ENTITY FRAMEWORK CORE
FOR MERE MORTALS

Philip Japikse (@skimedic)
skimedic@outlook.com
www.skimedic.com/blog

Microsoft MVP, ASPInder, MCSD, MCDBA, CSM, PSM II, PSD
Consultant, Teacher, Writer

All slides copyright Philip Japikse http://www.skimedic.com
Phil.About()

➢ Director of Consulting/Chief Architect
➢ Speaker: http://www.skimedic.com/blog/page/Abstracts.aspx
➢ Microsoft MVP, ASPInder, MCSD, MCDBA, CSM, PSM II, PSD
➢ Founder, Agile Conferences, Inc.
➢ http://www.cincydeliver.org
➢ President, Cincinnati .NET User’s Group
THE CASE FOR OBJECT RELATIONAL MAPPERS (ORMS)
THE PROBLEM

➢ SQL Databases:
  ➢ Store data relationally using scalar values (with some exceptions)

➢ Applications:
  ➢ Model data after the domain using complex, object oriented entities

➢ Translating back and forth takes a lot of code
➢ Data access code isn’t an application differentiator
OBJECT RELATIONAL MAPPING DEFINED

“Object-relational mapping (ORM, O/RM, and O/R mapping tool) in computer science is a programming technique for converting data between incompatible type systems using object-oriented programming languages.”

ORM BENEFITS

➢ ORMs convert relational data to domain models and back

➢ Developers work with classes that support the domain instead of defined by the confines of SQL based data stores.

➢ Modern ORMs typically implement:
  ➢ Unit of Work Pattern, Repository Pattern, Implicit Transaction support
UNIT OF WORK DESIGN PATTERN

Maintains a list of objects affected by a business transaction and coordinates the writing out of changes and the resolution of concurrency problems.

--Martin Fowler
REPOSITORY DESIGN PATTERN

Mediates between the domain and data mapping layers using a collection-like interface for accessing domain objects.

--Martin Fowler
IMPLICIT TRANSACTION SUPPORT

➢ All operations executed in a Unit of Work are wrapped in a transaction
➢ Requires ORM and DBMS support
➢ Explicit transactions can be used for multiple Units of Work
COMMON ARGUMENTS AGAINST ORMS

➢ Too much “magic”
➢ “They are slow”
➢ “Our data is too complex”
➢ “They are hard to learn”
➢ The “DBAs won’t let us”
ORM USE CASES

The Bad

- Reporting solutions
- ETL Operations
- Set based operations

The Good

- CRUD operations
- Forms over data
- Line of business applications
IT’S NOT THE RING OF POWER!
EF PROJECT STATUS
ENTITY FRAMEWORK CORE

➢ Newest version of Entity Framework - complete re-write from EF 6.x
  ➢ Lightweight, Modularized
  ➢ Cross Platform (built on .NET Core)
  ➢ Based on an ‘Opt-in’ model – only load needed packages

➢ EF Core 2.2
  ➢ Many more features added
  ➢ Still some missing features from EF 6.x
EF CORE 3.0/.NET CORE 3

➢ EF Core 3 won’t run on .NET Framework
➢ Only .NET Core 3/NetStandard 2.1
➢ Breaking Changes from 2.x
   ➢ dotnet ef must be manually installed as a global tool
   ➢ dotnet tool install --global dotnet-ef --version <exact-version>
   ➢ FromSql -> FromSqlRaw/FromSqlInterpolated
➢ DbQuery moved to DbSet
➢ Plus many more:

https://docs.microsoft.com/en-us/ef/core/what-is-new/ef-core-3.0/breaking-changes
(SOME) MISSING* FEATURES IN CURRENT VERSION OF EF CORE 2.2

➢ EDMX Designer
  ➢ Not coming back!
➢ Spatial Data Types (2.2)
➢ Command Interception*
➢ Alternate inheritance mapping patterns
  ➢ Implemented: Table Per Hierarchy (TPH)
  ➢ Missing: Table Per Type (TPT), Table Per Concrete Type (TPC)

➢ Stored Procedure Mapping
➢ Some Data Annotations
➢ Raw SQL (Non-Entity Types)
➢ Lazy loading
➢ Data Initialization
➢ Group-By Translation

DATA ANNOTATIONS REMOVED/CHANGED IN EF CORE

**Removed**
- Only works in Fluent API
- Composite Keys
- Index, Composite Indices
- ComplexType
- Fluent API (2.0)
- Owned attribute (2.1)

**Changed**
- Table
- Must specify schema properly
FEATURES ADDED TO EF CORE (NOT IN EF 6)

- Batching of Statements (1.0)
- Shadow State Properties (1.0)
- Alternate Keys (1.0)
- Client side key generation (1.0)
- Mixed Client/Server evaluation (1.0)
- Raw SQL with LINQ (1.0)
- Field Mapping (1.1)

