Tuning Default Mappings for the Data Model

Julie Lerman

EF Core Expert and Serial DDD Advocate

@julielerman | thedatafarm.com

The data model can be built

But is it being built correctly?

Overview

Explore the data model created from default mappings

Glean more observations from migration files and database

Note problems and apply configurations to fix them

Exploring the Default Data Model Visually

Owned Entities Masquerade as Entities in a Data Model

This is part of EF Core's mechanism for handling their change tracking and persistence. It all happens under the covers.

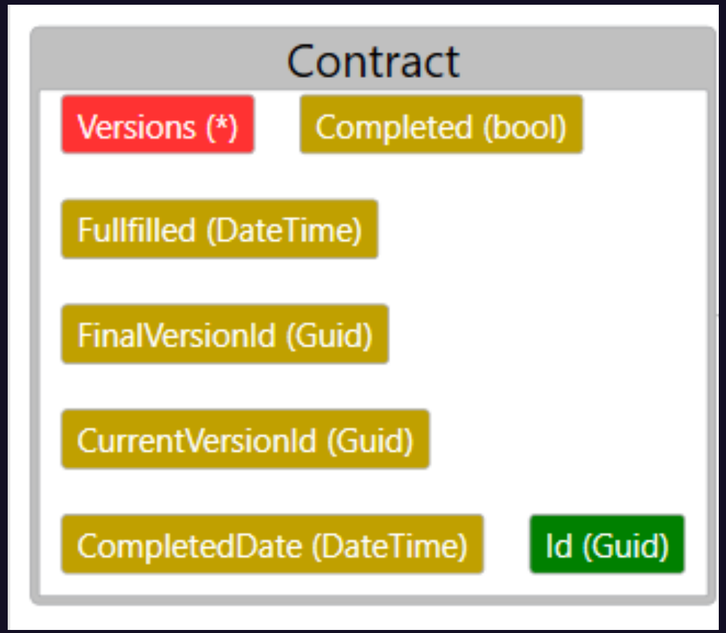
Atypical Properties Recognized by Convention

Getter & Setter
At least one is public

IEnumerable Collections
(public)

Enums
with or without value assignments

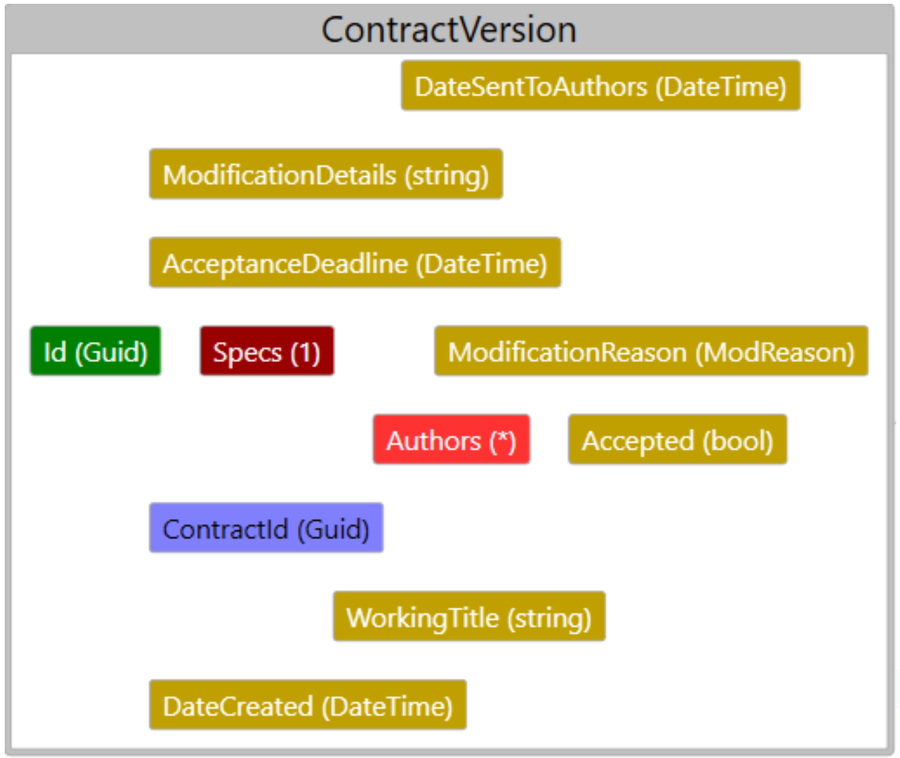
```
public IEnumerable<ContractVersion> Versions
  => _versions.ToList();

private readonly List<ContractVersion> _versions
  = new List<ContractVersion>();

private void AddVersion(ContractVersion version)
{
  _versions.Add(version);
  CurrentVersionId = version.Id;
}
```



