3.3.0 – Entity Validation

Introduction

In lesson 3.1.0, all the Entity classes were created. As they were created, Visual Studio added some basic property validations. For example, examine the **`Category.cs`** file in the Entities folder of the **eStoreData** project to see:

```
Category.cs
eStoreData                    eStoreData.Entities.Category                    Category()
 1  namespace eStoreData.Entities
 2  {
 3      using System;
 4      using System.Collections.Generic;
 5      using System.ComponentModel.DataAnnotations;
 6      using System.ComponentModel.DataAnnotations.Schema;
 7      using System.Data.Entity.Spatial;
 8
 9      [Table("Category")]
10      public partial class Category
11      {
12          [System.Diagnostics.CodeAnalysis.SuppressMessage("Microsoft.Usage", "CA2214:DoNotCallOverridableMethodsInConstructors")]
13          public Category()
14          {
15              Products = new HashSet<Product>();
16          }
17
18          public int CategoryID { get; set; }
19
20          [Required]
21          [StringLength(30)]
22          public string CategoryName { get; set; }
23
24          [StringLength(100)]
25          public string Description { get; set; }
26
27          [System.Diagnostics.CodeAnalysis.SuppressMessage("Microsoft.Usage", "CA2227:CollectionPropertiesShouldBeReadOnly")]
28          public virtual ICollection<Product> Products { get; set; }
29      }
30  }
31
```

Figure 1: Category Entity

Entity Classes

Examining the code above, there are some important lines:

- Line 9: This line is an annotation that tells Visual Studio the name of the database table this Entity class relates to.
- Lines 12-16: DO NOT delete. These will be used aid in navigation between **Primary** and **Foreign** keys; for this table, it is related to the **Product** Entity.
- Line 20: This annotation means that the property, **CategoryName**, is required
- Line 21: This annotation means that the length of the name must not exceed 30 characters in length.
- Line 27: DO NOT delete
- Line 28: This is a navigation to the **Product** Entity. Because it is an `ICollection`, it means that a single **Category** can have many **Product** Entities. In the **`Product.cs`** Entity class file there is a corresponding line:
  `public virtual Category Category { get; set; }`

This means a **Product** belongs to only one **Category**.

Validation Messages

The only thing missing from the default validation annotations, automatically generated by Visual Studio, is some friendly error messages. We need to add our own messages to make the errors easily understood when they happen.

Modify line 20, of **`Category.cs`**, to be:

```
[Required (ErrorMessage ="Category Name is a required field.")]
```

For all `[Required]` annotations, you will need to add a similar custom error message relating to that field.

Modify line 21 to be:

```
[StringLength(30, ErrorMessage = "Category Name is too long. Max is 30 characters.")]
```

For all `[StringLength(##)]`, where ## is the maximum allowed length, annotations, you will need to add a similar custom error message relating to that field, and the maximum length allowed.

Other Validations

Adding error messages to the automatically generated validation annotations is a good start. Some fields may need additional validations that are based on business rules.

[MinLength]

For **CategoryName**, what would be a reasonable minimum length for the name? For example, we could say that the minimum length is 3. To do this, add the following line of code below the string length annotation:

```
[MinLength(10, ErrorMessage = "Category Name is too short. Min is 3 characters.")]
```

[Range]

When validating numerical data, it is standard practice to validate the number against the smallest value allowed to the largest value allowed. This data annotation does that. Its structure is `[Range(minValue, maxValue, ErrorMessage)]`. An example would be for **OrderCost**, **SellingPrice**, **QuantityOnHand**, **ReOrderLevel**, and **OnOrder** in the **Product.cs** class file (only **OrderCost** shown):

```
[Range(0.01, double.MaxValue, ErrorMessage ="Invalid Order Cost. Must be greater than
0.01.")]
```

Note: Range can only be used for properties that are numbers (decimal, int, double) but **NOT** for a **PK** or **FK** field.

[Key]

If a database primary key ends with "id", "Id", or "ID", the `[Key]` annotation is not required, otherwise the `[Key]` annotation **MUST** be there. For this course, always add the `[Key]` for the primary key field!

Composite Primary Key

Sometimes you will see to properties that are both shown as a Primary Key. When you have this the table is called an **Associative Entity** which is used to resolve a `Many:Many` relationship between two tables. To annotate this, do the following:

```
[Key]
[Column (Order=1)]
public int PrimaryKey1ID { get; set; }

[Key]
[Column (Order=2)]
public int PrimaryKey2ID { get; set; }
```

eStore

Complete the validations for **`Category.cs`**, and **`Product.cs`**.

Exercise

Complete Exercise 3.3.1.