Validation & Data-Modeling Attributes (EF Core / ASP.NET Core)

A compact reference of common **validation** (`DataAnnotations`) and **data-modeling** attributes used with EF Core and ASP.NET Core. Each attribute includes a short explanation and a copy-paste example.

Quick example model (uses several attributes)

```
using System;
using System.ComponentModel.DataAnnotations;
using System.ComponentModel.DataAnnotations.Schema;
using Microsoft.EntityFrameworkCore; // for [Precision], [Index]

[Table("Students")]
[Index(nameof(Email), IsUnique = true)]
public class Student
{
    [Key]
    [DatabaseGenerated(DatabaseGeneratedOption.Identity)]
    public int Id { get; set; }

    [Required(ErrorMessage = "Name is required")]
    [StringLength(100, MinimumLength = 2)]
    public string Name { get; set; } = "";

    [EmailAddress]
    [Required]
    public string Email { get; set; } = "";

    [Phone]
    public string? PhoneNumber { get; set; }

    [Range(16, 120)]
    public int Age { get; set; }

    [Column(TypeName = "decimal(10,2)")]
    // Or for EF Core 6+: [Precision(10,2)]
    public decimal Fee { get; set; }

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

```
    public ICollection<Course> Courses { get; set; } = new List<Course>();

    [NotMapped]
    public string TempNote { get; set; } = "";
}
```

Validation Attributes (System.ComponentModel.DataAnnotations)

[Required]

Purpose: Property must be provided (non-null / non-empty for strings in model binding).

```
[Required]
public string Name { get; set; }
```

[StringLength(max, MinimumLength = min)]

Purpose: Limit string length and optionally set minimum length.

```
[StringLength(100, MinimumLength = 2)]
public string Title { get; set; }
```

[MaxLength] / [MinLength]

Purpose: Influence schema and validation for arrays/strings (MaxLength affects column size in EF).

```
[MaxLength(50)]
public string ShortCode { get; set; }
```

[Range(min, max)]

Purpose: Ensure numeric or date values fall within a range.

```
[Range(1, 100)]
public int Quantity { get; set; }
```

[RegularExpression("pattern")]

Purpose: Validate a string with a regex.

```
[RegularExpression(@"^[A-Z]{3}-\d{4}$", ErrorMessage = "Code must be like
ABC-1234")]
public string Code { get; set; }
```

[EmailAddress], [Phone], [Url], [CreditCard]

Purpose: Common format validators.

```
[EmailAddress]
public string Email { get; set; }

[Phone]
public string? Phone { get; set; }
```

[Compare("OtherProperty")]

Purpose: Compare two properties (e.g., Password and ConfirmPassword).

```
[Compare("Password")]
public string ConfirmPassword { get; set; }
```

[DataType(DataType.Date)], [DisplayFormat(...)]

Purpose: Provide UI/display hints and formatting.

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

[Display(Name = "...")]

Purpose: Friendly display name for UI and error messages.

```
[Display(Name = "Full name")]
public string Name { get; set; }
```

[CustomValidation(typeof(ValidatorType), "MethodName")]

Purpose: Hook a static validation method.

```
public static class MyValidators
{
    public static ValidationResult ValidateAge(object value, ValidationContext
ctx)
    {
        if (value is int age && age >= 18) return ValidationResult.Success!;
        return new ValidationResult("Must be adult");
    }
}

[CustomValidation(typeof(MyValidators), "ValidateAge")]
public int Age { get; set; }
```

Creating custom validation attributes

Purpose: Implement domain-specific rules by inheriting `ValidationAttribute`.

```
public class NotPastDateAttribute : ValidationAttribute
{
    public override bool IsValid(object? value)
    {
        if (value is DateTime dt) return dt.Date >= DateTime.UtcNow.Date;
        return true;
    }
}

[NotPastDate(ErrorMessage = "Start date can't be in the past")]
public DateTime StartDate { get; set; }
```

Data-Modeling Attributes (affect EF Core mapping)

> Note: Some advanced constructs (composite keys, many complex indexes, conditional mappings) require the Fluent API (`OnModelCreating`).

[Key]

Purpose: Mark primary key.

```
[Key]
public int Id { get; set; }
```

[DatabaseGenerated(DatabaseGeneratedOption.Identity | Computed | None)]

Purpose: Configure value generation strategy.

