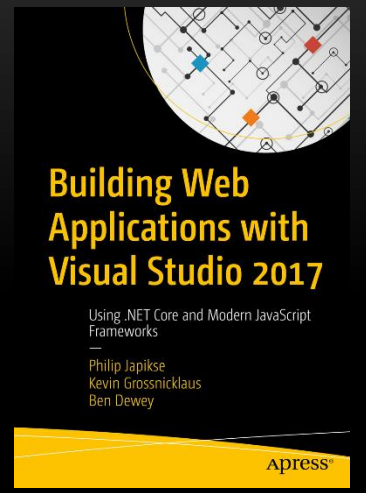
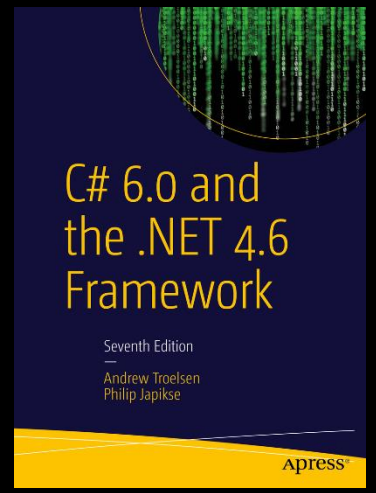
ENTITY FRAMEWORK CORE: WHAT YOU NEED TO KNOW

Philip Japikse (@skimedic)
skimedic@outlook.com
www.skimedic.com/blog
Microsoft MVP, ASPInsider, MCSD, MCDBA, CSM, CSP
Consultant, Teacher, Writer

Phil.About()

- Consultant, Coach, Author, Teacher
  - Lynda.com (http://bit.ly/skimediclyndacourses)
  - Apress.com (http://bit.ly/apressbooks)
- Microsoft MVP, ASPInsider, MCSD, MCDBA, CSM, CSP
- Founder, Agile Conferences, Inc.
  - http://www.dayofagile.org
- President, Cincinnati .NET User's Group





AGENDA

- Entity Framework Project Status
- Which Entity Framework is right for you?
- Getting Started
- Top 10 Features Demos

EF PROJECT STATUS

WHAT IS ENTITY FRAMEWORK CORE 1

- Newest version of Entity Framework - complete re-write from EF 6.x
- Lightweight, Modularized
- Cross Platform (built on .NET Core)

- Just released as RTM (1.1.1)
  - Still some missing features from EF 6.x
  - Check http://bit.ly/ef6_efcore to see the current status

ENTITY FRAMEWORK PROJECT STATUS

Entity Framework 6.2

Entity Framework 6.1

Entity Framework 6.0

Beta 1 coming soon!
Bug fixes, VS2017 support
http://bit.ly/ef6roadmap

Runtime on NuGet
Tooling on Microsoft Download Center
Latest version included in Visual Studio

Entity Framework Core 2.0

Entity Framework Core 1.1

Entity Framework Core 1.0

Entity Framework 5.0

Entity Framework 4.1 – 4.3

New runtime components on NuGet
Core runtime components in .NET
Tooling in Visual Studio

Entity Framework 4

Entity Framework 3.5 SP1

Runtime in .NET Framework
Tooling in Visual Studio

Courtesy of Rowan Miller- https://github.com/rowanmiller/Demo-EFCore

NEW PLATFORMS

.NET FRAMEWORK	.NET CORE	XAMARIN
WINDOWS APPLICATIONS	CROSS-PLATFORM SERVICES	MOBILE APPLICATIONS

.NET STANDARD LIBRARY

COMMON INFRASTRUCTURE

Compilers | Languages | Runtime components

EF6.x | EF Core

TOOLS

Visual Studio

Visual Studio Code

Xamarin Studio

Courtesy of Rowan Miller- https://github.com/rowanmiller/Demo-EFCore

CHOOSING BETWEEN EF 6 AND EF CORE

WHEN CHOOSING EF 6.X VS EF CORE 1, CHOSE WISELY

EF 6.x

- Windows only
- Full featured
  - 8 years of development
  - Patches and minor releases still coming
- Widely used
  - Rich set of database providers
  - Lots of articles, books, examples

EF Core

- Windows or Cross Platform
- Brand new - missing features from EF 6
  - EF Core 1.1 closed the gap
  - Documentation is catching up
- Many Improvements over EF 6.x
  - Much faster performance
  - New features not in EF 6.x
- Works with EF 6.x*

(SOME) MISSING* FEATURES IN CURRENT VERSION OF EF CORE 1

- EDMX Designer
  - Not coming back!
- Alternate inheritance mapping patterns
  - Implemented: Table Per Hierarchy (TPH)
  - Missing: Table Per Type (TPT), Table Per Concrete Type (TPC)
- Complex/Value types
- Spatial Data Types
- Lazy loading
- Command Interception
- Stored Procedure Mapping
- Data Initialization

http://bit.ly/ef6_efcore

IT'S NOT THE RING OF POWER!

