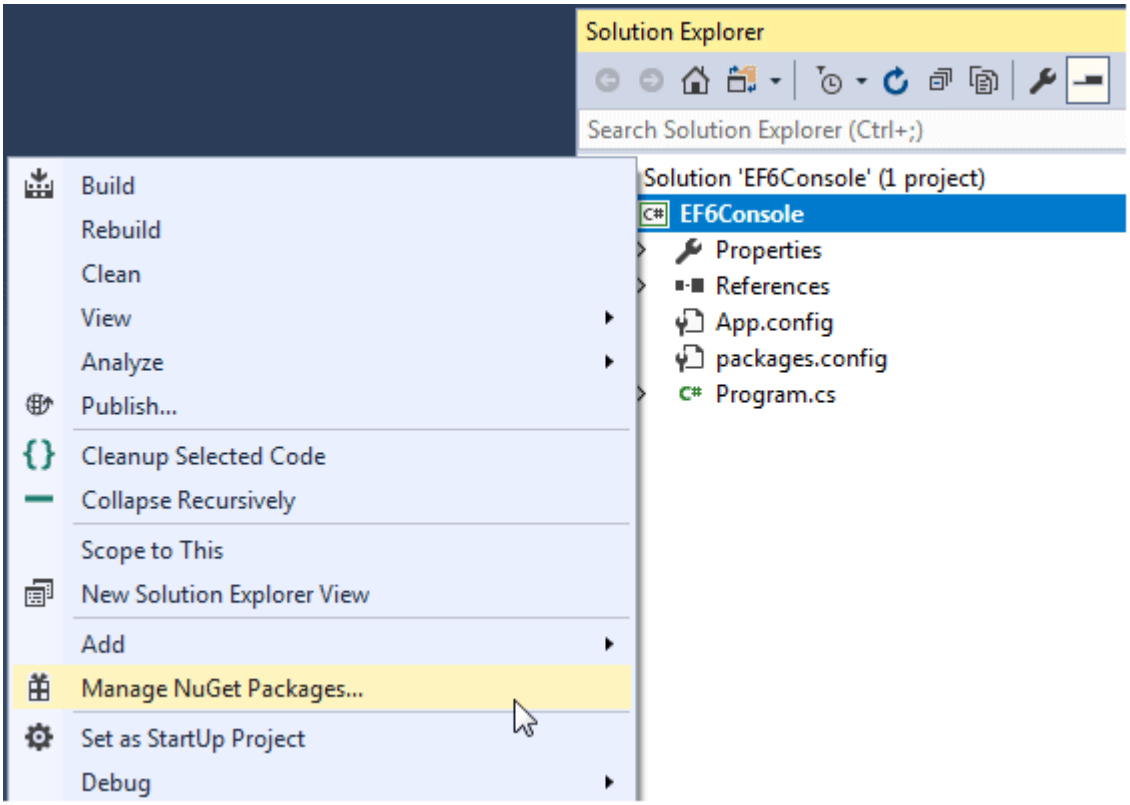
Hướng dẫn Code First

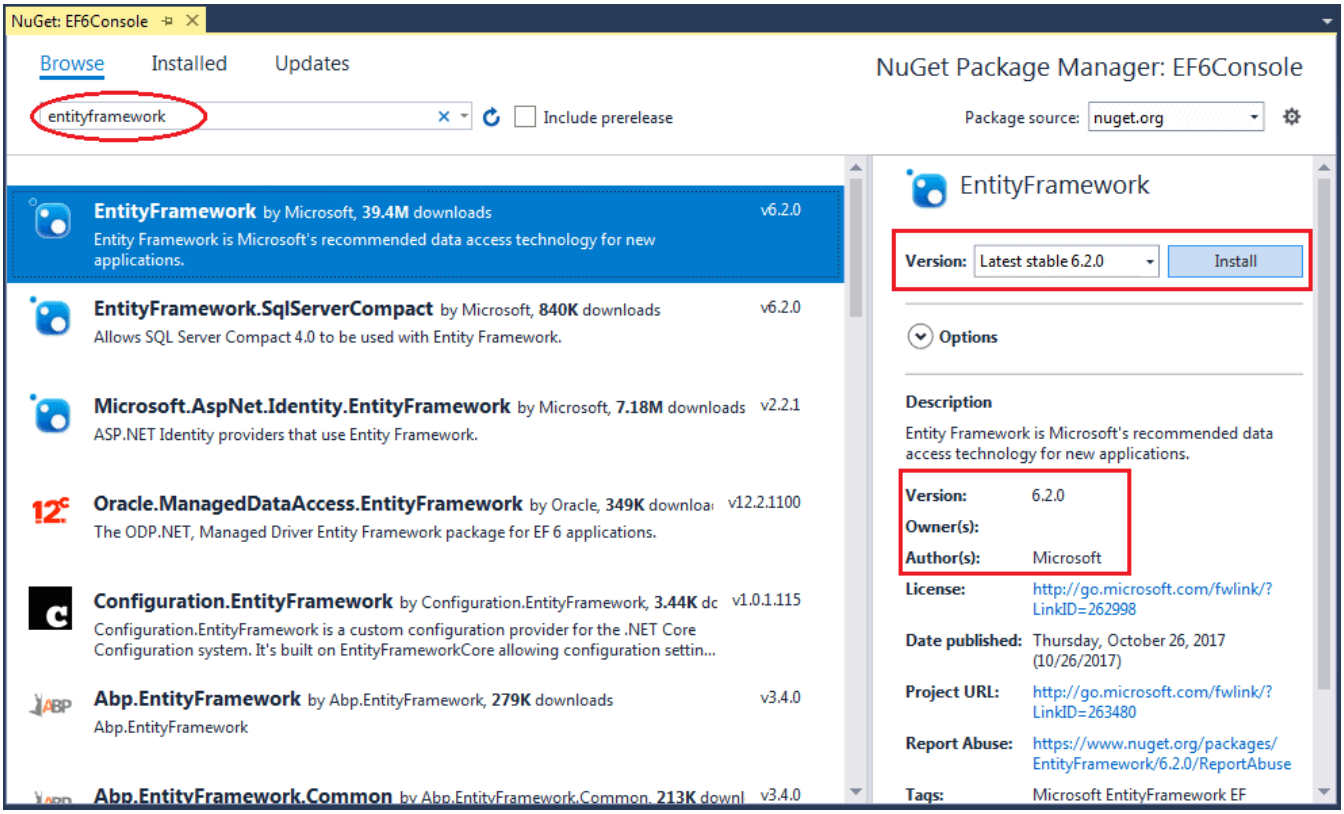
Bước 1. Tạo dự án loại Console App, đặt tên dự án là EF6Console