- DbContext Pooling (2.0)
- Like query operator (2.0)
- Global Query Filters (2.0)
- String interpolation with raw SQL (2.0)
- Scalar function mapping (2.0)
- Explicitly compiled LINQ queries (2.0)
- Attach a graph of new and existing entities (2.0)
FEATURES ADDED TO EF CORE IN 2.1 (NOT IN EF 6)

- Query Types w/o keys
- ChangeTracker Events
- Property Value Conversions (2,1)
- Entity ctors with parameters (2.1)
- Eager loading for derived types (2.1)
EF COMPONENTS
MAJOR COMPONENTS OF EF

- DbContext
- ChangeTracker
- DbSet
- Entities
- Database Providers
DBCONTEXT

➢ Represents a session with the database and is used in query and save operations
➢ Database property exposes a DatabaseFacade for db related info and operations
➢ Implements a combination of the Unit of Work and Repository patterns
➢ Provides additional model configuration via Fluent API
➢ Is configured using DbContextOptions/DbContextOptionsBuilder
DBCONTEXT CHANGES FROM EF 6

- Fully embraces dependency injection
- Requires DbContextOptions for configuration
- OnConfiguring provides fall back mechanism
- DbContext Design Time Factory (new)
DBCONTEXT DESIGN TIME FACTORY

- EF Core Tooling Commands requires parameterless ctor or
- IDesignTimeDbContextFactory<TContext>
- Used by EF Migrations to create DbContext
CHANGE TRACKER

- Provides access to change tracking information and operations for entity instances tracked by the context.
- Also tracks the original values.
- Works with the DbContext when SaveChanges is called.
DBSET<T> WHERE T: {ENTITY CLASS}

- Represents a collection of all entities of a given type in a Context.
- Implements IQueryable<T> in addition to collection interfaces
- Can be used to query and save instances of entities.
- LINQ statements used against a DbSet<T> are translated into queries against the data store.
- Some parts might be evaluated in memory depending on the data store
- Items removed, added, or changed in a DbSet are not persisted until SaveChanges is called on the Context.
ENTITIES

- Simple .NET classes designed to model the domain
- Often referred to as Plain Old CLR Objects (POCOs)
- Mapped to the relational data through conventions, data annotations and/or the fluent API
- Relate to other entities in the model through navigation properties
EF CORE PROVIDER MODEL

- EF Core uses a provider model to allow use with different data stores
- Concepts common to most providers are contained in the core components
- Provider specific concepts are in specific components
DATABASE PROVIDERS AVAILABLE

Third Party support:
- MySql (Official, Pomelo)
- SQLite
- PostgreSQL (Third party)
- Db2 (IBM)
- Oracle (paid only)
- MyCat
- Firebird
- SQL Compact

Supported by EF Core directly:
- SQL Server
  - Used in this course
- SQLite
- InMemory
  - Covered in Essential EF Core 2.0 Part 2 with testing
- Cosmos (2.2)
FEATURE DEMOS
CONFIGURATION

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

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

- OnConfiguring

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

➢ EF Core has significant performance improvements over EF 6
➢ Performance is a top priority for the EF Core team

➢ EF Core batches multiple statements into a single call
  ➢ Uses table valued parameters to process changes in a single network call
  ➢ Reduced network traffic and cost (for cloud based databases)

➢ Batch size can be configured through the DbContextOptions
exec sp_executesql N'SET NOCOUNT ON;
UPDATE [Blogs] SET [Url] = @p0 WHERE [BlogId] = @p1;
SELECT @@ROWCOUNT;
UPDATE [Blogs] SET [Url] = @p2 WHERE [BlogId] = @p3;
SELECT @@ROWCOUNT;

DECLARE @toInsert2 TABLE ([Name] nvarchar(max), [Url] nvarchar(max), [_Position] [int]);
INSERT INTO @toInsert2 VALUES (@p4, @p5, 0), (@p6, @p7, 1), (@p8, @p9, 2), (@p10, @p11, 3), (@p12, @p13, 4), (@p14, @p15, 5);

DECLARE @inserted2 TABLE ([BlogId] int, [_Position] [int]);
MERGE [Blogs] USING @toInsert2 AS i ON 1=0 WHEN NOT MATCHED THEN
INSERT ([Name], [Url]) VALUES (i.[Name], i.[Url]) OUTPUT INSERTED.[BlogId], i._Position INTO @inserted2;

SELECT [t].[BlogId] FROM [Blogs] t INNER JOIN @inserted2 i ON ([t].[BlogId] = [i].[BlogId]) ORDER BY [i]._Position;