EF CORE BASICS

WHAT IS ENTITY FRAMEWORK?

- Entity Framework (EF) is an object-relational mapper that enables .NET developers to work with relational data using domain-specific objects. It eliminates the need for most of the data-access code that developers usually need to write.

THE DATABASE CONTEXT

- Defines tables as DbSets
- Derives from DbContext
  - DbContext class provides the meat of the functionality

USING CODE “FIRST”

- Really means Code “Only” – No EDMX
- Tables are defined by classes
  - Add DbSet<type> to Context for each table
- Fields are defined by properties
  - Restrictions/Rules set with
    - Data Annotations
    - Fluent API in OnModelCreating
- Relations are defined by Classes/Lists for Parent/Children
- Changes are handled with Migrations

CREATING CLASSES AND PROPERTIES

- Create Class per table
  - Add Table attribute to change schema/name (or use Fluent API)
  - Add DbSet<Type> to Context
- Create Properties
  - Primary Key
  - Rowversion – mark with [Timestamp] attribute
- Create Relations

```
public class Category
{
    [Key]
    public int Id { get; set; }
    [StringLength(100)]
    public string CategoryName { get; set; }
    [Timestamp]
    public byte[] TimeStamp { get; set; }
    public List<Product> Products { get; set; }
}
public class Product
{
    [Key]
    public int Id { get; set; }
    public Category Category { get; set; }
}
```

FREQUENTLY USED ATTRIBUTES FOR FIELDS

- Key – sets the primary key
  - Optional if field is named Id or <Type>Id
  - Defaults to Identity - use DatabaseGenerated(DatabaseGeneratedOption)
    - Identity, Computed, or None
- Timestamp – use on byte[] type to create RowVersion
- Required – sets field as non-nullable
- StringLength – sets length on nvarchar fields
- DefaultValue
- DataType(DataType.Text || DataType.Date)
  - Used to set hints on field creation - many more types to choose from

CREATING RELATIONS

- Parent
  - Add List<Type>
- Child
  - Specify [ForeignKey("fieldname")]
  - Not needed if following the standard naming style
- Add <Type>Id
  - If not nullable, will cascade delete
  - Can set cascade options via Fluent API

```
public class Category
{
    [Key]
    public int Id { get; set; }
    public List<Product> Products {get;set;}
}

public partial class Product
{
    [Key]
    public int Id { get; set; }
    [Required]
    public int CategoryId { get; set; }
    [ForeignKey("CategoryId")]
    public Category Category
        { get; set; }
}
```

USING THE FLUENT API

- Must use to set precision on fields
- Use to set cascade delete options
  - Much clearer than the obscure rules based on nullability and required attributes
  - Must use it to set cascade on one to one relationships

```
modelBuilder.Entity<Product>()
   .ToTable("Products", "Catalog");
modelBuilder.Entity<ProductPhoto>()
   .ToTable("ProductPhotos", "Catalog");

modelBuilder.Entity<Product>()
   .Property(x => x.CurrentPrice)
   .HasPrecision(10, 4);

modelBuilder.Entity<Product>()
   .HasOptional(x => x.Image)
   .WithRequired(x => x.ProductParent)
   .WillCascadeOnDelete(true);
```

LOAD DATA FROM DATABASE

- Create new instance of your Context (e.g. ProductContext)
  - Tables are exposed as properties on the context
- Use LINQ to get data from the tables
- Remember when LINQ executes
  - Use ToList(), First(), etc. where appropriate

```
return _db.Products.FirstOrDefault(
 x => x.ProductID == id);
return _db.Products.ToList();
return _db.Products
    .FirstOrDefault();
return _db.Categories
 .Include(x => x.Products)
 .ThenInclude(p=>p.Orders)
 .ToList();
```

