**Loading Related Data**

Bildiyimiz kimi Database-lə əlaqə qurub, datalar üzərində işləmək üçün ORM (Object Relational Mapping) istifadə olunur.

ORM-dən istifadə edərkən işləmə sürətinə görə fərqli şəkildə SELECT sorğuları həyata keçirilir.

C# daxilində, Relation-lardan asılı olaraq SELECT sorğuları bir qədər fərqlənir.

Yəni, JOIN-lər SELECT sorğusuna həm manual, həm də avtomatik olaraq əlavə oluna bilər.

**Eager Loading**

Eager Loading generate olunan SELECT sorğuya Related Data-ların parça-parça əlavə edilməsini və bu prosesin istəkli şəkildə baş verməsini təmin edən bir yoldur.

Eager Loading ilə SELECT sorğusu həyata keçirilən zaman əlavə yüklənmənin qarşısı alınır.

Əlavə yüklənmə dedikdə, Entity class'ın daxilində hər hansısa Navigation property mövcuddursa, yəni hər hansısa bir table ilə relation varsa, bu zaman Relational datalar üçün SELECT sorğusu yaradılarkən JOIN-lər sorğuda yer almayacaq.

List<TEntity> entityList = await \_context.Set<TEntity>().ToListAsync();

Yuxarıda yerləşən LINQ Method Syntax ilə yaradılan sorğunun nəticəsi aşağıdaki kimi olacaq

SELECT \* FROM Entities

Sözügedən əlavə yüklənmənin qarşısı alınması nəticəsində göründüyü kimi, heç bir JOIN prosesi query-də yer almayıb. Bu da o deməkdir ki, Navigation olaraq qeyd olunan property null olaraq qayıdacaq.

Relational olan data(lar) ehtiyac yarandığı təqdirdə, bu JOIN(lər) bizim tərəfimizdən manual olaraq əlavə edilməlidir. Include və ThenInclude method'lar məhz bunun üçündür.

Əgər LINQ Method Syntax və ya LINQ Query Syntax vasitəsilə SELECT sorğusu yaradacağıqsa, by default Eager Loading işə düşəcək.

Eager Loading əvəzinə Lazy Loading-in işə düşməsini istəyiriksə, bu zaman DbContext-lə bağlı müəyyən konfiqurasiyalara ehtiyac var.

**Include**

SELECT sorğusu yaradılan zaman Relational Data-ların da əldə olunması üçün hər bir Navigation Property nəzərə alınaraq Include method tətbiq edilir beləliklə, sorğu nəticəsində Relation Data-lar da əldə edilir. Eager Loading-i həyata keçirmək üçün istifadə olunan bir method-dur.

Aşağıda yerləşən query execute olunduqda, Navigation property null gəlmək əvəzinə, datalar yüklənmiş olaraq qayıdacaq.

List<TEntity> entityList = await \_context.Set<TEntity>().Include(e => e.RelationalProperty).ToListAsync();

və ya

List<TEntity> entityList = await \_context.Set<TEntity>().Include(relationalPropertyName).ToListAsync();

Yuxarıda yerləşən LINQ Method Syntax ilə yaradılan sorğunun nəticəsi aşağıdaki kimi olacaq

SELECT \* FROM Entities e JOIN RelationTable rt ON e.RelationalTableId = rt.Id

**Lazy Loading**

Lazy Loading, yəni tənbəl yüklənmə ilə SELECT sorğusu həyata keçirilərkən ilk olaraq relation-lar nəzərə alınmadan datalar əldə edilir.

Navigation property-lərin datasının gətirilməsi isə bir qədər fərqlidir.

Navigation property'lərə ilk müraciət olunan zaman avtomatik olaraq SQL sorğusu yaradılır və beləliklə, datalar əldə edilir. Tənbəl yüklənmə adlandırılmasının səbəbi də məhz budur.

Əgər EntityFrameworkCore istifadə edəcəyiksə və Lazy Loading-in işə düşməsini istəyiriksə, DbContext üçün müəyyən konfiqurasiyalara ehtiyac vardır.

İlk olaraq NuGet Package-dən Microsoft.EntityFrameworkCore.Proxies paketi yüklənməlidir.

