無料電子ブック

学習

Entity Framework

Free unaffiliated eBook created from
Stack Overflow contributors.

#entity-framework

You can share this PDF with anyone you feel could benefit from it, downloaded the latest version from: entity-framework

It is an unofficial and free Entity Framework ebook created for educational purposes. All the content is extracted from Stack Overflow Documentation, which is written by many hardworking individuals at Stack Overflow. It is neither affiliated with Stack Overflow nor official Entity Framework.

Use the content presented in this book at your own risk; it is not guaranteed to be correct nor accurate, please send your feedback and corrections to info@zzzprojects.com

1: Entity Frameworkをいめる

Entity FrameworkEFは、.NETがドメインのオブジェクトをしてリレーショナルデータをできるようにするオブジェクトリレーショナルマッパーORMです。これにより、がくのあるデータアクセスコードのほとんどがになります。

Entity Frameworkをすると、EF Designerでコードをしたり、ボックスやをしてモデルをできます。これらのアプローチのをして、のデータベースをターゲットにするか、しいデータベースをすることができます。

Entity Frameworkは、Microsoftが.NET FrameworkおよびマイクロソフトがするデータアクセステクノロジにするなORMです。

バージョン

バージョン	
1.0	2008811
4.0	2010-04-12
4.1	2011-04-12
4.1アップデート1	2011-07-25
4.3.1	2012-02-29
5.0	2012-08-11
6.0	20131017
6.1	2014-03-17
コア1.0	2016627

リリースノート https : //msdn.microsoft.com/en-ca/data/jj574253.aspx

Examples

Cのエンティティフレームワークをするコードファースト

コードではまず、GUIデザイナや.edmxファイルをせずにエンティティクラスをできます。それは、あなたがにあなたのモデルをすることができますので、 のコードとされたとエンティティフレームワークがにあなたのためのマッピングにってデータベースをします。または、のデータベー

スでこのアプローチをすることもできます。のデータベースでは、 にコードとばれます 。たとえば、テーブルにのリストをするは、のようにします。

```
public class Planet
{
    public string Name { get; set; }
    public decimal AverageDistanceFromSun { get; set; }
}
```

エンティティクラスとデータベースののブリッジであるコンテキストをします。 1つまたはの`DbSet<>`プロパティをします。

```
using System.Data.Entity;

public class PlanetContext : DbContext
{
    public DbSet<Planet> Planets { get; set; }
}
```

これをうには、をします。

```
using(var context = new PlanetContext())
{
    var jupiter = new Planet
    {
        Name = "Jupiter",
        AverageDistanceFromSun = 778.5
    };

    context.Planets.Add(jupiter);
    context.SaveChanges();
}
```

このでは、する`Planet`して`Name`のをつプロパティ`"Jupiter"`と`AverageDistanceFromSun`をつプロパティ`778.5`

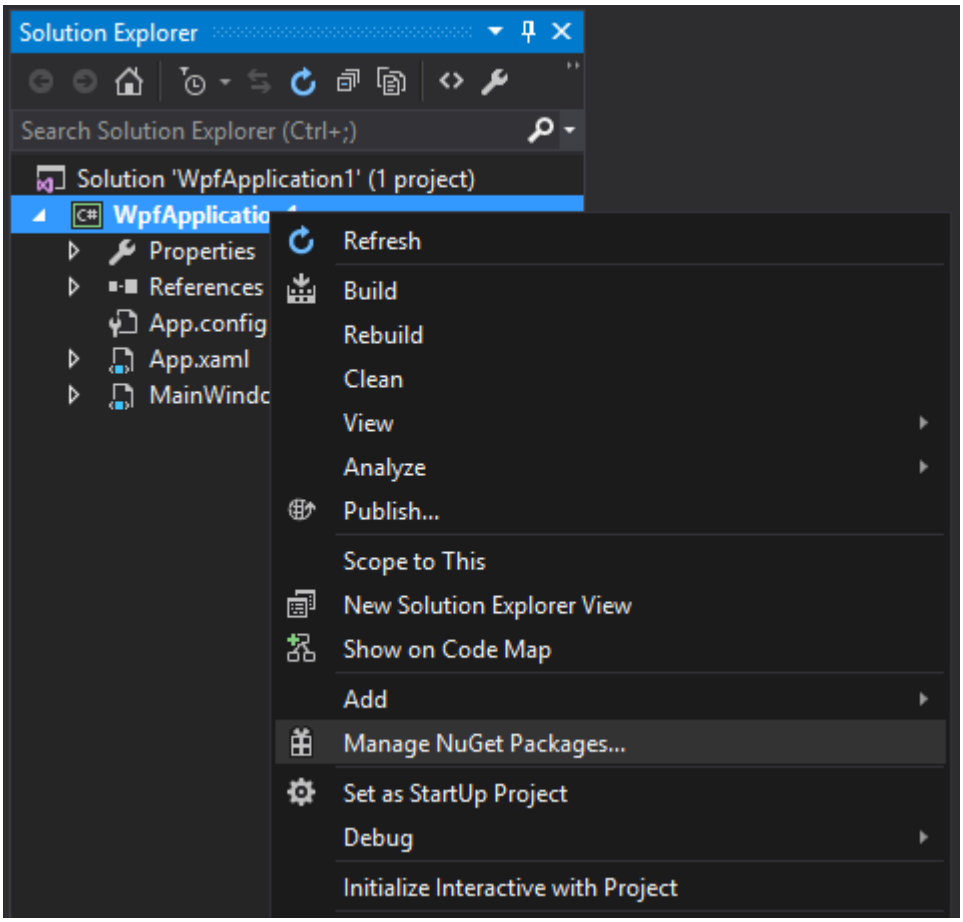
に、 `DbSet`の`Add()`メソッドをしてこの`Planet`をコンテキストに`Add()`し、 `SaveChanges()`メソッドをしてデータベースへのをコミットできます。

または、データベースからをりすことができます。

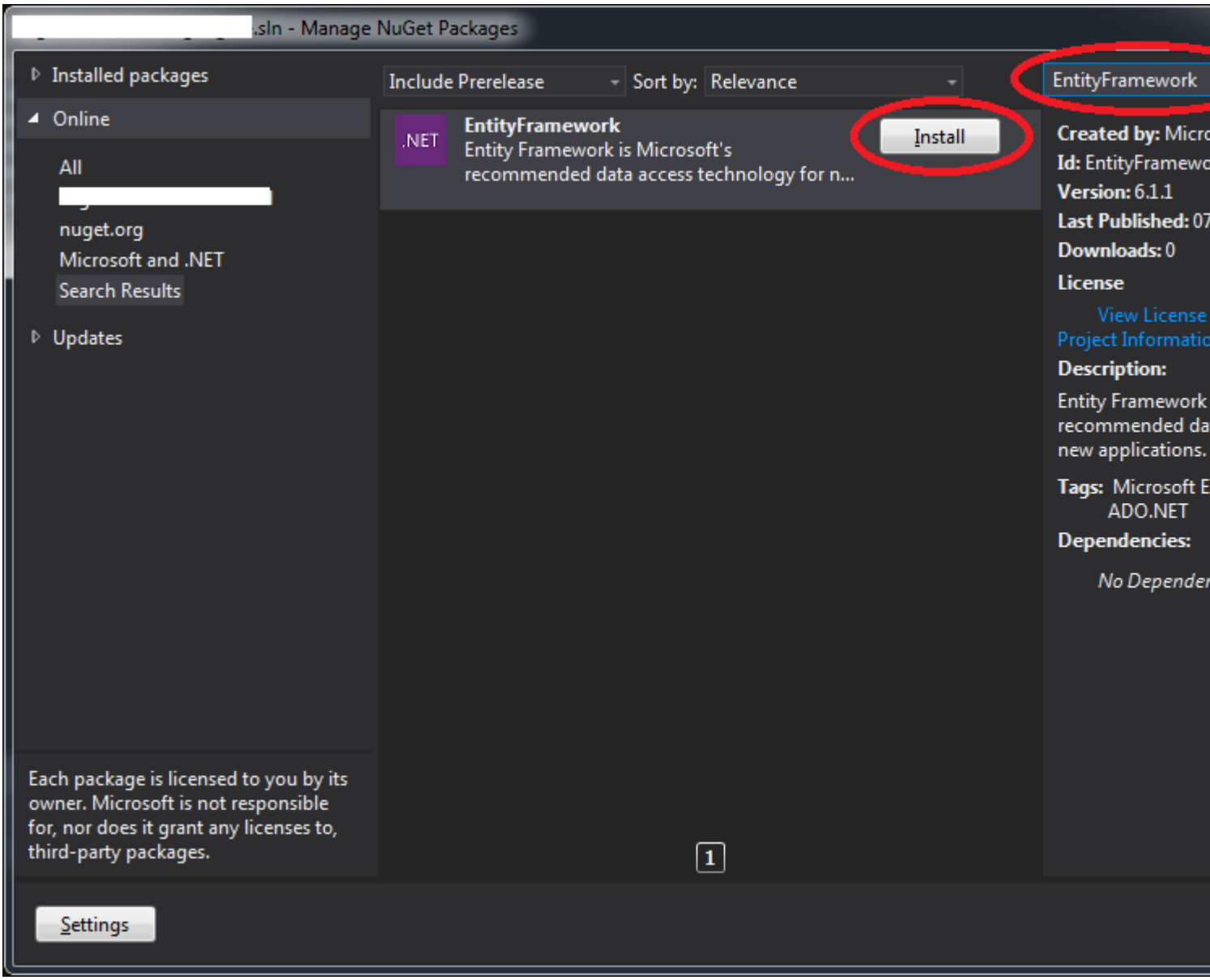
```
using(var context = new PlanetContext())
{
    var jupiter = context.Planets.Single(p => p.Name == "Jupiter");
    Console.WriteLine($"Jupiter is {jupiter.AverageDistanceFromSun} million km from the
sun.");
}
```

Entity Framework NuGetパッケージのインストール

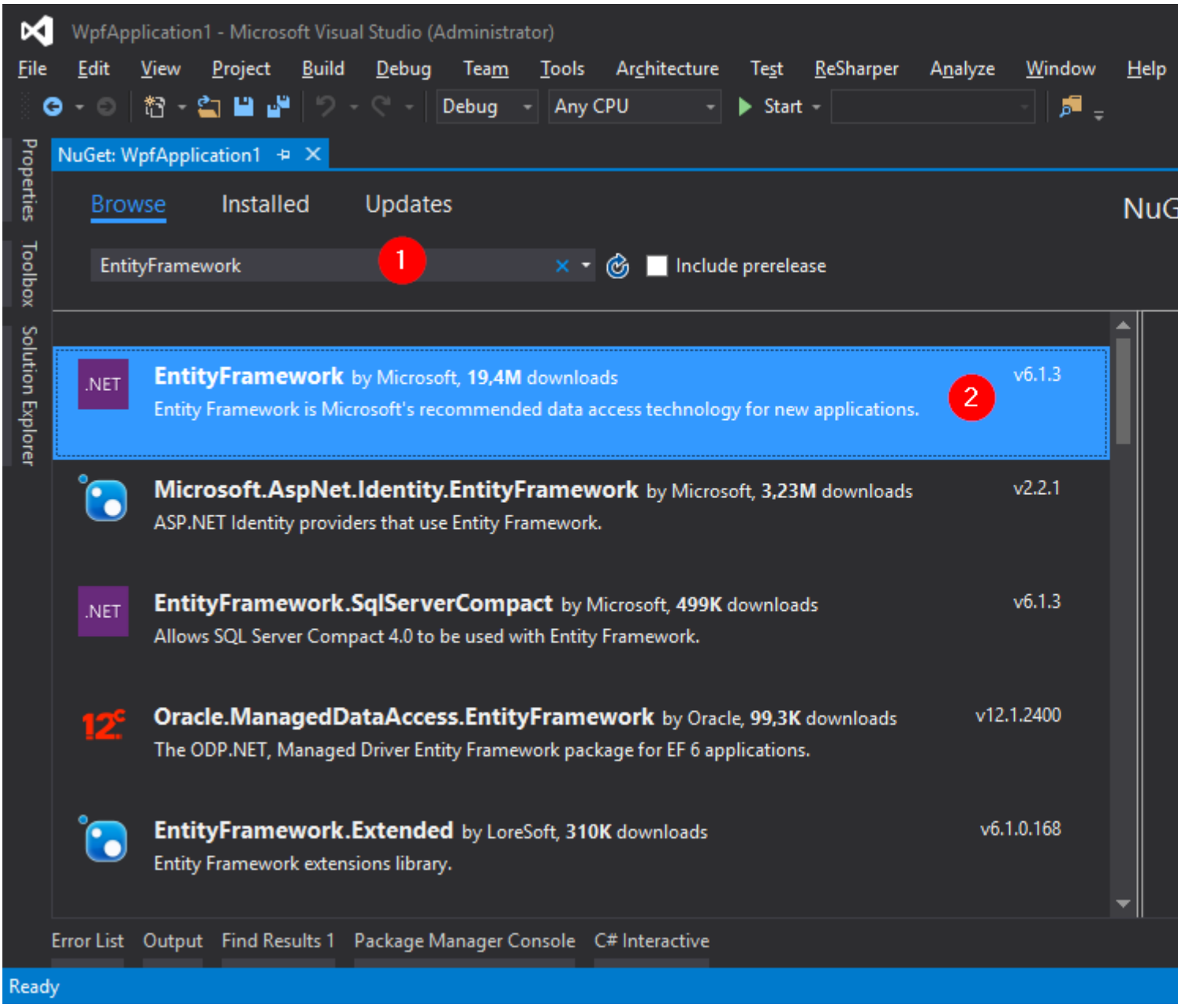
Visual Studioでソリューションエクスプローラウィンドウをき、プロジェクトを`クリック`して、メニューから*[Manage NuGet Packages]*をします。



`EntityFramework`のボックスに`EntityFramework`というタイプのウィンドウがきます。



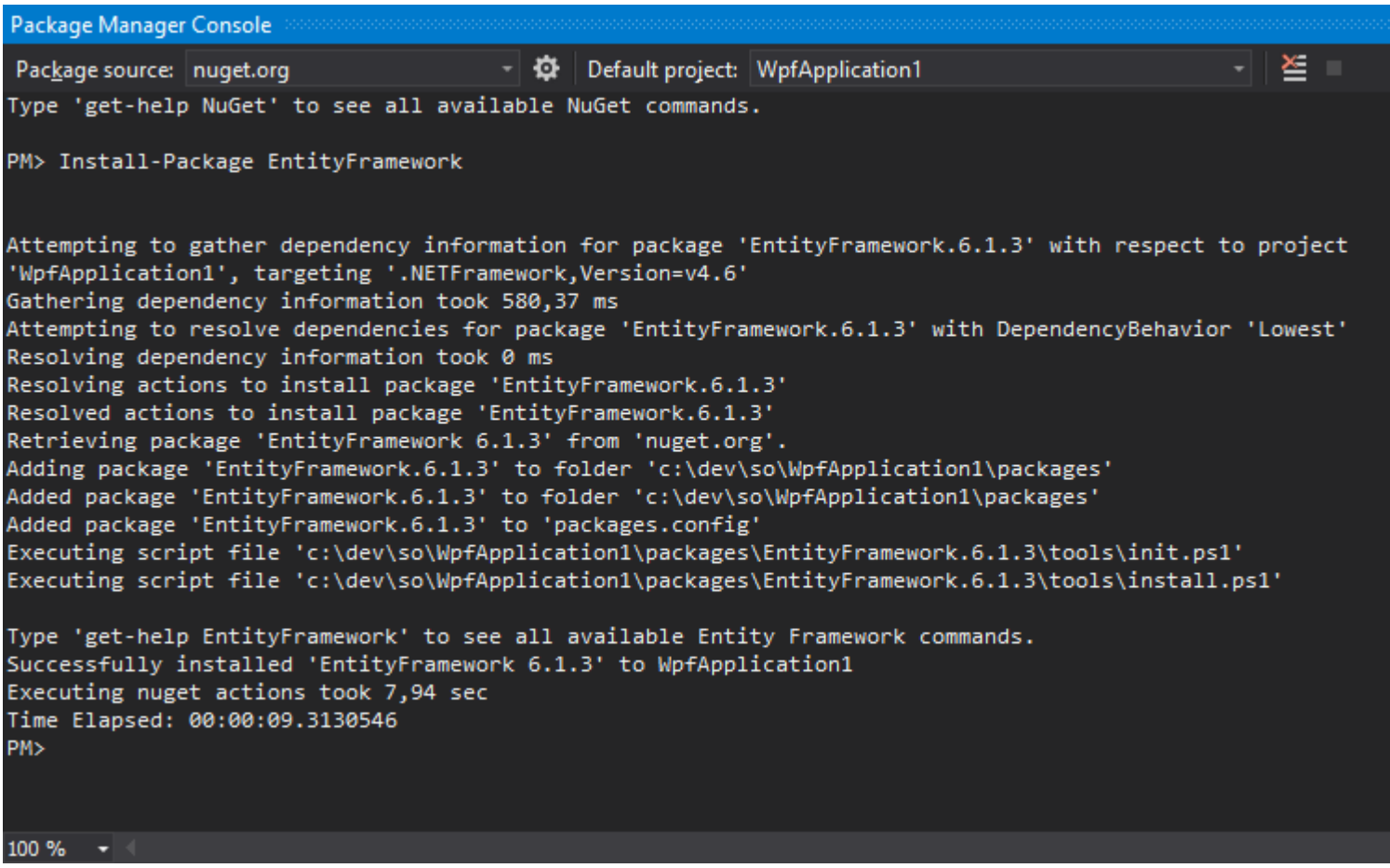
または、Visual Studio 2015をしているは、のようにされます。



に、「インストール」をクリックします。

また、パッケージマネージャコンソールをしてエンティティフレームワークをインストールすることもできます。 ツールメニュー - > *NuGet Package Manager - > Package Manager Console*をしてにくには、のようにします。

```
Install-Package EntityFramework
```



これにより、Entity Frameworkがインストールされ、プロジェクトのアセンブリへのがにされます。

Entity Frameworkとはですか

データアクセスのためのADO.Netコードのとは、でなです。マイクロソフトは、アプリケーションのデータベースのアクティビティをするために、「Entity Framework」とばれる**O / RM**フレームワークをしています。

エンティティフレームワークは、オブジェクト/リレーショナルマッピングO / RMフレームワークです。にデータベースのデータにアクセスしてするためのされたメカニズムをするADO.NETのです。

O / RMとはですか

ORMは、ドメイン·オブジェクトからMS SQL Serverのようなリレーショナル·データベースに、くのプログラミングをすることなくされたでデータをするためのツールです。 O / RMには、の3つのながあります。

1. ドメインクラスオブジェクト
2. リレーショナルデータベースオブジェクト
3. リレーショナルデータベースオブジェクト **ex**テーブル、ビュー、およびストアドプロシージャへのドメインオブジェクトのマップにするのマッピング

ORMをすると、データベースとドメインクラスをすることができます。これにより、アプリケーションのとがします。また、なCRUDCreate、Read、UpdateDeleteをして、がでくがないようにします。

オンラインでEntity Frameworkをいめるをむ https://riptutorial.com/ja/entity-framework/topic/815/entity-frameworkをいめる

2: EFにおける

Examples

AsNoTrackingの

い

```
var location =  dbContext.Location
                    .Where(l => l.Location.ID == location_ID)
                    .SingleOrDefault();

return location;
```

のコードは、エンティティをまたはせずにすだけなので、コストのはけることができます。

いえ

```
var location =  dbContext.Location.AsNoTracking()
                    .Where(l => l.Location.ID == location_ID)
                    .SingleOrDefault();

return location;
```

`AsNoTracking()`をするとき、エンティティがコンテキストによってされないことをにEntity Frameworkにえています。これは、データストアからのデータをするににです。ただし、されていないエンティティにをえたいは、 `SaveChanges`びすにそれらをすることをれないでください。

なデータのみのみみ

コードでよくられるの1つは、すべてのデータをロードすることです。これにより、サーバーのがにします。

たとえば、「」というモデルがあり、そのに10のフィールドがありますが、すべてのフィールドがになわけではありません。そのモデルの 'LocationName'パラメータだけがだとしましょう。

い

```
var location =  dbContext.Location.AsNoTracking()
                        .Where(l => l.Location.ID == location_ID)
                        .SingleOrDefault();

return location.Name;
```

いえ

```
var location =  dbContext.Location
```

```
                            .Where(l => l.Location.ID == location_ID)
                            .Select(l => l.LocationName);
                            .SingleOrDefault();

return location;
```

"い"のコードは "LocationName"だけをし、それはしません。

このではされていないため、 `AsNoTracking()` はありません。とにかくするものはもありません。

でさらにフィールドをする

```
var location = dbContext.Location
                    .Where(l => l.Location.ID == location_ID)
                    .Select(l => new { Name = l.LocationName, Area = l.LocationArea })
                    .SingleOrDefault();

return location.Name + " has an area of " + location.Area;
```

のとじように、フィールド 'LocationName'と 'LocationArea'のみがデータベースからされます。タイプには、なのをできます。

メモリではなく、なはデータベースでクエリをします。

テキサスにいくつのがあるのかをえたいとします。

```
var counties = dbContext.States.Single(s => s.Code == "tx").Counties.Count();
```

クエリーはしいが、である。 `States.Single(…)` は、データベースからをロードします。に、 `Counties` は、すべてのフィールドをつ254すべてを2のクエリでロードします。ロードされた `Counties` コレクションのメモリで `.Count()` がされます 。
たちはのないくのデータをロードしましたが、もっとうまくいくことができます

```
var counties = dbContext.Counties.Count(c => c.State.Code == "tx");
```

