Mapping DDD Domain Models with EF

Julie Lerman

MOST TRUSTED AUTHORITY ON ENTITY FRAMEWORK

@julielerman thedatafarm.com

Overview

Pros & Cons of Mapping Domain with EF

Revisit Rich Domain Models in Solution

How will EF Mappings react to patterns?

Private Setters and Constructors

Private Collections & Hidden Properties

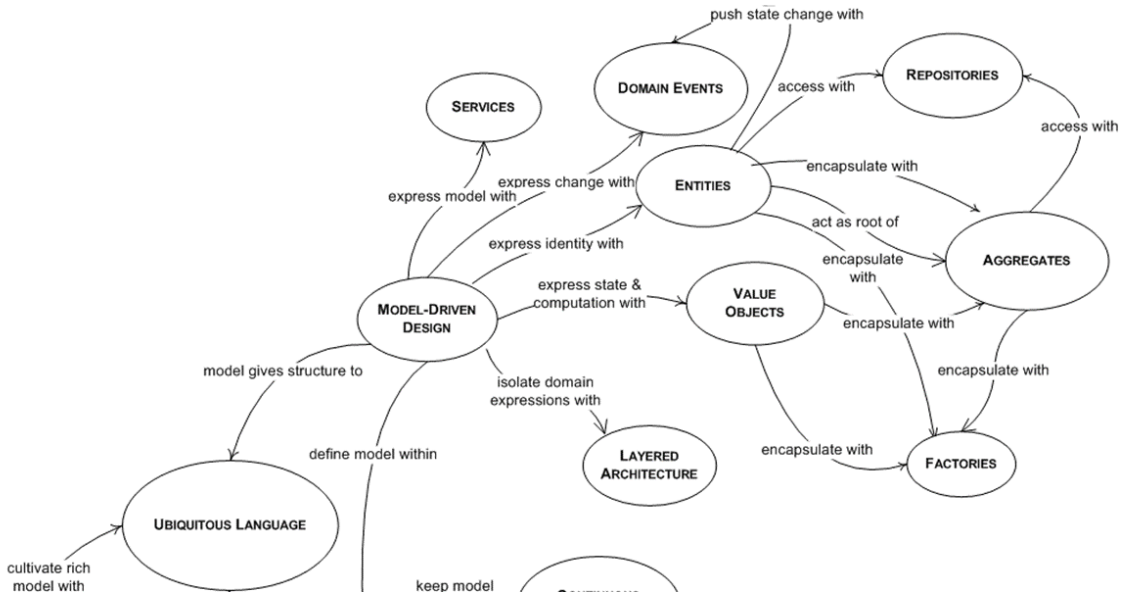
Favoring One-Way Navigations

Value Objects over 1:1 Relationships

Avoiding some of EF Relationship Magic

Consider CQRS Pattern for Some Models

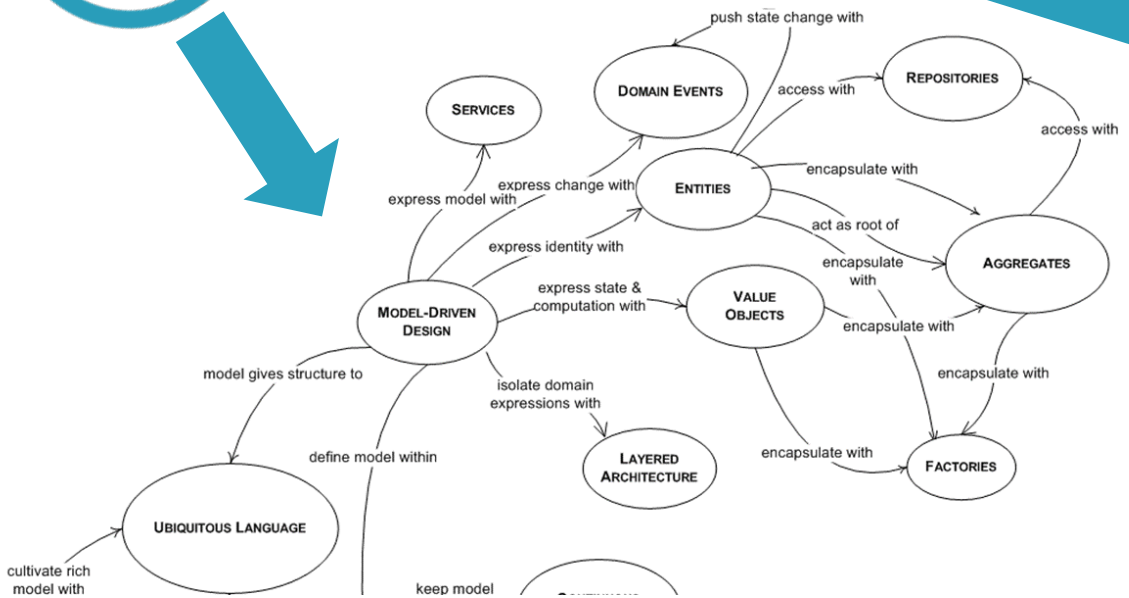
Domain Modeling Does Not Involve Persistence



