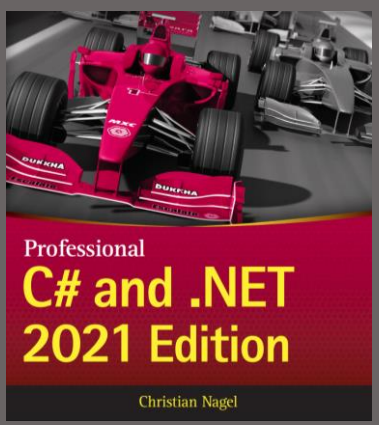
DATA PROGRAMMING – EF CORE

Christian Nagel
www.cninnovation.com



CHRISTIAN NAGEL

- Training
- Coaching
- Coding
- Writing

- csharp.christiannagel.com
- www.cninnovation.com
- @christiannagel
- Microsoft MVP (.NET)

INTRODUCTION

- Experience
- Expectations

COURSE MATERIALS

- Professional C# 7 and .NET Core 2.0
- Slides
- Course Samples
- https://www.github.com/cnilearn

TOPICS

- Overview
- Getting Started
- Creating a Model
- Querying Data
- Relationships
- Saving Data
- Handle Concurrency
- Migrations
- N-Tier Applications


OVERVIEW

HISTORY (.NET FRAMEWORK)

EF	.NET	Features	Issues
1.0	3.5 Update 1	CSDL, MSL, SSDL	.vs. LINQ to SQL
4.0	4.0	Easier, POCO, DDL…	Release cycles
4.1		Code First	
4.2		Bug Fixes	
4.3		Migrations	
5.0	4.5	Performance, SQL Server enhancements	or 4.4?
6.0		All NuGet	ASP.NET Web Forms compat

HISTORY (.NET CORE)

EF Core	Features
1.0	NoSQL, Platforms, DI, tooling
1.1	Mapping to fields, explicit loading, resiliency
2.0	Table splitting, owned types, query filters, DbContext pooling, compiled queries, track graphs, group join, EF functions…
2.1	Lazy loading, parameters in entity constructors, value conversions (e.g. enums), data seeding, LINQ GroupBy Translation, Query types, Include for derived types, System.Transaction support, optimization of correlated subqueries, Owned attribute, state change events, raw SQL parameter analyzer
2.2	Spatial data, collections of owned entities, query tags
3.X	Cosmos DB Provider, IAsyncEnumerable, nullable reference types…
5.0	Many-to-many, map entity types to queries, event counters, property bags, scaffold singularizes, split queries…

https://docs.microsoft.com/de-at/ef/core/what-is-new/ef-core-2.1
https://github.com/aspnet/EntityFrameworkCore/issues?q=is%3Aissue+milestone%3A3.0.0

EF MAPPING VARIANTS

Database First	Mapping via XML (CSDL, MSL, SSDL)
Model First	
Code First	Mapping via Code

- Mapping via XML is discontinued with EF Core 1.0
- Code First – a database can be created first as well
  - Code First was originally named Code Only, but changed to Code First for similar names

ENTITY FRAMEWORK CORE

- Lightweight
- Extensible
- Cross-Platform

- Top-Level APIs similar to EF 6
- Big differences behind the scenes

CHALLENGES WITH ENTITY FRAMEWORK (PRE CORE)

- Long history going back >15 years
  - Older APIs and design patterns
  - Uses APIs not available on all platforms
  - Seldom used code/features
  - Monolithic implementation
  - Unintuitive behaviors throughout code
- Not optimized for density/devices
  - High memory footprint
- Tight coupling to relational concepts

OVERVIEW EF CORE

- Lightweight, extensible version of EF
- New Platforms
  - Universal Windows Platform, ASP.NET Core
- New Data Stores
  - Non-relational data stores – Azure Cosmos DB, Redis

NEW CORE

- Same top level experience as EF6
  - DbContext/DbSet…
- Just the commonly used features
  - Plus many new features
- Code-based modelling only
  - Still supports creating model from existing database

