

Entity Framework



Antonia Sacchitella

Analyst @icubedsrl

antonia.Sacchitella@icubed.it



Entity Framework

ORM di Microsoft basato sul .NET Framework

Insieme di tecnologie ADO.NET per lo sviluppo software

Definisce un modello di astrazione dei dati

Traduce il nostro codice in query comprensibili dal DBMS

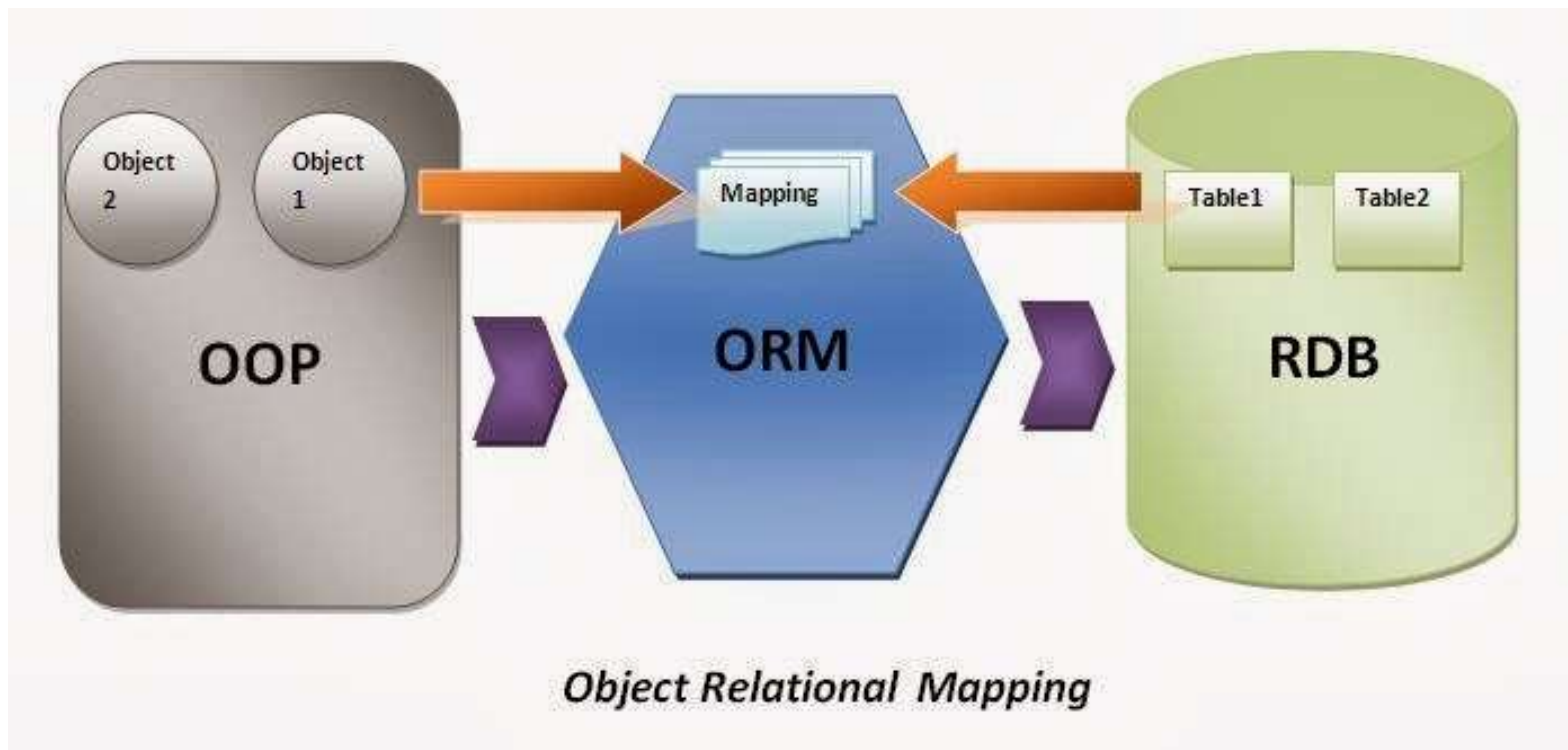
Disaccoppiamento tra applicazione e dati

- Posso mantenere la stessa rappresentazione anche se cambia il modello fisico (es. da SQL Server ad Oracle)

Open source

- <https://github.com/aspnet/EntityFramework>

Cos'è un ORM?

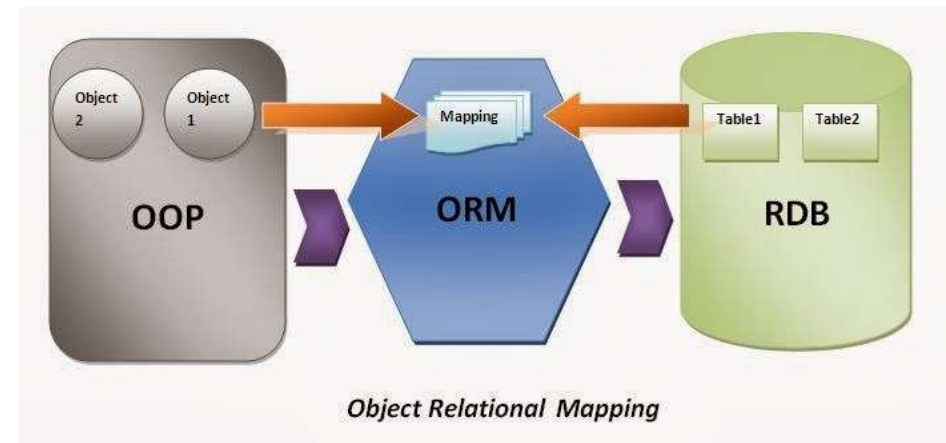


Cos'è un ORM?

