EBook Gratuito

APPRENDIMENTO

Entity Framework Core

Free unaffiliated eBook created from
Stack Overflow contributors.

#entity-framework-core

Sommario

Di

Capitolo 1: Iniziare con Entity Framework Core

Osservazioni

Entity Framework (EF) Core è una versione leggera ed estensibile della popolare tecnologia di accesso ai dati di Entity Framework.

EF Core è un mappatore di oggetti relazionale (O / RM) che consente agli sviluppatori .NET di lavorare con un database utilizzando oggetti .NET. Elimina la necessità della maggior parte del codice di accesso ai dati che gli sviluppatori di solito devono scrivere.

Examples

Aggiunta di pacchetti al progetto

Per aggiungere EntityFrameworkCore al progetto, aggiornare il file `project.json` (aggiungere nuove righe nelle sezioni delle `dependencies` e degli `tools`):

```
"dependencies": {
    ...
    "Microsoft.EntityFrameworkCore.SqlServer": "1.0.0",
    "Microsoft.EntityFrameworkCore.SqlServer.Design": "1.0.0",
    "Microsoft.EntityFrameworkCore.Design": {
      "version": "1.0.0",
      "type": "build"
    },
},
"tools": {
    ...
    "Microsoft.EntityFrameworkCore.Tools": "1.0.0-preview2-final"
}
```

Non dimenticare di eseguire il `dotnet restore` per scaricare effettivamente questi pacchetti da Internet.

Se si utilizza un RDBMS diverso da Microsoft SQLServer, sostituire `Microsoft.EntityFrameworkCore.SqlServer` con la versione corretta (`Microsoft.EntityFrameworkCore.Sqlite` , `Npgsql.EntityFrameworkCore.PostgreSQL` o altro - consultare la documentazione RDBMS per il pacchetto consigliato).

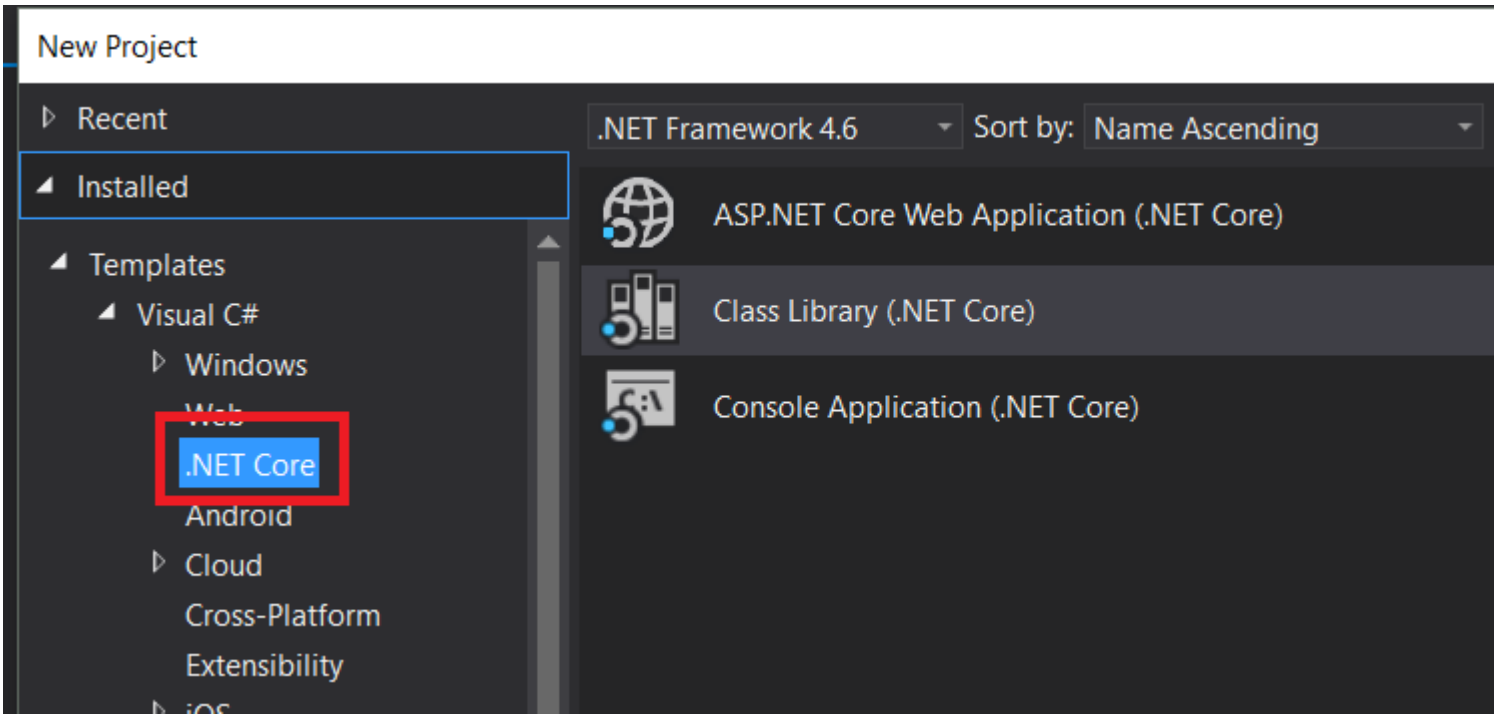
Primo database in Entity Framework Core con una libreria di classi e SQL Server

Okay, mi ci è voluto un giorno per capirlo, quindi sto postando i passi che ho seguito per far funzionare il mio Database in un `Class Project (.NET Core)` , con un'app Web .NET Core.