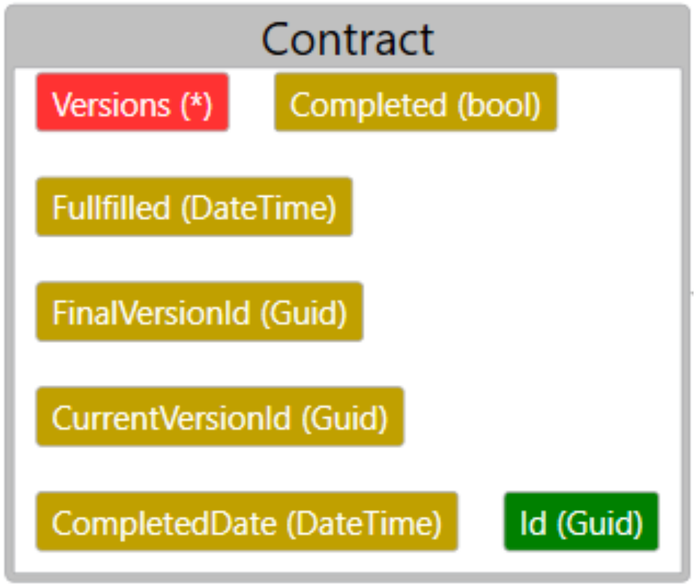
Versions Property is Fully Encapsulated using an IEnumerable

Pre-Assigning Enum Values Future-Protects Them

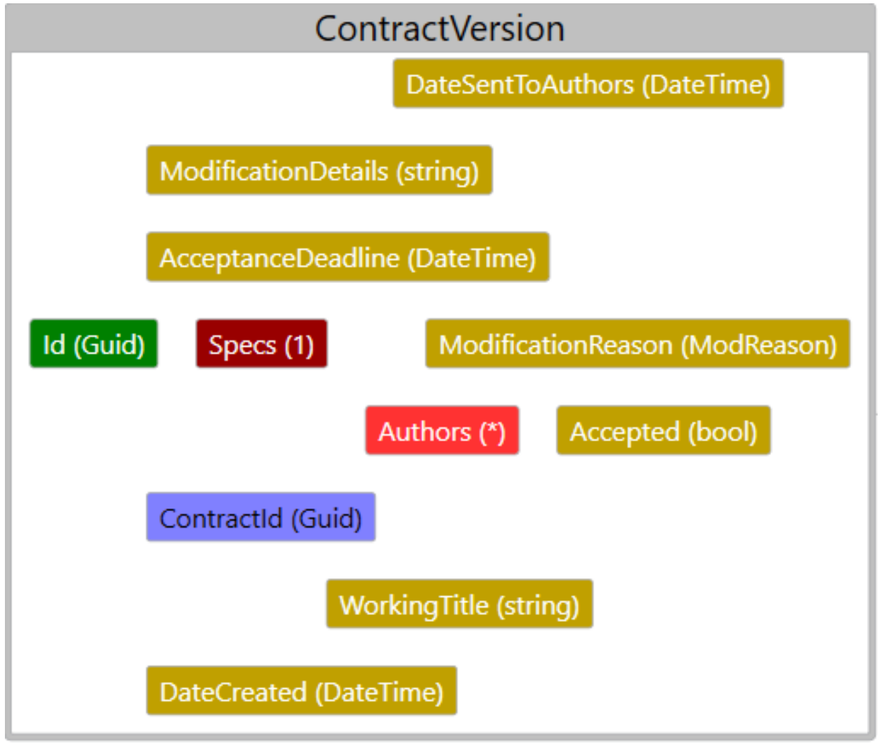
```
public enum ModReason
{
    NewContract = 1,
    FixTypos = 2,
    ChangeAttributes = 3,
    ChangeSpecs = 4,
    Other = 10
}
```