Bước 2. Install Entity Framework 6

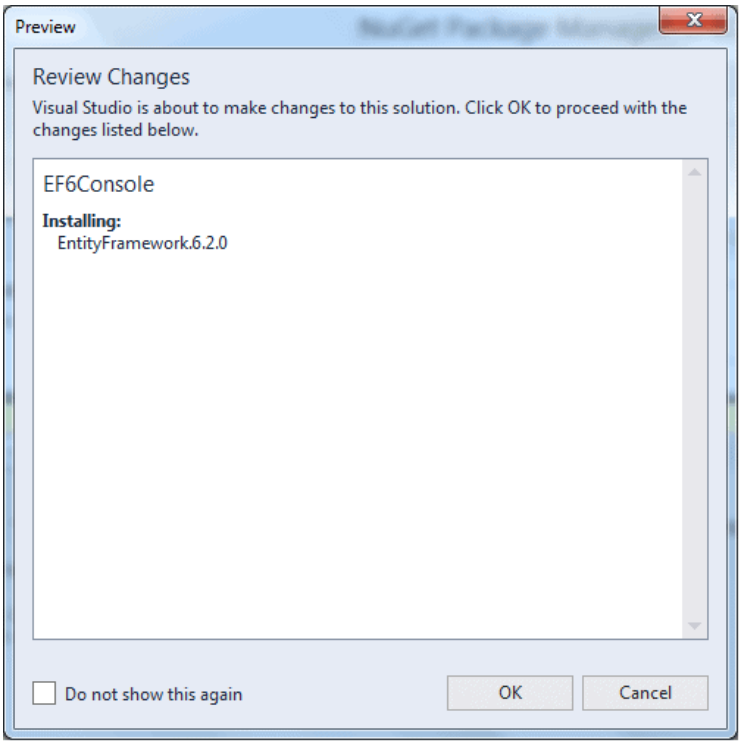
Click chuột phải trên Project và chọn **Manage NuGet Packages...**.

