Mapping Real-World Classes in EF Core 6

Introducing Our Real World Classes

Torben Jensen
Developer/Cloud Architect

Version Check

This version was created by using:

- EF Core 6
- .NET 6
- Visual Studio 2022 Community Edition
- C# 10
- SQL Server 2019
- SQL Server Management Studio 18

Not Applicable

This course is NOT applicable to:

- EF Core 1.0 – EF Core 5.0

CARVED ROCK FITNESS

Online retail shop

EF Core

SQL Server

About the Demos

Course assets

Facilitate teaching

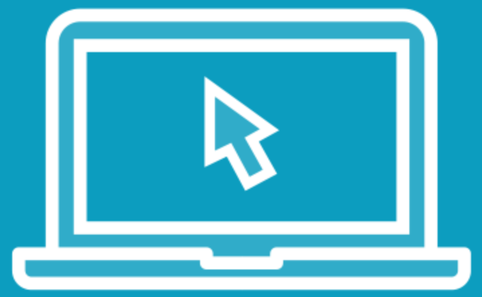
LINQ to Entities

Know the Fundamentals

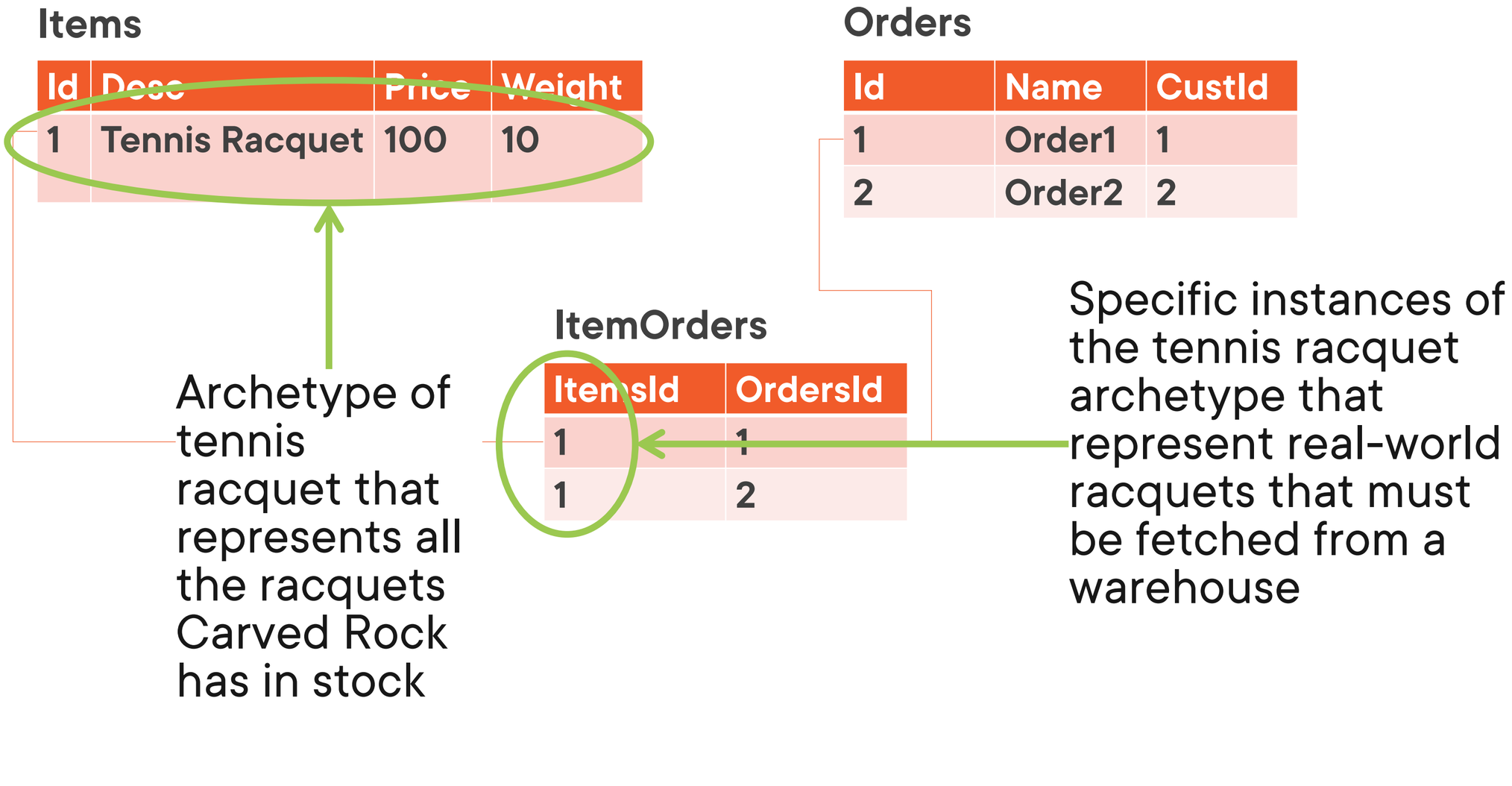
Entity Framework Core 6 Fundamentals

Julie Lerman

Demo



Getting to know the data model


Items
Id
Desc
Price
Weight
1
Tennis Racquet
100
10
Orders
Id
Name
CustId
1
Order1
1
2
Order2
2
ItemOrders
ItemsId
OrdersId
1
1
1
2
Archetype of tennis racquet that represents all the racquets Carved Rock has in stock
Specific instances of the tennis racquet archetype that represent real-world racquets that must be fetched from a warehouse

Demo

Inferring the data model

Code First Migrations

https://docs.microsoft.com/en-us/ef/core/managing-schemas/migrations

Items

Id	Desc	Price	Weight
1	Shoe	100	10
2	Laces	20	1

Orders

Id	Name	CustId
1	Order1	1
2	Order2	2

ItemOrders

ItemsId	OrdersId
1	1
2	1
2	1
1	2
2	2

Skip Navigations

```
public class Order
{
    0 references
    public Order()
    {
        Items = new HashSet<Item>();
    }

    1 reference
    public int Id { get; set; }

    [Required, MaxLength(50)]
    0 references
    public string Name { get; set; }
    0 references
    public int CustomerId { get; set; }
    0 references
    public Customer Customer { get; set; }

    1 reference
    public ICollection<Item> Items { get; set; }
}
```

```
6   public class Item
7   {
        0 references