UPDATE DATA RECORD

- Must retrieve object from Context
- Update fields
- Call SaveChanges

```
ProductPhoto photo =
 _db.ProductPhotoes.FirstOrDefault(
 x => x.ProductPhotoID == _photoId);

photo. PhotoFileName = "Updated";

_db.SaveChanges();
```

ADD NEW RECORD

- Instantiate a new object
  - Set the properties
- Call Add on DbSet
- Call SaveChanges
- Primary Key is populated by EF*

```
ProductPhoto _photo = new
ProductPhoto
{
    ThumbnailPhotoFileName =
"123456789022",
    ModifiedDate = DateTime.Now
};
_db.ProductPhotos.Add(_photo);
_db.SaveChanges();
```

DELETE A RECORD

- Set entity state to deleted
- OR
- Call Remove on DbSet
  - Must have instance of object to delete

- Call SaveChanges

```
_productContext.ProductPhotos
   .Remove(_photo);
Context.SaveChanges();

Context.Entry(entity).State =
   EntityState.Deleted;

Context.SaveChangesAsync();
```

EF CORE

CONFIGURATION

- Uses DbContextOptionsBuilder
  - instead of just name of connection string
- Constructor Injection

```
public SpyStoreContext(DbContextOptions<SpyStoreContext> options):base(options) {}
```

- OnConfiguring

```
protected override void OnConfiguring(DbContextOptionsBuilder optionsBuilder)
{
    if (!optionsBuilder.IsConfigured)
    {
        optionsBuilder.UseSqlServer($“{connectionString}");
    }
}
```

TOP 10 FEATURES FOR ENTITY FRAMEWORK

TOP 10 FEATURES YOU NEED TO KNOW (IN NO PARTICULAR ORDER)

- Improved Performance in EF Core
- Batching (EFC 1.0+)
- Field Mapping (EFC 1.1)
- Connection Resiliency (EF 6, EFC 1.1)
- Concurrency (EF6, New APIs ECC 1.1)
- Mixed Mode Evaluation (EFC 1.0+)
- FromSql (EF6, EFC 1.0+)
- Improved Migrations (EFC 1.0+)
- Computed Columns (EF6, EFC 1.1*)
- Find (EF, EFC 1.1)

DEMO

Performance Improvements

BATCHING OF STATEMENTS

BATCHING OF STATEMENTS FOR INSERT, UPDATE, DELETE

- EF Core batches multiple statements into a single call
  - Uses table valued parameters to process changes in a single network call
  - Improved performance through reduced network traffic
  - Reduces cost for cloud based databases
- Batch size can be configured through the DbContextOptions

DEMO

Batching of Create, Update, Delete statements

FIELD MAPPING

FIELD MAPPING/BACKING FIELDS

- Allows EF to read and/or write to fields instead of properties
- Conventions
  - [m]_<camel-cased property name>
  - [m]_<property name>
- Fluent API
  - modelBuilder.Entity<Blog>()
    .Property(b=>b.Url)
    .HasField("_theUrl")
- Used when materializing objects
  - Public getters/setters (if they exist) used at other times
- Can control when the fields are used
  - Field
  - FieldDuringConstruction
  - Property

DEMO

Field Mapping

CONNECTION RESILIENCY

CONNECTION RESILIENCY

- Built in retry mechanism defined by relational database providers
  - Default – no retry
  - SqlServerRetryingExecutionStrategy
    - Optimized for SQL Server and SQL Azure
- Custom Execution Strategy
  - Specify retry count and max delay

- Throws RetryLimitExceededException
  - Actual exception is inner exception

DEMO

Connection Resiliency

CONCURRENCY

CONCURRENCY

- SQL Server uses Timestamp (rowversion) properties
  - Coded as a byte[] in C#
- Updates and Deletes are modified
  - Where <pk> = @p1 and <timestamp> = @p2
- Error throws DbUpdateConcurrencyException
  - Provides access to entities not updated/deleted
  - EF Core 1.1 added back familiar API calls
- Developer decides how to handle concurrency errors

DEMO

Concurrency

MIXED MODE QUERY EVALUATION

EF CORE SUPPORTS MIXED EVALUATION

- EF Core supports queries being evaluated on the server and the client
  - What executes where is provider specific
- Useful for including C# functions into the LINQ query/project
- Be careful where the client functions are injected
  - Poor usage can crush performance
- Can't "disable" – can only set EF to throw exceptions
  - Automate testing is vital for discovery
  - Single[OrDefault] – Client side
  - First[OrDefault] – Server side

