Her çözüm şu yapıları göreceksiniz:

- **Entities:** Veritabanı tablolarını temsil eden sınıflar.
- **Context:** Veritabanı bağlantısını ve konfigürasyonunu yöneten DbContext sınıfı.
- **DTOs:** Sunum katmanına özel veri taşıma nesneleri.
- **Repositories (Interfaces & Implementations):** Veri erişim mantığını soyutlayan arayüzler ve sınıflar.
- **Program.cs:** Uygulamanın nasıl çalıştığını gösteren, tüm mantığı bir araya getiren ana program.

Ödev 1: Kütüphane Yönetim Sistemi (Library Management System)

Bu bölümde yazar ve kitaplar arasındaki bire-çok ilişkiyi modelleyip, DTO kullanarak optimize edilmiş bir listeleme yapacağız.

1. Entity Sınıfları (Entities.cs)

```csharp
// Entities.cs
public class Author
{
    public int Id { get; set; }
    public string FullName { get; set; }
    public int BirthYear { get; set; }
    public ICollection<Book> Books { get; set; } = new List<Book>();
}

public class Book
{
    public int Id { get; set; }
    public string Title { get; set; }
    public int PageCount { get; set; }
    public int AuthorId { get; set; }
    public Author Author { get; set; }
}```

#### 2. DbContext Sınıfı (LibraryContext.cs)
```csharp
// LibraryContext.cs
using Microsoft.EntityFrameworkCore;

public class LibraryContext : DbContext
{
    public DbSet<Author> Authors { get; set; }
    public DbSet<Book> Books { get; set; }

    protected override void OnConfiguring(DbContextOptionsBuilder
optionsBuilder)
    {
        // Projenize uygun bir connection string kullanın.
```
```

```
        // Örnek olarak SQLite kullanılmıştır.
        optionsBuilder.UseSqlite("Data Source=library.db");
    }
}
```

3. DTO Sınıfı (Dtos.cs)

```
// Dtos.cs
public class BookWithAuthorNameDto
{
    public string Title { get; set; }
    public string AuthorFullName { get; set; }
}
```

4. Repository Arayüzleri (IRepositories.cs)

```
// IRepositories.cs
public interface IAuthorRepository
{
    void Add(Author author);
    void SaveChanges();
}

public interface IBookRepository
{
    void Add(Book book);
    void Update(Book book);
    List<BookWithAuthorNameDto> GetAllBooksWithAuthorNames();
    Book GetById(int id);
    void SaveChanges();
}
```

5. Repository Sınıfları (Repositories.cs)

```
// Repositories.cs
using Microsoft.EntityFrameworkCore;

public class AuthorRepository : IAuthorRepository
{
    private readonly LibraryContext _context;
    public AuthorRepository(LibraryContext context) { _context = context; }

    public void Add(Author author) => _context.Authors.Add(author);
    public void SaveChanges() => _context.SaveChanges();
}

public class BookRepository : IBookRepository
{
    private readonly LibraryContext _context;
```

```
    public BookRepository(LibraryContext context) { _context = context; }

    public void Add(Book book) => _context.Books.Add(book);
    public Book GetById(int id) => _context.Books.Find(id);
    public void Update(Book book) => _context.Books.Update(book);
    public void SaveChanges() => _context.SaveChanges();

    public List<BookWithAuthorNameDto> GetAllBooksWithAuthorNames()
    {
        // Optimizasyon:
        // 1. Select ile sadece gerekli alanları (projeksiyon) çekiyoruz.
        // 2. Bu, veritabanından gereksiz veri çekilmesini önler.
        // 3. AsNoTracking() ile değişiklik takibini kapatarak performansı
artırıyoruz.
        return _context.Books
            .AsNoTracking() // Okuma işlemlerinde performansı artırır.
            .Include(b => b.Author) // Yazar bilgisine erişmek için.
            .Select(b => new BookWithAuthorNameDto
            {
                Title = b.Title,
                AuthorFullName = b.Author.FullName
            })
            .ToList();
    }
}
```