Model the Domain

Persist the Data

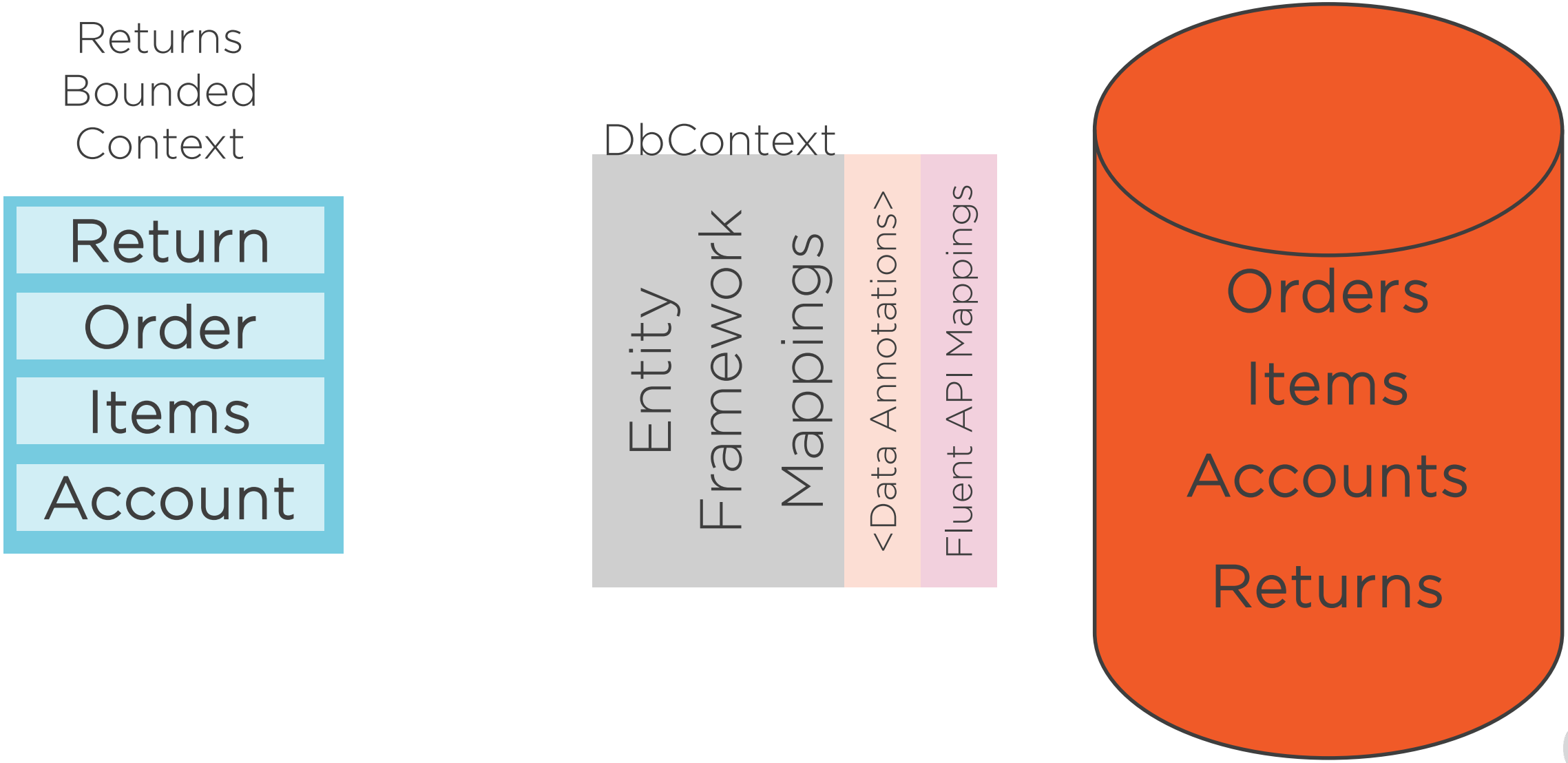
Bigger Picture Considerations



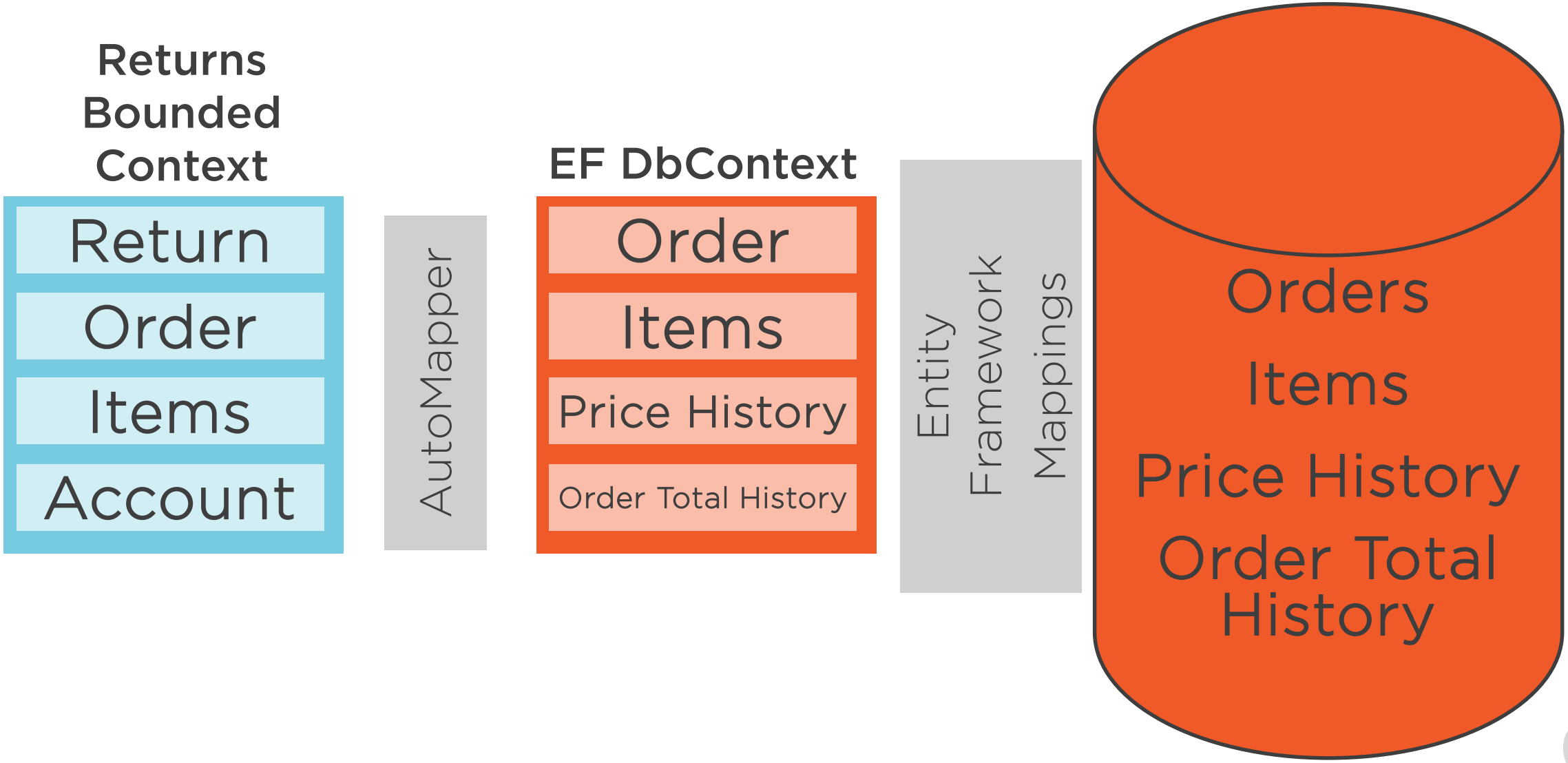
Model the Domain

Persist the Data

Some Times EF Mappings Are Enough



Other Times You'll Need a Separate Data Model



DDD Prefers

Rich Domain Models over Anemic Models

Rich Domain Model

```
public class NewCart
{
  public int CartId { get; private set; }
  public string CartCookie { get; private set; }
  public DateTime Initialized { get; private set; }
  public DateTime Expires { get; private set; }
  public string SourceUrl { get; private set; }
  public int CustomerId { get; private set; }
  public ICollection<CartItem> CartItems { get; private set; }

  public static NewCart CreateCartFromProductSelection(. . .)
  { . . . }

  private NewCart(string sourceUrl, string customerCookie)
  { . . . }

  private void InsertNewCartItem(int productId, int quantity, . . . )
  {
    CartItems.Add(CartItem.Create(
                  productId, quantity, displayedPrice, CartCookie));
  }
```

Anemic Model

```
public class Address
{
  public int AddressId { get; set; }
  public string Street { get; set; }
  public string City { get; set; }
  public string StateProvince { get; set; }
  public string PostalCode { get; set; }
  public AddressType AddressType { get;set;}
  public int CustomerId { get; set; }
  public Customer Customer { get; set; }
}
```

Aggregate

An AGGREGATE is a cluster of associated objects that we treat as a unit for the purpose of data changes.

Aggregate Root

The root is the only member of the AGGREGATE that outside objects are allowed to hold references to, although objects within the boundary may hold references to each other.