È una tecnica per convertire dati da type system incompatibili
Da database ad object-oriented

3 caratteristiche fondamentali

- **Mapping**
 - Definisce come il database si «incastra» negli oggetti e viceversa
- **Fetching**
 - Sa come recuperare i dati dal database e materializzare i rispettivi oggetti
- **Persistenza del grafo**
 - Sa come salvare le modifiche agli oggetti, generando le query SQL corrispondenti



Entity Framework

Entity Client Data Provider

- Livello di astrazione, che rende utilizzabile EF con più sorgenti dati

Entity Data Model

- Rappresenta il modello di mapping tra database ed oggetti

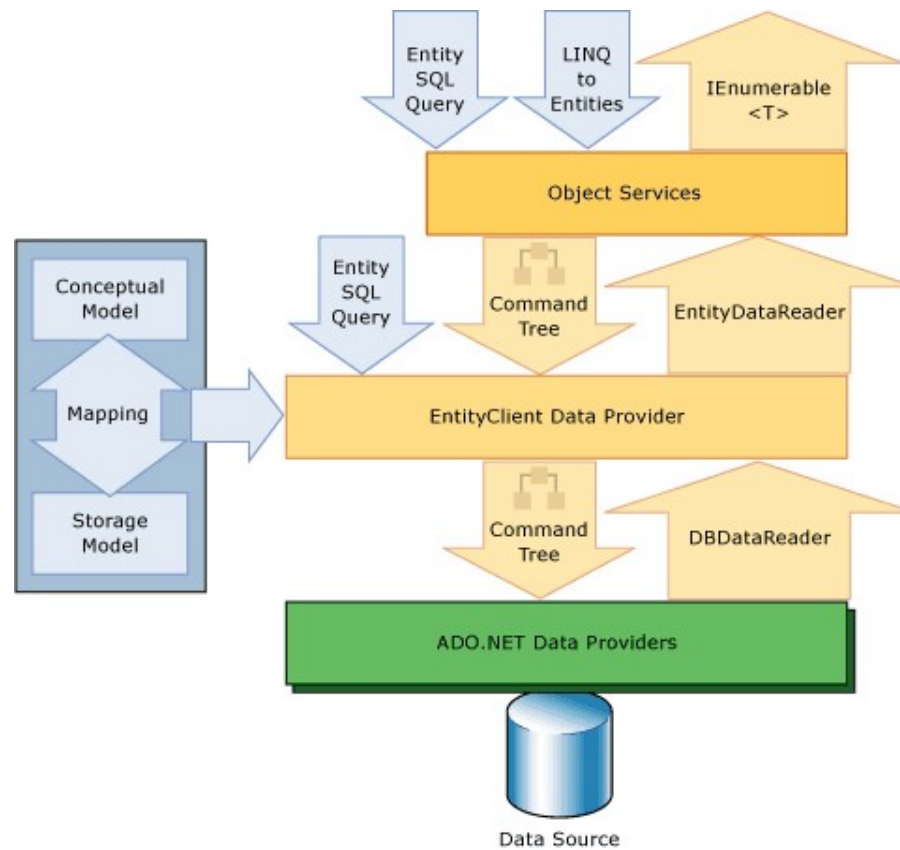
LINQ to Entities

- Flavour di LINQ che consente di utilizzare tutti gli operatori in unione con le entity di EF

Entity SQL

- Linguaggio speciale per interrogare EF con un linguaggio indipendente dal database utilizzato, consentendo di creare facilmente query dinamiche

Come funziona



Diversi approcci

Database-First

- Il modello viene importato da un DB esistente
- Se modifico il database posso (quasi) sempre aggiornare il modello

Model-First

- Il modello del database viene creato dal designer di Visual Studio
- L'implementazione fisica è basata sul modello generato
- Non favorisce il riutilizzo del codice né la separazione tra contesto ed entità
- Poichè il modello definisce il DB, eventuali sue modifiche verranno perse

Code-First

- Il modello viene creato dal nostro codice
- L'implementazione fisica è basata sul nostro codice

Perché Code-First?