```
[DatabaseGenerated(DatabaseGeneratedOption.Identity)]
public int Id { get; set; }
```

[Column("ColumnName")] / [Column(TypeName = "...")]

Purpose: Custom column name or SQL type.

```
[Column("first_name")]
public string FirstName { get; set; }

[Column(TypeName = "decimal(18,2)")]
public decimal Price { get; set; }
```

[Table("TableName")]

Purpose: Custom table name.

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

[NotMapped]

Purpose: Exclude property from EF model / DB schema.

```
[NotMapped]
public string TempNote { get; set; }
```

[ForeignKey("NavigationOrFkProperty")]

Purpose: Specify which property is the FK or target nav for a FK property.

```
public class Order
{
    public int CustomerId { get; set; }

    [ForeignKey("CustomerId")]
    public Customer Customer { get; set; }
}
```

Or to point at a navigation:

```
[ForeignKey("Customer")]
public int CustomerRefId { get; set; }
```

[InverseProperty("OtherNav")]

Purpose: Disambiguate multiple relationships between the same two entity types.

```
public class Person
{
    public int Id { get; set; }

    [InverseProperty("Manager")]
    public ICollection<Person> DirectReports { get; set; }

    [InverseProperty("DirectReports")]
    public Person? Manager { get; set; }
}
```

[ConcurrencyCheck]

Purpose: Mark a property for optimistic concurrency checks.

```
[ConcurrencyCheck]
public int Version { get; set; }
```

[Timestamp]

Purpose: Special concurrency token; typically `byte[]` / `rowversion` in SQL Server.

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

[Index(...)] (Microsoft.EntityFrameworkCore)

Purpose: Create an index (including uniqueness). Requires EF Core attribute support (or use Fluent API).

```
[Index(nameof(Email), IsUnique = true)]
public class User { public string Email { get; set; } }
```

[Precision(precision, scale)] (EF Core 6+)

Purpose: Specify decimal precision & scale (alternative to `Column(TypeName=...)`).

```
[Precision(10, 2)]
public decimal Amount { get; set; }
```

Patterns & Examples

Explicit join table with extra fields (Enrollment)

```
public class Enrollment
{
    public int StudentId { get; set; }
```

```
    public Student Student { get; set; } = null!;

    public int CourseId { get; set; }
    public Course Course { get; set; } = null!;

    [Required]
    public DateTime EnrolledOn { get; set; }

    [Range(0,100)]
    public int? Grade { get; set; }
}

// OnModelCreating (Fluent API):
modelBuilder.Entity<Enrollment>().HasKey(e => new { e.StudentId, e.CourseId });
```

Unique index (Fluent API alternative)

```
modelBuilder.Entity<Student>()
    .HasIndex(s => s.Email)
    .IsUnique();
```

Validating objects manually (outside MVC model binding)

```
var student = new Student { Name = "", Email = "bad" };
var context = new ValidationContext(student);
var results = new List<ValidationResult>();
bool valid = Validator.TryValidateObject(student, context, results,
validateAllProperties: true);

foreach (var r in results)
    Console.WriteLine(r.ErrorMessage);
```

When to use Attributes vs Fluent API

- **Attributes**: quick and readable; suitable for common constraints and validations. Good for small/medium projects.
- **Fluent API**: required for advanced mapping (composite keys, conditional indexes, complex relationships, table splitting, value conversions). Preferred when you want all DB mapping centralized.

Short reference table (attributes grouped)

- **Validation:** `[Required]`, `[StringLength]`, `[MaxLength]`, `[MinLength]`, `[Range]`, `[RegularExpression]`, `[EmailAddress]`, `[Phone]`, `[Url]`, `[Compare]`, `[CustomValidation]`, `ValidationAttribute` subclasses
- **Data Modeling:** `[Key]`, `[DatabaseGenerated]`, `[Column]`, `[Table]`, `[NotMapped]`, `[ForeignKey]`, `[InverseProperty]`, `[ConcurrencyCheck]`, `[Timestamp]`, `[Index]`, `[Precision]`

Final notes

- Composite keys *cannot* be configured with attributes; use Fluent API: `HasKey(...)`.
- Some attributes (e.g. `[Index]`, `[Precision]`) require EF Core version that exposes them (EF Core 5/6+). When in doubt, prefer Fluent API for portability.

End of file.