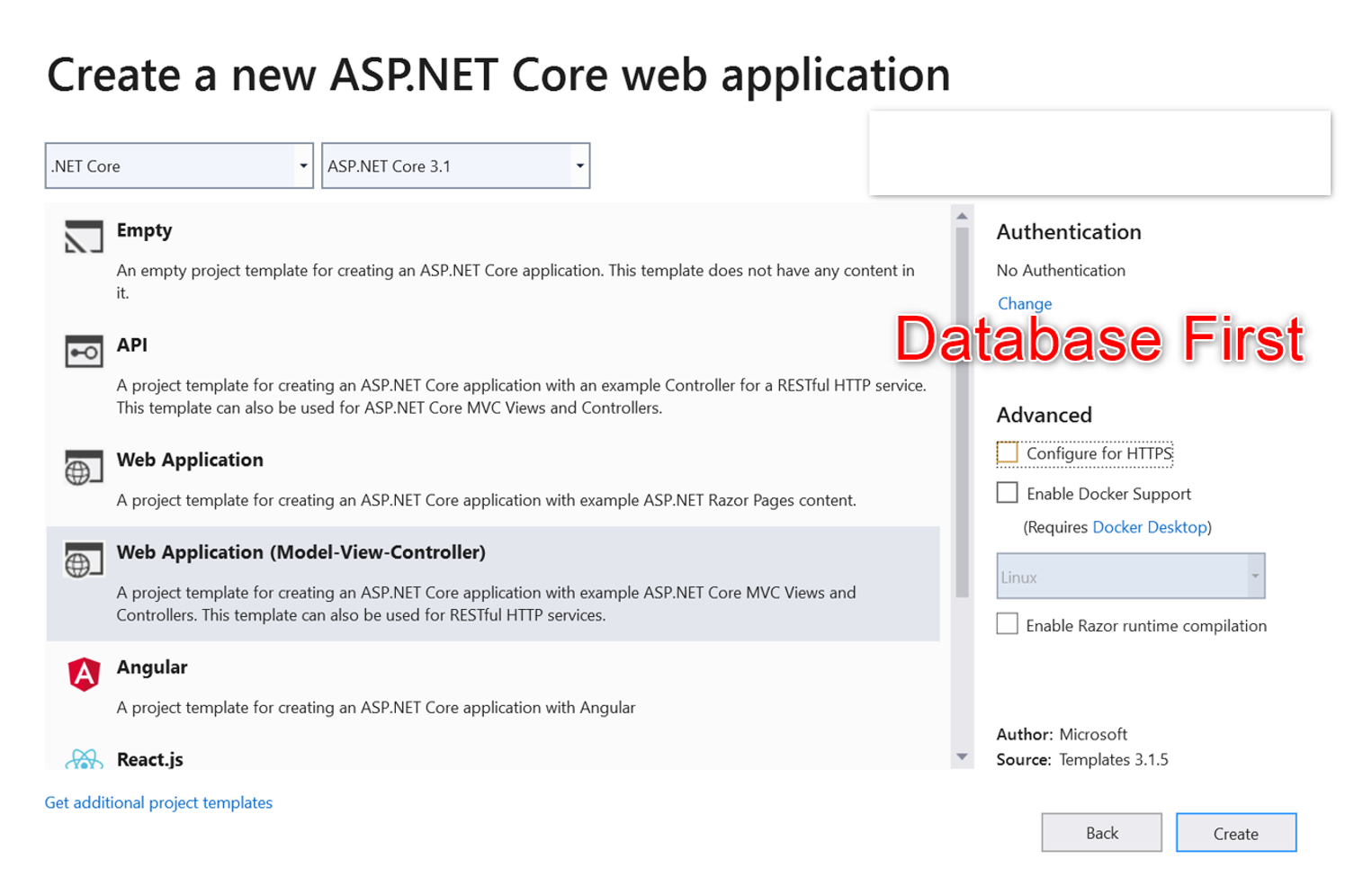
exec sp\_insertdetails

@name = 'Peter'

,@phone = 012;