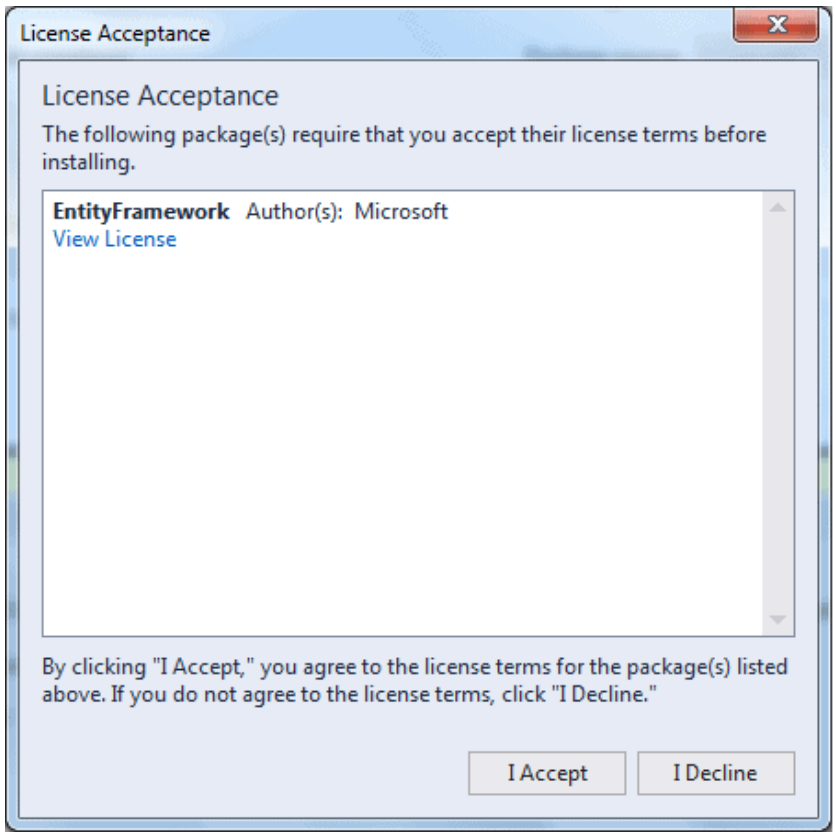


Trong cửa sổ **Manage NuGet Packages** gõ EntityFramework vào hộp tìm kiếm và nhấn enter.

