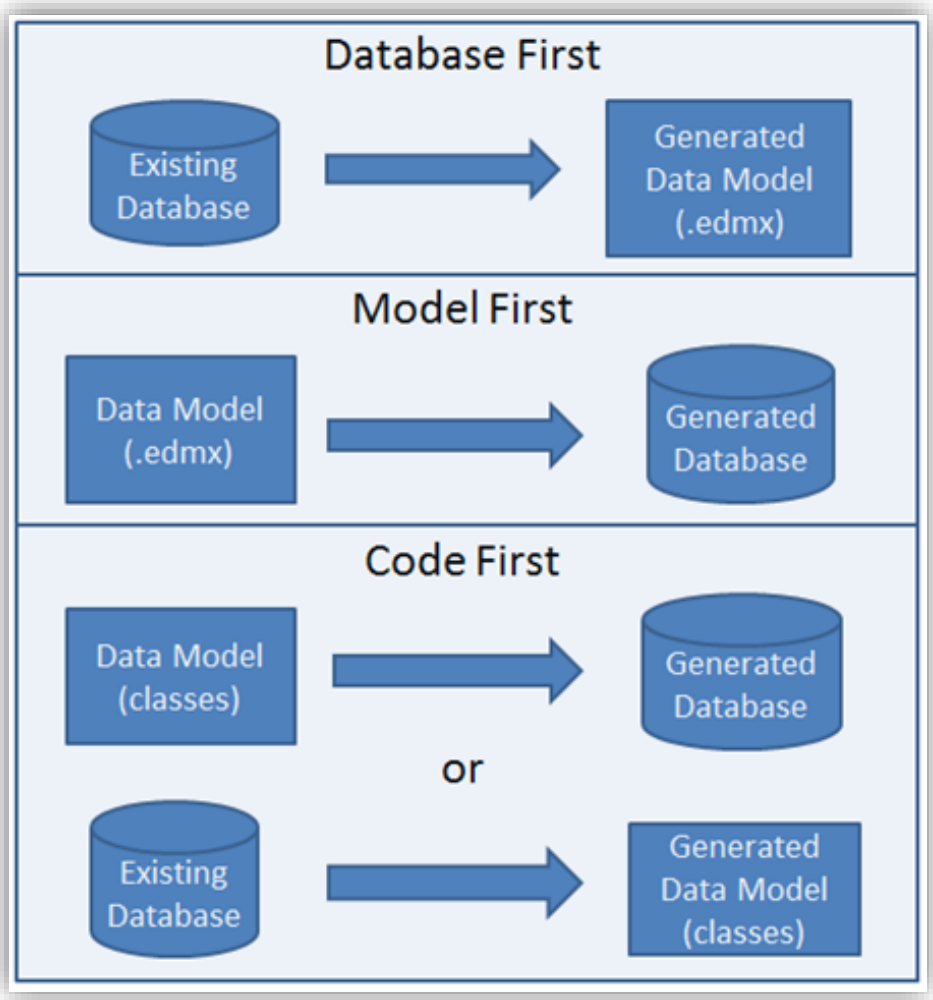
ENTITY FRAMEWORK BASICS: CODE FIRST APPROACH

2017

ENTITY FRAMEWORK APPROACHES




DATA MODEL

ENTITY MODEL

```
// Базовый тип для всех сущностей в приложении
public abstract class BaseEntity {
    public int Id { get; set; } // PRIMARY KEY
}
```

Данный подход не обязателен, но позволит не повторять базовое свойство во всех сущностях.

