Include / Exclude von Entities

- Convention
  - DbSet-Property im Context
  - Indirekt via Navigation Property
- Fluent API
  - Entry im Model Builder
  - Ignore im Model Builder
- Data Annotations

```
public class ShopContext : DbContext
{
    public DbSet<Category> Categories { get; set; }

    protected override void OnModelCreating(
        ModelBuilder modelBuilder)
    {
        modelBuilder.Entity<AuditEntry>();
        modelBuilder.Ignore<Metadata>();
    }
}

public class Category
{
    public int Id { get; set; }
    public string Name { get; set; }
    public ICollection<Product> Products { get; set; }
    public ICollection<Metadata> Metadata { get; set; }
}
public class Product { /* ... */ }
public class AuditEntry { /* ... */ }
[NotMapped]
public class Metadata { /* ... */ }
```


OST

Include / Exclude von Properties

- Convention
  - Alle public Properties mit Getter/Setter
- Fluent API
  - Ignore im Model Builder
- Data Annotations

```
public class ShopContext : DbContext
{
    public DbSet<Category> Categories { get; set; }

    protected override void OnModelCreating(
        ModelBuilder modelBuilder)
    {
        modelBuilder.Entity<Category>()
            .Property(b => b.Name);
        modelBuilder.Entity<Category>()
            .Ignore(b => b.LoadedFromDatabase);
    }
}

public class Category
{
    public int Id { get; set; }
    public string Name { get; set; }
    [NotMapped]
    public DateTime LoadedFromDatabase { get; set; }
}
```

Keys

- Convention
  - Property mit dem Namen «[Entity]Id»
  - Beispiel 1: Category.Id
  - Beispiel 2: Category.CategoryId
- Fluent API
  - Einzige Möglichkeit für «Composite Keys»
- Data Annotations

```
public class ShopContext : DbContext
{
    public DbSet<Category> Categories { get; set; }

    protected override void OnModelCreating(
        ModelBuilder modelBuilder)
    {
        modelBuilder.Entity<Category>()
            .HasKey(e => e.Id);
    }
}
public class Category
{
    [Key]
    public int Id { get; set; }
    public string Name { get; set; }
}
public class Tanslation
{
    public string Language { get; set; }
    public int CategoryId { get; set; }
}
```

Required / Optional

- Convention
  - Value Types werden «NOT NULL» (int)
  - Nullable Value Types werden «NULL» (int?)
  - Reference Types werden «NULL»
- Fluent API
- Data Annotations

```
public class ShopContext : DbContext
{
    public DbSet<Category> Categories { get; set; }

    protected override void OnModelCreating(
        ModelBuilder modelBuilder)
    {
        modelBuilder.Entity<Category>()
            .Property(e => e.Name)
            .IsRequired();
    }
}
public class Category
{
    public int Id { get; set; }
    [Required]
    public string Name { get; set; }
    public bool? IsActive { get; set; }
}
```


OST

Maximum Length

- Convention
  - Keine Restriktion / z.B. NVARCHAR(MAX)
  - 450 Zeichen bei Keys
- Fluent API
- Data Annotations

```
public class ShopContext : DbContext
{
    public DbSet<Category> Categories { get; set; }

    protected override void OnModelCreating(
        ModelBuilder modelBuilder)
    {
        modelBuilder.Entity<Category>()
            .Property(e => e.Name)
            .HasMaxLength(500);
    }
}
public class Category
{
    public int Id { get; set; }
    [MaxLength(500)]
    public string Name { get; set; }
    public bool? IsActive { get; set; }
}
```


OST

Indexes

- Convention
  - Werden bei Foreign Keys automatisch erstellt
- Fluent API
  - Non-unique Index
  - Unique Index
  - Multi-column Index
- Data Annotations
  - Non-unique Index
  - Unique Index
  - Multi-column Index

```
public class ShopContext : DbContext
{
    public DbSet<Category> Categories { get; set; }

    protected override void OnModelCreating(
        ModelBuilder modelBuilder)
    {
        modelBuilder.Entity<Category>()
            .HasIndex(b => b.Name);
        modelBuilder.Entity<Category>()
            .HasIndex(b => b.Name)
            .IsUnique();
        modelBuilder.Entity<Category>()
            .HasIndex(b => new { b.Name, b.IsActive });
    }
}
[Index(nameof(Name))]
[Index(nameof(Name), IsUnique = true)]
[Index(nameof(Name), nameof(IsActive))]
public class Category
{
    public int Id { get; set; }
    public string Name { get; set; }
    public bool? IsActive { get; set; }
}
```