Passaggio 1: installare .NET Core

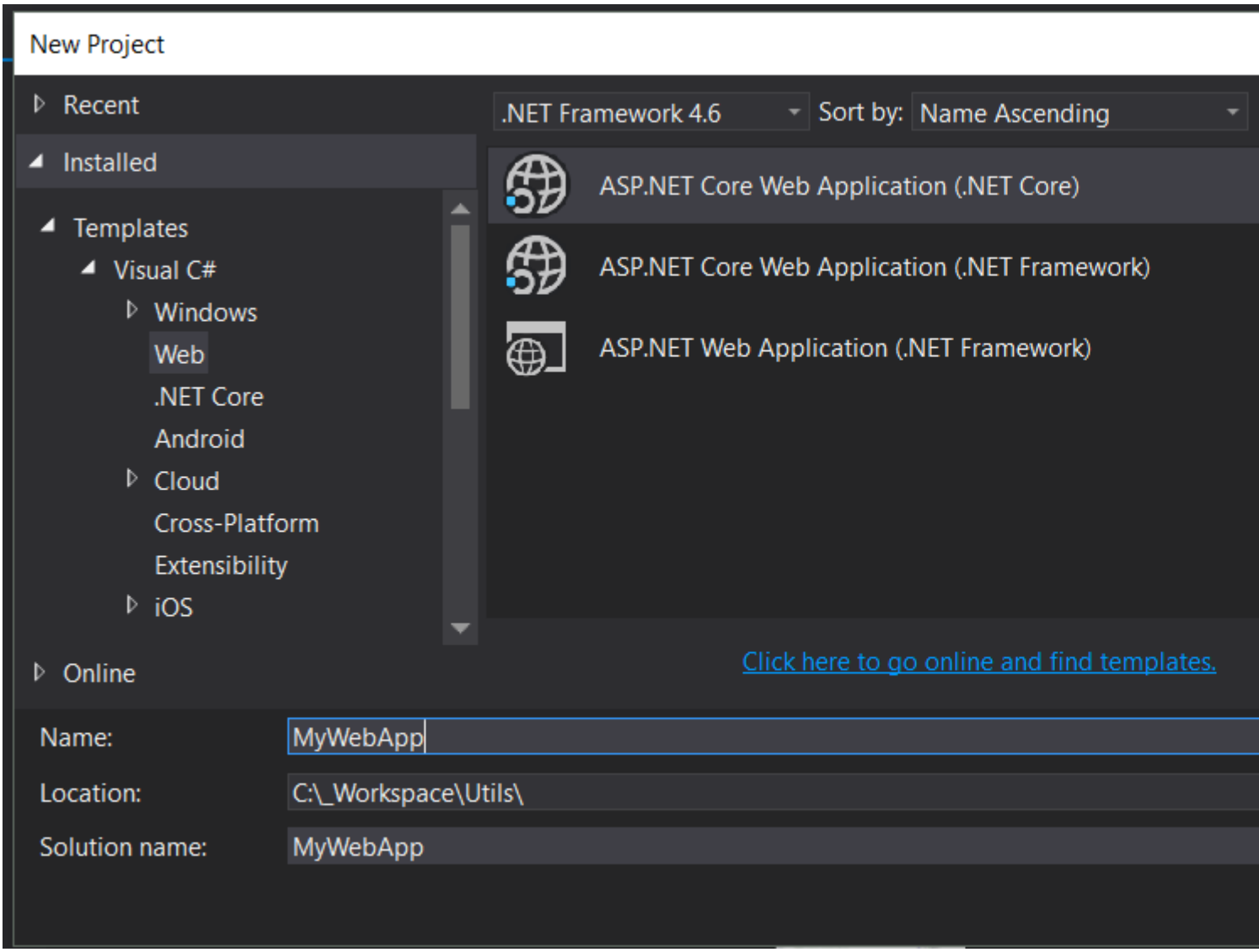
Assicurati di utilizzare .NET Core non DNX `(Hint: You should be able to see the .NET Core option when creating a New Project)` - Se NON Scarica da qui

Se hai problemi con l'installazione di .NET Core (l'errore è qualcosa come Visual Studio 2015 Update 3 non installato correttamente) - Puoi eseguire l'installazione usando il comando: [`DotNetCore.1.0.0-VS2015Tools.Preview2.exe SKIP_VSU_CHECK=1`] - Che impedirà l'installazione che esegue il problema di Github di controllo di Visual Studio

