# Entity Framework Core - Fluent API & Data Modeling Attributes

This document combines **Data Annotation Attributes** and **Fluent API Methods** for Entity Framework Core, complete with definitions, examples, and when to use each.

## 🧩 DATA MODELING ATTRIBUTES

### **1. [Key]**

Marks a property as the primary key.

public class Student  
{  
 [Key]  
 public int Id { get; set; }  
}

### **2. [Required]**

Makes a property mandatory (cannot be null).

public class Student  
{  
 [Required]  
 public string Name { get; set; }  
}

### **3. [MaxLength] / [StringLength]**

Limits the maximum length of a string.

[StringLength(50)]  
public string Name { get; set; }  
  
[MaxLength(20)]  
public string Code { get; set; }

### **4. [Column]**

Specifies column name or data type in database.

[Column("Full\_Name", TypeName = "varchar(100)")]  
public string Name { get; set; }

### **5. [Table]**

Specifies the database table name for an entity.

[Table("StudentsTable")]  
public class Student { ... }

### **6. [ForeignKey(“NavigationOrFkProperty”)]**

Explicitly defines a foreign key relationship.

public class Employee  
{  
 public int EmployeeId { get; set; }  
 public string Name { get; set; }  
  
 [ForeignKey("Department")]  
 public int DeptId { get; set; }  
  
 public Department Department { get; set; }  
}

Used when EF cannot infer the relationship automatically or when multiple relationships exist between the same entities.

### **7. [InverseProperty(“OtherNav”)]**

Used to specify which navigation property on the other side of the relationship it corresponds to.

public class Teacher  
{  
 public int Id { get; set; }  
 public string Name { get; set; }  
  
 [InverseProperty("PrimaryTeacher")]  
 public ICollection<Student> PrimaryStudents { get; set; }  
  
 [InverseProperty("AssistantTeacher")]  
 public ICollection<Student> AssistantStudents { get; set; }  
}  
  
public class Student  
{  
 public int Id { get; set; }  
  
 [ForeignKey("PrimaryTeacher")]  
 public int PrimaryTeacherId { get; set; }  
  
 [ForeignKey("AssistantTeacher")]  
 public int AssistantTeacherId { get; set; }  
  
 public Teacher PrimaryTeacher { get; set; }  
 public Teacher AssistantTeacher { get; set; }  
}

### **8. [ConcurrencyCheck]**

Marks a property to be checked for concurrency conflicts.

[ConcurrencyCheck]  
public string Email { get; set; }

### **9. [Timestamp]**

Defines a binary column used for concurrency control.

[Timestamp]  
public byte[] RowVersion { get; set; }

Used to detect when multiple users edit the same record simultaneously.

### **10. [DataType] and [DisplayFormat]**

Formats how data is displayed and stored.

[DataType(DataType.Date)]  
[DisplayFormat(DataFormatString = "{0:yyyy-MM-dd}", ApplyFormatInEditMode = true)]  
public DateTime DateOfBirth { get; set; }

### **11. [Display(Name = “…”)]**

Changes how the property name appears in UI (e.g., forms or views).

[Display(Name = "Student Name")]  
public string Name { get; set; }

**Effect in UI (e.g., Razor form):**

<label asp-for="Name"></label> <!-- Renders: Student Name -->

# 🧩 FLUENT API METHODS

Defined inside OnModelCreating(ModelBuilder modelBuilder) in your DbContext.

protected override void OnModelCreating(ModelBuilder modelBuilder)  
{  
 // Fluent API configurations here  
}

### **Entity Configuration**

modelBuilder.Entity<Student>().ToTable("StudentsTable"); // Rename table  
modelBuilder.Entity<Student>().Property(s => s.Name).HasColumnName("FullName"); // Rename column

### **Keys and Identity**

modelBuilder.Entity<Student>().HasKey(s => s.Id); // Primary key  
modelBuilder.Entity<Enrollment>().HasKey(e => new { e.StudentId, e.CourseId }); // Composite key  
modelBuilder.Entity<Student>().Property(s => s.Id).ValueGeneratedOnAdd(); // Auto increment

