ENTITY FRAMEWORK: INTRODUCCIÓN

Prerequisitos: Conocimientos básicos de .Net Framework, C#, Visual Studio y MS SQL Server.

Aclaración: Este documento tiene las generalidad de EF para proyectos con .Net Framework y **especialmente para proyectos con .Net Core**.

¿Qué es Entity Framework (EF)?

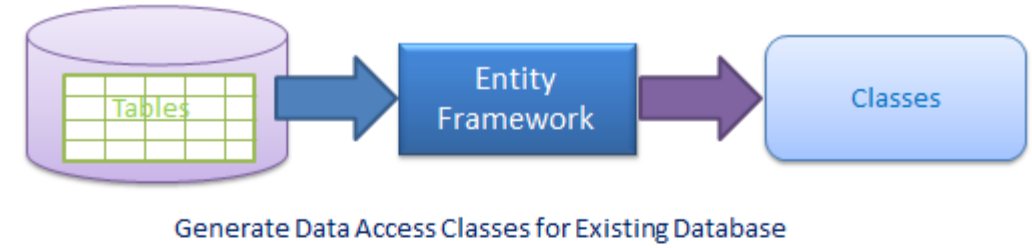
La estrategia de acceso a datos de toda aplicación .NET es el uso de los objetos de ADO.Net (Activex Data Object), que tiene todos los objetos necesarios para crear una conexión a un origen de datos que pueda ser tratado como origen de datos tabular: por ejemplo, base de datos relacional, sistema de mails, planillas de excel homogéneas, archivos xml, etc. Pero con la necesidad particular de tener que escribir comandos específicos de acceso a los datos (consultas y persistencia de los datos), en el lenguaje específico de cada origen de datos.

Microsoft provee EF que un framework de ORM (object relational mapping) para automatizar las operaciones que realiza una aplicación sobre una base de datos. Este ORM permite a los desarrolladores trabajar con datos relacionales como si fueran objetos del dominio de la aplicación que modelan, eliminando la necesidad de escribir código en la base de datos (stored procedures, etc.) y usando como medio el lenguaje LINQ dentro de la aplicación en el lenguaje .NET por ejemplo C# o VB.NET.

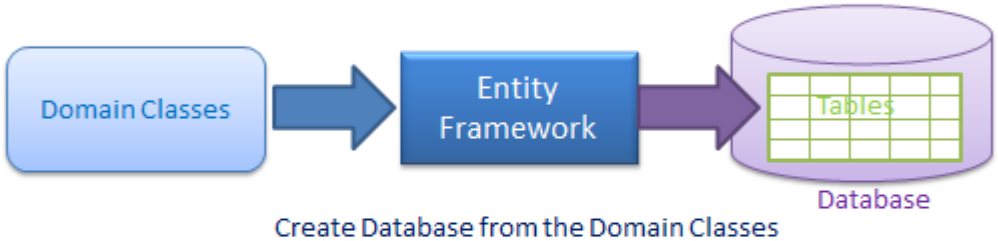
EF es muy útil en tres escenarios:

1) Si se dispone de una base de datos existente o desea diseñar su base de datos y a partir de allí tomarla con EF (**Database first**).
2) Si desea centrarse en el diseño modelando las clases de dominio de la aplicación (modelo de objetos) y a partir de allí con EF generar la base de datos basada en esos objetos del dominio del negocio (**Code first**).
3) Si desea diseñar el esquema de la base de datos en el diseñador visual de EF y a partir de allí EF generará las clases y la base de datos (**Model first**).

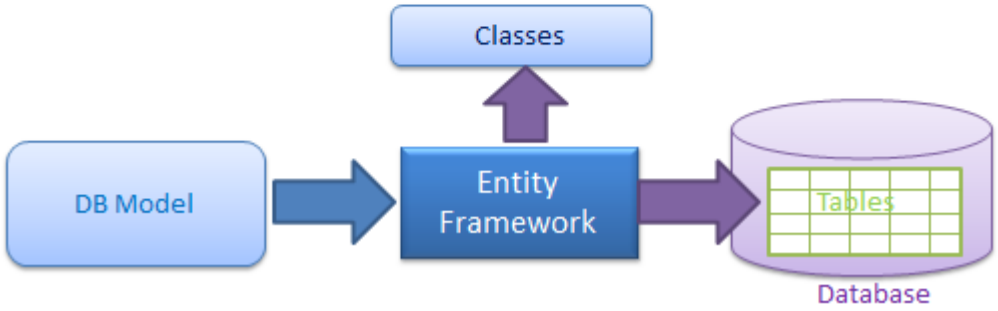
Ilustramos estos tres escenarios con imágenes:



Generate Data Access Classes for Existing Database



Create Database from the Domain Classes



Create Database and Classes from the DB Model design

Con .Net Framework se pueden usar los tres escenarios mencionados anteriormente, y si desea profundizar estos escenarios, se recomiendan estos links:

Entity Framework 6 DataBase First	https://www.entityframeworktutorial.net/entityframework6/introduction.aspx
Entity Framework 6 Code First	https://www.entityframeworktutorial.net/code-first/what-is-code-first.aspx

Con .Net Core sólo se usan los escenarios Code First y Database First, este link: https://www.entityframeworktutorial.net/efcore/entity-framework-core.aspx

¿Qué es un ORM?

Un ORM es una herramienta para almacenar datos desde objetos de dominio de un modelo de negocio orientado a objetos, en una base de datos relacional como Sql Server u otras, en forma automatizada si demasiada programación.

