Neues in EFCore 6

Olaf Lischke

Speaker: Olaf Lischke

macht .NET, seit es .NET gibt

versucht, Projekte und Seminare zu kombinieren

singt Tenor in drei Chören

zockte schon auf dem ZX 81, heute ausschließlich auf PC

fotografiert, seit er eine Kamera halten kann

fliegt, wenn Wetter und Zeit es zulassen (TMG/SEP)

Themenüberblick

- Neue Features
- Performance-Steigerungen
- Verbesserungen am Tooling
- Verbesserungen an LINQ
- Änderungen an einzelnen Providern (Auswahl)
- Allgemeine Breaking Changes

Neue Features – Teil I

- n:m-Abstraktion:
  Zuordnungstabellen ohne weitere Spalten werden abstrahiert
  Konfiguration im ModelBuilder vereinfacht

- neue Data-Annotations:

```
[Unicode]
[Precision(precision: 10, scale:2)]
[EntityTypeConfiguration(typeof(BookConfiguration))]
```

-Klassenattribut für `Book`-Klasse

verweist auf `BookConfiguration : IEntityTypeConfiguration<Book>`

- Savepoints API
  für PostgreSQL (seit EFCore 5) und SQLite vollständig,
  für SQL Server nur Save & Rollback, kein Release von Savepoints

```
transaction
    .Save("MySavepoint") || .Rollback("MySavepoint")
```

Neue Features – Teil II

- Pre-Konventions konfigurierbar

  Eigener Handler in DBContext:

```
protected override void ConfigureConventions(ModelConfigurationBuilder configurationBuilder)
{
    // Pre-convention model configuration goes here
}
```

  FluentApi für die globale Konfiguration von Typen:

```
configurationBuilder.Properties<string>().AreUnicode(false);
configurationBuilder.IgnoreAny<PropertyToIgnore>();
```

  und vieles mehr...

- Migration Bundles

  Migrations als .exe erstellen

  EF Core CLI: `dotnet ef migrations bundle` || PMC: `Bundle-Migration`

Performance-Steigerungen – Teil I

- TechEmpower Fortunes benchmark

  https://www.techempower.com/benchmarks/#section=data-r20

  EFCore 6 nur noch 5% hinter Dapper

- Maßnahmen:
  - DbContext-Pooling
  - Recycling/GarbageCollection für zugrundeliegende ADO.NET-Elemente
  - Logging-Suppression
  - Thread-Safety-Checks reduziert
  - Details: DevBlog: announcing-entity-framework-core-6-0-preview-4-performance-edition

```
services.AddDbContextPool<MyContext>(options =>
    options.UseNpgsql(connectionString));
```

Performance-Steigerungen – Teil II

- Compiled Model

  Beschleunigt Startgeschwindigkeit der App

  → EF Core CLI:

  ```
  dotnet ef dbcontext optimize -c MyContext -o MyFolder -n My.Namespace
  ```

  → Package Manager Konsole (PMC):

  ```
  Optimize-DbContext -Context MyContext -OutputDir MyFolder -Namespace My.Namesp
  ```

  → Konfiguration:

  ```
  optionsBuilder.UseModel(My.Namespace.MyContextModel.Instance);
  ```

  <u>Einschränkungen:</u>

  - Kein Lazy Loading, keine Change-Tracking Proxies
  - Keine Global Query Filters
  - Neu-Kompilierung nach Model-Änderungen erforderlich

Verbesserungen am Tooling

- Reverse-Engineering (formerly known as "Database First"):

  Übernahme der Kommentare aus dem Datenmodell

  Unterstützung für Nullable Reference Types (NRT)

- Model Building:

  `.HasConversion<ValueConverter>()` inkl. null-Unterstützung

Verbesserungen an LINQ/IQueryable

- `GroupBy()`
  bislang sehr häufig erst im RAM ausgeführt, nun verstärkt in SQL übersetzt
  Unterstützung auch für
    - FirstOrDefault
    - Navigation-Properties
    - Details und jede Menge Samples: #linq-query-enhancements

- `String.Concat()` mit mehreren Parametern
  verwendbar in `where`-Ausdrücken, entsprechend in SQL übersetzt

- `EF.Functions.Random()` wird zu `Rand()` in SQL

Neuerungen für PostgreSQL

- "UTC everywhere"
  - Zeitangabe gilt immer als UTC-Zeitangabe, wenn nichts anderes angegeben.
  - **Daher:** DateTime (C#) → `timestamptz`, keine implizite Konvertierungen mehr
  - Altes Verhalten erzwingen: `AppContext.SetSwitch("Npgsql.EnableLegacyTimestampBehavior", true);`
- `DateOnly`/`TimeOnly`-Unterstützung
- NodaTime-Unterstützung (https://nodatime.org/)
- Neue Postgres-Typen: `multirange → NpgsqlRange<T>, ltree`

Bisher:

DateTime - Kind	PostgreSQL Typ
UTC	timestamptz
Local	timestamp
Unspecified	timestamp

Neuerungen für SQL Server- Teil I

- Unterstützung für sparse columns (NULL-optimierte Spalten):

```
modelBuilder.Entity<Dog>()
            .Property(e => e.IsTrained)
            .IsSparse();
```

- LINQ:
  - `IsNullOrWhitespace()`

    Generiertes SQL enthält keine

    ```
    (LTRIM(RTRIM([u].[FirstName])) = N'')
    ```

    Konstrukte mehr
  - Freitext-Suche: `Contains` und `FreeText` nicht mehr zwingend `String`

    Prima für JSON/XML mit Convertern (s.o.)

Neuerungen für SQL Server – Teil II

- Temporal Tables


```
modelBuilder.Entity<Employee>()
                .ToTable("Employees",
                        e => e.IsTemporal(s => {
                                                s.HasPeriodStart("ValidFrom");
                                                s.HasPeriodEnd("ValidTo");
                                                s.UseHistoryTable("EmployeeHistoricalData
                                                }
                                              )
                        );
```

Neuerungen für SQLite

- Connection Pooling

  Obwohl Datenbank "nur" Dateisystemzugriff, setzt DbContext hier jetzt auch auf Connection Pooling

- `ToString()` übersetzt in SQL

  `ToString()` in LINQ wird zu `CAST(… AS TEXT)` in SQL

- `DateOnly`/`TimeOnly`-Unterstützung

Allgemeine Breaking Changes

- Abhängige, optionale Objekte, die sich eine Tabelle teilen…
- n:m-Zuordnungen, die nur aus zwei Spalten bestehen…
- Zuordnung von OnDelete() korrigiert
- `DbSet` ist nicht mehr `IAsyncEnumerable`

Vielen Dank!

Slides auf
https://github.com/olaflischke/bastaspring2022

Für spätere Fragen:
olaf.lischke@lischke-edv.de