- New Core
  - Core = metadata, change tracking, query pipeline…
  - Easier to replace/extend components
  - Replace confusing APIs & behavior
  - Optimized for memory and CPU usage
  - Pay-per-play components

NEW FEATURES

- Batching during SaveChanges
- Unique constraints
- Client eval in LINQ queries
- SQL server sequences
- Shadow state

SUMMARY

- History
- EF Core
- EF 6 is still maintained, can be used with .NET Core

GETTING STARTED

MODEL

- Models can be defined as simple POCO types

```
public class Book
{
   public int BookId { get; set; }
   public string Title { get; set; } = string.Empty;
   public string? Publisher { get; set; }
}
```

CONTEXT

- Connection to the database

```
public class BooksContext : DbContext
{
    public DbSet<Book> Books { get; set; } = default!;

    protected override void OnConfiguring(DbContextOptionsBuilder optionsBuilder)
    {
        optionsBuilder.UseSqlServer(ConnectionString);
    }
}
```

CONTEXT – ALTERNATIVE FOR DI

- NuGet Package Microsoft.Extensions.DependencyInjection

```
public class BooksContext : DbContext
{
    public BooksContext(DbContextOptions<BooksContext> options)
        : base (options) { }

    public DbSet<Book> Books { get; set; } = default!;
}
```

```
var services = new ServiceCollection();
services.AddDbContext<BooksContext>(options =>
{
    options.UseSqlServer(ConnectionString);
});
Container = services.BuildServiceProvider();
```

CREATE THE DATABASE

- Database can be created programmatically

```
using var context = new BooksContext();
bool created = context.Database.EnsureCreated();
```

DATA SEEDING (2.1)

- Populate databases with initial set of data
- Migrations can be applied

```
modelBuilder.Entity<Book>().HasData(
  new Book
  {
    BookId = 1,
    Title = "Professional C# 7",
    Publisher = "Wrox Press"
  });
```

READ AND WRITE

- Create Records

```
using var context = new BooksContext();
context.Books.Add(new Book
  { Title = "Professional C# 7", Publisher = "Wrox Press" });
int changed = context.SaveChanges();
```

- Read Records

```
using var context = new BooksContext();
var books = context.Books;

foreach (var book in books)
{
    Console.WriteLine($"{book.BookId} {book.Title} {book.Publisher}");
}
```

LOGGING

- Simple Logging
  - optionsBuilder.LogTo (5.0)
- EnableDetailedErrors, EnableSensitive (5.0)
- Create ILoggerFactory (LoggerFactory.Create)
  - Configure EF-context with UseLoggerFactory

SUMMARY

- Microsoft.EntityFrameworkCore
- Microsoft.EntityFrameworkCore.SqlServer

- Model
- Context

CREATING A MODEL

MODEL DEFINITION

- Conventions
- Data Annotations
- Fluent API

INCLUDING/EXCLUDING TYPES

- Conventions
  - Types are Exposed via DbSet Properties
  - OnModelCreating can add more types
  - Navigation Properties
- Data Annotations
  - [NotMapped]
- Fluent API
  - modelBuilder.Ignore

INCLUDING/EXCLUDING PROPERTIES

- Conventions
  - Properties with getter and setter
- Data Annotations
  - [NotMapped]
- Fluent API
  - modelBuilder.Entity<Book>().Ignore(b => b.LoadTime)

TABLE MAPPING (RELATIONAL ONLY)

- Conventions
  - Name of the DbSet Property
- Data Annotations
  - [Table]
- Fluent API
  - modelBuilder.Entity<Book>().ToTable

COLUMN MAPPING (RELATIONAL ONLY)

- Conventions
  - Property Name
- Data Annotations
  - [Column]
- Fluent API
  - modelBuilder.Entity<Book>().Property(b => b.Id).HasColumnName("book_id");

DATA TYPES (SQL SERVER)