OST

Entity Type Configuration

- Nachteile der Fluent API
  - Viel Text
  - Unstrukturiert
- Entity Type Configuration
  - Eine Mapping Configuration pro Entity Type
  - Fluent API identisch wie in «OnModelCreating»
  - Registrierung der Entity Type Configuration via «modelBuilder.ApplyConfiguration(…)»

```
internal class CategoryTypeConfig
    : IEntityTypeConfiguration<Category>
{
    public void Configure(
        EntityTypeBuilder<Category> builder)
    {
        builder
            .Property(p => p.Timestamp)
            .IsRowVersion();
    }
}
public class ShopContext : DbContext
{
    protected override void OnModelCreating(
        ModelBuilder modelBuilder)
    {
        modelBuilder.ApplyConfiguration(
            new CategoryTypeConfig()
        );
    }
}
```


OST

Tabellen

- Convention
  - Tabellenname = DbSet-Name
  - Beispiel: dbo.Categories

```
public class ShopContext : DbContext
{
    public DbSet<Category> Categories { get; set; }

    protected override void OnModelCreating(
        ModelBuilder modelBuilder)
    {
        modelBuilder.Entity<Category>()
            .ToTable("Category", schema: "dbo");
    }
}
```

- Fluent API
  - Name der Tabelle zwingend
  - Name des Schemas optional

- Data Annotations
  - Name der Tabelle zwingend
  - Name des Schemas optional

```
[Table("Category", Schema = "dbo")]
public class Category
{
    public int Id { get; set; }
    public string Name { get; set; }
}
```


OST

Spalten

- Convention
  - Spaltenname = Property-Name
  - Beispiel: Name
- Fluent API

```
public class ShopContext : DbContext
{
    public DbSet<Category> Categories { get; set; }

    protected override void OnModelCreating(
        ModelBuilder modelBuilder)
    {
        modelBuilder.Entity<Category>()
            .Property(e => e.Name)
            .HasColumnName("CategoryName", order: 1);
    }
}
```

- Data Annotations

```
public class Category
{
    public int Id { get; set; }
    [Column("CategoryName", Order = 1)]
    public string Name { get; set; }
}
```


OST

Datentypen / Default Values

- Convention
  - Siehe Anhang: Data Type Mappings
  - Keine Default Values
- Fluent API
  - Datentyp-Name des Zielsystems
  - Default (Wert / Gültige SQL Expression)
- Data Annotations
  - Datentyp-Name des Zielsystems
  - Default Values nicht unterstützt

```
public class ShopContext : DbContext
{
    public DbSet<Category> Categories { get; set; }

    protected override void OnModelCreating(
        ModelBuilder modelBuilder)
    {
        modelBuilder.Entity<Category>()
            .Property(e => e.Name)
            .HasColumnName("CategoryName")
            .HasColumnType("NVARCHAR(500)")
            .HasDefaultValue("---");
    }
}

public class Category
{
    public int Id { get; set; }
    [Column("CategoryName", TypeName = "NVARCHAR(500)"]
    public string Name { get; set; }
}
```


OST

Relationship – one-to-many / Fully Defined

- Convention
  - ☑ Collection Navigation Property (1-Ende)
  - ☑ Reference Navigation Property (N-Ende)
  - ☑ Foreign Key Property
- Fluent API
  - HasOne / WithMany ODER
  - HasMany / WithOne
- Data Annotations
  - Auf Navigation Property wird Foreign Key Property definiert

```
public class ShopContext : DbContext
{
    public DbSet<Category> Categories { get; set; }

    protected override void OnModelCreating(
        ModelBuilder modelBuilder)
    {
        modelBuilder.Entity<Product>()
            .HasOne(p => p.Category)
            .WithMany(b => b.Products)
            .HasForeignKey(p => p.CategoryId)
            .HasConstraintName("FK_Product_CategoryId");
    }
}

public class Category
{
    public int Id { get; set; }
    public ICollection<Product> Products { get; set; }
}
public class Product
{
    public int Id { get; set; }
    public int CategoryId { get; set; }
    [ForeignKey(nameof(CategoryId))]
    public Category Category { get; set; }
}
```


