LINQ-EF Core Alıştırmaları ve Ödevleri-2

Hazırlayan: Engin Niyazi Ergül (Gemini Revizyonu ile)

Giriş: Bu ödev serisi, Entity Framework Core kullanarak veritabanı işlemleri yaparken modern ve performans odaklı yaklaşımları benimsemenizi hedefler. Her ödevde **Repository Design Pattern** kullanarak kodunuzu daha modüler ve test edilebilir hale getirmeniz, **DTO (Data Transfer Object)** kullanarak sunum katmanı ile veri katmanını ayırmanız ve **sorgu optimizasyon tekniklerini** uygulamanız beklenmektedir. Bu çalışmada kendinizi olabildiğince zorlamanızı bekliyorum :)

Ödev 1: Kütüphane Yönetim Sistemi (Library Management System)

Senaryo: Basit bir kütüphane sistemi tasarlayacaksınız. Sistemde **Yazarlar (Author)** ve **Kitaplar (Book)** olacak. Bir yazarın birden fazla kitabı olabilir, ancak her kitap sadece bir yazara aittir. Bu bire-çok ilişkiyi EF Core'da modellemeniz beklenmektedir.

İstenen Bilgiler:

- **Yazar:** Id, Tam Ad (FullName), Doğum Yılı (BirthYear).
- **Kitap:** Id, Başlık (Title), Sayfa Sayısı (PageCount).

Görevler:

1. **Entity ve DbContext Oluşturma:** `Author` ve `Book` entity sınıflarını ve `LibraryContext` adında bir DbContext sınıfı hazırlayın.
2. **Repository Pattern Uygulaması:** Veritabanı işlemlerini soyutlamak için `IAuthorRepository` ve `IBookRepository` adında interfaceler ve bu interfaceleri uygulayan somut repository sınıfları oluşturun. Tüm veritabanı işlemleri bu repository'ler üzerinden yapılacaktır.
3. **Veritabanı Oluşturma:** `add-migration` ve `update-database` komutlarını kullanarak veritabanını oluşturun.
4. **Veri Ekleme:** `AuthorRepository` ve `BookRepository` sınıflarını kullanarak en az 3 yazar ve her yazara ait en az 2'şer kitap ekleyin.
5. **Veri Listeleme (Optimizasyon ve DTO Kullanımı):**
    - **DTO Oluşturma:** Sadece kitapların başlıklarını ve yazarının tam adını içeren bir `BookWithAuthorNameDto` sınıfı oluşturun.
    - **Neden DTO?** Bu yaklaşım, veritabanından sadece ihtiyaç duyulan verileri çekerek performansı artırır, gereksiz veri transferini önler ve sunum katmanını veritabanı modelinizden (entity'lerden) ayırır.
    - **Sorgulama:** `BookRepository` içinde, tüm kitapları `BookWithAuthorNameDto` tipine yansıtarak listeleyen bir metot yazın.
    - **Optimizasyon İpuçları:**
        - Sorgunuzu, veritabanından yalnızca `Title` ve yazarın `FullName` alanlarını çekecek şekilde yapılandırın.

    - Bu sorgu sadece veri okuma amaçlı olduğu için, EF Core'un nesneleri takip etmesine gerek yoktur. Sorgunuzu bu durumu göz önünde bulundurarak daha verimli hale getirin. (Bu, değişiklik takibini kapatarak performansı artırır.)
6. **Veri Güncelleme:** `BookRepository` üzerinden, belirli bir Id'ye sahip kitabın sayfa sayısı (PageCount) bilgisini güncelleyin ve değişikliği veritabanına kaydedin.

Ödev 2: Şirket Departman ve Çalışan Yönetimi (Company Department and Employee Management)

Senaryo: Bir şirketin organizasyon yapısını modelleyeceksiniz. Şirkette **Departmanlar (Department)** ve bu departmanlara bağlı **Çalışanlar (Employee)** olacak. Bir departmanda birden fazla çalışan olabilir, ancak her çalışan yalnızca bir departmana bağlıdır.

İstenen Bilgiler:

- **Departman:** Id, Ad (Name).
- **Çalışan:** Id, Tam Ad (FullName), Pozisyon (Position), Maaş (Salary).