8       public Item()
9       {
10          Orders = new HashSet<Order>();
11      }
12
        1 reference
13      public int Id { get; set; }
14
15      [Required, MaxLength(255)]
        0 references
16      public string Description { get; set; }
17
18      [Column("UnitPrice", TypeName = "decimal(22,2)")]
        1 reference
19      public decimal Price { get; set; }
20
21      [Column("UnitPriceAfterVAT", TypeName = "decimal(22
        1 reference
22      public decimal PriceAfterVat { get; set; }
23
24      [Column("UnitWeight", TypeName = "float(36)")]
        0 references
25      public float Weight { get; set; }
26
        1 reference
27      public ICollection<Order> Orders { get; set; }
28
29  }
```

```
7 references
public class Order
{
    0 references
    public Order()
    {
        Items = new HashSet<Item>();
    }

    1 reference
    public int Id { get; set; }

    [Required, MaxLength(50)]
    0 references
    public string Name { get; set; }
    0 references
    public int CustomerId { get; set; }
    0 references
    public Customer Customer { get; set; }

    1 reference
    public ICollection<Item> Items { get; set; }
}
```

```
5 references
public class Item
{
    0 references
    public Item()
    {
        Orders = new HashSet<Order>();
    }

    1 reference
    public int Id { get; set; }

    [Required, MaxLength(255)]
    0 references
    public string Description { get; set; }

    [Column("UnitPrice", TypeName = "decimal(22,2)")]
    1 reference
    public decimal Price { get; set; }

    [Column("UnitPriceAfterVAT", TypeName = "decimal(22
    1 reference
    public decimal PriceAfterVat { get; set; }

    [Column("UnitWeight", TypeName = "float(36)")]
    0 references
    public float Weight { get; set; }

    1 reference
    public ICollection<Order> Orders { get; set; }

}
```