Eric Evans, Domain-Driven Design

Private Setters

Aggregate Roots

One Way Navigations

& Other DDD Patterns

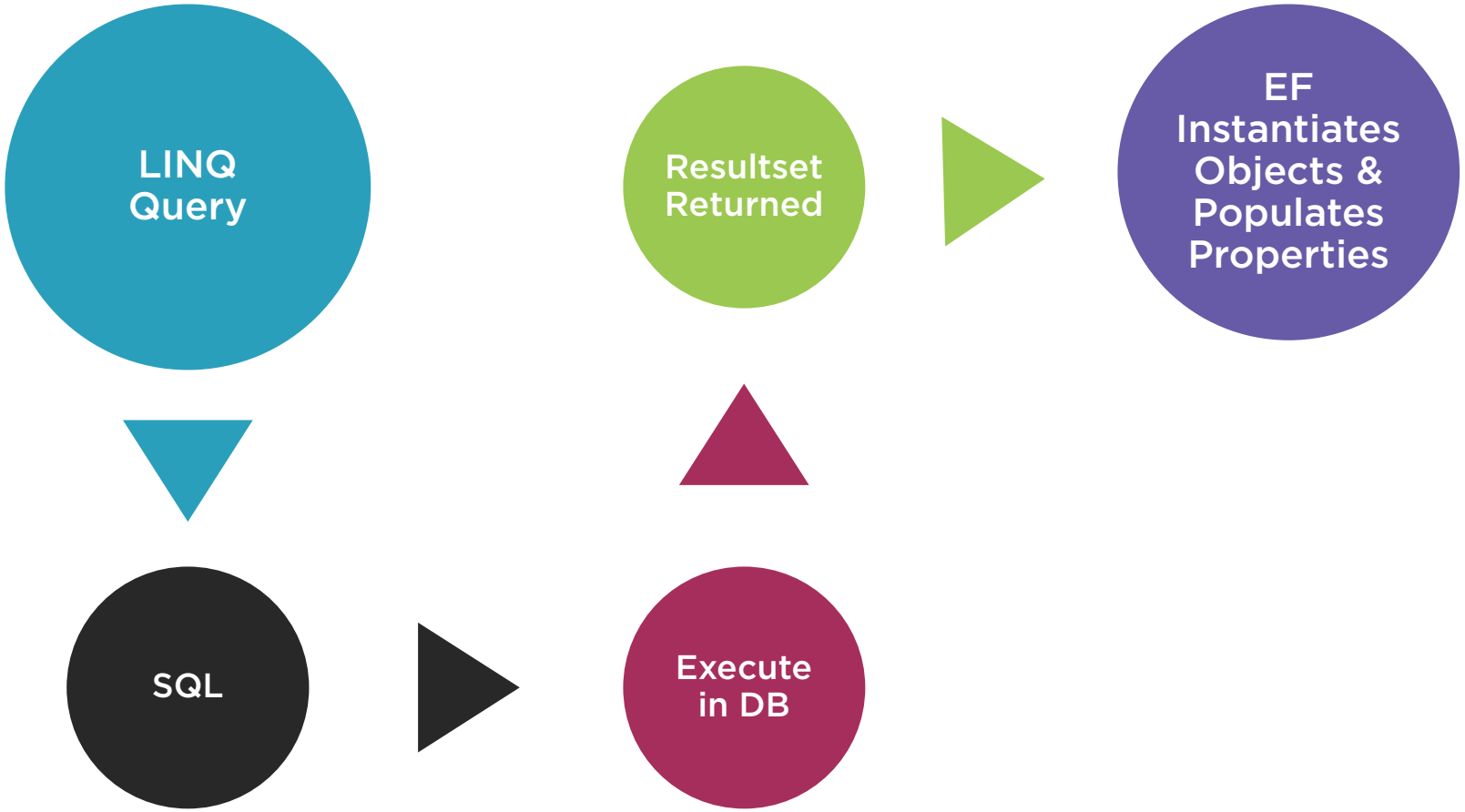
Can EF persist this stuff?

DDD Prefers

Private Setters & Constructors

how to persist?

EF Query Execution



EF Query Execution

```
 public int CartId { get; private set; }
 public string CartCookie { get; private set; }
 public DateTime Initialized { get; private set; }
 public DateTime Expires { get; private set; }


public static NewCart CreateCartFromProductSelection
  (string sourceUrl, string customerCookie,
   int productId, int quantity,decimal displayedPrice){
    var cart = new NewCart(sourceUrl, customerCookie);
    cart.InitializeCart();
    cart.InsertNewCartItem(productId, quantity, displayedPrice);
    return cart;
}

private NewCart(string sourceUrl, string customerCookie){
   ...
}
```

EF Instantiates Objects & Populates Properties

Reflection

Entity Framework Requires a Parameterless Constructor to Materialize Entities

- All classes have parameterless constructor by default
- If you create a custom constructor, then you have to add aparameterless ctor
- Only needed if you are asking EF to materialize a full entity

EF Cannot Handle

Private Collections

Private Properties

Hidden Collections

Great for constraining my aggregate root

Not so great for Entity Framework queries!