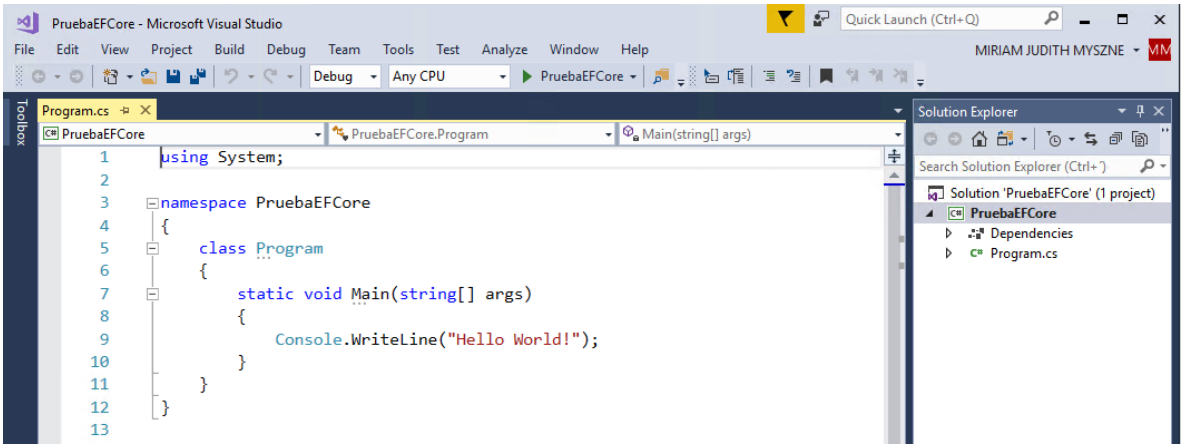
Un ORM tiene tres componentes principales:

- Clases de los objetos del dominio del negocio a modelar
- Objetos de la base de datos relacional
- Información de mapeo entre clases de negocio y los objetos de la base de datos (tablas, vistas y procedimientos almacenados).

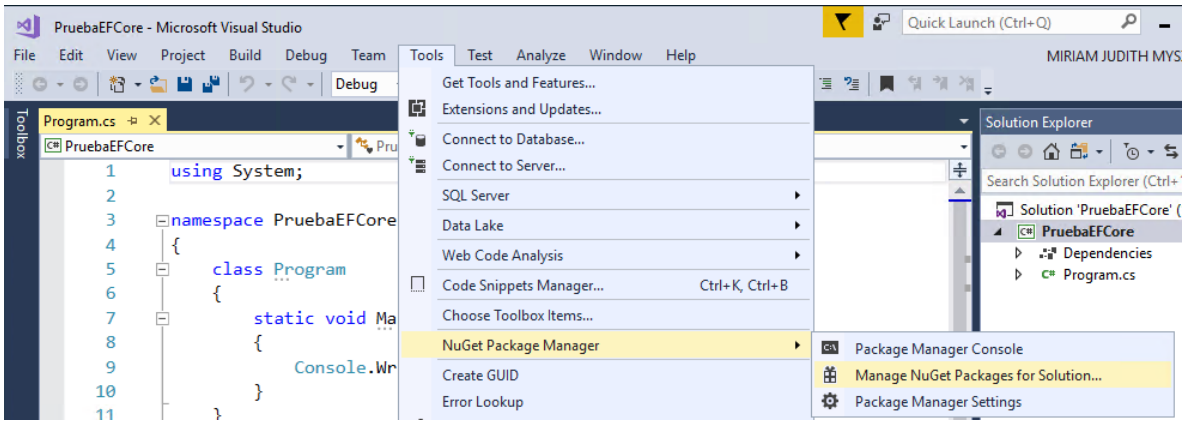
Un ORM mantiene el diseño de la base de datos separado del diseño de las clases de dominio. Esto permite que la aplicación que usa EF es mantenible y extendible. Y automatiza las operaciones CRUD estándar (Create=Insertar, Read=Leer, Update=Actualizar y Delete=Eliminar) evitando que los desarrolladores tengan escribir código que realizar estas operaciones.

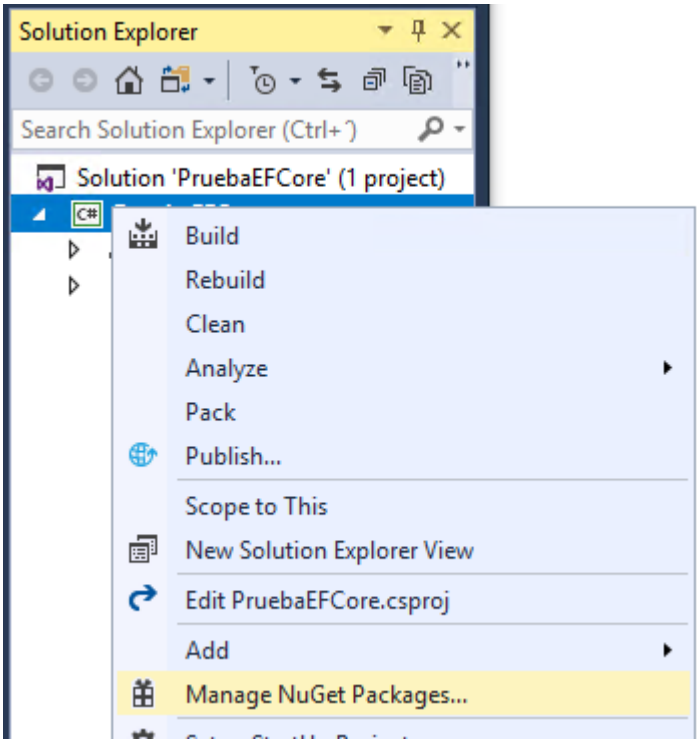
Instalación de EF Core en un proyecto C# con Visual Studio 2017/2019

Veremos el ejemplo con un proyecto de consola llamado **PruebaEFCore**:

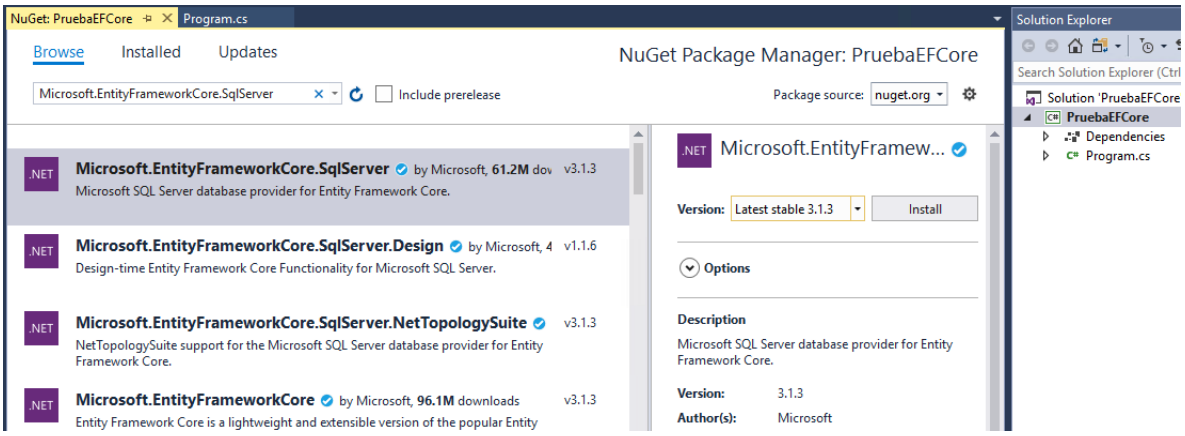


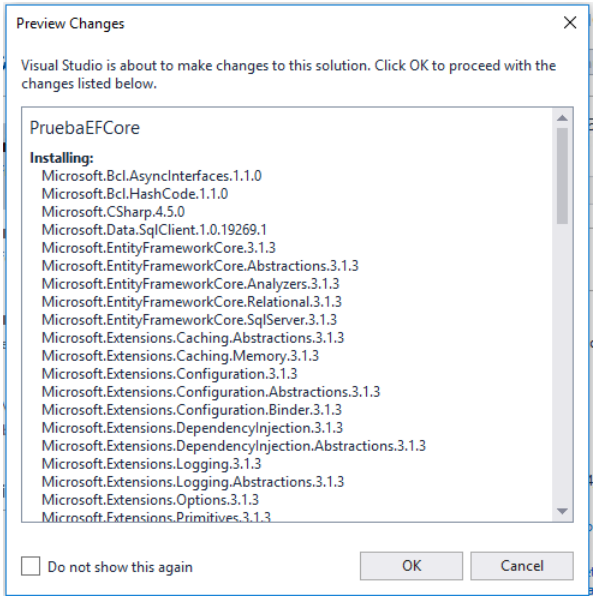
Debemos cargar el paquete NuGet de Entity Framework Core, lo podemos hacer desde el menú Tools o con click derecho desde el proyecto:

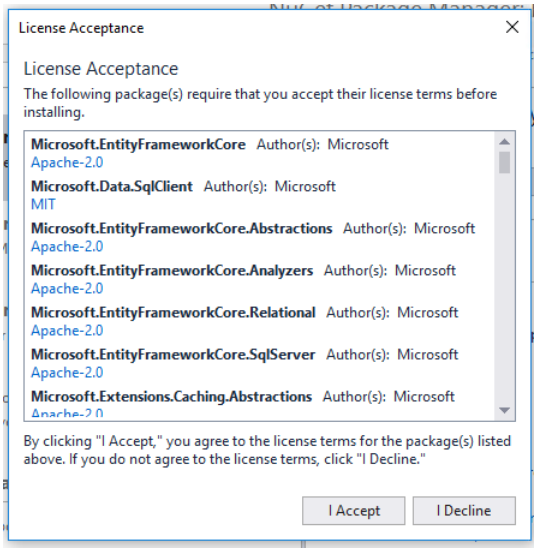




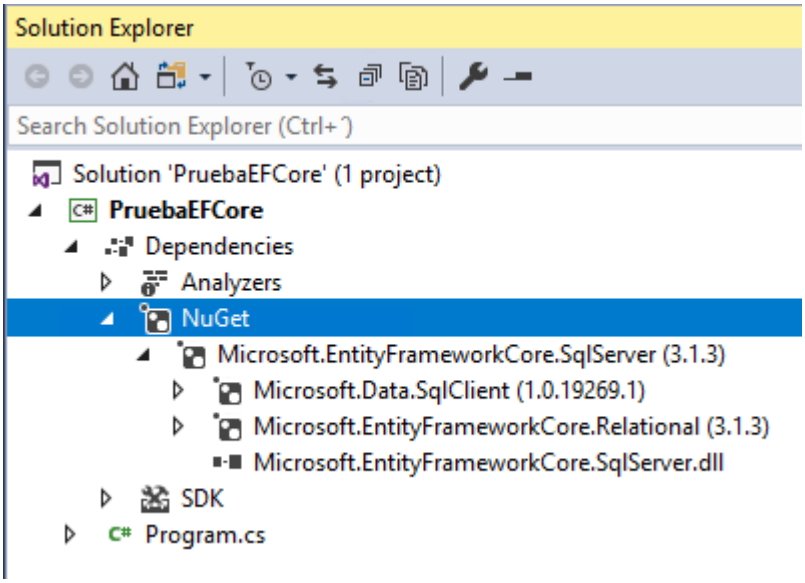
Buscamos el paquete **Microsoft.EntityFrameworkCore.SqlServer** y lo instalamos en el proyecto:


NuGet: PruebaEFCore
Program.cs
Browse
Installed
Updates
NuGet Package Manager: PruebaEFCore
Microsoft.EntityFrameworkCore.SqlServer
Include prerelease
Package source: nuget.org
Microsoft.EntityFrameworkCore.SqlServer by Microsoft, 61.2M dov v3.1.3
Microsoft SQL Server database provider for Entity Framework Core.
Microsoft.EntityFrameworkCore.SqlServer.Design by Microsoft, 4 v1.1.6
Design-time Entity Framework Core Functionality for Microsoft SQL Server.
Microsoft.EntityFrameworkCore.SqlServer.NetTopologySuite v3.1.3
NetTopologySuite support for the Microsoft SQL Server database provider for Entity Framework Core.
Microsoft.EntityFrameworkCore by Microsoft, 96.1M downloads v3.1.3
Entity Framework Core is a lightweight and extensible version of the popular Entity
Microsoft.EntityFramew...
Version: Latest stable 3.1.3
Install
Options
Description
Microsoft SQL Server database provider for Entity Framework Core.
Version: 3.1.3
Author(s): Microsoft
Solution Explorer
Search Solution Explorer (Ctrl
Solution 'PruebaEFCore
PruebaEFCore
Dependencies
Program.cs


