Entity Framework

Table of Contents

[Database First 3](#_Toc155876174)

[Steps 3](#_Toc155876175)

[Complex Type 3](#_Toc155876176)

[Code First 3](#_Toc155876177)

[New project 3](#_Toc155876178)

[Install NuGet Entity Framework 3](#_Toc155876179)

[Add Connection String to App.config/Web.config 3](#_Toc155876180)

[Model 3](#_Toc155876181)

[DBContext 4](#_Toc155876182)

[Migration 4](#_Toc155876183)

[Enable Migrations 4](#_Toc155876184)

[Add Migration 4](#_Toc155876185)

[Update Migration 4](#_Toc155876186)

[Data Annotation 4](#_Toc155876187)

[Table 4](#_Toc155876188)

[Column 4](#_Toc155876189)

[Key 4](#_Toc155876190)

[Primary Key 4](#_Toc155876191)

[Composite keys 5](#_Toc155876192)

[Columns 5](#_Toc155876193)

[Required 5](#_Toc155876194)

[MaxLength and MinLength 5](#_Toc155876195)

[NotMapped 6](#_Toc155876196)

[ComplexType 6](#_Toc155876197)

[ConcurrencyCheck 7](#_Toc155876198)

[TimeStamp 7](#_Toc155876199)

[Index 7](#_Toc155876200)

[Index 7](#_Toc155876201)

[Unique Index 7](#_Toc155876202)

[Multiple-Column Indexes 7](#_Toc155876203)

[Relationships 7](#_Toc155876204)

[Foreign Keys 7](#_Toc155876205)

[Fluent API 8](#_Toc155876206)

[Default Schema 8](#_Toc155876207)

[Table Name 8](#_Toc155876208)

[Primary Key 8](#_Toc155876209)

[Primary Key without auto number 8](#_Toc155876210)

[Composite Primary Key 8](#_Toc155876211)

[Column 8](#_Toc155876212)

[Index 9](#_Toc155876213)

[Multiple Column Index 9](#_Toc155876214)

[Not Map 9](#_Toc155876215)

[Complex Type 9](#_Toc155876216)

[One-to-many Relationship 9](#_Toc155876217)

[Many-to-many Relationship 9](#_Toc155876218)

[One-to-zero/one Relationship 9](#_Toc155876219)

[One-to-one Relationship 9](#_Toc155876220)

[Cascade Delete 10](#_Toc155876221)

[Loading 10](#_Toc155876222)

[Lazy Loading 10](#_Toc155876223)

[Eager Loading 10](#_Toc155876224)

[Explicit Loading 10](#_Toc155876225)

# Code First with Existing Database

## New project

In Visual Studio

## Install NuGet Entity Framework

In Visual Studio NuGet package manager

## Create new item

* Create a new file, ADO.NET Entity Data Model
* Name EFCodeFirstDBDBContext
* Select Code First from database
* Create a connection string to the existing database
* Finish wizard

This created:

* DBContext file
* Model.cs classes
* Added a Connection String to the App.config/Web.config

## Model

Move the Model.cs classes into Model folder

ICollection vs IList – IList enables to access with indexer

## DBContext

Move the DBContext file into Context folder

# Code First with NO Database – Not Good

## New project

In Visual Studio

## Install NuGet Entity Framework

In Visual Studio NuGet package manager

## Add Connection String to App.config/Web.config

<connectionStrings>

<add name="DefaultConnection" connectionString="Data Source=RonnieDev;Initial Catalog=Pluto2;Persist Security Info=Tr

ue;User ID=sa;Password=qweQWE123!@#" providerName="System.Data.SqlClient"/>

</connectionStrings>

## Model

public class Course

{

public int Id { get; set; }

public string Title { get; set; }

public string Description { get; set; }

public CourseLevelEnum Level { get; set; }

public float FullPrice { get; set; }

public Author Author { get; set; }

public IList<Tag> Tags { get; set; }

}

public class Author

{

public int Id { get; set; }

public string Name { get; set; }

public IList<Course> Courses { get; set; }

}

public class Tag

{

public int Id { get; set; }

public string Name { get; set; }

public IList<Course> Courses { get; set; }

}

## DBContext

public class PlutoContext : DbContext

{

public DbSet<Course> Courses { get; set; }

public DbSet<Author> Authors { get; set; }

public DbSet<Tag> Tags { get; set; }

public PlutoContext() : base("name=DefaultConnection")

{

}

}

## Migration

### Enable Migrations

Execute command in Package Manager Console (only once per project)

enable-migrations

It might be a migration issue.cs file missing, include into the project.