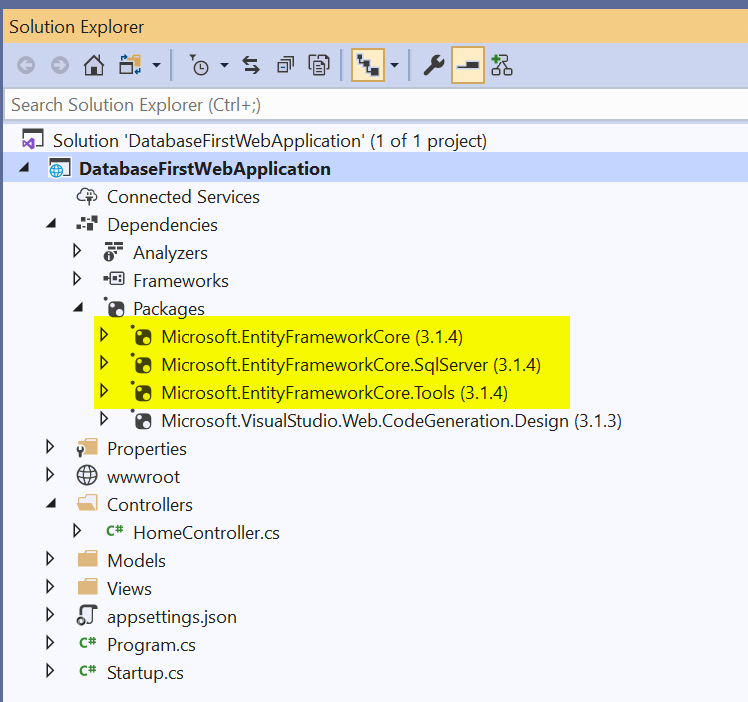






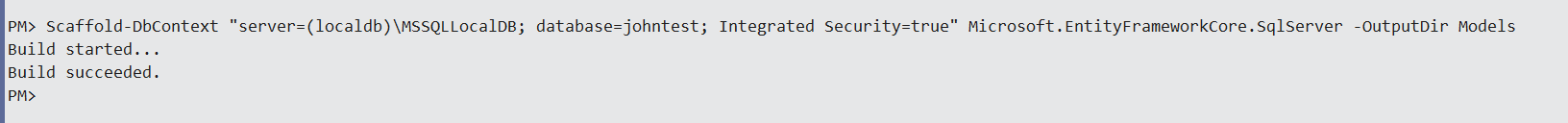
(A) Using **Scaffold**

(1) The Web Application (Model-View-Controller) project already has **EntityFrameworkCore** in the Framework **Microsoft.AspNetCore.App** (built-in), but it is best to get the latest versions

