Eager Loading of Related Data

Article • 01/19/2023

Eager loading

You can use the `Include` method to specify related data to be included in query results. In the following example, the blogs that are returned in the results will have their `Posts` property populated with the related posts.

C#

```
using (var context = new BloggingContext())
{
    var blogs = context.Blogs
        .Include(blog => blog.Posts)
        .ToList();
}
```

> **Tip**
>
> Entity Framework Core will automatically fix-up navigation properties to any other entities that were previously loaded into the context instance. So even if you don't explicitly include the data for a navigation property, the property may still be populated if some or all of the related entities were previously loaded.

You can include related data from multiple relationships in a single query.

C#

```
using (var context = new BloggingContext())
{
    var blogs = context.Blogs
        .Include(blog => blog.Posts)
        .Include(blog => blog.Owner)
        .ToList();
}
```

> **Caution**

Eager loading a collection navigation in a single query may cause performance issues. For more information, see **Single vs. split queries**.

Including multiple levels

You can drill down through relationships to include multiple levels of related data using the `ThenInclude` method. The following example loads all blogs, their related posts, and the author of each post.

C#

```
using (var context = new BloggingContext())
{
    var blogs = context.Blogs
        .Include(blog => blog.Posts)
        .ThenInclude(post => post.Author)
        .ToList();
}
```

You can chain multiple calls to `ThenInclude` to continue including further levels of related data.

C#

```
using (var context = new BloggingContext())
{
    var blogs = context.Blogs
        .Include(blog => blog.Posts)
        .ThenInclude(post => post.Author)
        .ThenInclude(author => author.Photo)
        .ToList();
}
```

You can combine all of the calls to include related data from multiple levels and multiple roots in the same query.

C#

```
using (var context = new BloggingContext())
{
    var blogs = context.Blogs
        .Include(blog => blog.Posts)
        .ThenInclude(post => post.Author)
```

```
        .ThenInclude(author => author.Photo)
        .Include(blog => blog.Owner)
        .ThenInclude(owner => owner.Photo)
        .ToList();
}
```

You may want to include multiple related entities for one of the entities that is being included. For example, when querying `Blogs`, you include `Posts` and then want to include both the `Author` and `Tags` of the `Posts`. To include both, you need to specify each include path starting at the root. For example, `Blog -> Posts -> Author` and `Blog -> Posts -> Tags`. It doesn't mean you'll get redundant joins; in most cases, EF will combine the joins when generating SQL.

C#

```
using (var context = new BloggingContext())
{
    var blogs = context.Blogs
        .Include(blog => blog.Posts)
        .ThenInclude(post => post.Author)
        .Include(blog => blog.Posts)
        .ThenInclude(post => post.Tags)
        .ToList();
}
```

Tip

You can also load multiple navigations using a single `Include` method. This is possible for navigation "chains" that are all references, or when they end with a single collection.

C#

```
using (var context = new BloggingContext())
{
    var blogs = context.Blogs
        .Include(blog => blog.Owner.AuthoredPosts)
        .ThenInclude(post => post.Blog.Owner.Photo)
        .ToList();
}
```

Filtered include

When applying Include to load related data, you can add certain enumerable operations to the included collection navigation, which allows for filtering and sorting of the results.

Supported operations are: `Where`, `OrderBy`, `OrderByDescending`, `ThenBy`, `ThenByDescending`, `Skip`, and `Take`.

Such operations should be applied on the collection navigation in the lambda passed to the Include method, as shown in example below:

C#

```
using (var context = new BloggingContext())
{
    var filteredBlogs = context.Blogs
        .Include(
            blog => blog.Posts
                .Where(post => post.BlogId == 1)
                .OrderByDescending(post => post.Title)
                .Take(5))
        .ToList();
}
```

Each included navigation allows only one unique set of filter operations. In cases where multiple Include operations are applied for a given collection navigation (`blog.Posts` in the examples below), filter operations can only be specified on one of them:

