EF Core: Global Query Filters

Practical Usages

Mayis Harutyunyan, Tigran Davtyan

Agenda

- Query Filters overview
- Query Filters usage
- Query Filters Practical Examples
  - Soft Deletion
  - Multi-tenancy
- Query Filters Problems
- Applying Query Filters on many Entities - Demo
- Combining Multiple Query Filters - Demo

EF Core: Global Query Filters

Overview

- They are LINQ query predicates
- Started from EF Core 2
- They are defined in OnModelCreating (usually)
- EF Core automatically applies on Entity
- Include/navigation property

EF Core: Global Query Filters

Usage

Basic usage:

```
modelBuilder.Entity<Campaign>().HasQueryFilter(p => !p.IsDeleted &&
EF.Property<string>(b, "_tenantId") == _tenantId);

modelBuilder.Entity<Campaign>().HasQueryFilter(b =>
EF.Property<string>(b, "_tenantId") == _tenantId);
```

Navigation Property usage:

```
modelBuilder.Entity<Campaign>().HasQueryFilter(b => b.Recipients.Count >
0);
```

Disabling Filters

```
db.Campaigns.Include(b => b.Recipients).IgnoreQueryFilters()
```

EF Core: Global Query Filters

Navigation Example

```
modelBuilder.Entity<Blog>().HasMany(b => b.Posts).WithOne(p => p.Blog);
modelBuilder.Entity<Blog>().HasQueryFilter(b => b.Posts.Count > 0);
modelBuilder.Entity<Post>().HasQueryFilter(p => p.Title.Contains(“fish"));

var filteredBlogs = db.Blogs.ToList();

SELECT [b].[BlogId], [b].[Name], [b].[Url]
FROM [Blogs] AS [b]
WHERE (
    SELECT COUNT(*)
    FROM [Posts] AS [p]
    WHERE ([p].[Title] LIKE N'%fish%') AND ([b].[BlogId] = [p].[BlogId])) > 0
```

EF Core: Global Query Filters

Problems

It is currently not possible to define multiple query filters on the same entity - only the last one will be applied. However, you can define a single filter with multiple conditions using the logical `AND` operator

Currently EF Core does not detect cycles in global query filter definitions, so you should be careful when defining them. If specified incorrectly, cycles could lead to infinite loops during query translation.

Using required navigation to access entity which has global query filter defined may lead to unexpected results.

EF Core: Global Query Filters

Practical Usage - Soft Deletion

- The main benefits of using soft delete in your application are inadvertent deletes can be restored and history is preserved.
- You can combine soft delete with other uses of Query Filters, like multi-tenant uses but you need to be more careful when you are looking for soft deleted entries.
- Don’t soft delete a one-to-one entity class as it can cause problems.
- For entity classes that has relationships you need to consider what should happen to the dependant relationships when the top entity class is soft deleted.

EF Core: Global Query Filters

Practical Usage - MultiTenancy

One way to implement a multi-tenant application is to use a discriminator column (aka a `tenant_id` column on every table). This is a risky proposition. Every query must remember to filter by the `tenant_id`. One missed query and you expose data from one tenant to another.

```
modelBuilder.Entity<Campaign>().HasQueryFilter(b => b.TenantId == _tenantId);
```

```
- Data Isolation
```

EF Core: Global Query Filters

Required Relation Problems

```
modelBuilder.Entity<Blog>().HasMany(b => b.Posts).WithOne(p => p.Blog).IsRequired();
modelBuilder.Entity<Blog>().HasQueryFilter(b => b.Url.Contains("fish"));

db.Blogs.Add(
    new Blog
    {
        Url = "http://sample.com/blogs/fish",
        Posts = new List<Post>
        {
            new Post { Title = "Fish care 101" },
            new Post { Title = "Caring for tropical fish" },
            new Post { Title = "Types of ornamental fish" }
        }
    });

db.Blogs.Add(
    new Blog
    {
        Url = "http://sample.com/blogs/cats",
        Posts = new List<Post>
        {
            new Post { Title = "Cat care 101" },
            new Post { Title = "Caring for tropical cats" },
            new Post { Title = "Types of ornamental cats" }
        }
    });

var allPosts = db.Posts.ToList();
var allPostsWithBlogsIncluded = db.Posts.Include(p => p.Blog).ToList();

SELECT [p].[PostId], [p].[BlogId], [p].[Content], [p].[IsDeleted], [p].[Title], [t].[BlogId], [t].[Name], [t].[Url]
FROM [Posts] AS [p]
INNER JOIN (
    SELECT [b].[BlogId], [b].[Name], [b].[Url]
    FROM [Blogs] AS [b]
    WHERE [b].[Url] LIKE N'%fish%'
) AS [t] ON [p].[BlogId] = [t].[BlogId]
```

Questions / Answers