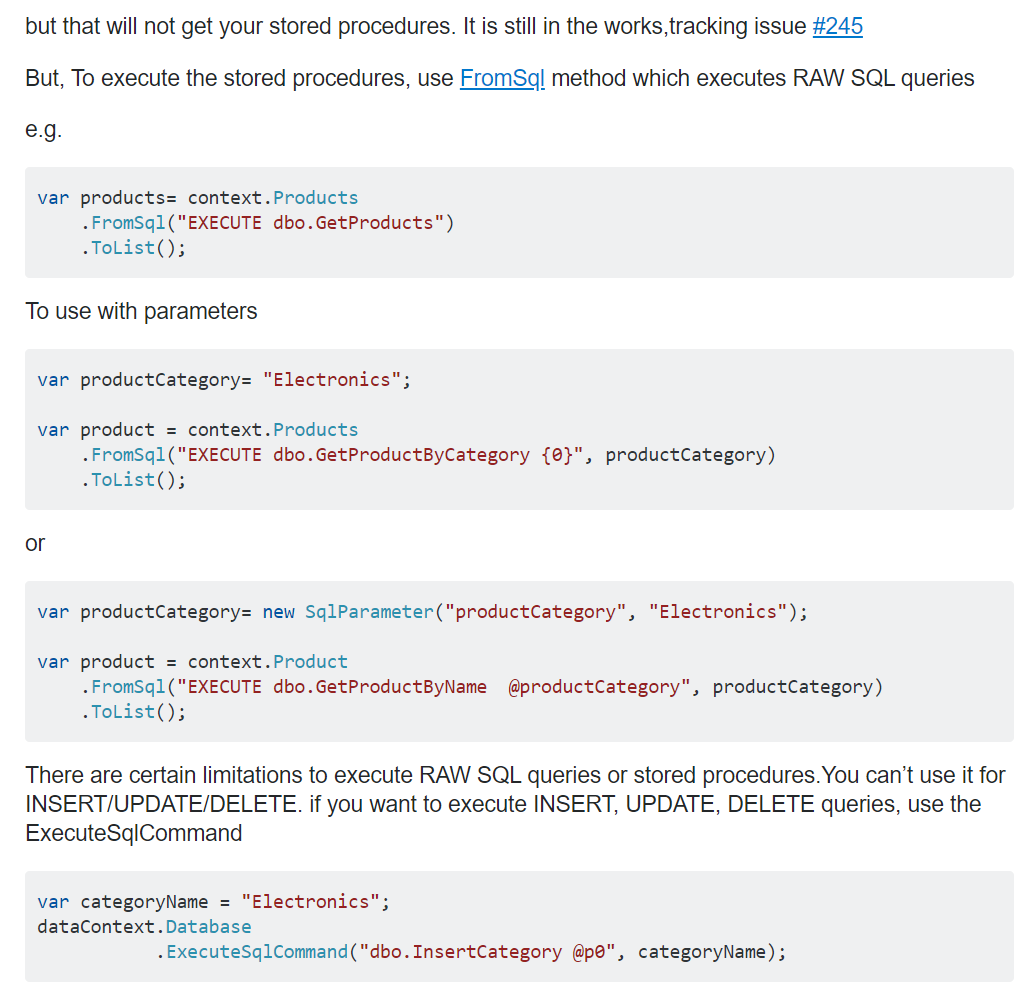


(2) Now, do **scaffolding** - open up package manager console (must be done all on one line)

PM> Scaffold-DbContext "server=(localdb)\MSSQLLocalDB; database=johntest; Integrated Security=true" Microsoft.EntityFrameworkCore.SqlServer -OutputDir Models



**n.b. this will NOT generate code for your stored procedures (To access stored procedures, you need to do the following)**



<https://www.learnentityframeworkcore.com/raw-sql>

You can use FromRawSql when you just want to run a query as opposed to a procedure

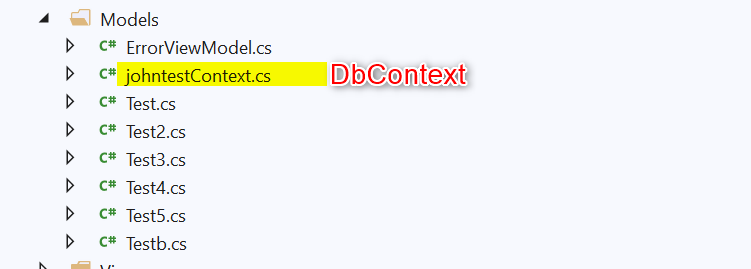
1. using (var context = new SampleContext())
2. {
3. var books = context.Books.FromSqlRaw("SELECT BookId, Title, AuthorId, Isbn FROM Books").ToList();
4. }

Parameterized Queries link

You are always advised to parameterize user input to prevent the possibility of a SQL injection attack being successful. Entity Framework Core will parameterize SQL if you use format strings or string interpolation:

1. // Format string
2. var author = db.Authors.FromSqlRaw("SELECT \* From Authors Where AuthorId = {0}", id).FirstOrDefault();
3. // String interpolation
4. var author = db.Authors.FromSqlRaw($"SELECT \* From Authors Where AuthorId = {id}").FirstOrDefault();

