# EF Core

Sommario

[EF Core 1](#_Toc22111547)

[Install 2](#_Toc22111548)

[Scaffolding del database 2](#_Toc22111549)

[Creare migrazioni 2](#_Toc22111550)

[Aggiornare il Database 2](#_Toc22111551)

[Rimuovere una migrazione 2](#_Toc22111552)

[Ripristinare una migrazione 3](#_Toc22111553)

[Generare ScriptSQL 3](#_Toc22111554)

[DbContext 4](#_Toc22111555)

[Querying in Entity Framework Core 4](#_Toc22111556)

[Eager Loading 4](#_Toc22111557)

[Multiple Include 5](#_Toc22111558)

[ThenInclude 5](#_Toc22111559)

[Insert Data 5](#_Toc22111560)

[Updating Data 5](#_Toc22111561)

[Deleting Data 6](#_Toc22111562)

[Working with Stored Procedure in Entity Framework Core 6](#_Toc22111563)

[Eseguire una SP 6](#_Toc22111564)

[Installare System.Linq.Dynamic.Core (NuGet) 7](#_Toc22111565)

[Definire una relazione one-to-many 7](#_Toc22111566)

[Definizione della relazione Fluent API one-to-many 8](#_Toc22111567)

[Definizione di una relazione one-to-one 8](#_Toc22111568)

[Definizione della relazione Fluent API one-to-one 8](#_Toc22111569)

## Install

Da NuGet installare quanto segue :

1. Microsoft.EntityFrameworkCore.SqlServer
2. Microsoft.EntityFrameworkCore.Tools

Si possono eseguire comandi da riga comando e si possono attivare tramite :

**dotnet ef**

## Scaffolding del database

E’ possibile eseguire la creazione del modello a partire dal db eseguendo il comando :

**Scaffold-DbContext [-Connection] [-Provider] [-OutputDir] [-Context] [-Schemas>] [-Tables>]**

**[-DataAnnotations] [-Force] [-Project] [-StartupProject] [<CommonParameters>]**

Ad esempio :

**Scaffold-DbContext "Server=DESKTOP-5G6VD0K\SQLEXPRESS;Database=CeccoLab;Trusted\_Connection=True;" Microsoft.EntityFrameworkCore.SqlServer -OutputDir Models**

Nella cartella Models verranno create le classi che rappresentano le entità del Db ed in più la classe che eredita il DbContext.

## Creare migrazioni

Da PowerShell :

Add-Migration nomemigrazione

Da console :

dotnet ef migrations add nomemigrazione

Ad esempio:

dotnet ef migrations add Studenti -c CeccoLabContext

## Aggiornare il Database

Da PowerShell :

Update-Database

Da console :

dotnet ef database update

## Rimuovere una migrazione

***Per rimuovere l'ultima migrazione, usare questo comando***:

Da PowerShell :

Remove-Migration

Da console :

dotnet ef database remove

## Ripristinare una migrazione

Da PowerShell :

Update-Database LastGoodMigration (indicare l’ultima migrazione buona)

Da console :

dotnet ef database update LastGoodMigration

e poi eseguire il Remove-Migration.

## Generare ScriptSQL

Da PowerShell :

Script-Migration

Da console :

dotnet ef database script

## DbContext

La classe DbContext è parte integrante di Entity Framework. Un'istanza di DbContext rappresenta una sessione con il database che può essere utilizzata per eseguire query e salvare istanze delle entità in un database. DbContext è una combinazione dei pattern Unit Of Work e Repository.

DbContext in EF Core ci consente di svolgere le seguenti attività:

1. Gestire la connessione al database
2. Configura modello e relazione
3. Database di query
4. Salvataggio dei dati nel database
5. Configura il rilevamento delle modifiche
6. caching
7. Gestione delle transazioni

Per usare DbContext nella nostra applicazione, dobbiamo creare la classe che deriva da DbContext, nota anche come classe di contesto. Questa classe di contesto include in genere le proprietà <TEntity> DbSet per ciascuna entità nel modello. Considera il seguente esempio di classe di contesto in EF Core.

## Querying in Entity Framework Core

EF Core ha una nuova funzionalità in LINQ-to-Entities in cui possiamo includere funzioni C# nella query.

private static void Main(string[] args)

{

var context = new SchoolContext();

var studentsWithSameName = context.Students

.Where(s => s.FirstName == GetName())

.ToList();

}

public static string GetName() {

return "Bill";

}

## Eager Loading

E’ possibile effettuare il caricamento immediato dei dati utilizzando l’ Include()

var studentWithGrade = context.Students

.Where(s => s.FirstName == "Bill")

.Include(s => s.Grade)

.FirstOrDefault();

l’include deve essere obbligatoriamente utilizzato dopo il FromSql se utilizzato :

var studentWithGrade = context.Students

.FromSql("Select \* from Students where FirstName ='Bill'")

.Include(s => s.Grade)

.FirstOrDefault();

## Multiple Include

Utilizzare il metodo Include() più volte per caricare più proprietà di navigazione della stessa entità.

