1.Grade:

Problem:

Write a class called Grade with m1,m2,m3,m4,m5,average as integers. GetDetails which accepts 5 subject marks private string CalculateGrade(average) check the average and return grade if average >=80 and <=100 return first if average >=50 and <80 return second if average >=30 and <50 return third if average >=0 and <30 return fail else return invalid marks write DisplayDetails() and call CalculateGrade in DisplayDetails()

Solution:

```
using System;
class Grade
{
   private int m1, m2, m3, m4, m5;
   private int average;

   // Method to get subject marks
   public void GetDetails()
   {
      Console.Write("Enter marks for subject 1: ");
      m1 = Convert.ToInt32(Console.ReadLine());

      Console.Write("Enter marks for subject 2: ");
      m2 = Convert.ToInt32(Console.ReadLine());

      Console.Write("Enter marks for subject 3: ");
      m3 = Convert.ToInt32(Console.ReadLine());

      Console.Write("Enter marks for subject 4: ");
      m4 = Convert.ToInt32(Console.ReadLine());

      Console.Write("Enter marks for subject 5: ");
      m5 = Convert.ToInt32(Console.ReadLine());
   }

   // Method to calculate grade based on average
   private string CalculateGrade(int average)
   {
      if (average >= 80 && average <= 100)
      {
         return "First";
      }
      else if (average >= 50 && average < 80)
      {
         return "Second";
      }
      else if (average >= 30 && average < 50)
      {
         return "Third";
      }
      else if (average >= 0 && average < 30)
      {
         return "Fail";
```

}
else

```
    {
        return "Invalid marks";
    }
}

// Method to display details and call CalculateGrade
public void DisplayDetails()
{
    average =
```

2.Functions :

Problem:

Create a function PayRate which takes rate as input parameter and returns a table with EmployeeID, RateChangeDate, Rate. PayFrequency, ModifiedDate as columsn. All the employees whose rate is greater the rate parameter. Use Adventureworks database and use Employee Payment History . Use Sql functions with Multivalued Table function

Solution

```
CREATE FUNCTION PayRate (@rate money)
                    RETURNS @table TABLE
                    (EmployeeID int NOT NULL,
                    RateChangeDate datetime NOT NULL,
                    Rate money NOT NULL,
                    PayFrequency tinyint NOT NULL,
                    ModifiedDate datetime NOT NULL)
                    AS
                    BEGIN
                        INSERT @table
                        SELECT * FROM
                HumanResources.EmployeePayHistory
                        WHERE Rate > @rate
                    RETURN
                    END


SELECT * FROM PayRate(45)
```

3.DisplayStudent Marks:

Problem Statement

Write a Program which behaves as prescribed in the below problem statement

Create a program to store list of marks using generic collection class.
The program should allow storage of int values only and it
should sort the list of marks before they are displayed using
DisplayGlist function.
The program should ignore non integer values.

- Take input/output as specified
- Print the expected output using the expected logic/algorithm/data

- Code is structured correctly and according to the problem statement

Instructions

- Ensure your code compiles without any errors/warning/deprecations
- Follow best practices while coding
- Avoid too many & unnecessary usage of white spaces (newline, spaces, tabs, ...), except to make the code readable
- Use appropriate comments at appropriate places in your exercise, to explain the logic, rational, solutions, so that evaluator can know them
- Try to retain the original code given in the exercise, to avoid any issues in compiling & running your programs
- Always test the program thoroughly, before saving/submitting exercises/project
- For any issues with your exercise, contact your coach

Example

```
sample Input:
51
88
92
56
xyz
Expected Output :
51
56
88
92
```

Warnings

- Take care of whitespace/trailing whitespace
- Trim the output and avoid special characters
- Avoid printing unnecessary values other than expected/asked output

```
using System;
using System.Collections.Generic;

        /*
        Question:
                41.     Create a program to store list of marks using generic collection class. The program should allow storage of int values only and it should
                sort the list of marks before they are displayed using DisplayGlist function. The program should ignore non integer values.  Values are
                passed to the program using command line arguments.
        */
namespace LearnCsharp
{
        class M7StudentMarks12
        {
           public static void Main(string[] args)
           {
                //Complete/Update the code below
                        List<int> lMarks;
        lMarks = new List<int>();
```

```
        int iVal,iVa1=0;
        string[] marks=new string[100];
        int i=0;
        do
        {
           marks[i]=Console.ReadLine();
           if(marks[i] == null)
           break;
           if(int.TryParse(marks[i],out iVal) )
           lMarks.Add(iVal);
           i++;

        }while(marks[i] == null);

        /* foreach (string sItem in args)
        {
           if (int.TryParse(sItem, out iVal))
           {
                    iVal--;
              lMarks.Add(iVa1);
           }
        } */

        lMarks.Sort();
        DisplayGlist(lMarks);

     }

     private static void DisplayGlist(List<int> lMarks)
     {
        foreach (int lObj in lMarks)
        {
           Console.WriteLine(lObj);
        }
     }
  }
}
```

4.Dictionary:

Your task here is to implement a **C#** code based on the following specifications. Note that your code should match the specifications in a precise manner. Consider default visibility of classes, data fields and methods unless mentioned otherwise.

Specifications:

```
class Source:
  method definitons:
     Count(Dictionary<string, string> dict): get count of key/value pairs in Dictionary
        return type: int
        visibility: public
```

CheckKey(Dictionary<int, string> dict): Method to check if key 3 is available in Dictionary.
return type: int

visibility: public

Values(Dictionary<string, string> dict): Method to get the values in Dictionary
return type: string
visibility: public

Task:

Given a class **Source,** your task here is to implement the below given methods:

- **Count(Dictionary<string, string> dict):** Method to get the **count** of key/value pairs in Dictionary
- **CheckKey(Dictionary<int, string> dict):** Method to check if key **3** is available in Dictionary. If key is present in Dictionary return **value** paired with key **3** else return **"Could not find the specified key."**
- **Values(Dictionary<string, string> dict):** Method to get the **values** in Dictionary

IMPORTANT:

- If you want to test your program you can implement a **Main()** method given in the stub and you can use **RUN CODE** to test your Main(), provided you have made valid function calls with valid data required.

```
using System;
using System.Collections;
using System.Linq;
using System.Collections.Generic;

class Source {
  public string CheckKey(Dictionary<int, string> dict){
  string result;

  if(dict.TryGetValue(3, out result))
  {
    return (result);
  }
  else
  {
   return ("Could not find the specified key.");
  }
  }
  // public bool CheckForPair(Dictionary<int, string> dict){
  //      return (dict.Contains(new KeyValuePair<int,string>(1,"One")));
  // }
  public int Count(Dictionary<string, string> dict){

    return (dict.Count);
  }
  public string Values(Dictionary<string, string> dict){
  Dictionary<string, string>.ValueCollection valueColl=dict.Values;
  var str = "";
  foreach(string s in valueColl) {
      str += s + " ";
  }
  return str;
  }
```

```
}
```

5.Constraints:

Problem

Create Department table with Deptid,DeptName,Hod and create Employee Table with EmployeeID,FirstName,LastName,DepartmentID. Create Primarykey and Foreign Key

Solution

```
Use Master

Create Database NewOrganizationDB

use OrganizationDB

Create table Department
(
Deptid int Primary Key identity(1000,100) ,
DeptName varchar(50) not null,
Hod varchar(50) not null
)

CREATE TABLE Employee(
EmployeeID int IDENTITY (1,1) NOT NULL,
FirstName nvarchar(50) NOT NULL,
LastName nvarchar(50) NOT NULL,
DepartmentID int NULL,
CONSTRAINT PK_EmployeeID PRIMARY KEY(EmployeeID),
CONSTRAINT FK_Employee_Department FOREIGN KEY(DepartmentID)
REFERENCES Department(Deptid) ON Delete set null
)

//when we enter only partial data, colnames in mandatory
// do not enter data for identity column
insert into Department(DeptName,Hod) Values('Sales','Samatha')
insert into Department(DeptName,Hod) Values('Accounts','Priyanka')
insert into Department(DeptName,Hod) Values('Marketing','Smruthi')
insert into Department(DeptName,Hod) Values('IT','Rekha')
insert into Department(DeptName,Hod) Values('Testing','Raghu')

insert into Employee(FirstName,LastName,DepartmentID) values('samatha','Ramakrishna',Null)
insert into Employee(FirstName,LastName,DepartmentID) values('Samadrita','Chaterjee',Null)
insert into Employee(FirstName,LastName,DepartmentID) values('Supriya','Karn',Null)
insert into Employee(FirstName,LastName,DepartmentID) values('Margana','Neelima',Null)
insert into Employee(FirstName,LastName,DepartmentID) values('Rimpa','Satpathi',Null)

insert into Employee(FirstName,LastName,DepartmentID) values('Krishita','Viroja',1000)
insert into Employee(FirstName,LastName,DepartmentID) values('Priyanka','Kanubai Sagar',1000)
insert into Employee(FirstName,LastName,DepartmentID) values('Shruti','Kumari',1100)
insert into Employee(FirstName,LastName,DepartmentID) values('Smruthi','KalpanaDutta',1100)
```

insert into Employee(FirstName,LastName,DepartmentID) values('Gadde','Apoorva',1200)

Select * Department

Select * Employee

delete from Department where Deptid = 1000

6.Collections:

Problem Statement

Write a Program which behaves as prescribed in the below problem statement

Create a program which can save list of messages in the stack object and same should be processed (display) using a function ProcessStack.Prgram takes the input as list of messages in a single line, and displays the all the messages, each message in a separate line.

Example

sample Input:
"email from Ram at 10:10 am" "email from Ramesh at 10:15 am" "email from Rajan at 10:20 am" "email from Rakesh at 10:25 am"

Expected Output:
email from Rakesh at 10:25 am
email from Rajan at 10:20 am
email from Ramesh at 10:15 am
email from Ram at 10:10 am

Warnings

- Take care of whitespace/trailing whitespace
- Trim the output and avoid special characters
- Avoid printing unnecessary values other than expected/asked output

Hints

- Use Split() method from the String class to process the input

Solution

```
using System;

using System.Collections;

namespace LearnCsharp
{
 class M7WorkingWithStack8
 {
    public static void Main(string[] args)
    {
```

```
//Write Your Code  Here
string str=Console.ReadLine();
```

```
  string[] msgs=str.Split('"' );
        Stack sObj;
        sObj = new Stack();

for(int i=0;i<msgs.Length;i++)
{
 if(msgs[i].Length > 1)
        sObj.Push(msgs[i]);

}
        ProcessStack(sObj);

      }

      private static void ProcessStack(Stack qObj)
      {
        while (qObj.Count > 0)
        {
          Console.WriteLine(qObj.Pop().ToString());
        }
      }
 }

}
```

7. Centigrade-Farenheit:

Problem

Write a Program in C# which will convert Centigrade to Fahrenheit

Solution:

```
using System;
class TemperatureConverter
{
   static void Main()
   {
      Console.WriteLine("Temperature Converter: Celsius to Fahrenheit");

      // Input temperature in Celsius
      Console.Write("Enter temperature in Celsius: ");
      double celsius = Convert.ToDouble(Console.ReadLine());

      // Convert Celsius to Fahrenheit
      double fahrenheit = CelsiusToFahrenheit(celsius);

      // Display the result
      Console.WriteLine($"Temperature in Fahrenheit: {fahrenheit} °F");
   }

   // Function to convert Celsius to Fahrenheit
```

```
static double CelsiusToFahrenheit(double celsius)
{
```

```
        // Formula: (°C × 9/5) + 32
        return (celsius * 9 / 5) + 32;
    }
}
```

8.Book Serialization:

Your task here is to implement a C# code based on the following specifications. Note that your code should match the specifications in a precise manner. Consider **default visibility** of classes, data fields and methods unless mentioned otherwise.

Specifications:

```
class definitions:
 class Book:
    method definitons:
      Name: Implement getter setter method (use Auto Implementation Property)
         return type: string
         visibility: public
      Price: Implement getter setter method (use Auto Implementation Property)
         return type: string
         visibility: public
      Author: Implement getter setter method (use Auto Implementation Property)
         return type: string
         visibility: public
      Year: Implement getter setter method (use Auto Implementation Property)
         return type: string
         visibility: public
      Book(string name, string price, string author, string year) : constructor
         visibility: public
      Ser(List<Book> books) : method to implement serialization
         return type: stream
         visibility: public
         return: stream(serialized list in binary format)
      Deser(FileStream s): method to implement deserialization
         return type: List
         visibility:  public
         return: deserialized list
      main(String args[]): method of type static void
            List<Book> list: List
            s: FileStream
            method calls:
               Ser(list)
               Deser(s)
```

Task:

Create a Book class with **string Name, string Price, string Author, string Year attributes,** your task is to implement the below given methods in order to perform serialization and deserialization.

- Define getter setter method using **Auto Implementation Property**
- Define parameterized constructor.

- Implement **Ser(List<Book> books)** method to serialize List<Book>. The serialization, which takes place should be done by sending the Serialize message to the BinaryFormatter object and serialize it to the file called bks.txt.(The serialization relies on a binary stream, represented by an instance of class Filestream)
- Implement **Deser(FileStream s)** method to deserialize the list from the file .

Note:

The class which needs to be serialized needs to have the [Serializable] attribute.

IMPORTANT:

- If you want to test your program you can implement a Main() function given in the stub and you can use RUN CODE to test your Main() provided you have made valid function calls with valid data required.

Sample Input

```
Book first = new Book( "Alchemist", 175, "Paulo Coelho", 1988 );
```

```
using System;
using System.Collections.Generic;
using System.IO;
using System.Linq;
using System.Runtime.Serialization.Formatters.Binary;
using System.Text;
using System.Threading.Tasks;
using System.Security.Cryptography;
using System.Net;
[Serializable]
public class Book
  {
     public string Name { get; set; }
     public string Price { get; set; }
     public string Author { get; set; }
     public string Year { get; set; }
     public Book(string name, string price, string author, string year){
        Name = name; Price = price; Author = author; Year = year;
     }
     public FileStream Ser(List<Book> books) {
        BinaryFormatter f = new BinaryFormatter();
        FileStream s = new FileStream(@"bks.txt",FileMode.Create,FileAccess.Write);
        f.Serialize(s, books);
        s.Close();
        return s;

  }
  public List<Book> Deser(FileStream s) {
            BinaryFormatter f = new BinaryFormatter();
     s = new FileStream(@"bks.txt",FileMode.Open,FileAccess.Read);
     List<Book> lst = (List<Book>)f.Deserialize(s);
     return lst;
```

```
    }
}
```

9.Array and multiples of 5:

Problem Statement

Write a Program which behaves as prescribed in the below problem statement

Write a program to initialize a single dimensional array of any size with integer values . Display the complete array content with count of the numbers which are divisible by 5 to the end.
Size of the array is first input to the program followed by the elements of the array.

- Assume all input values are >= 5
- Take input/output as specified
- Print the expected output using the expected logic/algorithm/data
- Code is structured correctly and according to the problem statement

Instructions

- Ensure your code compiles without any errors/warning/deprecations
- Follow best practices while coding
- Avoid too many & unnecessary usage of white spaces (newline, spaces, tabs, ...), except to make the code readable
- Use appropriate comments at appropriate places in your exercise, to explain the logic, rational, solutions, so that evaluator can know them
- Try to retain the original code given in the exercise, to avoid any issues in compiling & running your programs
- Always test the program thoroughly, before saving/submitting exercises/project
- For any issues with your exercise, contact your coach

Example

Sample Input :
7 //Size of the array
5
10
15
16
1
10
21
Expected Output :
Count of elements divide by 5: 4

Warnings

- Take care of whitespace/trailing whitespace
- Trim the output and avoid special characters
- Avoid printing unnecessary values other than expected/asked output

Solution

using System;

```
namespace LearnCsharp
{

 class CountOfDivideByFive
 {
    public static void Main(string[] args)
    {
  //Write Your Code Here
    int[] iArray;
    int Count=int.Parse(Console.ReadLine());
          iArray = new int[Count];
          int iCount = 0;
          for (int iVal1 = 0; iVal1 < Count; iVal1++)
          {
             iArray[iVal1] = int.Parse(Console.ReadLine());
          }

          foreach (int item in iArray)
          {
             if ((item % 5) == 0)
                iCount++;

             Console.WriteLine(item);
          }
          Console.WriteLine("Count of elements divide by 5: " + iCount);
    }
 }

}
```

10. AccountDetails:

Problem Statement - Account Details

Complete the class Account and AccountDetails as per the below requirement

class Account :

Create the following instance/static members:

accountNo : int

balance : double

accountType : string

counter :int static

Define parameterized constructor with two parameters to initialize balance and accountType. accountNo should be initialized by incrementing counter.

- Implement the below operations:
- void depositAmount(double amount)

- To add amount to account balance

- void printAccountDetails()
- To display account details as per format given in Example Section

class AccountDetails :

- Create GetData() method and follow the below instructions.
- Accept balance, account type and amount as input for two account objects from Console(Refer Example section for input format)
- create first object using the input data and display account details
- Deposit amount using the input data and display the new account balance
- create second account object using the input data and display account details.
- Set account balance to new balance using input data and display the new account balance

Note : Don't Implement the Main method

Example

```
Sample Input:
100.5
Savings
25.5                                    // balance type amount for first account
200
Current
50.5                                    // balance type amount for second account

Expected Output:
[Acct No : 1, Type : Savings, Balance : 100.5]
New Balance : 126.0
[Acct No : 2, Type : Current, Balance : 200.0]
New Balance : 250.5
Sample Input:
0
Current
100
0
Current
50

Expected Output:
[Acct No : 1, Type : Current, Balance : 0.0]
New Balance : 100.0
[Acct No : 2, Type : Current, Balance : 0.0]
New Balance : 50.0
```

Instructions

- Do not change the provided class/method names unless instructed
- Ensure your code compiles without any errors/warning/deprecations
- Follow best practices while coding
- Avoid too many & unnecessary usage of white spaces (newline, spaces, tabs, ...), except to make the code readable
- Use appropriate comments at appropriate places in your exercise, to explain the logic, rational, solutions, so that evaluator can know them
- Try to retain the original code given in the exercise, to avoid any issues in compiling & running your programs

- Always test the program thoroughly, before saving/submitting exercises/project
- For any issues with your exercise, contact your coach

Warnings

- Take care of whitespace/trailing whitespace
- Trim the output and avoid special characters
- Avoid printing unnecessary values other than expected/asked output

Solution

```
using System;
public class Account {
        //CODE START
        public static int counter = 0;
        public int  accountNo;
        public double balance;
        public string accountType;

        public Account(double balance, string accountType) {
                accountNo = ++Account.counter;
                this.balance = balance;
                this.accountType = accountType;
        }

        public void depositAmount(double amount) {
                balance += amount;
        }

        public void printAccountDetails() {
                Console.WriteLine("[Acct No : " + accountNo
                                                + ", Type : " + accountType
                                                + ", Balance : {0:N}" + "]",balance);

                //Console.WriteLine(details);
        }


}

public  class AccountDetails{

            public  void GetData() {



    double balance1 = Convert.ToDouble(Console.ReadLine());
                string type1 = Console.ReadLine();
                double amount1 = Convert.ToDouble(Console.ReadLine());
                double balance2 = Convert.ToDouble(Console.ReadLine());
                String type2 = Console.ReadLine();
                double amount2 = Convert.ToDouble(Console.ReadLine());


                Account a1 = new Account(balance1, type1);
                a1.printAccountDetails();
```

```
a1.depositAmount(amount1);
Console.WriteLine("New Balance : {0:N}",a1.balance);
```

```
            Account a2 = new Account(balance2, type2);
            a2.printAccountDetails();
            a2.balance = amount2;
            Console.WriteLine("New Balance : {0:N}",a2.balance);

        }

    }
```

11. Joins:

Create two tables Department and Employee with relationship and implement all the joins

Solution

```
Use Master

Create Database OrganizationDB

use OrganizationDB

Create table Department
(
Deptid int Primary Key identity(1000,100) ,
DeptName varchar(50) not null,
Hod varchar(50) not null
)

CREATE TABLE Employee(
EmployeeID int IDENTITY (1,1) NOT NULL,
FirstName nvarchar(50) NOT NULL,
LastName nvarchar(50) NOT NULL,
DepartmentID int NULL,
CONSTRAINT PK_EmployeeID PRIMARY KEY(EmployeeID),
CONSTRAINT FK_Employee_Department FOREIGN KEY(DepartmentID)
REFERENCES Department(Deptid)
)

//when we enter only partial data, colnames in mandatory
// do not enter data for identity column
insert into Department(DeptName,Hod) Values('Sales','Samatha')
insert into Department(DeptName,Hod) Values('Accounts','Priyanka')
insert into Department(DeptName,Hod) Values('Marketing','Smruthi')
insert into Department(DeptName,Hod) Values('IT','Rekha')
insert into Department(DeptName,Hod) Values('Testing','Raghu')

insert into Employee(FirstName,LastName,DepartmentID) values('samatha','Ramakrishna',Null)
insert into Employee(FirstName,LastName,DepartmentID) values('Samadrita','Chaterjee',Null)
insert into Employee(FirstName,LastName,DepartmentID) values('Supriya','Karn',Null)
insert into Employee(FirstName,LastName,DepartmentID) values('Margana','Neelima',Null)
insert into Employee(FirstName,LastName,DepartmentID) values('Rimpa','Satpathi',Null)
insert into Employee(FirstName,LastName,DepartmentID) values('Krishita','Viroja',Null)
```

insert into Employee(FirstName,LastName,DepartmentID) values('Priyanka','Kanubai Sagar',Null)
insert into Employee(FirstName,LastName,DepartmentID) values('Shruti','Kumari',Null)

```
insert into Employee(FirstName,LastName,DepartmentID) values('Smruthi','KalpanaDutta',Null)
insert into Employee(FirstName,LastName,DepartmentID) values('Gadde','Apoorva',Null)

Select * Department

Select * Employee

//retireves only common data
select e1.EmployeeID,e1.FirstName,d1.Depid,d1.DeptName
from Employee e1 Join Department d1
on d1.Depid = e1.DepartmentID

select e1.EmployeeID,e1.FirstName,d1.Depid,d1.DeptName
from Employee e1 left outer Join Department d1
on d1.Depid = e1.DepartmentID

select e1.EmployeeID,e1.FirstName,d1.Depid,d1.DeptName
from Employee e1 right outer Join Department d1
on d1.Depid = e1.DepartmentID

select e1.EmployeeID,e1.FirstName,d1.Depid,d1.DeptName
from Employee e1 Full outer Join Department d1
on d1.Depid = e1.DepartmentID
```

12. List of Student Names:

Problem Statement

Write a Program which behaves as prescribed in the below problem statement

Create a program which will accept list of student names as input, these names have to be filtered, sorted and displayed, filter criteria is to ensure that names do not have values like “Nobody”, “Somebody”.