Identifying Missing Data and Understanding Why EF Core Missed It




```
public class Contract : BaseEntity<Guid>
{
    private Contract(){ }
    public Contract(DateTime initDate, List<Author> authors, str

    private void AddVersion(ContractVersion version)...

    public string ContractNumber => _contractNumber;
    public DateTime DateInitiated => _initiated;
✓   public Guid CurrentVersionId { get; private set; }
✓   public Guid FinalVersionId { get; private set; }
✓   public bool Completed { get; private set; }
✓   public DateTime CompletedDate { get; private set; }
✓   public DateTime Fullfilled { get; private set; }
✓   public IEnumerable<ContractVersion> Versions => _versions.To

    private string _contractNumber;
    private DateTime _initiated;
    private readonly List<ContractVersion> _versions = new List<
```




```
private ContractVersion(SpecificationSet specs,BaseAttribute
                        DateTime? deadline,bool revisedSpecs
private bool _hasRevisedSpecSet;
public SpecificationSet Specs { get; private set; }
public Guid ContractId { get; private set; }
public string WorkingTitle { get; private set; }
public DateTime DateCreated { get; private set; }
public DateTime DateSentToAuthors  { get; private set; }
public DateTime AcceptanceDeadline { get; private set; }
public string ModificationDetails { get; private set; }
public ModReason ModificationReason { get; private set; }
public bool Accepted { get; private set; }

private readonly List<Author> _authors = new List<Author>()
public IEnumerable<Author> Authors => _authors.ToList();
public void SentToAuthors(DateTime datesent)
{
    DateSentToAuthors= datesent;
}
```

Missing Properties of Contract & Contract Version

```
public string ContractNumber => _contractNumber;
public DateTime DateInitiated => _initiated;
public Guid CurrentVersionId { get; private set; }
public Guid FinalVersionId { get; private set; }
public bool Completed { get; private set; }
public DateTime CompletedDate { get; private set; }
public DateTime Fullfilled { get; private set; }
public IEnumerable<ContractVersion> Versions => _versions.ToList();

private string _contractNumber;
private DateTime _initiated;
private readonly List<ContractVersion> _versions = new List<Contrac
```