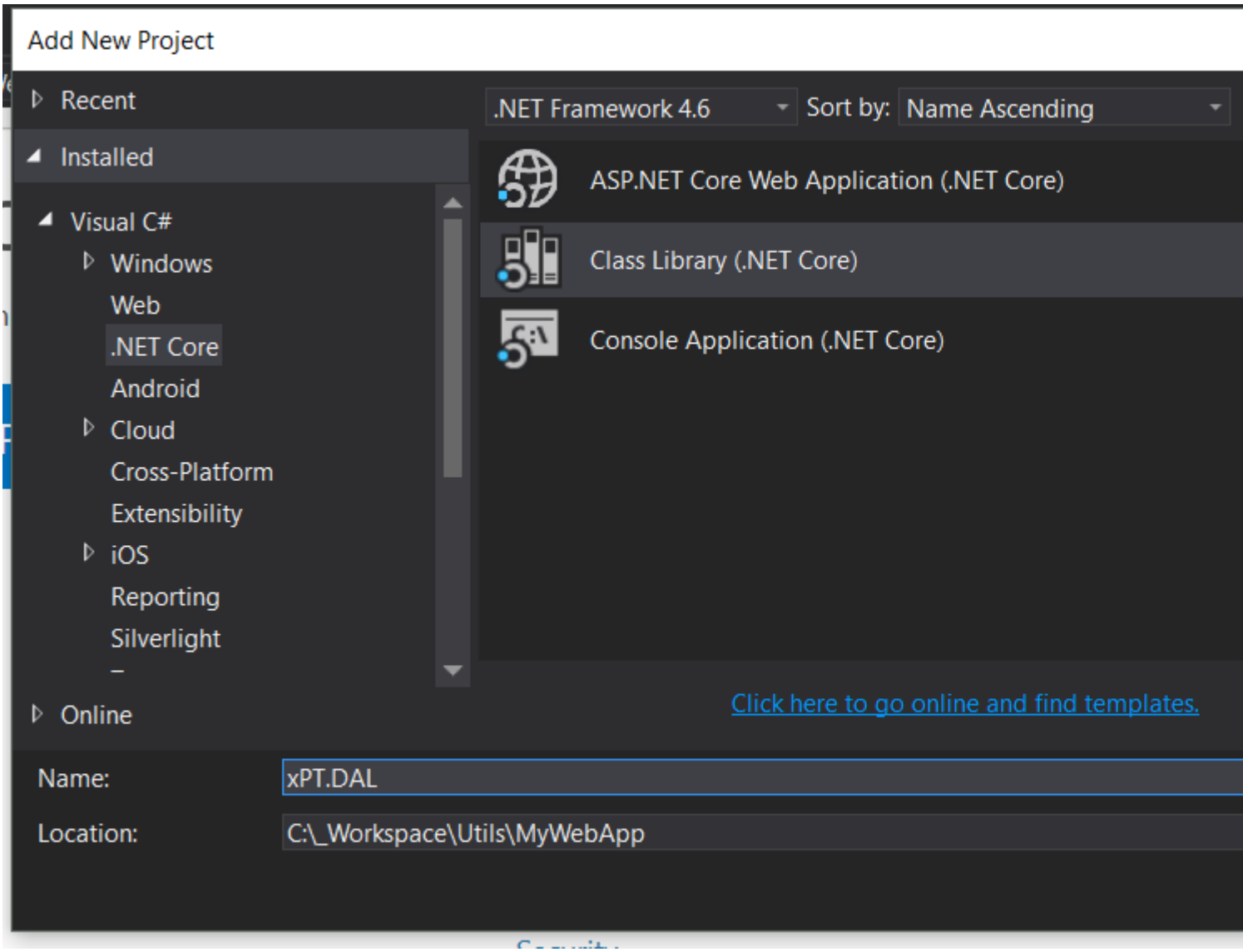


Passaggio 2: crea i progetti

Creare una nuova applicazione Web principale ASP.NET -> Quindi selezionare Applicazione Web nella schermata successiva



Aggiungi un progetto di `Class Library (.NET Core)`



Passaggio 3: installazione dei pacchetti EF

Aprire il file `project.json` della libreria di classi e incollare quanto segue, quindi salvare il file:

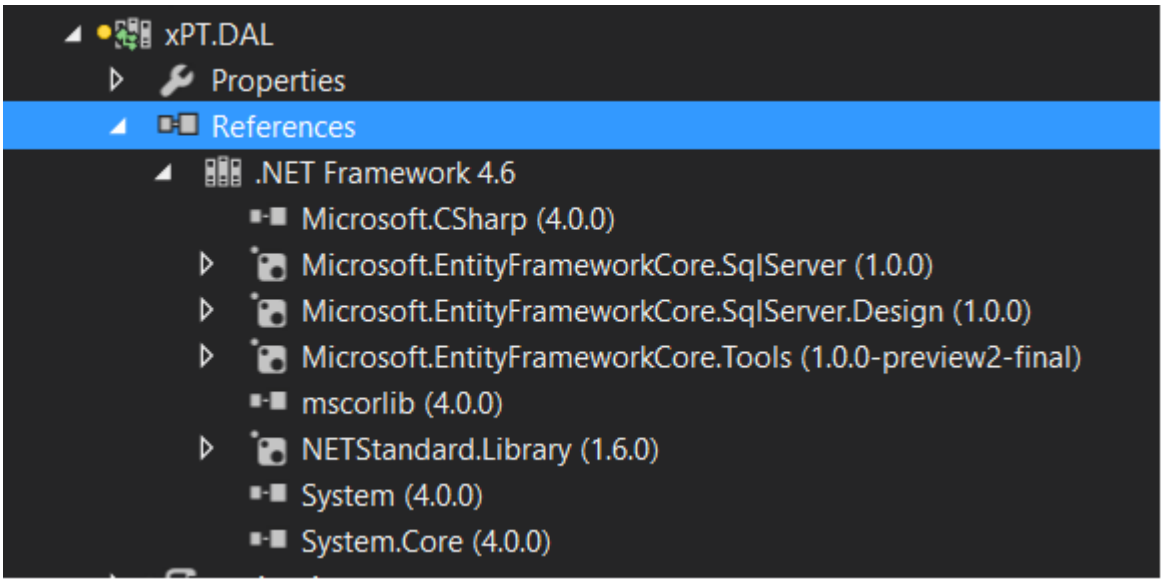
```
{
  "version": "1.0.0-*",

  "dependencies": {
    "Microsoft.EntityFrameworkCore.SqlServer": "1.0.0",
    "Microsoft.EntityFrameworkCore.SqlServer.Design": "1.0.0",
    "Microsoft.EntityFrameworkCore.Tools": "1.0.0-preview2-final",
    "NETStandard.Library": "1.6.0"
  },
  "tools": {
    "Microsoft.EntityFrameworkCore.Tools": "1.0.0-preview2-final"
  },

  "frameworks": {
    "net46": {
    },
    "netcoreapp1.0": {
      "dependencies": {
```

```
        "Microsoft.NETCore.App": {
          "type": "platform",
          "version": "1.0.0-*"
        }
      }
    }
  }
}
```

Questo dovrebbe ripristinare i pacchetti in `References`



---------- O

È possibile installarli utilizzando Nuget Package Manager eseguendo i seguenti comandi nella console di Gestione pacchetti

```
Install-Package Microsoft.EntityFrameworkCore.SqlServer

Install-Package Microsoft.EntityFrameworkCore.Tools -Pre

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

Nota: installare un pacchetto alla volta, se si verifica un errore dopo l'installazione