- Take input/output as specified
- Print the expected output using the expected logic/algorithm/data
- Code is structured correctly and according to the problem statement

Instructions

- Ensure your code compiles without any errors/warning/deprecations
- Follow best practices while coding
- Avoid too many & unnecessary usage of white spaces (newline, spaces, tabs, ...), except to make the code readable
- Use appropriate comments at appropriate places in your exercise, to explain the logic, rational, solutions, so that evaluator can know them
- Try to retain the original code given in the exercise, to avoid any issues in compiling & running your programs
- Always test the program thoroughly, before saving/submitting exercises/project
- For any issues with your exercise, contact your coach

Example

Sample Input:
Ravi
Somebody

```
Nobody
Ani
Nobody
Vishwanath
Somebody
Nitin

Expected output :
Ani
Nitin
Ramesh
Ravi
Tanvir
Vishwanath
```

Warnings

- Take care of whitespace/trailing whitespace
- Trim the output and avoid special characters
- Avoid printing unnecessary values other than expected/asked output

```
using System;

using System.Collections;

namespace LearnCsharp
{

          class NamesWithArrayList
          {
             public static void Main(string[] args)
             {
                              //Update the code below
                ArrayList alObj;
        alObj = new ArrayList();
int max=10;
string item="";
for(int i=0;i<max;i++)
{
   item=Console.ReadLine();
   if(item != null)
   {
      alObj.Add(item);
   }


}

        for(int j=0;j<alObj.Count;j++)
        {
           if(alObj[j].ToString()=="Somebody".Trim() || alObj[j].ToString()=="Nobody".Trim())
           alObj.Remove(alObj[j]);
```

```
}
```

```
        alObj.Sort();

        foreach (var item1 in alObj)
        {
           Console.WriteLine(item1);
        }

     }
        }
}
```

13.

Login:

Problem:

Write a Program Called Login with username as string and password as string . Accept username and password in GetDetails() . ValidateUser(username,password) return true if valid user return false if invalid user. Call ValidateUser in DisplayResult . if validuser display login successful else display login failed

Solution

```
using System;
class LoginProgram
{
   private string username;
   private string password;

   // Method to accept username and password
   public void GetDetails()
   {
      Console.Write("Enter username: ");
      username = Console.ReadLine();

      Console.Write("Enter password: ");
      password = Console.ReadLine();
   }

   // Method to validate user
   private bool ValidateUser(string username, string password)
   {
      // For demonstration purposes, let's consider a simple validation
      // You can replace this with your own logic, such as checking against a database
      return username == "admin" && password == "admin123";
   }
   // Method to display result based on validation
   public void DisplayResult()
   {
      bool isValidUser = ValidateUser(username, password);

      if (isValidUser)
      {
         Console.WriteLine("Login successful!");
```

```
        }
        else
        {
            Console.WriteLine("Login failed. Invalid username or password.");
        }
    }

    static void Main()
    {
        LoginProgram loginProgram = new LoginProgram();

        // Get user details
        loginProgram.GetDetails();

        // Validate user and display result
        loginProgram.DisplayResult();
    }
}
```

Write a Program Called Login with username as string and password as string . Accept username and password in GetDetails() . ValidateUser(username,password) return true if valid user return false if invalid user. Call ValidateUser in DisplayResult . if validuser display login successful else display login failed

14: Min Max:

Problem Statement - Find Maximum and Minimum Age

Complete the main method to accept the age of n students and find the maximum and minimum age .

The first input is the number n representing the number of age values you need to enter as integers

Followed by the age values in a separate line.

The output should display as shown below in sample input /output.

Following requirements should be taken care in the program.

1. Input should be taken through Console
2. Program should print the output as described in the Example Section below
3. The number n representing the number of students should be allowed in the range of 1 to 20
4. If n is entered less than 1 or more than 20 , it should print message as INVALID_INPUT.

Example

Sample Input 1:
5
34
56
12
89
43
Sample Ouptut 1:
MIN=12
MAX=89

Sample Input 2:
25
Expected Output:
INVALID_INPUT

Sample Input 3:
8
78
44
23
65
45
9
23
39
Expected Output:
MIN=9
MAX=78

Instructions

- Do not change the provided class/method names unless instructed
- Ensure your code compiles without any errors/warning/deprecations
- Follow best practices while coding
- Avoid too many & unnecessary usage of white spaces (newline, spaces, tabs, ...), except to make the code readable
- Use appropriate comments at appropriate places in your exercise, to explain the logic, rational, solutions, so that evaluator can know them
- Try to retain the original code given in the exercise, to avoid any issues in compiling & running your programs
- Always test the program thoroughly, before saving/submitting exercises/project
- For any issues with your exercise, contact your coach

Warnings

- Take care of whitespace/trailing whitespace
- Trim the output and avoid special characters
- Avoid printing unnecessary values other than expected/asked output using System;

```
namespace LearnCsharp
{
        class FindMaxMinAge{
            public static void Main(string[] args) {
    int n = Convert.ToInt32(Console.ReadLine());
    if (n < 1 || n > 20)
    {
       Console.WriteLine("INVALID_INPUT");
    }
    else
    {
       int[] ages = new int[n];

       for (int i = 0; i < n; i++)
       {
```

```
ages[i] = Convert.ToInt32(Console.ReadLine());
```

```
        }


        int min = ages[0];
        for (int i = 0; i < n; i++)
        {
           if (ages[i] < min)
              min = ages[i];
        }

        Console.WriteLine("MIN=" + min);


        int max = ages[0];

        for (int i = 0; i < n; i++)
        {
           if (ages[i] > max)
              max = ages[i];
        }

        Console.WriteLine("MAX=" + max);
     }

           }
         }
}
```

15. Palendrome Destroyer:

Problem Statement

John asked for a puzzle from one of his friends. He has given a string and he has to decode the given string according to the set of rules –

1. Reverse each word of the space-separated string.
2. Eliminate palindrome word.

Palindrome words are those words that can be read the same from either side. For example – “aba” is the same as the reverse of “aba”; Therefore, it is a palindrome.

Input Format

- First line contains the value of **N**, no. of queries(String).
- Next **N** lines contains string.

Output Format

- For each query print the decoded string in a newline.

Constraints

- 1 <= **N** <= 100
- 1 <= **length of s** <= 1000

Sample Input

```
2
i love my country
she is madam
```

Sample Output

```
evol ym yrtnuoc
ehs si
```

Explanation

Query 1 -

- After reversing each word - **I evol ym yrtnuoc**.
- After eliminating palindrome words - **evol ym yrtnuoc**

Query 2 -

- After reversing each word - **ehs is madam**
- After eliminating palindrome words - **ehs is**

Solution :

```
using System;
using System;
using System.Linq;

class Program
{
   static void Main(string[] args)
   {
      int N = Convert.ToInt32(Console.ReadLine());

      for (int i = 0; i < N; i++)
      {
         string input = Console.ReadLine();
         string[] words = input.Split(' ');

         string decodedString = DecodeString(words);
         Console.WriteLine(decodedString);
      }
```

```
}
```

```
    static string DecodeString(string[] words)
    {
        string decoded = "";

        foreach (string word in words)
        {
            string reversed = ReverseWord(word);

            if (!IsPalindrome(reversed))
            {
                decoded += reversed + " ";
            }
        }

        return decoded.Trim();
    }

    static string ReverseWord(string word)
    {
        char[] charArray = word.ToCharArray();
        Array.Reverse(charArray);
        return new string(charArray);
    }

    static bool IsPalindrome(string word)
    {
        int left = 0;
        int right = word.Length - 1;

        while (left < right)
        {
            if (word[left] != word[right])
            {
                return false;
            }

            left++;
            right--;
        }

        return true;
    }
}
```

16. Sort:

Problem Statement - Sort Numbers

Complete the main method to Accept n numbers and display the numbers in ascending order as output ,if n is even. If n is odd, then display the numbers in descending order

Following requirements should be taken care in the program.

1. Input should be taken through Console

2. Program should print the output as described in the Example Section below
3. The first input n should represent the total number of values entered followed by the actual values to be sorted.
4. n should be within the range of 1 to 20 . If n is entered as less than 1 or more than 20 , it should show message as INVALID_INPUT.

Example

Sample Input 1:
7
23
45
67
97
65
34
74
Expected Output:
97 74 67 65 45 34 23

Sample Input 2:
6
77
44
22
65
28
43
Expected Output2:
22 28 43 44 65 77

Sample Input 3:
0
Expected Output 3:
INVALID_INPUT

Sample Input 4:
30
Expected Output 4:
INVALID_INPUT

Instructions

- Do not change the provided class/method names unless instructed
- Ensure your code compiles without any errors/warning/deprecations
- Follow best practices while coding
- Avoid too many & unnecessary usage of white spaces (newline, spaces, tabs, ...), except to make the code readable
- Use appropriate comments at appropriate places in your exercise, to explain the logic, rational, solutions, so that evaluator can know them
- Try to retain the original code given in the exercise, to avoid any issues in compiling & running your programs
- Always test the program thoroughly, before saving/submitting exercises/project
- For any issues with your exercise, contact your coach

Warnings

- Take care of whitespace/trailing whitespace

- Trim the output and avoid special characters
- Avoid printing unnecessary values other than expected/asked output

Solution

```
using System;

namespace LearnCsharp
{
        class Sortnumbers{
          public static void Main(string[] args) {

    int n = Convert.ToInt32(Console.ReadLine());

    if (n < 1 || n > 20)
    {

      Console.WriteLine("INVALID_INPUT");
    }

    else
    {
      int[] arr = new int[n];

      for (int i = 0; i < n; i++)
      {

        arr[i] = Convert.ToInt32(Console.ReadLine());

      }

      if (n % 2 == 0)
      {
        Array.Sort(arr);
        for (int i = 0; i < arr.Length; i++)
        {
          Console.Write(arr[i] + " ");
        }


      }
      else
      {

        for (int i = 0; i < n; i++)
        {

          for (int j = 0; j < n; j++)
          {

            if (arr[i] > arr[j])
            {
              int temp;
```

```
temp = arr[i];
arr[i] = arr[j];
```

```
                arr[j] = temp;
              }
            }
          }

          for (int i = 0; i < arr.Length; i++)
          {
            Console.Write(arr[i] + " ");
          }



       }

      }
          }
        }
}
```

17. SumPrime Number Loop:

DESCRIPTION

Write a function *sumprimes(ls)* that takes as input a list of integers ls and returns the sum of all the prime numbers in ls.

Input:

The input has a list of values separated by space.

Output:

A single number representing sum of all prime numbers in the given list.

Sample Input 1:

3 3 1 13

Sample Output 1:

19

Solution:

```
using System;
using System.Linq;

class Program
{
  static bool IsPrime(int num)
```

```
{
   if (num < 2)
```

```
        return false;

    for (int i = 2; i <= Math.Sqrt(num); i++)
    {
        if (num % i == 0)
            return false;
    }

    return true;
}

static int SumPrimes(int[] numbers)
{
    return numbers.Where(IsPrime).Sum();
}

static void Main()
{
    Console.WriteLine("Enter a list of integers separated by space:");
    string input = Console.ReadLine();

    // Split the input string into an array of integers
    int[] numbers = input.Split(' ').Select(int.Parse).ToArray();

    // Calculate and display the sum of prime numbers
    int result = SumPrimes(numbers);
    Console.WriteLine($"Sum of prime numbers: {result}");
}
}
```

18. Voter Eligibility:

Problem:

Write a C# Program Which accepts the name and age and display whether the person is eligible to vote

Solution

```
using System;

class VotingEligibilityChecker
{
    static void Main()
    {
        Console.WriteLine("Voting Eligibility Checker");

        // Input name
        Console.Write("Enter your name: ");
        string name = Console.ReadLine();

        // Input age
        Console.Write("Enter your age: ");
        int age = Convert.ToInt32(Console.ReadLine());
```

```
// Check eligibility
bool isEligible = CheckVotingEligibility(age);
```

```
    // Display the result
    Console.WriteLine($"{name}, you are {(isEligible ? "eligible" : "not eligible")} to vote.");
  }

  // Function to check voting eligibility
  static bool CheckVotingEligibility(int age)
  {
    // Voting age in most countries is 18
    const int votingAge = 18;

    return age >= votingAge;
  }
}
```

19. Stored Procedures With Parametes and Result:

Problem:

Create a stored procedure called prcGetEmployeeDetails with EmpId as input parameter and DepName and ShiftId as output parameter. If the empid exists, retrieve Department Name, ShiftID form Department and EmployeeDepartmentHistory and return 1 else return 0.

Call prcGetEmployeeDetails stored procedure in another stored procedure prcDisplayEmployeeStatus

Step1:

```
Use AdventureWorks2019
go
```

Step2:

```
CREATE PROCEDURE prcGetEmployeeDetail @EmpId int, @DepName char(50) OUTPUT, @ShiftId int OUTPUT
        AS
        BEGIN
        IF EXISTS(SELECT * FROM HumanResources.Employee WHERE EmployeeID = @EmpId)
        BEGIN
                SELECT @DepName = d.Name, @ShiftId = h.ShiftID
  FROM HumanResources.Department d JOIN
  HumanResources.EmployeeDepartmentHistory h
                ON d.DepartmentID = h.DepartmentID
                WHERE EmployeeID = @EmpId AND h.Enddate IS NULL
                RETURN 0
        END
        ELSE
        RETURN 1
END
```

Step3:

```
        CREATE PROCEDURE prcDisplayEmployeeStatus @EmpId int
        AS
        BEGIN
   DECLARE @DepName char(50)
   DECLARE @ShiftId int
```

DECLARE @ReturnValue int
EXEC @ReturnValue = prcGetEmployeeDetail @EmpId,

```
  @DepName OUTPUT,@ShiftId OUTPUT
   IF (@ReturnValue = 0)
   BEGIN
     PRINT 'The details of an employee with ID: ' +
    convert(char(10), @EmpId)
     PRINT 'Department Name: ' + @DepName

         PRINT 'Shift ID: ' + convert( char(1), @ShiftId)
 SELECT ManagerID, Title FROM
         HumanResources.Employee
         WHERE EmployeeID = @EmpID
END
ELSE
 PRINT 'No records found for the given employee'
END
```

Step4:

```
EXEC prcDisplayEmployeeStatus 2
```

20. SumPrime:

Number Loop:

DESCRIPTION

Write a function *sumprimes(ls)* that takes as input a list of integers ls and returns the sum of all the prime numbers in ls.

Input:

The input has a list of values separated by space.

Output:

A single number representing sum of all prime numbers in the given list.

Sample Input 1:

3 3 1 13

Sample Output 1:

19

Solution:

```
using System;
using System.Linq;

class Program
{
   static bool IsPrime(int num)
```

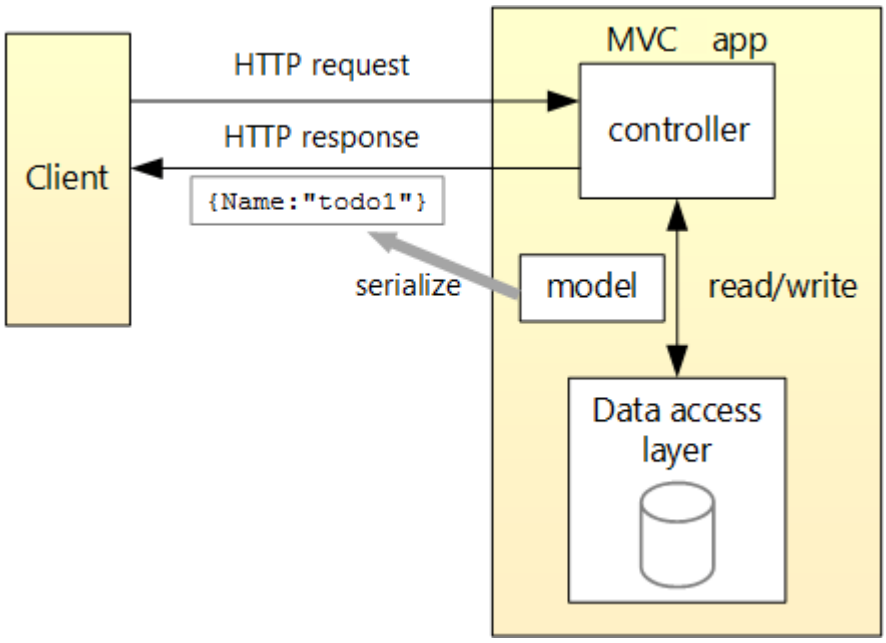
```
{
    if (num < 2)
        return false;
```

```
        for (int i = 2; i <= Math.Sqrt(num); i++)
        {
            if (num %
                i == 0)
                return
                false;
        }

        return true;
    }

    static int SumPrimes(int[] numbers)
    {
        return numbers.Where(IsPrime).Sum();
    }

    static void Main()
    {
        Console.WriteLine("Enter a list of integers separated by
        space:");string input = Console.ReadLine();

        // Split the input string into an array of integers
        int[] numbers = input.Split(' ').Select(int.Parse).ToArray();

        // Calculate and display the sum of prime
        numbersint result = SumPrimes(numbers);
        Console.WriteLine($"Sum of prime numbers: {result}");
    }
}
```

21. Basic Web API

API	Description	Request body	Response body
GET /api/todoitems	Get all to-do items	None	Array of to-do items
GET /api/todoitems/{id}	Get an item by ID	None	To-do item
POST /api/todoitems	Add a new item	To-do item	To-do item
PUT /api/todoitems/{id}	Update an existing item	To-do item	None
DELETE /api/todoitems/{id}	Delete an item	None	None

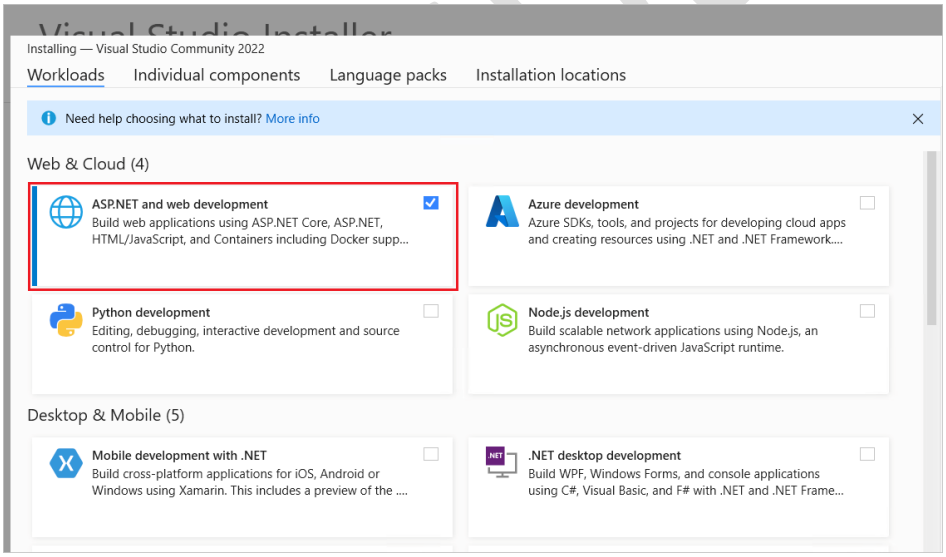
The following diagram shows the design of the app.



