# Code First Using the Entity Framework

# Project Set Up

## Setup Steps for Visual Studio

From within VS (using point and click):

* Open the WorkingWithDatabases solution
* Add a console project called CodeFirstEFDemo.csproj to the solution
* Use NuGet to add a reference to the following packages:
  + Microsoft.EntityFrameworkCore
  + Microsoft.EntityFrameworkCore.SqlServer
  + Microsoft.EntityFrameworkCore.Design
  + Microsoft.EntityFrameworkCore.Tools

## Setup Steps for Visual Studio Code

From Command Prompt:

* Create console project and add to the solution by typing:

dotnet new console --output CodeFirstEF-demo

dotnet sln add CodeFirstEF-demo

* Install the following packages by typing:

dotnet add package Microsoft.EntityFrameworkCore

dotnet add package Microsoft.EntityFrameworkCore.SqlServer

dotnet add package Microsoft.EntityFrameworkCore.Design

dotnet tool install --global dotnet-ef

## Now do the following regardless of project type:

**Add a Film class to the project and automatically create an associated database table**

* Add a Folder to the project called Models.
* Add a class to the Models folder called Film.
* Replace all the code in the file with the following:

using System.ComponentModel.DataAnnotations;

namespace CodeFirstEFDemo.Models

{

internal class Film

{

[Key]

public int FilmId { get; set; }

public string Title { get; set; }

public string Overview { get; set; }

}

}

* Add a second class to the Models folder called FilmReviewContext.
* Replace all the code in the file with the following:

using Microsoft.EntityFrameworkCore;

namespace CodeFirstEFDemo.Models

{

internal class FilmReviewContext : DbContext

{

public virtual DbSet<Film> Films { get; set; }

protected override void OnConfiguring(DbContextOptionsBuilder optionsBuilder)

{

optionsBuilder.UseSqlServer(@"Server=**(local)**; Initial Catalog=FilmReviews; trusted\_connection=true;TrustServerCertificate=True; ;MultipleActiveResultSets=true;Encrypt=False");

}

}

}

* **Note, the above script assumes you are using a local copy of the Community edition of SQL Serve**r. If you are using the **SQLExpress** edition then the Data Source will need to be altered to “**Data Source=.\SQLExpress**;…” or the server name or IP address. You may also need to modify the connection credentials to specify a username and password.
* You now need to generate the database migration logic that, when run, will create the database in SQL Server:
  + Visual Studio:
    - Open the Package Manager Console and enter:

Add-Migration Initial

* + Visual Studio Code:
    - Open a terminal window and enter:

dotnet-ef migrations add initial --context FilmReviewContext

* You should see a folder called Migrations has been added to the project that contains some C# files.
* Replace any code in the Program.cs file with the following:

using CodeFirstEFDemo.Models;

using Microsoft.EntityFrameworkCore;

namespace EFCodeFirstFilmReviewsDB

{

internal class Program

{

static void Main(string[] args)

{

using (FilmReviewContext db = new FilmReviewContext())

{

db.Database.Migrate();

}

using (FilmReviewContext db = new FilmReviewContext())

{

db.Films.Add(new Film

{

Title = "Titanic",

Overview = "Would you like ice with that?",

});

db.Films.Add(new Film

{

Title = "ET",

Overview = "Phone Home",

});

db.SaveChanges();

}

}

}

}

* Review the code and try to work out what it’s doing.
* Build and run the code and confirm it behaves as you expected. Use SQL Server Management Studio (SSMS) to see if the tables have been created correctly.

**Enhance the Film class and automatically update the associated database table**

* Add code to the Film class that makes the Title property mandatory and restricts the length of the Overview property to 100 characters:

public class Film

{

[Key]

public int FilmId { get; set; }

**[Required]**

public string? Title { get; set; }

**[Required, MaxLength(100)]**

public string Overview { get; set; }

}

* Create the next database migration:
  + Visual Studio:
    - Open the Package Manager Console and enter:

Add-Migration Title\_Max\_Length

* + Visual Studio Code:
    - Open a terminal window and enter:

dotnet-ef migrations add title\_max\_length --context FilmReviewContext

* Note the changes made to the Migrations folder.
* Rerun the program and ensure the relevant changes have been applied to the table.