Görevler:

1. **Entity ve DbContext Oluşturma:** Senaryoya uygun `Department` ve `Employee` entity'lerini ve `CompanyContext` adında bir DbContext'i oluşturun.
2. **Repository Pattern Uygulaması:** `IDepartmentRepository` ve `IEmployeeRepository` interfacelerini ve somut sınıflarını oluşturun.
3. **Veritabanı Oluşturma:** Migration'ları kullanarak veritabanını oluşturun.
4. **Veri Ekleme:** Repository'leri kullanarak "Software", "Accounting" ve "Human Resources" adında üç departman ve her departmana en az 2 çalışan ekleyin.
5. **Veri Sorgulama (Filtreleme ve Optimizasyon):**
    - **DTO Oluşturma:** Çalışanların listeleneceği ekran için `Id`, `FullName` ve `Position` bilgilerini içeren bir `EmployeeDto` oluşturun.
    - **Sorgulama:** `EmployeeRepository` içinde, "Software" departmanında çalışan ve maaşı (Salary) belirli bir miktardan yüksek olan çalışanların listesini `EmployeeDto` olarak döndüren bir metot yazın.
    - **Optimizasyon İpuçları:**
        - Bu sorgu, veritabanı üzerindeki yükü azaltmak için yalnızca `EmployeeDto` için gerekli olan kolonları seçmelidir (projeksiyon).
        - Sonuçlar sadece görüntüleneceği için, EF Core'un değişiklik takibi mekanizmasını kapatarak sorgu performansını iyileştirin.
6. **Veri Silme:** Bir çalışanı Id'sine göre `EmployeeRepository` üzerinden bulup işten çıkarın (veritabanından silin).

Ödev 3: Müşteri Sipariş Sistemi (Customer Order System)

Senaryo: Basit bir e-ticaret senaryosu canlandırılacak. Sistemde **Müşteriler (Customer)** ve bu müşterilerin verdiği **Siparişler (Order)** tutulacak. Bir müşterinin birden fazla siparişi olabilir, ancak her sipariş sadece bir müşteriye aittir.

İstenen Bilgiler:

- **Müşteri:** Id, Ad (FirstName), Soyad (LastName), E-posta (Email).
- **Sipariş:** Id, Sipariş Tarihi (OrderDate), Toplam Tutar (TotalAmount).

Görevler:

1. **Entity ve DbContext Oluşturma:** `Customer` ve `Order` entity sınıflarını ve `ECommerceContext` adında bir DbContext'i oluşturun.
2. **Repository Pattern Uygulaması:** `ICustomerRepository` ve `IOrderRepository` interfacelerini ve somut sınıflarını hazırlayın.
3. **Veritabanı Oluşturma:** Veritabanını migration'lar ile oluşturun.
4. **Veri Ekleme:** Repository'leri kullanarak en az 2 müşteri ve her müşteriye ait en az 3'er sipariş ekleyin.
5. **İlişkili Veri Çekme (Optimizasyonlu):**
    - **DTO Oluşturma:** Sipariş detaylarını göstermek için `OrderDate` ve `TotalAmount` içeren bir `OrderDto` sınıfı oluşturun.
    - **Sorgulama:** `CustomerRepository` içinde, belirli bir müşterinin (Id ile bulunacak) tüm siparişlerini `OrderDto` listesi olarak döndüren bir metot yazın. Siparişler, sipariş tarihine göre en yeniden en eskiye doğru sıralanmalıdır.
    - **Optimizasyon İpuçları:**
        - Sorgu, müşteriyle ilişkili siparişleri çekerken, bu sipariş verilerini doğrudan `OrderDto`'ya yansıtmalıdır. Bu, tüm `Order` entity'sinin belleğe yüklenmesini engeller.
        - Bu bir okuma işlemi olduğundan, performansı artırmak için değişiklik takibini devre dışı bırakmayı unutmayın.
6. **Veri Güncelleme:** Bir müşterinin e-posta (Email) adresini `CustomerRepository` aracılığıyla değiştirin ve veritabanını güncelleyin.

Ödev 4: Film ve Kategori Sistemi (Movie and Category System)

Senaryo: Bir film veritabanı oluşturacaksınız. Sistemde **Kategoriler (Category)** ve bu kategorilere ait **Filmler (Movie)** olacak. Bir kategoride birden çok film olabilir, ancak her film yalnızca bir kategoriye aittir.

İstenen Bilgiler:

- **Kategori:** Id, Ad (Name).
- **Film:** Id, Başlık (Title), Yönetmen (Director), Yapım Yılı (ReleaseYear).

Görevler:

1. **Entity ve DbContext Oluşturma:** `Category` ve `Movie` entity'lerini ve `MovieContext` adında bir DbContext sınıfını oluşturun.
2. **Repository Pattern Uygulaması:** `ICategoryRepository` ve `IMovieRepository` interfacelerini ve somut sınıflarını oluşturun.
3. **Veritabanı Oluşturma:** Migration'lar ile veritabanını ayağa kaldırın.
4. **Veri Ekleme:** Repository'leri kullanarak en az 3 kategori ve her kategoriye ait en az 2 film ekleyin.