Prerequisites

- Visual Studio
- Visual Studio Code
- Visual Studio for Mac

  - Visual Studio 2022 Preview with the ASP.NET and web development workload.



Create a web project

- Visual Studio
- Visual Studio Code
- Visual Studio for Mac
  - From the File menu, select New > Project.
  - Enter Web API in the search box.
  - Select the ASP.NET Core Web API template and select Next.
  - In the Configure your new project dialog, name the project TodoApi and select Next.
  - In the Additional information dialog:

- Confirm the Framework is .NET 8.0 (Long Term Support).
- Confirm the checkbox for Use controllers(uncheck to use minimal APIs) is checked.
- Confirm the checkbox for Enable OpenAPI support is checked.
- Select Create.

Add a NuGet package

A NuGet package must be added to support the database used in this tutorial.

- From the Tools menu, select NuGet Package Manager > Manage NuGet Packages for Solution.
- Select the Browse tab.
- Enter Microsoft.EntityFrameworkCore.InMemory in the search box, and then select Microsoft.EntityFrameworkCore.InMemory.
- Select the Project checkbox in the right pane and then select Install.

Note

For guidance on adding packages to .NET apps, see the articles under Install and manage packages at Package consumption workflow (NuGet documentation). Confirm correct package versions at NuGet.org.

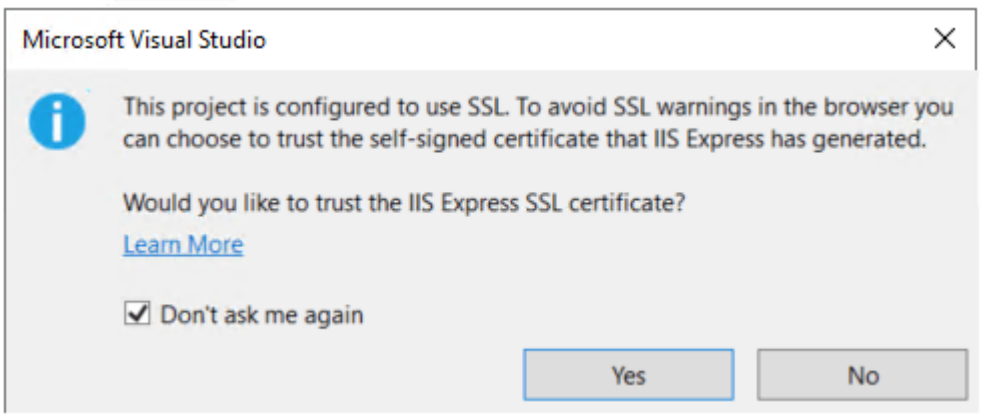
Test the project

The project template creates a WeatherForecast API with support for Swagger.

- Visual Studio
- Visual Studio Code
- Visual Studio for Mac

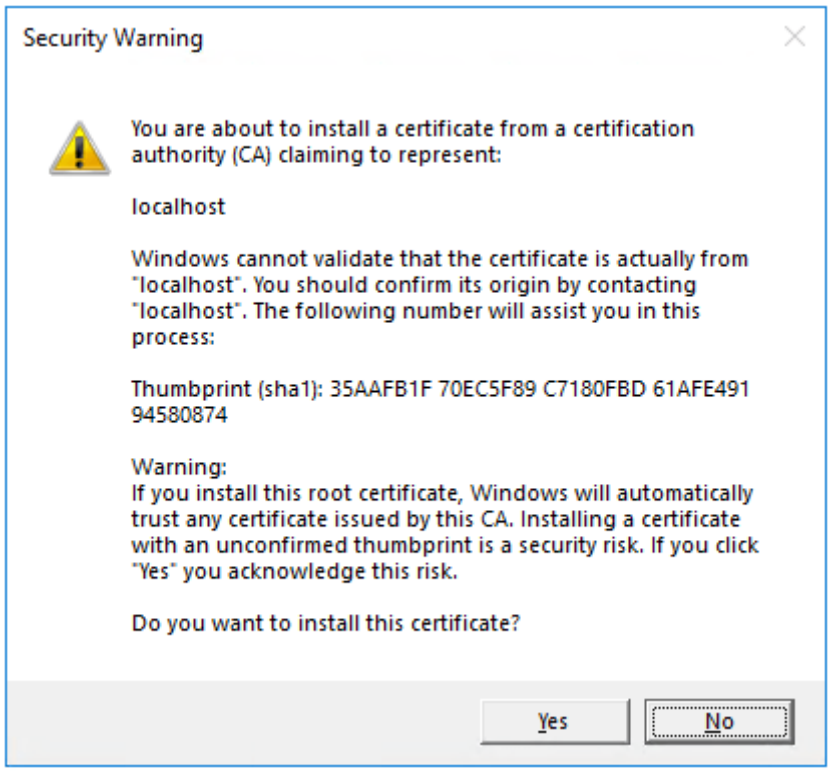
Press Ctrl+F5 to run without the debugger.

Visual Studio displays the following dialog when a project is not yet configured to use SSL:



Select Yes if you trust the IIS Express SSL certificate.

The following dialog is displayed:



Select Yes if you agree to trust the development certificate.For information on trusting the Firefox browser, see Firefox SEC_ERROR_INADEQUATE_KEY_USAGE certificate error.

Visual Studio launches the default browser and navigates to https://localhost:<port>/swagger/index.html, where <port> is a randomly chosen port number set at the project creation.

The Swagger page /swagger/index.html is displayed. Select GET > Try it out > Execute. The page displays:

- The Curl command to test the WeatherForecast API.
- The URL to test the WeatherForecast API.
- The response code, body, and headers.
- A drop-down list box with media types and the example value and schema.

If the Swagger page doesn't appear, see this GitHub issue.

Swagger is used to generate useful documentation and help pages for web APIs. This tutorial uses Swagger to test the app. For more information on Swagger, see ASP.NET Core web API documentation with Swagger / OpenAPI.

Copy and paste the Request URL in the browser: https://localhost:<port>/weatherforecast

JSON similar to the following example is returned:

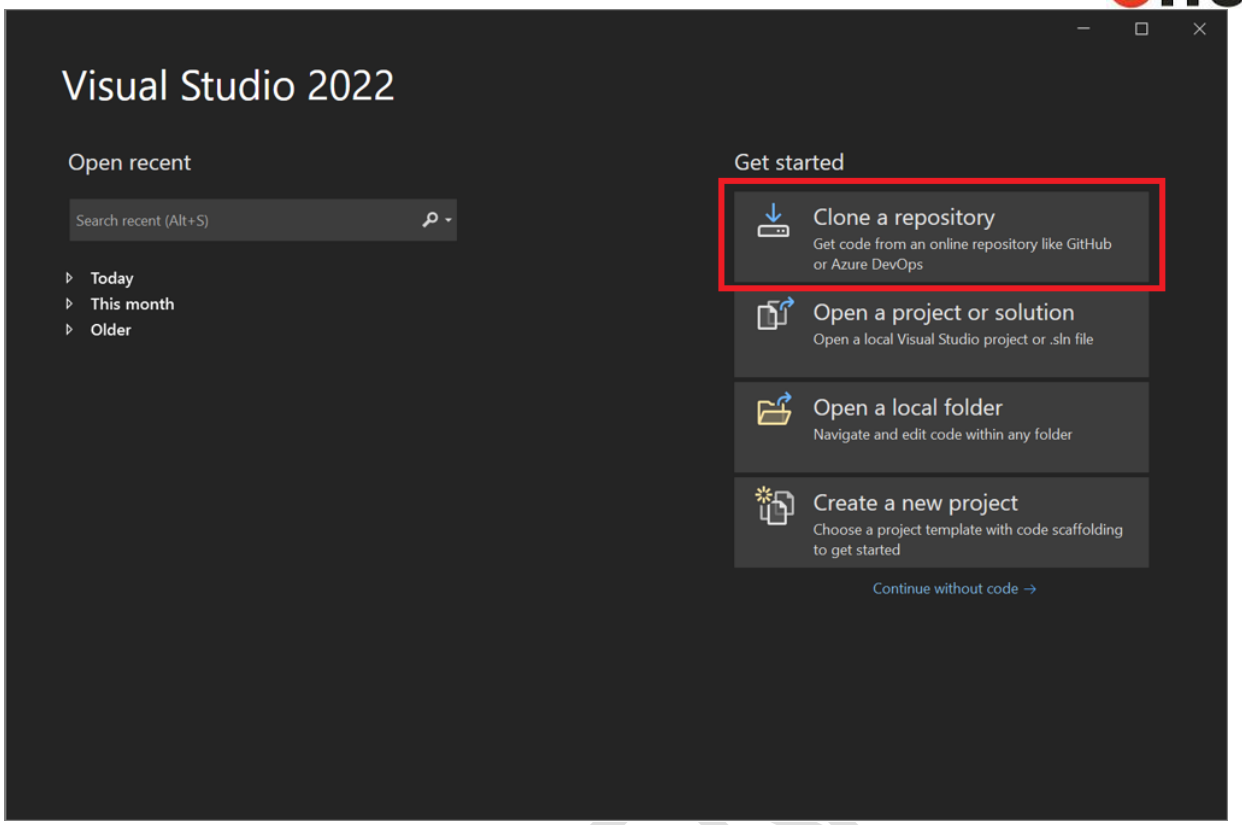
JSONCopy

```
[
  {
    "date": "2019-07-16T19:04:05.7257911-06:00",
    "temperatureC": 52,
    "temperatureF": 125,
    "summary": "Mild"
  },
  {
```

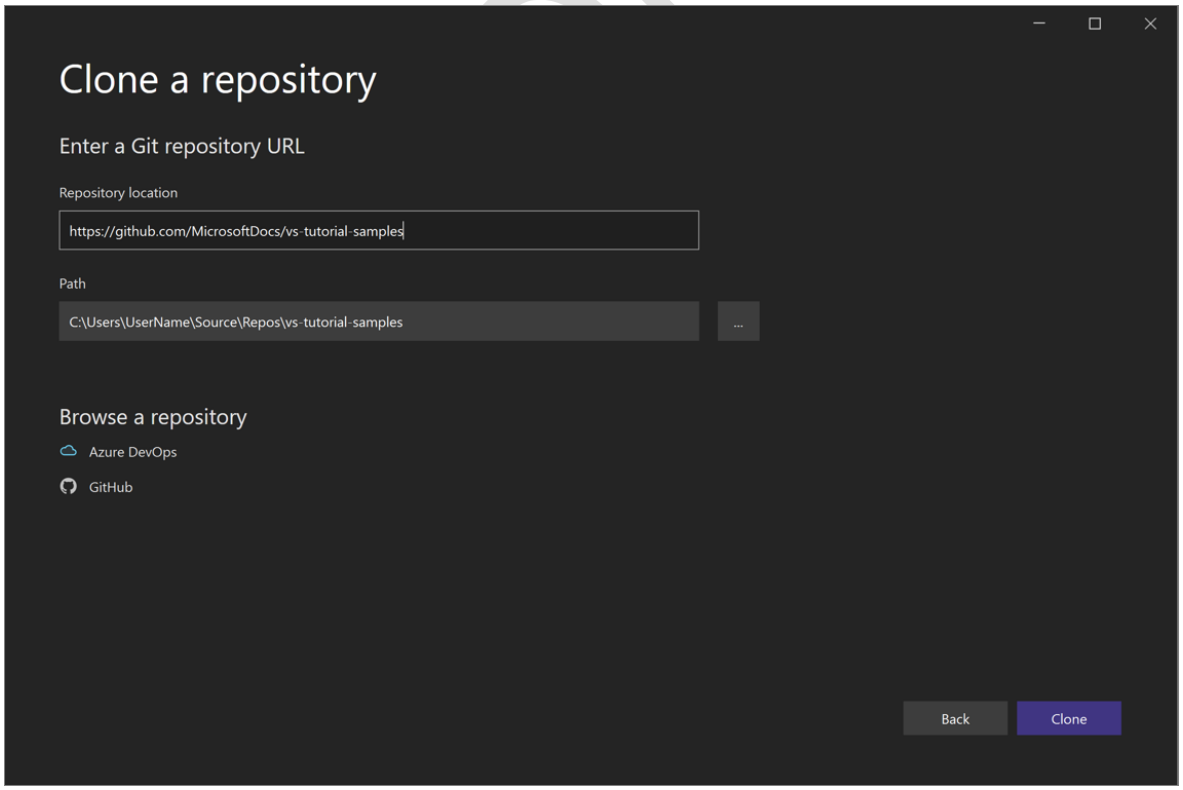
```
    "date": "2019-07-17T19:04:05.7258461-06:00",
    "temperatureC": 36,
    "temperatureF": 96,
    "summary": "Warm"
  },
  {
    "date": "2019-07-18T19:04:05.7258467-06:00",
    "temperatureC": 39,
    "temperatureF": 102,
    "summary": "Cool"
  },
  {
    "date": "2019-07-19T19:04:05.7258471-06:00",
    "temperatureC": 10,
    "temperatureF": 49,
    "summary": "Bracing"
  },
  {
    "date": "2019-07-20T19:04:05.7258474-06:00",
    "temperatureC": -1,
    "temperatureF": 31,
    "summary": "Chilly"
  }
]
```


22. Clone Repository:

1. Open Visual Studio.
2. On the start window, select **Clone a repository**.



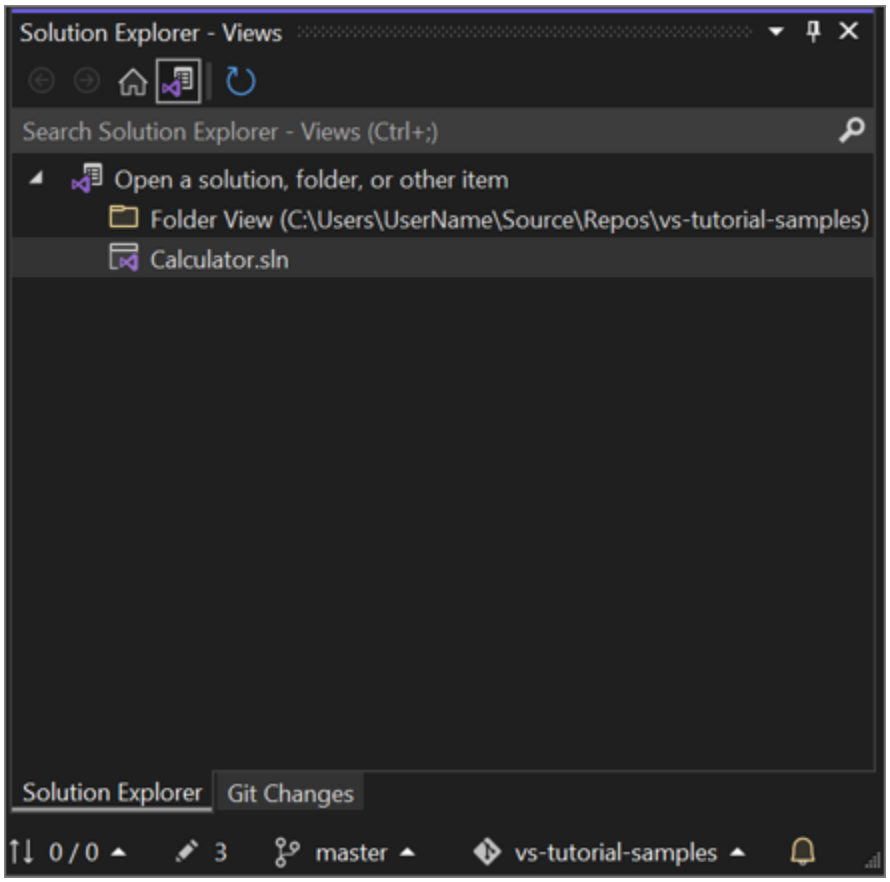
3. Enter or type the repository location, and then select the **Clone** button.



4. If you're not already signed in, you might be prompted to sign into Visual Studio or your GitHub account.

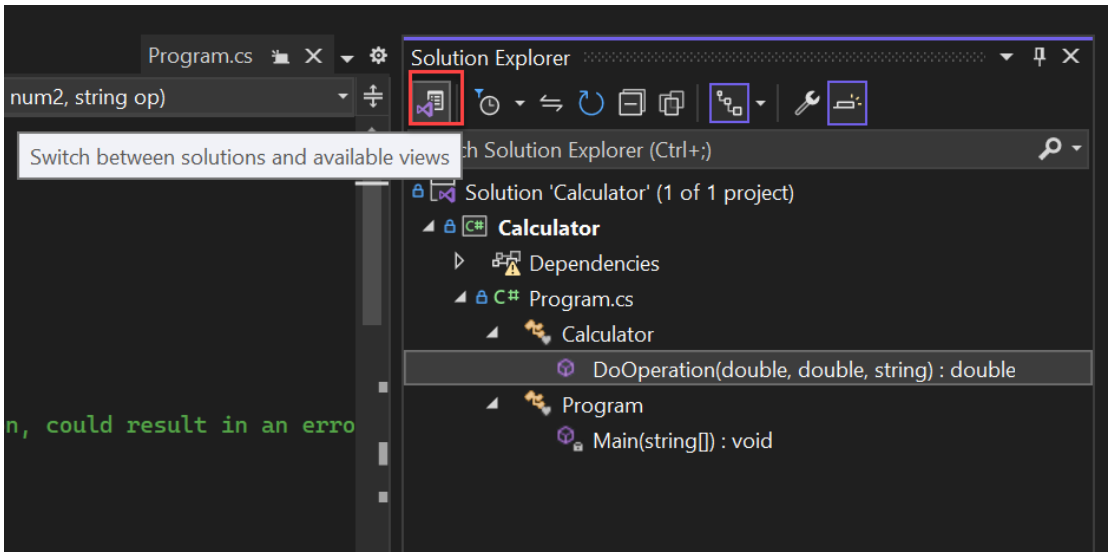
View files in Solution Explorer

1. Next, Visual Studio loads the solution(s) from the repository by using the **Folder View** in **Solution Explorer**.



You can view a solution in **Solution View** by double-clicking its .sln file.

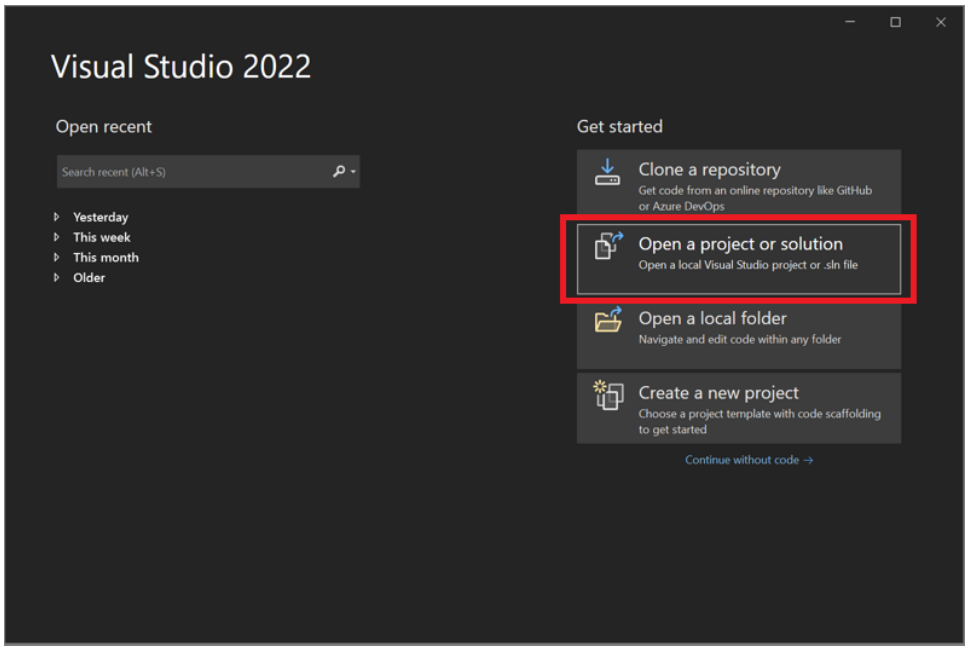
Or, you can select the **Switch Views** button, and then select **Program.cs** to view a solution's code.



Open a project locally from a previously cloned GitHub repo

1. Open Visual Studio.
2. On the start window, select **Open a project or solution**.

   Visual Studio opens an instance of File Explorer, where you can browse to your solution or project, and then select it to open it.



Tip

If you've opened the project or solution recently, select it from the **Open recent** section to quickly open it again.

Start coding!