**Further enhance the Film class and automatically update the associated database table**

* Add a Revenue property to the Film class:

public class Film

{

[Key]

public int FilmId { get; set; }

[Required]

public string? Title { get; set; }

[Required, MaxLength(100)]

public string Overview { get; set; }

**public long? Revenue { get; set; }**

}

* Create the next database migration:
  + Visual Studio:
    - Open the Package Manager Console and enter:

Add-Migration Revenue

* + Visual Studio Code:
    - Open a terminal window and enter:

dotnet-ef migrations add revenue --context FilmReviewContext

* Note the changes made to the Migrations folder.
* Update the code in the Main method of the Program class

using (FilmReviewContext db = new FilmReviewContext())

{

db.Database.Migrate();

}

using (FilmReviewContext db = new FilmReviewContext())

{

db.Films.Add(new Film

{

Title = "Titanic",

Overview = "Would you like ice with that?",

ReleaseDate = new DateTime(1997, 11, 18),

**Revenue = 1845034188**

});

db.Films.Add(new Film

{

Title = "ET",

Overview = "Phone Home",

ReleaseDate = new DateTime(2004, 03, 19),

**Revenue = 72258126**

});

**db.Films.Add(new Film**

**{ Title= "Jaws",**

**Overview = "Afraid to go into the water",**

**Revenue = 1000000000**

**});**

db.SaveChanges();

}

* Rerun the program and use SSMS to ensure the relevant changes have been applied to the database table.

**Add a Review class and update the Film class**

* Add a Review class to the Models Folder:

public class Review

{

[Key]

public int ReviewId { get; set; }

public int FilmId { get; set; }

public string? Commentary { get; set; }

[Range(1, 10)]

public int Rating { get; set; }

public Film Film { get; set; }

}

* Enhance the Film class to have a relationship with the Review class:

public class Film

{

[Key]

public int FilmId { get; set; }

[Required]

public string? Title { get; set; }

[Required, MaxLength(100)]

public string Overview { get; set; }

public long? Revenue { get; set; }

**public List<Review> Reviews{ get; set; }**

}

* Update the FilmReviewContext class to include a DBSet property for the Reviews:

using Microsoft.EntityFrameworkCore;

namespace CodeFirstEFDemo.Models

{

internal class FilmReviewContext : DbContext

{

public virtual DbSet<Film> Films { get; set; }

**public virtual DbSet<Review> Reviews{ get; set; }**

protected override void OnConfiguring(DbContextOptionsBuilder optionsBuilder)

{

optionsBuilder.UseSqlServer(@"Server=(local); Initial Catalog=FilmReviews; trusted\_connection=true");

}

}

}

* Create the next database migration:
  + Visual Studio:
    - Open the Package Manager Console and enter:

Add-Migration Update\_Model\_Classes

* + Visual Studio Code:
    - Open a terminal window and enter:

dotnet-ef migrations add update\_model\_classses --context FilmReviewContext

* Note the changes made to the Migrations folder.
* Add the following code to the Main method of the Program class:

using (FilmReviewContext db = new FilmReviewContext())

{

db.Database.Migrate();

}

using (FilmReviewContext db = new FilmReviewContext())

{

Film film = db.Films.Single(f => f.Title == "Titanic");

db.Reviews.Add(new Models.Review

{

Commentary = "Gives you a sinking feeling in your stomach",

Rating = 8,

Film = film

});

db.Reviews.Add(new Models.Review

{

Commentary = "Makes you want to phone home",

Rating = 6,

FilmId = db.Films.Single(f => f.Title == "ET").FilmId

});

db.Reviews.Add(new Review

{

Commentary = "It'd 'Be Good' if it wasn't for those pesky kids ",

Rating = 9,

Film = db.Films.Single(d => d.Title == "ET"),

});

db.SaveChanges();

}

* Rerun the program and use SSMS to ensure the relevant changes have been applied to the database table.