5. **Veri Listeleme (DTO ve Projeksiyon):**
    - **DTO Oluşturma:** Her filmin adını, yönetmenini ve ait olduğu kategorinin adını içeren bir `MovieDetailDto` sınıfı oluşturun (`Title`, `Director`, `CategoryName`).
    - **Sorgulama:** `MovieRepository` içinde, tüm filmleri `MovieDetailDto` listesi olarak döndüren bir metot yazın.
    - **Optimizasyon İpuçları:**
        - Veritabanından sadece `MovieDetailDto` için gereken alanları (`Movie.Title`, `Movie.Director`, `Category.Name`) seçmek üzere bir projeksiyon sorgusu oluşturun.
        - Bu listeleme sorgusunda, EF Core'un değişiklik takibi özelliğini kapatarak bellek kullanımını ve işlem süresini azaltın.
6. **Veri Silme:** `CategoryRepository`'de, adı "Comedy" olan kategoriyi ve bu kategoriye bağlı tüm filmleri silen bir metot yazın. (Bu, ilişkili verilerin silinmesi senaryosunu ele almanızı gerektirir.)

Ödev 5: Blog Sistemi (Blog System with Posts, Categories, and Comments)

Senaryo: Bu ödevde daha katmanlı bir ilişki yapısı kuracaksınız: **Kategoriler (Category)**, **Blog Yazıları (Post)** ve **Yorumlar (Comment)**.

- Bir `Category` birden fazla `Post` içerebilir. Her `Post` sadece bir `Category`'ye aittir.
- Bir `Post` birden fazla `Comment` alabilir. Her `Comment` sadece bir `Post`'a aittir.

İstenen Bilgiler:

- **Kategori:** Id, Ad (Name).
- **Post:** Id, Başlık (Title), İçerik (Content), Yayın Tarihi (PublishedDate).
- **Yorum:** Id, Yorum Yapan (AuthorName), Mesaj (Message).

Görevler:

1. **Entity ve DbContext Oluşturma:** Bu üç katmanlı ilişkiyi modelleyecek `Category`, `Post`, `Comment` entity'lerini ve `BlogContext`'i oluşturun.
2. **Repository Pattern Uygulaması:** `ICategoryRepository`, `IPostRepository` ve `ICommentRepository` için interfaceler ve sınıflar oluşturun.
3. **Veritabanı Oluşturma:** Migration'ları kullanarak veritabanını oluşturun.
4. **Veri Ekleme:** Birkaç kategori, bu kategorilere ait postlar ve bu postlara ait yorumları repository'ler aracılığıyla ekleyin.
5. **İleri Düzey İlişkili Veri Çekme (Hiyerarşik DTO):**
    - **DTO Oluşturma:** İç içe bir yapı kurun. Yorumları listelemek için `CommentDto`, bir postu ve yorumlarını listelemek için `PostWithCommentsDto` (içinde `List<CommentDto>` barındıran) ve son olarak bir kategoriyi ve postlarını göstermek için `CategoryWithPostsDto` (içinde `List<PostWithCommentsDto>` barındıran) sınıfları oluşturun.
    - **Neden Hiyerarşik DTO?** Bu yapı, hem veritabanı modelinizi dışarıya sızdırmamanızı sağlar hem de tam olarak hangi verinin gerekli olduğunu belirterek karmaşık sorguları optimize eder.

    - **Sorgulama:** `CategoryRepository` içinde, belirli bir kategorideki (Id ile bulunacak) tüm postları ve bu postlara yapılmış yorumları içeren `CategoryWithPostsDto` nesnesini döndüren bir metot yazın.
    - **Optimizasyon İpuçları:**
        - Bu karmaşık sorguyu tek bir veritabanı çağrısında çözmek için ilişkili tabloları (`Post` ve `Comment`) sorguya dahil etmelisiniz.
        - Sorgunun sonunda, veritabanından gelen sonuçları doğrudan hiyerarşik DTO yapınıza yansıtın (projeksiyon).
        - Bu derin ve karmaşık okuma operasyonunda, performansı ciddi şekilde etkileyeceği için değişiklik takibini mutlaka devre dışı bırakın.
6. **Veri Güncelleme:** `CommentRepository`'yi kullanarak, belirli bir yoruma ait mesaj (Message) içeriğini güncelleyin.