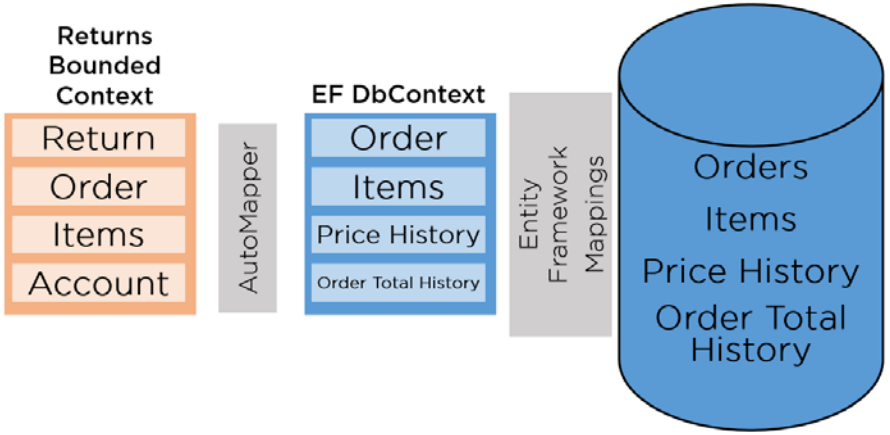
Works with EF
but watch out for CartItems!

```
public int CartId { get; private set; }
public string CartCookie { get; private set; }
public DateTime CartCookieExpires { get; private set; }
public ICollection<CartItem> CartItems;

private int _totalItems;
public int TotalItems => _totalItems;

public CartItem InsertNewCartItem
  (int productId, int quantity, decimal displayedPrice) {
  var item = CartItem.Create
    (productId, quantity, displayedPrice, CartId);
  CartItems.Add(item);
  UpdateItemCount();
  return item;
}
```

Other times, you'll need
a separate data model



DDD Prefers

One Way Navigations

A bidirectional association means that both objects can be understood only together.

When application requirements do not call for traversal in both directions, adding a traversal direction reduces interdependence and simplifies the design."