Focus sul domain design

C# potrebbe risultarci più familiare delle query di SQL

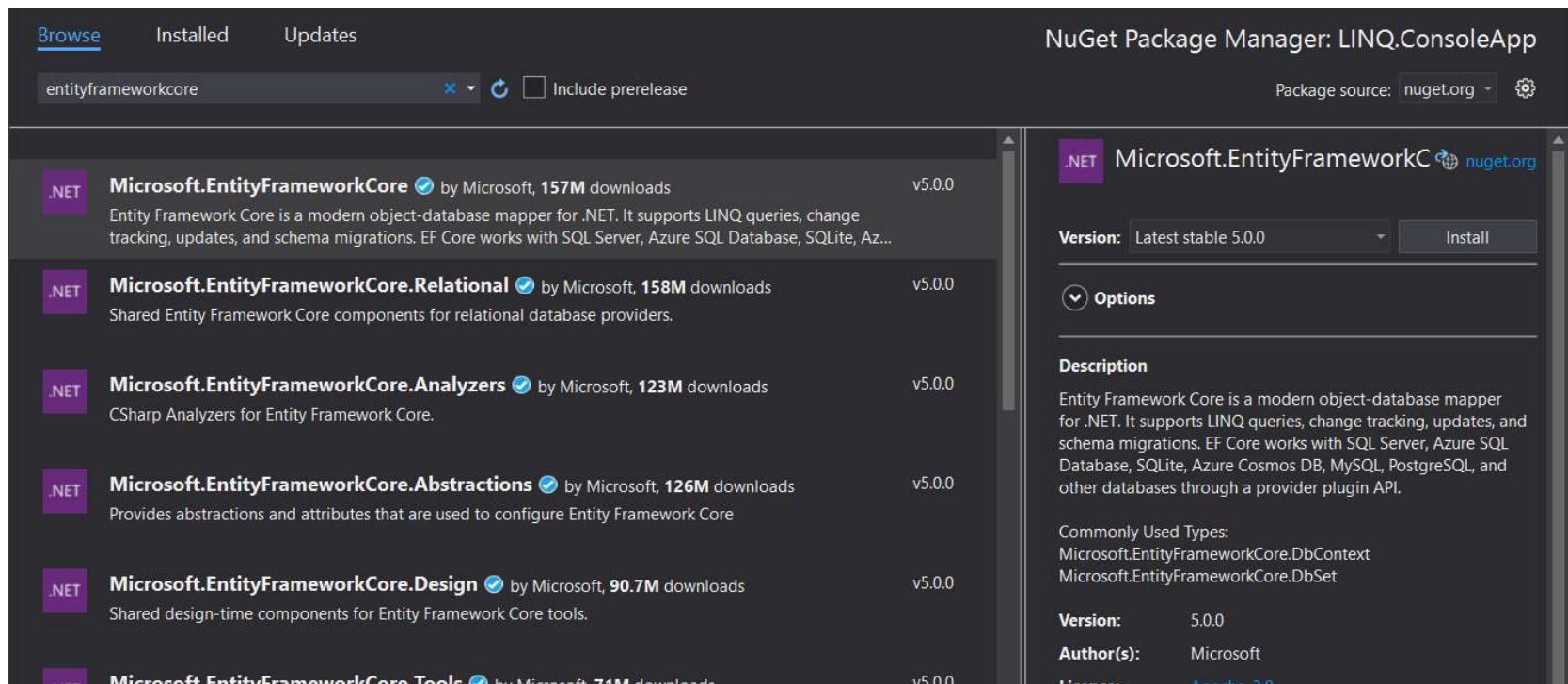
- E sarebbe l'unico linguaggio da apprendere

Possiamo mettere facilmente sotto source control il nostro database (niente script SQL solo codice C#)

Evitiamo la mole di codice auto generato da EDMX

Se scegliamo di sviluppare in .NET Core, l'EDMX non è supportato

Configurazione di EF



The screenshot displays the NuGet Package Manager interface for a project named 'LINQ.ConsoleApp'. The search bar contains 'entityframeworkcore'. The results list several packages by Microsoft, all at version 5.0.0. The package 'Microsoft.EntityFrameworkCore' is selected, showing its description, options, and commonly used types.

Package Name	Author	Downloads	Version
Microsoft.EntityFrameworkCore	Microsoft	157M	v5.0.0
Microsoft.EntityFrameworkCore.Relational	Microsoft	158M	v5.0.0
Microsoft.EntityFrameworkCore.Analyzers	Microsoft	123M	v5.0.0
Microsoft.EntityFrameworkCore.Abstractions	Microsoft	126M	v5.0.0
Microsoft.EntityFrameworkCore.Design	Microsoft	90.7M	v5.0.0
Microsoft.EntityFrameworkCore.Tools	Microsoft	74M	v5.0.0

Microsoft.EntityFrameworkCore by Microsoft, 157M downloads, v5.0.0

Entity Framework Core is a modern object-database mapper for .NET. It supports LINQ queries, change tracking, updates, and schema migrations. EF Core works with SQL Server, Azure SQL Database, SQLite, Az...

Options

Description

Entity Framework Core is a modern object-database mapper for .NET. It supports LINQ queries, change tracking, updates, and schema migrations. EF Core works with SQL Server, Azure SQL Database, SQLite, Azure Cosmos DB, MySQL, PostgreSQL, and other databases through a provider plugin API.

Commonly Used Types:

- Microsoft.EntityFrameworkCore.DbContext
- Microsoft.EntityFrameworkCore.DbSet

Version: 5.0.0

Author(s): Microsoft

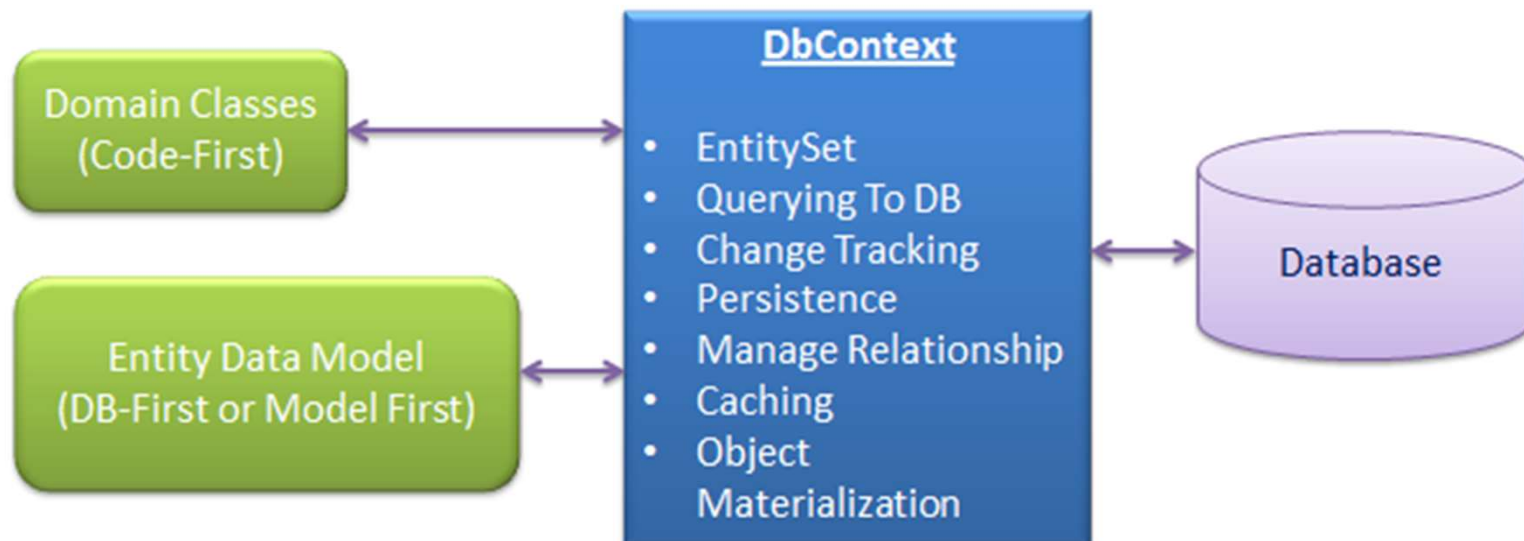
License: Apache 2.0

Demo

Predisposizione ambiente



II DbContext 1/2



II DbContext

```
public class Context : DbContext
{
    public DbSet<Student> Students { get; set; }

    public Context() : base() { }

    public Context() : base("MyContext") { }
}
```

II DbContext

```
public class Context : DbContext
{
    public Context(
        DbContextOptions<TicketingContext> options
    ) : base(options) { }
}

{
    // ...
    "ConnectionStrings": {
        "TicketingDb": "Server=tcp:democrito.database.windows.net,1433;Initial Catalog=Ticketing;
            Persist Security Info=False;User ID=sa;Password=xxxxxxx;MultipleActiveResultSets=False;
            Encrypt=True;TrustServerCertificate=False;Connection Timeout=30;"
    }
}
```

Il DbSet

E' una classe che rappresenta le entity

Serve per fare operazioni CRUD

E' definito come **DbSet<TEntity>**

I metodi più utilizzati sono:

- **Add, Remove, Find, SqlQuery**

```
public DbSet<Student> Students { get; set; }
```

Type Discovery

Nel *DbContext*:

```
public DbSet<Student> Students { get; set; }
```

Definizione della classe *Student*:

```
public class Student
{
    public int StudentID { get; set; }
    public string StudentName { get; set; }
    public DateTime DateOfBirth { get; set; }

    public Teacher Teacher { get; set; }
}
```

Primary Key

```
public class Student
{
    public int StudentID { get; set; }

    public string StudentName { get; set; }
    public DateTime DateOfBirth { get; set; }
    public Teacher Teacher { get; set; }
}
```

Convenzione sul nome della chiave primaria:

- *Id*
- *<NomeClasse>ID*

```
public class Student
{
    public int MyPrimaryKey { get; set; }

    public string StudentName { get; set; }
    public DateTime DateOfBirth { get; set; }
    public Teacher Teacher { get; set; }
}
```

Non usa la convenzione di code-first
Genera una *ModelValidationException*
se non gestita con le *DataAnnotations*

Foreign Key

```
public class Student
{
    public int StudentID { get; set; }

    public string StudentName { get; set; }
    public DateTime DateOfBirth { get; set; }

    public int CourseId { get; set; }
    public Course Course { get; set; }
}
```

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

    public string CourseName { get; set; }
    public Teacher Teacher { get; set; }
    public ICollection<Student> Students { get; set; }
}
```

La *ForeignKey* viene generata automaticamente da code-first ogni volta che viene individuata una navigation property

Sempre bene rispettare le stesse convenzioni della *PrimaryKey*

Navigation Properties

Cos'è la proprietà Teacher?

```
public class Student
{
    // ...
    public Teacher Teacher { get; set; }
}
```

È una **Navigation Property**.

È la rappresentazione in EF di una Relazione tra due entità.

DataAnnotations e Fluent API

Le DataAnnotations sono attributi che servono a specificare il comportamento per fare l'override delle convenzioni di code-first

Possono influenzare le singole proprietà

- Namespace *System.ComponentModel.DataAnnotations*
- *Key, Required, MaxLength...*

Possono influenzare lo schema del database

- Namespace *System.ComponentModel.DataAnnotations.Schema*
- *Table, Column, NotMapped...*

Le DataAnnotations sono limitate. Per il set completo bisogna andare di Fluent API

DataAnnotations: Key

Override della convenzione sulla *PrimaryKey*

Viene applicato alle proprietà di una classe

```
[Key]  
public int MyPrimaryKey { get; set; }
```

DataAnnotations: TimeStamp

Si può applicare solo ad un array di Byte

Viene aggiunto in automatico da Entity Framework

Serve per un controllo sulla concorrenza

```
[TimeStamp]  
public Byte[] RowVersion { get; set; }
```

DataAnnotations: Required

Indica al database che quella colonna non può essere NULL
In ASP.NET MVC viene usato anche per la validazione

```
[Required]  
public string Name { get; set; }
```

DataAnnotations: MaxLenght, MinLenght

Possono essere applicati a stringhe o array

Possono essere usati in coppia

EntityValidationError se non rispettati durante una update

```
[MaxLenght(50), MinLenght(8)]  
public string Name { get; set; }
```

DataAnnotations: Table

Rappresenta l'override del nome della tabella

Può essere solo applicato ad una classe e non alle proprietà

Si può anche inserire uno schema differente

```
[Table("Studente", Schema = "MySchema")]  
public class Student { /* ... */ }
```


DataAnnotations: Column

Rappresenta l'override del nome della proprietà

Può essere applicato solo ad una proprietà

Si può usare in combinata con *Order* e *TypeName*

```
[Column("Nome", Order = 5, TypeName = "varchar")]  
public string Name { get; set; }
```

DataAnnotations: ForeignKey

Rappresenta l'override della convenzione sulla chiave esterna
Viene applicato solo alle proprietà di una classe