Preview Changes
Visual Studio is about to make changes to this solution. Click OK to proceed with the changes listed below.
PruebaEFCore
Installing:
Microsoft.Bcl.AsyncInterfaces.1.1.0
Microsoft.Bcl.HashCode.1.1.0
Microsoft.CSharp.4.5.0
Microsoft.Data.SqlClient.1.0.19269.1
Microsoft.EntityFrameworkCore.3.1.3
Microsoft.EntityFrameworkCore.Abstractions.3.1.3
Microsoft.EntityFrameworkCore.Analyzers.3.1.3
Microsoft.EntityFrameworkCore.Relational.3.1.3
Microsoft.EntityFrameworkCore.SqlServer.3.1.3
Microsoft.Extensions.Caching.Abstractions.3.1.3
Microsoft.Extensions.Caching.Memory.3.1.3
Microsoft.Extensions.Configuration.3.1.3
Microsoft.Extensions.Configuration.Abstractions.3.1.3
Microsoft.Extensions.Configuration.Binder.3.1.3
Microsoft.Extensions.DependencyInjection.3.1.3
Microsoft.Extensions.DependencyInjection.Abstractions.3.1.3
Microsoft.Extensions.Logging.3.1.3
Microsoft.Extensions.Logging.Abstractions.3.1.3
Microsoft.Extensions.Options.3.1.3
Do not show this again
OK
Cancel


License Acceptance
License Acceptance
The following package(s) require that you accept their license terms before installing.
Microsoft.EntityFrameworkCore Author(s): Microsoft
Apache-2.0
Microsoft.Data.SqlClient Author(s): Microsoft
MIT
Microsoft.EntityFrameworkCore.Abstractions Author(s): Microsoft
Apache-2.0
Microsoft.EntityFrameworkCore.Analyzers Author(s): Microsoft
Apache-2.0
Microsoft.EntityFrameworkCore.Relational Author(s): Microsoft
Apache-2.0
Microsoft.EntityFrameworkCore.SqlServer Author(s): Microsoft
Apache-2.0
Microsoft.Extensions.Caching.Abstractions Author(s): Microsoft
By clicking "I Accept," you agree to the license terms for the package(s) listed above. If you do not agree to the license terms, click "I Decline."
I Accept
I Decline

Una vez instalado en el proyecto verá la librería correspondiente:



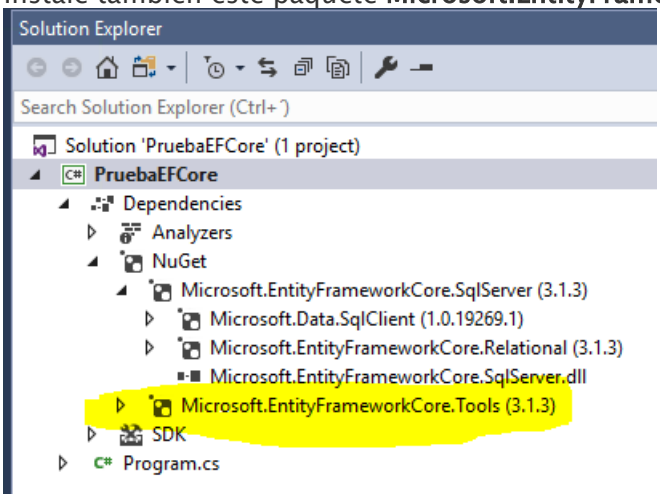
También podría instalarlo desde la consola Nuget con esta instrucción:

```
PM> Install-Package Microsoft.EntityFrameworkCore.SqlServer
```

Instalación de herramientas adicionales EF Core en un proyecto C# con Visual Studio 2017/2019

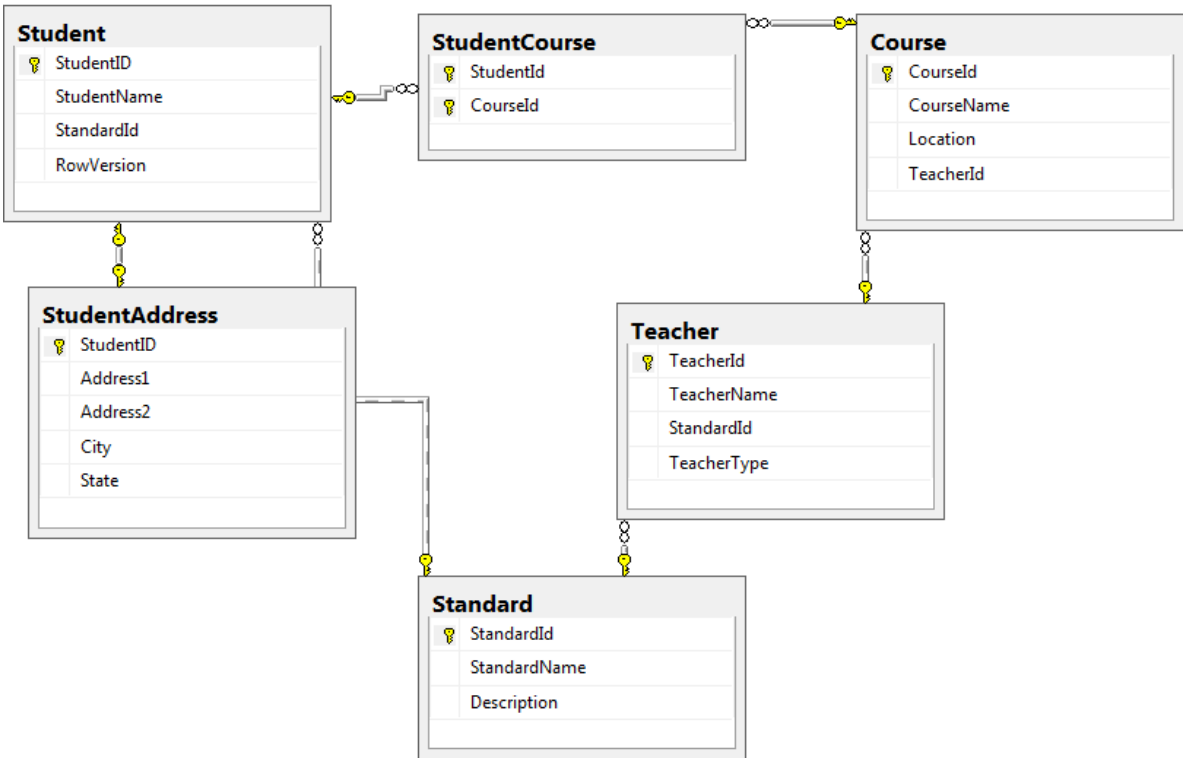
Instalando esta herramienta adicional, agiliza tareas relacionadas como migraciones, scaffolding para crear CRUDs, etc.