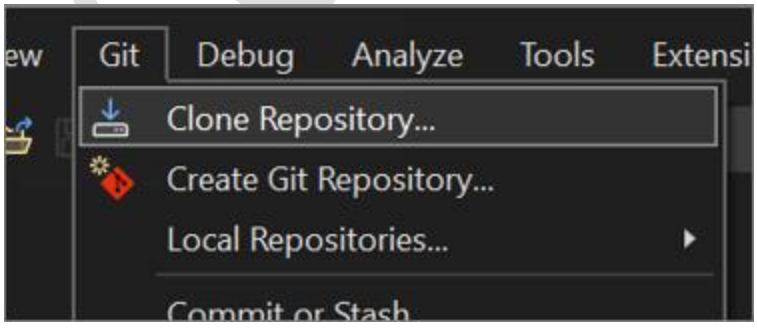
Use the IDE

You can also use the **Git** menu or the **Select Repository** control in the Visual Studio IDE to interact with a repository's folders and files.

Here's how.

To clone a repo and open a project

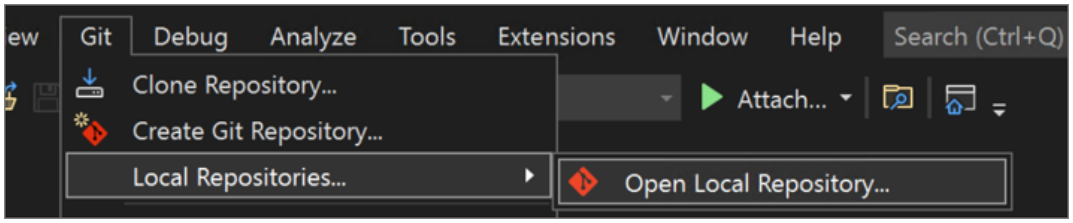
1. In the Visual Studio IDE, select the **Git** menu, and then select **Clone Repository**.



2. Follow the prompts to connect to the Git repository that includes the files you're looking for.

To open local folders and files

1. In the Visual Studio IDE, select the **Git** menu, select **Local Repositories**, and then select **Open Local Repository**.



2. Follow the prompts to connect to the Git repository that has the files you're looking for.

23. Create a GitHub repo:

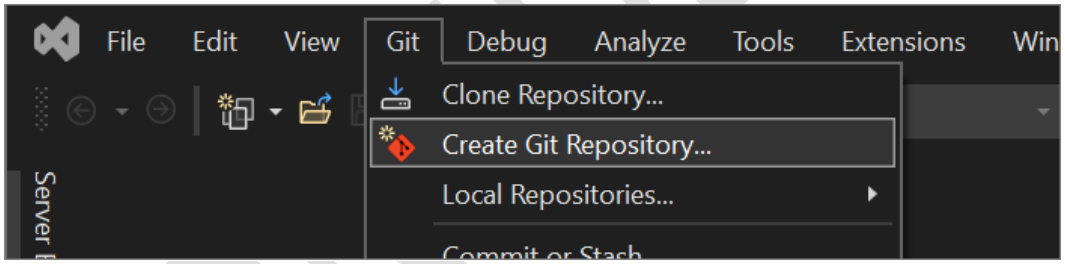
Create a GitHub repo

1. Open Visual Studio, and then select **Create a new project**.

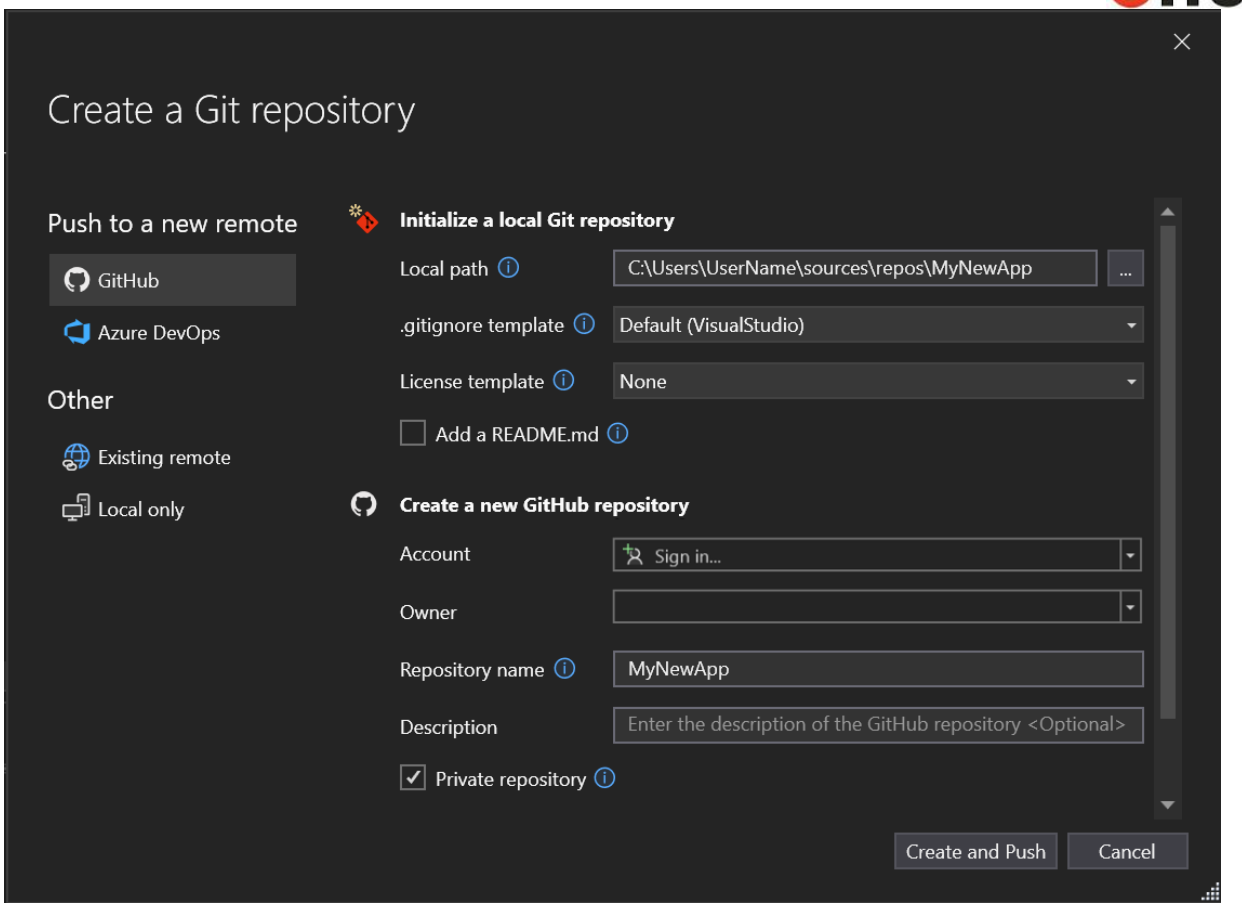
   Tip

   If you don't already have a project in Visual Studio to add to a repo, you can quickly **create a new C# console app** and name it **MyNewApp**. Visual Studio populates your new app with default "*Hello, World!*" code.

2. From the **Git** menu, select **Create Git Repository**.



3. In the **Create a Git repository** dialog, under the **Push to a new remote** section, choose **GitHub**.
4. In the **Create a new GitHub repository** section of the **Create a Git repository** dialog, enter the name of the repo you want to create. (If you haven't yet signed in to your GitHub account, you can do so from this screen, too.)



Under **Initialize a local Git Repository**, you can use the **.gitignore template** option to specify any intentionally untracked files that you want Git to ignore. To learn more about .gitignore, see Ignoring files. And to learn more about licensing, see Licensing a repository.

5. After you sign in and enter your repo info, select the **Create and Push** button to create your repo and add your app.

24. Stored Procedure:

Problem Statement

This stored procedure retrieves a list of all products from the Production.Product table:

Solution

```
USE AdventureWorks2019;
GO

-- Create a stored procedure without parameters
CREATE PROCEDURE GetAllProducts
AS
BEGIN
   -- Your SQL query to retrieve all products
   SELECT
      ProductID,
      Name,
      Color,
      StandardCost,
```

```
    ListPrice
  FROM
    Production.Product;
END;
GO
```

In this example:

USE AdventureWorks2019; specifies the database to be used.

CREATE PROCEDURE GetAllProducts declares the start of the stored procedure named GetAllProducts.

BEGIN and END enclose the body of the stored procedure.

Inside the procedure, there is a simple SELECT query to retrieve product information from the Production.Product table.

To execute this stored procedure and get the list of all products, you can use the following syntax:

```
-- Execute the stored procedure
EXEC GetAllProducts;
```

This stored procedure doesn't have any parameters, and it simply returns the information for all products in the Production.Product table.

25. WebAPI using EFCore CodeFirst Approach:

Setting Up the ASP.NET Core Web API Project

As a first step, set up an ASP.NET Core Web API Project.

reate a new project called EFCoreCodeFirstSample

Configuring EF Core

the next step is to set up the **EF Core**.

Following are the steps for configuring the **EF Core**:

- Defining the Model
- Creating a Context File
- Generating the Database from Code Using Migrations

Defining the Model

First, define the model. We will start by creating a folderModels within the root of the application.

add a new class Employee.cs inside:

```
using System;
using System.ComponentModel.DataAnnotations;
using System.ComponentModel.DataAnnotations.Schema;
namespace EFCoreCodeFirstSample.Models
{
```

```
public class Employee
{
[Key]
[DatabaseGenerated(DatabaseGeneratedOption.Identity)]
public long EmployeeId { get; set; }
public string FirstName { get; set; }
public string LastName { get; set; }
public DateTime DateOfBirth { get; set; }
public string PhoneNumber { get; set; }
public string Email { get; set; }
}
}
```

The code above defines the classEmployee with some properties. Additionally, we have decorated the EmployeeId property with Key and DatabaseGenerated attributes. We did this because we will be converting this class into a database table and the columnEmployeeId will serve as our primary key with the auto-incremented identity.

Creating a Context File

As the next step, create a context class, define database connection and register the context

Following the above article, define the context file EmployeeContext.cs(it requires installed Microsoft.EntityFrameworkCore 3.0.0 package):

```
using Microsoft.EntityFrameworkCore;
namespace EFCoreCodeFirstSample.Models
{
public class EmployeeContext : DbContext
{
public EmployeeContext(DbContextOptions options)
: base(options)
{
}
public DbSet<Employee> Employees { get; set; }
}
}
```

and define the database connection in the appsettings.json file as:

```
{
"Logging": {
"LogLevel": {
"Default": "Information",
```

```
"Microsoft": "Warning",
"Microsoft.Hosting.Lifetime": "Information"
}
},
"ConnectionString": {
"EmployeeDB": "server=MY_SERVER;database=EmployeeDB;User ID=MY_USER;password=MY_PASSWORD;"
},
"AllowedHosts": "*"
}
```

Of course, modify the ConnectionStringproperty to match with that of ours.

Then install the Microsoft.EntityFrameworkCore.SqlServer package and register our context in the Startup.cs:

```
public void ConfigureServices(IServiceCollection services)
{
services.AddDbContext<EmployeeContext>(opts =>
opts.UseSqlServer(Configuration["ConnectionString:EmployeeDB"]));
services.AddControllers();
}
```

Generating the Database from Code Using Migrations

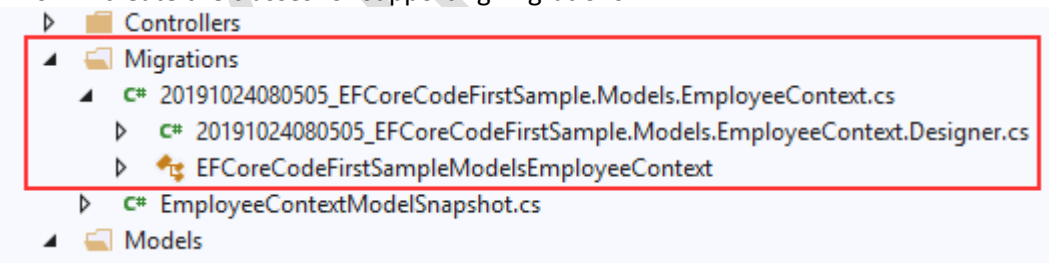
Our next step is to add Code-First Migrations. Migrations automate the creation of database based on our Model.

The EF Core packages required for migration will be added with .NET Core project setup.

install the Microsoft.EntityFrameworkCore.Tools package and run the following command in the Package Manager console:

PM> Add-Migration EFCoreCodeFirstSample.Models.EmployeeContext

This will create the classes for supporting migrations.



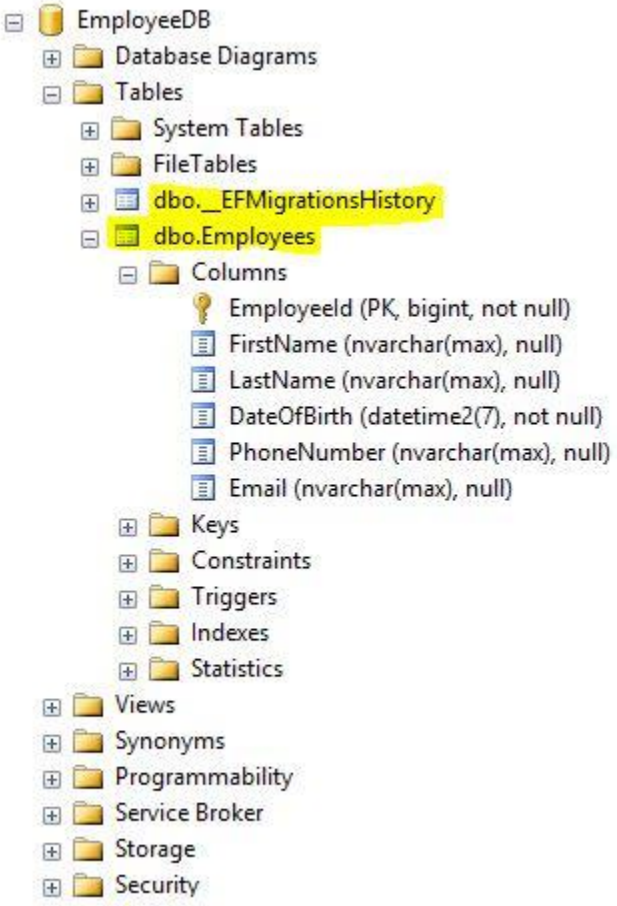
Now apply those changes to the database.

Run the following command:

PM> update-database

This will update the database based on our models.

Now verify that the database and tables are created by opening SQL Server Management Studio or Visual Studio Server Explorer:



We can see the database EmployeeDBis created with a table Employeeswhich contains the columns based on the fields we defined in our model.

Each time we make changes to our entities and do a migration, we can see new migration files created in our solution and new entries in the table__EFMigrationsHistory.

When using the EF Core Code-First approach the best practice is to make all modifications to the database through the model and then update the database by doing the migration. Ideally, **we should not make any manual changes to the database.**

With that, the EF Core setup is complete.

Seeding Data, Reverting Migrations and Creating DB Scripts

Seeding Data

Data seeding allows us to provide initial data during the creation of a database. Then, EF Core migrations will automatically determine what insert, update or delete operations need to be applied when upgrading the database to a new version of the model.

Create seed data now. For this, we need to override the OnModelCreating method in the EmployeeContext class:

```
protected override void OnModelCreating(ModelBuilder modelBuilder)
{
modelBuilder.Entity<Employee>().HasData(new Employee
```

```
{
EmployeeId = 1,
FirstName = "Uncle",
LastName = "Bob",
Email = "uncle.bob@gmail.com",
DateOfBirth = new DateTime(1979, 04, 25),
PhoneNumber = "999-888-7777"
}, new Employee
{
EmployeeId = 2,
FirstName = "Jan",
LastName = "Kirsten",
Email = "jan.kirsten@gmail.com",
DateOfBirth = new DateTime(1981, 07, 13),
PhoneNumber = "111-222-3333"
});
}
```

Here we have provided two Employee records that will be inserted into the database as part of the migration.

run the migration commands once again:

Add-Migration EFCoreCodeFirstSample.Models.EmployeeContextSeed

update-database

This will create a new migration file in our Migrations folder and update the database with the seed data we provided:

Now the Employee table in our database will look like this:

```
/****** Script for SelectTopNRows command from SSMS  ******/
SELECT TOP 1000 [EmployeeId]
      ,[FirstName]
      ,[LastName]
      ,[DateOfBirth]
      ,[PhoneNumber]
      ,[Email]
  FROM [EmployeeDB].[dbo].[Employees]
```

Results | Messages

EmployeeId	FirstName	LastName	DateOfBirth	PhoneNumber	Email
1	Uncle	Bob	1979-04-25 00:00:00.0000000	999-888-7777	uncle.bob@gmail.com
2	Jan	Kirsten	1981-07-13 00:00:00.0000000	111-222-3333	jan.kirsten@gmail.com

Reverting Migrations

After making changes to our **EF Core** model, the database schema will be out of sync. To bring it to sync with the model, add another migration.

add a new property Gender in our employee model and then do a migration.

It is a good practice to give meaningful names to the migration like a commit message in a version control system. For example, if we add a new field Gender to the Employee model, we may give a name like AddEmployeeGender.

Add-Migration EFCoreCodeFirstSample.Models.AddEmployeeGender

Sometimes we add a migration and then realize we need to make additional changes to our model before applying it. To remove the last migration, we can use the command:

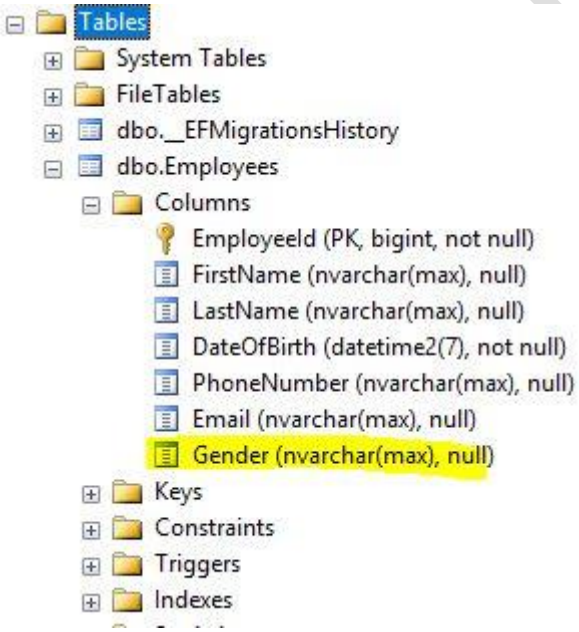
Remove-Migration

If we already applied a migration (or several migrations) to the database but need to revert it, we can use the same command to apply migrations, but specify the name of the migration we want to roll back to.

say we already applied the migration to add the Gender column to the database by using the below command.

update-database

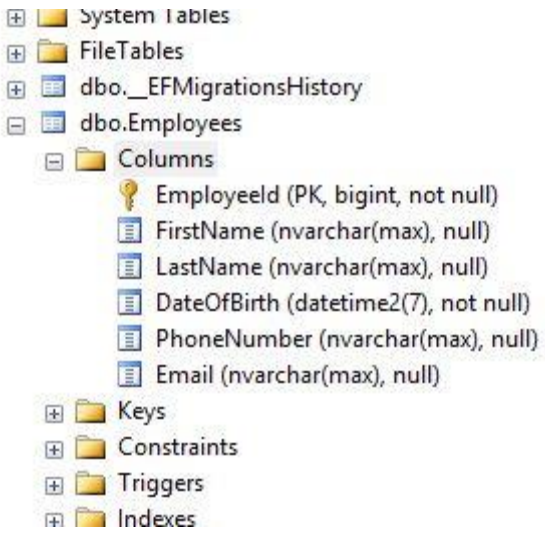
Now we can see the new column Gender added to the Employee table:



Now to revert this migration. We can use the same command by specifying the name of the previous migration:

update-database EFCoreCodeFirstSample.Models.EmployeeContextSeed

Once this is executed, we can see that the column Gender is removed from the Employee table:



We should remove the Gender property from the Employee class as well.

Creating DB Scripts

While deploying our migrations to a production database, it's useful to generate a SQL script. We can further tune the script to match the production database. Also, we can use the script along with various deployment tools.

The command to generate the script is:

Script-Migration

Once we apply this command, we can see a SQL script generated with all changes related to our migrations.

Creating the Repository

Now that we have configured the EF Core, we need a mechanism to access the data context from our API. Directly accessing the context methods from the API controller is a bad practice and we should avoid that.

implement a simple data repository using the repository pattern. We have explained this pattern in detail in one of our other articles: Implementing the repository pattern.

add a new folder under Models and name itRepository. Then create a new interface calledIDataRepository:

```
namespace EFCoreCodeFirstSample.Models.Repository
{
public interface IDataRepository<TEntity>
{
IEnumerable<TEntity> GetAll();
TEntity Get(long id);
void Add(TEntity entity);
void Update(TEntity dbEntity, TEntity entity);
void Delete(TEntity entity);
}
```

```
}
```

We will later inject this interface into our API Controller and API will be communicating with the data context using this interface.