ここでは、1つのクエリのみをします.SQLでは、カウントとにされます。、フィールド、およびオブジェクトのをしました。

コレクションの `IQueryable<T>IEnumerable<T>` をべることで、クエリのをにることができます。

のクエリをでする

クエリをする、のクエリをにできますが、じコンテキストではできません。 1つのクエリのが10の、いのは20になりますが、いのは10になります。

い

```
IEnumerable<TResult1> result1;
IEnumerable<TResult2> result2;

using(var context = new Context())
{
    result1 = await context.Set<TResult1>().ToListAsync().ConfigureAwait(false);
    result2 = await context.Set<TResult1>().ToListAsync().ConfigureAwait(false);
}
```

いえ

```
public async Task<IEnumerable<TResult>> GetResult<TResult>()
{
    using(var context = new Context())
    {
        return await context.Set<TResult1>().ToListAsync().ConfigureAwait(false);
    }
}



IEnumerable<TResult1> result1;
IEnumerable<TResult2> result2;

var result1Task = GetResult<TResult1>();
var result2Task = GetResult<TResult2>();

await Task.WhenAll(result1Task, result2Task).ConfigureAwait(false);

var result1 = result1Task.Result;
var result2 = result2Task.Result;
```

とプロキシをにする

データをしたいだけでもしないは、のとプロキシのをオフにすることができます。これにより、パフォーマンスがし、ロードもされます。

い

```
using(var context = new Context())
{
    return await context.Set<MyEntity>().ToListAsync().ConfigureAwait(false);
}
```

いえ

```
using(var context = new Context())
{
    context.Configuration.AutoDetectChangesEnabled = false;
    context.Configuration.ProxyCreationEnabled = false;

    return await context.Set<MyEntity>().ToListAsync().ConfigureAwait(false);
}
```

に、これらをソリューションにしたいは、コンテキストのコンストラクタからこれらをオフにすることがです。

```
public class MyContext : DbContext
{
    public MyContext()
        : base("MyContext")
    {
        Configuration.AutoDetectChangesEnabled = false;
        Configuration.ProxyCreationEnabled = false;
    }

    //snip
}
```

スタブエンティティの

`Product`と`Category`がのにあるとします。

```
public class Product
{
    public Product()
    {
        Categories = new HashSet<Category>();
    }
    public int ProductId { get; set; }
    public string ProductName { get; set; }
    public virtual ICollection<Category> Categories { get; private set; }
}

public class Category
{
    public Category()
    {
        Products = new HashSet<Product>();
    }
    public int CategoryId { get; set; }
    public string CategoryName { get; set; }
    public virtual ICollection<Product> Products { get; set; }
}
```

`Category`を`Product`にするは、 `Product`をみんでその`Categories`にカテゴリをするがあります。

い

```
var product = db.Products.Find(1);
var category = db.Categories.Find(2);
product.Categories.Add(category);
db.SaveChanges();
```

`db`は`DbContext`サブクラスです。

これにより、 `Product`と`Category`のジャンクションに1つのレコードがされます。ただし、このには2つの`Id`しかまれていません。 1つのさなレコードをするために、なエンティティを2つロード

するのはです。

よりなは、のデータは`Id`のみをむスタブエンティティ 、つまりエンティティオブジェクトをメモリにすることです。これはのようになります。

いえ

```
// Create two stub entities
var product = new Product { ProductId = 1 };
var category = new Category { CategoryId = 2 };

// Attach the stub entities to the context
db.Entry(product).State = System.Data.Entity.EntityState.Unchanged;
db.Entry(category).State = System.Data.Entity.EntityState.Unchanged;

product.Categories.Add(category);
db.SaveChanges();
```

なはじですが、データベースへの2のラウンドトリップはされます。

をぐ

がすでにするかどうかをしたいは、なクエリでです。えば

```
var exists = db.Categories.Any(c => c.Id == 1 && c.Products.Any(p => p.Id == 14));
```

このも、エンティティがメモリにロードされません。これはテーブルをし、をすだけです。

オンラインでEFにおけるをむ https://riptutorial.com/ja/entity-framework/topic/2714/efにおける

3: Entity Frameworkコードファースト

Examples

のデータベースにする

Entity Frameworkでもなタスクをするには、ローカルインスタンスのMSSQLののデータベース`ExampleDatabase`にするには、2つのクラスのみをするがあります。

まず、エンティティクラスがデータベーステーブル`dbo.People`マップされます。

```
class Person
{
    public int PersonId { get; set; }
    public string FirstName { get; set; }
}
```

クラスは、Entity Frameworkのをし、キー`PersonId`とvarcharmaxプロパティ`FirstName`をつとされるテーブル`dbo.People`マップします。

に、 `System.Data.Entity.DbContext`からし、にエンティティオブジェクトをし、データベースからし、をし、データベースにするコンテキストクラスです。

```
class Context : DbContext
{
    public Context(string connectionString) : base(connectionString)
    {
        Database.SetInitializer<Context>(null);
    }

    public DbSet<Person> People { get; set; }
}
```

コンテキストのコンストラクタでは、データベースをnullにするがあることにしてください。Entity Frameworkでデータベースをするはなく、にアクセスしたいだけです。

は、そのテーブルのデータをすることができます。えば、データベースののの`FirstName`をのようなコンソールアプリケーションからします。

```
class Program
{
    static void Main(string[] args)
    {
        using (var ctx = new Context("DbConnectionString"))
        {
            var firstPerson = ctx.People.FirstOrDefault();
            if (firstPerson != null) {
                firstPerson.FirstName = "John";
                ctx.SaveChanges();
```

```
                }
            }
        }
    }
```

のコードでは、 "DbConnectionString"をしてContextのインスタンスをしました。これは `app.config` ファイルでのようにするがあります

```
<connectionStrings>
  <add  name="DbConnectionString"
  connectionString="Data Source=.;Initial Catalog=ExampleDatabase;Integrated Security=True"
  providerName="System.Data.SqlClient"/>
</connectionStrings>
```

オンラインでEntity Frameworkコードファーストをむ https://riptutorial.com/ja/entity-framework/topic/5337/entity-frameworkコードファースト

4: Entity Frameworkとのマッピングコード11およびき

このトピックでは、Entity Frameworkコードファーストをして、1のとのをマッピングするについてします。

Examples

1のマッピング

つまり、のような2つのなるエンティティがあるとしましょう。

```
public class Person
{
   public int PersonId { get; set; }
   public string Name { get; set; }
}

public class Car
{
   public int CarId { get; set; }
   public string LicensePlate { get; set; }
}

public class MyDemoContext : DbContext
{
   public DbSet<Person> People { get; set; }
   public DbSet<Car> Cars { get; set; }
}
```

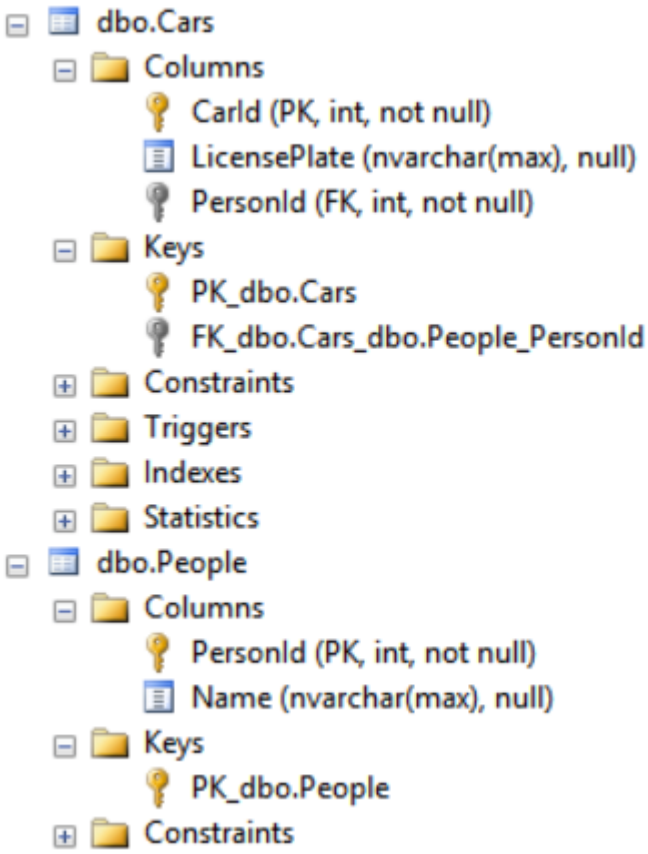
そして、それらのに1のをしたい、つまり、1が0、1のをつことができ、1のが1のににします。すべてのはであるため、がをっていると、はそのにします。

これをうには、モデルクラスをするだけです

```
public class Person
{
   public int PersonId { get; set; }
   public string Name { get; set; }
   public virtual ICollection<Car> Cars { get; set; } // don't forget to initialize (use
HashSet)
}

public class Car
{
   public int CarId { get; set; }
   public string LicensePlate { get; set; }
```

```
   public int PersonId { get; set; }
   public virtual Person Person { get; set; }
}
```

それはそれです:)あなたはすでにあなたのをセットアップしています。データベースでは、これはもちろんキーでされます。



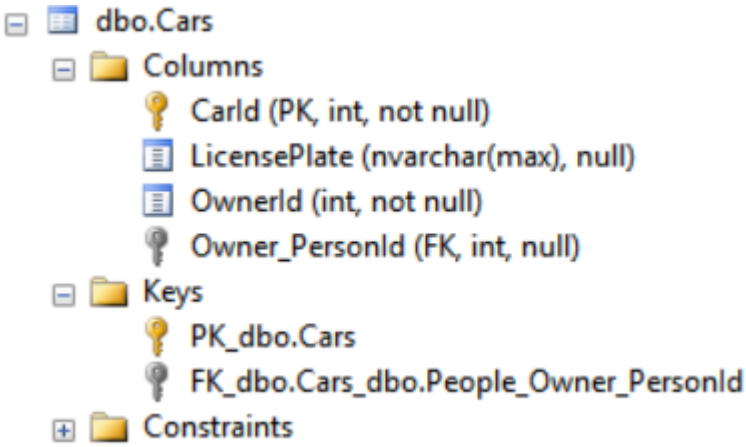
のマッピングに

のでは、EFがどのがキーであるのか、どこをするのかをしています。どうやってをする。のプロパティつ`Person`というの`Person`して`PersonId`プロパティは、というにEFをリード`PersonId`キーであり、それはタイプによってされるテーブルのキーをし`Person`。

しかし、タイプで*PersonId*を*OwnerId*と*Person* to *Owner*にするとどうなりますか

```
public class Car
{
   public int CarId { get; set; }
   public string LicensePlate { get; set; }
   public int OwnerId { get; set; }
   public virtual Person Owner { get; set; }
}
```

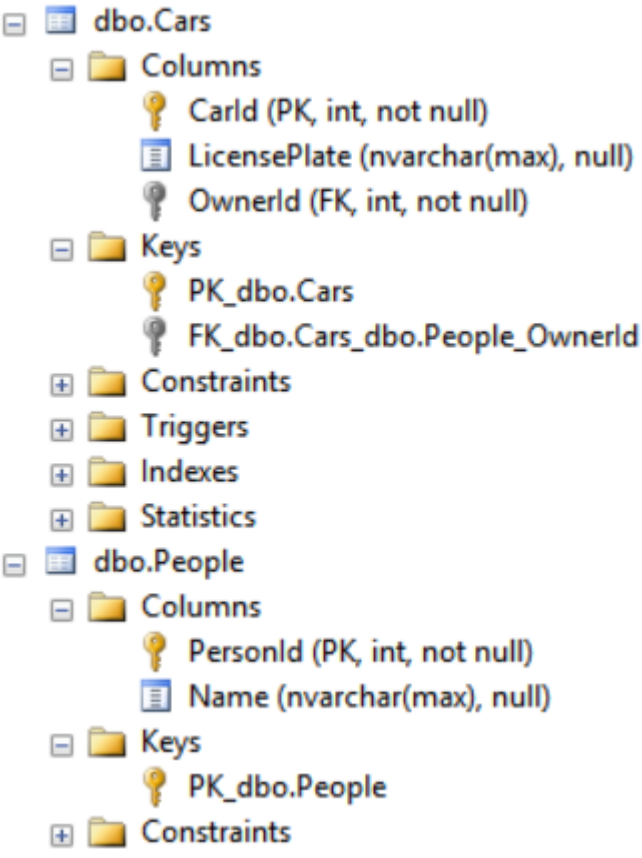
ながら、この、はしいDBスキーマをするにはです。



ない;モデルのとキーにするヒントをEFにてることができます。 `OwnerId`プロパティをFKとしてするように、 `Car`タイプをするだけです。エンティティタイプのコンフィグレーションをし、 `OnModelCreating()`でします

```
public class CarEntityTypeConfiguration : EntityTypeConfiguration<Car>
{
  public CarEntityTypeConfiguration()
  {
     this.HasRequired(c => c.Owner).WithMany(p => p.Cars).HasForeignKey(c => c.OwnerId);
  }
}
```

これは、に、 `Car`にはのプロパティ`Owner` *HasRequired* があり、 `Owner`のタイプでは、 `Cars`プロパティはのエンティティ *WithMany* をするためにされます。に、キーをすプロパティがされます *HasForeignKey* 。これはたちがむスキーマをします



Personからもをすることができます

```
public class PersonEntityTypeConfiguration : EntityTypeConfiguration<Person>
{
  public PersonEntityTypeConfiguration()
  {
    this.HasMany(p => p.Cars).WithRequired(c => c.Owner).HasForeignKey(c => c.OwnerId);
  }
}
```

アイデアはじですが、サイドがなります「このはくのをしており、それぞれのにはながいます」。 Personまたは Carからリレーションシップをコンフィグレーションするはありません。をめることもできますが、このはにじをするようにしてください

0または1のマッピング

のでは、がいなくてもはできません。そのをのからのオプションにしたいはどうすればいいですかまあ、1のやりをっているのはです。 CarのPersonIdをnullにするだけです

```
public class Car
{
    public int CarId { get; set; }
    public string LicensePlate { get; set; }
    public int? PersonId { get; set; }
    public virtual Person Person { get; set; }
}
```

そして、をうからHasOptional またはWithOptionalをします

```
public class CarEntityTypeConfiguration : EntityTypeConfiguration<Car>
{
  public CarEntityTypeConfiguration()
  {
     this.HasOptional(c => c.Owner).WithMany(p => p.Cars).HasForeignKey(c => c.OwnerId);
  }
}
```

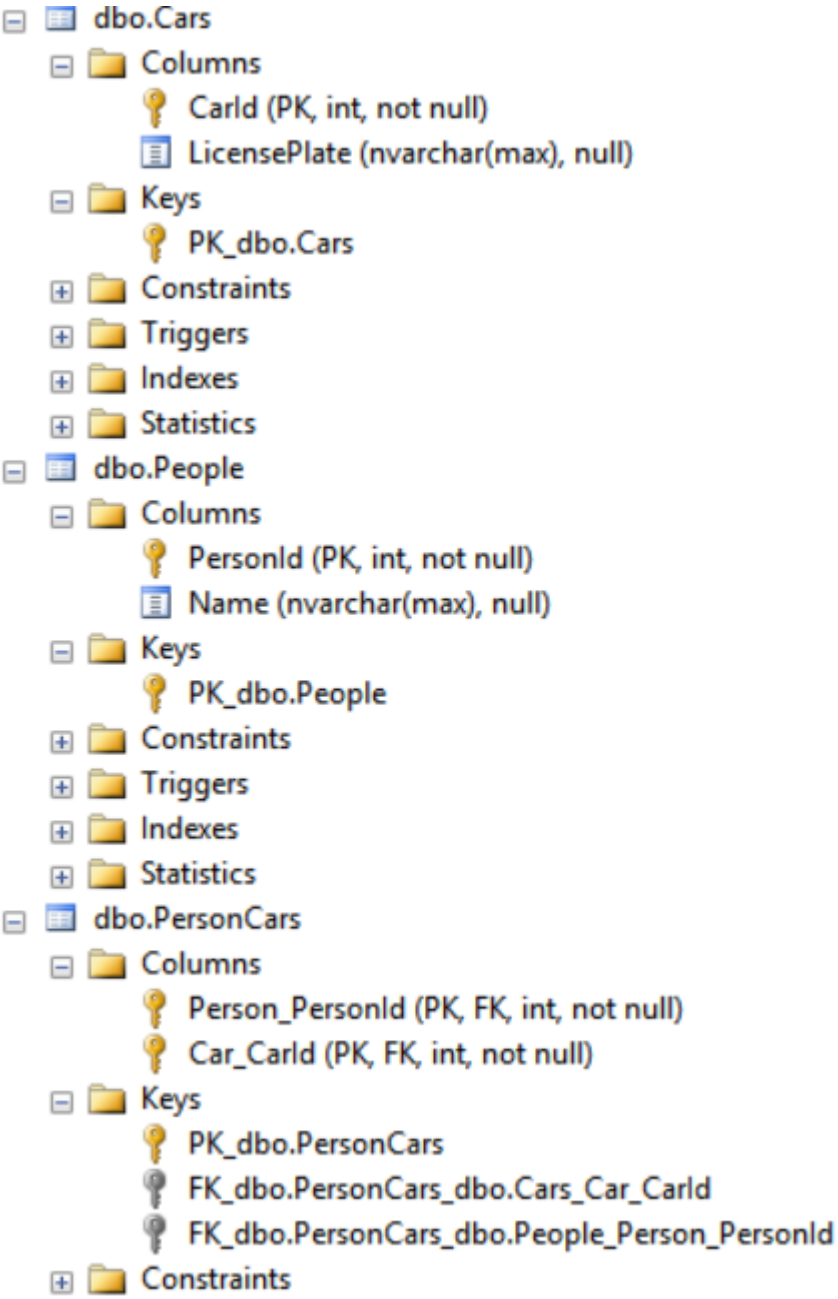
すべてのがのをち、すべてのがのをつことができるのシナリオにりましょうただし、はです。これはのです。もなは、EFにをってをさせることです。

のようにモデルをしてください

```
 public class Person
{
   public int PersonId { get; set; }
   public string Name { get; set; }
   public virtual ICollection<Car> Cars { get; set; }
}

public class Car
{
   public int CarId { get; set; }
```

```
    public string LicensePlate { get; set; }
    public virtual ICollection<Person> Owners { get; set; }
}
```

スキーマ



ほぼです。ごのように、EF はとのペアリングをできるテーブルのをしていました。

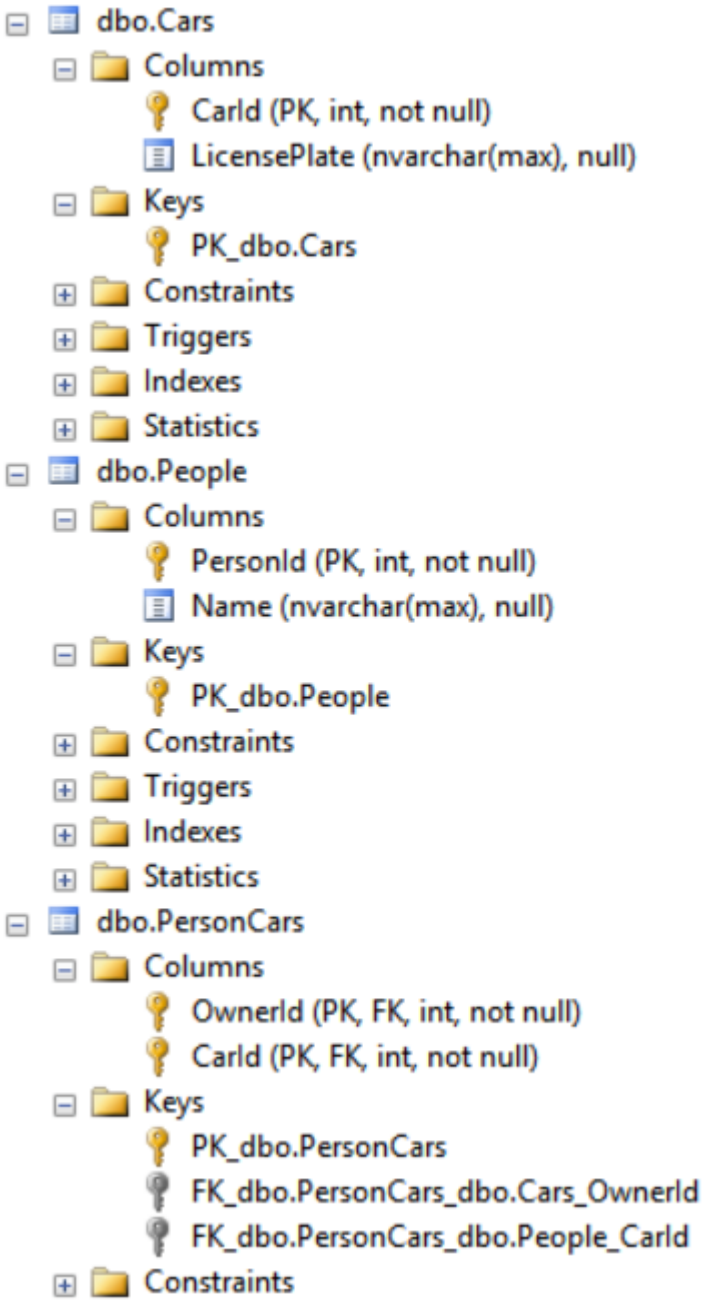
のカスタマイズ

ジョインテーブルのフィールドのをもうしにすることができます。これは、のをしてうことができますをどちらのからうかはありません。

```
public class CarEntityTypeConfiguration : EntityTypeConfiguration<Car>
{
    public CarEntityTypeConfiguration()
    {
```

```
        this.HasMany(c => c.Owners).WithMany(p => p.Cars)
            .Map(m =>
                {
                    m.MapLeftKey("OwnerId");
                    m.MapRightKey("CarId");
                    m.ToTable("PersonCars");
                }
            );
    }
}
```