- Conventions
  - datetime2(7) for DateTime, nvarchar(max) for string
- Data Annotations
  - [Column(TypeName="varchar(200)")]
- Fluent API
  - modelBuilder.Entity<Book>().Property(b => b.Title).HasColumnType("nvarchar(200)");
  - Alternative SQL Server: ForSqlServerHasColumnType

PRIMARY KEYS

- Conventions
  - Property named Id or <TypeName>id
- Data Annotations
  - [Key]
- Fluent API
  - modelBuilder.Entity<Book>().HasKey(b => b.BookId)
  - Composite Keys configured via Fluent API

GENERATED PROPERTIES

- No value generated
- Value generated on add
- Value generated on add or update

GENERATED PROPS - CONVENTIONS

- Primary keys of integer or GUID, values generated on add
- All others no value generation

GENERATED PROPS – DATA ANNOTATIONS

- No value
  - [DatabaseGenerated(DatabaseGeneratedOption.None)]
- On Add
  - [DatabaseGenerated(DatabaseGeneratedOption.Identity)]
- On Add or Update
  - [DatabaseGenerated(DatabaseGeneratedOption.Computed)]
  - Needs additional information
  - Dependent on Provider

GENERATED PROPS – FLUENT API

- No value
  - modelBuilder.Entity<Book>().Property(b => b.BookId).**ValueGeneratedNever()**)
- On Add
  - modelBuilder.Entity<Book>().Property(b => b.Inserted).**ValueGeneratedOnAdd()**)
- On Add or Update
  - modelBuilder.Entity<Book>().Property(b => b.LastUpdated)
    .**ValueGeneratedOnAddOrUpdate()**)

REQUIRED / OPTIONAL

- Conventions (C# 8)
  - Nullable Types are optional
  - Not-nullable Types are required
- Data Annotations
  - [Required]
- Fluent API
  - modelBuilder.Entity<Book>().Property(b => b.Title).**IsRequired()**;

MAXIMUM LENGTH

- Conventions
  - Defined by Provider
  - SQL Server: data – nvarchar(max), keys: nvarchar(450)
- Data Annotions
  - [MaxLength(200)]
- Fluent API
  - modelBuilder.Entity<Book>().Property(b => b.Title).**HasMaxLength(50);**

SHADOW PROPERTIES

- Do not exist in entity
- Maintained in the Change Tracker

- Conventions
  - Relationship detected, but no foreign key property found
- Fluent API
  - modelBuilder.Entity<Book>().Property<DateTime>("LastUpdated");

MAPPING TO FIELDS (1.1)

- Mapping to read-only properties
- Mapping to fields
- Annotations
  - BackingField attribute (5.0)
- Fluent API
  - HasField()

```
modelBuilder.Entity<Book>().Property(b => b.BookId).HasField("_bookId");
modelBuilder.Entity<Book>().Property(b => b.Publisher).HasField("_publisher");

modelBuilder.Entity<Book>().Property<string>("JustABackingField")
  .HasField("_internalState");
```

INDEXES

- Conventions
  - Created in each property that is used as a foreign key
- Annotations
  - IndexAttribute on table (5.0)
- Fluent API
  - modelBuilder.Entity<Book>().HasIndex(b => b.Url);
  - Unique Index (IsUnique)
  - Index over multiple columns (HasIndex)

ALTERNATE KEYS

- Alternate unique identifier
- Targets of a foreign key

- Conventions
  - Property as a target of a relationship (HasPrincipalKey)
- Fluent API
  - HasAlternateKey

SEQUENCES (RELATIONAL ONLY)

- Fluent API
  - HasSequence
  - can use
    - StartsAt
    - IncrementsBy
    - HasDefaultValueSql

INHERITANCE (TPH) - CONVENTIONS

- Table per Hierarchy
- For types included in the model
- Discriminator Column
- Value: Typename

```
public class Payment
{
  public int PaymentId { get; set; }
  public double Amount { get; set; }
}

public class CreditcardPayment : Payment
{
  public string CreditCardNumber { get; set; }
}

public class BankContext : DbContext
{
  public DbSet<Payment> Payments { get; set; }
  public DbSet<CreditcardPayment> CreditCardPayments { get; set; }
}
```