```
Microsoft.EntityFrameworkCore.Tools
```

Quindi modifica il contenuto della tua sezione `project.json` framework a questo:

```
  "frameworks": {
    "net46": {
    },
    "netcoreapp1.0": {
      "dependencies": {
        "Microsoft.NETCore.App": {
          "type": "platform",
          "version": "1.0.0-*"
```

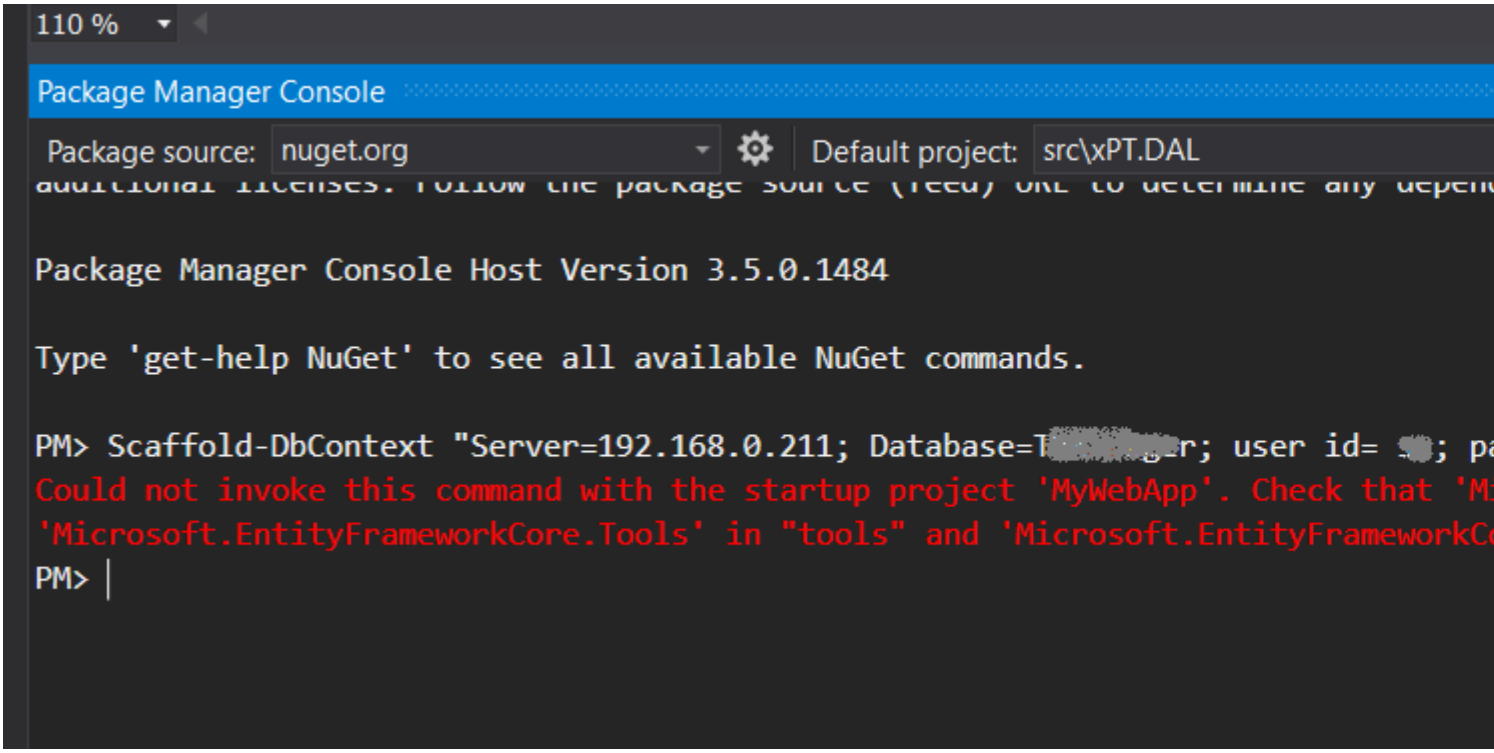
```
        }
      }
    }
  }
```

Passaggio 4: creazione del modello di database

Ora per generare il Database eseguire il seguente comando nella `Package Manager Console` (NON dimenticare di modificare la stringa di connessione nel database)

