.NET Entity Framework Core 10

Изменения и новые возможности

Дмитрий Александрович Корзанов

В этом докладе

- Комплексные типы
- Трансляция LINQ и SQL
- ExecuteUpdateAsync: json и lambda
- Именованные фильтры
- Улучшения безопастности
- SQL Azure и SQL Server
- Azure Cosmos DB для NoSQL

Комплексные типы

Разделение таблиц и JSON

```
public class Customer
{
    ...
    public Address ShippingAddress { get; set; }
    public Address? BillingAddress { get; set; }
}
```

Комплексные типы

Разделение таблиц и JSON

```
modelBuilder.Entity<Customer>(b =>
{
    b.ComplexProperty(c => c.ShippingAddress);
    b.ComplexProperty(c => c.BillingAddress);
});
```

```
CREATE TABLE [Customers] (
    [Id] int NOT NULL IDENTITY,
    [BillingAddress_City] nvarchar(max) NOT NULL,
    [BillingAddress_PostalCode] nvarchar(max) NOT NULL,
    [BillingAddress_Street] nvarchar(max) NOT NULL,
    [BillingAddress_StreetNumber] int NOT NULL,
    [ShippingAddress_City] nvarchar(max) NOT NULL,
    [ShippingAddress_PostalCode] nvarchar(max) NOT NULL,
    [ShippingAddress_Street] nvarchar(max) NOT NULL,
    [ShippingAddress_StreetNumber] int NOT NULL,
    CONSTRAINT [PK_Customers] PRIMARY KEY ([Id])
);
```

Комплексные типы

Разделение таблиц и JSON

```
modelBuilder.Entity<Customer>(b =>
{
    b.ComplexProperty(c => c.ShippingAddress, c => c.ToJson());
    b.ComplexProperty(c => c.BillingAddress, c => c.ToJson());
});
```

```
CREATE TABLE [Customers] (
    [Id] int NOT NULL IDENTITY,
    [ShippingAddress] json NOT NULL,
    [BillingAddress] json NOT NULL,
    CONSTRAINT [PK_Customers] PRIMARY KEY ([Id])
);
```

Комплексные типы

Поддержка структур

```
public class Customer
{
    ...
    public Address ShippingAddress { get; set; }
    public Address? BillingAddress { get; set; }
}
```

```
public struct Address
{
    public required string Street { get; set; }
    public required string City { get; set; }
    public required string ZipCode { get; set; }
}
```

Трансляция LINQ и SQL

Параметризованные коллекции

До EF 8

```
int[] ids = [1, 2, 3];
var blogs = await context.Blogs
    .Where(b => ids.Contains(b.Id))
    .ToListAsync();
```

```
SELECT [b].[Id], [b].[Name]
FROM [Blogs] AS [b]
WHERE [b].[Id] IN (1, 2, 3)
```

Трансляция LINQ и SQL

Параметризованные коллекции

EF 8

```
int[] ids = [1, 2, 3];
var blogs = await context.Blogs
    .Where(b => ids.Contains(b.Id))
    .ToListAsync();
```

```
@__ids_0='[1,2,3]'

SELECT [b].[Id], [b].[Name]
FROM [Blogs] AS [b]
WHERE [b].[Id] IN (
    SELECT [i].[value]
    FROM OPENJSON(@__ids_0) WITH ([value] int '$') AS [i]
)
```

Трансляция LINQ и SQL

Параметризованные коллекции

EF 10

```
int[] ids = [1, 2, 3];
var blogs = await context.Blogs
    .Where(b => EF.Constant(ids).Contains(b.Id))
    .ToListAsync();
```

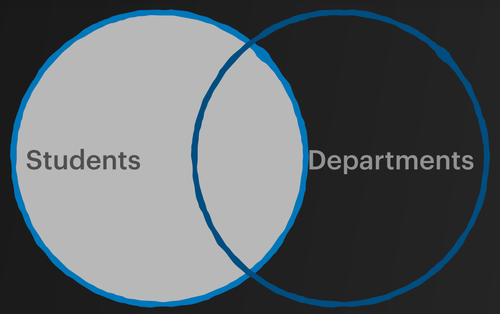
```
protected override void OnConfiguring(
    DbContextOptionsBuilder optionsBuilder)
        => optionsBuilder.UseSqlServer(
            "<CONNECTION STRING>”,
            o => o.UseParameterizedCollectionMode(
                ParameterTranslationMode.Constant));
```

```
SELECT [b].[Id], [b].[Name]
FROM [Blogs] AS [b]
WHERE [b].[Id] IN (@ids1, @ids2, @ids3, @ids4, @ids5, @ids6, @ids7)
```

Трансляция LINQ и SQL

LeftJoin и RightJoin

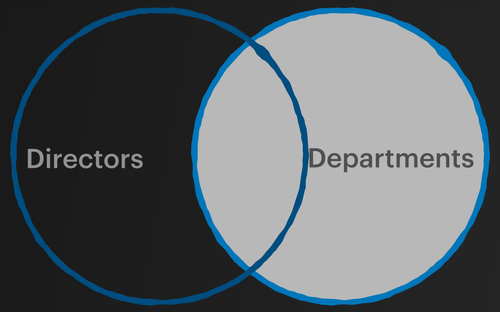
```
var query = context.Students
    .LeftJoin(
        context.Departments,
        student => student.DepartmentID,
        department => department.ID,
        (student, department) => new
        {
            Student = student.FullName,
            Department = department?.Name ?? "[NONE]"
        });
```