### **Property Configuration**

modelBuilder.Entity<Student>().Property(s => s.Name).IsRequired(); // Required  
modelBuilder.Entity<Student>().Property(s => s.Name).HasMaxLength(100); // Max length  
modelBuilder.Entity<Student>().Property(s => s.IsActive).HasDefaultValue(true); // Default value  
modelBuilder.Entity<Student>().Property(s => s.DateOfBirth).HasColumnType("date"); // SQL type

### **Relationships**

#### One-to-Many

modelBuilder.Entity<Department>()  
 .HasMany(d => d.Employees)  
 .WithOne(e => e.Department)  
 .HasForeignKey(e => e.DeptId);

#### One-to-One

modelBuilder.Entity<Person>()  
 .HasOne(p => p.Passport)  
 .WithOne(ps => ps.Person)  
 .HasForeignKey<Passport>(ps => ps.PersonId);

#### Many-to-Many

modelBuilder.Entity<Student>()  
 .HasMany(s => s.Courses)  
 .WithMany(c => c.Students)  
 .UsingEntity(j => j.ToTable("Enrollments"));

#### Self-Referencing

modelBuilder.Entity<Employee>()  
 .HasOne(e => e.Manager)  
 .WithMany(m => m.Subordinates)  
 .HasForeignKey(e => e.ManagerId);

### **Indexes and Constraints**

modelBuilder.Entity<Student>().HasIndex(s => s.Email).IsUnique(); // Unique index  
modelBuilder.Entity<Student>().HasAlternateKey(s => s.RollNumber); // Alternate key

### **Concurrency and Timestamp**

modelBuilder.Entity<Student>()  
 .Property(s => s.RowVersion)  
 .IsRowVersion()  
 .IsConcurrencyToken();

### **Ignore or Exclude**

modelBuilder.Entity<Student>().Ignore(s => s.TempCalculation); // Ignore property  
modelBuilder.Ignore<TemporaryLog>(); // Ignore entity

### **Computed Columns**

modelBuilder.Entity<Employee>()  
 .Property(e => e.FullName)  
 .HasComputedColumnSql("[FirstName] + ' ' + [LastName]");

## 🧠 Fluent API vs Data Annotations

| Feature | Data Annotations | Fluent API |
| --- | --- | --- |
| Style | Attribute-based | Code-based |
| Location | Inside entity class | Inside OnModelCreating |
| Scope | Limited | Complete control |
| Priority | Lower | ✅ Fluent API overrides |

## 🧩 Common Fluent API Categories

| Category | Methods |
| --- | --- |
| **Entity** | ToTable, HasKey, Ignore, HasIndex |
| **Property** | HasColumnName, HasMaxLength, IsRequired, HasDefaultValue, HasColumnType, ValueGeneratedOnAdd |
| **Relationships** | HasOne, HasMany, WithOne, WithMany, HasForeignKey, UsingEntity |
| **Constraints** | HasAlternateKey, IsUnique, IsConcurrencyToken, IsRowVersion |
| **Advanced** | HasComputedColumnSql, OwnsOne, OwnsMany, HasCheckConstraint |

## 🧩 Example: Complete Fluent API Setup

public class SchoolContext : DbContext  
{  
 public DbSet<Student> Students { get; set; }  
 public DbSet<Course> Courses { get; set; }  
  
 protected override void OnModelCreating(ModelBuilder modelBuilder)  
 {  
 modelBuilder.Entity<Student>().ToTable("Students");  
  
 modelBuilder.Entity<Student>()  
 .Property(s => s.Name)  
 .IsRequired()  
 .HasMaxLength(100);  
  
 modelBuilder.Entity<Course>()  
 .HasMany(c => c.Students)  
 .WithMany(s => s.Courses)  
 .UsingEntity(j => j.ToTable("Enrollments"));  
  
 modelBuilder.Entity<Student>().HasIndex(s => s.Email).IsUnique();  
 modelBuilder.Entity<Student>().Ignore(s => s.TempField);  
 }  
}

✅ **Best Practice Summary** - Use **Data Annotations** for simple configurations. - Use **Fluent API** for complex relationships and fine-grained control. - Group Fluent API settings logically: Keys → Properties → Relationships. - Prefer IEntityTypeConfiguration<T> classes for clean, modular configuration.