Add a new field to an ASP.NET Core MVC app

04/14/2017 3 minutes to read Contributors

In this article

By Rick Anderson

This tutorial will add a new field to the `Movies` table. We'll drop the database and create a new one when we change the schema (add a new field). This workflow works well early in development when we don't have any production data to perserve.

Once your app is deployed and you have data that you need to perserve, you can't drop your DB when you need to change the schema. Entity Framework Code First Migrations allows you to update your schema and migrate the database without losing data. Migrations is a popular feature when using SQL Server, but SQLlite doesn't support many migration schema operations, so only very simply migrations are possible. See SQLite Limitations for more information.

Adding a Rating Property to the Movie Model

Open the *Models/Movie.cs* file and add a `Rating` property:

C# Copy

```
public class Movie
{
    public int ID { get; set; }
    public string Title { get; set; }

    [Display(Name = "Release Date")]
    [DataType(DataType.Date)]
    public DateTime ReleaseDate { get; set; }
    public string Genre { get; set; }
```

```
    public decimal Price { get; set; }
    public string Rating { get; set; }
}
```

Because you've added a new field to the `Movie` class, you also need to update the binding whitelist so this new property will be included. In *MoviesController.cs*, update the `[Bind]` attribute for both the `Create` and `Edit` action methods to include the `Rating` property:

C# Copy

```
[Bind("ID,Title,ReleaseDate,Genre,Price,Rating")]
```

You also need to update the view templates in order to display, create, and edit the new `Rating` property in the browser view.

Edit the */Views/Movies/Index.cshtml* file and add a `Rating` field:

HTML Copy

```
<table class="table">
    <thead>
        <tr>
            <th>
                @Html.DisplayNameFor(model => model.movies[0].Title)
            </th>
            <th>
                @Html.DisplayNameFor(model => model.movies[0].ReleaseDate)
            </th>
            <th>
                @Html.DisplayNameFor(model => model.movies[0].Genre)
            </th>
            <th>
                @Html.DisplayNameFor(model => model.movies[0].Price)
            </th>
            <th>
                @Html.DisplayNameFor(model => model.movies[0].Rating)
            </th>
            <th></th>
        </tr>
    </thead>
    <tbody>
        @foreach (var item in Model.movies)
        {
            <tr>
                <td>
                    @Html.DisplayFor(modelItem => item.Title)
                </td>
                <td>
```

```
            @Html.DisplayFor(modelItem => item.ReleaseDate)
        </td>
        <td>
            @Html.DisplayFor(modelItem => item.Genre)
        </td>
        <td>
            @Html.DisplayFor(modelItem => item.Price)
        </td>
        <td>
            @Html.DisplayFor(modelItem => item.Rating)
        </td>
        <td>
```

Update the */Views/Movies/Create.cshtml* with a `Rating` field.

The app won't work until we update the DB to include the new field. If you run it now, you'll get the following `SqliteException`:

Copy

```
SqliteException: SQLite Error 1: 'no such column: m.Rating'.
```

You're seeing this error because the updated Movie model class is different than the schema of the Movie table of the existing database. (There's no `Rating` column in the database table.)

There are a few approaches to resolving the error:

1. Drop the database and have the Entity Framework automatically re-create the database based on the new model class schema. With this approach, you lose existing data in the database — so you can't do this with a production database! Using an initializer to automatically seed a database with test data is often a productive way to develop an app.

2. Manually modify the schema of the existing database so that it matches the model classes. The advantage of this approach is that you keep your data. You can make this change either manually or by creating a database change script.

3. Use Code First Migrations to update the database schema.

For this tutorial, we'll drop and re-create the database when the schema changes. Run the following command from a terminal to drop the db:

`dotnet ef database drop`

Update the `SeedData` class so that it provides a value for the new column. A sample change is shown below, but you'll want to make this change for each `new Movie`.

C# Copy

```
new Movie
{
    Title = "When Harry Met Sally",
    ReleaseDate = DateTime.Parse("1989-1-11"),
    Genre = "Romantic Comedy",
    Rating = "R",
    Price = 7.99M
},
```

Add the `Rating` field to the `Edit`, `Details`, and `Delete` view.

Run the app and verify you can create/edit/display movies with a `Rating` field. templates.

Previous - Add search Next - Add validation