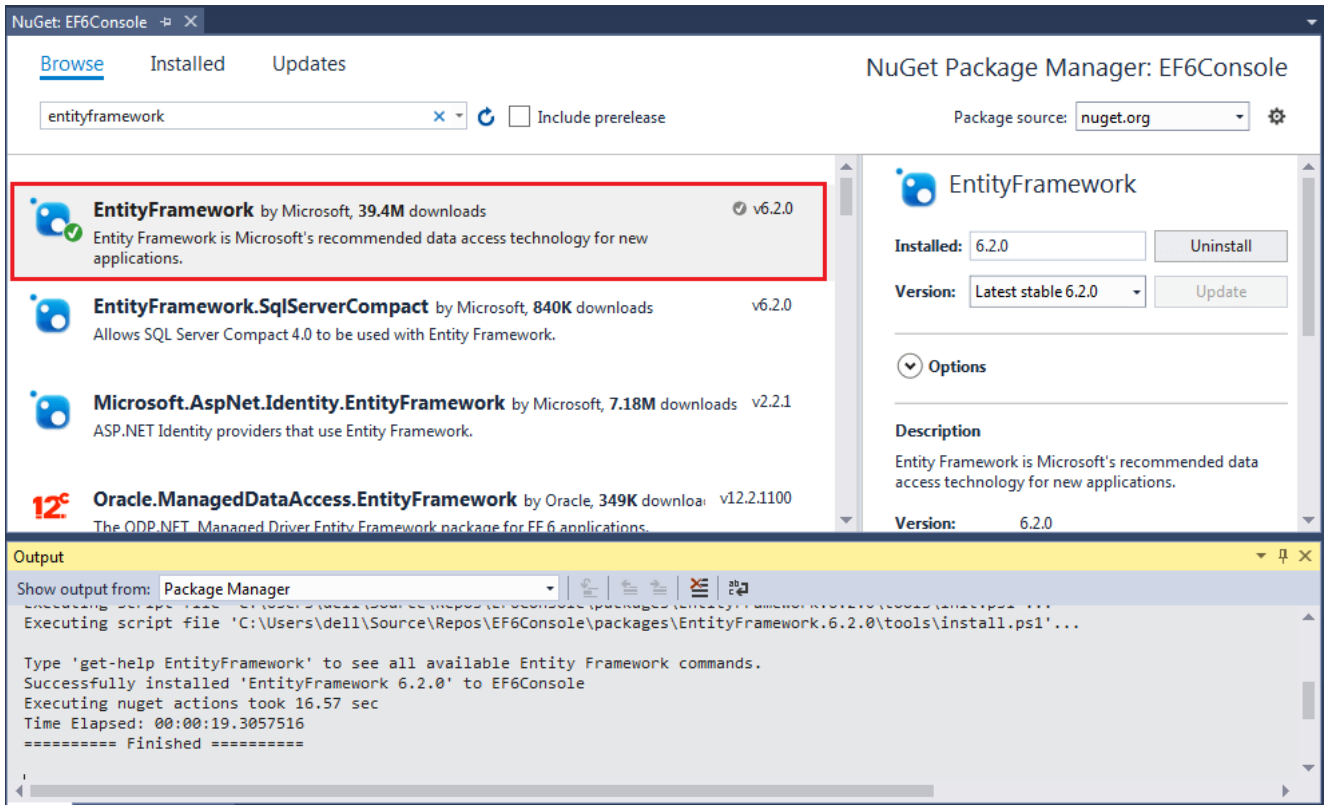


Chọn EntityFramework và click nút **Install**.

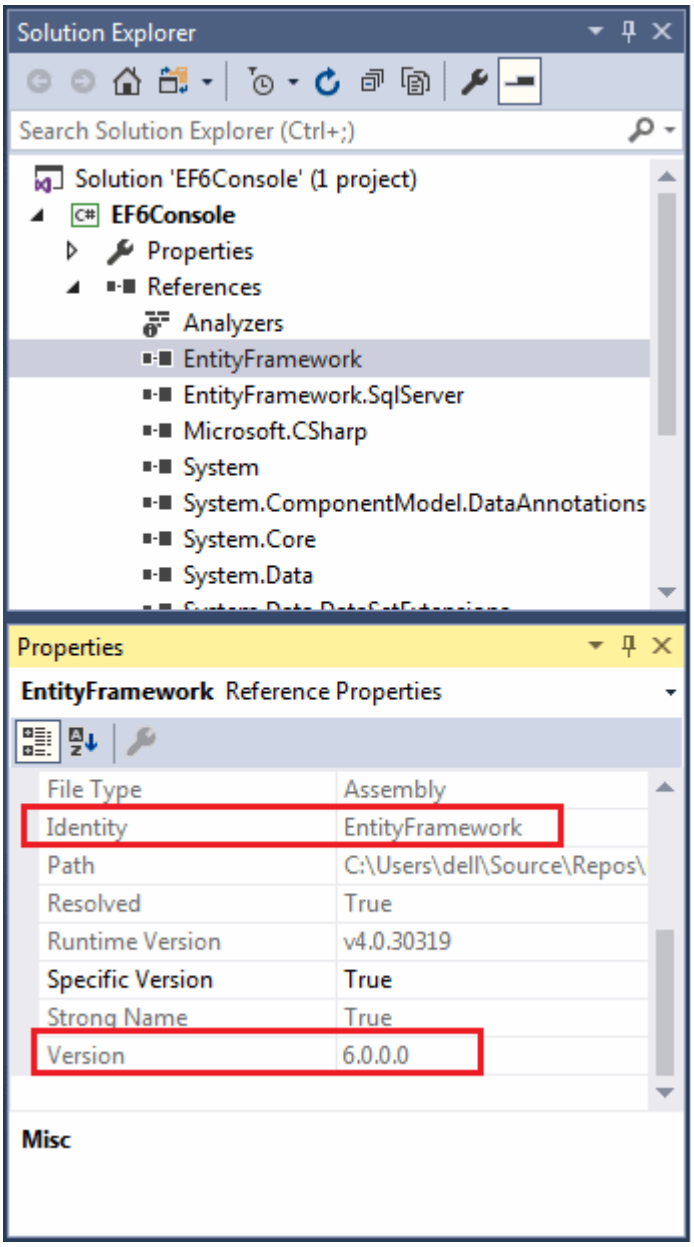




Click **"I Accept"** để đồng ý cài đặt.



After installation, make sure that the appropriate version of EntityFramework.dll is included in the project.



Now, we are ready to use Entity Framework in our project. Let's create our first simple code-first example in the next chapter.