Fluent API vs. Data Annotations

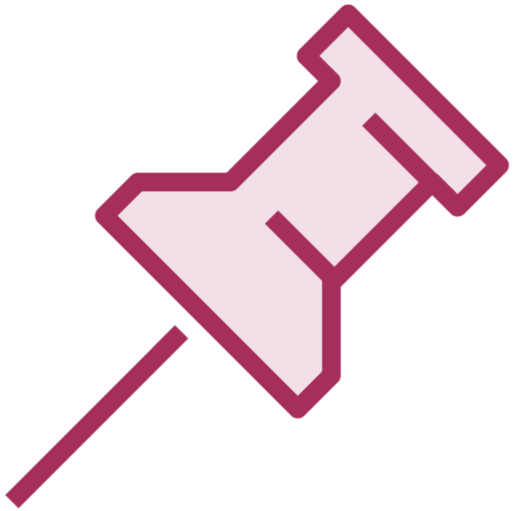
Fluent API	Data Annotations
Implemented in DbContext using fluent notation	Use [attribute] notation directly in your entity classes
Can be complex to use	Very easy to use
Keeps entity classes clean	May clutter entity classes
Larger featureset	Smaller featureset
Works when you can't modify entity classes	Only works when you have direct control over entity classes

Demo

Configuring the database

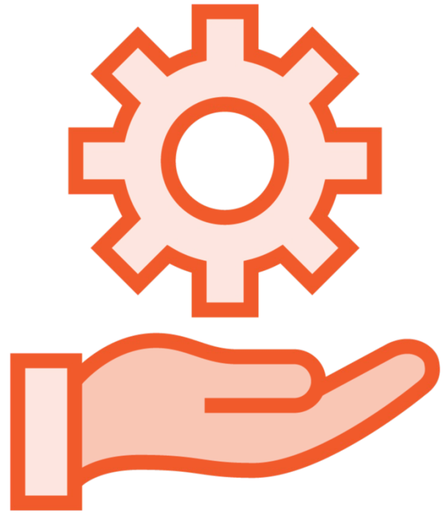
- Required fields
- Max length
- Precision
- Column type

What Did We Just Do?



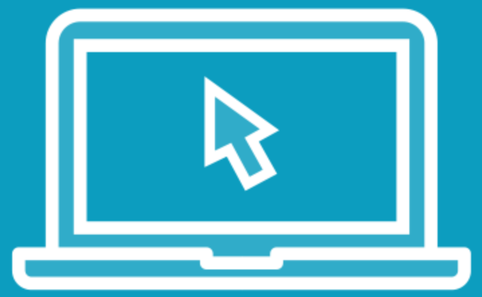
Attributes

Overengineering



Fluent API

Demo



Configuring the database

- Composite keys
- Alternate keys
- Default values
- Computed columns
- Backing fields

```
entity.Property(e =>
     e.Username).HasField("_validUsername");
```

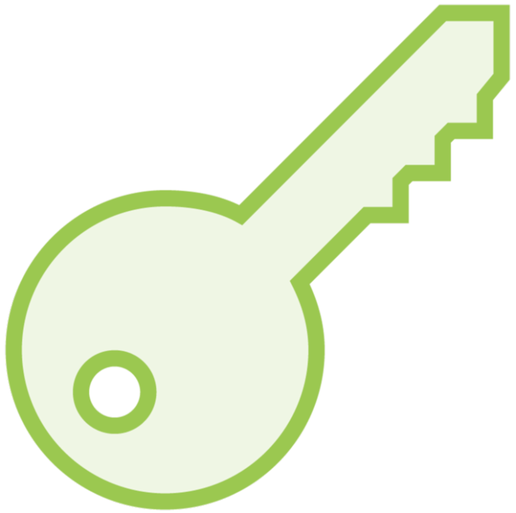
```
public string Username
{
    get
    {
        return _username;
    }
}
```

- Manually map backing field

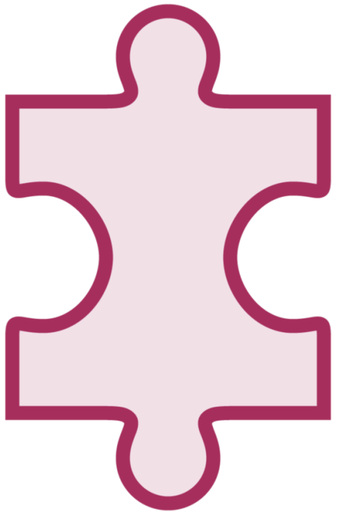
- EF Core will recognize this backing field by convention

What Did We Just Do?

Fluent API



Set up keys



Business logic

Summary

- **Attributes**
- **Fluent API**
- **Database design impacts everything**
- **Preserve flexibility**