```
Scaffold-DbContext "Server=. ; Database=DATABASE; user id= USER ; password = PASSWORD;"
Microsoft.EntityFrameworkCore.SqlServer
```

Questo ti darà l'errore sul progetto di avvio:



Per questo è necessario aggiungere gli stessi riferimenti aggiunti alla Libreria di classi all'app Web .NET

Quindi apri il tuo `project.json` per l'app Web,

Sotto `dependencies` , aggiungere:

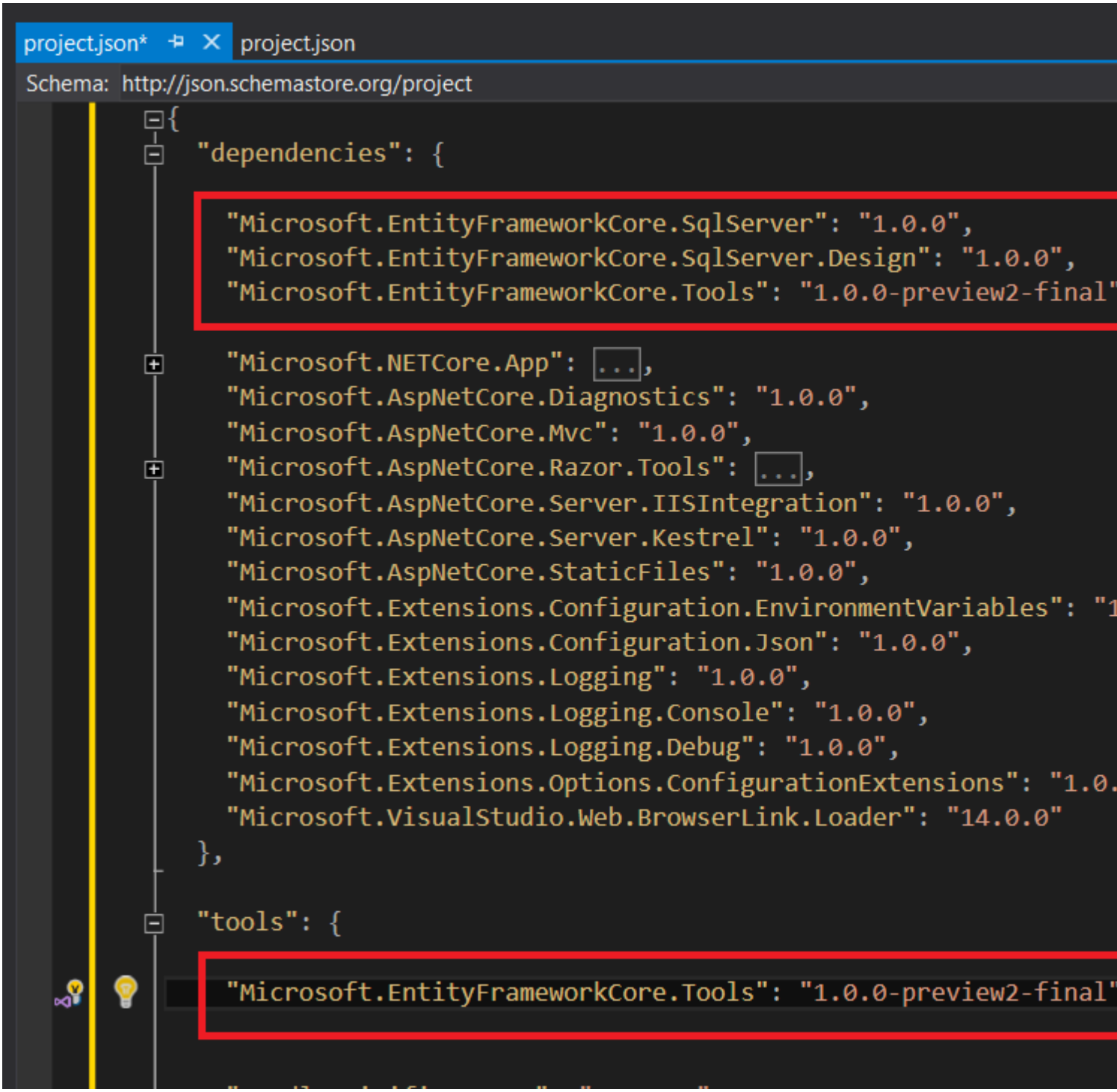
```
"Microsoft.EntityFrameworkCore.SqlServer": "1.0.0",
"Microsoft.EntityFrameworkCore.SqlServer.Design": "1.0.0",
"Microsoft.EntityFrameworkCore.Tools": "1.0.0-preview2-final",
```

e sotto gli `tools` aggiungi:

```
"Microsoft.EntityFrameworkCore.Tools": "1.0.0-preview2-final",
```

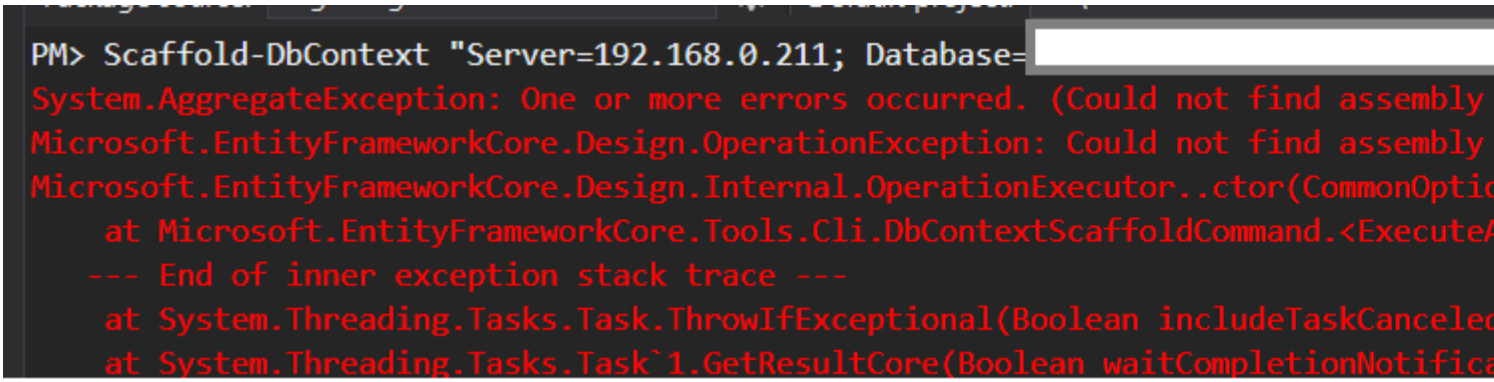
Dopo aver effettuato le modifiche, salva il file.

Questo è quello che sembra il mio project.json

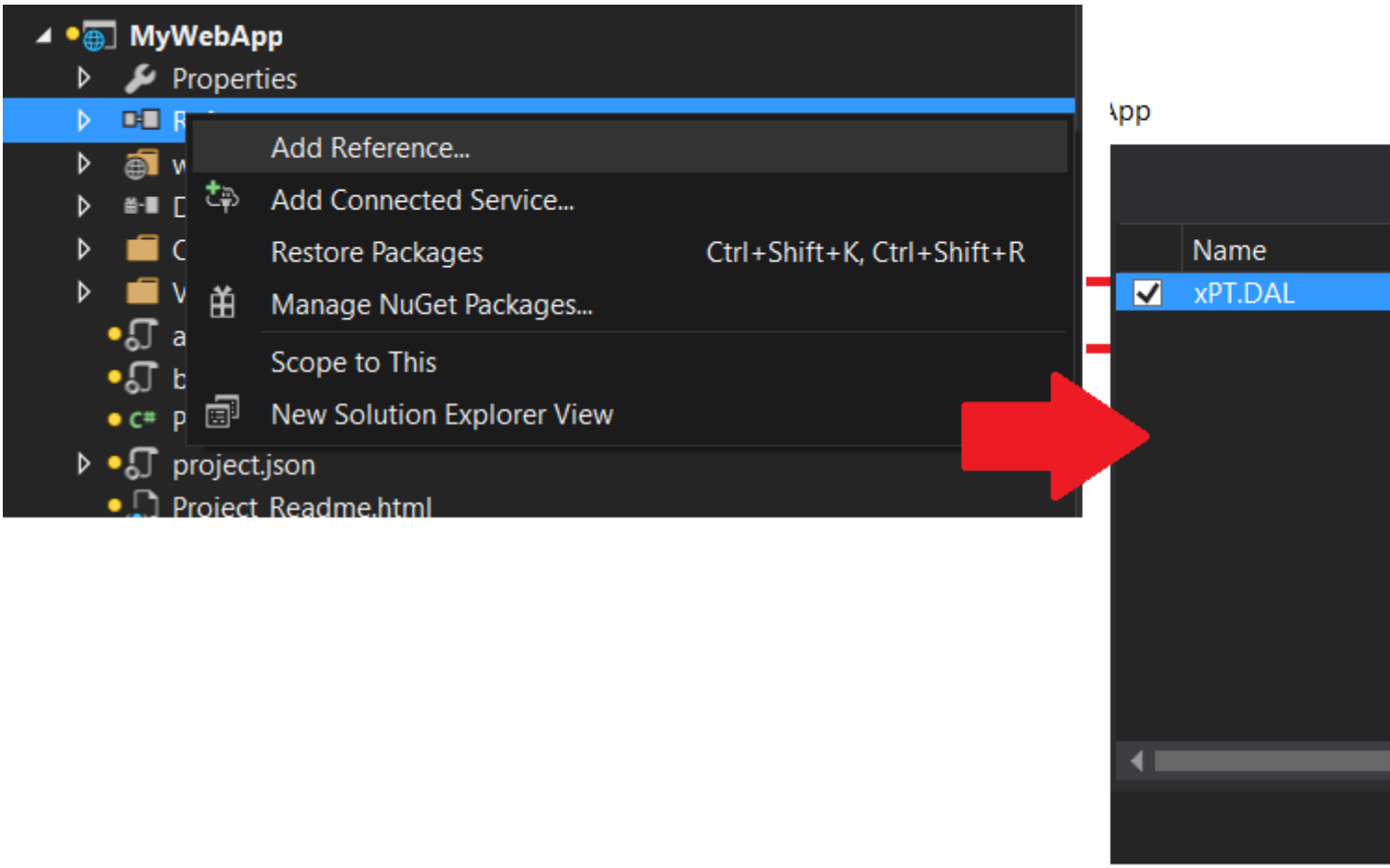


Quindi eseguire nuovamente il comando in Package Manager Console rispetto alla libreria di classi:

Se non hai già aggiunto il riferimento della tua libreria di classi all'app Web, riceverai questo errore:



per risolvere questo aggiungi un riferimento della tua biblioteca di classe alla tua app Web:

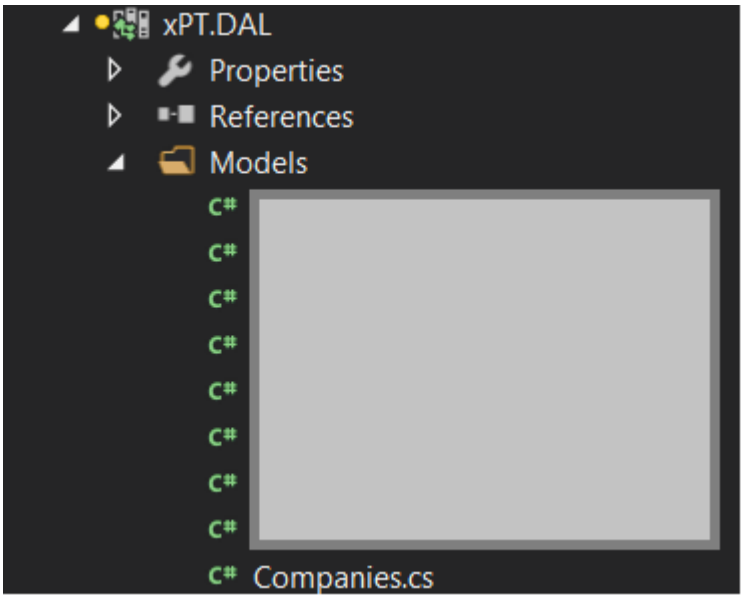


Finalmente

Esegui nuovamente il comando - nella `Package Manager Console` :

```
Scaffold-DbContext "Server=. ; Database=DATABASE; user id= USER ; password = PASSWORD;"
Microsoft.EntityFrameworkCore.SqlServer -OutputDir Models
```

Questo dovrebbe creare le Entità sotto la cartella Modelli, nella libreria delle classi



Passare una stringa di connessione

Nel mio caso, abbiamo un'applicazione multi-tenant, in cui ogni cliente ha il proprio database, ad esempio Client_1, Client_2, Client_3. Quindi la stringa di connessione doveva essere dinamica.

Quindi abbiamo aggiunto una proprietà stringa di connessione a un costruttore e l'abbiamo passata al contesto nel metodo `OnConfiguring`