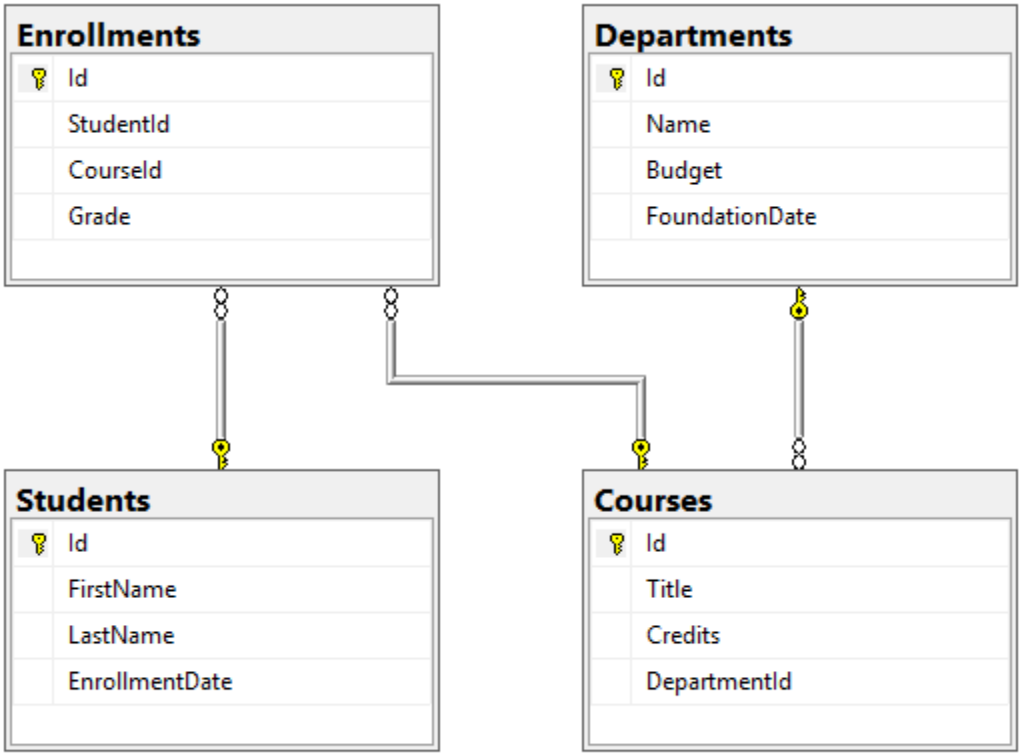
ENTITY MODEL

```
public class Course : BaseEntity {
    public string Title { get; set; }
    public int Credits { get; set; }
    public int DepartmentId { get; set; }

    public virtual Department Department { get; set; }
    public virtual ICollection<Enrollment> Entrollments { get; set; }
}

public class Department : BaseEntity {
    public string Name { get; set; }
    public decimal Budget { get; set; }
    public DateTime FoundationDate { get; set; }

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

ENTITY MODEL

```
public class Student : BaseEntity {
    public string FirstName { get; set; }
    public string LastName { get; set; }
    public DateTime EnrollmentDate { get; set; }

    public virtual ICollection<Enrollment> Enrollments { get; set; }
}

public class Enrollment : BaseEntity {
    public int Grade { get; set; }

    public int StudentId { get; set; }
    public virtual Student Student { get; set; }
    public int CourseId { get; set; }
    public virtual Course Course { get; set; }
}
```

ENTITY MODEL



ENTITY MODEL

```
public class Enrollment : BaseEntity {
    ...
    public int? Grade { get; set; }
    // или
    public Nullable<int> Grade { get; set; }
    ...
}
```

Данный подход позволит сделать поле **Grade** не обязательным.

ENTITY CONTEXT

```
public class SchoolContext : DbContext
{
    public SchoolContext(string nameOrConnectionString)
        : base(nameOrConnectionString)
    { }

    public IDbSet<Student> Students { get; set; }
    public IDbSet<Enrollment> Enrollments { get; set; }
    public IDbSet<Course> Courses { get; set; }
    public IDbSet<Department> Departments { get; set; }

}
```

CONNECTION STRINGS

Connection string позволяет определить желаемый источник данных для **Entity Framework**.
По умолчанию, **Entity Framework** будет искать **connection string** с именем **равным имени контекста**.