でもみするのはとてもこのはくのち*hasMany*の、それぞれのはくのをする*WithMany*を。あなたがOwnerIdにするには、のキーをマップするように、この*MapLeftKey*、CarIdにキー*MapRightKey*テーブルPersonCarsと*ToTable*。これにより、にそのスキーマがられます。



カスタムエンティティ

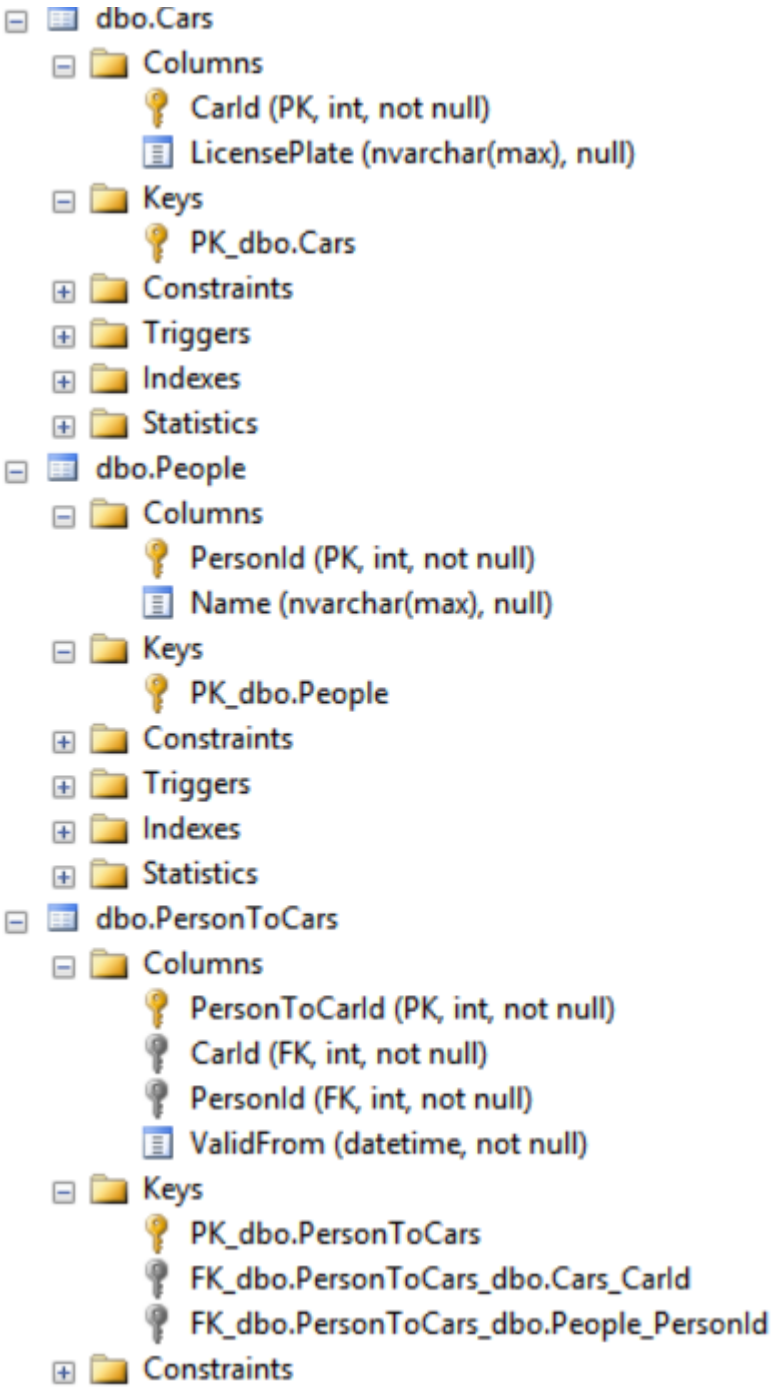
は、EFがエンティティをしてテ―ブルをできるようにするファンではないことをめなければなりません。あなたはテ―ブルをすることができないので、ととのけになをすることはできませんそれがなをいましょう。

また、の`CarId`はキ―のです。そのため、がしいをするは、いをしてしいものをするがあります。EFはあなたからこれをしますが、これはあなたのわりに、な/がインデックスのにつながるがあることはうまでもありません-いこと、これらの2つのをうがあることをながあり 、そのためには。

この、1つののとの1ののへのをつエンティティをすることができます。には、のけを2つの1のみわせとしてます

```
public class PersonToCar
{
   public int PersonToCarId { get; set; }
   public int CarId { get; set; }
   public virtual Car Car { get; set; }
   public int PersonId { get; set; }
   public virtual Person Person { get; set; }
   public DateTime ValidFrom { get; set; }
}

public class Person
{
  public int PersonId { get; set; }
  public string Name { get; set; }
  public virtual ICollection<PersonToCar> CarOwnerShips { get; set; }
}

public class Car
{
  public int CarId { get; set; }
  public string LicensePlate { get; set; }
  public virtual ICollection<PersonToCar> Ownerships { get; set; }
}

public class MyDemoContext : DbContext
{
  public DbSet<Person> People { get; set; }
  public DbSet<Car> Cars { get; set; }
  public DbSet<PersonToCar> PersonToCars { get; set; }
}
```

これにより、よりくのがになり、よりになります。ではカスタムデ―タをアソシエ―ションにすることができ、すべてのアソシエ―ションにのプライマリキ―があるため、そののやオ―ナ―のをできます。



これはには21ののみわせにぎないので、のでしたすべてのオプションをできます。

オンラインでEntity Frameworkとのマッピングコード11およびをむ https://riptutorial.com/ja/entity-framework/topic/9413/entity-frameworkとのマッピングコード1-1および

5: Entity Frameworkとのマッピングコードファースト11およびバリエーション

き

このトピックでは、Entity Frameworkをして11のをマップするについてします。

Examples

10または11マッピング

だから、あなたはのモデルをっているとびいましょう

```
public class Person
{
  public int PersonId { get; set; }
  public string Name { get; set; }
}

public class Car
{
  public int CarId { get; set; }
  public string LicensePlate { get; set; }
}

public class MyDemoContext : DbContext
{
  public DbSet<Person> People { get; set; }
  public DbSet<Car> Cars { get; set; }
}
```

そして、あなたはのようなをできるようにしたいとえています.1が1または0のをでき、すべてのが1のににしていますはであるため、CarAがPersonAにしていればPersonA ' 'CarA。

モデルをししてみましょうナビゲーションプロパティとキープロパティをしてください

```
public class Person
{
  public int PersonId { get; set; }
  public string Name { get; set; }
  public int CarId { get; set; }
  public virtual Car Car { get; set; }
}

public class Car
{
  public int CarId { get; set; }
  public string LicensePlate { get; set; }
  public int PersonId { get; set; }
```

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

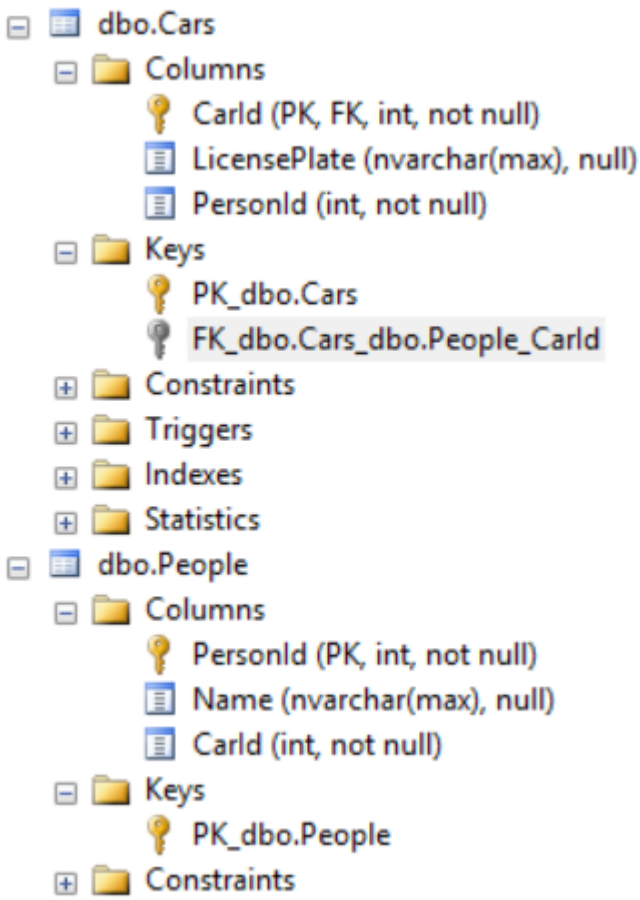
そして

```
public class CarEntityTypeConfiguration : EntityTypeConfiguration<Car>
{
  public CarEntityTypeConfiguration()
  {
     this.HasRequired(c => c.Person).WithOptional(p => p.Car);
  }
}
```

このまでに、これはであるべきです。にはな *HasRequired* があり、そのにはオプションの *WithOptional* があります。このをどちらのからするかはありませんが、Has / WithとRequired / Optionalのなみわせをするはしてください。 `Person`からると、のようになります。

```
public class PersonEntityTypeConfiguration : EntityTypeConfiguration<Person>
{
  public PersonEntityTypeConfiguration()
  {
     this.HasOptional(p => p.Car).WithOptional(c => c.Person);
  }
}
```

さて、dbスキーマをてみましょう



にてください Carにするために、 PeopleにはFKがないことがわかります。また、 CarのFKは PersonIdではなくCarIdです。 FKののスクリプトはのとおりです。

```
ALTER TABLE [dbo].[Cars]  WITH CHECK ADD  CONSTRAINT [FK_dbo.Cars_dbo.People_CarId] FOREIGN
KEY([CarId])
REFERENCES [dbo].[People] ([PersonId])
```

これは、モデルにあるCarIdとPersonId foregnのキープロパティはにされることをします。それらはデータベースにありますが、されるようにキーではありません。これは、11のマッピングではFKをEFモデルにできないためです。リレーショナルデータベースでは11のマッピングがかなりになるからです。

アイデアは、すべてのがちょうど1のをでき、そのはそのにしかしていないということです。または、にけられていないのがあるかもしれません。

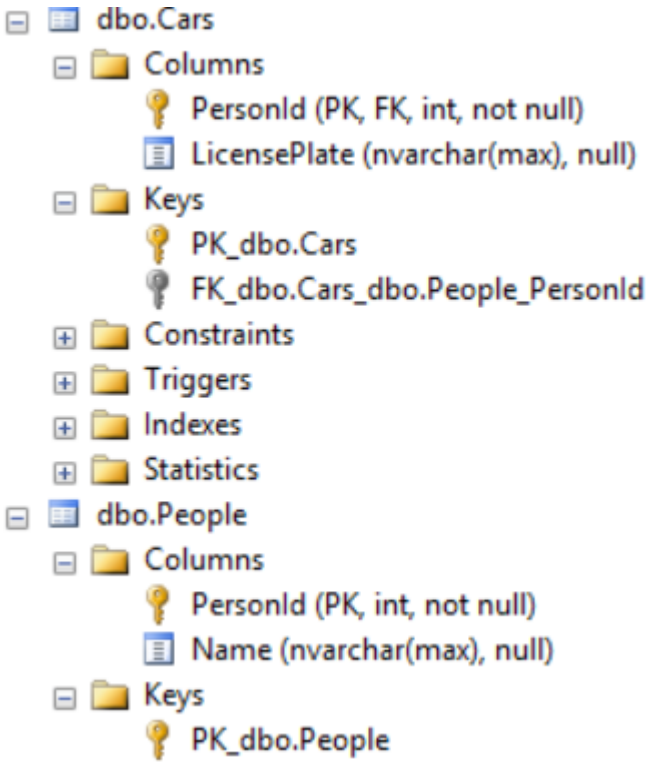
だから、これはどのようにキーでできますかもちろん、そこにがPersonIdでCar 、およびCarIdで People。 すべてのが、つだけのをつことができることをするために、 PersonIdででなければならないであろうCar。 PersonIdがPeopleでであれば、 PersonIdがNULL をたないのの、どのようにのレコードをできますかできませんは、SQL Server 2008でフィルタリングされたユニークなインデックスをできますが、のRDBMSはもちろんのこと、このテクニカルをれてみましょう。あなたがのをするはもちろん...

Peopleテーブルと Carテーブルにじプライマリキーされたレコードのじがある、このルールをするののです。そして、これをうには、 CarIdCar々のPKにPKとFKのでなければなりません。そして、これによってスキーマがする。がこれをうとき、はCar PersonId PK / FKにをけ、それにじてします

```
public class Person
{
  public int PersonId { get; set; }
  public string Name { get; set; }
  public virtual Car Car { get; set; }
}

public class Car
{
  public string LicensePlate { get; set; }
  public int PersonId { get; set; }
  public virtual Person Person { get; set; }
}

public class CarEntityTypeConfiguration : EntityTypeConfiguration<Car>
{
  public CarEntityTypeConfiguration()
  {
     this.HasRequired(c => c.Person).WithOptional(p => p.Car);
     this.HasKey(c => c.PersonId);
  }
}
```

ではありませんが、しいかもしれません。それでも、このソリューションをするは、のにするた

め、をするがあります。このモデルからされたスキーマはのとおりです。



したがって、このはデータベーススキーマによってされるのではなく、Entity Frameworkによってされます。だからこそあなたはこれをうときににでなければなりません。

11のマッピング

11のマッピングがなもしいことです。

これをキーでどのようにできるかをしてみましょう。ここでも、 `CarId`で`People`をし`CarId`で`Car` 、そして`PersonId`をしで`PersonId`で`People` 。

のレコードをするはどうなりますかこれをさせるためには、このカーレコードに`PersonId`されているがあります。この`PersonId`をにするには、 `People`のするレコードがするがあります。それでは、のレコードをしてみましょう。しかし、これがするためには、な`CarId`がになければなりませんが、そのはまだされていませんされたのレコードをにするがあるため、できません。しかし、されたレコードは、のレコードをするためすることはできません。にするがありますキー - ception :)。

だから、これはなですることもできません。ここでも、キーのいずれかをするがあります。あなたがとすのはあなたです。キーがされているを「」とび、キーなしでっているを「プリンシパル」とびます。また、のをするために、PKはFKでなければなりません。したがって、FKをしてモデルにインポートすることはサポートされていません。

ここにがあります

```
public class CarEntityTypeConfiguration : EntityTypeConfiguration<Car>
{
```

```
  public CarEntityTypeConfiguration()
  {
    this.HasRequired(c => c.Person).WithRequiredDependent(p => p.Car);
    this.HasKey(c => c.PersonId);
  }
}
```

これで、あなたはにそれのをているはずです:)あなたはもうをふこともできることをえておいてください。/プリンシパルバ―ジョンのWithRequiredをすることにしてください。

```
public class PersonEntityTypeConfiguration : EntityTypeConfiguration<Person>
{
  public PersonEntityTypeConfiguration()
  {
    this.HasRequired(p => p.Car).WithRequiredPrincipal(c => c.Person);
  }
}
```

DBスキ―マをチェックすると、11またはゼロのとまったくじであることがわかります。もう、これはスキ―マによってされるのではなく、EFによってされるからです。だから、びしてください:)

1または01またはゼロのマッピン グ

に、がオプションのをにてみましょう。

ここまでで、これらのではにしているはずです。だから、はにりませんし、2つのFK-sとなをえていて、これらのをしないについてします。 EFそのもののにあります。

するがあるはのとおりです。

```
public class CarEntityTypeConfiguration : EntityTypeConfiguration<Car>
{
  public CarEntityTypeConfiguration()
  {
    this.HasOptional(c => c.Person).WithOptionalPrincipal(p => p.Car);
    this.HasKey(c => c.PersonId);
  }
}
```

もう、からもできます。しいをするようにしてください:)

オンラインでEntity Frameworkとのマッピングコードファースト11およびバリエーションをむ
https://riptutorial.com/ja/entity-framework/topic/9412/entity-frameworkとのマッピングコードファースト-11およびバリエーション

6: Entity-Framework with Postgresql

Examples

NpgsqlddexproviderをしてPostgresSqlでEntity Framework 6.1.3をするためにな

1C\ Windows \ Microsoft.NET \ Framework \ v4.0.30319 \ ConfigとC\ Windows \ Microsoft.NET \ Framework64 \ v4.0.30319 \ ConfigのからMachine.configのバックアップをった

2のにコピーしてします。

a `<system.data> <DbProviderFactories>` つけてします。

```
            <add name="Npgsql Data Provider" invariant="Npgsql" support="FF"
            description=".Net Framework Data Provider for Postgresql Server"
            type="Npgsql.NpgsqlFactory, Npgsql, Version=2.2.5.0, Culture=neutral,
PublicKeyToken=5d8b90d52f46fda7" />
```

bエントリのにすでにするは、verisonをチェックしてします。

3. のファイルをしたファイルにきえます。
4. VS2013のコマンドプロンプトをとしてします。
5. Npgsqlがすでにインストールされているは、コマンド "gacutil -u Npgsql"をアンインストールして、 "gacutil -i [path of dll]"コマンドでNpgsql 2.5.0のしいバージョンをインストールしてください
6. Mono.Security 4.0.0.0にしてをう
7. NpgsqlDdexProvider-2.2.0-VS2013.zipをダウンロードし、NpgsqlDdexProvider.vsixをしますVisual Studioのすべてのインスタンスをじます
8. EFTools6.1.3-beta1ForVS2013.msiがつかりました。
9. しいプロジェクトをした、Manage Nuget Packages.10からEntityFramework6.1.3、NpgSql 2.5.0、NpgSql.EntityFramework2.5.0をインストールします。あなたのMVCプロジェクト

オンラインでEntity-Framework with Postgresqlをむ https://riptutorial.com/ja/entity-framework/topic/7647/entity-framework-with-postgresql

7: Entity-Frameworkコードのの

Examples

をにする

エンティティフレームワークでコードファーストマイグレーションをにするには、のコマンドをします。

```
Enable-Migrations
```

パッケージマネージャコンソールで

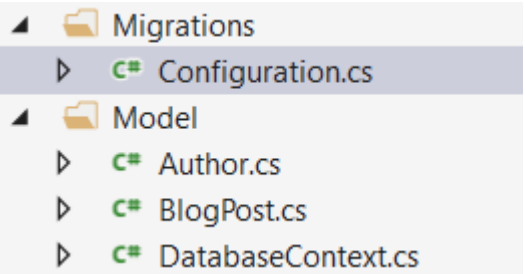
EFによってされるデータベースオブジェクトをむな`DbContext`がです。このでは、データベースコンテキストにオブジェクト`BlogPost`と`Author`がまれます。

```
internal class DatabaseContext: DbContext
{
    public DbSet<Author> Authors { get; set; }

    public DbSet<BlogPost> BlogPosts { get; set; }
}
```

コマンドをすると、のがされます。

```
PM> Enable-Migrations
Checking if the context targets an existing database...
Code First Migrations enabled for project <YourProjectName>.
PM>
```

さらに、しいフォルダ`Migrations`は、1つのファイル`Configuration.cs`とともにされます。



のステップは、のデータベースをするのデータベーススクリプトをすることですのを。

のをする

をにしたら このをしてください、すべてのデータベーステーブル、インデックス、およびのをむのをできるようになりました。

このコマンドをしてをすることができます

```
Add-Migration <migration-name>
```

このコマンドは、のとにされる2つのメソッド`Up`と`Down`をむしいクラスをします。

のにづいてコマンドをして、 *Initial*というマイグレーションをします。

```
PM> Add-Migration Initial
Scaffolding migration 'Initial'.
The Designer Code for this migration file includes a snapshot of your current Code
First model. This snapshot is used to calculate the changes to your model when you
scaffold the next migration. If you make additional changes to your model that you
want to include in this migration, then you can re-scaffold it by running
'Add-Migration Initial' again.
```

しいファイルタイムスタンプ _Initial.csがされますなものだけがここにされます。

```
public override void Up()
{
   CreateTable(
       "dbo.Authors",
        c => new
           {
               AuthorId = c.Int(nullable: false, identity: true),
               Name = c.String(maxLength: 128),
           })
        .PrimaryKey(t => t.AuthorId);

    CreateTable(
       "dbo.BlogPosts",
       c => new
           {
                Id = c.Int(nullable: false, identity: true),
                Title = c.String(nullable: false, maxLength: 128),
                Message = c.String(),
                Author_AuthorId = c.Int(),
            })
        .PrimaryKey(t => t.Id)
        .ForeignKey("dbo.Authors", t => t.Author_AuthorId)
        .Index(t => t.Author_AuthorId);
}

public override void Down()
{
    DropForeignKey("dbo.BlogPosts", "Author_AuthorId", "dbo.Authors");
    DropIndex("dbo.BlogPosts", new[] { "Author_AuthorId" });
    DropTable("dbo.BlogPosts");
    DropTable("dbo.Authors");
}
```

ごのように、メソッド`Up()` 2つのテーブル`Authors`と`BlogPosts`がされ、それにじてフィールドがされます。また、2つのテーブルのは、フィールド`Author_AuthorId`することによってされます。に、メソッド`Down()`は、アクティビティをりそうとします。