Bước 3. Click chuột phải trên dự án và add Class lần lượt tạo các lớp mới chi tiết như sau:

Teacher.cs

```
public class Teacher
    {
        public Teacher()
        {
        }
```

```
        public int TeacherId { get; set; }
        public string TeacherName { get; set; }
        public Nullable<int> TeacherType { get; set; }

        public Nullable<int> StandardId { get; set; }
        public virtual Standard Standard { get; set; }
    }
```

Standard.cs

```
public class Standard
    {
        public Standard()
        {
        }
        public int StandardId { get; set; }
        public string StandardName { get; set; }
        public string Description { get; set; }

        public virtual ICollection<Teacher> Teachers { get; set; }
        public virtual ICollection<Student> Students { get; set; }
    }
```

Student.cs

```
public class Student
    {
        public Student()
        {
            this.Courses = new HashSet<Course>();
        }
        public int StudentID { get; set; }
        public string StudentName { get; set; }
        public Nullable<int> StandardId { get; set; }
        public virtual Standard Standard { get; set; }

        public virtual StudentAddress StudentAddress { get; set; }

        public virtual ICollection<Course> Courses { get; set; }
    }
```

StudentAddress.cs

```
public class StudentAddress
    {
        [Key]
        [ForeignKey("Student")]
        public int StudentID { get; set; }
        public string Address1 { get; set; }
        public string Address2 { get; set; }
        public string City { get; set; }
        public string State { get; set; }

        public virtual Student Student { get; set; }
    }
```

Course.cs

```
public class Course
    {
        public Course()
        {
            this.Students = new HashSet<Student>();
        }

        public int CourseId { get; set; }
        public string CourseName { get; set; }
        public string Location { get; set; }

        public virtual ICollection<Student> Students { get; set; }
    }
```

Grade.cs

```
public class Grade
    {
        public int GradeId { get; set; }
        public string GradeName { get; set; }
        public string Section { get; set; }

        public ICollection<Student> Students { get; set; }
    }
```

Bước 4. Tạo Context Class

```
public class SchoolContext : DbContext
    {
        public SchoolContext() : base("name=SchoolDb")
        {
            //Database.SetInitializer<SchoolContext>(new DropCreateDatabaseIfModelChanges<SchoolContext>());

            //Database.SetInitializer<SchoolContext>(new DropCreateDatabaseIfModelChanges<SchoolContext>());
            Database.SetInitializer<SchoolContext>(new DropCreateDatabaseAlways<SchoolContext>());
            //Database.SetInitializer<SchoolDBContext>(new SchoolDBInitializer());
        }

        public DbSet<Student> Students { get; set; }
        public DbSet<Grade> Grades { get; set; }
        public DbSet<Teacher> Teachers { get; set; }
        public DbSet<Standard> Standards{ get; set; }
        public DbSet<Course> Courses { get; set; }
    }
```

Bước 5. Thiết lập chuỗi kết nối trong App.config

```
<?xml version="1.0" encoding="utf-8"?>
<configuration>
    <connectionStrings>
        <add name="SchoolDb" connectionString="server=(local)\sqlexpress;database=SchoolDb2;uid=sa;pwd=123456;"
             providerName="System.Data.SqlClient" />
    </connectionStrings>
</configuration>
```

Bước 6. Code-based Migrations

Để tạo CSDL & cập nhật những thay đổi cấu trúc của cơ sở dữ liệu, chúng ta thực hiện các bước sau từ **Package Manager Console** trong Visual Studio:

1. **Enable-Migrations:** bật migration và tạo ra lớp `Configuration`.cs trong dự án
2. **Add-Migration:** tạo ra lớp migration class có 2 phương thức `Up()` và `Down()`. Up dùng để tạo ra các bảng và định nghĩa csdl, Down để xóa csdl nếu chúng ta rollback.
3. **Update-Database:** Thực hiện file migration sau cùng để tạo & thay đổi cấu trúc csdl.

Bước 7. Khởi tạo một số dữ liệu ban đầu

Trong file configuration.cs được tạo ra khi chạy lệnh **enable-migrations, viết lệnh để khởi tạo dữ liệu trong phương thức Seed() như sau**

```
internal sealed class Configuration : DbMigrationsConfiguration<EF6Console.SchoolContext>
    {
        public Configuration()
        {
            AutomaticMigrationsEnabled = false;
        }

        protected override void Seed(EF6Console.SchoolContext context)
        {
            //  This method will be called after migrating to the latest version.

            //  You can use the DbSet<T>.AddOrUpdate() helper extension method
            //  to avoid creating duplicate seed data.
            /*
            */

            var s1 = new Standard() { StandardName = "IT", Description = "Information technology" };
            var s2 = new Standard() { StandardName = "DB", Description = "Database theory" };
            var s3 = new Standard() { StandardName = "GD", Description = "Graphical design" };

            context.Standards.AddOrUpdate(s1);
            context.Standards.AddOrUpdate(s2);
            context.Standards.AddOrUpdate(s3);
            context.SaveChanges();
        }
    }
```

8. Chạy lại lệnh update-database

update-Database

Database Initialization Strategies in EF 6 Code-First

You already created a database after running your Code-First application the first time, but what about the second time onwards? Will it create a new database every time you run the application? What about the production environment? How do you alter the database when you change your domain model? To handle these scenarios, you have to use one of the database initialization strategies.