Instale también este paquete **Microsoft.EntityFrameworkCore.Tools**



Base de datos de ejemplo para ambos escenarios: Database First y Code First

Para ejercitar con EF usaremos la siguiente base de datos **SchoolDB**:



En sql server:

1) Cree una base de datos vacía llamada **SchoolDB** conectándose a la instancia **(localdb)\MSSQLLocalDB** sino no posee una instancia servidor.
2) Cree una base de datos llamada **SchoolDB** y ejecute el script **scriptSCHOOLDB.sql para completar la estructura y datos.**

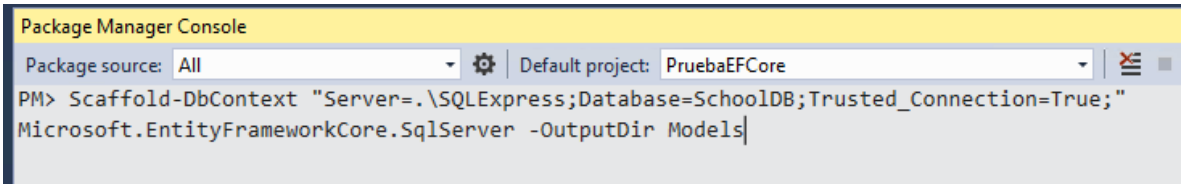
En la base de datos tenemos el esquema con las siguientes tablas y relaciones:

- **Una a uno: *Student*** y ***StudentAddress*** tiene una relación uno a uno, por ejemplo un estudiante tiene 0 o 1 domicilio.
- **Uno a muchos: *Standard*** y ***Teacher*** tiene una relación uno a muchos, por ejemplo muchos docentes pueden tener asociados el mismo Standard.
- **Mucho a muchos: *Student*** y ***Course*** tiene una relación de muchos a muchos con la table ***StudentCourse*** como intermediaria y con la clave primaria compuesta ***StudentId*** y ***CourseId***. Un estudiante puede tener muchos cursos y un curso puede tener muchos estudiantes.

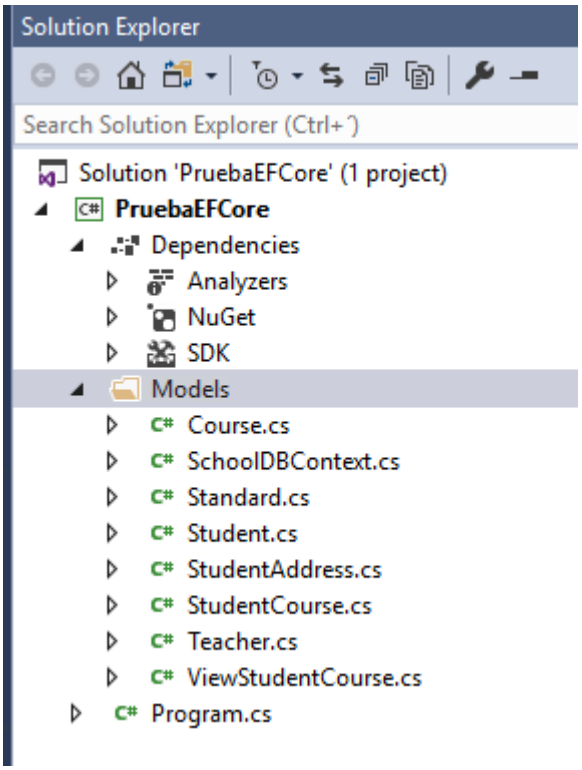
Creación del contexto de EF vía scafolding con escenario Database First

Desde la consola escriba este comando desde la consola de Package Manager Console:

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



Este comando indica crear un Contexto de EF que sería algo similar a la conexión a la base de datos de que se usará, y se le indica la instancia, el nombre de la base de datos, el tipo de autenticación (en este caso con las credenciales de windows), y en outputdir el nombre de la carpeta del proyecto donde quedarán las clases de modelo.



Vemos la clase generada para la tabla **Student**:

Student.cs | SchoolDBContext.cs | Program.cs

PruebaEFCore | PruebaEFCore.Models.Student | Standard

```
1   using System;
2   using System.Collections.Generic;
3
4   namespace PruebaEFCore.Models
5   {
6       public partial class Student
7       {
8           public Student()
9           {
10              StudentCourse = new HashSet<StudentCourse>();
11          }
12
13          public int StudentId { get; set; }
14          public string StudentName { get; set; }
15          public int? StandardId { get; set; }
16          public byte[] RowVersion { get; set; }
17
18          public virtual Standard Standard { get; set; }
19          public virtual StudentAddress StudentAddress { get; set; }
20          public virtual ICollection<StudentCourse> StudentCourse { get; set; }
21      }
22  }
```

Vemos la clase **SchoolDBContext** con el mapeo de la estructura de la base de datos a clases de negocio, y también la información de conexión. Podemos agregar que **cada DBSet es como una lista de objetos**.