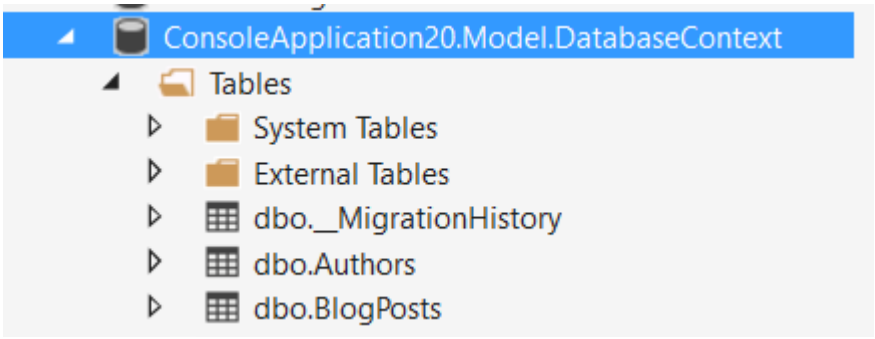
にがあるは、のコマンドをしてをデータベースにできます。

```
Update-Database
```

すべてののこのは*Initial* -migrationがデータベースにされた、シードメソッドがされますな

```
PM> update-database
Specify the '-Verbose' flag to view the SQL statements being applied to the target
database.
Applying explicit migrations: [201609302203541_Initial].
Applying explicit migration: 201609302203541_Initial.
Running Seed method.
```

SQLエクスプローラで、アクティビティのをできます。



`Add-Migration`と`Update-Database`のコマンド`Add-Migration`は、アクティビティをするためのいくつかのオプションがあります。すべてのオプションをするには、

```
get-help Add-Migration
```

そして

```
get-help Update-Database
```

のデータのシード

をにしてしたは、データベースのデータをにまたはするがあります。いくつかのがありますが、なのは、 `enable-migrations`びしたにされるConfigurationファイルのSeedメソッドをできます。

`Seed()`はデータベースコンテキストをパラメータとしてし、このでEFをできます。

```
protected override void Seed(Model.DatabaseContext context);
```

`Seed()`では、すべてのタイプのアクティビティをできます。らかのがした、なトランザクションされたパッチもがロールバックされます。

がのにのみデータをするのは、のようになります。

```
protected override void Seed(Model.DatabaseContext context)
```

```
{
    if (!context.Customers.Any()) {
        Customer c = new Customer{ Id = 1, Name = "Demo" };
        context.Customers.Add(c);
        context.SaveData();
    }
}
```

EFがするれたは、メソッド`AddOrUpdate()`です。このでは、プライマリキーにづいてデータをしたり、データがしないはデータをすることができますこのは、されたConfiguration.csのソースコードからされます。

```
protected override void Seed(Model.DatabaseContext context)
{
    context.People.AddOrUpdate(
      p => p.FullName,
      new Person { FullName = "Andrew Peters" },
      new Person { FullName = "Brice Lambson" },
      new Person { FullName = "Rowan Miller" }
    );
}
```

> のパッチに`Seed()`がびされることにしてください。パッチにデータをまたはシードするがあるは、のアプローチをするがあります。

にSqlをする

たとえば、のをからにするです。この、されたフィールドがに`NULL`いるにしては、でいくつかのデフォルトをするがあり`NULL`。デフォルトがシンプルえば "0"の、カラムで`default`または`defaultSql`プロパティをすることができ`default`。それほどではないは、の`Up()`または`Down()`メンバーで`Sql()`をできます。

ここにがあります。データセットのとしてメールアドレスをむクラス*Author*をします。これで、メールアドレスをフィールドにすることにしました。のをするには、`fullname@example.com`ようなダミーのメールアドレスをするというスマートなえがあります。なはスペースなしのなです。`[Required] Email [Required]`フィールドに`[Required]`をすると、のがされます。

```
public partial class AuthorsEmailRequired : DbMigration
{
    public override void Up()
    {
        AlterColumn("dbo.Authors", "Email", c => c.String(nullable: false, maxLength: 512));
    }

    public override void Down()
    {
        AlterColumn("dbo.Authors", "Email", c => c.String(maxLength: 512));
    }
}
```

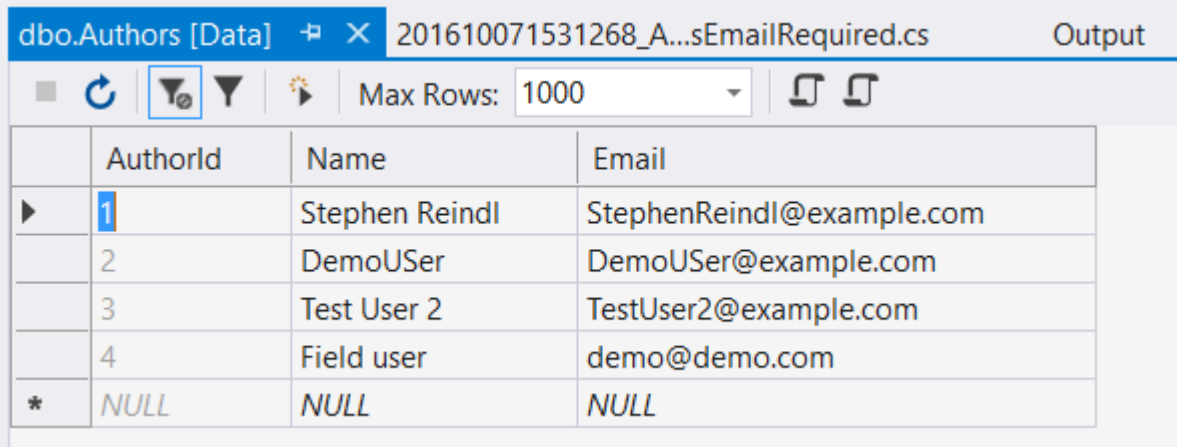
これは、いくつかのNULLフィールドがデータベースにあるにはします。

'NULL'を 'Email'、 'App.Model.DatabaseContext.dbo.Authors'にできません。はNULL をしません。 UPDATEにします。

`AlterColumn`コマンドのにのようにすると、

```
Sql(@"Update dbo.Authors
    set Email = REPLACE(name, ' ', '') + N'@example.com'
    where Email is null");
```

`update-database`びしがし、テーブルはのようになりますサンプルデータがされます。



そのの

データベースのすべてのタイプのDMLおよびDDLアクティビティーに`Sql()`をできます。トランザクションのとしてされます。 SQLにがすると、ながし、ロールバックがされます。

コードで "Update-Database"をする

でするアプリケーションでは、データベースのがなことがよくあります。 `Add-Migration`コマンドをしてデータベースパッチをしたは、のでもプログラムをするがあります。

にするはのとおりです。

- にVisual Studioがインストールされていない
- の/にはがされていません。

は、するをし、それらをにするのコードシーケンスです。エラーがしてもデータがわれないように、なトランザクションとをってください。

```
void UpdateDatabase(MyDbConfiguration configuration) {
   DbMigrator dbMigrator = new DbMigrator( configuration);
   if ( dbMigrator.GetPendingMigrations().Any() )
   {
      // there are pending migrations run the migration job
      dbMigrator.Update();
   }
```

```
}
```

`MyDbConfiguration`はあなたのソースのどこかであなたのセットアップです

```
public class MyDbConfiguration : DbMigrationsConfiguration<ApplicationDbContext>
```

エンティティフレームワークコードのの

1. コンソールアプリケーションをします。
2. 「Package Manager Console」で`Install-Package EntityFramework`して、EntityFrameworkのnugetパッケージを`Install-Package EntityFramework`。
3. app.configファイルにをします。に`providerName="System.Data.SqlClient"`をめることがです。
4. あなたがむようにパブリッククラスをします。えば、"`Blog`"
5. DbContextをするContextClassをすると、「`BlogContext`」のようなものがあります
6. DbSetのコンテキストでプロパティをします。のようなものがあります

```
public class Blog
{
    public int Id { get; set; }

    public string Name { get; set; }
}
public class BlogContext: DbContext
{
    public BlogContext(): base("name=Your_Connection_Name")
    {
    }

    public virtual DbSet<Blog> Blogs{ get; set; }
}
```

7. コンストラクタここではYour_Connection_Nameにをすことがです。
8. パッケージマネージャコンソールで、`Enable-Migration`コマンドをします。これにより、プロジェクトにフォルダがされます
9. `Add-Migration Your_Arbitrary_Migraiton_Name`コマンドをすると、Migrationsフォルダに、UpとDownの2つのメソッドをつクラスがされます。
10. ブログテーブルをしてデータベースをするには`Update-Database`コマンドをします

オンラインでEntity-Frameworkコードののをむ https://riptutorial.com/ja/entity-framework/topic/7157/entity-frameworkコードのの

8: EntityFrameworkによるコードファースト

Examples

ごとのテーブル

このアプローチは、すべてのをするためにデータベースに1つのテーブルをします。

```
public abstract class Person
{
    public int Id { get; set; }
    public string Name { get; set; }
    public DateTime BirthDate { get; set; }
}

public class Employee : Person
{
    public DateTime AdmissionDate { get; set; }
    public string JobDescription { get; set; }
}

public class Customer : Person
{
    public DateTime LastPurchaseDate { get; set; }
    public int TotalVisits { get; set; }
}

// On DbContext
public DbSet<Person> People { get; set; }
public DbSet<Employee> Employees { get; set; }
public DbSet<Customer> Customers { get; set; }
```

されるテーブルはのようになります。

テーブルPeopleフィールドIDDiscrimitator AdmissionDate JobDescription LastPurchaseDate TotalVisits

'Discriminator'はのサブクラスのをし、 'AdmissionDate'、 'JobDescription'、 'LastPurchaseDate'、 'TotalVisits'はnullにできます。

- データベースはくのページングをとすることがありますが、くのにしてはがないため、パフォーマンスがします。
- シンプルでいやすくする
- よりくのサブクラスとフィールドをにできます

- 3ウィキペディア3
- ヌルなフィールドをたくさんする

テーブルのごと

このアプローチでは、データベースにn + 1のテーブルがされ、をすべてしますnはサブクラスの
。

の

```
public abstract class Person
{
    public int Id { get; set; }
    public string Name { get; set; }
    public DateTime BirthDate { get; set; }
}

[Table("Employees")]
public class Employee : Person
{
    public DateTime AdmissionDate { get; set; }
    public string JobDescription { get; set; }
}

[Table("Customers")]
public class Customer : Person
{
    public DateTime LastPurchaseDate { get; set; }
    public int TotalVisits { get; set; }
}

// On DbContext
public DbSet<Person> People { get; set; }
public DbSet<Employee> Employees { get; set; }
public DbSet<Customer> Customers { get; set; }
```

されるテーブルはのようになります。

テーブルPeopleフィールドID

フィールドPersonId AdmissionDate JobDescription

テーブルフィールドPersonId LastPurchaseDate TotalVisits

すべてのテーブルの 'PersonId'がキーとなり、People.Idのになります

- されたテーブル
- カラムやサブクラスのが
- nullなはありません

- データをするにはがです
- サブクラスのはよりです

オンラインでEntityFrameworkによるコードファーストをむ https://riptutorial.com/ja/entity-framework/topic/7715/entityframeworkによる-コードファースト-

9: entity-frameworkの.t4テンプレート

Examples

モデルににインターフェイスをする

がなにはかなりきいのモデルをしてする、インターフェイスをしてモデルをするのははがかかります。このような、モデルになるいをすることができます。

のでは、のをつクラスににインターフェイスをするをします。

おいのモデルにするためにく.ttファイルをEntityClassOpeningを、のを、これがされますIPolicyNumberつエンティティにインタフェースをPOLICY_NOコラムを、そしてIUniqueIdにUNIQUE_ID

```
public string EntityClassOpening(EntityType entity)
{
    var stringsToMatch = new Dictionary<string,string> { { "POLICY_NO", "IPolicyNumber" }, {
"UNIQUE_ID", "IUniqueId" } };
    return string.Format(
        CultureInfo.InvariantCulture,
        "{0} {1}partial class {2}{3}{4}",
        Accessibility.ForType(entity),
        _code.SpaceAfter(_code.AbstractOption(entity)),
        _code.Escape(entity),
        _code.StringBefore(" : ", _typeMapper.GetTypeName(entity.BaseType)),
        stringsToMatch.Any(o => entity.Properties.Any(n => n.Name == o.Key)) ? " : " +
string.Join(", ", stringsToMatch.Join(entity.Properties, l => l.Key, r => r.Name, (l,r) =>
l.Value)) : string.Empty);
}
```

これはのケースですが、 .ttテンプレートをできるというをしています。

エンティティクラスへのXMLドキュメントの

されたすべてのモデルクラスには、デフォルトでされたドキュメンテーションコメントはありません。あなたは、すべてのされたエンティティクラスのXMLドキュメントコメントをしたいは、[モデル] .ttのこのをつける モデルは、のEDMXファイルです

```
foreach (var entity in typeMapper.GetItemsToGenerate<EntityType>(itemCollection))
{
    fileManager.StartNewFile(entity.Name + ".cs");
    BeginNamespace(code); // used to write model namespace
#>
<#=codeStringGenerator.UsingDirectives(inHeader: false)#>
```

のにすように、XMLドキュメントのコメントをUsingDirectivesのにすることができます。

```
foreach (var entity in typeMapper.GetItemsToGenerate<EntityType>(itemCollection))
{
```

```
    fileManager.StartNewFile(entity.Name + ".cs");
    BeginNamespace(code);
#>
/// <summary>
/// <#=entity.Name#> model entity class.
/// </summary>
<#=codeStringGenerator.UsingDirectives(inHeader: false)#>
```

されたドキュメントのコメントには、にすエンティティがまれているがあります。

```
/// <summary>
/// Example model entity class.
/// </summary>
public partial class Example
{
    // model contents
}
```

オンラインでentity-frameworkの.t4テンプレートをむ https://riptutorial.com/ja/entity-framework/topic/3964/entity-frameworkの-t4テンプレート

10: SQLiteをしたエンティティフレームワーク

き

SQLiteは、サーバレスのトランザクションレスSQLデータベースです。にできる.NET SQLiteライブラリとEntity Framework SQLiteプロバイダのをすることにより、.NETアプリケーションでできます。このトピックでは、Entity Framework SQLiteプロバイダのセットアップとについてします。

Examples

SQLiteプロバイダでEntity Frameworkをするプロジェクトをする

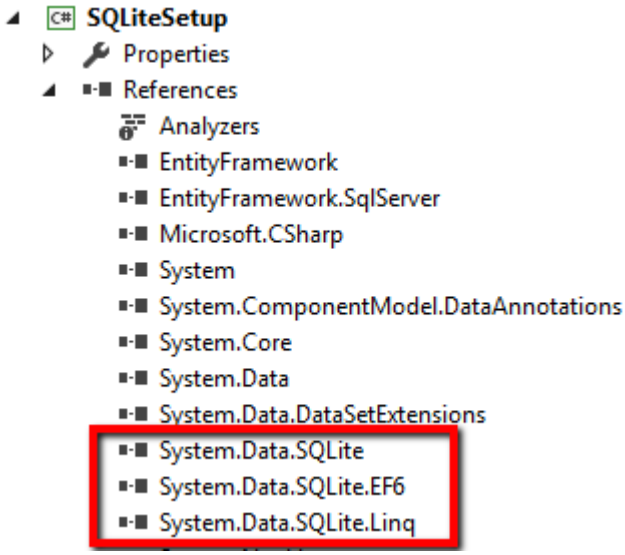
Entity Frameworkライブラリには、SQL Serverプロバイダのみがしています。 SQLiteをするには、のとがです。なはすべてNuGetでできます。

SQLiteライブラリをインストールする

NuGet Package Manager Consoleをして、すべてのされたdepedenciesをインストールできます。 `Install-Package System.Data.SQLite`コマンド`Install-Package System.Data.SQLite`し`Install-Package System.Data.SQLite`。

```
PM> Install-Package System.Data.SQLite
Attempting to gather dependency information for package 'System.Data.SQLite.1.0.104' with respect to proje
Attempting to resolve dependencies for package 'System.Data.SQLite.1.0.104' with DependencyBehavior 'Lowes
Resolving actions to install package 'System.Data.SQLite.1.0.104'
Resolved actions to install package 'System.Data.SQLite.1.0.104'
Adding package 'EntityFramework.6.0.0' to folder 'c:\users\jason.tyler\documents\visual studio 2015\Projec
Added package 'EntityFramework.6.0.0' to folder 'c:\users\jason.tyler\documents\visual studio 2015\Project
Added package 'EntityFramework.6.0.0' to 'packages.config'
Executing script file 'c:\users\jason.tyler\documents\visual studio 2015\Projects\SQLiteSetup\packages\Ent
Executing script file 'c:\users\jason.tyler\documents\visual studio 2015\Projects\SQLiteSetup\packages\Ent

Type 'get-help EntityFramework' to see all available Entity Framework commands.
Successfully installed 'EntityFramework 6.0.0' to SQLiteSetup
Adding package 'System.Data.SQLite.Core.1.0.104' to folder 'c:\users\jason.tyler\documents\visual studio 2
Added package 'System.Data.SQLite.Core.1.0.104' to folder 'c:\users\jason.tyler\documents\visual studio 20
Added package 'System.Data.SQLite.Core.1.0.104' to 'packages.config'
Successfully installed 'System.Data.SQLite.Core 1.0.104' to SQLiteSetup
Adding package 'System.Data.SQLite.EF6.1.0.104' to folder 'c:\users\jason.tyler\documents\visual studio 20
Added package 'System.Data.SQLite.EF6.1.0.104' to folder 'c:\users\jason.tyler\documents\visual studio 201
Added package 'System.Data.SQLite.EF6.1.0.104' to 'packages.config'
Executing script file 'c:\users\jason.tyler\documents\visual studio 2015\Projects\SQLiteSetup\packages\Sys
Successfully installed 'System.Data.SQLite.EF6 1.0.104' to SQLiteSetup
Adding package 'System.Data.SQLite.Linq.1.0.104' to folder 'c:\users\jason.tyler\documents\visual studio 2
Added package 'System.Data.SQLite.Linq.1.0.104' to folder 'c:\users\jason.tyler\documents\visual studio 20
Added package 'System.Data.SQLite.Linq.1.0.104' to 'packages.config'
Successfully installed 'System.Data.SQLite.Linq 1.0.104' to SQLiteSetup
Adding package 'System.Data.SQLite.1.0.104', which only has dependencies, to project 'SQLiteSetup'.
Adding package 'System.Data.SQLite.1.0.104' to folder 'c:\users\jason.tyler\documents\visual studio 2015\P
Added package 'System.Data.SQLite.1.0.104' to folder 'c:\users\jason.tyler\documents\visual studio 2015\Pr
Added package 'System.Data.SQLite.1.0.104' to 'packages.config'
Successfully installed 'System.Data.SQLite 1.0.104' to SQLiteSetup
```

のように、`System.Data.SQLite`インストールすると、するすべてのライブラリがインストールされます。これには、SQLiteのEFプロバイダである`System.Data.SQLite.EF6`まれます。このプロジェクトでは、SQLiteプロバイダをするためになアセンブリもするようになりました。



アンマネージライブラリをむ

SQLiteライブラリは、`SQLite.Interop.dll`というのアンマネージアセンブリにしています。これは、SQLiteパッケージでダウンロードされたパッケージアセンブリにまれており、プロジェクトの

ビルドににビルドディレクトリにコピーされます。ただし、されていないため、リストにはまれません。しかし、このアセンブリは、SQLiteアセンブリがするためのアプリケーションとともにされることがもです。

このアセンブリはビットにします。つまり、サポートするビット**x86 / x64**ごとにのアセンブリをめるがあります。

プロジェクトの**App.config**の

SQLiteをEntity Frameworkプロバイダとしてするには、 `app.config`ファイルにいくつかのがです。

な

パッケージをインストールすると、 `app.config`ファイルがにされ、SQLiteおよびSQLite EFになエントリがされます。ながら、これらのエントリにはいくつかのエラーがあります。それらはしくするにするがあります。

に、 `DbProviderFactories`をファイルにします。これは`system.data`にあり、をみます

```
<DbProviderFactories>
  <remove invariant="System.Data.SQLite.EF6" />
  <add name="SQLite Data Provider (Entity Framework 6)" invariant="System.Data.SQLite.EF6"
description=".NET Framework Data Provider for SQLite (Entity Framework 6)"
type="System.Data.SQLite.EF6.SQLiteProviderFactory, System.Data.SQLite.EF6" />
  <remove invariant="System.Data.SQLite" /><add name="SQLite Data Provider"
invariant="System.Data.SQLite" description=".NET Framework Data Provider for SQLite"
type="System.Data.SQLite.SQLiteFactory, System.Data.SQLite" />
</DbProviderFactories>
```

これは、のエントリをむようにすることができます

```
<DbProviderFactories>
    <add name="SQLite Data Provider" invariant="System.Data.SQLite.EF6" description=".NET
Framework Data Provider for SQLite" type="System.Data.SQLite.SQLiteFactory,
System.Data.SQLite" />
</DbProviderFactories>
```

これで、EF6 SQLiteプロバイダがSQLiteファクトリをするようにしました。

SQLiteをする

は、ルートのファイルにできます。 SQLiteデータベースにアクセスするためのをします。

```
<connectionStrings>
    <add name="TestContext" connectionString="data source=testdb.sqlite;initial
catalog=Test;App=EntityFramework;" providerName="System.Data.SQLite.EF6"/>
```

```
</connectionStrings>
```