```
public partial class ClientContext
{
    private readonly string _connectionString;

    public ClientContext(string connectionString) : base()
    {
        _connectionString = connectionString;
    }

    protected override void OnConfiguring(DbContextOptionsBuilder optionsBuilder)
    {
        optionsBuilder.UseSqlServer(_connectionString);
    }
}
```

e usato in questo modo:

```
    public void TestConnection()
    {
        var clientId = 1;

        var connectionString = string.Format("Server=192.168.0.211; Database=Client_{0}; user
id= USER; password = PWD;", clientId);

        using (var clientContext = new ClientContext(connectionString))
        {
            var assets = clientContext.Users.Where(s => s.UserId == 1);
        }
```

```
    }
```

Modello, interrogazione e salvataggio dei dati

Modello

Con EF Core, l'accesso ai dati viene eseguito utilizzando un modello. Un modello è costituito da classi di entità e un contesto derivato che rappresenta una sessione con il database, consentendo di eseguire query e salvare dati.

È possibile generare un modello da un database esistente, codificare manualmente un modello in modo che corrisponda al proprio database o utilizzare EF Migrations per creare un database dal modello (e modificarlo man mano che il modello cambia nel tempo).

```
using Microsoft.EntityFrameworkCore;
using System.Collections.Generic;

namespace Intro
{
    public class BloggingContext : DbContext
    {
        public DbSet<Blog> Blogs { get; set; }
        public DbSet<Post> Posts { get; set; }

        protected override void OnConfiguring(DbContextOptionsBuilder optionsBuilder)
        {

optionsBuilder.UseSqlServer(@"Server=(localdb)\mssqllocaldb;Database=MyDatabase;Trusted_Connection=True

        }
    }

    public class Blog
    {
        public int BlogId { get; set; }
        public string Url { get; set; }

        public List<Post> Posts { get; set; }
    }

    public class Post
    {
        public int PostId { get; set; }
        public string Title { get; set; }
        public string Content { get; set; }

        public int BlogId { get; set; }
        public Blog Blog { get; set; }
    }
}
```

Interrogazione

Le istanze delle classi di entità vengono recuperate dal database utilizzando Language Integrated Query (LINQ).

```
using (var db = new BloggingContext())
{
    var blogs = db.Blogs
        .Where(b => b.Rating > 3)
        .OrderBy(b => b.Url)
        .ToList();
}
```

Salvataggio dei dati

I dati vengono creati, eliminati e modificati nel database utilizzando le istanze delle classi di entità.

```
using (var db = new BloggingContext())
{
    var blog = new Blog { Url = "http://sample.com" };
    db.Blogs.Add(blog);
    db.SaveChanges();
}
```

Eliminazione dei dati

Le istanze delle classi di entità vengono recuperate dal database utilizzando Language Integrated Query (LINQ).

```
using (var db = new BloggingContext())
{
    var blog = new Blog { Url = "http://sample.com" };
    db.Blogs.Attach(blog);
    db.Blogs.Remove(blog);
    db.SaveChanges();
}
```

Aggiornamento dei dati

I dati vengono aggiornati nel database utilizzando le istanze delle classi di entità.

```
using (var db = new BloggingContext())
{
    var blog = new Blog { Url = "http://sample.com" };
    var entity = db.Blogs.Find(blog);
    entity.Url = "http://sample2.com";
    db.SaveChanges();
}
```

Leggi Iniziare con Entity Framework Core online: https://riptutorial.com/it/entity-framework-core/topic/3796/iniziare-con-entity-framework-core

Capitolo 2: Aggiornamento di una relazione da molti a molti

introduzione

Come aggiornare una relazione da molti a molti in EF Core:

Examples

MVC POST Modifica esempio

Supponiamo di avere una classe di prodotto con più colori che può essere su molti prodotti.

```
public class Product
{
    public int ProductId { get; set; }
    public ICollection<ColorProduct> ColorProducts { get; set; }
}

public class ColorProduct
{
    public int ProductId { get; set; }
    public int ColorId { get; set; }

    public virtual Color Color { get; set; }
    public virtual Product Product { get; set; }
}

public class Color
{
    public int ColorId { get; set; }
    public ICollection<ColorProduct> ColorProducts { get; set; }
}
```

Usando questa estensione per renderla più facile:

```
public static class Extensions
{
    public static void TryUpdateManyToMany<T, TKey>(this DbContext db, IEnumerable<T>
currentItems, IEnumerable<T> newItems, Func<T, TKey> getKey) where T : class
    {
        db.Set<T>().RemoveRange(currentItems.Except(newItems, getKey));
        db.Set<T>().AddRange(newItems.Except(currentItems, getKey));
    }

    public static IEnumerable<T> Except<T, TKey>(this IEnumerable<T> items, IEnumerable<T>
other, Func<T, TKey> getKeyFunc)
    {
        return items
            .GroupJoin(other, getKeyFunc, getKeyFunc, (item, tempItems) => new { item,
tempItems })
```

```
            .SelectMany(t => t.tempItems.DefaultIfEmpty(), (t, temp) => new { t, temp })
            .Where(t => ReferenceEquals(null, t.temp) || t.temp.Equals(default(T)))
            .Select(t => t.t.item);
    }
}
```

L'aggiornamento dei colori di un prodotto sarebbe simile a questo (un metodo POST di modifica MVC)