SchoolDBContext.cs* | Program.cs

PruebaEFCore.Models.SchoolDBContext | SchoolDBContext(DbContextOptions<SchoolDBContext

```
public partial class SchoolDBContext : DbContext
{
    public SchoolDBContext()
    {
    }

    public SchoolDBContext(DbContextOptions<SchoolDBContext> options)
        : base(options)
    {
    }
    public virtual DbSet<Course> Course { get; set; }
    public virtual DbSet<Standard> Standard { get; set; }
    public virtual DbSet<Student> Student { get; set; }
    public virtual DbSet<StudentAddress> StudentAddress { get; set; }
    public virtual DbSet<StudentCourse> StudentCourse { get; set; }
    public virtual DbSet<Teacher> Teacher { get; set; }
    public virtual DbSet<ViewStudentCourse> ViewStudentCourse { get; set; }

    protected override void OnConfiguring(DbContextOptionsBuilder optionsBuilder)
    {
        if (!optionsBuilder.IsConfigured)
        {
#warning To protect potentially sensitive information in your connection string, you should move it out
            optionsBuilder.UseSqlServer("Server=.\\SQLExpress;Database=SchoolDB;Trusted_Connection=
        }
```

Para más detalles sobre la información del DBContext consulte este link:

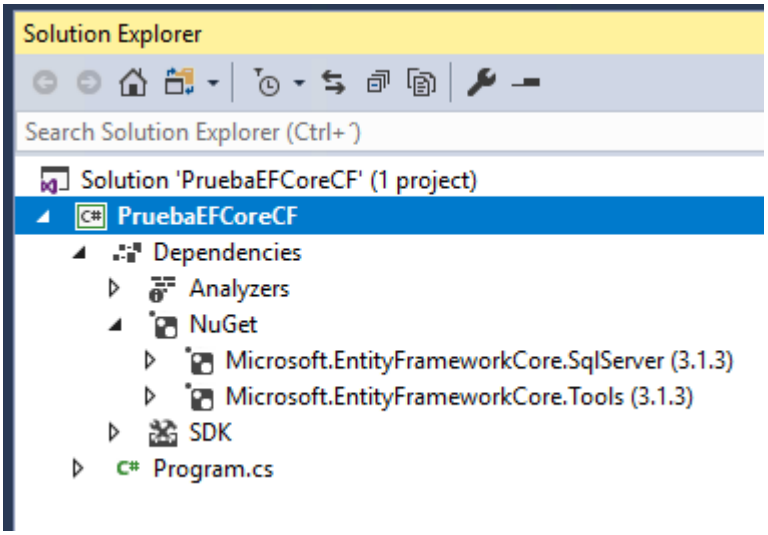
https://www.entityframeworktutorial.net/efcore/entity-framework-core-dbcontext.aspx

Creación del contexto de EF vía scafolding con escenario Code First

En otro proyecto de consola llamado **PruebaEFCoreCF**, agregamos ambos paquetes Nuget:

Microsoft.EntityFrameworkCore.SqlServer

Microsoft.EntityFrameworkCore.Tools



Y creamos dos clases del model para a partir de allí crear la base de datos:

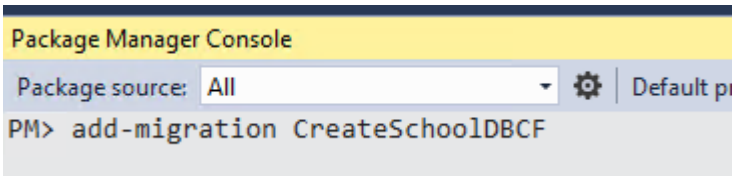
```
public class Student
{
    public int StudentId { get; set; }
    public string Name { get; set; }
}

public class Course
{
    public int CourseId { get; set; }
    public string CourseName { get; set; }
}
```

Y agregamos la clase **SchoolContext** que hereda de **DBContext**. Tenga cuidado no pise la base de datos del ejemplo con Database First, llamela **SchoolDBCF**

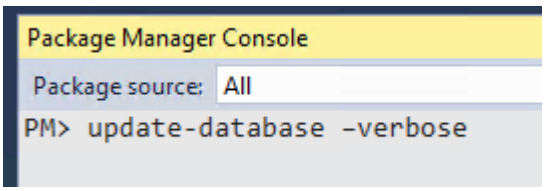
```
public class SchoolContext : DbContext
    {
        public DbSet<Student> Students { get; set; }
        public DbSet<Course> Courses { get; set; }

        protected override void OnConfiguring(DbContextOptionsBuilder
optionsBuilder)
        {

optionsBuilder.UseSqlServer(@"Server=.\SQLEXPRESS;Database=SchoolDBCF
;Trusted_Connection=True;");
        }
    }
```

Debemos crear una migración y actualizarla para que genere la base de datos con las dos tablas:

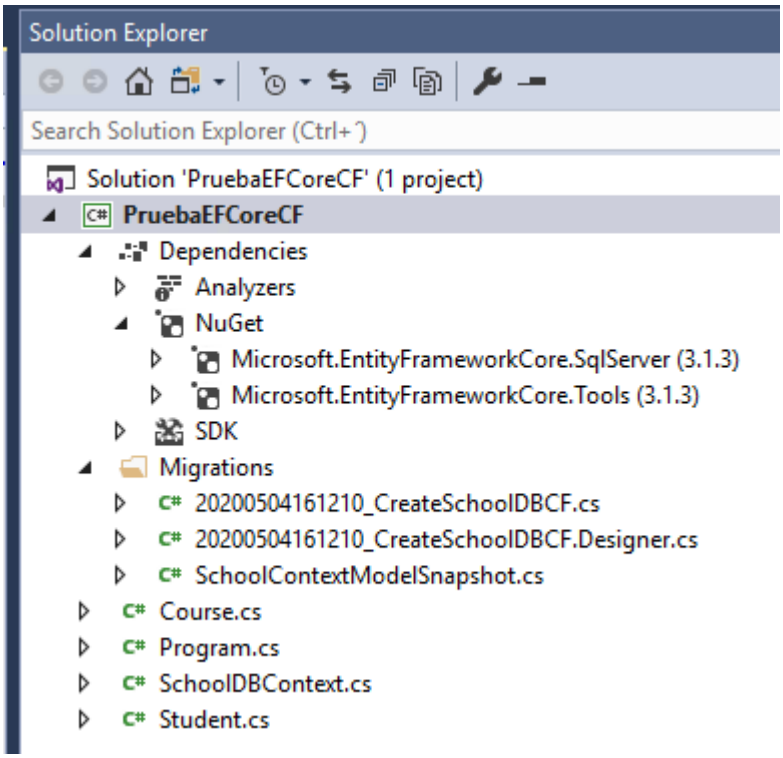
add-migration CreateSchoolDBCF

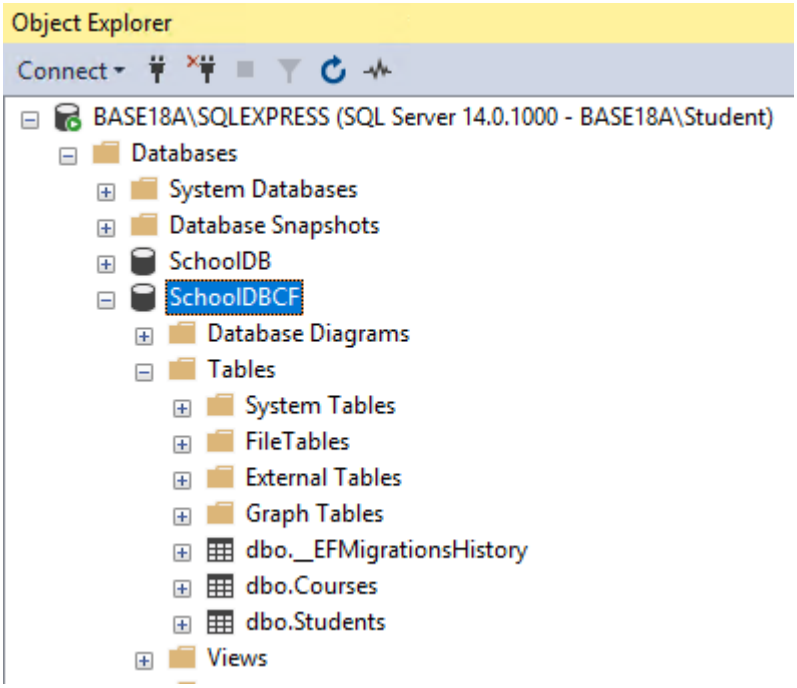


update-database -verbose



Se creó la migración y la base de datos:

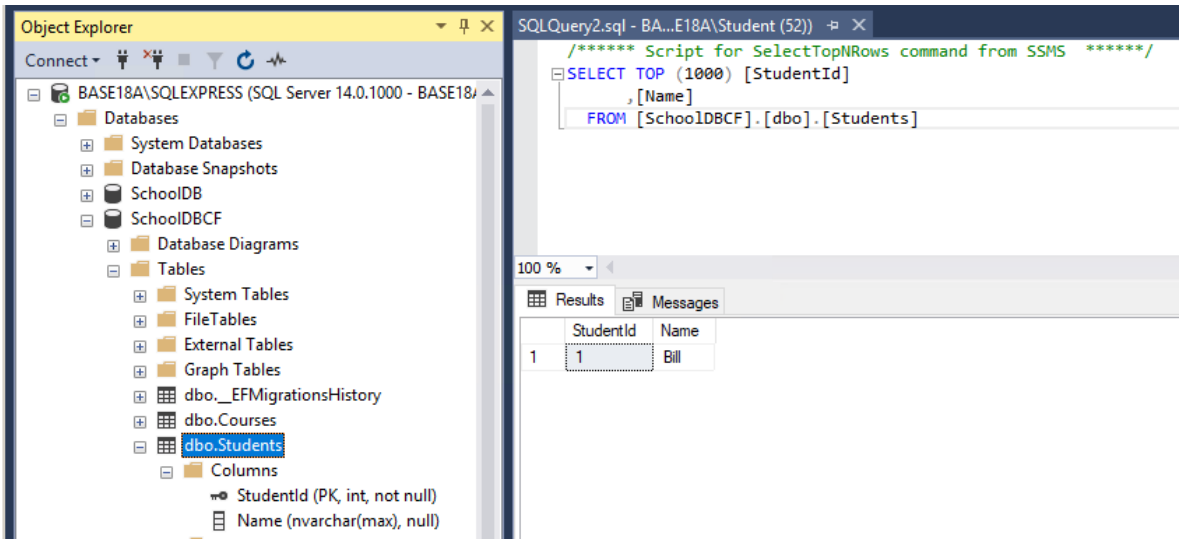

Solution Explorer
Search Solution Explorer (Ctrl+;)
Solution 'PruebaEFCoreCF' (1 project)
PruebaEFCoreCF
Dependencies
Analyzers
NuGet
Microsoft.EntityFrameworkCore.SqlServer (3.1.3)
Microsoft.EntityFrameworkCore.Tools (3.1.3)
SDK
Migrations
20200504161210_CreateSchoolDBCF.cs
20200504161210_CreateSchoolDBCF.Designer.cs
SchoolContextModelSnapshot.cs
Course.cs
Program.cs
SchoolDBContext.cs
Student.cs


Object Explorer
Connect
BASE18A\SQLEXPRESS (SQL Server 14.0.1000 - BASE18A\Student)
Databases
System Databases
Database Snapshots
SchoolDB
SchoolDBCF
Database Diagrams
Tables
System Tables
FileTables
External Tables
Graph Tables
dbo.__EFMigrationsHistory
dbo.Courses
dbo.Students
Views