```
<configuration>
  . . .
<connectionStrings>
  <add name="SchoolContext"
       connectionString="Data Source=.;
         Initial Catalog=SchoolDb;
         Integrated Security=True"
       providerName="System.Data.SqlClient" />
</connectionStrings>
  . . .
</configuration>
```

DATABASE INITIALIZERS

`CreateDatabaseIfNotExists` — при запуске приложения проверяет наличие базы данных. Если БД отсутствует — создает новую. В случае, если модель данных в БД отличается от модели в приложении — выбрасывает `Exception`. **Используется по-умолчанию**.

`DropCreateDatabaseIfModelChanges` — базовое поведение идентично `CreateDatabaseIfNotExists`. Если модели отличается — пересоздает базу.

`DropCreateDatabaseAlways` — при запуске приложения всегда пересоздает БД.

```
static SchoolContext() {
    Database.SetInitalizer(new CreateDatabaseIfNotExists<SchoolContext>());
}
```

Для инициализации БД данными необходимо создать класс, наследуемый от одного из представленных инициализаторов и переопределить метод **`Seed()`**.

CUSTOM DATABASE INITIALIZER

```
public class SchoolContexttInitializer :
     CreateDatabaseIfNotExists<SchoolContext>
{
     protected override void Seed(SchoolContext context)
     {
          var students = new List<Student>
          {
               new Student {
                    FirstName = "Иван",
                    LastName = "Иванов",
                    EnrollmentDate = DateTime.Parse("2011-07-01")
               },
               new Student {
                    FirstName = "Петр",
                    LastName = "Петров",
                    EnrollmentDate = DateTime.Parse("2012-08-02")
               }
          }

          students.ForEach(std => context.Studens.Add(std));
          context.SaveChanges();
     }
}
```

CODE FIRST CONVETIONS

CODE FIRST CONVENTIONS

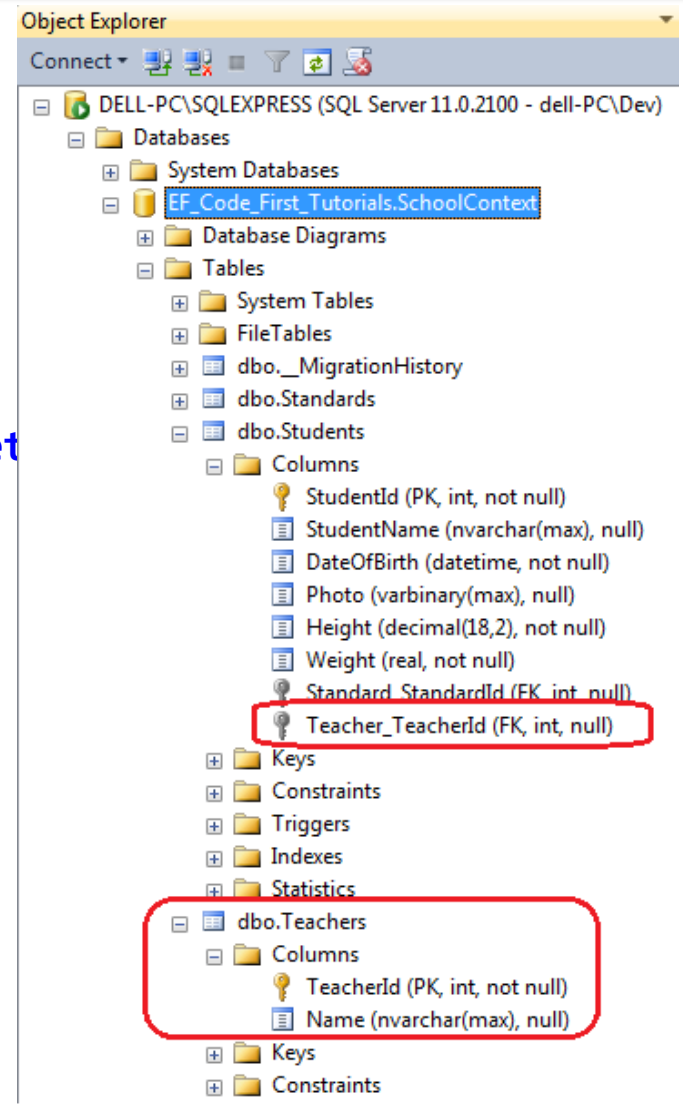
Code-First Conventions — набор базовых правил для автоматической конфигурации модели данных при использовании подхода **Entity Framework Code First**.