6. Kullanım Örneği (Program.cs)

```
// Program.cs
using (var context = new LibraryContext())
{
    // Veritabanını oluştur (migration sonrası)
    context.Database.EnsureCreated();

    IAuthorRepository authorRepo = new AuthorRepository(context);
    IBookRepository bookRepo = new BookRepository(context);

    // --- 4. Veri Ekleme ---
    if (!context.Authors.Any())
    {
        var author1 = new Author { FullName = "Yaşar Kemal", BirthYear = 1923 };
        var author2 = new Author { FullName = "Orhan Pamuk", BirthYear = 1952 };
        var author3 = new Author { FullName = "Elif Şafak", BirthYear = 1971 };

        authorRepo.Add(author1);
        authorRepo.Add(author2);
        authorRepo.Add(author3);
        authorRepo.SaveChanges();

        bookRepo.Add(new Book { Title = "İnce Memed", PageCount = 436, AuthorId
= author1.Id });
```

```
        bookRepo.Add(new Book { Title = "Yer Demir Gök Bakır", PageCount = 290,
AuthorId = author1.Id });
        bookRepo.Add(new Book { Title = "Masumiyet Müzesi", PageCount = 535,
AuthorId = author2.Id });
        bookRepo.Add(new Book { Title = "Kar", PageCount = 448, AuthorId =
author2.Id });
        bookRepo.Add(new Book { Title = "Aşk", PageCount = 420, AuthorId =
author3.Id });
        bookRepo.Add(new Book { Title = "Havva'nın Üç Kızı", PageCount = 415,
AuthorId = author3.Id });
        bookRepo.SaveChanges();
        Console.WriteLine("Veriler eklendi.");
    }

    // --- 5. Veri Listeleme (Optimizasyon ve DTO Kullanımı) ---
    Console.WriteLine("\n--- Kitaplar ve Yazarları ---");
    var booksDto = bookRepo.GetAllBooksWithAuthorNames();
    foreach (var book in booksDto)
    {
        Console.WriteLine($"Başlık: {book.Title}, Yazar:
{book.AuthorFullName}");
    }

    // --- 6. Veri Güncelleme ---
    Console.WriteLine("\n--- Kitap Sayfa Sayısı Güncelleme ---");
    var bookToUpdate = bookRepo.GetById(1); // ID'si 1 olan kitabı bul.
    if (bookToUpdate != null)
    {
        Console.WriteLine($"'{bookToUpdate.Title}' kitabının eski sayfa sayısı:
{bookToUpdate.PageCount}");
        bookToUpdate.PageCount = 450;
        bookRepo.Update(bookToUpdate);
        bookRepo.SaveChanges();
        Console.WriteLine($"'{bookToUpdate.Title}' kitabının yeni sayfa sayısı:
{bookToUpdate.PageCount}");
    }
}
```

Ödev 2: Şirket Departman ve Çalışan Yönetimi

Bu ödevde, departman ve çalışanlar arasındaki ilişkiyi kurup, belirli filtrelere göre DTO ile veri çekeceğiz ve silme işlemi yapacağız.

1. Entity Sınıfları

```
public class Department
{
    public int Id { get; set; }
    public string Name { get; set; }
    public ICollection<Employee> Employees { get; set; } = new List<Employee>();
```

```
}

public class Employee
{
    public int Id { get; set; }
    public string FullName { get; set; }
    public string Position { get; set; }
    public decimal Salary { get; set; }
    public int DepartmentId { get; set; }
    public Department Department { get; set; }
}
```

2. DbContext Sınıfı

```
public class CompanyContext : DbContext
{
    public DbSet<Department> Departments { get; set; }
    public DbSet<Employee> Employees { get; set; }

    protected override void OnConfiguring(DbContextOptionsBuilder
optionsBuilder)
    {
        optionsBuilder.UseSqlite("Data Source=company.db");
    }
}
```

3. DTO Sınıfı

```
public class EmployeeDto
{
    public int Id { get; set; }
    public string FullName { get; set; }
    public string Position { get; set; }
}
```

4. Repository Arayüzleri ve Sınıfları

```
// --- Arayüzler ---
public interface IDepartmentRepository { void Add(Department department); }
public interface IEmployeeRepository
{
    void Add(Employee employee);
    void Delete(int id);
    List<EmployeeDto> GetSoftwareEmployeesWithHighSalary(decimal minSalary);
    void SaveChanges();
}

// --- Sınıflar ---
public class DepartmentRepository : IDepartmentRepository
```

```
{
    private readonly CompanyContext _context;
    public DepartmentRepository(CompanyContext context) { _context = context; }
    public void Add(Department department) =>
_context.Departments.Add(department);
}

public class EmployeeRepository : IEmployeeRepository
{
    private readonly CompanyContext _context;
    public EmployeeRepository(CompanyContext context) { _context = context; }

    public void Add(Employee employee) => _context.Employees.Add(employee);

    public void Delete(int id)
    {
        var employee = _context.Employees.Find(id);
        if (employee != null)
        {
            _context.Employees.Remove(employee);
        }
    }

    public void SaveChanges() => _context.SaveChanges();

    public List<EmployeeDto> GetSoftwareEmployeesWithHighSalary(decimal
minSalary)
    {
        // Optimizasyon:
        // 1. Where ile veritabanı seviyesinde filtreleme yapılıyor.
        // 2. Select ile DTO projeksiyonu uygulanıyor.
        // 3. AsNoTracking() ile okuma performansı artırılıyor.
        return _context.Employees
            .AsNoTracking()
            .Where(e => e.Department.Name == "Software" && e.Salary > minSalary)
            .Select(e => new EmployeeDto
            {
                Id = e.Id,
                FullName = e.FullName,
                Position = e.Position
            })
            .ToList();
    }
}
```