Once scaffolded, the Models folder will look like the following



ConnectionString present in Context file - best to move to appsettings.json (see others - and come back to this)

using System;

using Microsoft.EntityFrameworkCore;

using Microsoft.EntityFrameworkCore.Metadata;

namespace DatabaseFirstWebApplication.Models

{

    public partial class johntestContext : DbContext

    {

        public johntestContext()

        {

        }

        public johntestContext(DbContextOptions<johntestContext> options)

            : base(options)

        {

        }

        public virtual DbSet<Test> Test { get; set; }

        public virtual DbSet<Test2> Test2 { get; set; }

        public virtual DbSet<Test3> Test3 { get; set; }

        public virtual DbSet<Test4> Test4 { get; set; }

        public virtual DbSet<Test5> Test5 { get; set; }

        public virtual DbSet<Testb> Testb { get; set; }

        protected override void OnConfiguring(DbContextOptionsBuilder optionsBuilder)

        {

            if (!optionsBuilder.IsConfigured)

            {

#warning To protect potentially sensitive information in your connection string, you should move it out of source code. See http://go.microsoft.com/fwlink/?LinkId=723263 for guidance on storing connection strings.

                optionsBuilder.UseSqlServer("server=(localdb)\\MSSQLLocalDB; database=johntest; Integrated Security=true");

            }

        }

        protected override void OnModelCreating(ModelBuilder modelBuilder)

        {

Now, just to get basic data displayed to screen , we'll add the context to the controller directly. This is NOT good practice, but just want to see that data appears here.

public class HomeController : Controller

    {

        private readonly ILogger<HomeController> \_logger;

        // shouldn't read hardcode, but just for a test

        private johntestContext \_context = new johntestContext();

        public HomeController(ILogger<HomeController> logger)

        {

            \_logger = logger;

        }

        //public IActionResult Index()

        public IActionResult Index()

        {

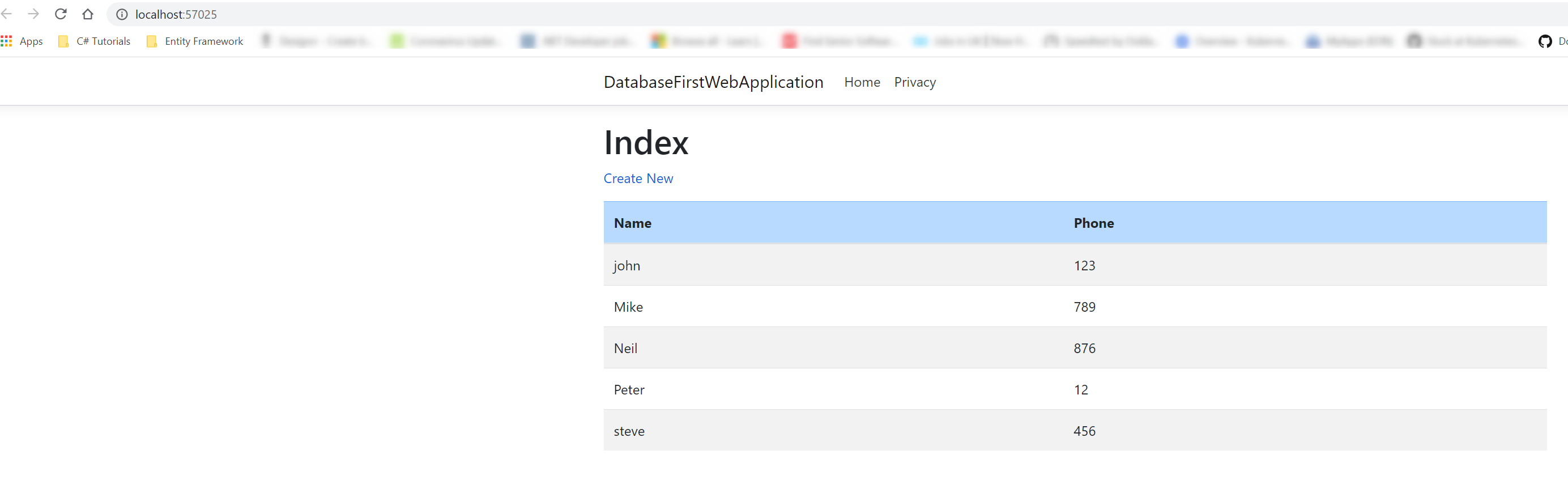
            var myList = new List<Test>();

            myList = \_context.Test.ToList();

            return View(myList);