There are four different database initialization strategies:

1. **CreateDatabaseIfNotExists:** This is the **default** initializer. As the name suggests, it will create the database if none exists as per the configuration. However, if you change the model class and then run the application with this initializer, then it will throw an exception.
2. **DropCreateDatabaseIfModelChanges:** This initializer drops an existing database and creates a new database, if your model classes (entity classes) have been changed. So, you don't have to worry about maintaining your database schema, when your model classes change.
3. **DropCreateDatabaseAlways:** As the name suggests, this initializer drops an existing database every time you run the application, irrespective of whether your model classes have changed or not. This will be useful when you want a fresh database every time you run the application, for example when you are developing the application.
4. **Custom DB Initializer:** You can also create your own custom initializer, if the above do not satisfy your requirements or you want to do some other process that initializes the database using the above initializer.

To use one of the above DB initialization strategies, you have to set the DB Initializer using the `Database` class in a context class, as shown below:

```
public class SchoolDBContext: DbContext
{
    public SchoolDBContext(): base("SchoolDBConnectionString")
    {
       Database.SetInitializer<SchoolDBContext>(new CreateDatabaseIfNotExists<SchoolDBContext>());

       //Database.SetInitializer<SchoolDBContext>(new DropCreateDatabaseIfModelChanges<SchoolDBContext>());
       //Database.SetInitializer<SchoolDBContext>(new DropCreateDatabaseAlways<SchoolDBContext>());
       //Database.SetInitializer<SchoolDBContext>(new SchoolDBInitializer());
    }

    public DbSet<Student> Students { get; set; }
    public DbSet<Standard> Standards { get; set; }
}
```

You can also create your custom DB initializer, by inheriting one of the initializers, as shown below:

```
public class SchoolDBInitializer :  CreateDatabaseIfNotExists<SchoolDBContext>
{
    protected override void Seed(SchoolDBContext context)
    {
        base.Seed(context);
    }
}
```

In the above example, the `SchoolDBInitializer` is a custom initializer class that is derived from `CreateDatabaseIfNotExists`. This separates the database initialization code from a context class.

Set the DB Initializer in the Configuration File

You can also set the db initializer in the configuration file. For example, to set the default initializer in app.config:

```
<?xml version="1.0" encoding="utf-8" ?>
<configuration>
    <appSettings>
    <add key="DatabaseInitializerForType SchoolDataLayer.SchoolDBContext, SchoolDataLayer"

value="System.Data.Entity.DropCreateDatabaseAlways`1[[SchoolDataLayer.SchoolDBContext,
SchoolDataLayer]], EntityFramework" />
    </appSettings>
</configuration>
```

You can set the custom DB initializer, as follows:

```
<?xml version="1.0" encoding="utf-8" ?>
<configuration>
    <appSettings>
    <add key="DatabaseInitializerForType SchoolDataLayer.SchoolDBContext, SchoolDataLayer"
            value="SchoolDataLayer.SchoolDBInitializer, SchoolDataLayer" />
    </appSettings>
</configuration>
```

Turn off the DB Initializer

You can turn off the database initializer for your application. Suppose that you don't want to lose existing data in the production environment, then you can turn off the initializer, as shown below:

```
public class SchoolDBContext: DbContext
{
    public SchoolDBContext() : base("SchoolDBConnectionString")
    {
        //Disable initializer
        Database.SetInitializer<SchoolDBContext>(null);
```

```
    }
    public DbSet<Student> Students { get; set; }
    public DbSet<Standard> Standards { get; set; }
}
```

You can also turn off the initializer in the configuration file, for example:

```
<?xml version="1.0" encoding="utf-8" ?>
<configuration>
    <appSettings>
    <add key="DatabaseInitializerForType SchoolDataLayer.SchoolDBContext, SchoolDataLayer"
            value="Disabled" />
    </appSettings>
</configuration>
```