```
public class Student {
    public int StudentId { get; set; }
    ...
    public Teacher Teacher { get; set; }
    public Standard Standard { get; set; }
}

public class Teacher {
    public int TeacherId { get; set; }
    public string TeacherName { get; set; }
    public IList<Student> Students { get; set; }
}
```

CODE FIRST CONVENTIONS

```
public class ShoolContext : DbContext
{
    public SchoolContext() : base() { }

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

- **Code-First** подключает все типы, для которых создан `DbSet` в классе-контексте.
- **Code-First** подключает все ссылочные типы, описанные, как свойства в базовых сущностях (тех, для которых в контексте есть `DbSet`).
- **Code-First** подключает все классы-наследники, даже если только для базового класса есть `DbSet`.



PRIMARY KEY CONVENTION

Entity Framework использует свойства с именами **Id** или **<ClassName>Id** в качестве первичного ключа. Тип свойства не имеет значения, но в случае, если свойство имеет **числовой** тип или **GUID** — столбец в базе будет обозначен как **Identity**.

```
public class Standard
{
    public int StdId { get; set; }
    public string StandardName { get; set; }
    public IList<Student> Students { get; set; }
}
```

```
'System.Data.Entity.ModelConfiguration.ModelValidationException' occurred
in EntityFramework.dll
EntityType 'Standard' has no key defined. Define the key for this
EntityType.
```

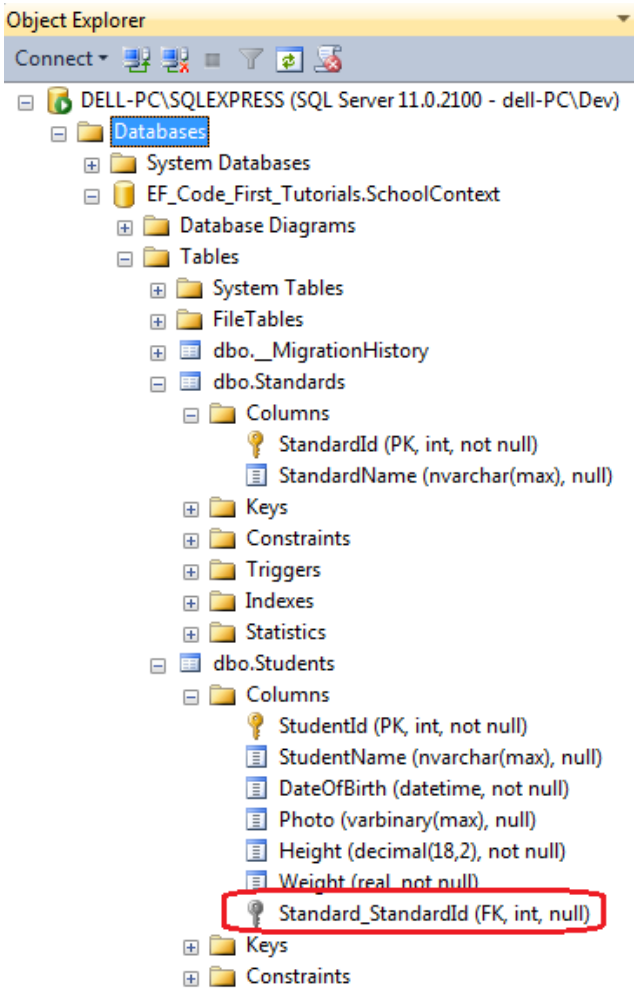
RELATION CONVENTION

Code First использует навигационные свойства для описания отношений между сущностями.