ここでなことは`provider`。これは`System.Data.SQLite.EF6`されています。これは、EFに、このをするときに、SQLiteをするようにします。された`data source`はなるであり、SQLiteデータベースのとにします。

あなたのSQLite DbContext

すべてのインストールとがしたら、SQLiteデータベースでする`DbContext`をできます。

```
public class TestContext : DbContext
{
    public TestContext()
        : base("name=TestContext") { }
}
```

`name=TestContext`することで、`app.config`ファイルにあるTestContextをしてコンテキストをするがあることをしています。そのはSQLiteをするようにされていたので、このコンテキストはSQLiteデータベースをします。

オンラインでSQLiteをしたエンティティフレームワークをむ https://riptutorial.com/ja/entity-framework/topic/9280/sqliteをしたエンティティフレームワーク

11: エンティティフレームワークのベストプラクティスシンプルプロフェッショナル

き

このでは、Entity Frameworkをするでなをします。

シンプル1つのクラス1つのインターフェースをつ

プロフェッショナル SOLIDアーキテクチャのがされるため

はもっとしたくない...それをしもう

Examples

1- Entity Framework @データレイヤー

このでは、2つのテーブルをつ「Company」というなデータベースをします。

[dbo]。[Categories][CategoryID]、[CategoryName]

[dbo]。[][ProductID]、[CategoryID]、[ProductName]

1-1 Entity Frameworkコードをする

このでは、Entity Frameworkコードプロジェクトライブラリをします このでどのようにうことができますかに、のクラスがあります

```
public partial class CompanyContext : DbContext
public partial class Product
public partial class Category
```

1-2 インターフェースの

たちは、のための1つのインタフェースをします

```
public interface IDbRepository : IDisposable
{
    #region Tables and Views functions

    IQueryable<TResult> GetAll<TResult>(bool noTracking = true) where TResult : class;
    TEntity Add<TEntity>(TEntity entity) where TEntity : class;
    TEntity Delete<TEntity>(TEntity entity) where TEntity : class;
    TEntity Attach<TEntity>(TEntity entity) where TEntity : class;
    TEntity AttachIfNot<TEntity>(TEntity entity) where TEntity : class;
```

```
    #endregion Tables and Views functions

    #region Transactions Functions

    int Commit();
    Task<int> CommitAsync(CancellationToken cancellationToken = default(CancellationToken));

    #endregion Transactions Functions

    #region Database Procedures and Functions

    TResult Execute<TResult>(string functionName, params object[] parameters);

    #endregion Database Procedures and Functions
}
```

1-3 インターフェースの

```
/// <summary>
/// Implementing basic tables, views, procedures, functions, and transaction functions
/// Select (GetAll), Insert (Add), Delete, and Attach
/// No Edit (Modify) function (can modify attached entity without function call)
/// Executes database procedures or functions (Execute)
/// Transaction functions (Commit)
/// More functions can be added if needed
/// </summary>
/// <typeparam name="TEntity">Entity Framework table or view</typeparam>
public class DbRepository : IDbRepository
{
    #region Protected Members

    protected DbContext _dbContext;

    #endregion Protected Members

    #region Constractors

    /// <summary>
    /// Repository constructor
    /// </summary>
    /// <param name="dbContext">Entity framework databse context</param>
    public DbRepository(DbContext dbContext)
    {
        _dbContext = dbContext;

        ConfigureContext();
    }

    #endregion Constractors

    #region IRepository Implementation

    #region Tables and Views functions

    /// <summary>
    /// Query all
    /// Set noTracking to true for selecting only (read-only queries)
    /// Set noTracking to false for insert, update, or delete after select
    /// </summary>
    public virtual IQueryable<TResult> GetAll<TResult>(bool noTracking = true) where TResult :
```

```
class
    {
        var entityDbSet = GetDbSet<TResult>();

        if (noTracking)
            return entityDbSet.AsNoTracking();

        return entityDbSet;
    }

    public virtual TEntity Add<TEntity>(TEntity entity) where TEntity : class
    {
        return GetDbSet<TEntity>().Add(entity);
    }

    /// <summary>
    /// Delete loaded (attached) or unloaded (Detached) entitiy
    /// No need to load object to delete it
    /// Create new object of TEntity and set the id then call Delete function
    /// </summary>
    /// <param name="entity">TEntity</param>
    /// <returns></returns>
    public virtual TEntity Delete<TEntity>(TEntity entity) where TEntity : class
    {
        if (_dbContext.Entry(entity).State == EntityState.Detached)
        {
            _dbContext.Entry(entity).State = EntityState.Deleted;
            return entity;
        }
        else
            return GetDbSet<TEntity>().Remove(entity);
    }

    public virtual TEntity Attach<TEntity>(TEntity entity) where TEntity : class
    {
        return GetDbSet<TEntity>().Attach(entity);
    }

    public virtual TEntity AttachIfNot<TEntity>(TEntity entity) where TEntity : class
    {
        if (_dbContext.Entry(entity).State == EntityState.Detached)
            return Attach(entity);

        return entity;
    }

    #endregion Tables and Views functions

    #region Transactions Functions

    /// <summary>
    /// Saves all changes made in this context to the underlying database.
    /// </summary>
    /// <returns>The number of objects written to the underlying database.</returns>
    public virtual int Commit()
    {
        return _dbContext.SaveChanges();
    }

    /// <summary>
    /// Asynchronously saves all changes made in this context to the underlying database.
```

```
    /// </summary>
    /// <param name="cancellationToken">A System.Threading.CancellationToken to observe while
waiting for the task to complete.</param>
    /// <returns>A task that represents the asynchronous save operation.  The task result
contains the number of objects written to the underlying database.</returns>
    public virtual Task<int> CommitAsync(CancellationToken cancellationToken =
default(CancellationToken))
    {
        return _dbContext.SaveChangesAsync(cancellationToken);
    }

    #endregion Transactions Functions

    #region Database Procedures and Functions

    /// <summary>
    /// Executes any function in the context
    /// use to call database procesdures and functions
    /// </summary>>
    /// <typeparam name="TResult">return function type</typeparam>
    /// <param name="functionName">context function name</param>
    /// <param name="parameters">context function parameters in same order</param>
    public virtual TResult Execute<TResult>(string functionName, params object[] parameters)
    {
        MethodInfo method = _dbContext.GetType().GetMethod(functionName);

        return (TResult)method.Invoke(_dbContext, parameters);
    }

    #endregion Database Procedures and Functions

    #endregion IRepository Implementation

    #region IDisposable Implementation

    public void Dispose()
    {
        _dbContext.Dispose();
    }

    #endregion IDisposable Implementation

    #region Protected Functions

    /// <summary>
    /// Set Context Configuration
    /// </summary>
    protected virtual void ConfigureContext()
    {
        // set your recommended Context Configuration
        _dbContext.Configuration.LazyLoadingEnabled = false;
    }

    #endregion Protected Functions

    #region Private Functions

    private DbSet<TEntity> GetDbSet<TEntity>() where TEntity : class
    {
        return _dbContext.Set<TEntity>();
    }
```

```
    #endregion Private Functions

}
```

2-Entity Framework @ビジネス

このでは、アプリケーションビジネスをします。

プレゼンテーションごとに、になすべてのをむビジネスインターフェイスとクラスをすることをおめします。

では、としてのビジネスをします

```
/// <summary>
/// Contains Product Business functions
/// </summary>
public interface IProductBusiness
{
    Product SelectById(int productId, bool noTracking = true);
    Task<IEnumerable<dynamic>> SelectByCategoryAsync(int CategoryId);
    Task<Product> InsertAsync(string productName, int categoryId);
    Product InsertForNewCategory(string productName, string categoryName);
    Product Update(int productId, string productName, int categoryId);
    Product Update2(int productId, string productName, int categoryId);
    int DeleteWithoutLoad(int productId);
    int DeleteLoadedProduct(Product product);
    IEnumerable<GetProductsCategory_Result> GetProductsCategory(int categoryId);
}



/// <summary>
/// Implementing Product Business functions
/// </summary>
public class ProductBusiness : IProductBusiness
{
    #region Private Members

    private IDbRepository _dbRepository;

    #endregion Private Members

    #region Constructors

    /// <summary>
    /// Product Business Constructor
    /// </summary>
    /// <param name="dbRepository"></param>
    public ProductBusiness(IDbRepository dbRepository)
    {
        _dbRepository = dbRepository;
    }

    #endregion Constructors

    #region IProductBusiness Function
```

```
    /// <summary>
    /// Selects Product By Id
    /// </summary>
    public Product SelectById(int productId, bool noTracking = true)
    {
        var products = _dbRepository.GetAll<Product>(noTracking);

        return products.FirstOrDefault(pro => pro.ProductID == productId);
    }

    /// <summary>
    /// Selects Products By Category Id Async
    /// To have async method, add reference to EntityFramework 6 dll or higher
    /// also you need to have the namespace "System.Data.Entity"
    /// </summary>
    /// <param name="CategoryId">CategoryId</param>
    /// <returns>Return what ever the object that you want to return</returns>
    public async Task<IEnumerable<dynamic>> SelectByCategoryAsync(int CategoryId)
    {
        var products = _dbRepository.GetAll<Product>();
        var categories = _dbRepository.GetAll<Category>();

        var result = (from pro in products
                      join cat in categories
                      on pro.CategoryID equals cat.CategoryID
                      where pro.CategoryID == CategoryId
                      select new
                      {
                          ProductId = pro.ProductID,
                          ProductName = pro.ProductName,
                          CategoryName = cat.CategoryName
                      }
                              );

        return await result.ToListAsync();
    }

    /// <summary>
    /// Insert Async new product for given category
    /// </summary>
    public async Task<Product> InsertAsync(string productName, int categoryId)
    {
        var newProduct = _dbRepository.Add(new Product() { ProductName = productName,
CategoryID = categoryId });

        await _dbRepository.CommitAsync();

        return newProduct;
    }

    /// <summary>
    /// Insert new product and new category
    /// Do many database actions in one transaction
    /// each _dbRepository.Commit(); will commit one transaction
    /// </summary>
    public Product InsertForNewCategory(string productName, string categoryName)
    {
        var newCategory = _dbRepository.Add(new Category() { CategoryName = categoryName });
        var newProduct = _dbRepository.Add(new Product() { ProductName = productName, Category
= newCategory });
```

```
        _dbRepository.Commit();

        return newProduct;
    }

    /// <summary>
    /// Update given product with tracking
    /// </summary>
    public Product Update(int productId, string productName, int categoryId)
    {
        var product = SelectById(productId,false);
        product.CategoryID = categoryId;
        product.ProductName = productName;

        _dbRepository.Commit();

        return product;
    }

    /// <summary>
    /// Update given product with no tracking and attach function
    /// </summary>
    public Product Update2(int productId, string productName, int categoryId)
    {
        var product = SelectById(productId);
        _dbRepository.Attach(product);

        product.CategoryID = categoryId;
        product.ProductName = productName;

        _dbRepository.Commit();

        return product;
    }

    /// <summary>
    /// Deletes product without loading it
    /// </summary>
    public int DeleteWithoutLoad(int productId)
    {
        _dbRepository.Delete(new Product() { ProductID = productId });

        return _dbRepository.Commit();
    }

    /// <summary>
    /// Deletes product after loading it
    /// </summary>
    public int DeleteLoadedProduct(Product product)
    {
        _dbRepository.Delete(product);

        return _dbRepository.Commit();
    }

    /// <summary>
    /// Assuming we have the following procedure in database
    /// PROCEDURE [dbo].[GetProductsCategory] @CategoryID INT, @OrderBy VARCHAR(50)
    /// </summary>
    public IEnumerable<GetProductsCategory_Result> GetProductsCategory(int categoryId)
```

```
    {
        return
_dbRepository.Execute<IEnumerable<GetProductsCategory_Result>>("GetProductsCategory",
categoryId, "ProductName DESC");
    }


    #endregion IProductBusiness Function
}
```

3-ビジネスレイヤ@プレゼンテーションレイヤーMVCの

このでは、プレゼンテーションレイヤのビジネスレイヤをします。また、プレゼンテーションレイヤのとしてMVCをしますのプレゼンテーションレイヤはできます。

にIoCをするがありますUnityをしますが、のIoCをできます。その、プレゼンテーションレイヤー

3-1 MVCにUnityタイプをする

3-1-1「ASP.NET MVCのユニティブートストラップ」をするNuGet backage

3-1-2 UnityWebActivator.Startをします。 Global.asax.csファイルApplication_Start

3-1-3 UnityConfig.RegisterTypesをのようにします。

```
    public static void RegisterTypes(IUnityContainer container)
    {
        // Data Access Layer
        container.RegisterType<DbContext, CompanyContext>(new PerThreadLifetimeManager());
        container.RegisterType(typeof(IDbRepository), typeof(DbRepository), new
PerThreadLifetimeManager());

        // Business Layer
        container.RegisterType<IProductBusiness, ProductBusiness>(new
PerThreadLifetimeManager());

    }
```

3-2ビジネスレイヤ@プレゼンテーションレイヤーMVCの

```
public class ProductController : Controller
{
    #region Private Members

    IProductBusiness _productBusiness;

    #endregion Private Members

    #region Constractors

    public ProductController(IProductBusiness productBusiness)
    {
        _productBusiness = productBusiness;
    }
```

```
    #endregion Constractors

    #region Action Functions

    [HttpPost]
    public ActionResult InsertForNewCategory(string productName, string categoryName)
    {
        try
        {
            // you can use any of IProductBusiness functions
            var newProduct = _productBusiness.InsertForNewCategory(productName, categoryName);

            return Json(new { success = true, data = newProduct });
        }
        catch (Exception ex)
        {   /* log ex*/
            return Json(new { success = false, errorMessage = ex.Message});
        }
    }

    [HttpDelete]
    public ActionResult SmartDeleteWithoutLoad(int productId)
    {
        try
        {
            // deletes product without load
            var deletedProduct = _productBusiness.DeleteWithoutLoad(productId);

            return Json(new { success = true, data = deletedProduct });
        }
        catch (Exception ex)
        {   /* log ex*/
            return Json(new { success = false, errorMessage = ex.Message });
        }
    }

    public async Task<ActionResult> SelectByCategoryAsync(int CategoryId)
    {
        try
        {
            var results = await _productBusiness.SelectByCategoryAsync(CategoryId);

            return Json(new { success = true, data = results },JsonRequestBehavior.AllowGet);
        }
        catch (Exception ex)
        {   /* log ex*/
            return Json(new { success = false, errorMessage = ex.Message
},JsonRequestBehavior.AllowGet);
        }
    }
    #endregion Action Functions
}
```

ユニットテストレイヤ—@エンティティフレームワーク

テストでは、、ビジネスレイヤをテストします。これをうために、データレイヤEntity Frameworkのをします。

ビジネス·レイヤ—·ファンクションのテストをうためにEntity Frameworkのをするにはどうしたら

よいでしょうか

えはですたちはIDbRepositoryインターフェースのためにのをい、テストをうことができます

4-1 インターフェースの

```
class FakeDbRepository : IDbRepository
{
    #region Protected Members

    protected Hashtable _dbContext;
    protected int _numberOfRowsAffected;
    protected Hashtable _contextFunctionsResults;

    #endregion Protected Members

    #region Constractors

    public FakeDbRepository(Hashtable contextFunctionsResults = null)
    {
        _dbContext = new Hashtable();
        _numberOfRowsAffected = 0;
        _contextFunctionsResults = contextFunctionsResults;
    }

    #endregion Constractors

    #region IRepository Implementation

    #region Tables and Views functions

    public IQueryable<TResult> GetAll<TResult>(bool noTracking = true) where TResult : class
    {
        return GetDbSet<TResult>().AsQueryable();
    }

    public TEntity Add<TEntity>(TEntity entity) where TEntity : class
    {
        GetDbSet<TEntity>().Add(entity);
        ++_numberOfRowsAffected;
        return entity;
    }

    public TEntity Delete<TEntity>(TEntity entity) where TEntity : class
    {
        GetDbSet<TEntity>().Remove(entity);
        ++_numberOfRowsAffected;
        return entity;
    }

    public TEntity Attach<TEntity>(TEntity entity) where TEntity : class
    {
        return Add(entity);
    }

    public TEntity AttachIfNot<TEntity>(TEntity entity) where TEntity : class
    {
        if (!GetDbSet<TEntity>().Contains(entity))
            return Attach(entity);
```

```
        return entity;
    }

    #endregion Tables and Views functions

    #region Transactions Functions


    public virtual int Commit()
    {
        var numberOfRowsAffected = _numberOfRowsAffected;
        _numberOfRowsAffected = 0;
        return numberOfRowsAffected;
    }

    public virtual Task<int> CommitAsync(CancellationToken cancellationToken =
default(CancellationToken))
    {
        var numberOfRowsAffected = _numberOfRowsAffected;
        _numberOfRowsAffected = 0;
        return new Task<int>(() => numberOfRowsAffected);
    }

    #endregion Transactions Functions

    #region Database Procedures and Functions

    public virtual TResult Execute<TResult>(string functionName, params object[] parameters)
    {
        if (_contextFunctionsResults != null &&
_contextFunctionsResults.Contains(functionName))
            return (TResult)_contextFunctionsResults[functionName];

        throw new NotImplementedException();
    }

    #endregion Database Procedures and Functions

    #endregion IRepository Implementation

    #region IDisposable Implementation

    public void Dispose()
    {

    }

    #endregion IDisposable Implementation

    #region Private Functions

    private List<TEntity> GetDbSet<TEntity>() where TEntity : class
    {
        if (!_dbContext.Contains(typeof(TEntity)))
            _dbContext.Add(typeof(TEntity), new List<TEntity>());

        return (List<TEntity>)_dbContext[typeof(TEntity)];
    }

    #endregion Private Functions
}
```

4-2ユニットテストの

```
[TestClass]
public class ProductUnitTest
{
    [TestMethod]
    public void TestInsertForNewCategory()
    {
        // Initialize repositories
        FakeDbRepository _dbRepository = new FakeDbRepository();

        // Initialize Business object
        IProductBusiness productBusiness = new ProductBusiness(_dbRepository);

        // Process test method
        productBusiness.InsertForNewCategory("Test Product", "Test Category");

        int _productCount = _dbRepository.GetAll<Product>().Count();
        int _categoryCount = _dbRepository.GetAll<Category>().Count();

        Assert.AreEqual<int>(1, _productCount);
        Assert.AreEqual<int>(1, _categoryCount);
    }

    [TestMethod]
    public void TestProceduresFunctionsCall()
    {
        // Initialize Procedures / Functions result
        Hashtable _contextFunctionsResults = new Hashtable();
        _contextFunctionsResults.Add("GetProductsCategory", new
List<GetProductsCategory_Result> {
            new GetProductsCategory_Result() { ProductName = "Product 1", ProductID = 1,
CategoryName = "Category 1" },
            new GetProductsCategory_Result() { ProductName = "Product 2", ProductID = 2,
CategoryName = "Category 1" },
            new GetProductsCategory_Result() { ProductName = "Product 3", ProductID = 3,
CategoryName = "Category 1" }});

        // Initialize repositories
        FakeDbRepository _dbRepository = new FakeDbRepository(_contextFunctionsResults);

        // Initialize Business object
        IProductBusiness productBusiness = new ProductBusiness(_dbRepository);

        // Process test method
        var results = productBusiness.GetProductsCategory(1);

        Assert.AreEqual<int>(3, results.Count());
    }
}
```

オンラインでエンティティフレームワークのベストプラクティスシンプルプロフェッショナルをむ https://riptutorial.com/ja/entity-framework/topic/8879/エンティティフレームワークのベストプラクティス-シンプル-プロフェッショナル-

12: エンティティの

Entity Frameworkのエンティティは、 System.Data.Entity.EntityStateによってされるさまざまなをつことができます。これらのはのとおりです。

```
Added
Deleted
Detached
Modified
Unchanged
```

Entity FrameworkはPOCOとします。つまり、エンティティは、のをするプロパティとメソッドをたないなクラスです。エンティティのは、 `ObjectStateManager`コンテキストによってされます。

このトピックでは、エンティティのをするさまざまなについてします。

Examples

のエンティティの

`EntityState.Added`は、2つのくじでできます。

1. コンテキストのエントリのをすることによって

   ```
   context.Entry(entity).State = EntityState.Added;
   ```

2. それをコンテキストの`DbSet`にすることによって

   ```
   context.Entities.Add(entity);
   ```

`SaveChanges`びすと、エンティティがデータベースにされます。それは、 このプロパティがすでにをっていたとしても、それがIDセット、インクリメントプライマリキーをっている 、 `SaveChanges` 、エンティティのプライマリキープロパティはしくされたをみます 。

オブジェクトグラフの

オブジェクトグラフ するエンティティののを[`Added`することは、のエンティティを`Added`としてすることとはなります このを。

このでは、とそのをしています

クラスモデル

```
public class Planet
{
```

```
    public Planet()
    {
        Moons = new HashSet<Moon>();
    }
    public int ID { get; set; }
    public string Name { get; set; }
    public ICollection<Moon> Moons { get; set; }
}

public class Moon
{
    public int ID { get; set; }
    public int PlanetID { get; set; }
    public string Name { get; set; }
}
```

コンテキスト

```
public class PlanetDb : DbContext
{
    public property DbSet<Planet> Planets { get; set; }
}
```