        }

Had to create Index view (List view - but then needed amending since it commented out the Name column (since it thought it was primary key like an Idenity column)



Index.cshtml (View)

@model IEnumerable<DatabaseFirstWebApplication.Models.Test>

@{

    ViewData["Title"] = "Index";

}

<h1>Index</h1>

<p>

    <a **asp-action**="Create">Create New</a>

</p>

<table class="table table-striped ">

    <thead class=table-primary>

        <tr>

            <th>

                @Html.DisplayNameFor(model => model.Name)

            </th>

            <th>

                @Html.DisplayNameFor(model => model.Phone)

            </th>

            @\*<th></th>\*@

        </tr>

    </thead>

    <tbody>

@foreach (var item in Model) {

        <tr>

            <td>

                @Html.DisplayFor(modelItem => item.Name)

            </td>

            <td>

                @Html.DisplayFor(modelItem => item.Phone)

            </td>

            @\*<td>

                <a asp-action="Edit" asp-route-id="@item.Name">Edit</a> |

                <a asp-action="Details" asp-route-id="@item.Name">Details</a> |

                <a asp-action="Delete" asp-route-id="@item.Name">Delete</a>

            </td>\*@

        </tr>

}

    </tbody>

</table>

To get the results from a stored procedure (**SELECT**), do the following:

public class HomeController : Controller

    {

        private readonly ILogger<HomeController> \_logger;

        // shouldn't read hardcode, but just for a test

        private johntestContext \_context = new johntestContext();

        public HomeController(ILogger<HomeController> logger)

        {

            \_logger = logger;

        }

        //public IActionResult Index()

        public IActionResult Index()

        {

            /\*

            var myList = new List<Test>();

            myList = \_context.Test.ToList();

            return View(myList);

            \*/

            var test = new List<Test>();

            string myName = "steve";

