Entity Framework

Table of Contents

[Code First 3](#_Toc155887735)

[Code First with NO Database – Not Good 3](#_Toc155887736)

[Add Connection String to App.config/Web.config 3](#_Toc155887737)

[Model 3](#_Toc155887738)

[DBContext 4](#_Toc155887739)

[Database First – Not Good 4](#_Toc155887740)

[Steps 4](#_Toc155887741)

[Complex Type 4](#_Toc155887742)

[Data Annotation 5](#_Toc155887743)

[Table 5](#_Toc155887744)

[Column 5](#_Toc155887745)

[Key 5](#_Toc155887746)

[Primary Key 5](#_Toc155887747)

[Composite keys 5](#_Toc155887748)

[Columns 5](#_Toc155887749)

[Required 5](#_Toc155887750)

[MaxLength and MinLength 6](#_Toc155887751)

[NotMapped 6](#_Toc155887752)

[ComplexType 7](#_Toc155887753)

[ConcurrencyCheck 7](#_Toc155887754)

[TimeStamp 7](#_Toc155887755)

[Index 8](#_Toc155887756)

[Index 8](#_Toc155887757)

[Unique Index 8](#_Toc155887758)

[Multiple-Column Indexes 8](#_Toc155887759)

[Relationships 8](#_Toc155887760)

[Foreign Keys 8](#_Toc155887761)

[Fluent API 9](#_Toc155887762)

[Default Schema 9](#_Toc155887763)

[Table Name 9](#_Toc155887764)

[Primary Key 9](#_Toc155887765)

[Primary Key without auto number 9](#_Toc155887766)

[Composite Primary Key 9](#_Toc155887767)

[Column 9](#_Toc155887768)

[Index 10](#_Toc155887769)

[Multiple Column Index 10](#_Toc155887770)

[Not Map 10](#_Toc155887771)

[Complex Type 10](#_Toc155887772)

[One-to-many Relationship 10](#_Toc155887773)

[Many-to-many Relationship 10](#_Toc155887774)

[One-to-zero/one Relationship 10](#_Toc155887775)

[One-to-one Relationship 10](#_Toc155887776)

[Cascade Delete 11](#_Toc155887777)

[Loading 11](#_Toc155887778)

[Lazy Loading 11](#_Toc155887779)

[Eager Loading 11](#_Toc155887780)

[Explicit Loading 11](#_Toc155887781)

# Code First

|  |  |  |
| --- | --- | --- |
|  | **Code First with Existing Database** | **With No Database** |
|  | **Preferred – BEST OPTION** | **Use left option** |
| Project | New Project in Visual Studio | |
| Entity Framework | EF6 - Install EntityFramework  EF Core – Install EntityFrameworkCore.(Design, Tools, SQLServer) | |
|  | Import model from DB   * 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.cs * Model.cs classes * Connection String to App/Web.config | Add Connection String to App /Web.config |
|  | * Move Model.cs files into Model folder * ICollection vs IList – IList enables to access with indexer * Make collections as virtual functions |  |
|  | * Move DBContext.cs file into Context folder |  |
| Init Migration | EF6 - enable-migrations – only once per project, add Migrations folder to project  EF Core – NO need to do this, migrations is enabled by default | |
| Existing Database | add-migration name -IgnoreChanges -Force  Because the current model already exists  use -Project ProjectName to add to a specific project |  |
| update-database -Project ProjectName |  |
| Code | Code Model.cs | |
|  | Code DBContext.cs | |
|  | add-migration name -Project ProjectName to add to a specific project | |
|  | check the new file.cs in migrations folder and verify Up() and Down() methods | |
|  | update-database -Project ProjectName | |

# Code First with NO Database – Not Good

## 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")

{

}

}

# 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)]

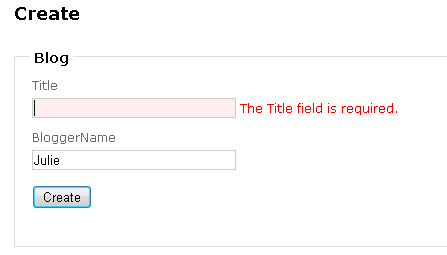
public string IssuingCountry { get; set; }

## 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; }

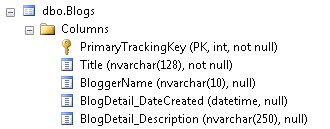
    [MaxLength(250)]

    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)]

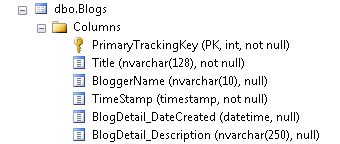
public string BloggerName { get; set; }

### 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();