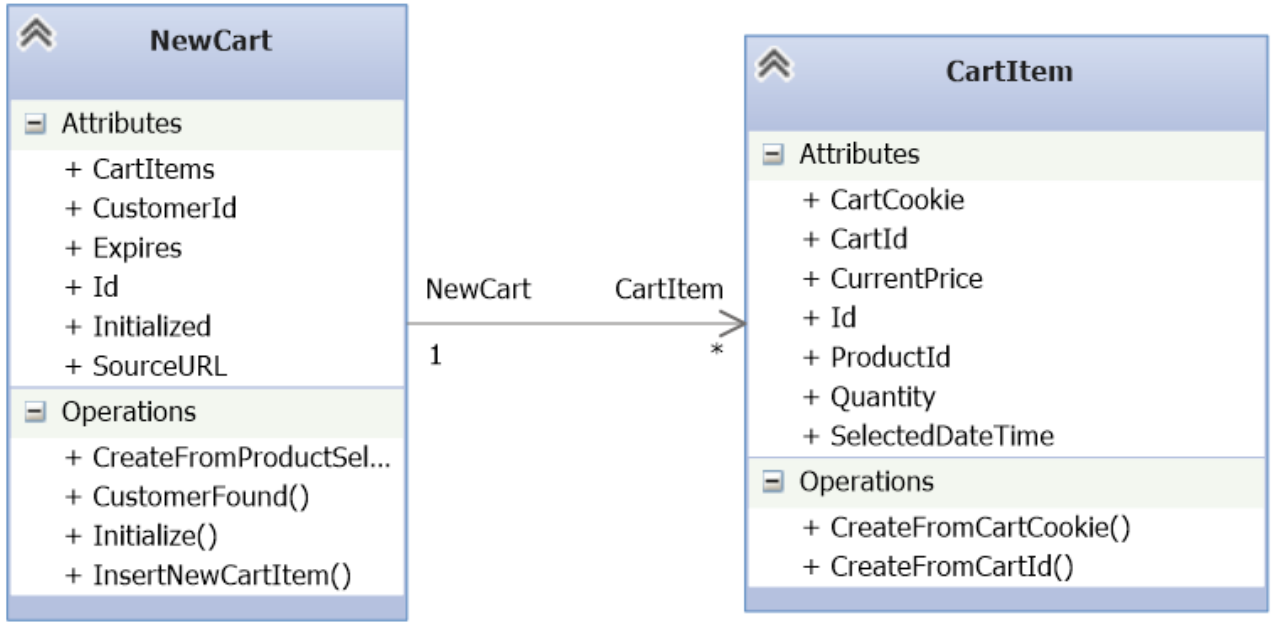
Eric Evans

Default to one-way navigations

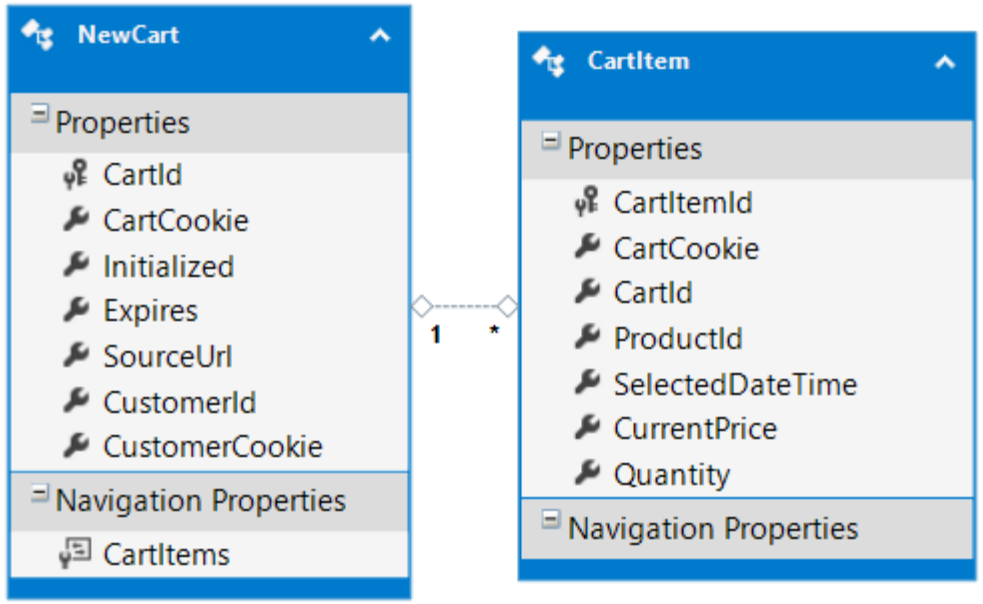
Only add in bi-directional,
if you can’t model your domain without it

EF Can Handle One Way Relationships

UML



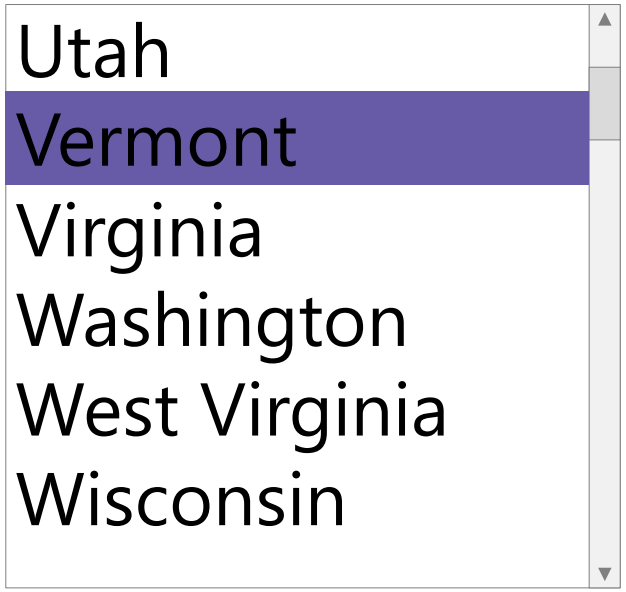
EF



Foreign Keys Are Your Friend

“But that’s database infrastructure”

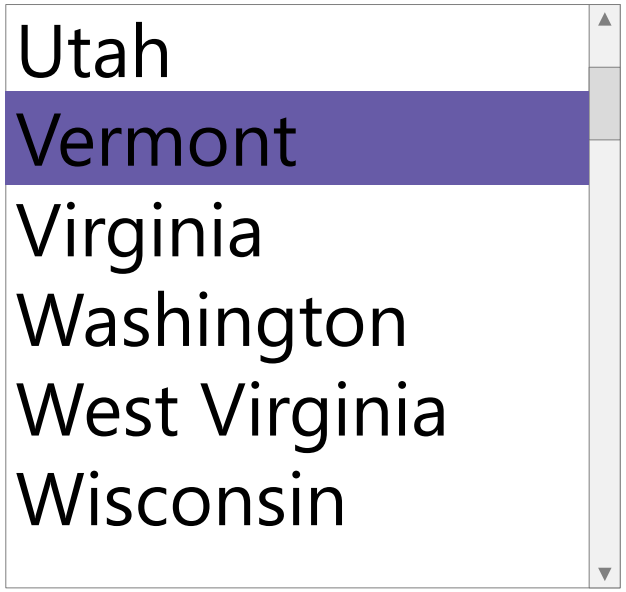
Setting Navigation Property and EF




```
listBox.DataSource=myContext.Regions.ToList();

myAddress.Region=(Region)myDropBox.SelectedItem;
myContext.Addresses.Add(myAddress);
myContext.SaveChanges();
```

Setting Foreign Key Property and EF




```
listBox.DataSource=myContext.Regions.ToList();
listBox.DataSource=myContext.Regions.Select(r=>r.Id,R.Name).ToList();
listBox.DataSource=myContext.RegionReferences.ToList();

myAddress.Region=(Region)myDropBox.SelectedItem;
myAddress.RegionId= myDropBox.SelectedValue;
myContext.Addresses.Add(myAddress);
myContext.SaveChanges();
```

EF's One-to-One Mapping Requires Bi-Directional Navigations

EF Needs All of This for 1:1 Relationship (also for 1:0..1)

Navigation Properties on both ends

Configuration Describing Full Relationship

Combination Primary/Foreign Key

DDD, EF and Many-to-Many Relationships

Just Say No!
(to *:*)