Next, create a concrete class that implements the interfaceIDataRepository. add a new folder under Models calledDataManager. Then create a new class EmployeeManager:

```
using System.Collections.Generic;
using System.Linq;
using EFCoreCodeFirstSample.Models.Repository;
namespace EFCoreCodeFirstSample.Models.DataManager
{
public class EmployeeManager : IDataRepository<Employee>
{
readonly EmployeeContext _employeeContext;
public EmployeeManager(EmployeeContext context)
{
_employeeContext = context;
}
public IEnumerable<Employee> GetAll()
{
return _employeeContext.Employees.ToList();
}
public Employee Get(long id)
{
return _employeeContext.Employees
.FirstOrDefault(e => e.EmployeeId == id);
}
public void Add(Employee entity)
{
_employeeContext.Employees.Add(entity);
_employeeContext.SaveChanges();
}
public void Update(Employee employee, Employee entity)
{
employee.FirstName = entity.FirstName;
employee.LastName = entity.LastName;
employee.Email = entity.Email;
employee.DateOfBirth = entity.DateOfBirth;
```

```
employee.PhoneNumber = entity.PhoneNumber;
_employeeContext.SaveChanges();
}
public void Delete(Employee employee)
{
_employeeContext.Employees.Remove(employee);
_employeeContext.SaveChanges();
}
}
}
```

The classEmployeeManager handles all database operations related to the employee. The purpose of this class is to separate the actual data operations logic from our API Controller.

This class has the following methods for supporting CRUD operations:

GetAll() – Gets all employee records from the database.

Get() – Gets a specific employee record from the database by passing an Id.

Add() – Creates a new employee record in the database.

Update() – Updates a specific employee record in the database.

Delete() – Removes a specific employee record from the database based on the Id.

As a next step, configure the repository using dependency injection. This can be done in the ConfigureServices method in the Startup.cs as below:

```
public void ConfigureServices(IServiceCollection services)
{
services.AddDbContext<EmployeeContext>(opts =>
opts.UseSqlServer(Configuration["ConnectionString:EmployeeDB"]));
services.AddScoped<IDataRepository<Employee>, EmployeeManager>();
services.AddControllers();
}
```

Creating the API Controller

Now that our DataManager is all set, Create the API Controller and create the endpoints for handling CRUD operations.

create the EmployeeController class in the Controllersfolder as below:

```
using System.Collections.Generic;
using EFCoreCodeFirstSample.Models;
using EFCoreCodeFirstSample.Models.Repository;
using Microsoft.AspNetCore.Mvc;
namespace EFCoreCodeFirstSample.Controllers
{
```

```
[Route("api/employee")]
[ApiController]
public class EmployeeController : ControllerBase
{
private readonly IDataRepository<Employee> _dataRepository;
public EmployeeController(IDataRepository<Employee> dataRepository)
{
_dataRepository = dataRepository;
}
// GET: api/Employee
[HttpGet]
public IActionResult Get()
{
IEnumerable<Employee> employees = _dataRepository.GetAll();
return Ok(employees);
}
// GET: api/Employee/5
[HttpGet("{id}", Name = "Get")]
public IActionResult Get(long id)
{
Employee employee = _dataRepository.Get(id);
if (employee == null)
{
return NotFound("The Employee record couldn't be found.");
}
return Ok(employee);
}
// POST: api/Employee
[HttpPost]
public IActionResult Post([FromBody] Employee employee)
{
if (employee == null)
{
return BadRequest("Employee is null.");
}
_dataRepository.Add(employee);
return CreatedAtRoute(
"Get",
```

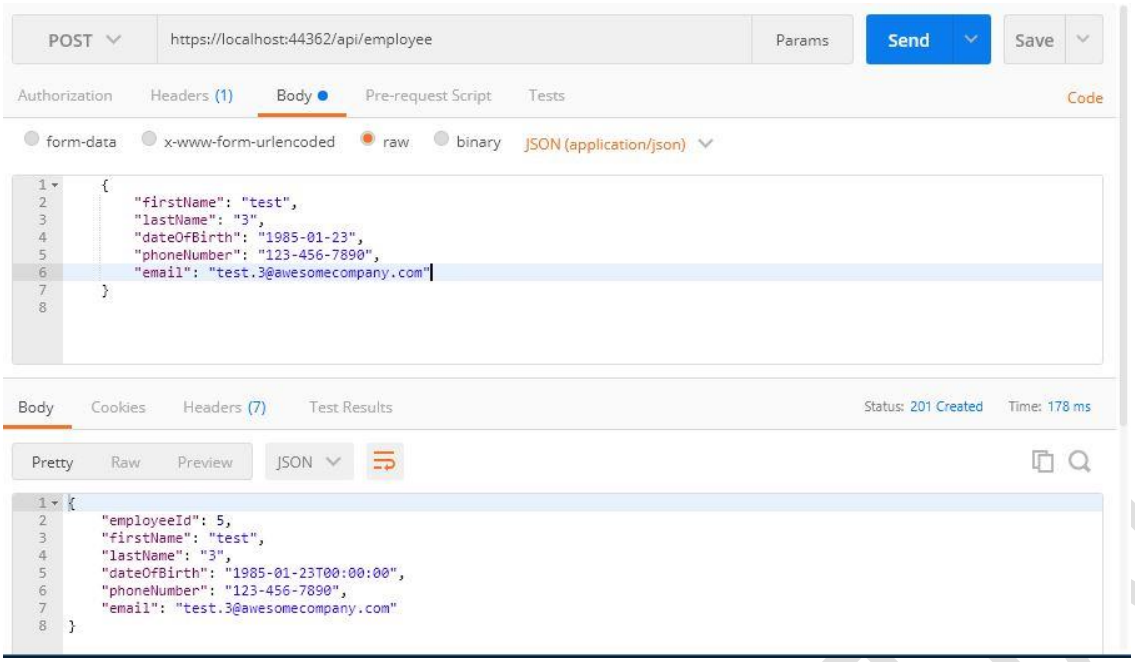
```
new { Id = employee.EmployeeId },
employee);
}
// PUT: api/Employee/5
[HttpPut("{id}")]
public IActionResult Put(long id, [FromBody] Employee employee)
{
if (employee == null)
{
return BadRequest("Employee is null.");
}
Employee employeeToUpdate = _dataRepository.Get(id);
if (employeeToUpdate == null)
{
return NotFound("The Employee record couldn't be found.");
}
_dataRepository.Update(employeeToUpdate, employee);
return NoContent();
}
// DELETE: api/Employee/5
[HttpDelete("{id}")]
public IActionResult Delete(long id)
{
Employee employee = _dataRepository.Get(id);
if (employee == null)
{
return NotFound("The Employee record couldn't be found.");
}
_dataRepository.Delete(employee);
return NoContent();
}
}
}
```

successfully created a Web API controller with endpoints for handling CRUD operations.

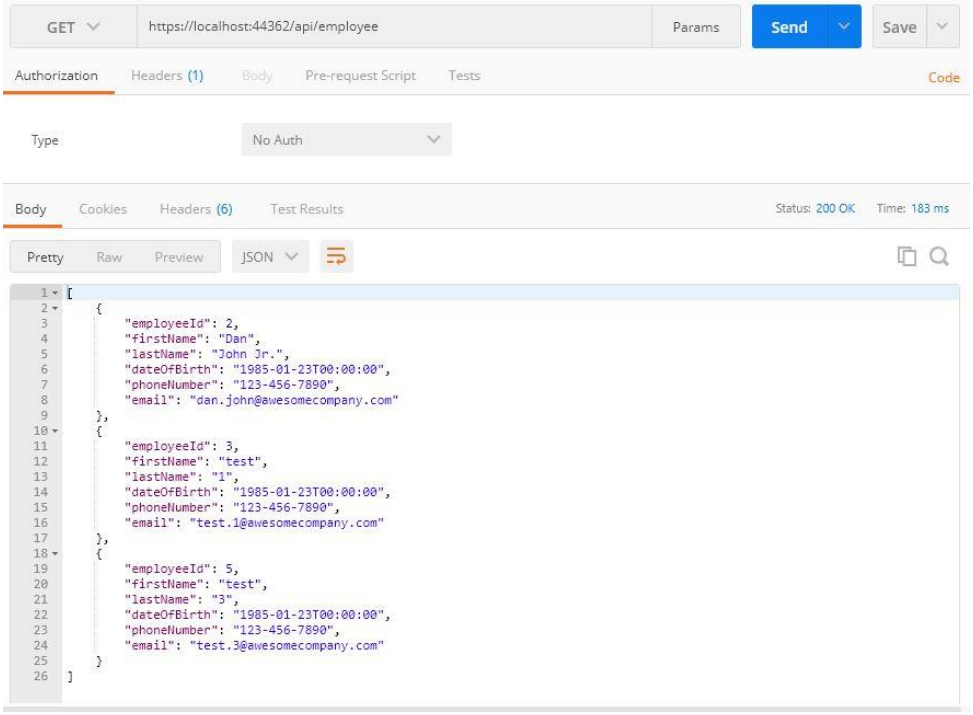
Testing the API

do a quick round of testing around our API endpoints using Postman.

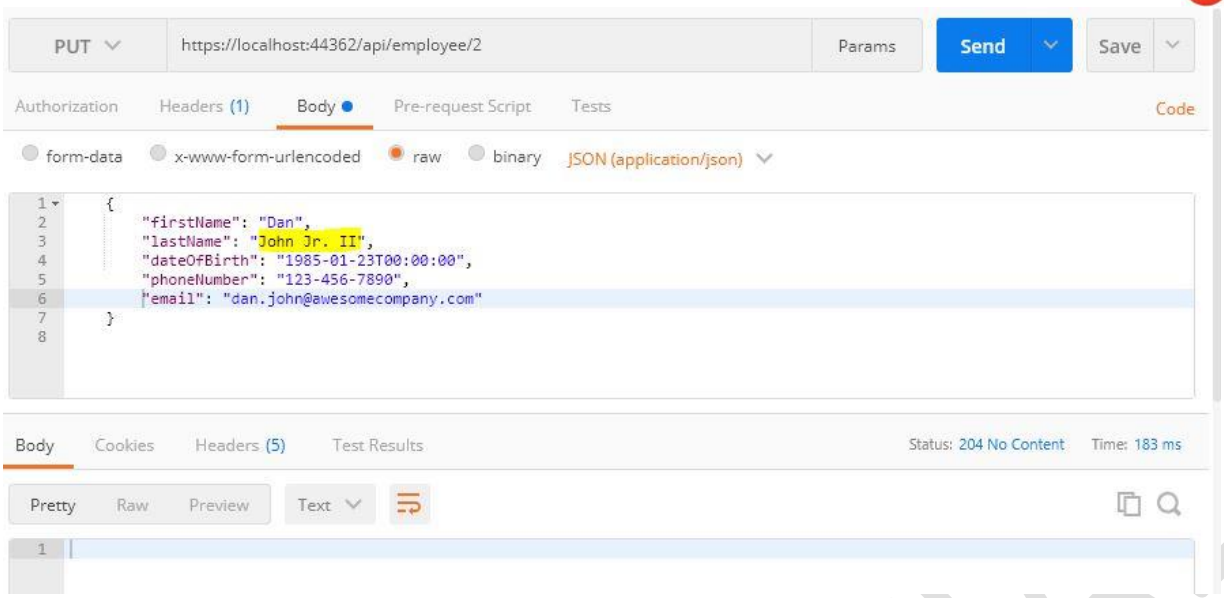
First, create a new Employee using a Post request:



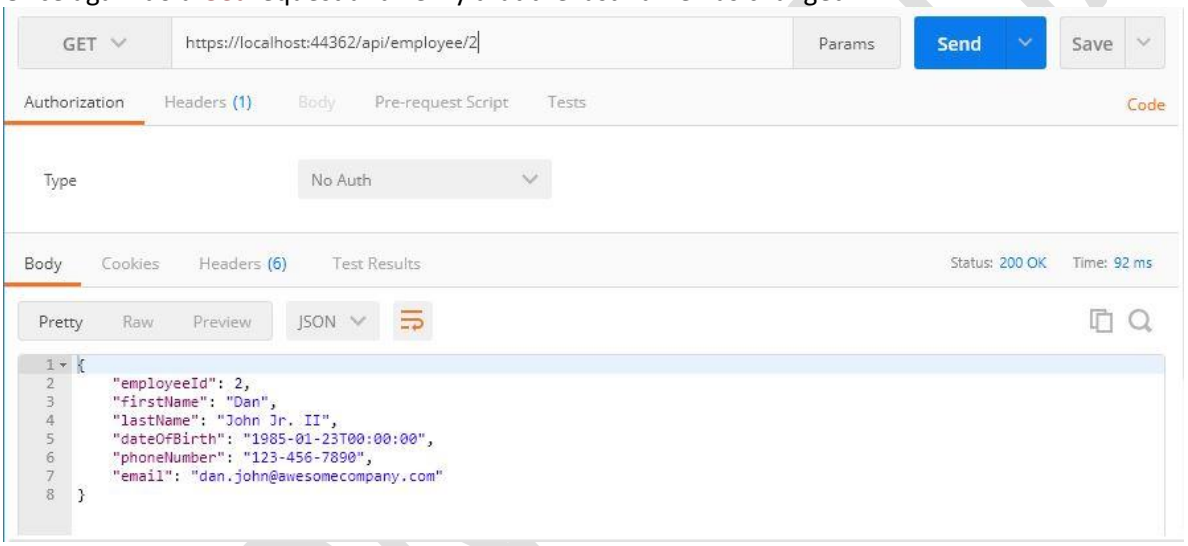
Next, do a Get request to get all Employees. We can see the new Employee record which was created in the previous request:



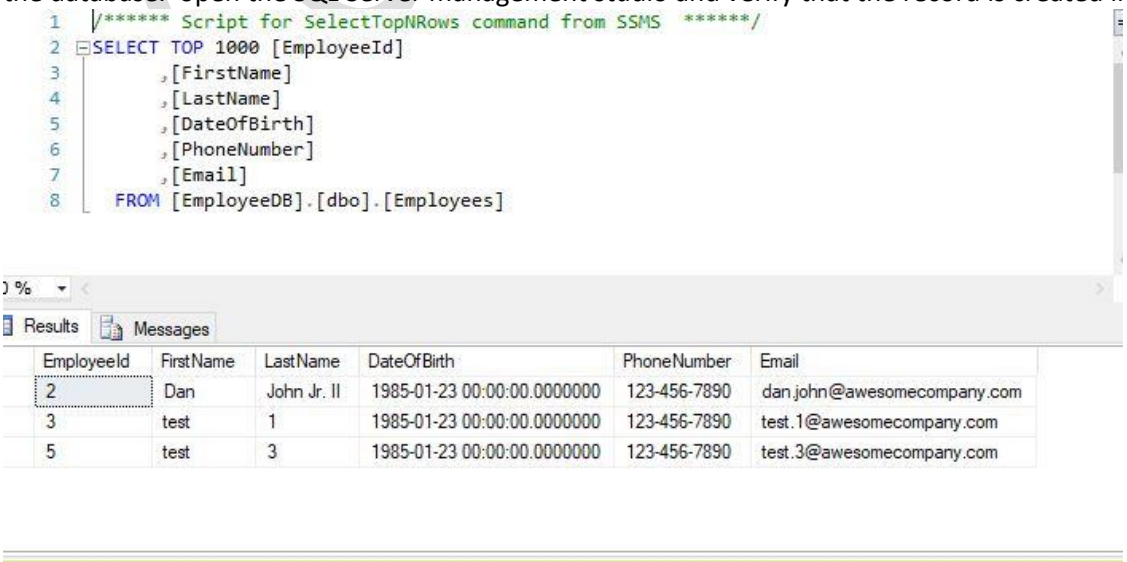
Now, do a Put request to test the update functionality by changing the last name:



Once again do a Get request and verify that the last name has changed:



Now that we have successfully tested the API endpoints, verify that the changes we made are actually persisted in the database. open the SQL Server management studio and verify that the record is created in the Employee table:



Conclusion

In this , we have learned the following topics.

- EF Core Code-First approach and when to use it
- Setting up a .NET Core Web API project with EF Core Code-First approach
- Creating a database from code by using migrations
- Setting up a repository to handle communication between API and the data context
- Create API endpoints for handling CRUD operations and testing them

26. WebAPI using EFCore DBFirst Approach:

Creating a Database and Tables

As the first step, we are going to create the database and tables.

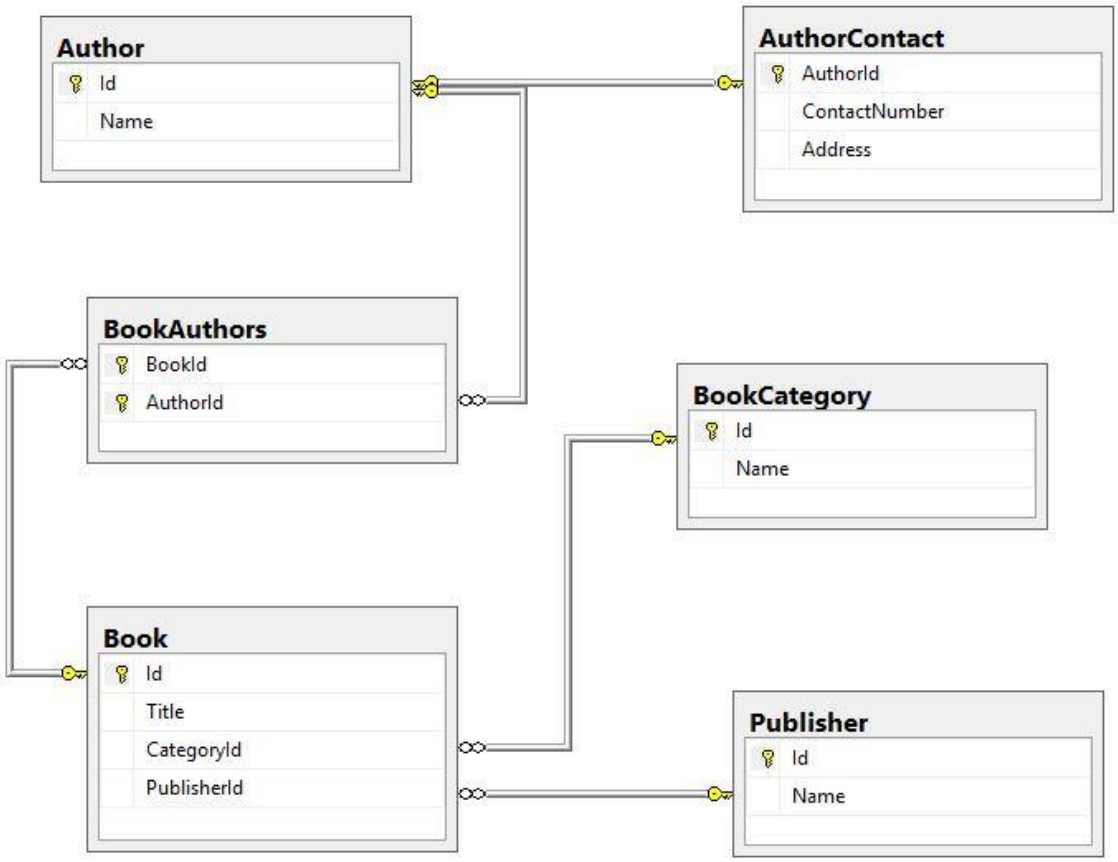
Create a database to manage books. We are going to create tables for storing information about Books, Authors, Publishers etc. and establish relationships between them.

This is the complete SQL script for creating database tables and relationships.

```
CREATE DATABASE BookStore
GO
USE BookStore
GO
CREATE TABLE Author
(
Id BIGINT IDENTITY(1, 1) NOT NULL,
NAME NVARCHAR(50) NOT NULL,
PRIMARY KEY (Id)
)
GO
CREATE TABLE AuthorContact
(
AuthorId BIGINT NOT NULL,
ContactNumber NVARCHAR(15) NULL,
Address NVARCHAR(100) NULL,
PRIMARY KEY (AuthorId),
FOREIGN KEY (AuthorId) REFERENCES Author(Id)
)
GO
CREATE TABLE BookCategory
(
Id BIGINT IDENTITY(1, 1) NOT NULL,
```

```
NAME NVARCHAR(50) NOT NULL,
PRIMARY KEY (Id)
)
GO
CREATE TABLE Publisher
(
Id BIGINT IDENTITY(1, 1) NOT NULL,
NAME NVARCHAR(100) NOT NULL,
PRIMARY KEY (Id)
)
GO
CREATE TABLE Book
(
Id BIGINT IDENTITY(1, 1) NOT NULL,
Title NVARCHAR(100) NOT NULL,
CategoryId BIGINT NOT NULL,
PublisherId BIGINT NOT NULL,
PRIMARY KEY (Id),
FOREIGN KEY (CategoryId) REFERENCES BookCategory(Id),
FOREIGN KEY (PublisherId) REFERENCES Publisher(Id)
)
GO
CREATE TABLE BookAuthors
(
BookId BIGINT NOT NULL,
AuthorId BIGINT NOT NULL
PRIMARY KEY (BookId, AuthorId),
FOREIGN KEY (BookId) REFERENCES Book(Id),
FOREIGN KEY (AuthorId) REFERENCES Author(Id)
)
```

After running the script, we can see the tables and relationships created as below:



Database design explained

Tables:

Author– Stores the information about the authors.