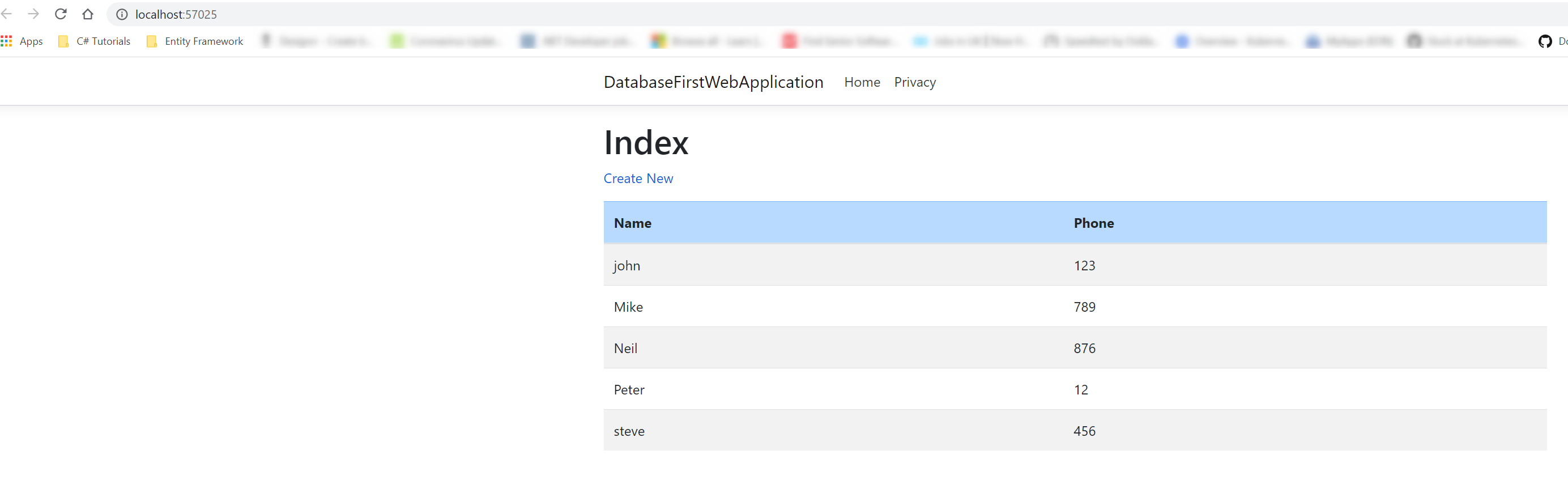
            test = \_context.Test

                    .FromSqlInterpolated($"sp\_getdetails @name={myName}")

                    .ToList();

            return View(test);

        }



To get the results from a stored procedure (**INSERT**), do the following:

public class HomeController : Controller

   {

       private readonly ILogger<HomeController> \_logger;

       // shouldn't read hardcode, but just for a test

       private johntestContext \_context = new johntestContext();

       public HomeController(ILogger<HomeController> logger)

       {

           \_logger = logger;

       }

       //public IActionResult Index()

       public IActionResult Index()

       {

           var myList = new List<Test>();

           myList = \_context.Test.ToList();

           //return View(myList);

           var test = new List<Test>();

           /\*

           test = \_context.Test

                   .FromSqlInterpolated($"sp\_getdetails @name={myName}")

                   .ToList();

           \*/

           var myName = new SqlParameter("@name", "Neil");

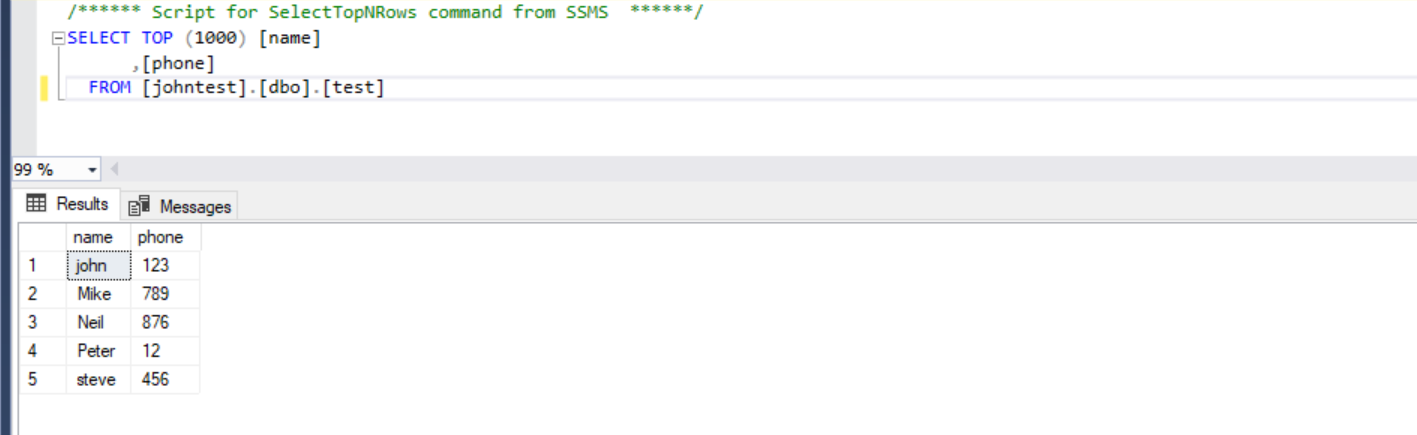
           var myPhone = new SqlParameter("@phone", 876);

           \_context.Database.ExecuteSqlRaw("dbo.sp\_insertdetails @name, @phone", myName, myPhone);

           // n.b. used ExecuteSqlRaw as opposed to ExecuteSqlCommand

           return View(myList);

       }



(B) NOT using Scaffold - create model and manually do mapping to dbContext etc.

see next document ModelFirstWebApplication.docx