EF Needs Both Ends for Many to Many

Category.cs

Maintenance.Domain | Maintenance.Domain.Ca | CategoryId

```
public class Category
{
    public Category()
    {
        Products = new List<Product>();
    }

    public int CategoryId { get; set; }
    public string Name { get; set; }

    public ICollection<Product> Products { get; set; }
}
}
```

Product.cs

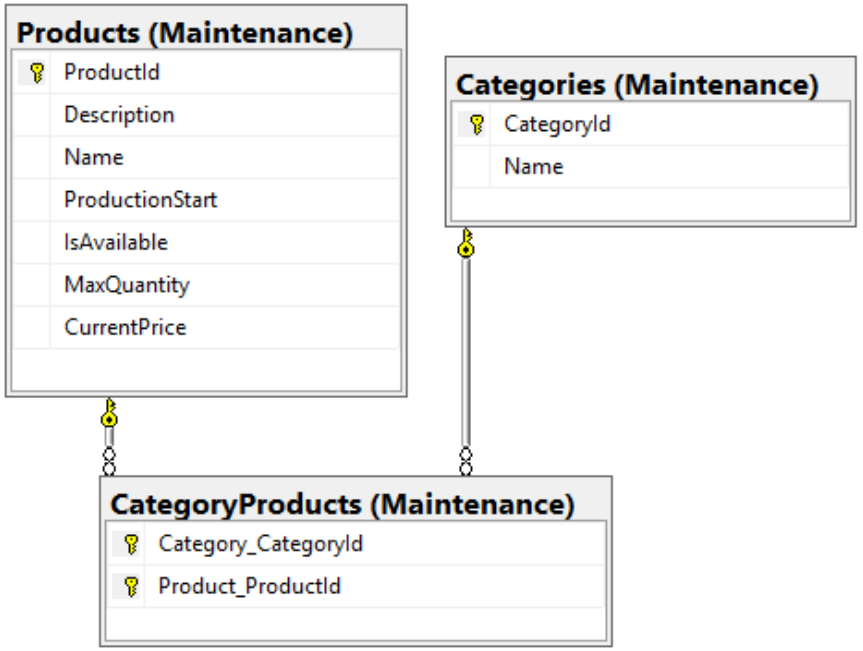
Maintenance.Domain | Maintenance.Domain.Prod | RemoveFromProductio

```
{
    public class Product
    {
        public Product()
        {
            IsAvailable = true;
            Categories=new List<Category>();
        }

        public int ProductId { get; set; }
        public string Description { get; set; }
        public string Name { get; set; }
        public DateTime ProductionStart { get; set; }
        public bool IsAvailable { get; set; }
        public ICollection<Category> Categories { get; set; }

        public int MaxQuantity { get; set; }
        public decimal CurrentPrice { get; set; }

        public void RemoveFromProduction()
        {
            IsAvailable = false;
        }
    }
}
```



DDD Prefers

Value Objects over 1:1 Relationships

String is a Value Object

```
string name= “Julie”

name.5thChar= “a”

name=“Julia”
```

Returns a new string!

```
String.Replace(“e”, “a”)
```

Returns a new string!

Value Object is Not an Entity

Value Object Attributes

- Identity is Combination of all values
- Immutable
- Ability to compare using all values
- No side effects

Common Value Objects

- Measurements 3&Feet, 2&Meters
- Financial Value 100&USD, 100&€

EF Groks Value Objects

Complex Type

Non-scalar properties of entity types that enable scalar properties to be organized within entities