### Add Migration

add-migration MigrationName -Force

### Update Migration

update-database

# Database First – Not Good

This takes a DB and generates code to use. I prefer the other one.

## Steps

* Create DB in SQL
* Create EDMX - .Net Diagram Model  
  Generates model, mapping, and code to C#.
* Update EDMX when DB changes

## Complex Type

Used when a store procedure/function returns a specific result set different from an existing entity.

# Data Annotation

<https://docs.microsoft.com/en-us/ef/ef6/modeling/code-first/data-annotations#notmapped>

## Table

[Table("InternalBlogs")]

public class Blog

## Column

[Column("BlogDescription", TypeName="ntext")]

public String Description {get; set;}

## Key

### Primary Key

Primary key, defaults to id or ClassName+ID

    [Key]

    [DatabaseGenerated(DatabasaeGenerationOption.None)]

public int PrimaryTrackingKey { get; set; }

### Composite keys

[Key]

[Column(Order = 1)]

public int PassportNumber { get; set; }

[Key]

[Column(Order = 2)]

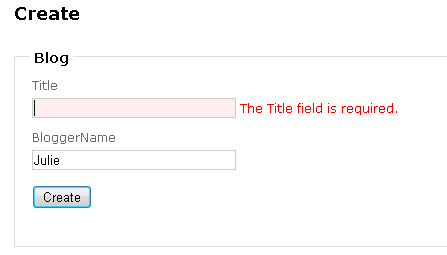
public string IssuingCountry { get; set; }

## Columns

### Required

[Required, ErrorMessage = ”Error message”]

public string Title { get; set; }



### MaxLength and MinLength

[MaxLength(10),MinLength(5)]

public string BloggerName { get; set; }

MaxLength will be used for the database nVarChar(10)

[MaxLength(10, ErrorMessage="BloggerName must be 10 characters or less"),MinLength(5)]

public string BloggerName { get; set; }



### NotMapped

[NotMapped]

public string BlogCode

{

get

{

return Title.Substring(0, 1) + ":" + BloggerName.Substring(0, 1);

}

}

### ComplexType

1 to 1 relationship between two tables/classes

[ComplexType]

public class BlogDetails

{

public DateTime? DateCreated { get; set; }

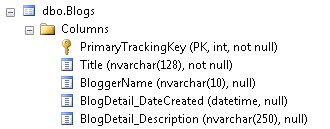
    [MaxLength(250)]

    public string Description { get; set; }

}

If the Blog Class add the following property

    public BlogDetails BlogDetail { get; set; }



### ConcurrencyCheck

[ConcurrencyCheck, MaxLength(10, ErrorMessage="BloggerName must be 10 characters or less"),MinLength(5)]

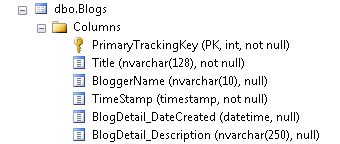
public string BloggerName { get; set; }

### TimeStamp

non-nullable timestamp

[Timestamp]

public Byte[] TimeStamp { get; set; }



## Index

### Index

[Index]

public int Rating { get; set; }

### Unique Index

[Index(IsUnique = true)]

[StringLength(200)]

public string Username { get; set; }

### Multiple-Column Indexes

[Index("IX\_BlogIdAndRating", 2)]

public int Rating { get; set; }

[Index("IX\_BlogIdAndRating", 1)]

public int BlogId { get; set; }

## Relationships

### Foreign Keys

public class Course

{

[ForeignKey("Author")]

public int AauthorId { get; set; }

public Author Aauthor { get; set; }

}

..