- Навигационное свойство, имеющее тип сущности, описывает отношение **«к одному»**.
- Навигационное свойство, имеющее тип, приводимый к `ICollection`, описывает отношение **«ко многим»**.

```
public class Student {
     ...
     public Standard Standard { get; set; } // навигационное свойство,
                                            //  определяющее отношение «к одному».
}

public class Standard {
     ...
     public IList<Student> Students { get; set; } // навигационное свойство,
                                                  //  определяющее отношение «ко многим».

}
```

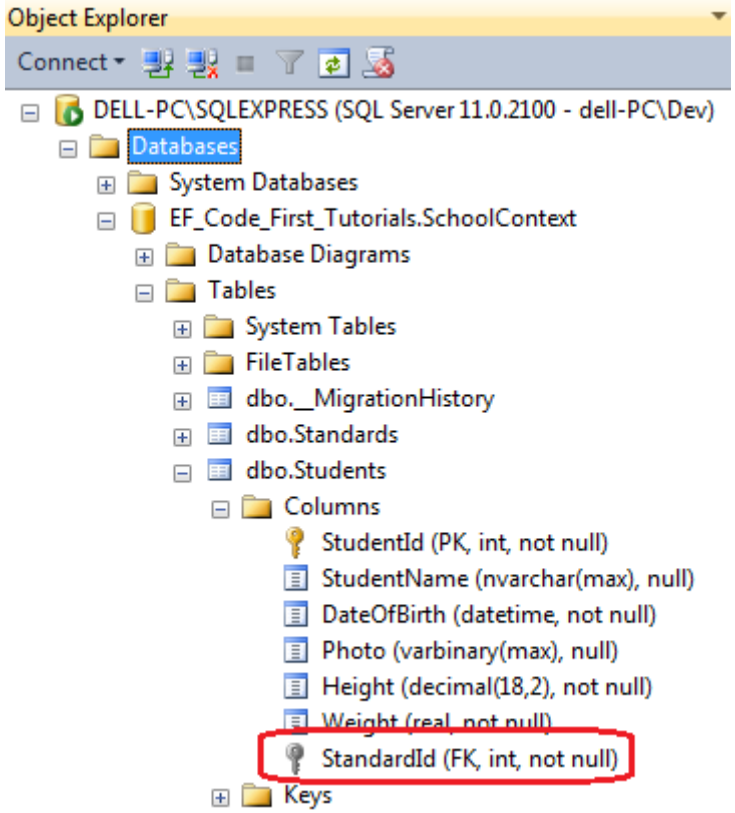
RELATION CONVENTION



FOREIGN KEY CONVETION

Для управления процессом создания внешнего ключа необходимо самостоятельно объявить для него свойство на стороне зависимой сущности:

```
public class Student
{
    ...
    public int StandardId { get; set; }

    public Standard Standard { get; set; }
}
```



DATA ANNOTATIONS

```
public class Standard // Throws an Exception
{
    public int StdId { get; set; }
    ...
}

// преобразовать в

using System.ComponentModel.DataAnnotations;

public class Standard // Will not throw Exception because KEY is configured
{
    [Key]
    public int StdId { get; set; }
    ...
}
```

DATA ANNOTATIONS

System.ComponentModel.DataAnnotations Attributes:

Атрибут	Описание
`[Key]`	Помечает свойство сущности как первичный ключ.
`[Timestamp]`	Помечает свойство сущности как авто-генерируемое timestamp-поле.
`[Required]`	Помечает поле, как обязательное. Актуально только для ссылочных типов.
`[MinLength]`	Определяет минимальную длину массива или строки.
`[MaxLength]`	Определяет максимальную длину массива или сткроки.
`[StringLength]`	Позволяет определить минимальную и максимальную длину для строки.

DATA ANNOTATIONS

System.ComponentModel.DataAnnotations.Schema Attributes:

Атрибут	Описание
[Table]	Позволяет переопределить имя таблицы для сущности.
[Column]	Позволяет переопределить имя колонки для свойства сущности.
[ForeignKey]	Позволяет пометить свойство сущности как внешний ключ.
[NotMapped]	Отключает генерирование колонки для свойства.
[ComplexType]	Помечает ссылочный тип как Complex Type.

FLUENT API

Fluent API — еще один механизм конфигурирования модели данных. **Fluent API** предоставляет большую гибкость и функциональность, чем DataAnnotations.