```
public class Student
{
    public int StudentId { get; set; }
    public int CourseId { get; set; }

    [ForeignKey("CourseId")]
    public Course Course { get; set; }
}

public class Course
{
    public int CourseId { get; set; }
    public ICollection<Student> Students { get; set; }
}
```

DataAnnotations: NotMapped

Ignora il mapping per proprietà che hanno getter e setter impostati
Viene usato per non creare colonne nel database

```
[NotMapped]  
public string Name { get; set; }
```

DataAnnotations: DatabaseGenerated

Utile per chiavi primarie auto incrementanti

```
[DatabaseGenerated(DatabaseGeneratedOption.Identity)]  
public int Id { get; set; }
```

Fluent API

Sono una alternativa completa alle DataAnnotations
Si definiscono dentro l'override di *OnModelCreating*

Tre tipologie di mapping supportate:

- **Model**: Schema e convenzioni
- **Entity**: Ereditarietà
- **Property**: chiavi primarie/esterne, colonne e altri attributi

Model e Entity Mapping

Configurazione dello schema per tutto il database

Configurazione dello schema per singola tabella

```
protected override void OnModelCreating(ModelBuilder modelBuilder)
{
    modelBuilder.HasDefaultSchema("Admin");

    //Map entity to table
    modelBuilder.Entity<Student>().ToTable("StudentInfo");
    modelBuilder.Entity<Student>().ToTable("StandardInfo", "anotherSchema");
}
```

Property Mapping

Configurazione della chiave primaria

```
modelBuilder.Entity<Student>().HasKey<int>(s => s.StudentId);
```

Configurazione di altre proprietà

```
modelBuilder.Entity<Student>().Property(p => p.Age)
    .HasColumnName("Eta")
    .HasColumnOrder(3)
    .HasColumnType("datetime2")
    .IsRequired();
```

Fluent API Configurations 1/2

Tutte le configurazioni sono fatte via Fluent API

- Problema: troppo codice dentro *OnModelCreating*, diventa ingestibile!
- Soluzione: organizziamo le configurazioni

```
protected override void OnModelCreating(ModelBuilder modelBuilder)
{
    modelBuilder.ApplyConfigurations<Student>(new StudentConfiguration());
    modelBuilder.ApplyConfigurations<Course>(new CourseConfiguration());
}
```


Fluent API Configurations 2/2

```
public class StudentConfiguration : IEntityTypeConfiguration<Student>
{
    public void Configure(EntityTypeBuilder<Ticket> builder)
    {
        builder.ToTable("StudentInfo");

        builder.HasKey<int>(s => s.StudentId);

        builder.Property(p => p.Age)
            .HasColumnName("Eta")
            .HasColumnOrder(3)
            .HasColumnType("datetime2")
            .IsRequired();
    }
}
```

Relazioni uno-a-uno 1/2

Una relazione uno-a-uno è, per definizione, una relazione per cui la chiave primaria di una tabella diventa chiave primaria e chiave esterna dell'altra

```
public class Student
{
    [Key]
    public int StudentId { get; set; }
    public string StudentName { get; set; }

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

```
public class Address
{
    public string Address { get; set; }
    public string City { get; set; }

    [Key, ForeignKey("Student")]
    public int StudentId { get; set; }
    public virtual Student Student { get; set; }
}
```

Relazioni uno-a-uno 2/2

Si può fare anche da Fluent API

```
protected override void OnModelCreating(ModelBuilder modelBuilder) {  
    modelBuilder.Entity<StudentAddress>().HasKey(e => e.StudentId);  
  
    modelBuilder.Entity<Student>()  
        .HasOptional(s => s.StudentAddress)  
        .WithRequired(ad => ad.Student);  
}
```

Relazioni uno-a-molti 1/2

Scenario: un insegnante può tenere più di un corso

```
public class Teacher
{
    public Teacher()
    {
        Courses = new List<Course>();
    }

    [Key]
    public int TeacherId { get; set; }
    public string FullName { get; set; }
    public virtual ICollection<Course> Courses { get; set; }
}
```

```
public class Course
{
    [Key]
    public int CourseId { get; set; }
    public string CourseName { get; set; }
    public int TeacherId { get; set; }

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

Relazione uno-a-molti 2/2

Si può fare anche da Fluent API

```
protected override void OnModelCreating(ModelBuilder modelBuilder)
{
    modelBuilder.Entity<Course>()
        .HasRequired<Teacher>(s => s.Teacher)
        .WithMany(s => s.Courses)
        .HasForeignKey(s => s.TeacherId);

    modelBuilder.Entity<Teacher>()
        .HasMany(s => s.Courses)
        .WithRequired(x => x.Teacher)
        .HasForeignKey(x => x.TeacherId);
}
```

Relazioni multi-a-molti 1/4

Scenario: uno studente è iscritto a più corsi e ogni corso può avere più studenti