Трансляция LINQ и SQL

LeftJoin и RightJoin

```
var query = context.Directors
    .RightJoin(
        context.Departments,
        director => director.DepartmentID,
        department => department.ID ,
        (director, department) => new
        {
            Director = director?.FullName ?? "[NONE]",
            Department = department.Name
        });
```



ExecuteUpdateAsync

Обновление свойств json

```
public class Blog
{
    ...
    public BlogDetails Details { get; set; }
}

public class BlogDetails
{
    public int Views { get; set; }
}

modelBuilder.Entity<Blog>().ComplexProperty(b => b.Details, bd => bd.ToJson());
```

EF метод

```
await context.Blogs.ExecuteUpdateAsync(
    s => s.SetProperty(b => b.Details.Views, b => b.Details.Views + 1));
```

Команда SQL Server

```
UPDATE [b]
SET [Details].modify('$.Views', JSON_VALUE([b].[Details], '$.Views' RETURNING int) + 1)
FROM [Blogs] AS [b]
```

ExecuteUpdateAsync

Простые лямбды

```
Expression<Func<SetPropertyCalls<Blog>, SetPropertyCalls<Blog>>> setters =
    s => s.SetProperty(b => b.Views, 8);

if (nameChanged)
{
    var blogParameter = Expression.Parameter(typeof(Blog), "b");

    setters = Expression.Lambda<Func<SetPropertyCalls<Blog>, SetPropertyCalls<Blog>>>(
        Expression.Call(
            instance: setters.Body,
            methodName: nameof(SetPropertyCalls<Blog>.SetProperty),
            typeArguments: [typeof(string)],
            arguments:
            [
                Expression.Lambda<Func<Blog, string>>(
                    Expression.Property(blogParameter, nameof(Blog.Name)), blogParameter),
                Expression.Constant("foo")
            ]),
        setters.Parameters);
}

await context.Blogs.ExecuteUpdateAsync(setters);
```

ExecuteUpdateAsync

Простые лямбды

```
await context.Blogs.ExecuteUpdateAsync(s =>
{
    s.SetProperty(b => b.Views, 8);
    if (nameChanged)
    {
        s.SetProperty(b => b.Name, "foo");
    }
});
```

Именованные фильтры

```
modelBuilder.Entity<Blog>()
    .HasQueryFilter("SoftDeletionFilter", b => !b.IsDeleted)
    .HasQueryFilter("TenantFilter", b => b.TenantId == tenantId);
```

```
var allBlogs = await context.Blogs
    .IgnoreQueryFilters([“SoftDeletionFilter"])
    .ToListAsync();
```

Улучшения безопасности

Логи без встроенных констант

Обычный запрос

```
Task<List<User>> GetUsersByRoles(BlogContext context, string[] roles)
    => context.Users
        .Where(b => roles.Contains(b.Role))
        .ToListAsync();
```

Запрос с константами

```
Task<List<User>> GetUsersByRoles(BlogContext context, string[] roles)
    => context.Users
        .Where(b => EF.Constant(roles).Contains(b.Role))
        .ToListAsync();
```

Улучшения безопасности

Логи без встроенных констант

SQL запрос

```
SELECT [u].[Id], [u].[Role]
FROM [Users] AS [u]
WHERE [u].[Role] IN (N'Administrator', N'Manager')
```

Лог с константами

```
SELECT [u].[Id], [u].[Role]
FROM [Users] AS [u]
WHERE [u].[Role] IN (?, ?)
```

Улучшения безопасности Предупреждения о SQL инъекциях

```
var users = context.Users.FromSqlRaw(
    “SELECT * FROM Users WHERE [" + fieldName + "] IS NULL");
```

SQL Azure и SQL Server

Поддержка поиска векторов

Создание модели

```
public class Blog
{
    ...
    public string Name { get; set; }

    [Column(TypeName = "vector(1536)")]
    public SqlVector<float> Embedding { get; set; }
}
```

SQL Azure и SQL Server

Поддержка поиска векторов

Наполнение данными

```
IEmbeddingGenerator<string, Embedding<float>> embeddingGenerator =
            new SampleGenerator();

var embedding = await embeddingGenerator
    .GenerateVectorAsync(“Complex types, ExecuteUpdateAsync...”);
context.Blogs.Add(new Blog
{
    Name = “New Entity Framework features",
    Embedding = new SqlVector<float>(embedding)
});
await context.SaveChangesAsync();
```

SQL Azure и SQL Server

Поддержка поиска векторов

Получение результата

```
var vector = GenerateVector("update json by ExecuteUpdateAsync");
var topSimilarBlogs = context.Blogs
    .OrderBy(b =>
        EF.Functions.VectorDistance("cosine", b.Embedding, vector))
    .Take(3)
    .ToListAsync();
```