Contract

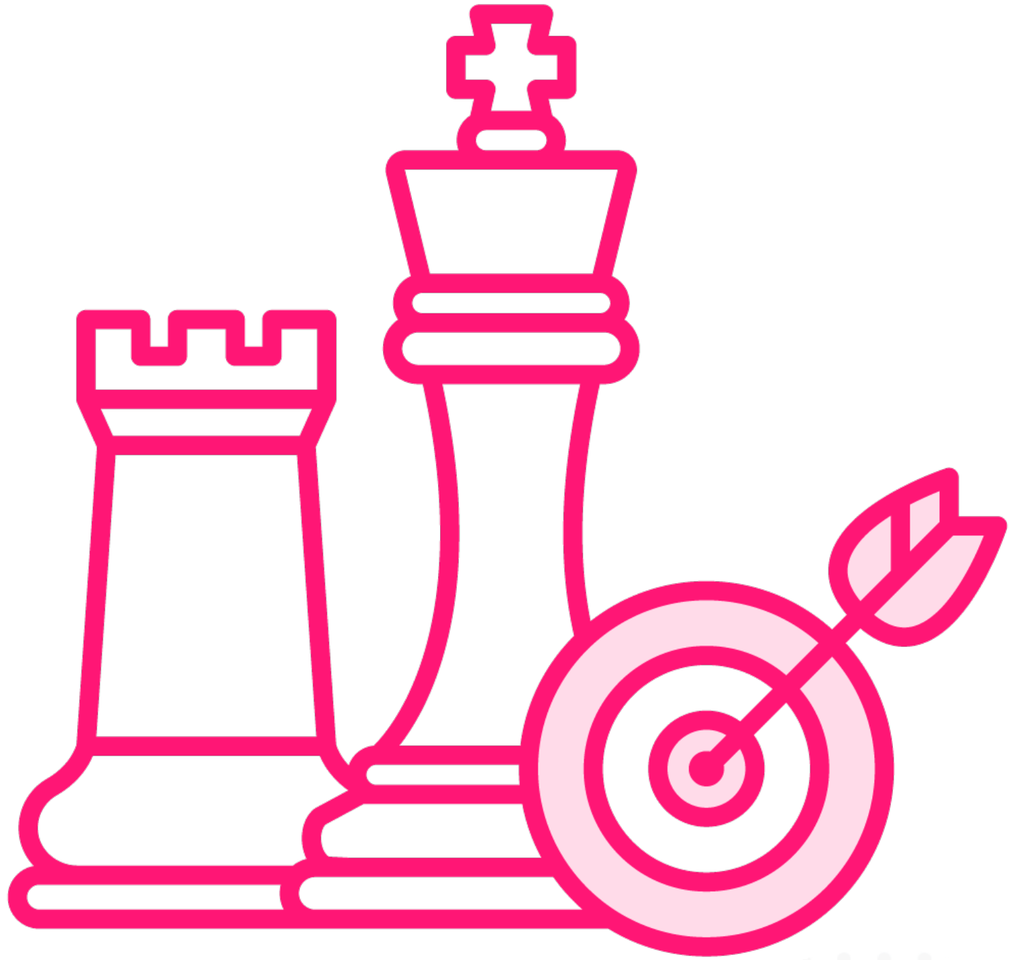
```
private bool _hasRevisedSpecSet;
public SpecificationSet Specs { get; private set; }
public Guid ContractId { get; private set; }
public string WorkingTitle { get; private set; }
public DateTime DateCreated { get; private set; }
public DateTime DateSentToAuthors  { get; private set; }
public DateTime AcceptanceDeadline { get; private set; }
public string ModificationDetails { get; private set; }
public ModReason ModificationReason { get; private set; }
public bool Accepted { get; private set; }

private readonly List<Author> _authors = new List<Author>();
public IEnumerable<Author> Authors => _authors.ToList();
public void SentToAuthors(DateTime datesent)
{
    DateSentToAuthors= datesent;
}
```

ContractVersion

```
Property:
public DateTime DateInitiated {get; private set;}


Discovered fields:
private DateTime _DateInitiated;   _[property name]
private DateTime m_DateInitiated;  m_[property name]
private DateTime dateInitiated;    [camel case]
private DateTime _dateInitiated;   _[camel case]
private DateTime m_dateInitiated;  m_[camel case]
```

Backing Field that EF Core Sees by Default

Related property must be visible e.g., public, has setter & getter
Field names must follow a pattern that EF Core looks for
Fields can be public or private



One of these things is not like the other

Mapping the Missing Fields

Properties with Backing Fields and a Local Field

Contract.ContractNumber and _contractNumber

Contract.DateInitiated and _initiated

_hasRevisedSpecSet

_hasRevisedSpecSet

ContractVersion local field
needs to be mapped!

All of the fixes were done using DbContext configurations and leaving the aggregate in tact.

Ensuring the Database is Created Suitably

Mapping the Decimal Field

```
modelBuilder.Entity<ContractVersion>()
  .OwnsOne(v => v.Specs)
   .Property(s=>
     s.PriceForAddlAuthorCopiesUSD)
     .HasColumnType(("decimal(7,2)"));
```

Mapping only the SpecificationSet Owned Entity of ContractVersion

```
configurationBuilder
  .Properties<decimal>()
  .HaveColumnType("decimal(7, 2)");
```

As the new default for all decimals (which can be overridden with property specific mappings)

Renaming a Table Tied to an Owned Entity Collection

Migrations is Recreating a Table?

You do not want this to happen during production!