INHERITANCE (TPH) – FLUENT API

- HasDiscriminator
- HasValue
- Only base type needs to be listed with the context properties

```
modelBuilder.Entity<Payment>()
    .HasDiscriminator("payment_type")
    .HasValue<Payment>("payment")
    .HasValue<CreditcardPayment>("creditcard");
```

INHERITANCE (TPT) – 5.0

- Table per Type
- Annotations
  - Use Table Attribute
- Fluent API
  - ToTable()

SUMMARY

- Conventions
- Attributes
- Fluent API

QUERYING DATA

BASIC QUERY – LOADING ALL DATA

```
using var context = new BooksContext();
var books = context.Books.ToList();
```

BASIC QUERY – LOADING A SINGLE ENTITY

```
using var context = new BooksContext();
var book = context.Books.Single(b => b.BookId == 1);
```

BASIC QUERY – FILTERING

```
using var context = new BooksContext();
var books = context.Books
    .Where(b => b.Title.Contains("Pro"))
    .ToList();
```

TRACKING

- Queries are tracked by default
- Do not track queries

```
using var context = new BooksContext();
var books = context.Books
    .AsNoTracking()
    .ToList();
```

- Change default tracking behavior

```
using var context = new BooksContext();
context.ChangeTracker.QueryTrackingBehavior = QueryTrackingBehavior.NoTracking;
```

RAW SQL QUERIES

- FromSQL (obsolete) -> FromSqlInterpolated & From SqlRaw

```
var books = context.Books
    .FromSqlInterpolated($"SELECT * FROM Books WHERE TITLE = '{title}'")
    .ToList();
```

- Execute Stored Procedures

```
var books = context.Books
    .FromSqlRaw("EXECUTE dbo.GetAllBooks")
    .ToList();
```

COMPILED QUERIES

- Performance for often used queries

```
var query = EF.CompileQuery((BooksContext context, string isbn) =>
  context.Books.Single(b => b.Isbn == isbn);
```

GLOBAL QUERY FILTERS

- All queries are extended
  - e.g. IsDeleted, TenantId

```
protected override void OnModelCreating(ModelBuilder modelBuilder)
{
  modelBuilder.Entity<Menu>().HasQueryFilter(m => !m.IsDeleted);

  modelBuilder.Entity<Menu>().HasQueryFilter(m =>
    EF.Property<string>(m, "TenantId") == _tenandId);
}
```

SUMMARY

- LINQ to EF
- Tracking
- Raw SQL Queries

RELATIONSHIPS

RELATIONSHIPS - TERMS

- Dependent Entity
  - Contains foreign key property
- Principal Entity
  - Contains primary/alternate key property
- Foreign Key
  - Store value of the principal key property of the relation
- Principal Key
  - Uniquely identify principal entity – primary or alternate key
- Navigation Property
  - Collection Navigation Property
    - References to many related entities
  - Reference Navigation Property
    - Reference to a single related entity
  - Inverse Navigation Property
    - Navigation property on the other end

RELATIONSHIPS - CONVENTIONS

- Fully Defined Relationship

```
public class Book
{
    public int BookId { get; set; }
    public string Title { get; set; }
    public List<Chapter> Chapters { get; set; }
}

public class Chapter
{
    public int ChapterId { get; set; }
    public int BookId { get; set; }
    public Book Book { get; set; }
}
```

RELATIONSHIP – CONVENTIONS (2)

- No foreign key property
  - A shadow property will be created
- Single Navigation Property
  - Inverse navigation not needed
- Cascade Delete
  - Cascade for required relationships
  - SetNull for optional relationships

RELATIONSHIPS – DATA ANNOTATIONS

- ForeignKey

```
public class Book
{
    public int BookId { get; set; }
    public string Title { get; set; }
    public List<Chapter> Chapters { get; set; }
}

public class Chapter
{
    public int ChapterId { get; set; }
    public string Title { get; set; }

    public int BookId { get; set; }
    [ForeignKey("BookId")]
    public Book Book { get; set; }
}
```