SQL Azure и SQL Server

Поддержка типов json

Создание модели

```
public class Blog
{
    public int Id { get; set; }
    public required BlogDetails Details { get; set; }
}

public class BlogDetails
{
    public int Viewers { get; set; }
}

modelBuilder.Entity<Blog>()
    .ComplexProperty(b => b.Details, b => b.ToJson());
```

SQL Azure и SQL Server

Поддержка типов json

Создание таблицы по модели

```
CREATE TABLE [Blogs] (
    [Id] int NOT NULL IDENTITY,
    [Details] json NOT NULL,
    CONSTRAINT [PK_Blogs] PRIMARY KEY ([Id])
);
```

Если уже есть колонки nvarchar для работы с данными JSON, то они по-умолчанию мигрируют на json

SQL Azure и SQL Server

Поддержка типов json

```
var highlyViewedBlogs = await context.Blogs
    .Where(b => b.Details.Viewers > 3)
    .ToListAsync();
```

```
SELECT [b].[Id], [b].[Name], [b].[Tags], [b].[Details]
FROM [Blogs] AS [b]
WHERE JSON_VALUE([b].[Details], '$.Viewers' RETURNING int) > 3
```

SQL Azure и SQL Server — Имена CONSTRAINT по-умолчанию

Использование HasDefaultValueSql

```
modelBuilder.Entity<Post>()
    .Property(p => p.CreatedDate)
    .HasDefaultValueSql("GETDATE()", "DF_Post_CreatedDate");
```

При создании новой таблицы

```
CREATE TABLE [Posts] (
    [Id] int NOT NULL IDENTITY,
    [CreatedDate] DATETIME2
        CONSTRAINT DF_Posts_CreateDate DEFAULT GETDATE(),
    CONSTRAINT [PK_Posts] PRIMARY KEY ([Id])
);
```

При изменении таблицы

```
ALTER TABLE [Posts]
ADD CONSTRAINT [DF_Posts_CreateDate]
DEFAULT GETDATE() FOR [CreatedDate];
```

Azure Cosmos DB для NoSQL

Полнотекстовый поиск

Подключение функционала

```
public class Blog
{
    public string Contents { get; set; }
}

modelBuilder.Entity<Blog>(b =>
{
    b.Property(x => x.Contents).EnableFullTextSearch();
    b.HasIndex(x => x.Contents).IsFullTextIndex();
});
```

Azure Cosmos DB для NoSQL

Полнотекстовый поиск

Использование функционала

```
var oneBlogs = await context.Blogs
    .Where(x => EF.Functions.FullTextContains(x.Contents, “one”))
    .ToListAsync();
var oneAndTwoBlogs = await context.Blogs
    .Where(x => EF.Functions.FullTextContainsAll(x.Contents, [“one”, “two”]))
    .ToListAsync();
var oneOrTwoBlogs = await context.Blogs
    .Where(x => EF.Functions.FullTextContainsAny(x.Contents, [“one”, “two”]))
    .ToListAsync();
var moreThenTenMatchesBlogs = await context.Blogs
    .Where(x => EF.Functions.FullTextScore(x.Contents, [“one”, “two”]) >= 10)
    .ToListAsync();
```

Azure Cosmos DB для NoSQL

Гибридный поиск

```
public class Blog
{
    ...
    public string Contents { get; set; }
    public float[] Vector { get; set; }
}

modelBuilder.Entity<Blog>(b =>
{
    b.Property(x => x.Contents).EnableFullTextSearch();
    b.HasIndex(x => x.Contents).IsFullTextIndex();
});
modelBuilder.Entity<Blog>()
    .Property(b => b.Vector)
    .IsVector(DistanceFunction.Cosine, dimensions: 1536);
```

Azure Cosmos DB для NoSQL

Гибридный поиск

Функция взаимного ранжирования

```
float[] vector = GenerateVector("update json by ExecuteUpdateAsync");
var hybrid = await context.Blogs.OrderBy(x => EF.Functions.Rrf(
        EF.Functions.FullTextScore(x.Contents, "database"),
        EF.Functions.VectorDistance(x.Vector, vector)))
    .Take(10)
    .ToListAsync();
```

Links

What's New in EF Core 10

https://learn.microsoft.com/ef/core/what-is-new/ef-core-10.0/whatsnew

Breaking changes in EF Core 10

https://learn.microsoft.com/ef/core/what-is-new/ef-core-10.0/breaking-changes

PostgreSQL version 10.0 Release notes

https://www.npgsql.org/efcore/release-notes/10.0.html

Global Query Filters

https://learn.microsoft.com/ef/core/querying/filters

YouTube videos:

- Remigiusz Zalewski: EF Core 10 Is INSANE - Overview of Game Changer features
- Milan Jovanović: Named Query Filters in EF Core 10 Are a Game Changer
- Israel Quiroz: What's New in EF Core 10! (Full Release Overview)

Simplify as much as possible
Make life easier

LinkedIn: https://linkedin.com/in/korzanov