Fluent API поддерживает следующие типы конфигураций:

Конфигурация	Возможности
На уровне всей модели данных **(Data Model Configuration)**	• Указание базовой схемы данных.
На уровне сущности **(Entity Configuration)**	• Привязка к одной или нескольким таблицам, а также указание схемы. • Приведение типа к Complex Type. • Модификация модели наследования.
На уровне свойства сущности **(Property Configuration)**	• Привязка к колонке • Указание имени колонки, типа колонки • Nullable or Not NULL Column • Приведение к внешнему ключу (Foreign key). • Конфигурирование отношений

FLUENT API

```
public class SchoolContext : DbContext
{
    ...
    protected override void OnModelCreating(DbModelBuilder modelBuilder) {
        // Configure domain classes using modelBuilder here

        base.OnModelCreating(modelBuilder);
    }
}
```

FLUENT API. ENTITY CONFIGURATION

Метод	Описание
`HasKey<TKey>()`	Указание свойства, которое будет использоваться как первичный ключ.
`HasMany<TTargetEntity>()`	Указание отношения «ко многим».
`HasOptional<TTargetEntity>()`	Указание **необязательного** отношения «к одному».
`HasRequired<TTargetEntity>()`	Указание **обязательного** отношения «к одному».
`Ignore<TProperty>()`	Игнорирование своейства при генерации таблицы.
`Property<TProperty>()`	Переход к конфигурации свойства сущности.
`ToTable()`	Указание имени таблицы для сущности.

FLUENT API. ENTITY CONFIGURATION

```
protected override void OnModelCreating(DbModelBuilder modelBuilder) {
     // конфигурирование имени таблиц для Student и Standard
     modelBuilder.Entity<Student>().ToTable("StudentInfo");
     modelBuilder.Entity<Standard>().ToTable("StandardInfo");

     // конфигурирование первичных ключей для Student и Standard
     modelBuilder.Entity<Student>().HasKey(s => s.StudentId);
     modelBuilder.Entity<Standard>().HasKey(s => s.StdId);

     // конфигурирование обязательного отношение Student к Teacher
     modelBuilder.Entity<Student>().HasRequired(s => s.Teacher);

     // конфигурирование отношения «один ко многим» для Student и Standard
     modelBuilder.Entity<Standard>().HasMany(s => s.Students)
          .WithRequired(s => s.Standard);
}
```

FLUENT API. PROPERTY CONFIGURATION

```
protected override void OnModelCreating(DbModelBuilder modelBuilder) {
     // конфигурирование свойства DateOfBirth для Student
     modelBuilder.Entity<Student>()
          .Property(s => s.DateOfBirth)
          .HasConlumnName("DoB")
          .HasColumnOrder(3)
          .HasColumnType("datetime2");

     // указание максимальной длины свойства TeacherName для Teacher
     modelBuilder.Entity<Teacher>()
          .Property(t => t.TeacherName)
          .HasMaxLength(50);
}
```

DATA LOAD STRATEGY

DATA LOAD STRATEGY. LAZY LOADING

Одним из самых важных механизмов
loading означает отсрочивание загруз
произойдет непосредственное обраще

```
SELECT    [Extent1].[Id] AS [Id],
          [Extent1].[Title] AS [Title],
          [Extent1].[Credits] AS [Credits],
FROM [dbo].[Courses] AS [Extent1]
```

```
using (var ctx = new SchoolContext())
{
    var courses = ctx.Courses.ToList();

    var course = courses[0];

    var department = course.Department;
}
```

```
exec sp_executesql N'SELECT
[Extent1].[Name] AS [Name],
[Extent1].[Budget] AS [Buget],
[Extent1].[FoundationDate] AS [FoundationDate]
FROM [dbo].[Departments] as [Extent1]
WHERE [Extent1].[Id] = @EntityKeyValue1
',N'@EntityKeyValue1 int',@EntityKeyValue1=1
```

DATA LOAD STRATEGY. EAGER LOADING

Eager loading — процесс **загрузки сущности** из БД в **пределах запроса на другую** сущность. Выполняется посредству использования метода **`Include()`**.

```
using (var ctx = new SchoolContext())
{
    var students = ctx.Students.Include("Courses").ToList();
    // или
    var students = ctx.Students.Include(s => s.Courses).ToList();
}
```

QUESTIONS?

THANKS FOR ATTENTION!