**Code first Database creation – Step by step guideline**

Sample -> the name of our base (what are we actually modeling)

1. Create a blank VS solution
2. Create a class library **“Sample.Models”**
   1. Add reference to **System.ComponentModel.DataAnnotations**
   2. **Example Model without any validation or anything:**

namespace Sample.Models

{

using System.ComponentModel.DataAnnotations;

public class Sample

{

[Key]

public int Id { get; set; }

public string Model { get; set; }

public TransmissionType TransmissionType { get; set; }

public short Year { get; set; }

public decimal Price { get; set; }

Foreign Key to another table

public int DealerId { get; set; }

public virtual Dealer Dealer { get; set; }

}

}

1. Create a class library **“Sample.Data”**
   1. Install **Entity Framework**
      1. If already installed, manage NuGet packages-> already installed
   2. Add reference to **“Sample.Models”**
   3. Create class SampleDbContext
      1. using System.Data.Entity;
      2. Inherit DbContext
      3. For every table:

public virtual IDbSet<TableName> TableName { get; set; }

1. Create Console application **“Sample.ConsoleClient”**
   1. Add reference to Entity Framework
   2. Set as startup project
   3. Add usings:

using Sample.Data;

using Sample.Data.Migrations;

* 1. Add reference to “Sample.Data” and “Sample.Models”
  2. Add the ConnectionString in the App.config file before <entityFramework>:

<connectionStrings>

<add name="ConnectionString Name" connectionString="Data Source=.; Initial Catalog = SampleDb; Integrated Security=True"

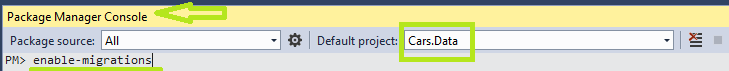
providerName="System.Data.SqlClient" ></add>

</connectionStrings>

* 1. Copy the same App.config (from the previous step) for all the class libraries
  2. Go back to class library **“Sample.Data”** and add base constructor with :base("ConnectionString Name")
  3. Go back to Console application **“Sample.ConsoleClient” and** in Main Method

Database.SetInitializer(new MigrateDatabaseToLatestVersion<SampleDbContext, Configuration>());

1. Enable Migrations in the dbcontext:
   1. Open Package Manager Console (search in the right upper corner for “console”)
   2. Enable-migrations (Sample.Data should be the Default Project)



After it is ready – Migrations Folder will appear in the **“Sample.Data”**

* 1. In Configuration.cs:
     1. Make it public, not internal! and paste:

public sealed class Configuration : DbMigrationsConfiguration<SampleDbContext>

{

public Configuration()

{

this.AutomaticMigrationsEnabled = true;

this.AutomaticMigrationDataLossAllowed = false;

}

}

1. Now – you can go back to models and insert the limitations:
   1. Class creation tips:
      1. One to many relationship: 1 *Sample* has many *SampleExamples*
      2. One to one relationship: 1 *SampleExample* has one *Sample*
      3. [MinLength(3)], [MaxLength(20)]
      4. Make something unique:
         1. Install **Entity Framework**
         2. using System.ComponentModel.DataAnnotations.Schema;
         3. [Index(IsUnique = true)]
   2. Example Model with :

namespace Sample.Models

{

using System.Collections.Generic;

using System.ComponentModel.DataAnnotations;

public class Dealer

{

private ICollection<City> cities;

public Dealer()

{

this.cities = new HashSet<City>();

}

[Key]

public int Id { get; set; }

[Required]

[MaxLength(50)]

public string Name { get; set; }

public virtual ICollection<City> Cities

{

get { return this.cities; }

set { this.cities = value; }

}

}

}

1. JsonImporter – In the console application:
   1. Add Json.NET from NuGet
   2. Add Json files in a folder, add them to the solution, select them, Properties-> Copy to Output Directory: Copy if newer

var files = Directory.GetFiles(Environment.CurrentDirectory).Where(fileName => fileName.EndsWith(".json")).ToList();

foreach (var file in files)

{

var fileContent = File.ReadAllText(file);

var fileCars = JsonConvert.DeserializeObject<IEnumerable<Car>>(fileContent);

cars.AddRange(fileCars);

Console.WriteLine("{0} read.", file);

}

1. [XmlImporter](https://code.msdn.microsoft.com/CSEFStoreXmlFiles-d58d5702) – In the console application:

[Another example](https://github.com/Team-Neptunium/Databases-Team-Neptunium/blob/master/BoardgameSimulator/BoardgameSimulator.Importer/XmlImporter.cs)

IEnumerable<Course> courses =

from c in document.Descendants("Course")

select new Course

{

CourseID = c.Element("CourseId") == null ?

Guid.NewGuid().ToString() : c.Element("CourseId").Value,

Title = c.Element("Title") == null ? null : c.Element("Title").Value,

Credits = c.Element("Credits") == null ? -1 : Int32.Parse(c.Element("Credits").Value),

Department = c.Element("Department") == null ? null : c.Element("Department").Value

};

1. Database Query tips (Let’s hope we never get here):
   1. var dataQuery = db.Cars.AsQueriable(); // this is a query!
   2. dataQuery = dataQuery.OrderBy(x => x.Id); // OrderBy returns a new query !!
   3. add .ToList() last in order to materialize the query
   4. Other tips:
2. [Exporting XML/JSON as C# classes](http://blog.codeinside.eu/2014/09/08/Visual-Studio-2013-Paste-Special-JSON-And-Xml/)
3. [Screenshots and some guidelines from soft uni](http://www.nakov.com/blog/2015/03/23/database-apps-entity-framework-sql-server-xml-json-practical-hands-on-lab-by-nakov/)
4. [Bulk operation to make loading data faster](http://weblog.west-wind.com/posts/2013/Dec/22/Entity-Framework-and-slow-bulk-INSERTs)

[Use a Unit of Work and a new Context for each set of Records](http://weblog.west-wind.com/posts/2013/Dec/22/Entity-Framework-and-slow-bulk-INSERTs#UseaUnitofWorkandanewContextforeachsetofRecords)

[Turn off AutoDetectChangesEnabled](http://weblog.west-wind.com/posts/2013/Dec/22/Entity-Framework-and-slow-bulk-INSERTs#TurnoffAutoDetectChangesEnabled)

[Batching](http://weblog.west-wind.com/posts/2013/Dec/22/Entity-Framework-and-slow-bulk-INSERTs#Batching)

[DbTable.AddRange()](http://weblog.west-wind.com/posts/2013/Dec/22/Entity-Framework-and-slow-bulk-INSERTs#DbTable.AddRange())

## Code first Tips

1. How to implement Guid id - 25:00
2. One to many relatioship - 45:00
3. No using of virtual properties in the constructor. AddOrUpdate - 1:53:00
4. Data annotations - 2:13:00
5. Default values - 2:20:00
6. Complex type - 2:35:00
7. InverseProperty 2:45:00
8. OnModelCreating, fluent api instead of attributes - 2:50:00

Entity Framework -> in LINQ, the queries should be Expression<func<>>, not func<> (1:13:00)