Ad esempio, il codice seguente carica le entità relative a Grado e StudentCourses di Student.

var studentWithGrade = context.Students.Where(s => s.FirstName == "Bill")

.Include(s => s.Grade)

.Include(s => s.StudentCourses)

.FirstOrDefault();

## ThenInclude

EF Core introduzione la nuova estensione ThenInclude() per caricare più livelli di entità correlate, ad esempio:

var student = context.Students.Where(s => s.FirstName == "Bill")

.Include(s => s.Grade)

.ThenInclude(g => g.Teachers)

.FirstOrDefault();

## Insert Data

using (var context = new SchoolContext())

{

var std = new Student()

{

FirstName = "Bill",

LastName = "Gates"

};

context.Students.Add(std);

// or

// context.Add<Student>(std);

context.SaveChanges();

}

## Updating Data

using (var context = new SchoolContext())

{

var std = context.Students.First<Student>();

std.FirstName = "Steve";

context.SaveChanges();

}

## Deleting Data

using (var context = new SchoolContext())

{

var std = context.Students.First<Student>();

context.Students.Remove(std);

// or

// context.Remove<Student>(std);

context.SaveChanges();

}

## Working with Stored Procedure in Entity Framework Core

## Eseguire una SP

Di seguito elenco i passi per eseguire un SP e raccogliere i dati in una classe che non è un’entità.

1. Definizione della classe

public class Testing

{

public int Id { get; set; }

public string Name { get; set; }

}

1. Nella classe context inserire la seguente riga :

public virtual **DbQuery**<Testing> Testing { get; set; }

1. Eseguire la Stored Procedure

var Testing = \_context.Testing.FromSql("exec testing");

## Installare System.Linq.Dynamic.Core (NuGet)

Consente la costruzione di query dinamiche, ad esempio :

string filtro1 = "Id == 1";

var query1 = \_context.Users

.Where("Id == @0", 1)

.Select("new(UserName, Avatar)").FirstOrDefault();

var query2 = \_context.Users

.Where(filtro1)

.Select("new(UserName, Avatar)").FirstOrDefault();

string filtro2 = "Username.Contains(\"Cecc\") && UserName.Contains(\"Mary\")";

List<Users> lst = \_context.Users.Where(filtro2).OrderBy("UserName").ToList();

Oppure in modo più complesso tramite la classe, che ho implementato, BuildLinqDynamicExpressionFilterModel.

## Definire una relazione one-to-many

L’obiettivo è quello di mettere in relazione la tabella Student (many) con la tabella Grade (one)

Di seguito la definizione delle tabelle :

public class Student

{

public int Id { get; set; }

public string Name { get; set; }

public int GradeId { get; set; } // Questa è la colonna di ForeignKey con la tabella // Grade

public Grade Grade { get; set; } // Tabella in join

}

public class Grade

{

public int Id { get; set; }

public string GradeName { get; set; }

public string Section { get; set; }

public ICollection<Student> Students { get; set; } // Tabella in join

}

Nel context occorre definire chiave primaria di tipo identity :

modelBuilder.Entity<Student>(entity =>

{

entity.HasKey(s => s.Id); // Primary Key

});

modelBuilder.Entity<Student>().Property(p => p.Id).ValueGeneratedOnAdd(); //Identity

modelBuilder.Entity<Grade>(entity =>

{

entity.HasKey(s => s.Id);

});

modelBuilder.Entity<Grade>().Property(p => p.Id).ValueGeneratedOnAdd();

## Definizione della relazione Fluent API one-to-many

// Definizione della relazione di tipo one-to-many

modelBuilder.Entity<Student>()

.HasOne<Grade>(s => s.Grade)

.WithMany(g => g.Students)

.HasForeignKey(s => s.GradeId);

## Definizione di una relazione one-to-one

L’obiettivo è quello di mettere in relazione la tabella Student (one) con la tabella StudentAddress (one).

Di seguito la definizione della nuova tabella StudentAddress.

public class StudentAddress

{

public int Id { get; set; }

public string Address { get; set; }

public string City { get; set; }

public string State { get; set; }

public string Country { get; set; }

public int StudentId { get; set; }

public Student Student { get; set; }

}

Nel context occorre definire chiave primaria di tipo identity :

modelBuilder.Entity<StudentAddress>(entity =>

{

entity.HasKey(s => s.Id);

});

modelBuilder.Entity<StudentAddress>().Property(p => p.Id).ValueGeneratedOnAdd();

## Definizione della relazione Fluent API one-to-one

modelBuilder.Entity<Student>()

.HasOne<StudentAddress>(s => s.Address)

.WithOne(g => g.Student)

.HasForeignKey<StudentAddress>(s => s.StudentId);