C#

```
using (var context = new BloggingContext())
{
    var filteredBlogs = context.Blogs
        .Include(blog => blog.Posts.Where(post => post.BlogId == 1))
        .ThenInclude(post => post.Author)
        .Include(blog => blog.Posts)
        .ThenInclude(post => post.Tags.OrderBy(postTag =>
postTag.TagId).Skip(3))
        .ToList();
}
```

Alternatively, identical operations can be applied for each navigation that is included multiple times:

C#

```
using (var context = new BloggingContext())
{
    var filteredBlogs = context.Blogs
        .Include(blog => blog.Posts.Where(post => post.BlogId == 1))
        .ThenInclude(post => post.Author)
        .Include(blog => blog.Posts.Where(post => post.BlogId == 1))
        .ThenInclude(post => post.Tags.OrderBy(postTag =>
postTag.TagId).Skip(3))
        .ToList();
}
```

⊗ Caution

In case of tracking queries, results of Filtered Include may be unexpected due to **navigation fixup**. All relevant entities that have been queried for previously and have been stored in the Change Tracker will be present in the results of Filtered Include query, even if they don't meet the requirements of the filter. Consider using `NoTracking` queries or re-create the DbContext when using Filtered Include in those situations.

Example:

C#

```
var orders = context.Orders.Where(o => o.Id > 1000).ToList();

// customer entities will have references to all orders where Id > 1000, rather
than > 5000
var filtered = context.Customers.Include(c => c.Orders.Where(o => o.Id >
5000)).ToList();
```

ⓘ Note

In case of tracking queries, the navigation on which filtered include was applied is considered to be loaded. This means that EF Core will not attempt to re-load its values using explicit loading or lazy loading, even though some elements could still be missing.

Include on derived types

You can include related data from navigation defined only on a derived type using `Include` and `ThenInclude`.

Given the following model:

C#

```
public class SchoolContext : DbContext
{
    public DbSet<Person> People { get; set; }
    public DbSet<School> Schools { get; set; }

    protected override void OnModelCreating(ModelBuilder modelBuilder)
    {
        modelBuilder.Entity<School>().HasMany(s => s.Students).WithOne(s =>
s.School);
    }
}

public class Person
{
    public int Id { get; set; }
    public string Name { get; set; }
}

public class Student : Person
{
    public School School { get; set; }
}

public class School
{
    public int Id { get; set; }
    public string Name { get; set; }

    public List<Student> Students { get; set; }
}
```

Contents of `School` navigation of all People who are Students can be eagerly loaded using many patterns:

- Using cast

  C#

```
context.People.Include(person => ((Student)person).School).ToList()
```

- Using `as` operator

```
C#
context.People.Include(person => (person as Student).School).ToList()
```

- Using overload of `Include` that takes parameter of type `string`

```
C#
context.People.Include("School").ToList()
```

Model configuration for auto-including navigations

You can configure a navigation in the model to be included every time the entity is loaded from the database using `AutoInclude` method. It has same effect as specifying `Include` with the navigation in every query where the entity type is returned in the results. Following example shows how to configure a navigation to be automatically included.

```
C#
modelBuilder.Entity<Theme>().Navigation(e => e.ColorScheme).AutoInclude();
```

After above configuration, running a query like below will load `ColorScheme` navigation for all the themes in the results.

```
C#
using (var context = new BloggingContext())
{
    var themes = context.Themes.ToList();
}
```

This configuration is applied on every entity returned in the result no matter how it appeared in the results. That means if an entity is in the result because of use of a navigation, using `Include` over another entity type or auto-include configuration, it will load all the auto-included navigations for it. The same rule extends to the navigations configured as auto-included on derived type of the entity.

If for a particular query you don't want to load the related data through a navigation, which is configured at model level to be auto-included, you can use `IgnoreAutoIncludes` method in your query. Using this method will stop loading all the navigations configured as auto-include by the user. Running a query like below will bring back all themes from database but won't load `ColorScheme` even though it's configured as auto-included navigation.

C#

```
using (var context = new BloggingContext())
{
    var themes = context.Themes.IgnoreAutoIncludes().ToList();
}
```

Note

Navigations to owned types are also configured as auto-included by convention and using `IgnoreAutoIncludes` API doesn't stop them from being included. They will still be included in query results.