Daha sonra, IoC üçün yazılan AddDbContext method daxilində UseLazyLoadingProxies() method-u aşağıdaki şəkildə call olunmalıdır.

builder.Services.AddDbContext<AppDbContext>(opt =>

{

opt.UseLazyLoadingProxies();

opt.UseSqlServer(yourConnectionString);

});

Həmçinin, bütün Entity-lər daxilində Relation-u işarəliyən bütün property-lər Virtual olaraq qeyd olunmalıdır.

Əks təqdirdə, System.InvalidOperationException ilə qarşılaşacağıq.

Exception mesaj: 'UseChangeTrackingProxies' requires all entity types to be public, unsealed, have virtual properties, and have a public or protected constructor.

public class TEntity

{

public int Id { get; set; }

public string Prop1 { get; set; }

public virtual ICollection<TEntity2> TEntities2 { get; set; }

}

public class TEntity2

{

public int Id { get; set; }

public string Prop2 { get; set; }

public int TEntityId { get; set; }

public TEntity TEntity { get; set; }

}

Navigation Property Virutal olaraq təyin edilmədiyi üçün SELECT sorğusu icra olunduqda, yəni Lazy Loading işə düşdükdə, proxy-lər düzgün işləməyəcək və beləliklə exception alacağıq.

**Manual Lazy Loading(ILazyLoader)**

Proxy-lər bütün platformalarda dəstəklənməyə bilər bu səbəbdən biz proxy-lər olmadan da Lazy Loading-i işə salmağı bacarmalıyıq.

İlk olaraq NuGet Package-dən Microsoft.EntityFrameworkCore.Abstractions paketi yüklənməlidir.

Manual olaraq həyata keçirəcəyimiz üçün Entity-lər daxilində yer alan Navigation property-lərin Virtual olaraq işarələnməsinə ehtiyac yoxdur.

Dependency Injection istifadə edilərək, ILazyLoader tipində parametr qəbul edən Constructor yazılmalı və private olan Field’a assign edilməlidir. Bu proses Lazy Loading-in baş verməsini istədiyimiz bütün Entity-lərə tətbiq edirik.

Daha sonra Navigation property üçün Entity daxilində ayrı bir property yaradılır. Navigation property-in Get və Set method-ları override olunaraq aşağıdaki şəkildə yazılır.

public class TEntity

{

private readonly ILazyLoader \_lazyLoader;

private TEntity2 \_tEntityObj;

public int Id { get; set; }

public string Prop1 { get; set; }

public TEntity2 TEntity2 {

get{ \_lazyLoader.Load(this, ref \_tEntityObj);}

set{\_tEntityObj = value;}

}

public TEntity(){}

public TEntity(ILazyLoader lazyLoader){\_lazyLoader = lazyLoader;}

}

public class TEntity2

{

public int Id { get; set; }

public string Prop2 { get; set; }

public int TEntityId { get; set; }

public TEntity TEntity { get; set; }

}

**Manual Lazy Loading(Delegate)**

Fərqi onadır ki, ILazyLoader yerinə Action<object,string> -dən istifadə olunur.

Burada object LazyLoading tətbiq olunacaq Entity-in tipini, string isə Navigation property-in adını özündə saxlayır.

Daha sonra Constructor-dan ILazyLoader yox, Action<object,string> qəbul edilir.

Lakin bu Delegate-in Load method-u olmadığı üçün biz əlavə olaraq Extension method yaradacayıq.

**N+1 Selects Problem**

Lazy Loading istifadə zamanı dataların, Navigation property-lərə müraciəti zamanı işə sorğu yaradaraq gətirdiyini artıq bilirik.Bu müraciətlər döngü içərisində hər bir entity obyekt üçün baş verdiyi zaman dəfələrlə sorğu generate olur və database-ə göndərilir. Bu proses də N+1 problemi adlanır. Çünki dəfələrlə sorğu göndərilməsi performans cəhətdən heç də yaxşı deyil. Həmçinin bu Database Connection-ın davamlı açıq qalmasına gətirib çıxarır. Eager Loading istifadəsi bu problemin qarşısını rahatlıqla alır.