RELATIONSHIPS – DATA ANNOTATIONS (2)

- InverseProperty

```
public class Book
{
    public int BookId { get; set; }
    public string Title { get; set; }
    public int? AuthorUserId { get; set; }
    public User Author { get; set; }
    public int? ReviewerUserId { get; set; }
    public User Reviewer { get; set; }
}

public class User
{
    public int UserId { get; set; }
    [InverseProperty("Author")]
    public List<Book> Books { get; set; }
    [InverseProperty("Reviewer")]
    public List<Book> ReviewedBooks { get; set; }
}
```

LOADING RELATED DATA

- Eager Loading
- Explicit Loading
- Lazy Loading

EAGER LOADING

- *Include* method to specify related data

```
using var context = new BooksContext();
var books = context.Books
    .Include(b => b.Authors)
    .ToList();
```

EAGER LOADING MULTIPLE LEVELS

- *ThenInclude* method for multiple levels

```
using var context = new BooksContext();
var books = context.Books
    .Include(b => b.Authors)
    .ThenInclude(a => a.Address)
    .ToList();
```

EXPLICIT LOADING (1.1)

- Explicit loading via DbContext.Entry

```
var blog = context.Blogs
  .Single(b => b.BlogId == 1);

context.Entry(blog)
  .Collection(b => b.Posts)
  .Load();

context.Entry(blog)
  .Reference(b => b.Owner)
  .Load();
```

LAZY LOADING (2.1)

- Lazy loading available since 2.1(uses explicit loading)

OWNED ENTITIES (2.0)

- Owned Entities (OwnsOne)
- Table Splitting (ToTable)

```
modelBuilder.Entity<Customer>()
  .OwnsOne(c => c.WorkAddress)
    .OwnsOne(a => a.Location);

modelBuilder.Entity<Customer>()
  .OwnsOne(c => c.PhysicalAddress)
  .ToTable("Customers_Location")
  .OwnsOne(a => a.Location);
```

COLLECTIONS OF OWNED ENTITIES (2.2)

- OwnsMany
- Relevant with NoSQL databases

EF.FUNCTIONS (2.0)

- Provider specific functionality
- Implemented with .NET Core 2.0 & SQL Server: Like

```
public IEnumerable<Blog> SearchBlogs(string term)
{
  var likeExpression = $"%{term}%";

  return _db.Blogs
    .Where(b => EF.Functions.Like(b.Url, likeExpression));
}
```

SUMMARY

- Defining relationships in the Model
- Query with eager loading

SAVING DATA

BASIC SAVE - ADD

- ChangeTracker associated
- Add entity to DbSet
- Creates an SQL INSERT statement
- SaveChanges

```
using var context = new BooksContext();
var book = new Book { Title = "Professional C# 7.0" };
context.Books.Add(book);
context.SaveChanges();
```

BASIC SAVE – UPDATE

- update entity
- creates an SQL UPDATE statement

```
using var context = new BooksContext();
var book = context.Books.First();
book.Title = “Professional C# 7.0”;
context.SaveChanges();
```

BASIC SAVE – DELETE

- delete entity
- creates an SQL DELETE statement

```
using var context = new BooksContext();
var book = context.Books.First();
context.Books.Remove(book);
context.SaveChanges();
```

CHANGE TRACKING

- Snapshot change tracking
  - Snapshot taken when entity is associated with context
  - *ChangeTracker.DetectChanges* scans changes
  - Configure via *ChangeTracker.AutoDetectChangesEnabled*

RELATIONS

- Tracked objects are added/modified/deleted

CASCADE DELETE

- Cascade
  - Dependent entities are deleted
- SetNull
  - Foreign key property are set to null
- Restrict
  - Dependent entities remain unchanged.