```
[HttpPost]
public IActionResult Edit(ProductVm vm)
{
if (ModelState.IsValid)
    {
        var model = db.Products
            .Include(x => x.ColorProducts)
            .FirstOrDefault(x => x.ProductId == vm.Product.ProductId);

        db.TryUpdateManyToMany(model.ColorProducts, vm.ColorsSelected
            .Select(x => new ColorProduct
            {
                ColorId = x,
                ProductId = vm.Product.ProductId
            }), x => x.ColorId);

        db.SaveChanges();

        return RedirectToAction("Index");
    }
   return View(vm);
}


public class ProductVm
{
    public Product Product { get; set; }

    public IEnumerable<int> ColorsSelected { get; set; }
}
```

Il codice è stato semplificato il più possibile, senza proprietà aggiuntive su nessuna classe.

Leggi Aggiornamento di una relazione da molti a molti online: https://riptutorial.com/it/entity-framework-core/topic/9527/aggiornamento-di-una-relazione-da-molti-a-molti

Capitolo 3: EF Core vs EF6.x

Osservazioni

Per gli ultimi aggiornamenti, fare riferimento a: Confronto delle caratteristiche

Examples

Confronto affiancato

La seguente tabella mette a confronto le funzioni disponibili (1) in EF Core e EF6.x.

Ha lo scopo di fornire un confronto ad alto livello e non elenca tutte le funzionalità, o tenta di fornire dettagli sulle possibili differenze tra il modo in cui funziona la stessa funzione.

Creare un modello	EF6.x	EF Core 1.0.0
Modellazione di base (classi, proprietà, ecc.)	sì	sì
Convegni	sì	sì
Convenzioni personalizzate	sì	Parziale
Annotazioni di dati	sì	sì
API fluente	sì	sì
Eredità: tabella per gerarchia (TPH)	sì	sì
Eredità: tabella per tipo (TPT)	sì	
Eredità: tabella per classe di calcestruzzo (TPC)	sì	
Proprietà dello stato dell'ombra		sì
Chiavi alternative		sì
Molti-a-molti: con l'entità join	sì	sì
Molti-a-molti: senza join entità	sì	
Generazione di chiavi: database	sì	sì
Generazione della chiave: cliente		sì
Tipi complessi / valore	sì	
Dati spaziali	sì	

Creare un modello	EF6.x	EF Core 1.0.0
Visualizzazione grafica del modello	sì	
Editor trascina / rilascia grafico	sì	
Formato del modello: codice	sì	sì
Formato del modello: EDMX (XML)	sì	
Modello di reverse engineering dal database: riga di comando		sì
Modello di reverse engineering dal database: procedura guidata VS.	sì	
Modello di aggiornamento incrementale dal database	sì	

Dati di query	EF6.x	EF Core 1.0.0
LINQ: query semplici	Stabile	Stabile
LINQ: query moderate	Stabile	Stabilizzazione
LINQ: query complesse	Stabile	In corso
LINQ: query utilizzando le proprietà di navigazione	Stabile	In corso
Generazione SQL "carina"	Povero	sì
Valutazione mista client / server		sì
Caricamento dei dati relativi: desideroso	sì	sì
Caricamento dei dati relativi: pigro	sì	
Caricamento dati correlati: esplicito	sì	
Query SQL non elaborate: tipi di modello	sì	sì
Query SQL non elaborate: tipi non mappati	sì	
Query SQL non elaborate: composizione con LINQ		sì

Salvataggio dei dati	EF6.x	EF Core 1.0.0
Salva I Cambiamenti	sì	sì
Modifica del tracciamento: istantanea	sì	sì
Change tracking: notifica	sì	sì
Accesso allo stato tracciato	sì	Parziale

Salvataggio dei dati	EF6.x	EF Core 1.0.0
Concorrenza ottimistica	sì	sì
Le transazioni	sì	sì
Raccolta di dichiarazioni		sì
Procedura memorizzata	sì	
Supporto grafico indipendente (N-Tier): API a basso livello	Povero	sì
Supporto grafico indipendente (N-Tier): end-to-end		Povero

Altre caratteristiche	EF6.x	EF Core 1.0.0
migrazioni	sì	sì
API di creazione / cancellazione del database	sì	sì
Dati seme	sì	
Resilienza della connessione	sì	
Ganci del ciclo di vita (eventi, intercettazione comandi, ...)	sì	

Provider di database	EF6.x	EF Core 1.0.0
server SQL	sì	sì
MySQL	sì	Solo pagato, non retribuito in arrivo (2)
PostgreSQL	sì	sì
Oracolo	sì	Solo pagato, non retribuito in arrivo (2)
SQLite	sì	sì
SQL Compact	sì	sì
DB2	sì	sì
InMemory (per test)		sì
Archiviazione tavolo di Azure		Prototipo
Redis		Prototipo

Modelli di applicazione	EF6.x	EF Core 1.0.0
WinForms	sì	sì

Modelli di applicazione	EF6.x	EF Core 1.0.0
WPF	sì	sì
consolle	sì	sì
ASP.NET	sì	sì
ASP.NET Core		sì
Xamarin		Disponibile a breve (3)
UWP		sì

Note:

(1): A partire dal 2016/10/18

(2): i fornitori pagati sono disponibili, i fornitori non pagati sono stati elaborati. I team che lavorano sui provider non pagati non hanno condiviso i dettagli pubblici della cronologia, ecc.

(3): EF Core è progettato per funzionare su Xamarin quando il supporto per .NET Standard è abilitato in Xamarin.

Leggi EF Core vs EF6.x online: https://riptutorial.com/it/entity-framework-core/topic/7513/ef-core-vs-ef6-x

Titoli di coda

S. No	Capitoli	Contributors
1	Iniziare con Entity Framework Core	Community, Dawood Awan, Dmitry, hasan, natemcmaster, NovaDev, tmg, uTeisT
2	Aggiornamento di una relazione da molti a molti	Paw Ormstrup Madsen
3	EF Core vs EF6.x	Frédéric, Ruud Lenders, uTeisT