DBCONTEXT POOLING (2.0)

- Works in conjunction with ASP.NET Core 2
- Creates a pool of DbContext instances
- Resets the ChangeTracker between uses

```csharp
services.AddDbContextPool<BloggingContext>(options =>
    options.UseSqlServer(connectionString));
```
CONNECTION RESILIENCY

- Built in retry mechanism defined by relational database providers
  - SqlServerRetryingExecutionStrategy
  - Optimized for SQL Server and SQL Azure
- Custom Execution Strategy
  - Specify retry count and max delay
- Throws RetryLimitExceededException
- Actual exception is inner exception

```csharp
optionsBuilder.UseSqlServer(connectionString, options=>options.EnableRetryOnFailure());
```
CONCURRENCY

➢ SQL Server uses Timestamp (rowversion) properties (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
➢ Developer decides how to handle concurrency errors

```csharp
public abstract class EntityBase
{
    //...
    [Timestamp]
    public byte[] TimeStamp { get; set; }
}
```
GLOBAL QUERY FILTERS (2.0)

- Model level filters defined directly on the model
- Automatically applied to any queries on that type
- Also applied to indirect queries (e.g. using Include or ThenInclude)
- Can be used for soft deletes or multi-tenancy
- Override with IgnoreQueryFilters()

```csharp
public class BloggingContext : DbContext
{
    public DbSet<Post> Posts { get; set; }
    public int TenantId { get; set; }
    protected override void OnModelCreating(ModelBuilder modelBuilder)
    {
        modelBuilder.Entity<Post>().HasQueryFilter( 
            p => !p.IsDeleted && p.TenantId == this.TenantId );
    }
}
```
STRING INTERPOLATION WITH RAW SQL QUERIES (2.0)

➢ Implemented in `FromSql` and `ExecuteSqlCommand`

➢ C# string interpolation items get converted into SQL parameters

```csharp
var city = "London";
var contactTitle = "Sales Representative";
using (var context = CreateContext())
{
    context.Set<Customer>()
        .FromSql($"SELECT * FROM "Customers""
                WHERE "City" = {city} AND
                "ContactTitle" = {contactTitle}")
        .ToArray();
}
```
POPULATING MODELS WITH RAW SQL QUERIES

➢ Models can be populated from raw SQL using FromSql on DbSet<T>
  ➢ All fields must be returned and the names must match
  ➢ Useful for times when Sprocs or UDFs perform better than LINQ/EF
  ➢ Can be mixed with LINQ statements*

➢ Can also populate POCOs that are not tables (Changing in 2.1)
  ➢ Must be in the Context as a DbSet<T>
  ➢ Must have a primary key defined
DBQUERY TYPES

➢ Do not have keys defined
➢ Are read-only (cannot be inserted, deleted or updated)
➢ Can be returned directly by queries.

➢ Usage scenarios:
  ➢ Mapping to views without primary keys
  ➢ Mapping to tables without primary keys
  ➢ Mapping to queries defined in the model
  ➢ Serving as the return type for FromSql() queries
HELPER FUNCTIONS (EF.FUNCTIONS)

- Implemented at the database provider level
- Like Query Operator (2.0)
- You must add in the % yourself
- DateFunctions (2.1)
- FreeText (2.1)
- Contains (2.2)

```csharp
var customers =
    from c in context.Customers where
        EF.Functions.Like(c.Name, "a%")
    select c;

//creates this query
SELECT [c].[Id], [c].[Name]
FROM [Customers] AS [c]
WHERE [c].[Name] LIKE N'a%';
```
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
EF CORE SUPPORTS MIXED EVALUATION (CHANGES IN 3.0)

- 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
- Can’t “disable” – can only set EF to throw exceptions
- Automate testing is vital for discovery

```csharp
optionsBuilder.UseSqlServer(connectionString)
    .ConfigureWarnings(
        warnings => warnings.Throw(
            RelationalEventId.QueryClientEvaluationWarning));
```
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
EF CORE 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]`
public BloggingContext()
{
    ChangeTracker.StateChanged += ChangeTracker_StateChanged;
    ChangeTracker.Tracked += ChangeTracker_Tracked;
}

private void ChangeTracker_Tracked(object sender, EntityTrackedEventArgs e)
{
    //See Demo
}

private void ChangeTracker_StateChanged(object sender, EntityStateStateChangedEventArgs e)
{
    //See Demo
}
Contact Me
skimedic@outlook.com
www.skimedic.com/blog
www.twitter.com/skimedic


www.hallwayconversations.com

Questions?

Thank You!

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

Get the code: https://github.com/skimedic/presentations/tree/master/DOTNETCORE/EFCoreSamples

All slides copyright Philip Japikse http://www.skimedic.com