こののインスタンスをして、とそのをします。

```
var mars = new Planet { Name = "Mars" };
mars.Moons.Add(new Moon { Name = "Phobos" });
mars.Moons.Add(new Moon { Name = "Deimos" });

context.Planets.Add(mars);

Console.WriteLine(context.Entry(mars).State);
Console.WriteLine(context.Entry(mars.Moons.First()).State);
```

```
Added
Added
```

ここでられるのは、 `Planet`をすることで、のが`Added`されるということです。

エンティティのを`Added`にすると、ナビゲーションプロパティ `Planet.Moons`などののエンティティに「ナビゲート」するプロパティのすべてのエンティティも、にコンテキストにアタッチされていないり、 `Added`としてマークされます 。

オンラインでエンティティのをむ https://riptutorial.com/ja/entity-framework/topic/5256/エンティティの

13: コードののデータアノテーション

Entity Framework Code-Firstは、DataAnnotationのセットをします。このは、ドメインクラスとプロパティにできます。 DataAnnotationは、デフォルトのコード･ファーストをオーバーライドします。

1. **System.ComponentModel.DataAnnotations**には、NULLまたはのサイズにするがまれています。
2. **System.ComponentModel.DataAnnotations.Schema**には、データベースのスキーマにをえるがまれています。

DataAnnotationsは、オプションのサブセットのみをします。 Fluent APIは、Code-Firstでなすべてのオプションをします。

Examples

[Key]

キーは、データベーステーブルの/レコードをにするテーブルのフィールドです。

このをして、デフォルトのコードファーストをきします。プロパティにすると、このクラスのキーとしてされます。

```
using System.ComponentModel.DataAnnotations;

public class Person
{
    [Key]
    public int PersonKey { get; set; }        // <- will be used as primary key

    public string PersonName { get; set; }
}
```

キーがなは、[Key]をのプロパティにすることもできます。キーののは、[キー、**Order = x]**のでするがあります。

```
using System.ComponentModel.DataAnnotations;

public class Person
{
    [Key, Column(Order = 0)]
    public int PersonKey1 { get; set; }    // <- will be used as part of the primary key

    [Key, Column(Order = 1)]
    public int PersonKey2 { get; set; }    // <- will be used as part of the primary key

    public string PersonName { get; set; }
}
```

[Key]がなければ 、EntityFrameworkは "Id"または "{ClassName} Id"というのプライマリキーとしてクラスのプロパティをするデフォルトのにります。

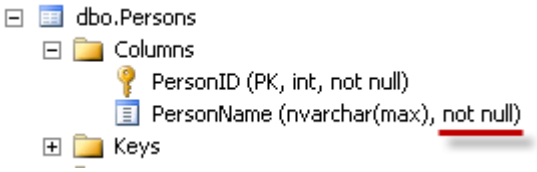
```
public class Person
{
    public int PersonID { get; set; }        // <- will be used as primary key

    public string PersonName { get; set; }
}
```

[]

ドメインクラスのプロパティにすると、データベースはNOT NULLをします。

```
using System.ComponentModel.DataAnnotations;

public class Person
{
    public int PersonID { get; set; }

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

NOT NULLをつのカラム



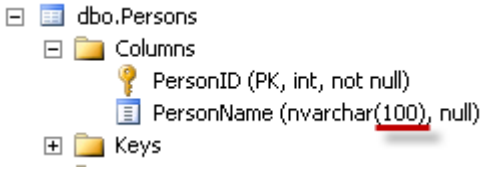
としてasp.net-mvcとともにすることもできます。

[MaxLength]および[MinLength]

[MaxLengthint]は、ドメインクラスのまたはのプロパティにできます。 Entity Frameworkは、のサイズをされたにします。

```
using System.ComponentModel.DataAnnotations;

public class Person
{
    public int PersonID { get; set; }

    [MinLength(3), MaxLength(100)]
    public string PersonName { get; set; }
}
```

されたのさをつの



[MinLengthhint]はですが、データベースにはしません。 PersonNameのさが3のPersonを/しようとすると、このコミットはします。たちがするのある`DbUpdateConcurrencyException`をします。

```
using (var db = new ApplicationDbContext())
{
     db.Staff.Add(new Person() { PersonName = "ng" });
     try
     {
          db.SaveChanges();
     }
     catch (DbEntityValidationException ex)
     {
          //ErrorMessage = "The field PersonName must be a string or array type with a minimum
length of '3'."
     }
}
```

[MaxLength]および**[MinLength]**は、としてasp.net-mvcとともにすることもできます。

[、]

プロパティののとのをします。

```
using System.ComponentModel.DataAnnotations;

public partial class Enrollment
{
    public int EnrollmentID { get; set; }

    [Range(0, 4)]
    public Nullable<decimal> Grade { get; set; }
}
```

Gradeにのを/しようとすると、このコミットはします。たちがするのある`DbUpdateConcurrencyException`をします。

```
using (var db = new ApplicationDbContext())
{
    db.Enrollments.Add(new Enrollment() { Grade = 1000 });

    try
    {
        db.SaveChanges();
    }
    catch (DbEntityValidationException ex)
    {
        // Validation failed for one or more entities
    }
}
```

としてasp.net-mvcとともにすることもできます。

[DatabaseGenerated]

データベースがプロパティのをするをします。なは3つあります。

1. `None`は、がデータベースによってされないことをします。
2. `Identity`は、がIDのであり、はのキーにされることをします。
3. `Computed`は、データベースがのをすることをします。

が`None`の、Entity Frameworkはプロパティにするをデータベースにコミットしません。

デフォルトでは `StoreGeneratedIdentityKeyConvention` づいて、キーのプロパティはIDとしてわれます。このをオーバーライドしてとしてうには、 `DatabaseGenerated`に`None`ます。

```
using System.ComponentModel.DataAnnotations.Schema;

public class Foo
{
    [Key]
    public int Id { get; set; } // identity (auto-increment) column
}

public class Bar
{
    [Key]
    [DatabaseGenerated(DatabaseGeneratedOption.None)]
    public int Id { get; set; } // non-identity column
}
```

のSQLは、カラムをつテーブルをします。

```
CREATE TABLE [Person] (
    Name varchar(100) PRIMARY KEY,
    DateOfBirth Date NOT NULL,
    Age AS DATEDIFF(year, DateOfBirth, GETDATE())
)
GO
```

ののレコードをすエンティティをするには、 `DatabaseGenerated`のを`Computed`するがあります。

```
[Table("Person")]
public class Person
{
    [Key, StringLength(100)]
    public string Name { get; set; }
    public DateTime DateOfBirth { get; set; }
    [DatabaseGenerated(DatabaseGeneratedOption.Computed)]
    public int Age { get; set; }
```

```
}
```

[NotMapped]

コード･ファースト･コンベンションでは、Entity Frameworkは、サポートされているデータであり、ゲッターとセッターのをつすべてのパブリックプロパティのをします。 **[NotMapped]**は、々がデータベーステーブルのをしないのプロパティにするがあります。

たちがデータベースにしたくないプロパティのは、とにづいたのフルネームです。これはオンザフライでることができ、データベースにするはありません。

```
public string FullName => string.Format("{0} {1}", FirstName, LastName);
```

"FullName"プロパティにはgetterとsetterがありません 。したがって、では、Entity Frameworkはデフォルトのをしません 。

たちがデータベースにしたくないプロパティののは、の "AverageGrade"です。オンデマンドでAverageGradeをすることはましくありません。そのわりに、それをするルーチンがあるかもしれません。

```
[NotMapped]
public float AverageGrade { set; get; }
```

"AverageGrade"に**[NotMapped]**をけるがあります。そうしないと、Entity Frameworkによってがされます。

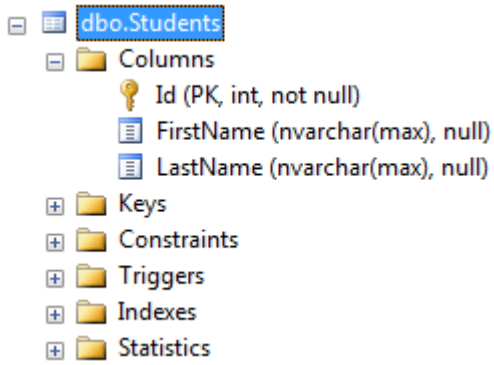
```
using System.ComponentModel.DataAnnotations.Schema;

public class Student
{
    public int Id { set; get; }

    public string FirstName { set; get; }

    public string LastName { set; get; }

    public string FullName => string.Format("{0} {1}", FirstName, LastName);

    [NotMapped]
    public float AverageGrade { set; get; }
}
```

のエンティティについては、 `DbMigration.cs`にされ`DbMigration.cs`

```
CreateTable(
    "dbo.Students",
     c => new
         {
             Id = c.Int(nullable: false, identity: true),
             FirstName = c.String(),
             LastName = c.String(),
```

```
        })
    .PrimaryKey(t => t.Id);
```

およびSQL Server Management Studio



[]

```
[Table("People")]
public class Person
{
    public int PersonID { get; set; }
    public string PersonName { get; set; }
}
```

Entity Frameworkに、のテーブルをするようにしますつまり、 `Person`または`Persons`

[Table]をしてテーブルのスキーマをすることもできます

```
[Table("People", Schema = "domain")]
```

[]

```
public class Person
{
    public int PersonID { get; set; }

    [Column("NameOfPerson")]
    public string PersonName { get; set; }
}
```

Entity Frameworkに、プロパティのをするわりに、のをするようにします。また、データベースのデータとのをでることもできます。

```
[Column("NameOfPerson", TypeName = "varchar", Order = 1)]
public string PersonName { get; set; }
```

[Index]

```
public class Person
```

```
{
    public int PersonID { get; set; }
    public string PersonName { get; set; }

    [Index]
    public int Age { get; set; }
}
```

またはのセットのデ―タベ―スインデックスをします。

```
[Index("IX_Person_Age")]
public int Age { get; set; }
```

これにより、ののがされます。

```
[Index(IsUnique = true)]
public int Age { get; set; }
```

これによりのインデックスがされます。

```
[Index("IX_Person_NameAndAge", 1)]
public int Age { get; set; }

[Index("IX_Person_NameAndAge", 2)]
public string PersonName { get; set; }
```

これにより、2つのをしてインデックスがされます。これをうには、じインデックスをし、のをするがあります。

Indexは、Entity Framework 6.1でされました。のバ―ジョンをしている、このセクションのはされません。

[ForeignKeystring]

EFのにわないキ―がなは、カスタムキ―をします。

```
public class Person
{
    public int IdAddress { get; set; }

    [ForeignKey(nameof(IdAddress))]
    public virtual Address HomeAddress { get; set; }
}
```

これは、じエンティティタイプにしてのがあるにもできます。

```
using System.ComponentModel.DataAnnotations.Schema;

public class Customer
{
    ...
```

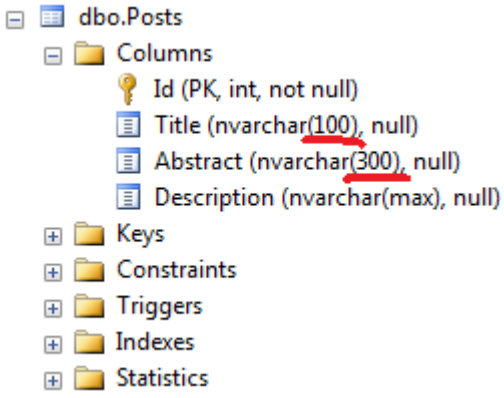
```
    public int MailingAddressID { get; set; }
    public int BillingAddressID { get; set; }

    [ForeignKey("MailingAddressID")]
    public virtual Address MailingAddress { get; set; }
    [ForeignKey("BillingAddressID")]
    public virtual Address BillingAddress { get; set; }
}
```

なしで`ForeignKey`、EFはそれらるかもしれないとのを`BillingAddressID`フエッチする`MailingAddress`、またはそれだけでのような、のにづいて、にのをいつくかもしれません`Address_MailingAddress_Id`とすることをするようにしてくださいわりにのデータベースでこれをしているはエラーになります。

[StringLengthint]

```
using System.ComponentModel.DataAnnotations;

public class Post
{
    public int Id { get; set; }

    [StringLength(100)]
    public string Title { get; set;}

    [StringLength(300)]
    public string Abstract { get; set; }

    public string Description { get; set; }
}
```

フィールドのをします。

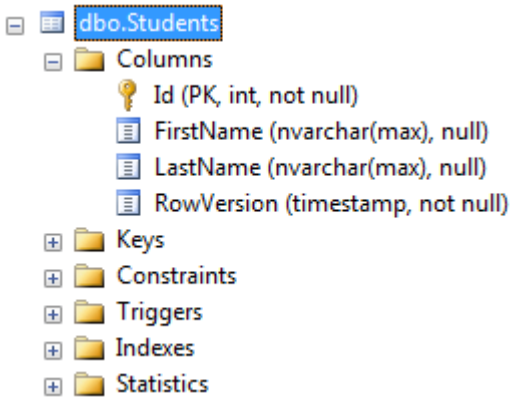


としてasp.net-mvcとともにすることもできます。

[タイムスタンプ]

[TimeStamp]は、されたEntityクラスの1バイトのプロパティにのみできます。 Entity Frameworkは、そのプロパティのデータベーステーブルにnullをしないタイムスタンプをします。 Entity FrameworkはにこのTimeStampをチェックでします。

```
using System.ComponentModel.DataAnnotations.Schema;

public class Student
{
    public int Id { set; get; }

    public string FirstName { set; get; }

    public string LastName { set; get; }

    [Timestamp]
    public byte[] RowVersion { get; set; }
}
```



[ConcurrencyCheck]

このはclassプロパティにされます。 ConcurrencyCheckは、のをにし、のために々のタイムスタンプをしないにできます。

```
using System.ComponentModel.DataAnnotations;

public class Author
{
    public int AuthorId { get; set; }

    [ConcurrencyCheck]
    public string AuthorName { get; set; }
}
```

のから、ConcurrencyCheckはAuthorクラスのAuthorNameプロパティにされます。したがって、コーード･ファーストは、なをチェックするために、コマンドWhereにAuthorNameをめます。

[InversePropertystring]

```
using System.ComponentModel.DataAnnotations.Schema;

public class Department
{
    ...

    public virtual ICollection<Employee> PrimaryEmployees { get; set; }
    public virtual ICollection<Employee> SecondaryEmployees { get; set; }
```

```
}

public class Employee
{
    ...

    [InverseProperty("PrimaryEmployees")]
    public virtual Department PrimaryDepartment { get; set; }

    [InverseProperty("SecondaryEmployees")]
    public virtual Department SecondaryDepartment { get; set; }
}
```

InversePropertyは、2つのエンティティにの がするに、2 つのをするためにできます。

これはEntity Frameworkに、ナビゲーションプロパティがのプロパティとするがあることをします。

エンティティフレームワークは、2つのエンティティにのがする、どのナビゲーションプロパティがののどのプロパティにマップするかをしません。

するクラスのするナビゲーションプロパティのがそのパラメータとしてです。

これは、じタイプののエンティティとのをつエンティティにしてもでき、をします。

```
using System.ComponentModel.DataAnnotations;
using System.ComponentModel.DataAnnotations.Schema;

public class TreeNode
{
    [Key]
    public int ID { get; set; }
    public int ParentID { get; set; }

    ...

    [ForeignKey("ParentID")]
    public TreeNode ParentNode { get; set; }
    [InverseProperty("ParentNode")]
    public virtual ICollection<TreeNode> ChildNodes { get; set; }
}
```

`ForeignKey`をして、のキーにされるをすることにもしてください。のでは、 `Employee`クラスの2つのプロパティに、をするために`ForeignKey`がされているがあります。

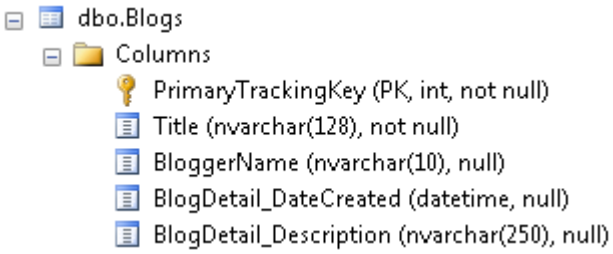
[]

```
using System.ComponentModel.DataAnnotations.Schema;

[ComplexType]
public class BlogDetails
{
    public DateTime? DateCreated { get; set; }
```

```
    [MaxLength(250)]
    public string Description { get; set; }
}

public class Blog
{
    ...

    public BlogDetails BlogDetail { get; set; }
}
```

Entity Frameworkでクラスをとしてマークします。

なタイプまたはドメインにおけるバリュオブジェクト は、ではできませんが、エンティティのとしてされます。これは、のBlogDetailsにキープロパティがないです。



これらは、のクラスにわたってドメインエンティティをし、それらのクラスをなエンティティにするときにちます。

オンラインでコードののデータアノテーションをむ https://riptutorial.com/ja/entity-framework/topic/4161/コードののデータアノテーション

14: コーののの

Conventionは、Code-Firstでするときに、ドメインクラスについてモデルをにするためののデフォルトです。コード - のは、 *System.Data.Entity.ModelConfiguration.Conventions* EF 5およびEF 6 でされています。

Examples

な

クラスのプロパティのが "ID"とはされませんまたはクラスのに "ID"とされている、プロパティはキーです。キーのプロパティのがまたはGUIDの、IDとしてされます。な

```
public class Room
{
    // Primary key
    public int RoomId{ get; set; }
    ...
}
```

の

`OnModelCreating`メソッドをオーバーライドすることによって、System.Data.Entity.ModelConfiguration.Conventionsでされているをできます。

のでは、PluralizingTableNameConventionをします。

```
public class EshopContext : DbContext
{
    public DbSet<Product> Products { set; get; }
    . . .

    protected override void OnModelCreating(DbModelBuilder modelBuilder)
    {
        modelBuilder.Conventions.Remove<PluralizingTableNameConvention>();
    }
}
```

デフォルトでは、EFはエンティティクラスに 's'をつけたDBテーブルをします。このでは、コードファーストはわりに、これPluralizingTableNameをするようにされて`dbo.Products`テーブル`dbo.Product`テーブルがされます。

タイプディスカバリー

デフォルトでコードファーストはモデルにまれます

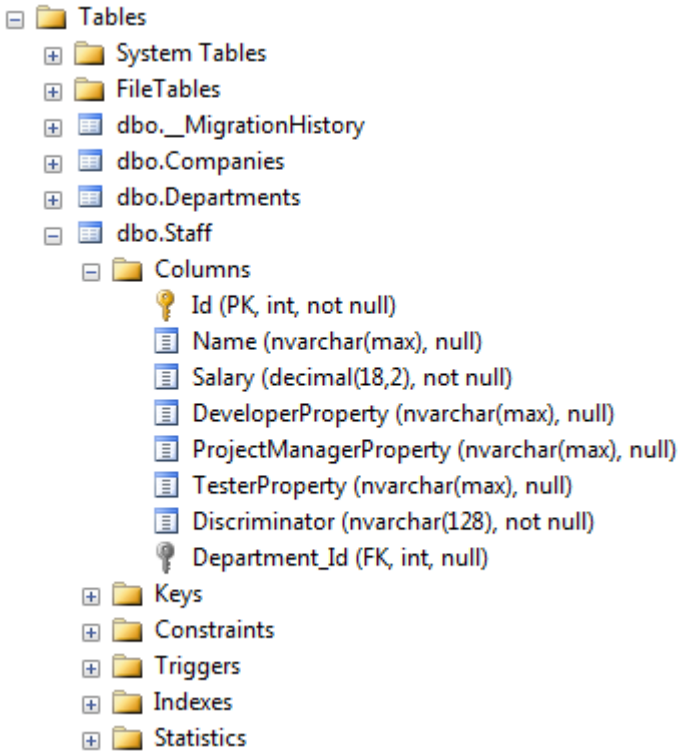
1. コンテキストクラスのDbSetプロパティとしてされた。
2.

なるアセンブリでされているでも、エンティティタイプにまれる。
3. クラスのみがDbSetプロパティとしてされているでもクラス

ここでのでは、々だけしていること、である`Company`として`DbSet<Company>`のコンテキストクラスに

```
public class Company
{
    public int Id { set; get; }
    public string Name { set; get; }
    public virtual ICollection<Department> Departments { set; get; }
}

public class Department
{
    public int Id { set; get; }
    public string Name { set; get; }
    public virtual ICollection<Person> Staff { set; get; }
}

[Table("Staff")]
public class Person
{
    public int Id { set; get; }
    public string Name { set; get; }
    public decimal Salary { set; get; }
}

public class ProjectManager : Person
{
   public string ProjectManagerProperty { set; get; }
}

public class Developer : Person
{
    public string DeveloperProperty { set; get; }
}

public class Tester : Person
{
    public string TesterProperty { set; get; }
}

public class ApplicationDbContext : DbContext
{
    public DbSet<Company> Companies { set; get; }
}
```

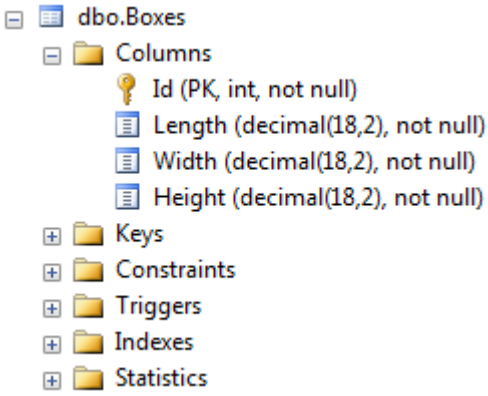
すべてのクラスがモデルにまれていることがわかります



DecimalPropertyConvention

デフォルトでは、Entity Frameworkはのプロパティをデータベーステーブルの1018,2にマップします。