OST

Relationship – one-to-many / Shadow Foreign Key

- Convention
  - ☑ Collection Navigation Property (1-Ende)
  - ☑ Reference Navigation Property (N-Ende)
  - ☐ Foreign Key Property
- Fluent API
  - HasOne / WithMany ODER
  - HasMany / WithOne
- Data Annotations
  - Foreign Key weggelassen

```
public class ShopContext : DbContext
{
    public DbSet<Category> Categories { get; set; }

    protected override void OnModelCreating(
        ModelBuilder modelBuilder)
    {
        modelBuilder.Entity<Product>()
            .HasOne(p => p.Category)
            .WithMany(b => b.Products)
            .HasForeignKey(p => p.CategoryId)
            .HasConstraintName("FK_Product_CategoryId");
    }
}

public class Category
{
    public int Id { get; set; }
    public ICollection<Product> Products { get; set; }
}
public class Product
{
    public int Id { get; set; }
    public int CategoryId { get; set; }
    [ForeignKey(nameof(CategoryId))]
    public Category Category { get; set; }
}
```


OST

Relationship – one-to-many / Single Navigation Property

- Convention
  - ☑ Collection Navigation Property (1-Ende)
  - ☐ Reference Navigation Property (N-Ende)
  - ☐ Foreign Key Property
- Fluent API
  - HasOne / WithMany ODER
  - HasMany / WithOne
- Data Annotations
  - Foreign Key weggelassen
  - Navigation Property weglassen

```
public class ShopContext : DbContext
{
    public DbSet<Category> Categories { get; set; }

    protected override void OnModelCreating(
        ModelBuilder modelBuilder)
    {
        modelBuilder.Entity<Product>()
            .HasOne<Category>()
            .WithMany(b => b.Products)
            .HasForeignKey(p => p.CategoryId)
            .HasConstraintName("FK_Product_CategoryId");
    }
}

public class Category
{
    public int Id { get; set; }
    public ICollection<Product> Products { get; set; }
}
public class Product
{
    public int Id { get; set; }
    public int CategoryId { get; set; }
    [ForeignKey(nameof(CategoryId))]
    public Category Category { get; set; }
}
```


OST

Relationship – one-to-many / Foreign Key

- Convention
  - ☐ Collection Navigation Property (1-Ende)
  - ☐ Reference Navigation Property (N-Ende)
  - ☑ Foreign Key Property
- Fluent API
  - HasOne / WithMany ODER
  - HasMany / WithOne
- Data Annotations
  - Foreign Key weggelassen
  - Navigation Property weglassen

```
public class ShopContext : DbContext
{
    public DbSet<Category> Categories { get; set; }

    protected override void OnModelCreating(
        ModelBuilder modelBuilder)
    {
        modelBuilder.Entity<Product>()
            .HasOne<Category>()
            .WithMany()
            .HasForeignKey(p => p.CategoryId)
            .HasConstraintName("FK_Product_CategoryId");
    }
}

public class Category
{
    public int Id { get; set; }
    public ICollection<Product> Products { get; set; }
}
public class Product
{
    public int Id { get; set; }
    public int CategoryId { get; set; }
    [ForeignKey(nameof(CategoryId))]
    public Category Category { get; set; }
}
```


OST

Relationship – many-to-many / Ohne Join Entity Type

- Konzeptionelles Modell


```
public class ShopContext : DbContext
{
    public DbSet<Category> Categories { get; set; }

    protected override void OnModelCreating(
        ModelBuilder modelBuilder)
    {
        modelBuilder.Entity<Category>()
            .HasMany(p => p.Products)
            .WithMany(b => b.Categories);
    }
}
```

```
public class Category
{
    public int Id { get; set; }
    public ICollection<Product> Products { get; set; }
}
public class Product
{
    public int Id { get; set; }
    public ICollection<Category> Categories{ get; set; }
}
```

Relationship – many-to-many / Ohne Join Entity Type

- Convention
  - Wird automatisch erkannt
  - Mapping-Tabelle wird automatisch generiert
- Fluent API
  - HasMany / WithMany
  - Ausgehend von Category oder Product
- Data Annotations
  - Wird nicht unterstützt