AuthorContact– Contains the contact information about the authors.

Book– Stores the information about the books.

Publisher– Keeps the information about the publishers.

BookCategory– Keeps the master list of all the categories.

BookAuthors– Represents the mapping between the books and the authors.

Relationships:

Let's take a look at how we implement the different types of relationships in our database design.

One-to-One(1:1)

In the above design, AuthorandAuthorContact have a 1:1 relationship between them. Each entry in theAuthor table has a corresponding entry in theAuthorContact table. They are related by theAuthorId foreign key.

This type of relationship is not very common. We could also keep the author contact information in theAuthor table. But in certain scenarios, there could be some valid reasons to split out information into different tables like security, performance etc.

One-to-Many(1:N)

In the above design, PublisherandBook have a 1:N relationship between them. A publisher can publish many books, but a book can have only one publisher. They are related by thePublisherId foreign key.

This is the most common type of relationship in any database.

Many-to-Many(M:N)

In the above design, BookandAuthor have an M:N relationships between them. A book can have many authors and at the same time, an author can write many books. They are related by an intermediate tableBookAuthors. This is also called an associative or junction table.

We can translate an M:N relationship to two 1:N relationships, but linked by an intermediary table.

Inserting Test Data

Now that we have created our tables and established relationships between them, let's insert some test data into them. Let's use the below DB script to insert data:

```
INSERT INTO BookCategory
VALUES
('Fantasy Fiction'),
('Spirituality'),
('Fiction'),
('Science Fiction')
INSERT INTO Publisher
VALUES
('HarperCollins'),
('New World Library'),
('Oneworld Publications')
INSERT INTO Author
VALUES
('Paulo Coelho'),
('Eckhart Tolle'),
('Amie Kaufman'),
('Jay Kristoff')
INSERT INTO AuthorContact
VALUES
(1, '111-222-3333', '133 salas 601 / 602, Rio de Janeiro 22070-010. BRAZIL'),
(2, '444-555-6666', '933 Seymour St, Vancouver, BC V6B 6L6, Canada'),
(3, '777-888-9999', 'Mentone 3194. Victoria. AUSTRALIA'),
(4, '222-333-4444', '234 Collins Street, Melbourne, VIC, AUSTRALIA')
INSERT INTO Book
VALUES
('The Alchemist', 1, 1),
('The Power of Now', 2, 2),
```

```
('Eleven Minutes', 3, 1),
('Illuminae', 4, 3)
INSERT INTO BookAuthors
VALUES
(1,1),
(2,2),
(3,1),
(4,3),
(4,4)
```

After running the above insert script, our database tables will look like this

BookCategory

Id	Name
1	Fantasy Fiction
2	Spirituality
3	Fiction
4	Science Fiction

Publishers

Id	Name
1	HarperCollins
2	New World Library
3	Oneworld Publications

Authors

Id	Name
1	Paulo Coelho
2	Eckhart Tolle
3	Amie Kaufman
4	Jay Kristoff

AuthorContact

AuthorId	ContactNumber	Address
1	111-222-3333	133 salas 601 / 602, Rio de Janeiro 22070-010. B...
2	444-555-6666	933 Seymour St, Vancouver, BC V6B 6L6, Canada
3	777-888-9999	Mentone 3194. Victoria. AUSTRALIA
4	222-333-4444	234 Collins Street, Melbourne, VIC, AUSTRALIA

Books

Id	Title	CategoryId	PublisherId
1	The Alchemist	1	1
2	The Power of Now	2	2
3	Eleven Minutes	3	1
4	Illuminae	4	3

BookAuthors

BookId	AuthorId
1	1
2	2
3	1
4	3
4	4

Data Modelling – Creating Models and a Context

So, now we have our database tables with data. Let's model our entities based on those.

As a first step, set up an ASP.NET Core Web API Project. Create a new project calledEFCoreDatabaseFirstSample**.**

Creating Models

Now it's time to create the EF model based on our existing database.

Go to **Tools –> NuGet Package Manager –> Package Manager Console**

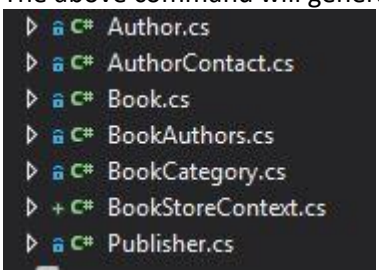
First, we need to install the following packages :

```
Install-Package Microsoft.EntityFrameworkCore.Tools
Install-Package Microsoft.EntityFrameworkCore.SqlServer
```

Then, we can create the models from the existing database using Scaffold-DbContext command:

```
Scaffold-DbContext "Server=.;Database=BookStore;Trusted_Connection=True;"
Microsoft.EntityFrameworkCore.SqlServer -OutputDir Models
```

The above command will generate the following classes:



BookStoreContext is the DB context class and other classes are the models.

Now, let's look at how EF Core represents the relationships.

One-to-One(1:1)

Let's take a look at the Author class:

```
public partial class Author
{
public Author()
{
BookAuthors = new HashSet<BookAuthors>();
}
public long Id { get; set; }
public string Name { get; set; }
public virtual AuthorContact AuthorContact { get; set; }
public virtual ICollection<BookAuthors> BookAuthors { get; set; }
}
```

Remember that theAuthor has a 1:1 relationship with theAuthorContact. To represent this we have anAuthorContactproperty in theAuthor class. This is called the **Navigation Property.**

One-to-Many(1:N)

Let's take a look at the Publisher & Book classes:

```
public partial class Publisher
{
```

```
public Publisher()
{
Books = new HashSet<Book>();
}
public long Id { get; set; }
public string Name { get; set; }
public virtual ICollection<Book> Books { get; set; }
}
public partial class Book
{
public Book()
{
BookAuthors = new HashSet<BookAuthors>();
}
public long Id { get; set; }
public string Title { get; set; }
public long CategoryId { get; set; }
public long PublisherId { get; set; }
public virtual BookCategory Category { get; set; }
public virtual Publisher Publisher { get; set; }
public virtual ICollection<BookAuthors> BookAuthors { get; set; }
}
```

Remember that thePublisherhas a 1:N relationship with theBook.

Here, the Publisher is called the **Principal Entity** and the Book is called **Dependent Entity**.

Publisher.PublisherId is the **Principal Key** and Book.PublisherId is the **Foreign Key.**

Publisher.Books is the **Collection Navigation** property.

Book.Publisher is the **Reference Navigation** property.

Many-to-Many(M:N)

Note: As of now, EF Core does not support many-to-many relationships without using an entity class for representing the join table. However, we can represent it by using an entity class for the join table. We could then map two separate one-to-many relationships.

Let's take a look at the Book, Author & BookAuthors classes. *(Book and Author classes are already shown above):*

```
public partial class BookAuthors
{
public long BookId { get; set; }
public long AuthorId { get; set; }
public virtual Author Author { get; set; }
```

```
public virtual Book Book { get; set; }
}
```

We can see that both the Book and the Author has a collection navigation propertyBookAuthors. We have established the M:N relationship between theBook and the Author by these two 1:N relationships.

Creating a Repository

Add a new folder under Models and name it Repository. We'll then create a new interface called IDataRepository:

```
public interface IDataRepository<TEntity, TDto>
{
IEnumerable<TEntity> GetAll();
TEntity Get(long id);
TDto GetDto(long id);
void Add(TEntity entity);
void Update(TEntity entityToUpdate, TEntity entity);
void Delete(TEntity entity);
}
```

We will later inject this interface into our controller. Then the API will communicate with the data context using this interface. Of course, we are going to register all the repo services in the Startup class, as you can find out by your self in our source code.

Next, let's create concrete classes that implement theIDataRepository interface. We'll add a new folder under Models called DataManager.

Querying & Loading Related Data

EF Core uses navigation properties in our model to load related entities. We use three common ORM patterns for loading related data.

When we use eager loading, we load the related data from the database as part of the initial query.

Explicit loading means that we load the related data explicitly from the database at a later time.

Lazy loading is a way of loading the related data from the database when we access the navigation property.

Eager loading

We can use theIncludemethod to specify related data that need to be included in the query results. In the following example, the Authors that are returned in the results will have their AuthorContacts property auto-populated.

Let's add a new class AuthorDataManager which implements theIDataRepository in the DataManager folder, and register it in the Startup class.

We'll then implement the GetAll():

```
public IEnumerable<Author> GetAll()
{
return _bookStoreContext.Author
.Include(author => author.AuthorContact)
```

```
.ToList();
}
```

The above code loads all the authors with their contact details at once since we are using eager loading. We shall verify this later when we test it.

Explicit loading

We can explicitly load a navigation property using the DbContext.Entry().

Let's add a new class BookDataManager which implements the IDataRepository interface and register it in the Startup class as well.

We'll then implement the Get()method:

```
public Book Get(long id)
{
_bookStoreContext.ChangeTracker.LazyLoadingEnabled = false;
var book = _bookStoreContext.Book
.SingleOrDefault(b => b.Id == id);
if (book == null)
{
return null;
}
_bookStoreContext.Entry(book)
.Collection(b => b.BookAuthors)
.Load();
_bookStoreContext.Entry(book)
.Reference(b => b.Publisher)
.Load();
return book;
}
```

The above code is used to get the details of a Book. See how we are explicitly loading the list of BookAuthors and Publisher later. We'll verify the explicit loading behavior later when we test this functionality.

Lazy loading

The simplest way to use lazy-loading is by installing the Microsoft.EntityFrameworkCore.Proxies package and enabling it with a call to UseLazyLoadingProxies.

This is shown in the below code

```
protected override void OnConfiguring(DbContextOptionsBuilder optionsBuilder)
{
if (!optionsBuilder.IsConfigured)
{
optionsBuilder
```

```
.UseLazyLoadingProxies()
.UseSqlServer("Server=.;Database=BookStore;Trusted_Connection=True;");
}
}
```

EF Core will then enable lazy loading for any navigation property that can be overridden. Only thing is that it must be virtual and on a class that can be inherited from.

For example, in the below Authorclass, the BookAuthors navigation property will be lazy-loaded:

```
public partial class Author
{
public long Id { get; set; }
public string Name { get; set; }
public virtual AuthorContact AuthorContact { get; set; }
public virtual ICollection<BookAuthors> BookAuthors { get; set; }
}
```

Let's then disable lazy-loading at a context level. This helps to avoid circular referencing issues:

```
public BookStoreContext(DbContextOptions<BooksStoreContext> options)
: base(options)
{
ChangeTracker.LazyLoadingEnabled = false;
}
```

We'll enable lazy-loading explicitly when we need to utilize it.

Let's implement the GetDto() method in the AuthorDataManagerclass:

```
public AuthorDto GetDto(long id)
{
_bookStoreContext.ChangeTracker.LazyLoadingEnabled = true;
using (var context = new BookStoreContext())
{
var author = context.Author
.SingleOrDefault(b => b.Id == id);
return AuthorDtoMapper.MapToDto(author);
}
}
public class AuthorDto
{
public AuthorDto()
{
}
```

```
public long Id { get; set; }
public string Name { get; set; }
public AuthorContactDto AuthorContact { get; set; }
}
public static class AuthorDtoMapper
{
public static AuthorDto MapToDto(Author author)
{
return new AuthorDto()
{
Id = author.Id,
Name = author.Name,
AuthorContact = new AuthorContactDto()
{
AuthorId = author.Id,
Address = author.AuthorContact.Address,
ContactNumber = author.AuthorContact.ContactNumber
}
};
}
}
```

In the code above, since we are using lazy loading, only the Author entity will be loaded initially. Later the AuthorContact property will be loaded only when we reference it inside the DTO mapper. We'll verify this behavior later when we test this.

Note: The referenced property can be lazy-loaded only inside the scope of the data context class. Once the context is out of scope, we will no longer have access to those.

Saving Related Data

In this section, we'll explain how we can Add, Update and Delete related entities.

Add

If we create several new related entities, adding one of them to the context will cause the others to be added too.

For example, in the below code, let's implement the Add() method in AuthorDataManager.

This will cause both Author and AuthorContact entities to be created:

```
public void Add(Author entity)
{
_bookStoreContext.Author.Add(entity);
_bookStoreContext.SaveChanges();
}
```

Update

Now let's implement the update. The below code implements the Update() method in AuthorDataManager class:

```
public void Update(Author entityToUpdate, Author entity)
{
entityToUpdate = _bookStoreContext.Author
.Include(a => a.BookAuthors)
.Include(a => a.AuthorContact)
.Single(b => b.Id == entityToUpdate.Id);
entityToUpdate.Name = entity.Name;
entityToUpdate.AuthorContact.Address = entity.AuthorContact.Address;
entityToUpdate.AuthorContact.ContactNumber = entity.AuthorContact.ContactNumber;
var deletedBooks = entityToUpdate.BookAuthors.Except(entity.BookAuthors).ToList();
var addedBooks = entity.BookAuthors.Except(entityToUpdate.BookAuthors).ToList();
deletedBooks.ForEach(bookToDelete =>
entityToUpdate.BookAuthors.Remove(
entityToUpdate.BookAuthors
.First(b => b.BookId == bookToDelete.BookId)));
foreach (var addedBook in addedBooks)
{
_bookStoreContext.Entry(addedBook).State = EntityState.Added;
}
_bookStoreContext.SaveChanges();
}
```

The above code will cause the Author,AuthorContactand BookAuthors entities to be updated. We'll verify this later when we test this.

Delete

Delete operation can be tricky with related entities. There are three actions EF can take when a parent entity is deleted.

- The child can be deleted
- The child's foreign key values can be set to null
- The child remains unchanged

We should configure the DeleteBehavior appropriately based on our application logic. In the below example, let's say when a publisher is deleted, we need the publisher's book also to be deleted.

First, let's configure this in the OnModelCreating method in our context:

```
modelBuilder.Entity<Book>(entity =>
{
entity.Property(e => e.Title)
.IsRequired()
```

```
.HasMaxLength(100);
entity.HasOne(d => d.Publisher)
.WithMany(p => p.Books)
.HasForeignKey(d => d.PublisherId)
.OnDelete(DeleteBehavior.Cascade)
.HasConstraintName("FK_Books_Publishers");
entity.HasOne(d => d.Category)
.WithMany(p => p.Book)
.HasForeignKey(d => d.CategoryId)
.OnDelete(DeleteBehavior.ClientSetNull)
.HasConstraintName("FK_Books_BookCategory");
});
```

Now let's implement the Delete() method in PublisherDataManager class:

```
public void Delete(Publisher entity)
{
_booksStoreContext.Remove(entity);
_booksStoreContext.SaveChanges();
}
```

The above code will delete the Publisherand any related Book entities. We'll verify this later when we test this functionality.

Now we have to register our DataManager classes inside the IOC and configure JSONOptions to ignore circular reference loops.

For that, first, we have to install the NewtonSoftJson package:

```
Install-Package Microsoft.AspNetCore.Mvc.NewtonsoftJson
```

Then, we can configure the services:

```
public void ConfigureServices(IServiceCollection services)
{
services.AddDbContext<BookStoreContext>(opts =>
opts.UseSqlServer(Configuration["ConnectionString:BooksDB"]));
services.AddScoped<IDataRepository<Author, AuthorDto>, AuthorDataManager>();
services.AddScoped<IDataRepository<Book, BookDto>, BookDataManager>();
services.AddScoped<IDataRepository<Publisher, PublisherDto>, PublisherDataManager>();
services.AddControllers()
.AddNewtonsoftJson(
```

```
options => options.SerializerSettings.ReferenceLoopHandling = Newtonsoft.Json.ReferenceLoopHandling.Ignore
);
}
```

This is the appsettings.json file:

```
{
"Logging": {
"LogLevel": {
"Default": "Warning"
}
},
"ConnectionString": {
"BooksDB": "Server=.;Database=BookStore;Trusted_Connection=True"
},
"AllowedHosts": "*"
}
```

Excellent. Now we can move on.

Creating the API Controller

Now that our DataManager is all set, let's create the API Controller and create the endpoints for handling CRUD operations. This is described in detail in one of our other articles: Creating a .NET Core Web API Controller

Following the above article, let's create the AuthorsController,BooksController and PublishersController class in the Controllers folder as shown below.

For keeping things simple and focused, we'll implement only those endpoints required to understand the concepts we discuss in this article.

Let's implement the GetAll,Get,Post and Put method in the AuthorsControllerclass:

```
[Route("api/authors")]
[ApiController]
public class AuthorsController : ControllerBase
{
private readonly IDataRepository<Author, AuthorDto> _dataRepository;
public AuthorsController(IDataRepository<Author, AuthorDto> dataRepository)
{
_dataRepository = dataRepository;
}
// GET: api/Authors
[HttpGet]
```

```
public IActionResult Get()
{
var authors = _dataRepository.GetAll();
return Ok(authors);
}
// GET: api/Authors/5
[HttpGet("{id}", Name = "GetAuthor")]
public IActionResult Get(int id)
{
var author = _dataRepository.GetDto(id);
if (author == null)
{
return NotFound("Author not found.");
}
return Ok(author);
}
// POST: api/Authors
[HttpPost]
public IActionResult Post([FromBody] Author author)
{
if (author is null)
{
return BadRequest("Author is null.");
}
if (!ModelState.IsValid)
{
return BadRequest();
}
_dataRepository.Add(author);
return CreatedAtRoute("GetAuthor", new { Id = author.Id }, null);
}
// PUT: api/Authors/5
[HttpPut("{id}")]
public IActionResult Put(int id, [FromBody] Author author)
{
if (author == null)
{
return BadRequest("Author is null.");
```

```
}
var authorToUpdate = _dataRepository.Get(id);
if (authorToUpdate == null)
{
return NotFound("The Employee record couldn't be found.");
}
if (!ModelState.IsValid)
{
return BadRequest();
}
_dataRepository.Update(authorToUpdate, author);
return NoContent();
}
}
```

Then let's implement the Get() method in the BooksController:

```
[Route("api/books")]
[ApiController]
public class BooksController : ControllerBase
{
private readonly IDataRepository<Book, BookDto> _dataRepository;
public BooksController(IDataRepository<Book, BookDto> dataRepository)
{
_dataRepository = dataRepository;
}
// GET: api/Books/5
[HttpGet("{id}")]
public IActionResult Get(int id)
{
var book = _dataRepository.Get(id);
if (book == null)
{
return NotFound("Book not found.");
}
return Ok(book);
}
}
```

Finally, let's implement the Delete() method in the PublisherController:

```
[Route("api/publishers")]
```