```
public class Box
{
    public int Id { set; get; }
    public decimal Length { set; get; }
    public decimal Width { set; get; }
    public decimal Height { set; get; }
}
```

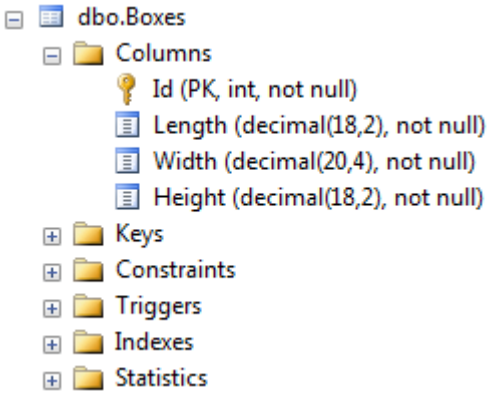


のプロパティのをすることができます。

1.なAPIをする

```
protected override void OnModelCreating(DbModelBuilder modelBuilder)
{
    modelBuilder.Entity<Box>().Property(b => b.Width).HasPrecision(20, 4);
```

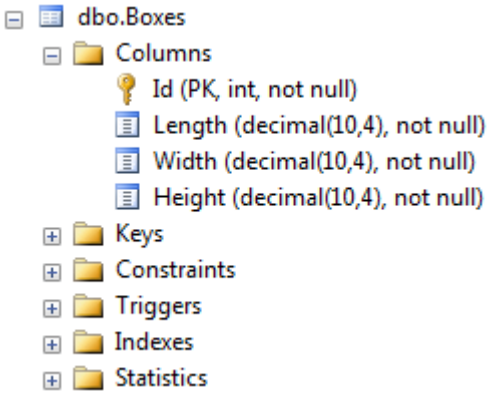
```
}
```



「」プロパティのみが20,4にマッピングされます。

2.をきえる

```
protected override void OnModelCreating(DbModelBuilder modelBuilder)
{
    modelBuilder.Conventions.Remove<DecimalPropertyConvention>();
    modelBuilder.Conventions.Add(new DecimalPropertyConvention(10, 4));
}
```



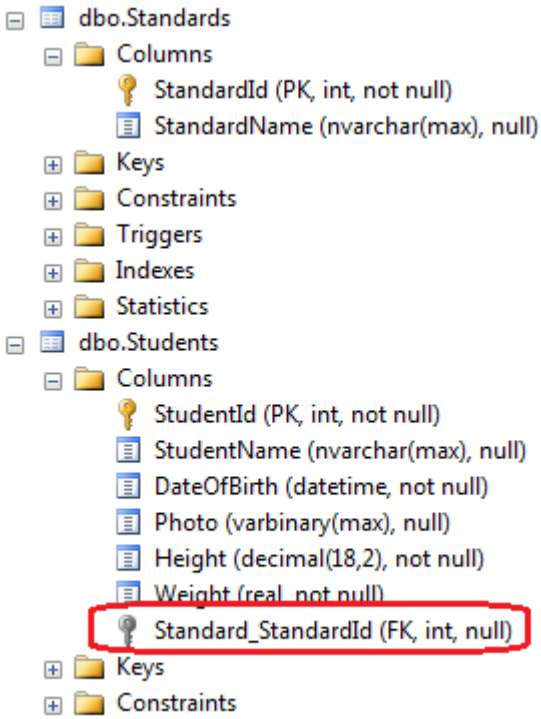
すべての10のプロパティは1010,4のにマップされます。

コードナビゲーションプロパティをして、2つのエンティティのをにします。このナビゲーションプロパティは、なまたはコレクションにすることができます。えば、StudentクラスのStandardナビゲーションプロパティとStandardクラスのICollectionナビゲーションプロパティをしました。したがって、Code Firstは、Standard_StandardIdキーをStudentsテーブルにすることにより、StandardsとStudents DBテーブルのにのをにしました。

```
public class Student
{

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

    //Navigation property
    public Standard Standard { get; set; }
```

```
}

public class Standard
{

    public int StandardId { get; set; }
    public string StandardName { get; set; }

    //Collection navigation property
    public IList<Student> Students { get; set; }

}
```

のエンティティは、Standard_StandardIdキーをしてのをしました。



キー

クラスAがクラスBとのにあり、クラスBがAのキーとじとタイプのプロパティをつ、EFはにそのプロパティをキーとみなします。

```
public class Department
{
    public int DepartmentId { set; get; }
    public string Name { set; get; }
    public virtual ICollection<Person> Staff { set; get; }
}

public class Person
{
    public int Id { set; get; }
    public string Name { set; get; }
    public decimal Salary { set; get; }
    public int DepartmentId { set; get; }
```

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

この、DepartmentIdはながないキーです。

オンラインでコードののをむ https://riptutorial.com/ja/entity-framework/topic/2447/コードのの

15: コードファースト - なAPI

HOW Entity FrameworkがPOCOクラスをデータベーステーブル、などにマップすることをするなは2つあります。 データと**Fluent API** 。

データはみやすく、しやすいものですが、エンティティの「にカスケード」をするなどのはありません。、Fluent APIはもうしですが、はるかになをします。

Examples

モデルのマッピング

EntityFramewok Fluent APIは、 コードファーストドメインモデルをになるデータベースにマッピングする、かつエレガントなです。これはのデータベースでコードファーストですることもできます 。 *Fluent API*をするは、 *OnModelCreating*メソッドでモデルをマップするか、*EntityTypeConfiguration*からしたマッパークラスをして、 *OnModelCreating*メソッドの*modelBuilder*にそのモデルをすることができます。 のはがむもので、そのをすつもりです。

ステップ1モデルをします。

```
public class Employee
{
    public int Id { get; set; }
    public string Surname { get; set; }
    public string FirstName { get; set; }
    public string LastName { get; set; }
    public short Age { get; set; }
    public decimal MonthlySalary { get; set; }

    public string FullName
    {
        get
        {
            return $"{Surname} {FirstName} {LastName}";
        }
    }
}
```

ステップ2マッパークラスをする

```
public class EmployeeMap
    : EntityTypeConfiguration<Employee>
{
    public EmployeeMap()
    {
        // Primary key
        this.HasKey(m => m.Id);
```

```
        this.Property(m => m.Id)
            .HasColumnType("int")
            .HasDatabaseGeneratedOption(DatabaseGeneratedOption.Identity);

        // Properties
        this.Property(m => m.Surname)
            .HasMaxLength(50);

        this.Property(m => m.FirstName)
            .IsRequired()
            .HasMaxLength(50);

        this.Property(m => m.LastName)
            .HasMaxLength(50);

        this.Property(m => m.Age)
            .HasColumnType("smallint");

        this.Property(m => m.MonthlySalary)
            .HasColumnType("number")
            .HasPrecision(14, 5);

        this.Ignore(m => m.FullName);

        // Table & column mappings
        this.ToTable("TABLE_NAME", "SCHEMA_NAME");
        this.Property(m => m.Id).HasColumnName("ID");
        this.Property(m => m.Surname).HasColumnName("SURNAME");
        this.Property(m => m.FirstName).HasColumnName("FIRST_NAME");
        this.Property(m => m.LastName).HasColumnName("LAST_NAME");
        this.Property(m => m.Age).HasColumnName("AGE");
        this.Property(m => m.MonthlySalary).HasColumnName("MONTHLY_SALARY");
    }
}
```

マッピングについてしましょう

- **HasKey** - キ―をします。 コンポジットのキ―もできます 。たとえば *this.HasKeym => new {m.DepartmentId*、m.PositionId} 。
- プロパティ - モデルのプロパティをできます。
- **HasColumnType** - デ―タベ―スレベルのタイプをします。 OracleとMS SQLのようななるデ―タベ―スではなることがあります。
- **HasDatabaseGeneratedOption** - プロパティがデ―タベ―スレベルでされるかどうかをします。のPKは*DatabaseGeneratedOption.Identity*です。デフォルトでは、そうしたくないは*DatabaseGeneratedOption.None*をするがあります。
- **HasMaxLength** - のさをします。
- **IsRequired** - プロパティをクエリとしてマ―クします。
- **HasPrecision** - ののをできます。
- **Ignore** - プロパティをにし、デ―タベ―スにマップしません。 FullNameはされました。なぜなら、このをテ―ブルにれたくないからです。
- **ToTable** - モデルのテ―ブルとスキ―マオプションをします。
- **HasColumnName** - プロパティをにけます。プロパティ―とがである、これはありません。

ステップ3マッピングクラスをにします。

マッパークラスをするようにEntityFrameworkにするがあります。これをうには、*OnModelCreating*メソッドの*modelBuilder.Configurations*にするがあります。

```
public class DbContext()
    : base("Name=DbContext")
{
    protected override void OnModelCreating(DbModelBuilder modelBuilder)
    {
        modelBuilder.Configurations.Add(new EmployeeMap());
    }
}
```

そしてそれはそれです。たちはすべてくつもりです。

キー

.HasKeyメソッドをすると、プロパティをエンティティのキーとしてにできます。

```
using System.Data.Entity;
// ..

public class PersonContext : DbContext
{
    // ..

    protected override void OnModelCreating(DbModelBuilder modelBuilder)
    {
        // ..

        modelBuilder.Entity<Person>().HasKey(p => p.PersonKey);
    }
}
```

キー

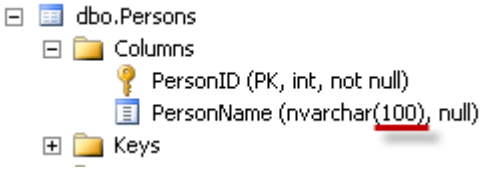
.HasKeyメソッドをすると、のプロパティをエンティティのキーとしてにできます。

```
using System.Data.Entity;
// ..

public class PersonContext : DbContext
{
    // ..

    protected override void OnModelCreating(DbModelBuilder modelBuilder)
    {
        // ..

        modelBuilder.Entity<Person>().HasKey(p => new { p.FirstName, p.LastName });
    }
}
```

.HasMaxLengthメソッドをすると、プロパティのをできます。

```
using System.Data.Entity;
// ..

public class PersonContext : DbContext
{
    // ..

    protected override void OnModelCreating(DbModelBuilder modelBuilder)
    {
        // ..

        modelBuilder.Entity<Person>()
                    .Property(t => t.Name)
                    .HasMaxLength(100);
    }
}
```

されたのさをつの

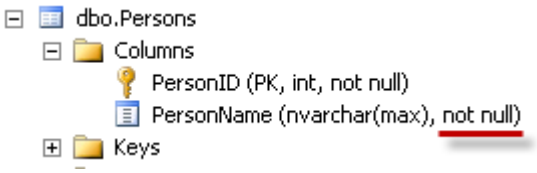


プロパティ—NOT NULL

.IsRequiredメソッドをすると、プロパティをとしてできます。つまり、にはNOT NULLがあります。

```
using System.Data.Entity;
// ..

public class PersonContext : DbContext
{
    // ..

    protected override void OnModelCreating(DbModelBuilder modelBuilder)
    {
        // ..

        modelBuilder.Entity<Person>()
                    .Property(t => t.Name)
                    .IsRequired();
    }
}
```

NOT NULLをつのカラム



キーのをする

モデルにナビゲーションプロパティがする、Entity Frameworkはにキーをします。のキーがですが、モデルにプロパティとしてまれていないは、Fluent APIをしてにできます。キーをしているときに`Map`メソッドをすることにより、キーにのをできます。

```
public class Company
{
    public int Id { get; set; }
}

public class Employee
{
    property int Id { get; set; }
    property Company Employer { get; set; }
}

public class EmployeeContext : DbContext
{
    protected override void OnModelCreating(DbModelBuilder modelBuilder)
    {
        modelBuilder.Entity<Employee>()
                    .HasRequired(x => x.Employer)
                    .WithRequiredDependent()
                    .Map(m => m.MapKey("EmployerId"));
    }
}
```

をした、`Map`メソッドをすると、`MapKey`をしてキーをに`MapKey`ます。このでは、Employer_IdのになったのはではEmployerIdです。

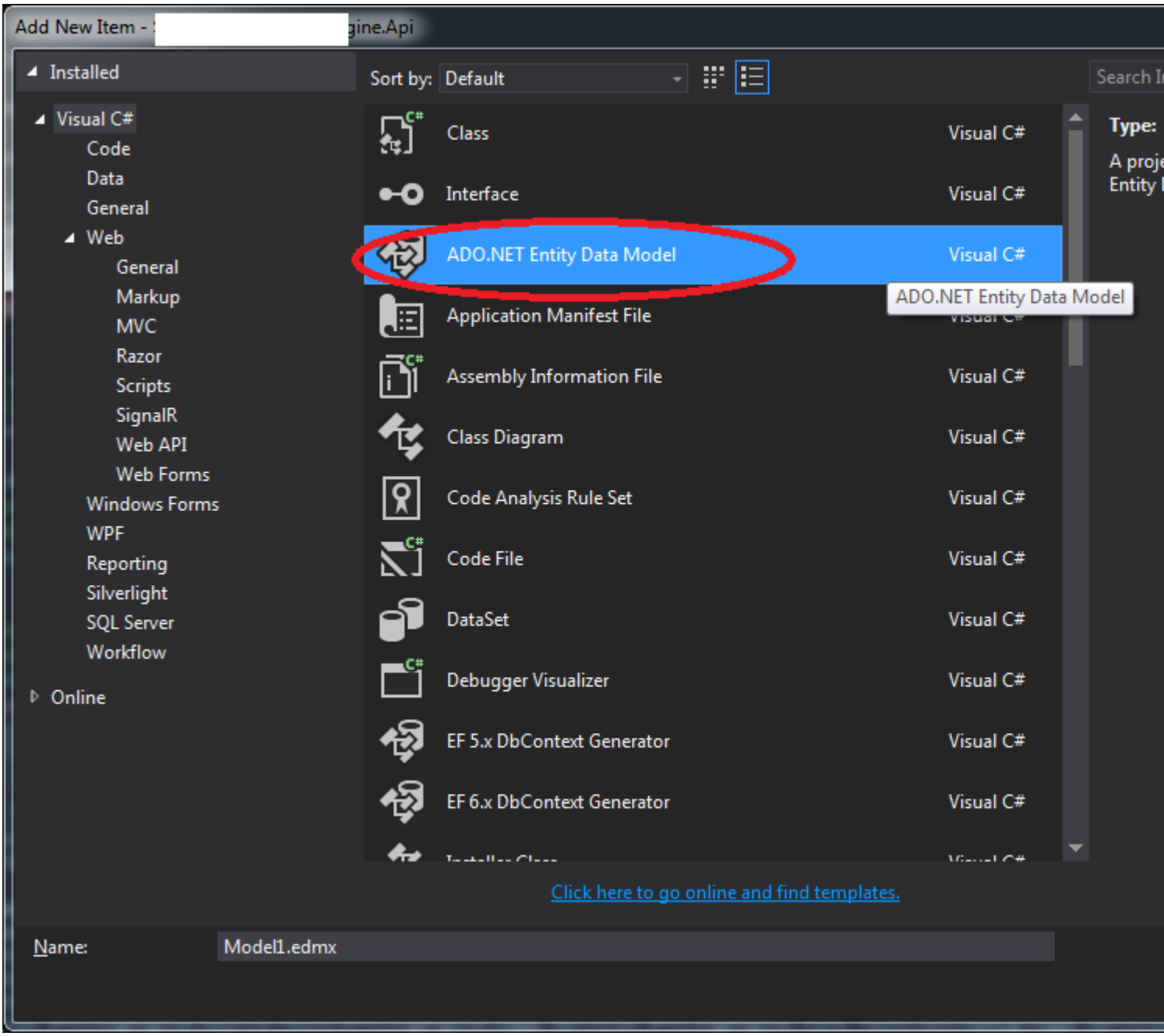
オンラインでコードファースト - なAPIをむ https://riptutorial.com/ja/entity-framework/topic/4530/コードファースト---なapi

16: データベースののモデル

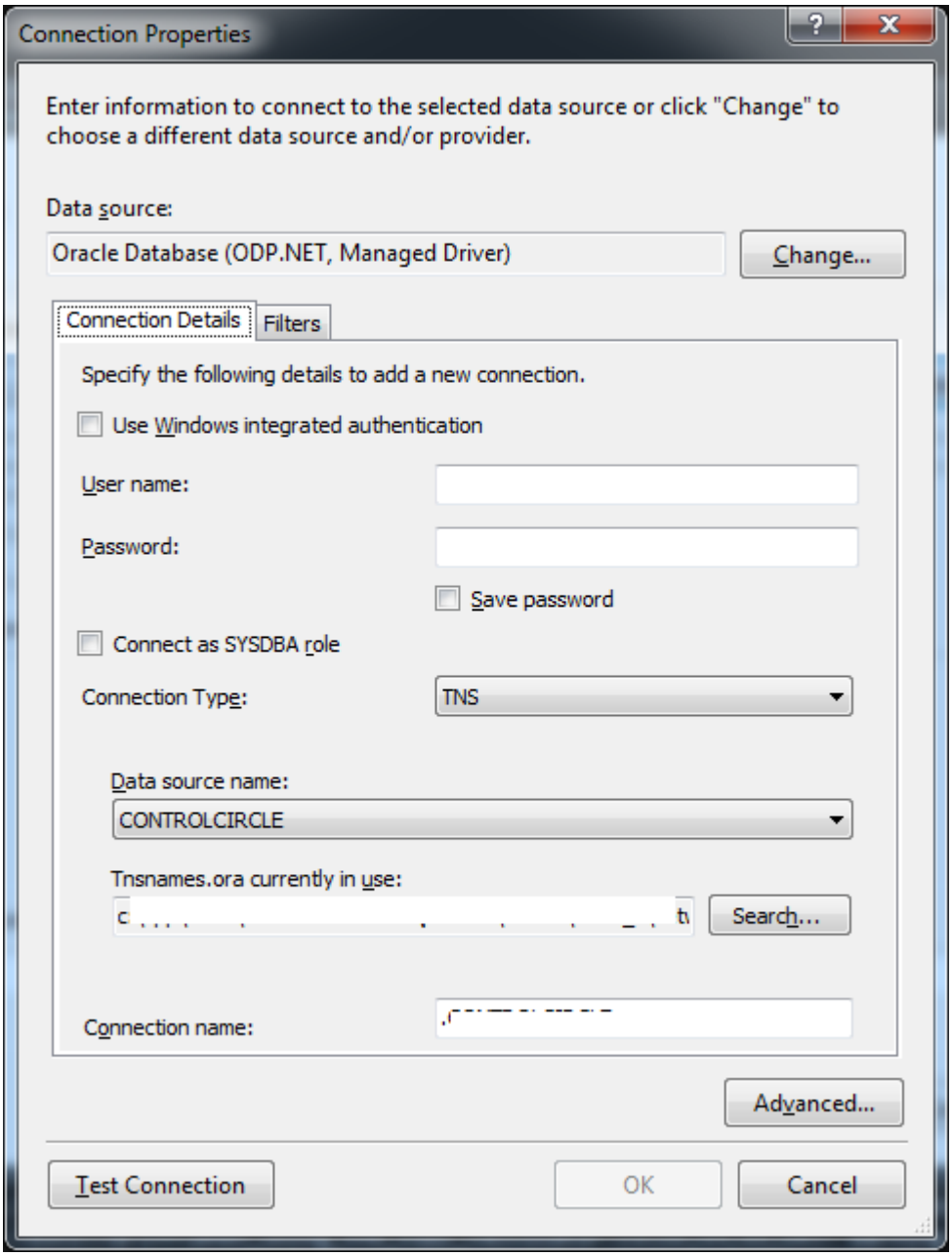
Examples

データベースからモデルをする

`Visual Studio Solution Explorer`にし、 `Project`をクリックしてモデルをします。 マウス。 `ADO.NET Entity Data Model`する



に、した`Generate from database`し、クリックし`Next`クリックし、のウィンドウに`New Connection...`あなたはだろうからモデルをするデータベースへとポイント`MSSQL`、 `MySQL`か、 `Oracle`



これをった、 `Test Connection`クリックしてがしくされているかどうかをしますここでしたは、これみません。

[`Next`をクリックし、なオプションエンティティをするスタイルやキーをするスタイルなどをします。

もう[`Next`クリックします。こので、データベースからモデルをするがあります。

されたモデルにデータをする

Entity Framework 5でされるT4コードでは、データはデフォルトではまれていません。のプロパティのにすべてのモデル、オープンテンプレートファイルはとEDMXにまれるデータをめるには `.tt`、その`using`のを`UsingDirectives`のように

```
foreach (var entity in typeMapper.GetItemsToGenerate<EntityType>
(itemCollection))
```

```
{
    fileManager.StartNewFile(entity.Name + ".cs");
    BeginNamespace(code);
#>
<#=codeStringGenerator.UsingDirectives(inHeader: false)#>
using System.ComponentModel.DataAnnotations;  // --> add this line
```

たとえば、キーのプロパティをす`KeyAttribute`がテンプレートにま`KeyAttribute`ているとします。モデルのに`KeyAttribute`にするには、のように`codeStringGenerator.Property`をむコードのをします。

```
var simpleProperties = typeMapper.GetSimpleProperties(entity);
    if (simpleProperties.Any())
    {
        foreach (var edmProperty in simpleProperties)
        {
#>
    <#=codeStringGenerator.Property(edmProperty)#>
<#
        }
    }
```

に、if-conditionをしてキーのプロパティをします。