TRANSACTIONS

- By default, SaveChanges is a transaction
- Create transaction with *context.Database.BeginTransaction*
- Use existing transaction with *context.Database.UseTransaction*
- Commit: *transaction.Commit*

BATCHING

- Send multiple Insert/Update/Delete in one request

```
optionsBuilder.UseSqlServer(
  ConnectionString, options => options.MaxBatchSize(10));
```

SUMMARY

- Objects are tracked
- INSERT/UPDATE/DELETE statements generated

CONCURRENCY CONFLICTS

CONCURRENCY CONFLICTS

- Three sets of values
  - Current values
  - Original values
  - Database values
- Uses concurrency tokens
- DbUpdateConcurrencyException
  - check DbUpdateConcurrencyException.Entries

MODEL - CONCURRENCY TOKENS

- Conventions
  - Never
- Data Annotations
  - [ConcurrencyCheck]
- Fluent API
  - modelBuilder.Entity<Book>().Property(b => b.Title).**IsConcurrentToken);**

TIMESTAMP AS CONCURRENCY TOKEN

- Byte[] required with SQL Server
- Data Annotations
  - [Timestamp]
- Fluent API
  - modelBuilder.Entity<Book>().Property(b => b.Timestamp
    .ValueGeneratedOnAddOrUpdate()
    .IsConcurrentToken);

SUMMARY

- All scenarios are possible
- Concurrency tokens to manage concurrency


MIGRATIONS

OVERVIEW

- Create the database from code
- Update the database schema

CUSTOMIZING SCHEMA

- Using Attributes
- Using Fluent API

.NET CORE CLI

- dotnet ef
- dotnet ef database drop/update
  - drop/update the database
- dotnet ef dbcontext list/scaffold
  - scaffold context type for a specified database
- dotnet ef migrations add/list/remove/script
  - add/list/remove migrations, generate SQL script

PROGRAMMATIC MIGRATION

- context.Database.Migrate()

SUMMARY

- Update Database Schema
- Migrations using Code
- Migrations using SQL

MORE FEATURES

CONTEXT POOLING (2.0)

- Reuse contexts

```
var services = new ServiceCollection()
  .AddDbContextPool<BooksContext>(
    c => c.UseSqlServer(ConnectionString, poolSize: 16));
```

CONTEXT FACTORY (5.0)

- Context factory for a different DbContext-Lifetime
- Dispose context manually
- Register context factory in the DI container (AddDbContextFactory)
- Inject IDbContextFactory
- Create context instances via CreateDbContext

MANY-TO-MANY RELATIONSHIPS

- HasMany/WithMany
- Mapping uses property bag by default
- UsingEntity for a custom mapping type

SPLIT QUERIES

- Split complex queries with *Include* into multiple queries
- AsSplitQuery with LINQ query
- Enabled globally: UseQuerySplittingBehavior

https://docs.microsoft.com/en-us/ef/core/querying/single-split-queries

EVENT COUNTERS

- Metrics information from EF Core
- Use dotnet counters for monitoring
- dotnet counters monitor Microsoft.EntityFrameworkCore -p <PID>

https://docs.microsoft.com/en-us/ef/core/logging-events-diagnostics/event-counters?tabs=windows

UNIT TESTING

- In-Memory EF Core Provider

```
var builder = new DbContextOptionsBuilder<BooksContext>()
  .UseInMemoryDatabase();
```

- Better: use an interface for the DbContext

EF CORE COSMOS PROVIDER

- NoSQL database with JSON documents
- Configure partition key: HasPartitionKey (convertible to string)
- OwnsMany / OwnsOne – embeds entities
  - change JSON property: ToJsonProperty
- Optimistic concurrency
  - UseETagConcurrency

N-TIER APPLICATIONS

ASP.NET CORE WEB API

- REST Calls

HTTP Verb	Activity
GET	Read
POST	Create
PUT	Update
DELETE	Delete

CONTEXT AND STATELESS CALLS

- Context is created with every invocation
- *Attach* Objects and set the state

SUMMARY

- Visual Studio scaffolding support for ASP.NET Web API and OData

THANKS!

- Have a great weekend!

www.cninnovation.com
blog.cninnovation.com
@christiannagel