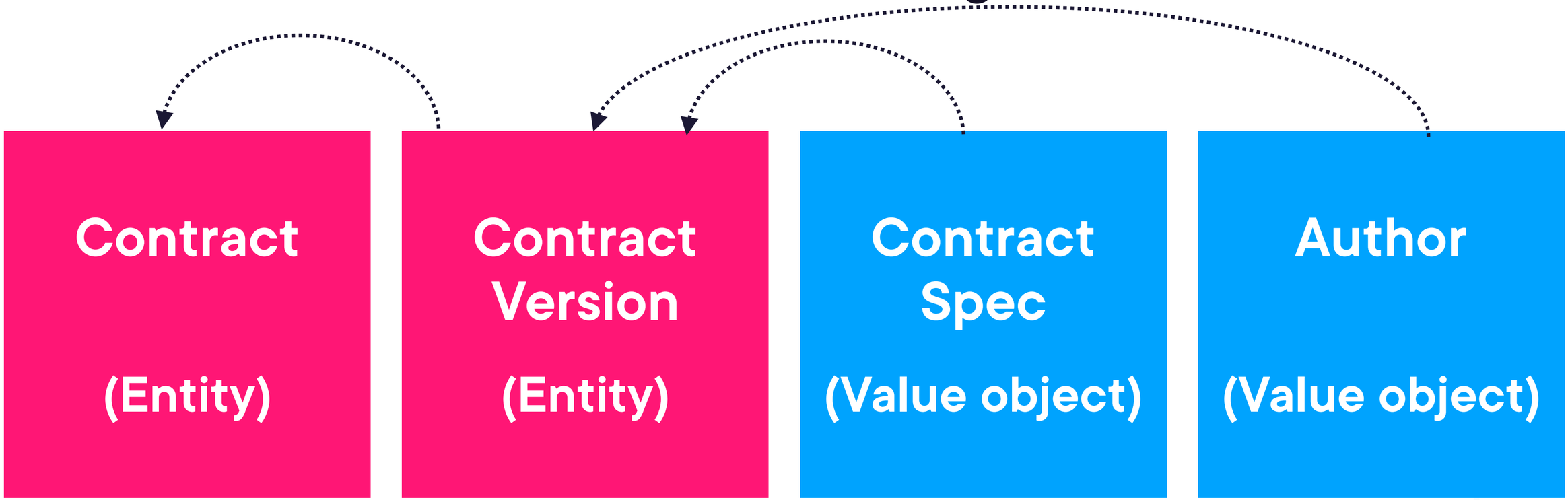
Handling Destructive Migrations Before Production

- Delete the Migrations folder
- Create a brand new starting point migration
- Delete the existing (dev) database

Use SQL to Affect Renames Directly

```
protected override void Up(MigrationBuilder
 migrationBuilder)
{
    migrationBuilder.Sql
      (“SQL to rename table and constraints”);
}

protected override void Down(MigrationBuilder
 migrationBuilder)
{
     migrationBuilder.Sql
       (“SQL to undo table and constraint renames” );
}
```

The Critical Classes of Contracting



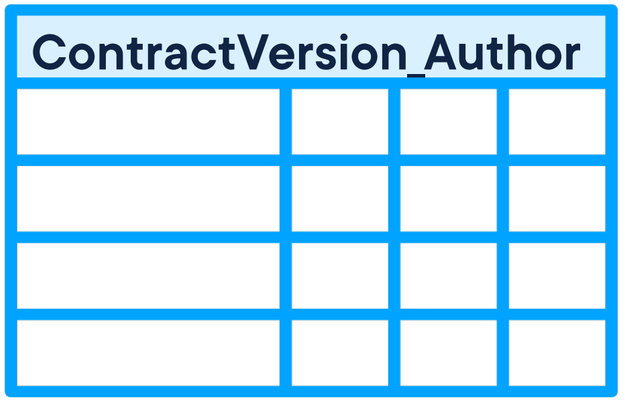
Tracks the lifetime of a contract and high level details

Tracks each iteration of the contract

Defines the collection of particulars of a version

Authors name with an ID if previously signed

What if Multiple Entities Had “OwnsMany<Author>”?



Summary

EF Core mapping conventions are pretty smart!

Leverage tools to validate the mappings

Fluent API lets you map most anomalies

EF Core cannot read your mind!

Don’t forget to check the database too!

Up Next:

Using Integration Tests to Validate Persistence