# Fluent API

In the DBContext derived class

protected override void OnModelCreating(DbModelBuilder modelBuilder)

{

modelBuilder.Configurations.Add(new VideoConfiguration());

base.OnModelCreating(modelBuilder);

}

public class VideoConfiguration : EntityTypeConfiguration<Video>

{

public VideoConfiguration()

{

Property(v => v.Name)

.IsRequired()

.HasMaxLength(255);

HasRequired(v => v.Genre)

.WithMany(g => g.Videos)

.HasForeignKey(v => v.GenreId);

HasMany(v => v.Tags)

.WithMany(t => t.Videos)

.Map(m =>

{

m.ToTable("VideoTags");

m.MapLeftKey("VideoId");

m.MapRightKey("TagId");

});

}

}

## Default Schema

modelBuilder.HasDefaultSchema("sales");

## Table Name

Entity<Course>.ToTable(“tbl\_Course”, “catalog”);

## Primary Key

modelBuilder.Entity<OfficeAssignment>().HasKey(t => t.InstructorID);

## Primary Key without auto number

modelBuilder.Entity<Department>().Property(t => t.DepartmentID)

.HasDatabaseGeneratedOption(DatabaseGeneratedOption.None);

## Composite Primary Key

modelBuilder.Entity<Department>().HasKey(t => new { t.DepartmentID, t.Name });

## Column

Entity<Course>.Property(t => t.Name)

.HasColumnName(“sName”)

.HasColumnType(“varchar”)

.HasColumnOrder(2)

.IsRequired()

.HasMaxLength(255)

.IsUnicode(false); // varChar vs nVarChar

## Index

modelBuilder

.Entity<Department>()

.Property(t => t.Name)

.HasColumnAnnotation("Index", new IndexAnnotation(new IndexAttribute()));

## Multiple Column Index

modelBuilder

.Entity<Department>()

.Property(t => t.Name)

.HasColumnAnnotation(

"Index",

new IndexAnnotation(new[]

{

new IndexAttribute("Index1"),

new IndexAttribute("Index2") { IsUnique = true }

})));

## Not Map

modelBuilder.Entity<Department>().Ignore(t => t.Budget);

## Complex Type

modelBuilder.ComplexType<Details>()

.Property(t => t.Location)

.HasMaxLength(20);

## One-to-many Relationship

Entity<Author>.HasMany(a => a.Courses)

.WithRequired(c => c.Author)

.HasForeignKey(c => c.AuthorId);

## Many-to-many Relationship

Entity<Course>.HasMany(c => c.Tags)

.WithMany(t => t.Courses)

.Map(m => {

m.ToTable(“CourseTag”);

m.MapLeftKey(“CourseId”);

m.MapRightKey(“TagId”);

});

## One-to-zero/one Relationship

Entity<Course>.HasOptional(c => c.Caption)

.WithRequired(c => c.Course);

## One-to-one Relationship

Entity<Course>.HasRequired(c => c.Cover)

.WithRequiredPrincipal(c => c.Course);

Entity<Cover>.HasRequired(c => c.Course)

.WithRequiredDependent(c => c.Cover);

## Cascade Delete

modelBuilder.Entity<Course>()

.HasRequired(t => t.Department)

.WithMany(t => t.Courses)

.HasForeignKey(d => d.DepartmentID)

.WillCascadeOnDelete(false);

# Loading

## Lazy Loading

Declare the property as a virtual

public virtual Author Author { get; set; }

Or add in DBBContext ctor

this.Configuration.LazyLoadingEnabled = false;

## Eager Loading

add Include cause the SQL to have a join with the relationship

var users = dlContext.Users.Include(x => x.DepartmentID).ToList();

## Explicit Loading

Explicit loading of joined data

First, load the main object

var user = dlContext.Users.FirstOrDefault(x => x.ID == 1);

Then, load the related data

dlContext.Departments.Where(x => x.ID == user.DepartmentID).Load();