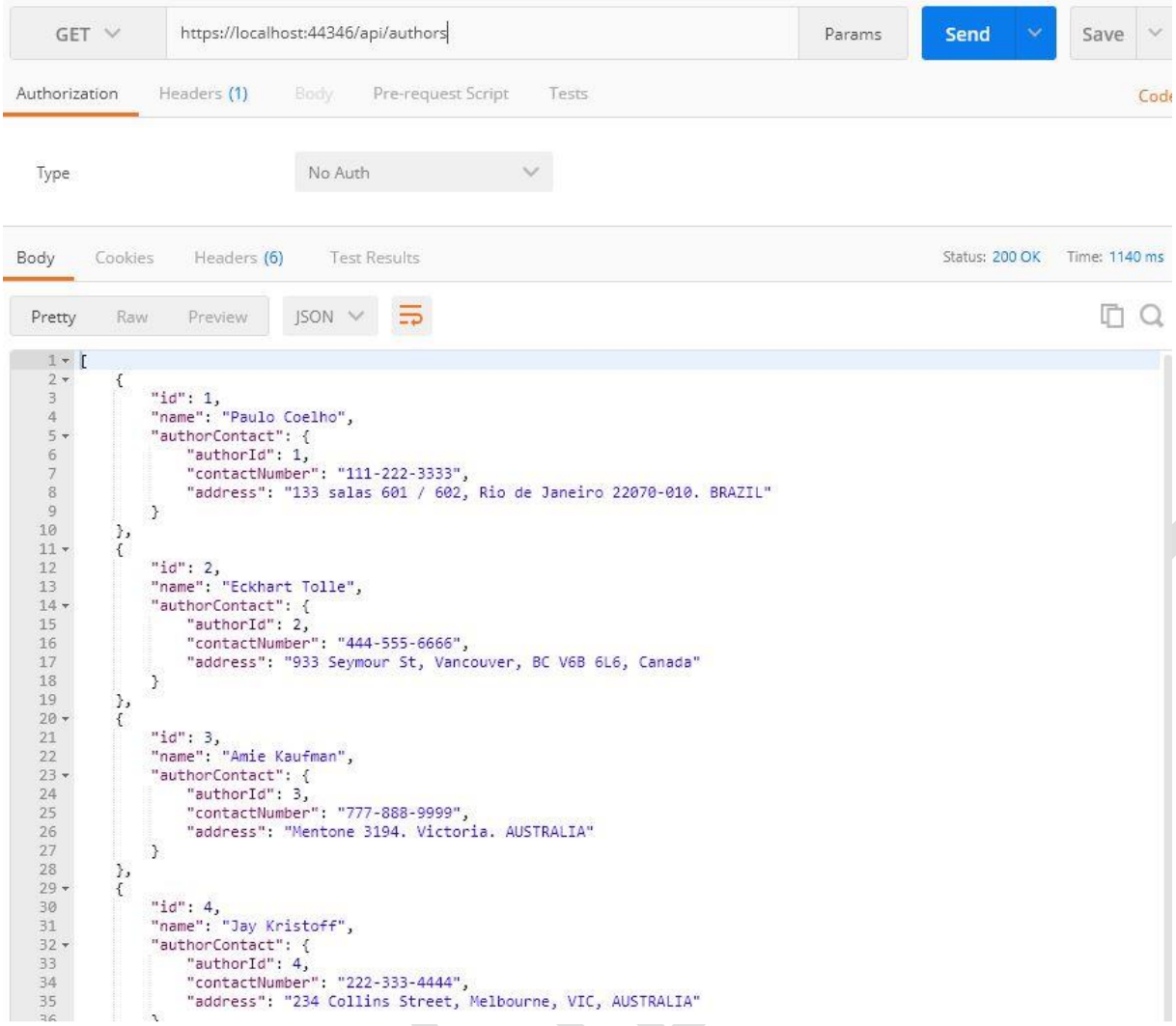
```
[ApiController]
public class PublishersController : ControllerBase
{
private readonly IDataRepository<Publisher, PublisherDto> _dataRepository;
public PublishersController(IDataRepository<Publisher, PublisherDto> dataRepository)
{
_dataRepository = dataRepository;
}
// DELETE: api/ApiWithActions/5
[HttpDelete("{id}")]
public IActionResult Delete(int id)
{
var publisher = _dataRepository.Get(id);
if (publisher == null)
{
return NotFound("The Publisher record couldn't be found.");
}
_dataRepository.Delete(publisher);
return NoContent();
}
}
```

Testing the API

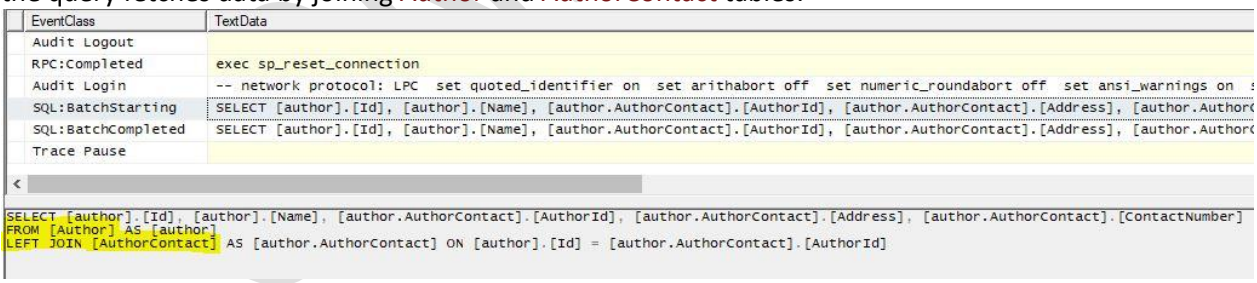
Now we'll test the controller methods using Postman. We'll also verify the results in the database. Later, we'll inspect the actual SQL queries executed in the database using the SQL Server Profiler.

Loading the Data

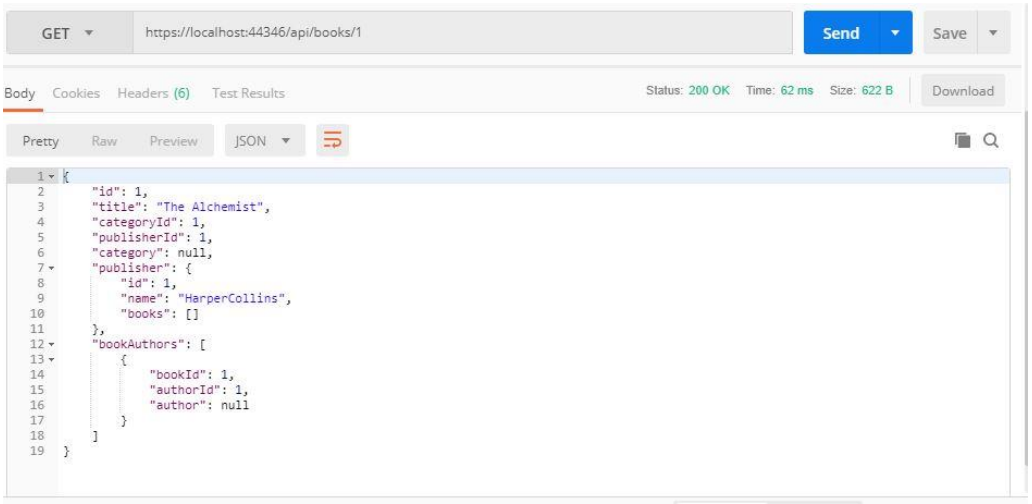
First, let's test the GetAll endpoint of Authors:



Remember that we used eager loading for implementing this functionality. If we look at the Profiler, we can see that the query fetches data by joining Author and AuthorContact tables:

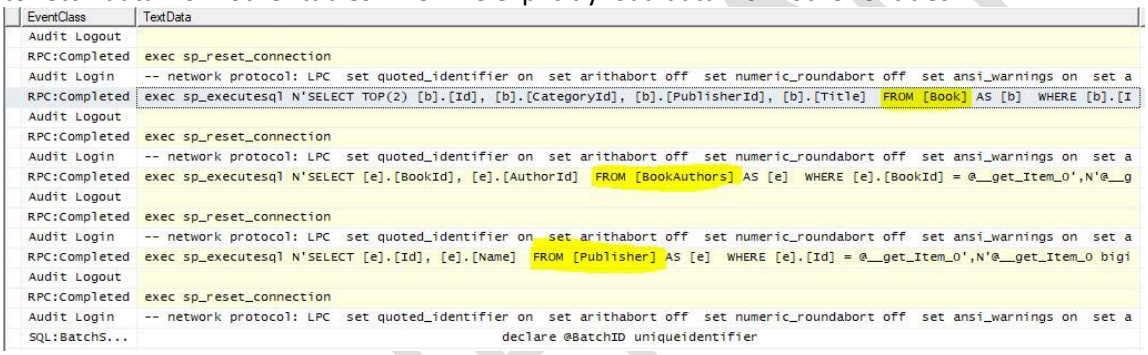


Next, let's test the Get endpoint of the Book:



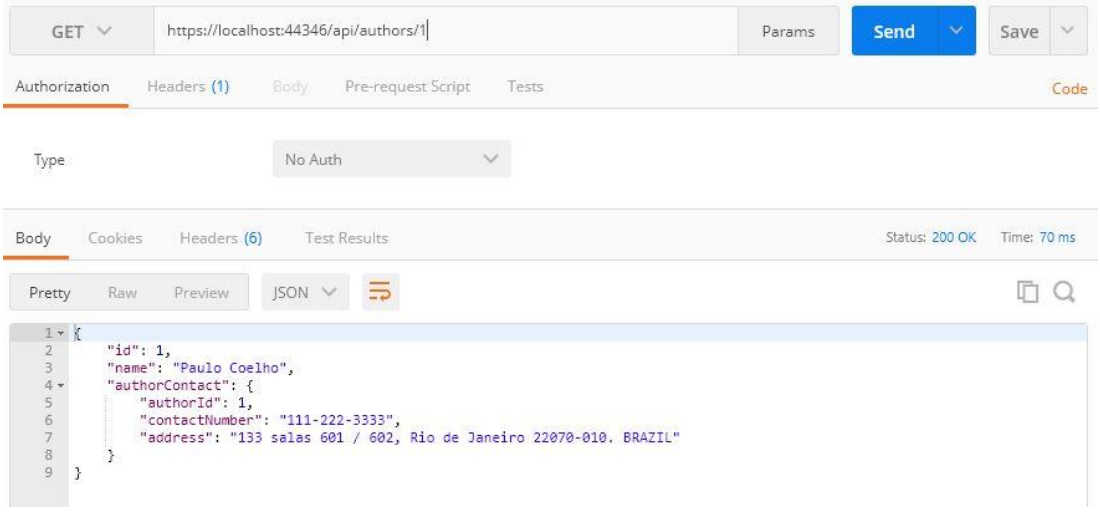
Remember that we used explicit loading to implement this functionality. Here note that only those properties that we chose to load explicitly have data. Other related properties are empty.

In the Profiler, we can see that initially, an SQL query fetches data from the Book table. Later, queries are generated to fetch data from other tables when we explicitly load data from other entities.

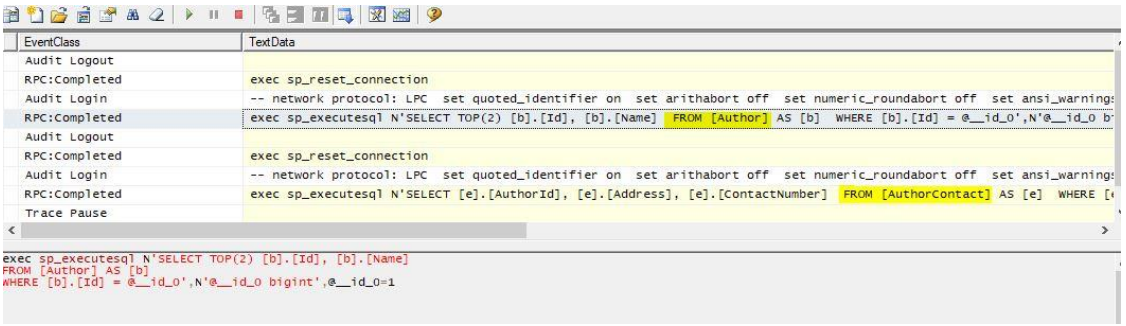


EventClass	TextData
Audit Logout	
RPC:Completed	exec sp_reset_connection
Audit Login	-- network protocol: LPC set quoted_identifier on set arithabort off set numeric_roundabort off set ansi_warnings on set a
RPC:Completed	exec sp_executesql N'SELECT TOP(2) [b].[Id], [b].[CategoryId], [b].[PublisherId], [b].[Title] FROM [Book] AS [b] WHERE [b].[I
Audit Logout	
RPC:Completed	exec sp_reset_connection
Audit Login	-- network protocol: LPC set quoted_identifier on set arithabort off set numeric_roundabort off set ansi_warnings on set a
RPC:Completed	exec sp_executesql N'SELECT [e].[BookId], [e].[AuthorId] FROM [BookAuthors] AS [e] WHERE [e].[BookId] = @__get_Item_0',N'@__g
Audit Logout	
RPC:Completed	exec sp_reset_connection
Audit Login	-- network protocol: LPC set quoted_identifier on set arithabort off set numeric_roundabort off set ansi_warnings on set a
RPC:Completed	exec sp_executesql N'SELECT [e].[Id], [e].[Name] FROM [Publisher] AS [e] WHERE [e].[Id] = @__get_Item_0',N'@__get_Item_0 bigi
Audit Logout	
RPC:Completed	exec sp_reset_connection
Audit Login	-- network protocol: LPC set quoted_identifier on set arithabort off set numeric_roundabort off set ansi_warnings on set a
SQL:BatchS...	declare @BatchID uniqueidentifier

Now, let's test the Get endpoint of Author:

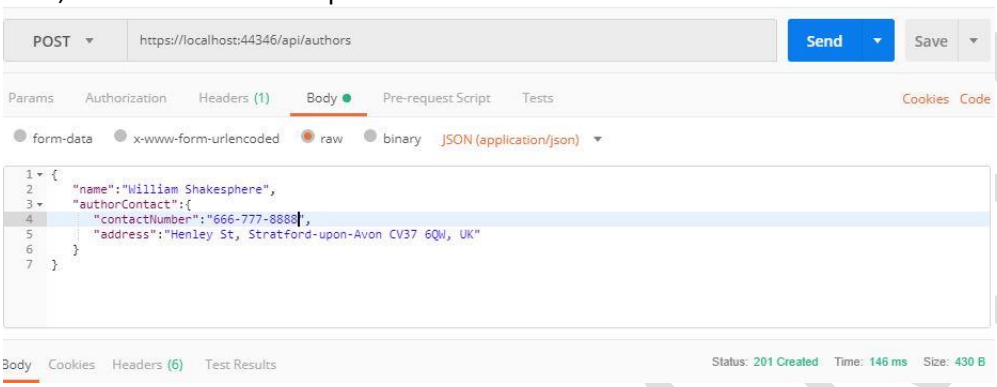


Remember that we used lazy loading to implement this functionality. In the Profiler, we can see that initially only data from the Author table is loaded. Later, when we refer the AuthorContact entity inside the DTO Mapper class, another query loads data from the AuthorContact table:

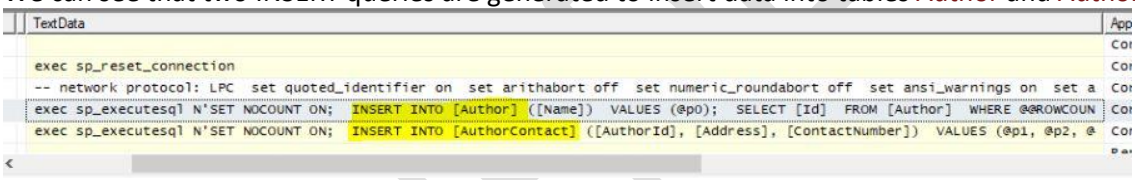


Updating Data

Now, let's test the Add endpoint of Author:



We can see that two INSERT queries are generated to insert data into tables Author and AuthorContact:



We can verify that our Add endpoint inserts data in both tables:

```
select * from Author
select * from AuthorContact
```

100 %

Results | Messages

	Id	Name
1	1	Paulo Coelho
2	2	Eckhart Tolle
3	3	Amie Kaufman
4	4	Jay Kristoff
5	5	William Shakespeare

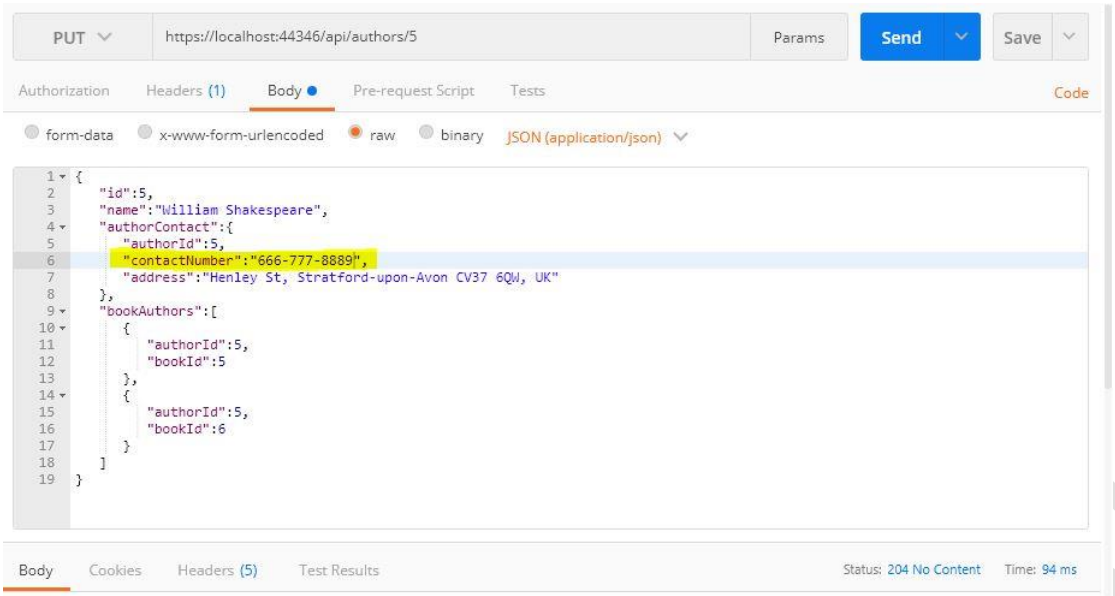
	AuthorId	ContactNumber	Address
1	1	111-222-3333	133 salas 601 / 602, Rio de Janeiro 22070-010. B...
2	2	444-555-6666	933 Seymour St, Vancouver, BC V6B 6L6, Canada
3	3	777-888-9999	Mentone 3194. Victoria. AUSTRALIA
4	4	222-333-4444	234 Collins Street, Melbourne, VIC, AUSTRALIA
5	5	666-777-8888	Henley St, Stratford-upon-Avon CV37 6QW, UK

Now let's test the Update endpoint of Authors.

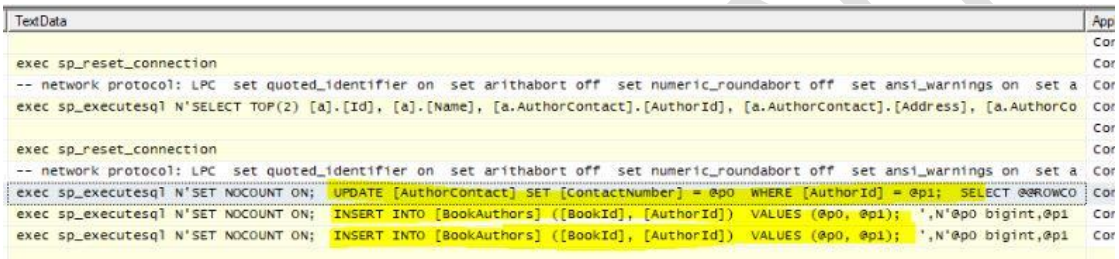
We'll insert some data into Publisher,BookCategory and Book table:

```
INSERT INTO Publisher
VALUES
('Simon & Schuster'),
('Oxford University Press')
INSERT INTO BookCategory
VALUES
('Tragedy'),
('Romance')
INSERT INTO Book
VALUES
('Hamlet', 5, 4),
('Romeo and Juliet', 6, 5)
```

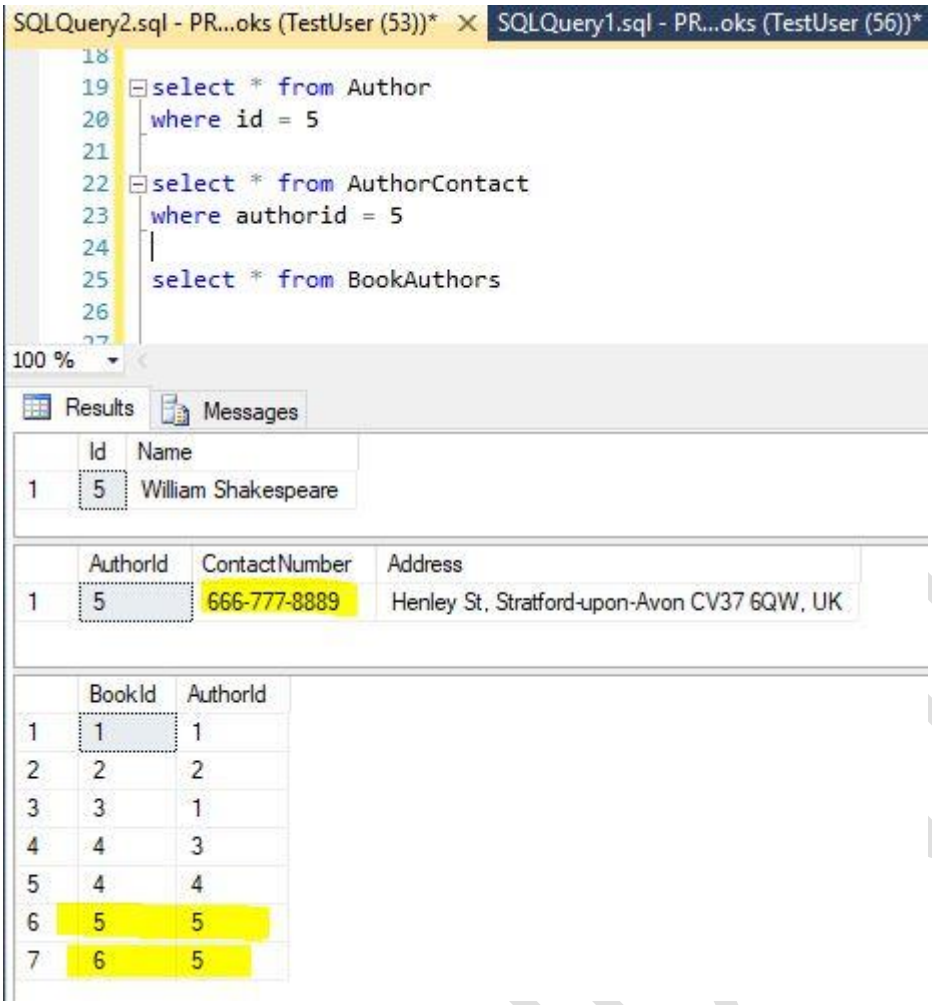
Let's modify the Author we just inserted. Let's edit the ContactNumber and map the newly added Books to this author:



In the Profiler, we can see an UPDATE query for the AuthorContact table and two INSERT queries for the BookAuthors table:



Let's verify the results in the database:

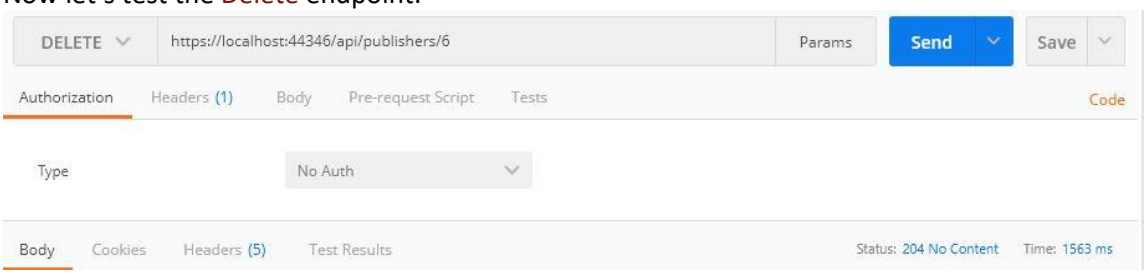


Finally, let's test the Delete endpoint of Publisher.