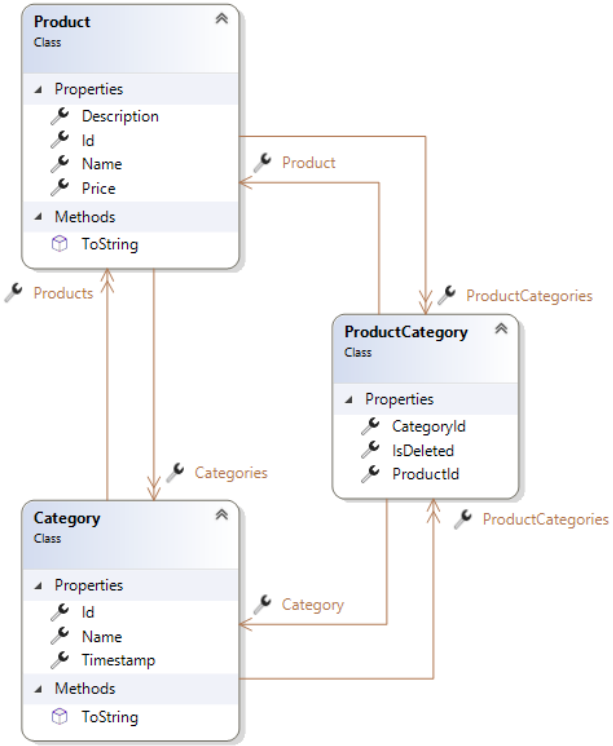
```
public class ShopContext : DbContext
{
    public DbSet<Category> Categories { get; set; }

    protected override void OnModelCreating(
        ModelBuilder modelBuilder)
    {
        modelBuilder.Entity<Category>()
            .HasMany(p => p.Products)
            .WithMany(b => b.Categories);
    }
}

public class Category
{
    public int Id { get; set; }
    public ICollection<Product> Products { get; set; }
}
public class Product
{
    public int Id { get; set; }
    public ICollection<Product> Products { get; set; }
}
```

Relationship – many-to-many / Mit Join Entity Type

- Konzeptionelles Modell




```
public class Category
{
    public int Id { get; set; }
    public ICollection<Product> Products { get; set; }
    public ICollection<ProductCategory> ProductCategories
        { get; set; }
}
public class Product
{
    public int Id { get; set; }
    public ICollection<Category> Categories { get; set; }
    public ICollection<ProductCategory> ProductCategories
        { get; set; }
}

public class ProductCategory
{
    public int ProductId { get; set; } // Optional
    public Product Product { get; set; }
    public int CategoryId { get; set; } // Optional
    public Category Category { get; set; }

    // Payload Property
    public bool IsDeleted { get; set; } = false;
}
```

Relationship – many-to-many / Mit Join Entity Type

- Convention / Data Annotations
  - Wird nicht unterstützt
- Fluent API
  - HasMany / WithMany
  - Ausgehend von Category oder Product

```
// Model Builder
modelBuilder.Entity<Category>()
    .HasMany(c => c.Products)
    .WithMany(p => p.Categories)
    .UsingEntity<ProductCategory>(
        // Right part - ProductCategory > Product
        pc => pc
            .HasOne(e => e.Product)
            .WithMany(e => e.ProductCategories)
            .HasForeignKey(e => e.ProductId), // Optional
        // Left part - ProductCategory > Category
        pc => pc
            .HasOne(e => e.Category)
            .WithMany(e => e.ProductCategories)
            .HasForeignKey(e => e.CategoryId) // Optional
    );
```

```
public class Category
{
    public int Id { get; set; }
    public ICollection<Product> Products { get; set; }
    public ICollection<ProductCategory> ProductCategories
        { get; set; }
}
public class Product
{
    public int Id { get; set; }
    public ICollection<Category> Categories { get; set; }
    public ICollection<ProductCategory> ProductCategories
        { get; set; }
}

public class ProductCategory
{
    public int ProductId { get; set; } // Optional
    public Product Product { get; set; }
    public int CategoryId { get; set; } // Optional
    public Category Category { get; set; }

    // Payload Property
    public bool IsDeleted { get; set; } = false;
}
```