DEMO

Mixed Mode Query Evaluation

POPULATING MODELS WITH RAW SQL QUERIES

POPULATING MODELS WITH RAW SQL QUERIES

- Models can be populated from raw SQL using FromSql on DbSet<T>
  - Select list names must match the names that properties are mapped to
  - All fields on the model must be returned
- Useful for times when Sprocs or UDFs perform better than LINQ/EF
- Can also populate POCOs that are not tables
  - Must be in the Context as a DbSet<T>
  - Must have a primary key defined
- Can be mixed with LINQ statements

DEMO

FromSQL

MIGRATIONS

EF CORE CONTEXT MIGRATIONS

- Used to modify schema of based on model and SQL Code
  - Can also scaffold existing database into Context and Models
- Supports more than one DbContext in a project
  - E.g. ApplicationDbContext (ASP.NET Identity) and MyDomainModelContext
- Can also create SQL script representing changes to the database

- Note: Migrations only work with projects that emit entry point

EF CORE 1 MIGRATIONS

- No longer stores a hash in the DB
  - <Context>ModelSnapshot.cs

- Run from the command line (or package manager console)
  - dotnet ef migrations [options] [add || list || remove || script]
  - dotnet ef database update [update || drop] [options]

- Reverse Engineer a Database:
  - dotnet ef dbcontext scaffold [arguments] [options]

CHANGES FROM EF6 MIGRATIONS

- The Good
  - No longer uses a hash to check database state
  - ModelSnapshot is C# file that contains all of the DDL
  - Database.Migrate method creates model AND runs all migrations

- The bad?
  - Database Initializers and Configuration Seed method are gone

DEMO

Migrations

COMPUTED COLUMNS

USING COMPUTED COLUMNS IN MODELS

- Same table computed columns supported with EF Core 1.0

```
entity.Property(e => e.LineItemTotal).HasColumnType("money")
  .HasComputedColumnSql("[Quantity]*[UnitCost]");
```

- UDF based computed columns supported with EF Core 1.1

```
entity.Property(e => e.OrderTotal).HasColumnType("money")
  .HasComputedColumnSql("Store.GetOrderTotal([id])");
```

DEMO

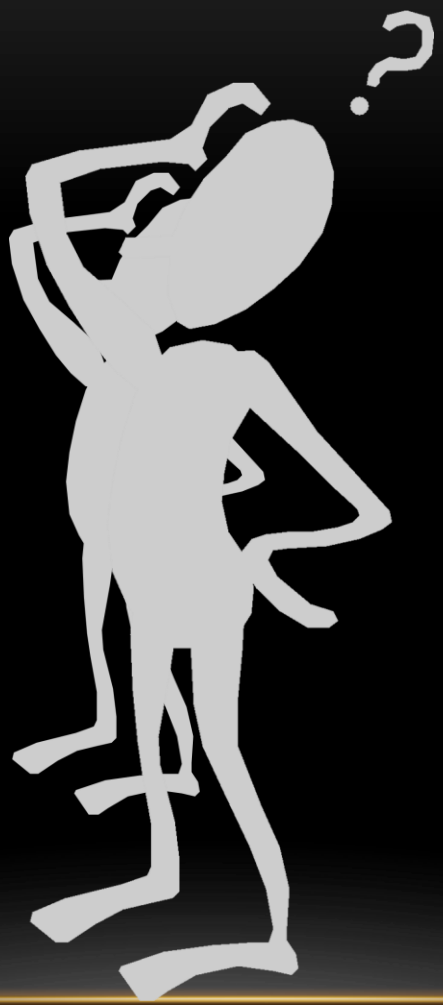
Computed Columns

FIND

DBSET<T> FIND METHOD

- Introduced in EF Core 1.1
  - Largely due to the developer community
- Searches on primary key(s)
  - Returns instance from DbChangeTracker is currently tracked
  - Else calls to database

Questions?



Contact Me

skimedic@outlook.com
www.skimedic.com/blog
www.twitter.com/skimedic

http://bit.ly/skimediclyndacourses
http://bit.ly/apressbooks

www.hallwayconversations.com

Thank You!

Learn more at https://docs.microsoft.com/en-us/ef/

Get the code: https://github.com/skimedic/presentations