Value Object

Not an Entity

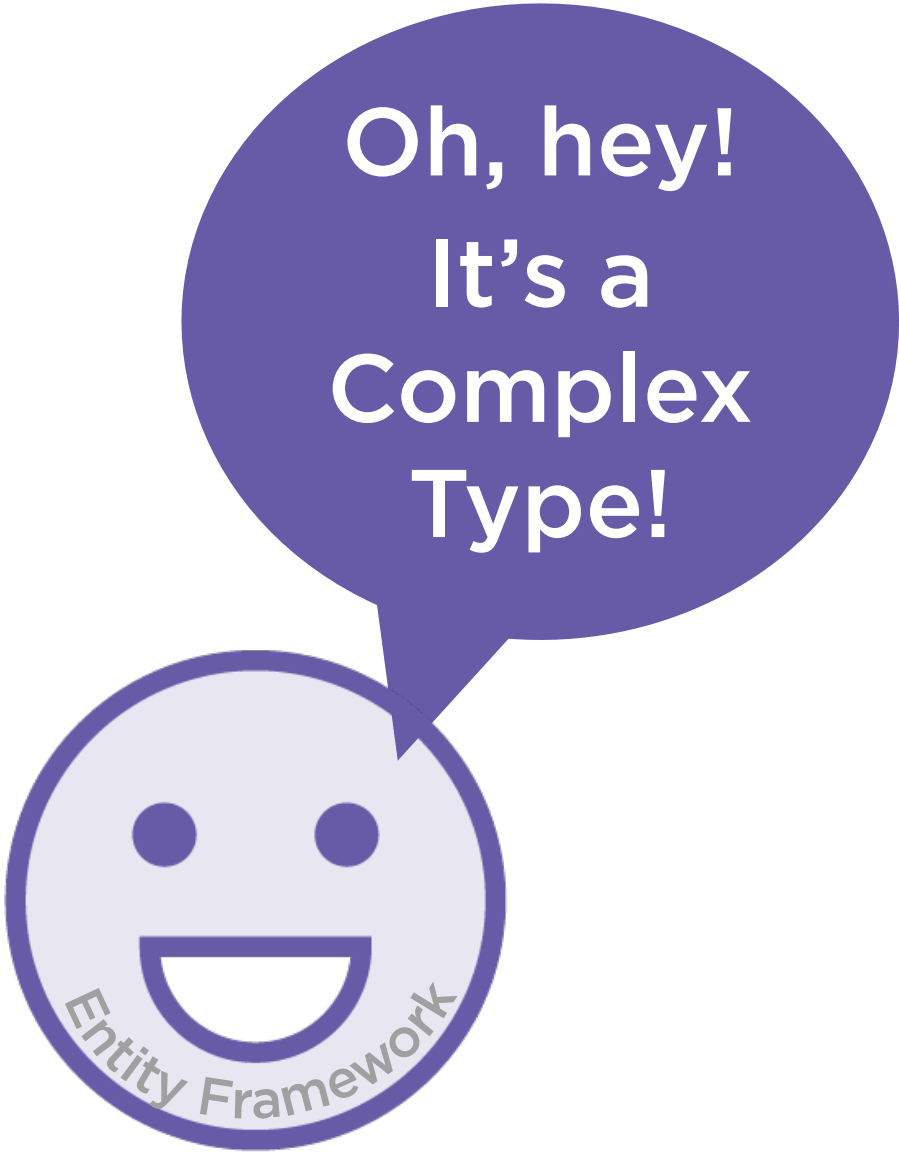
Identity is combination of all values

Immutable

Ability to compare using all values

No side effects

Used as a property of an entity



Seeing How EF Respond to Value Objects

Additional Advice about DDD Patterns and EF

Some Downsides with EF & DDD


SE HABLA CODE
LOS TECHIES

Domain modeling with Entity Framework scorecard

Posted by Jimmy Bogard on April 29, 2014

Constructors requiring invariants

Verdict: Pass

Private setters

Verdict: Pass

Private fields, no setters

Verdict: Fail

Encapsulated collections

Verdict: Fail

Encapsulated primitives (aka the NodaTime/Enumeration class problem)

Verdict: Fail, Pass with workaround

Value objects

Verdict: Don't use them anyway

Nullability concerns with value objects

Inability to map to fields

Cannot hide properties completely

Cannot encapsulate collections

Relationship Magic: Good & Not So Good

Beneficial EF relationship magic	Detrimental EF relationship magic
Eager/Lazy/Explicit loading	Confusing Side Effects with Writes to DB
Helpful for Writing LINQ Queries	Confusing Side Effects with Writes to DB
Insert/Update Foreign Key Fix-Up	Confusing Side Effects with Writes to DB

Favor Explicit Updates over EF Update Magic

Borrow from CQRS* to Balance Navigations

Read model

Include helpful navigation properties

Write model

Eliminate navigation properties & their side-effects

Control relationships via FK properties

*Command Query Responsibility Segregation

Review

Pros & Cons of Mapping Domain with EF

Revisit Rich Domain Models in Solution

How will EF Mappings react to patterns?

Private Setters and Constructors

Private Collections & Hidden Properties

Favoring One-Way Navigations

Value Objects over 1:1 Relationships

Avoiding some of EF Relationship Magic

Consider CQRS Pattern for Some Models

Resources

Helpful Pluralsight Courses

- EF6 Ninja Edition: What's New in EF6 bit.ly/PS-EF6
- Domain-Driven Design Fundamentals bit.ly/PS-DDD

Coding for Domain-Driven Design: Tips for Data-Focused Devs 3 Part Series
Julie Lerman, MSDN Magazine, August, Sept & Oct 2013 http://bit.ly/1GoZFZ8
Domain-Driven Design: Tackling Complexity in the Heart of Software
Eric Evans, 2003, Addison-Wesley Press
Patterns, Principals, and Practices of Domain-Driven Design
Scott Millett with Nick Tune, 2016, Wrox
CQRS Journey from MS Patterns & Practices: http://bit.ly/cqrsjourney
Blog Posts:
ardalis.com/exposing-private-collection-properties-to-entity-framework
udidahan.com/2009/01/24/ddd-many-to-many-object-relational-mapping
lostechies.com/jimmybogard/2014/03/12/avoid-many-to-many-mappings-in-orms
lostechies.com/jimmybogard/2014/04/29/domain-modeling-with-entity-framework-scorecard/

Mapping DDD Domain Models with EF

Julie Lerman

MOST TRUSTED AUTHORITY ON ENTITY FRAMEWORK

@julielerman thedatafarm.com