We'll insert a test publisher and two related books:

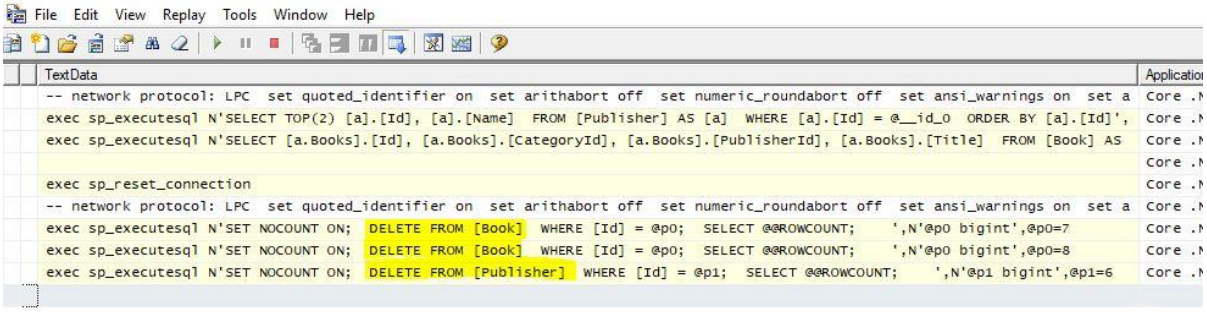
```
INSERT INTO Publisher
VALUES
('My Publisher')
INSERT INTO Book
VALUES
('My Publisher Book 1', 5, 6),
('My Publisher Book 2', 4, 6)
```

Now let's test the Delete endpoint.

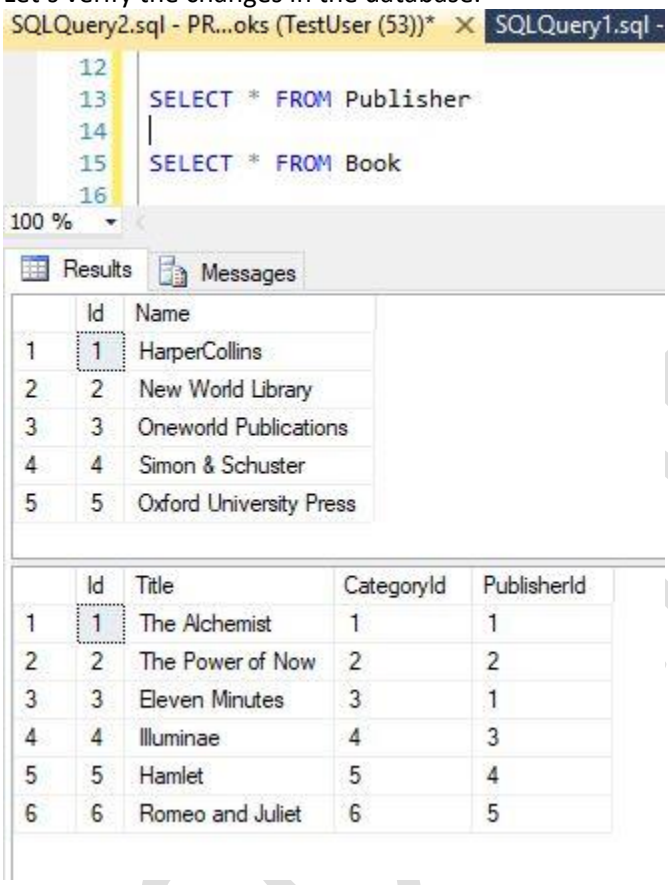


In the Profiler, we can see that the related data is first removed from theBook table. Then the publisher record is

deleted from thePublisher table.



Let's verify the changes in the database.



	Id	Name
1	1	HarperCollins
2	2	New World Library
3	3	Oneworld Publications
4	4	Simon & Schuster
5	5	Oxford University Press

	Id	Title	CategoryId	PublisherId
1	1	The Alchemist	1	1
2	2	The Power of Now	2	2
3	3	Eleven Minutes	3	1
4	4	Illuminae	4	3
5	5	Hamlet	5	4
6	6	Romeo and Juliet	6	5

Conclusion

In this , we have covered the following topics.

- EF Core Database-First approach and when to use it.
- Different types of relationships in a database.
- Creating a database and tables with relationships.
- Modeling the entities with relationships.
- Loading and saving related data using the repository pattern.
- Different patterns for loading related data.
- Creating API endpoints for operating on related data.
- Testing the endpoints and inspecting the generated database queries.

27. Address Book:

Problem Statement

Your task here is to implement a **C#** code based on the following specifications. Note that your code should match the specifications in a precise manner. Consider **default visibility** of classes, data fields and methods unless mentioned otherwise.

Specifications

```
class PersonAddress:
  method definition:
   Type: Implement getter setter method.
   return type: string
   visibility: public

   HouseNo: Implement getter setter method.
   return type: string
   visibility: public

   StreetName: Implement getter setter method.
   return tye: string
   visibility: public

   City: Implement getter setter method.
   return type: string
   visibility: public

class PersonalDetails:
  method definition:
   Name : Implement getter setter method.
   return type: string
   visibility: public

   Age: Implement getter setter method.
   return type: int
   visibility: public

   Company: Implement getter setter method.
   return type: string
   visibility: public
```

```
   PAN: Implement getter setter method.
   return type: string
   visibility: public

   AddressList: Implement getter setter method.
   return type: List<PersonAddress>
   visibility: public

class Program:
  method definition:
   Serialize: serialize List of PersonalDetails object to xml file
(xmloutput.xml)
   return type: string
   return message Successful if serialization is successful.

   DeSerialize: Deserializae xml file (xmloutput.xml) into List of
PersonalDetails object
   return type: List<PersonalDetails>
```

Task:

- Create classes **PersonAddress** and **PersonalDetails** according to above specifications
- Create a class **Program** and implement the below given methods:

1. **Serialize :** serialize list to xml file (xmloutput.xml)
2. **Deserialize :** deserialize xml file into list collection of PersonalDetails object.

- Use XMLElement attribute to define Order of xml tags.
- Don't add PAN property to xml file while serializing
- Use XMLAttribute for HouseNo from PersonAddress class to add as xml attribute (don't create xml tag for HouseNo)
- Serialize List of PersonalDetails object to 1 xml file.

IMPORTANT

If you want to test your program you can implement a Main() function and you can use RUN CODE to test your Main() provided you have made valid function calls with valid data required.

```
using System;
using System.Collections.Generic;
using System.IO;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
using System.Xml.Serialization;

class Program
{
    static void Main(string[] args)
    {
```

```
        //List<PersonalDetails> lstPersons = new List<PersonalDetails>();
        //List<PersonAddress> lstAddress1 = new List<PersonAddress>();
        //lstAddress1.Add(new PersonAddress { HouseNo = "123",Type="Home", StreetName = "MG road", City =
"BLR" });
        //lstAddress1.Add(new PersonAddress { HouseNo = "909",Type="Work", StreetName = "Outer road", City =
"BLR" });



        //List<PersonAddress> lstAddress2 = new List<PersonAddress>();
        // lstAddress2.Add(new PersonAddress { HouseNo = "40", Type = "Home", StreetName = "Shivajinagar", City =
"BLR" });
        //lstAddress2.Add(new PersonAddress { HouseNo = "909", Type = "Work", StreetName = "Outer road", City =
"BLR" });

        //lstPersons.Add(new PersonalDetails { Name = "Sachin", Age = 34, Company = "ABC", PAN = "AAAMN9880a",
AddressList = lstAddress1 });
        //lstPersons.Add(new PersonalDetails { Name = "Rohit", Age = 30, Company = "ABC", PAN = "BBAMN9880a",
AddressList = lstAddress2 });


        //Program p = new Program();
        //string msg= p.Serialize(lstPersons);

        // string xmlpath = @"Person.xml";
        //List<PersonalDetails> lstDesr = p.DeSerialize(xmlpath);
    }

    public string Serialize(List<PersonalDetails> person)
    {
        string msg = string.Empty;

        XmlSerializer serializer = new XmlSerializer(typeof(List<PersonalDetails>));
        using (TextWriter writer = new StreamWriter(@"Person.xml"))
        {
         serializer.Serialize(writer, person);

        }
         msg = "Successful";


        return msg;
     }

    public List<PersonalDetails> DeSerialize(string xmlPath)
    {
        XmlSerializer deserializer = new XmlSerializer(typeof(List<PersonalDetails>));
        TextReader reader = new StreamReader(xmlPath);
        object obj = deserializer.Deserialize(reader);
        List<PersonalDetails> XmlData = (List<PersonalDetails>)obj;
        reader.Close();
        return XmlData;
    }
}
```

```
public class PersonalDetails
{
   [XmlElement(Order = 1)]
   public string Name { get; set; }

   [XmlElement(Order = 2)]
   public int Age { get; set; }

   [XmlElement(Order =3)]
   public string Company { get; set; }

   [XmlElement(Order = 4)][XmlIgnore]
   public string PAN { get; set; }

   [XmlElement("Address",Order =5)]
   public List<PersonAddress> AddressList = new List<PersonAddress>();
}

public class PersonAddress
{
   [XmlAttribute("AddressType")]
   public string Type { get; set; }
   public string HouseNo { get; set; }
   public string StreetName { get; set; }
   public string City { get; set; }
}
```

28. Abstract class Motor Vehicle:

Motor Vehicle:

The problem is related to identifying the correct driving for a particular vehicle i.e., if it is a two wheeler or four wheeler, vehicle should be driven accordingly.

Your task here is to implement a **C#** code based on the following specifications. Note that your code should match the specifications in a precise manner. Consider **default visibility** of classes, data fields and methods are public unless mentioned otherwise.

Specification:

```
class definition:
        class : MotorVehicle
        type : abstract
        visibility : default
        method definition : DriveVehicle
        return type : string
        type : abstract
        visibility : public
        property : DriveStatus
        get visibility : public
        set visibility : private
```

```
class : TwoWheeler
inherits : MotorVehicle
method definition : DriveVehicle()
return type  : string
type : override
visibility : public

class : FourWheeler
inherits : MotorVehicle
method definition :  DriveVehicle()
return type  : string
type : override
visibility : public
```

Task:

class TwoWheeler:

- **string DriveVehicle():** Returns "You are driving two wheeler".

class TwoWheeler:

- **string DriveVehicle():** Returns "You are driving four wheeler".

class MotorVehicle:

- **DriveStatus:** This property should be set to "Preparing for drive" when object is intantiated.

Sample input:

```
        var twoWheeler = new TwoWheeler();
Console.WriteLine($"{twoWheeler.DriveStatus}");
var twoWheelerDrive = twoWheeler.DriveVehicle();
Console.WriteLine($"{twoWheelerDrive}");
var fourWheeler = new FourWheeler();
Console.WriteLine($"{fourWheeler.DriveStatus}");
var fourWheelerDrive = fourWheeler.DriveVehicle();
Console.WriteLine($"{fourWheelerDrive}");
```

Sample output:

```
Preparing for drive
You are driving two wheeler
Preparing for drive
You are driving four wheeler
```

Note:

- You can make suitable function calls and use **the RUN CODE** button to check your **main()** method output.

```
using System;
abstract class MotorVehicle
{
   public string DriveStatus { get; private set; }
   public MotorVehicle()
   {
      DriveStatus = "Preparing for drive";
   }
   public abstract string DriveVehicle();
}
class TwoWheeler : MotorVehicle
{
   public override string DriveVehicle()
   {
      return "You are driving two wheeler";
   }
}
class FourWheeler : MotorVehicle
{
   public override string DriveVehicle()
   {
      return "You are driving four wheeler";
   }
}
//class Source
//{
//   static void Main(string[] args)
//   {
//      var twoWheeler = new TwoWheeler();
//      Console.WriteLine($"{twoWheeler.DriveStatus}");
//      var twoWheelerDrive = twoWheeler.DriveVehicle();
//      Console.WriteLine($"{twoWheelerDrive}");
//      var fourWheeler = new FourWheeler();
//      Console.WriteLine($"{fourWheeler.DriveStatus}");
//      var fourWheelerDrive = fourWheeler.DriveVehicle();
//      Console.WriteLine($"{fourWheelerDrive}");
//   }
//}
```

29. Shop and Products:

1)Create a class Product with following public attributes

ProductCode:string

Name:string

Price:double

Brand:string

2)Create a class Shop with following public attributes

Name:string

ProdList:List<Product>

3) Add default constructor and parameterised constructor to initialize all member fields of the class.The order of initialization is as given below :

Shop(string name,List<Product> productList)

4)Create the following methods in the Shop class

· Public void AddProductToShop(Product p) : This method accepts a product object and adds the product to the product list of the current shop .If the product with same code and name already available in the list , then product should not be added to the list.

· Public bool RemoveProductFromShop(string productCode) :This method will get the productCode of the product and deletes the product with specified code from the current shop.If a product with a given code is found , then deletes the product and returns true.If product with productCode is not found then returns false.

5)Create a ProductBO class and add the below methods:

· public List<Product> FindProduct(List<Product> productList , string brand) : This method accepts a list of products and brand name as input parameter..This method will search products from productList and returns a list of products that matches the given brand name .

· public List<Product> FindProduct(List<Product> productList , double price) : This method accepts a list of products and price as input parameter.This method will search products from productList and and returns a list of products that matches the given price .

· public HashTable BrandWiseCount(List<Product> productList) : This method accepts a list of products and returns an object that contains brand-wise count of products.

using System;
using System.Collections.Generic;

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

class Product
   {
      public string ProductCode;

      public string Name;

      public double Price;
      public string Brand;
   }
   class Shop
   {
      public string Name;
      public List<Product> ProdList=new List<Product>();
      public Shop() { }
      public Shop(string name, List<Product> productList)
      {
         Name = name;
         ProdList = productList;
      }
      public void AddProductToShop(Product p)
      {
         if (ProdList.Count == 0)
            ProdList.Add(p);
         else
         {

            Product obj = ProdList.Find(x => x.Name.ToLower() == p.Name.ToLower() && x.ProductCode ==
p.ProductCode);
            if ((ProdList.Contains(obj)) == false)
               ProdList.Add(p);

         }

      }
      public bool RemoveProductFromShop(string productCode)
      {
         bool flag;
         Product obj = ProdList.Find(x =>  x.ProductCode == productCode);
         if ((ProdList.Contains(obj)) == true)
         {
            flag = true;
            ProdList.Remove(obj);
         }
         else
            flag = false;
         return flag;
      }

   }
```

```
class ProductBO
{
   public List<Product> FindProduct(List<Product> productList, string brand)
      {
      var res = from p in productList
            where p.Brand == brand
            select p;
      return res.ToList();
      }

   public List<Product> FindProduct(List<Product> productList, double price)
   {
      var res = from p in productList
            where p.Price == price
            select p;
      return res.ToList();

   }

   public Hashtable BrandWiseCount(List<Product> productList )
   {

     var res= from p in productList
            group p by p.Brand into g
            select new  { brand=g.Key,count=g.Count() };
      Hashtable ht1 = new Hashtable();
      foreach(var x in res)
      {
         ht1.Add(x.brand, x.count);

      }
      return ht1;


   }
}
```

30. Cab and Passengers:

1)Create a class Passenger with following public attributes

Pid: string

Name:string

Email:string

ContactNo:int

2)Create a class Cab with following public attributes

Cabid: string

RegNo:string

Type :string

Capacity :int

CostPerKm:double

PassengerList: List<Passenger>

Note : Here the Type variable is used to store type of cab like Micro,Mini,Shared etc.Capacity is used to store the total number of passengers that a cab can accommodate .

3) Add default constructor and parameterised constructor to initialize all member fields of the class.The order of initialization is as given below :

Cab(string cabid,string regno,string type,int capacity,double cost,List<Passenger> paslist)

4)Create the following methods in the Cab class

· Public void AddPassengerToCab(Passenger p) : This method accepts a passenger object and adds the passenger to the passenger list of the current cab .If the passenger with same id and name already available in the list , then passenger should not be added to the list.

· Public bool RemovePassengerFromCab(string id) :This method will get the id of the passenger and deletes the passenger with specified id from the current Cab.If a passenger with a given id is found , then deletes the passenger and returns true.If paasenger with id is not found then returns false.

5)Create a CabBO class and add the below methods:

· public List<Cab> FindCab(List<Cab> cabList , string type) : This method accepts a list of cabs and cab type as input parameter..This method will search cabs from cabList and returns a list of cabs that matches the given type .

· public List<Cab> FindPCab(List<Cab> cabList , int capacity) : This method accepts a list of cabs and capacity as input parameter.This method will search cabs from cabList and returns a list of cabs that matches the given capacity .

· public HashTable CapacityWiseCount(List<Cab> cabList) : This method accepts a list of cabs and returns an object that contains capacity-wise count of cabs.

```
using System;
using System.Collections;
using System.Collections.Generic;
using System.Linq;



class Passenger
   {
      public string Pid;
public string Name;
public string Email;
public int ContactNo;

   }

   class Cab
   {
      public string Cabid;
public string RegNo;
public string Type ;
public int Capacity;
     public double CostPerKm;
      public List<Passenger> PassengerList;
      public Cab() { }
      public Cab(string cabid, string regno, string type, int capacity, double cost, List<Passenger> paslist)
      {
         this.Cabid = cabid;
         this.RegNo = regno;
         this.Type = type;
         this.Capacity = capacity;
         this.CostPerKm = cost;
         this.PassengerList = paslist;
      }

      public void AddPassengerToCab(Passenger p)
      {
         Passenger obj = PassengerList.FirstOrDefault(x => x.Pid == p.Pid & x.Name == p.Name);
         if (obj == null)
            PassengerList.Add(p);

      }
      public bool RemovePassengerFromCab(string id)
      {
         int count = PassengerList.FindAll(x => x.Pid == id).Count();
         bool flag = false;
         if(count > 0)
```

```
        {
            flag = true;
            Passenger obj = PassengerList.FirstOrDefault(x => x.Pid == id);
            PassengerList.Remove(obj);
        }
        return flag;

    }



}

class CabBO
{

    public List<Cab> FindCab(List<Cab> cabList, string type)
    {
        List<Cab> clist1 = cabList.FindAll(x => x.Type == type);
        return clist1;

    }
    public List<Cab> FindCab(List<Cab> cabList, int capacity)
    {
        List<Cab> clist1 = cabList.FindAll(x => x.Capacity == capacity);
        return clist1;

    }

    public Hashtable CapacityWiseCount(List<Cab> cabList)
    {
        Hashtable ht = new Hashtable();
        var res = from c in cabList
                  group c by c.Capacity into grp
                  select new { capacity = grp.Key, count = grp.Count() };
        foreach( var x in res)
        {
            ht.Add(x.capacity, x.count);
        }
        return ht;

    }
}
```