```
public class Course
{
    [Key]
    public int CourseId { get; set; }
    public string CourseName { get; set; }
    public virtual ICollection<Student> Students { get; set; }
}
```

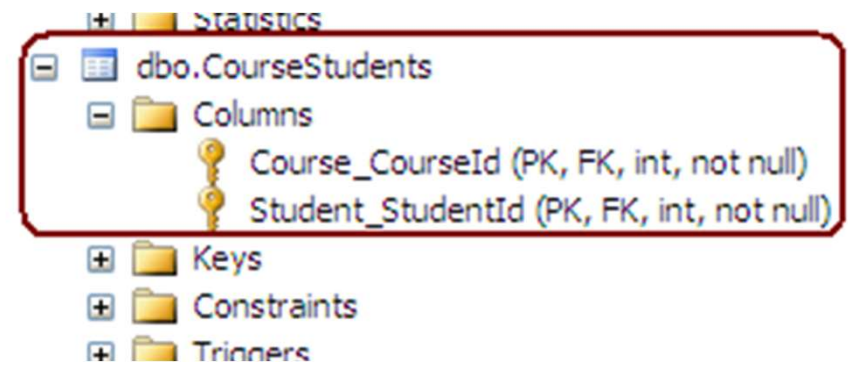
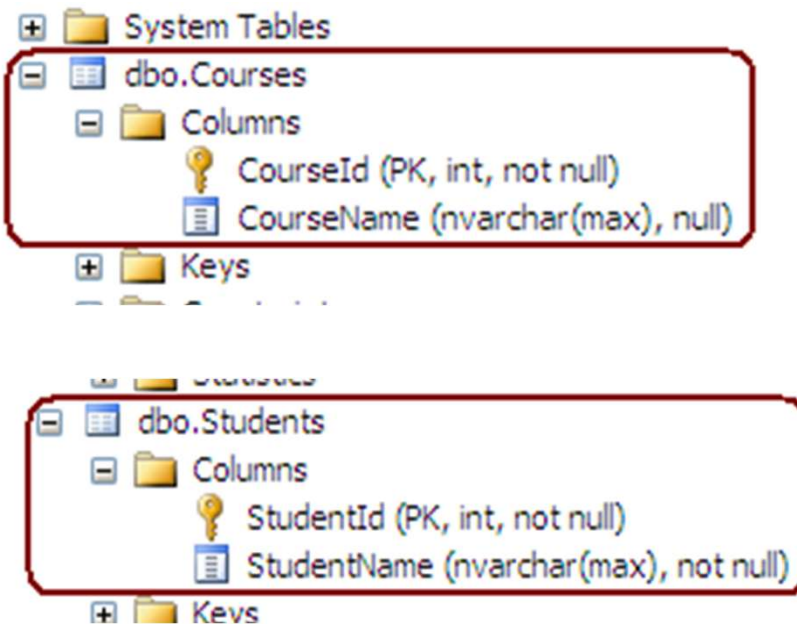
Relazioni multi-a-molti 2/4

Scenario: uno studente è iscritto a più corsi e ogni corso può avere più studenti

```
public class Student
{
    public Student()
    {
        Courses = new List<Course>();
    }

    [Key]
    public int StudentId { get; set; }
    public string StudentName { get; set; }
    public virtual ICollection<Course> Courses { get; set; }
}
```

Relazioni multi-a-molti 3/4



Relazioni multi-a-molti 4/4

Si può fare anche da Fluent API per specificare la tabella di join

```
protected override void OnModelCreating(ModelBuilder modelBuilder)
{
    modelBuilder.Entity<Student>()
        .HasMany<Course>(s => s.Courses)
        .WithMany(c => c.Students)
        .Map(cs =>
        {
            cs.MapLeftKey("StudentRefId");
            cs.MapRightKey("CourseRefId");
            cs.ToTable("StudentCourse");
        });
}
```

Demo

Modelliamo l'esercizio dell'agenzia viaggi
per riportarlo su Entity Framework



Aggiungere dati

Utile sapere:

- Il pattern *IDisposable*
- *Async/await*

```
using (var ctx = new Context())  
{  
    var person = new Person(1, "Mirko", "De Bonis");  
    ctx.People.Add(person);  
    await ctx.SaveChangesAsync();  
}
```

Fare query sui dati

Utile conoscere:

- Il pattern *IDisposable*
- *Linq*
- I dati che si vogliono ottenere ☺

```
using (var ctx = new Context())  
{  
    ctx.Students.Where(x => x.Name.StartsWith("A"))  
                .OrderBy(x => x.Age)  
                .ToList();  
}
```

Errori comuni 1/2

Giusto

```
var ages = dbContext.People
    .Where(x => x.LastName.StartsWith("A"))
    .OrderBy(x => x.Age)
    .Where(x => x.City == "Bologna")
    .ToList();
```

Sbagliato

```
var ages = dbContext.People
    .Where(x => x.LastName.StartsWith("A"))
    .ToList()
    .OrderBy(x => x.Age)
    .Where(x => x.City == "Bologna")
    .ToList();
```

Errori comuni 2/2

Giusto

```
var person = dbContext.People.Find(1);
```

Quasi giusto 😊

```
var person = dbContext.People  
    .Where(x => x.Id == 1);
```

CUD (Create / Update / Delete)

EF offre diversi modi per aggiungere, aggiornare o eliminare i dati nel database. Un'entità verrà inserita o aggiornata o eliminata in base al valore della sua proprietà **EntityState**.

Esistono due scenari per salvare i dati di un'entità:

- **Connesso**: la stessa istanza di **DbContext** viene utilizzata per il recupero e il salvataggio delle entità
- **Disconnesso**: l'istanza di **DbContext** utilizzata per il recupero e il salvataggio delle entità è diversa

CUD (Create / Update / Delete)

Connected Scenario

Inserimento

```
using(var ctx = new MyContext())
{
    var prd = new Product()
    {
        ID = 1,
        ProductCode = "PR0001"
    };

    ctx.Products.Add(prd);

    ctx.SaveChanges();
}
```


CUD (Create / Update / Delete)

Connected Scenario

Update / Delete

```
using(var ctx = new MyContext())
{
    var prd = ctx.Products.First<Product>();

    // UPDATE
    prd.ProductCode = "PR0002";
    ctx.SaveChanges();

    // DELETE
    ctx.Products.Remove(prd);
    ctx.SaveChanges();
}
```

CUD (Create / Update / Delete)

Disconnected Scenario

Inserimento