```
var simpleProperties = typeMapper.GetSimpleProperties(entity);
    if (simpleProperties.Any())
    {
        foreach (var edmProperty in simpleProperties)
        {
             if (ef.IsKey(edmProperty)) {
#>    [Key]
<#      }
#>
    <#=codeStringGenerator.Property(edmProperty)#>
<#
        }
    }
```

のをすると、されたすべてのモデルクラスは、データベースからモデルをしたに、キープロパティに`KeyAttribute`をちます。

```
using System;

public class Example
{
    public int Id { get; set; }
    public string Name { get; set; }
}
```

```
using System;
using System.ComponentModel.DataAnnotations;

public class Example
{
    [Key]
```

```
    public int Id { get; set; }
    public string Name { get; set; }
}
```

オンラインでデータベースののモデルをむ https://riptutorial.com/ja/entity-framework/topic/4414/データベースののモデル

17: データベース

Examples

CreateDatabaseIfNotExists

`IDatabaseInitializer`デフォルトでEntityFrameworkでされています。がすように、しないはデータベースをします。ただし、モデルをするとがスローされます。

```
public class MyContext : DbContext {
    public MyContext() {
        Database.SetInitializer(new CreateDatabaseIfNotExists<MyContext>());
    }
}
```

DropCreateDatabaseIfModelChanges

この`IDatabaseInitializer`は、モデルがにされたにデータベースをしてします。

```
public class MyContext : DbContext {
    public MyContext() {
        Database.SetInitializer(new DropCreateDatabaseIfModelChanges<MyContext>());
    }
}
```

DropCreateDatabaseAlways

この`IDatabaseInitializer`は、アプリケーションappドメインでコンテキストがされる`IDatabaseInitializer`データベースをしてします。データベースがされているというによるデータのにしてください。

```
public class MyContext : DbContext {
    public MyContext() {
        Database.SetInitializer(new DropCreateDatabaseAlways<MyContext>());
    }
}
```

カスタムデータベース

`IDatabaseInitializer`のをできます。

イニシャライザの。データベースを0にし、のにしますテストをするなどにです。これをうには、`DbMigrationsConfiguration`もです。

```
public class RecreateFromScratch<TContext, TMigrationsConfiguration> :
IDatabaseInitializer<TContext>
where TContext : DbContext
```

```
where TMigrationsConfiguration : DbMigrationsConfiguration<TContext>, new()
{
    private readonly DbMigrationsConfiguration<TContext> _configuration;

    public RecreateFromScratch()
    {
        _configuration = new TMigrationsConfiguration();
    }

    public void InitializeDatabase(TContext context)
    {
        var migrator = new DbMigrator(_configuration);
        migrator.Update("0");
        migrator.Update();
    }
}
```

MigrateDatabaseToLatestVersion

コードファーストマイグレーションをしてデータベースをバージョンにする`IDatabaseInitializer`の。このイニシャライザをするには、 `DbMigrationsConfiguration`もするがあります。

```
public class MyContext : DbContext {
    public MyContext() {
        Database.SetInitializer(
            new MigrateDatabaseToLatestVersion<MyContext, Configuration>());
    }
}
```

オンラインでデータベースをむ https://riptutorial.com/ja/entity-framework/topic/5526/データベース

18: トラッキングとノントラッキング

ビヘイビアは、Entity Frameworkがエンティティインスタンスにするをトラッカーにするかどうかをします。エンティティがされる、エンティティでされたはすべて`SaveChanges()`にデータベースに`SaveChanges()`ます。

Examples

トラッキングクエリ

- デフォルトでは、エンティティタイプをすクエリはしています
- つまり、これらのエンティティインスタンスにをえ、 `SaveChanges()`によってそれらのをすることができます。

- `book`へのは、 `SaveChanges()`にされ、データベースに`SaveChanges()`ます。

```
using (var context = new BookContext())
{
  var book = context.Books.FirstOrDefault(b => b.BookId == 1);
  book.Rating = 5;
  context.SaveChanges();
}
```

トラッキングのないクエリ

- が`read-only`シナリオでされる、トラッキングクエリはにちません
- のをするがないため`quicker to execute`が`quicker to execute`

```
using (var context = new BookContext())
{
    var books = context.Books.AsNoTracking().ToList();
}
```

EF Core 1.0では、 `context instance`レベルでデフォルトのトラッキングをすることもできます。

```
using (var context = new BookContext())
{
    context.ChangeTracker.QueryTrackingBehavior = QueryTrackingBehavior.NoTracking;

    var books = context.Books.ToList();
}
```

トラッキングと

- クエリののがエンティティでなくても、`contains entitycontains entity`でも`tracked by default`はそれがきき`tracked by default`

- anonymous typeをすのクエリでは、セットのBookのインスタンスwill be tracked

```
using (var context = new BookContext())
{
   var book = context.Books.Select(b => new { Book = b, Authors = b.Authors.Count() });
}
```

- セットにentityタイプがまれてdoes not、 no trackingはされdoes not
- のクエリでは、エンティティからののをつanonymous typeがされますがのentityタイプのno instancesはありません 、 はされません 。

```
using (var context = new BookContext())
{
   var book = context.Books.Select(b => new { Id = b.BookId, PublishedDate = b.Date });
}
```

オンラインでトラッキングとノントラッキングをむ https://riptutorial.com/ja/entity-framework/topic/6836/トラッキングとノントラッキング

19: トランザクション

Examples

Database.BeginTransaction

1つのトランザクションにしてのをできるため、いずれかのがしたにをロールバックできます。

```
using (var context = new PlanetContext())
{
    using (var transaction = context.Database.BeginTransaction())
    {
        try
        {
            //Lets assume this works
            var jupiter = new Planet { Name = "Jupiter" };
            context.Planets.Add(jupiter);
            context.SaveChanges();

            //And then this will throw an exception
            var neptune = new Planet { Name = "Neptune" };
            context.Planets.Add(neptune);
            context.SaveChanges();

            //Without this line, no changes will get applied to the database
            transaction.Commit();
        }
        catch (Exception ex)
        {
            //There is no need to call transaction.Rollback() here as the transaction object
            //will go out of scope and disposing will roll back automatically
        }
    }
}
```

コードをよりにするので、 `transaction.Rollback()` にびすことはのになることにしてください。また、されていませんそこにEntity Frameworkのためのあまりられて、クエリプロバイダがあるかもしれ `Dipsose` もにとなるに、 `transaction.Rollback()` のびしを。

オンラインでトランザクションをむ https://riptutorial.com/ja/entity-framework/topic/4944/トランザクション

20: モデルの

Examples

1の

UserTypeはくのユーザにします。<->ユーザは**1**つの**UserType**をちます。

なのナビゲーションプロパティ

```
public class UserType
{
    public int UserTypeId {get; set;}
}
public class User
{
    public int UserId {get; set;}
    public int UserTypeId {get; set;}
    public virtual UserType UserType {get; set;}
}

Entity<User>().HasRequired(u => u.UserType).WithMany().HasForeignKey(u => u.UserTypeId);
```

オプションのキーをつナビゲーションプロパティ `Nullable`でなければならない

```
public class UserType
{
    public int UserTypeId {get; set;}
}
public class User
{
    public int UserId {get; set;}
    public int? UserTypeId {get; set;}
    public virtual UserType UserType {get; set;}
}

Entity<User>().HasOptional(u => u.UserType).WithMany().HasForeignKey(u => u.UserTypeId);
```

ナビゲーションプロパティにじて/キープロパティをにじてする

```
public class UserType
{
    public int UserTypeId {get; set;}
    public virtual ICollection<User> Users {get; set;}
}
public class User
{
    public int UserId {get; set;}
    public int UserTypeId {get; set;}
    public virtual UserType UserType {get; set;}
}
```

```
Entity<User>().HasRequired(u => u.UserType).WithMany(ut => ut.Users).HasForeignKey(u =>
u.UserTypeId);
```

オプション

```
Entity<User>().HasOptional(u => u.UserType).WithMany(ut => ut.Users).HasForeignKey(u =>
u.UserTypeId);
```

オンラインでモデルのをむ https://riptutorial.com/ja/entity-framework/topic/4528/モデルの

21:

Examples

コード1の

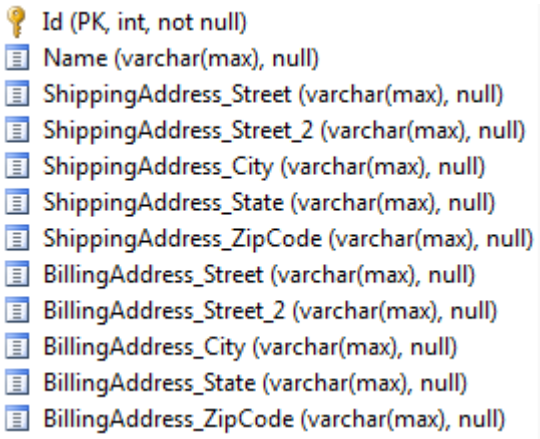
をすると、データベースのされたフィールドを、メイン·タイプのであるのタイプにマップできます。

```
[ComplexType]
public class Address
{
    public string Street { get; set; }
    public string Street_2 { get; set; }
    public string City { get; set; }
    public string State { get; set; }
    public string ZipCode { get; set; }
}
```

このは、のエンティティでできます。じエンティティタイプですることもできます。

```
public class Customer
{
    public int Id { get; set; }
    public string Name { get; set; }
    ...
    public Address ShippingAddress { get; set; }
    public Address BillingAddress { get; set; }
}
```

このエンティティタイプは、データベースのテーブルにされ、のようになります。



もちろん、このは、1nのけCustomer-Addressがモデルになりますが、このはなのをしています。

オンラインでをむ https://riptutorial.com/ja/entity-framework/topic/5527/

22: エンティティのみみ

モデルがしくしているは、EntityFrameworkをしてするデータをにみむことができます。あなたは、なみみ、なみみ、なみみという3つのがあります。

でされるモデル

```
public class Company
{
    public int Id { get; set; }
    public string FullName { get; set; }
    public string ShortName { get; set; }

    // Navigation properties
    public virtual Person Founder { get; set; }
    public virtual ICollection<Address> Addresses { get; set; }
}

public class Address
{
    public int Id { get; set; }
    public int CompanyId { get; set; }
    public int CountryId { get; set; }
    public int CityId { get; set; }
    public string Street { get; set; }

    // Navigation properties
    public virtual Company Company { get; set; }
    public virtual Country Country { get; set; }
    public virtual City City { get; set; }
}
```

Examples

レイジーローディング

レイジーローディングはデフォルトでになっています。レイジーローディングは、したプロキシクラスをし、ナビゲーションプロビジョニングをオーバーライドすることによってされます。ロードは、プロパティにめてアクセスするときにします。

```
int companyId = ...;
Company company = context.Companies
    .First(m => m.Id == companyId);
Person founder = company.Founder; // Founder is loaded
foreach (Address address in company.Addresses)
{
    // Address details are loaded one by one.
}
```

のナビゲーションプロパティにしてLazyローディングをオフにするには、virtualキーワードをプロパティからします。

```
public Person Founder { get; set; } // "virtual" keyword has been removed
```

Lazyのみみをににしたいは、 *Context*コンストラクタなどのをするがあります。

```
public class MyContext : DbContext
{
    public MyContext(): base("Name=ConnectionString")
    {
        this.Configuration.LazyLoadingEnabled = false;
    }
}
```

シリアルをしているは、ロ―ドをオフにしてください。シリアライザはすべてのプロパティにアクセスするため、デ―タベ―スからすべてのプロパティをロ―ドします。さらに、ナビゲ―ションプロパティでル―プをすることもできます。

なみみ

なみみでは、なすべてのエンティティをにみむことができます。 1つのデ―タベ―スびしですべてのエンティティをすることをするは、 *Eager*のみみがです。また、のレベルをみむこともできます。

するエンティティをロ―ドするには*2*つのオプションがあります。 くけするか、 *Include*メソッドのオ―バ―ロ―ドをできます。

くけされた

```
// Load one company with founder and address details
int companyId = ...;
Company company = context.Companies
    .Include(m => m.Founder)
    .Include(m => m.Addresses)
    .SingleOrDefault(m => m.Id == companyId);

// Load 5 companies with address details, also retrieve country and city
// information of addresses
List<Company> companies = context.Companies
    .Include(m => m.Addresses.Select(a => a.Country));
    .Include(m => m.Addresses.Select(a => a.City))
    .Take(5).ToList();
```

このメソッドは、Entity Framework 4.1でできます。 `using System.Data.Entity;`てがあることをしてください`using System.Data.Entity;`セット。

のオ―バ―ロ―ド。

```
// Load one company with founder and address details
int companyId = ...;
Company company = context.Companies
```

```
    .Include("Founder")
    .Include("Addresses")
    .SingleOrDefault(m => m.Id == companyId);

// Load 5 companies with address details, also retrieve country and city
// information of addresses
List<Company> companies = context.Companies
    .Include("Addresses.Country");
    .Include("Addresses.City"))
    .Take(5).ToList();
```

なみみ

*Lazy*ロ―ディングをオフにしたは、エントリの*Load*メソッドをにびして、してエンティティをロ―ドできます。 はのナビゲ―ションプロパティをロ―ドするためにされますが、 *Collection*はコレクションをするためにされます。

```
Company company = context.Companies.FirstOrDefault();
// Load founder
context.Entry(company).Reference(m => m.Founder).Load();
// Load addresses
context.Entry(company).Collection(m => m.Addresses).Load();
```

*Eager*のみみには、のメソッドのオ―バ―ロ―ドをしてエンティティをでみむことができます

```
Company company = context.Companies.FirstOrDefault();
// Load founder
context.Entry(company).Reference("Founder").Load();
// Load addresses
context.Entry(company).Collection("Addresses").Load();
```

フィルタのエンティティ。

*Query*メソッドをして、ロ―ドされたエンティティをフィルタリングできます。

```
Company company = context.Companies.FirstOrDefault();
// Load addresses which are in Baku
context.Entry(company)
    .Collection(m => m.Addresses)
    .Query()
    .Where(a => a.City.Name == "Baku")
    .Load();
```

のクエリ

するデ―タがであるがある、またはのサブセットのみがクエリをできるなどです。なをするがない、をにするがあります。

```
var dbContext = new MyDbContext();
var denormalizedType = from company in dbContext.Company
```

```
                        where company.Name == "MyFavoriteCompany"
                        join founder in dbContext.Founder
                        on company.FounderId equals founder.Id
                        select new
                        {
                            CompanyName = company.Name,
                            CompanyId = company.Id,
                            FounderName = founder.Name,
                            FounderId = founder.Id
                        };
```

あるいは、クエリで

```
var dbContext = new MyDbContext();
var denormalizedType = dbContext.Company
                        .Join(dbContext.Founder,
                                    c => c.FounderId,
                                    f => f.Id ,
                                    (c, f) => new
                                    {
                                        CompanyName = c.Name,
                                        CompanyId = c.Id,
                                        FounderName = f.Name,
                                        FounderId = f.Id
                                    })
                        .Select(cf => cf);
```

オンラインでエンティティのみみをむ https://riptutorial.com/ja/entity-framework/topic/4678/エンティティのみみ

23: なマッピングシナリオエンティティ、テーブル

き

エンティティまたはテーブルをサポートするようにEFモデルをする

Examples

エンティティ

だから、このようなエンティティクラスがあるとしましょう

```
public class Person
{
   public int PersonId { get; set; }
   public string Name { get; set; }
   public string ZipCode { get; set; }
   public string City { get; set; }
   public string AddressLine { get; set; }
}
```

に、このPersonエンティティをPersonIdとNameをつテーブルとアドレスのをつテーブルの2つのテーブルにマップしたいとします。もちろん、アドレスがするをするには、ここにPersonIdがです。ですから、にはエンティティを2つまたはそれのにすることです。したがって、、エンティティ。これをうには、プロパティをのテーブルにマッピングします。

```
public class MyDemoContext : DbContext
{
  public DbSet<Person> Products { get; set; }

  protected override void OnModelCreating(DbModelBuilder modelBuilder)
  {
      modelBuilder.Entity<Person>().Map(m =>
      {
         m.Properties(t => new { t.PersonId, t.Name });
         m.ToTable("People");
      }).Map(m =>
      {
        m.Properties(t => new { t.PersonId, t.AddressLine, t.City, t.ZipCode });
        m.ToTable("PersonDetails");
      });
  }
}
```

PeopleとPersonDetailsという2つのテーブルがされます。 personにはPersonIdとNameの2つのフィールドがあり、PersonDetailsにはPersonId、AddressLine、CityとZipCodeの4つのがあります

。 Peopleでは、PersonIdがキーです。 PersonDetailsでは、プライマリ·キーもPersonIdですが、PersonのPersonIdをするキーです。

People DbSetをすると、EFはPersonIdsのをして、のからデータをしてエンティティーをりみます。

また、のをすることもできます。

```
protected override void OnModelCreating(DbModelBuilder modelBuilder)
{
  modelBuilder.Entity<Person>().Map(m =>
  {
    m.Properties(t => new { t.PersonId });
    m.Property(t => t.Name).HasColumnName("PersonName");
    m.ToTable("People");
  }).Map(m =>
  {
    m.Property(t => t.PersonId).HasColumnName("ProprietorId");
    m.Properties(t => new { t.AddressLine, t.City, t.ZipCode });
    m.ToTable("PersonDetails");
  });
}
```

これによりじテーブルがされますが、PeopleテーブルにはNameカラムのわりにPersonNameカラムがあり、PersonDetailsテーブルにはPersonIdカラムのわりにProprietorIdがあります。

テーブル

そして、エンティティのをしたいとしましょう1つのエンティティを2つのテーブルにマッピングするのではなく、1つのテーブルを2つのエンティティにマッピングしたいとします。これはテーブルとばれます。 PersonId、Name、AddressLine、City、ZipCodeの5つのをつテーブルが1つあり、PersonIdがキーであるとします。そして、あなたはこのようなEFモデルをりたいとっています

```
public class Person
{
  public int PersonId { get; set; }
  public string Name { get; set; }
  public Address Address { get; set; }
}

public class Address
{
  public string ZipCode { get; set; }
  public string City { get; set; }
  public string AddressLine { get; set; }
  public int PersonId { get; set; }
  public Person Person { get; set; }
}
```

1つのことはすぐにびします.AddressにAddressIdはありません。これは、2つのエンティティがじテーブルにマップされているため、じキーもつがあるためです。 あなたがテーブルをう、これは

あなたがしなければならないものです。したがって、テーブルのに、アドレスエンティティをしてプライマリキーをするもあります。そして、ここにがあります

```
public class MyDemoContext : DbContext
{
  public DbSet<Person> Products { get; set; }
  public DbSet<Address> Addresses { get; set; }

  protected override void OnModelCreating(DbModelBuilder modelBuilder)
  {
     modelBuilder.Entity<Address>().HasKey(t => t.PersonId);
     modelBuilder.Entity<Person>().HasRequired(t => t.Address)
                                  .WithRequiredPrincipal(t => t.Person);

     modelBuilder.Entity<Person>().Map(m => m.ToTable("People"));
     modelBuilder.Entity<Address>().Map(m => m.ToTable("People"));
  }
}
```

オンラインでなマッピングシナリオエンティティ、テーブルをむ https://riptutorial.com/ja/entity-framework/topic/9362/なマッピングシナリオ-エンティティ-テーブル

クレジット

S. No		Contributors
1	Entity Frameworkをいめる	Adil Mammadov, Community, DavidG, Eldho, Jacob Linney, Martin4ndersen, Matas Vaitkevicius, Nasreddine, NovaDev, Parth Patel, Stephen Reindl, tmg
2	EFにおける	Amit Shahani, Anshul Nigam, DavidG, Gert Arnold, Jacob Linney, Kobi, lucavgobbi, Stephen Reindl, tmg, wertzui
3	Entity Frameworkコードファースト	Balázs Nagy, Jozef Lačný
4	Entity Frameworkとのマッピングコード11および	Akos Nagy
5	Entity Frameworkとのマッピングコードファースト11およびバリエーション	Akos Nagy
6	Entity-Framework with Postgresql	skj123
7	Entity-Frameworkコードのの	CGritton, hasan, Joshit, Mostafa, RamenChef, Stephen Reindl
8	EntityFrameworkによるコードファースト	lucavgobbi
9	entity-frameworkの.t4テンプレート	Matas Vaitkevicius, Tetsuya Yamamoto
10	SQLiteをしたエンティティフレームワーク	Jason Tyler
11	エンティティフレームワークのベストプラクティスシンプルプロフェッショナル	Braiam, Mina Matta

12	エンティティの	Gert Arnold
13	コードののデータアノテーション	bubi, CptRobby, Daniel A. White, Daniel Lemke, DavidG, Diego, Gert Arnold, Jozef Lačný, Mark Shevchenko, Matas Vaitkevicius , Parth Patel, Piotrek, tmg, Tushar patel
14	コードのの	MacakM, Parth Patel, Sivanantham Padikkasu, Stephen Reindl, tmg
15	コードファースト - なAPI	Adil Mammadov, Daniel Lemke, Jason Tyler, tmg
16	データベースののモデル	Matas Vaitkevicius, Tetsuya Yamamoto
17	データベース	Jozef Lačný
18	トラッキングとノントラッキング	hasan, Sampath, Stephen Reindl, tmg
19	トランザクション	CptRobby, DavidG, Gert Arnold
20	モデルの	SOfanatic, Tushar patel
21		CptRobby, Gert Arnold
22	エンティティのみみ	Adil Mammadov, Florian Haider, Gert Arnold, hasan, Joshit, Matas Vaitkevicius, tmg
23	なマッピングシナリオエンティティ、テーブル	Akos Nagy