Consultando el modelo desde el proyecto con Code First

Agregar un estudiante

```
using System;
using System.Linq;

namespace PruebaEFCoreCF
{
    class Program
    {
        static void Main(string[] args)
        {
            // Agregar un estudiante
            using (var context = new SchoolContext())
            {

                var std = new Student()
                {
                    Name = "Bill"
                };

                context.Students.Add(std);
                context.SaveChanges();
            }

            Console.WriteLine("Estudiante agregado!");
            Console.ReadKey();
        }
    }
}
```

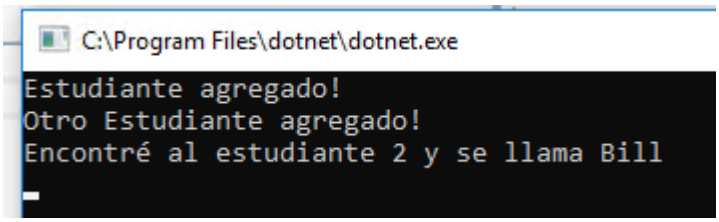


Puede agregar otro estudiante:

```
// Agregar otro estudiante
            using (var context = new SchoolContext())
            {

                var std = new Student()
                {
                    Name = "Mary"
                };

                context.Students.Add(std);
                context.SaveChanges();
            }

            Console.WriteLine("Otro Estudiante agregado!");
            Console.ReadKey();
```

Consultar un estudiante

```
//Consultar un estudiante con expresión Lambda
using (var context = new SchoolContext())
      {
      var consulta = context.Students.Where(s => s.StudentId == 2);

      var estudiante = consulta.FirstOrDefault<Student>();

      Console.WriteLine("Encontré al estudiante 2 y se llama " +
estudiante.Name );

      Console.ReadKey();
            }
```



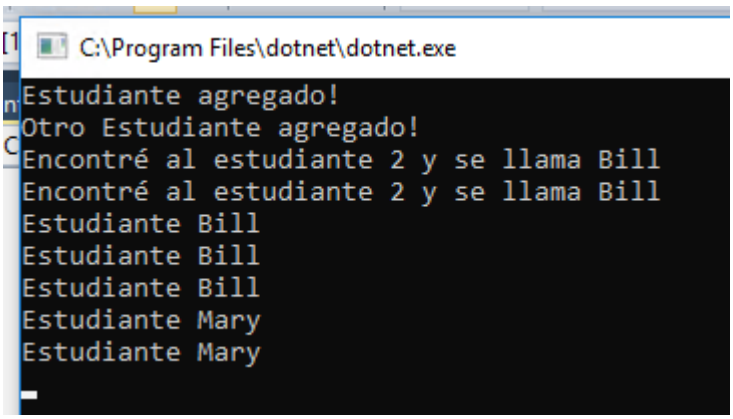
También la podemos escribir con una expresión de consulta:

```
            //Consultar un estudiante con expresión de consulta
using (var context = new SchoolContext())
      {
```

```
      var cons = from st in context.Students
                           where st.StudentId == 2
                           select st;

      var estu = cons.FirstOrDefault<Student>();
      Console.WriteLine("Encontré al estudiante 2 y se llama " +
estu.Name);
      Console.ReadKey();

            }
```

Listar todos los estudiantes:

```
// Obtener lista ordenada de estudiantes
using (var context = new SchoolContext())
            {
var estudiante = from s in context.Students
                              orderby s.Name
                              select s;

foreach (var item in estudiante)
            {
                Console.WriteLine("Estudiante {0} ", item.tName);
            }

Console.ReadKey();
```



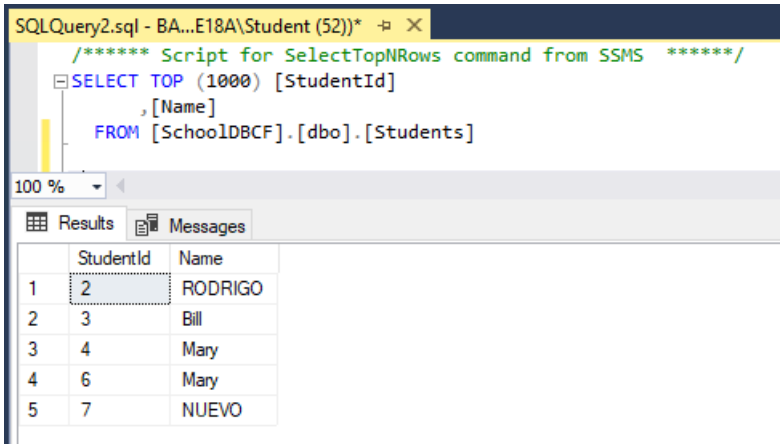
Actualización y eliminación de datos

Las operaciones de actualización, son CRUD: create, read, update y delete. El contexto automáticamente detecta los cambios y actualiza la información.

Haremos diversas modificaciones a los datos, y luego verficaremos el impacto de los cambios en la base de datos:

1) Agregar un nuevo estudiante
2) Modificar el nombre del estudiante 1
3) Eliminar el estudiante en la posición 10 de la colección (las colecciones comienzan en cero)

```
//CRUD
//Obtiene lista de estudiantes del contexto
using (var context = new SchoolContext())
{
   var listaEstudiantes = context.Students.ToList<Student>();

//Agrega un nuevo estudiante
context.Students.Add(new Student()
            {
                Name = "NUEVO",
            });

//Actualiza un estudiante
Student estudianteActualizado = listaEstudiantes.Where(s =>
s.StudentId == 2).FirstOrDefault<Student>();

estudianteActualizado.Name = "RODRIGO";


//Elimina un estudiante en posición 3
context.Students.Remove(listaEstudiantes.ElementAt<Student>(3));

//Ejecuta instrucciones Insert, Update & Delete en la base de datos
context.SaveChanges();
```



Para más información sobre instrucciones Linq, consulte este link:
https://www.tutorialsteacher.com/linq/what-is-linq