```
using(var ctx = new MyContext())
{
    var prd = new Product()
    {
        ID = 1,
        ProductCode = "PR0001"
    };

    ctx.Entry<Product>(prd).State = EntityState.Added;

    ctx.SaveChanges();
}
```

CUD (Create / Update / Delete)

Disconnected Scenario

Update / Delete

```
Product prd;  
using(var ctx = // ...  
{  
    prd = ctx.Pr  
}  
// ...  
prd.FirstName =  
// ...  
using(var ctx = new MyContext())  
{  
    // UPDATE  
    ctx.Entry<Product>(prd).State = EntityState.Modified;  
    ctx.SaveChanges();  
}  
}
```

Problemi di performance?

Si può bypassare uno strato di Entity Framework che si occupa della traduzione della query da Linq to Entities

```
var people = dbContext.Database.SqlQuery<Person>(
    "SELECT * FROM Person WHERE FirstName LIKE 'a%')
.ToList();
```

Caricamento dei Dati Correlati

Entity Framework Core consente di utilizzare le Navigation Property nel modello per caricare entità correlate

Esistono due modelli comuni utilizzati per caricare i dati correlati:

Eager Loading: i dati correlati vengono caricati dal database come parte della query iniziale

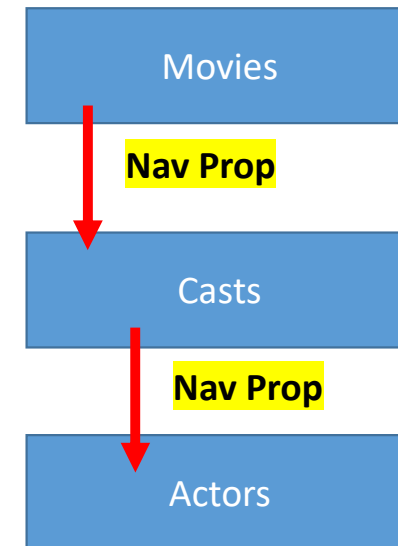
Lazy Loading: i dati correlati vengono caricati in modo trasparente dal database quando si accede alla proprietà di navigazione

Caricamento dei Dati Correlati

Eager Loading

È possibile utilizzare il metodo **Include** per specificare i dati correlati da includere nei risultati della query

```
using var ctx = new MyContext();  
  
var data = ctx.Movies  
    .Include(m => m.Casts)  
    .Include(c => c.actors)  
    .ToList();
```



Caricamento dei Dati Correlati

Lazy Loading – Metodo 1

Il modo più semplice per utilizzare il Lazy loading
installare il pacchetto `Microsoft.EntityFrameworkCore.Proxies`
abilitarlo con una chiamata a `UseLazyLoadingProxies()`

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

Caricamento dei Dati Correlati

Lazy Loading – Metodo 1

EF Core abiliterà quindi il Lazy Loading per qualsiasi Navigation Property che dovrà essere `virtual` e di un tipo che può essere ereditato (classe non sealed)

```
class Actor // non sealed
{
    // ...
    // Nav Prop virtual
    public virtual IEnumerable<Casts> Cast { get; set; }
}
```


Caricamento dei Dati Correlati

Lazy Loading – Metodo 2

```
class Actor
{
    public Actor(ILazyLoader lazyLoader) {
        LazyLoader = lazyLoader;
    }

    private ILazyLoader LazyLoader { get; set; }
    // ...
    private IEnumerable<Casts> _cast;

    public ICollection<Casts> Posts
    {
        get => LazyLoader.Load(this, ref _cast);
        set => _posts = value;
    }
}
```

Il Lazy Loading in EF Core può funzionare anche iniettando il servizio **ILazyLoader** in un'entità e modificando le Navigation Property.

Demo

Implementazione dei metodi CRUD



Migrations

Servono in caso di modifiche al modello da riflettere sul database

Nuova tipologia di inizializzazione del database da EF 4.3

- *MigrateDatabaseToLatestVersion*

Utili in caso di database già esistente

- Non si perdono eventuali stored procedure, trigger...

Due tipi di migrazioni

- *Automatiche*: poco invasive
- *Manuali o code-based*: richiedono un intervento specifico sul database

Migrazioni automatiche 1/4

Per abilitare le migrazioni bisogna avviare un comando dalla Package Manager Console

- *Enable-Migrations -EnableAutomaticMigration:\$true*

Se il comando ha successo, allora verrà creato il file */Migrations/Configuration.cs* che rappresenta la nuova strategia di inizializzazione

```
internal sealed class Configuration : DbMigrationsConfiguration<Context>
{
    public Configuration()
    {
        AutomaticMigrationsEnabled = true;
    }

    protected override void Seed(Context context) { }
}
```

Migrazioni code-based 1/2

Sono utili quando:

- Siamo in produzione
- Lo stato del database è già definito
- Vogliamo più controllo sulle modifiche automatiche

Servono due comandi dalla console:

- *Add-Migrations «Migration name»*
 - Crea una nuova classe con tutte le modifiche rispetto allo stato precedente del db
- *Update-Database*
 - Aggiorna il database con il modello

Si può anche fare rollback di una modifica:

- *Update-Database –TargetMigration:"Migration name"*

Migrazioni code-based 2/2

Viene creato un nuovo file per ogni migrazione

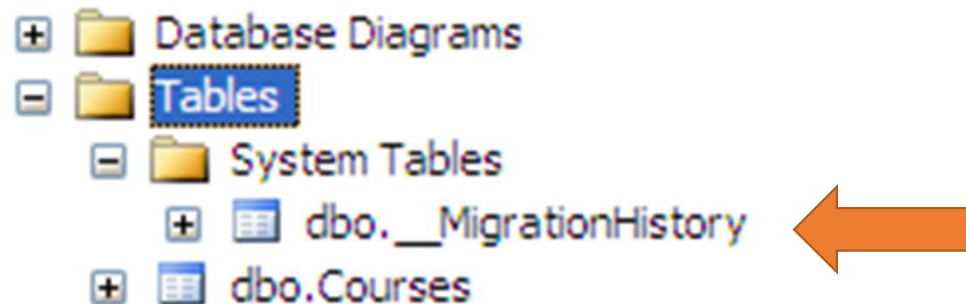
- TimeStamp + Nome migrazione . cs
- Eredita da *DbMigration*
- Contiene due metodi *Up* e *Down* per l'aggiornamento del database

```
public partial class FirstMigration : DbMigration
{
    public override void Up()
    {
        CreateTable("dbo.Students", c => new {
            StudentId = c.Int(nullable: false, identity: true),
            Name = c.String(),
            Age = c.Int(nullable: false)})
            .PrimaryKey(t => t.StudentId);
    }

    public override void Down()
    {
        DropTable("dbo.Students");
    }
}
```

Migrazioni

Se andiamo a vedere il nostro database...



Viene aggiunta una tabella al nostro database per mantenere lo storico delle Migration applicate.

Esercitazione

Realizzare un sistema informativo relativo ad un supermercato.

- Si richiede di tenere traccia delle informazioni relative all'entità **Reparto** caratterizzato da
 - Numero (int) Nome (string)
- A ciascun reparto possono appartenere uno o più **Dipendenti** con i seguenti dati
 - Codice (string), Cognome (string), Nome(string), Data di nascita (DateTime).
- Una caratteristica dei reparti è quella di contenere un certo numero di **Prodotti** di cui si conosce
 - Codice (string), Descrizione (string), Prezzo (decimal).
- Si vuole inoltre tenere traccia delle **Vendite** relative a ciascun prodotto.
- Ciascuna vendita sarà caratterizzata da
 - Numero vendita (int PK), Quantità (int), Data di vendita (DateTime).
- Realizzare il modello ER della base di dati descritta e le tabelle del database con EF.



Gestione della Concorrenza

EF Core implementa il controllo ottimistico della concorrenza

Nella situazione ideale, i cambiamenti effettuati non interferiranno tra loro e quindi potranno avere successo

- Nel peggiore dei casi, due o più processi tenteranno di apportare modifiche in conflitto e solo uno di essi dovrebbe riuscire

Gestione della Concorrenza

Per gestire la concorrenza occorre aggiungere una proprietà **Timestamp** alle entità:

```
class Movie {  
    // DATA ANNOTATION  
    [Timestamp]  
    public Byte[] RowVersion { get; set; }  
    // ...  
}
```

```
// FLUENT API  
modelBuilder.Entity<Movie>()  
    .Property(p => p.RowVersion)  
    .IsRowVersion();
```

Gestione della Concorrenza

Quando due o più processi vogliono modificare un record, EF effettua un check sul timestamp.

Se il timestamp del record nel DbSet è diverso da quello nel db sono in presenza di un conflitto

- occorre gestire l'eccezione **DbUpdateConcurrencyException**

```
try {  
    // ...  
    ctx.SaveChanges();  
} catch (DbUpdateConcurrencyException ex) {  
    // ...  
}
```

Gestione della Concorrenza

La risoluzione del conflitto di concorrenza implica l'unione delle modifiche in sospeso dal DbContext corrente con i valori nel database

Sono disponibili tre set di valori per aiutare a risolvere un conflitto di concorrenza:

- Current Values: sono i valori che l'applicazione stava tentando di scrivere nel database
- Original Values: sono i valori originariamente recuperati dal database, prima che venissero apportate le modifiche
- Database Values: sono i valori attualmente memorizzati nel database

```

catch (DbUpdateConcurrencyException ex)
{
    foreach (var entry in ex.Entries)
    {
        if (entry.Entity is Person)
        {
            var proposedValues = entry.CurrentValues;
            var databaseValues = await entry.GetDatabaseValuesAsync();

            foreach (var property in proposedValues.Properties)
            {
                var proposedValue = proposedValues[property];
                var databaseValue = databaseValues[property];

                // TODO: decide which value should be written to database
                // proposedValues[property] = <value to be saved>;

                // Refresh original values to bypass next concurrency check
                entry.OriginalValues.SetValues(databaseValues);
            }
        }
        else
        {
            throw new NotSupportedException(
                "Don't know how to handle concurrency conflicts for "
                + entry.Metadata.Name);
        }
    }
}

catch (DbUpdateConcurrencyException ex)
{
    foreach (var entry in ex.Entries)
    {
        if (entry.Entity is Person)
        {
            var proposedValues = entry.CurrentValues;
            var databaseValues = await entry.GetDatabaseValuesAsync();

            foreach (var property in proposedValues.Properties)
            {
                var proposedValue = proposedValues[property];
                var databaseValue = databaseValues[property];

                // TODO: decide which value should be written to database
                // proposedValues[property] = <value to be saved>;

                // Refresh original values to bypass next concurrency check
                entry.OriginalValues.SetValues(databaseValues);
            }
        }
        else
        {
            throw new NotSupportedException(
                "Don't know how to handle concurrency conflicts for "
                + entry.Metadata.Name);
        }
    }
}

```

L'eccezione contiene le entità su cui abbiamo avuto il conflitto

Per ogni entità posso accedere a Current Values e Database Values

Con queste informazioni posso decidere come gestire il conflitto.

... e poi tento di risolvere le modifiche

Domande?



Ricordate il feedback!



© 2021 iCubed Srl



La diffusione di questo materiale per scopi differenti da quelli per cui se ne è venuti in possesso è vietata.

iCubed s.r.l.

Piazza Duca D'Aosta, 12 20124 MILANO

Phone: +39 02 57501057

P.IVA 07284390965