5. Kullanım Örneği (Program.cs)

```
using (var context = new CompanyContext())
{
    context.Database.EnsureCreated();
    var departmentRepo = new DepartmentRepository(context);
    var employeeRepo = new EmployeeRepository(context);
```

```
    // --- 4. Veri Ekleme ---
    if (!context.Departments.Any())
    {
        var software = new Department { Name = "Software" };
        var accounting = new Department { Name = "Accounting" };
        var hr = new Department { Name = "Human Resources" };
        departmentRepo.Add(software); departmentRepo.Add(accounting);
departmentRepo.Add(hr);
        context.SaveChanges();

        employeeRepo.Add(new Employee { FullName = "Ali Veli", Position =
"Senior Developer", Salary = 30000, DepartmentId = software.Id });
        employeeRepo.Add(new Employee { FullName = "Ayşe Yılmaz", Position =
"Junior Developer", Salary = 18000, DepartmentId = software.Id });
        employeeRepo.Add(new Employee { FullName = "Fatma Kaya", Position =
"Accountant", Salary = 22000, DepartmentId = accounting.Id });
        employeeRepo.Add(new Employee { FullName = "Mehmet Demir", Position =
"HR Specialist", Salary = 21000, DepartmentId = hr.Id });
        employeeRepo.SaveChanges();
        Console.WriteLine("Departman ve çalışan verileri eklendi.");
    }

    // --- 5. Veri Sorgulama ---
    Console.WriteLine("\n--- Maaşı 20000'den yüksek olan yazılımcılar ---");
    var highSalarySoftwareDevs =
employeeRepo.GetSoftwareEmployeesWithHighSalary(20000);
    foreach (var dev in highSalarySoftwareDevs)
    {
        Console.WriteLine($"ID: {dev.Id}, Ad: {dev.FullName}, Pozisyon:
{dev.Position}");
    }

    // --- 6. Veri Silme ---
    Console.WriteLine("\n--- ID'si 2 olan çalışan işten çıkarılıyor ---");
    employeeRepo.Delete(2); // Ayşe Yılmaz'ı sil
    employeeRepo.SaveChanges();
    Console.WriteLine("Çalışan silindi.");
}
```

Ödev 3: Müşteri Sipariş Sistemi

Bu ödevde, bir müşterinin siparişlerini optimize bir şekilde listeleyip, müşteri bilgilerini güncelleyeceğiz.

1. Entity ve DbContext

```
// --- Entities ---
public class Customer
{
    public int Id { get; set; }
```

```
    public string FirstName { get; set; }
    public string LastName { get; set; }
    public string Email { get; set; }
    public ICollection<Order> Orders { get; set; } = new List<Order>();
}

public class Order
{
    public int Id { get; set; }
    public DateTime OrderDate { get; set; }
    public decimal TotalAmount { get; set; }
    public int CustomerId { get; set; }
    public Customer Customer { get; set; }
}

// --- DbContext ---
public class ECommerceContext : DbContext
{
    public DbSet<Customer> Customers { get; set; }
    public DbSet<Order> Orders { get; set; }
    protected override void OnConfiguring(DbContextOptionsBuilder
optionsBuilder) => optionsBuilder.UseSqlite("Data Source=ecommerce.db");
}
```

2. DTO Sınıfı

```
public class OrderDto
{
    public DateTime OrderDate { get; set; }
    public decimal TotalAmount { get; set; }
}
```

3. Repository Arayüzleri ve Sınıfları```csharp

// --- Arayüzler --- public interface ICustomerRepository { void Add(Customer customer); void Update(Customer customer); Customer GetById(int id); List GetOrdersByCustomerId(int customerId); void SaveChanges(); }

public interface IOrderRepository { void Add(Order order); }

// --- Sınıflar --- public class CustomerRepository : ICustomerRepository { private readonly ECommerceContext _context; public CustomerRepository(ECommerceContext context) { _context = context; }

```
public void Add(Customer customer) => _context.Customers.Add(customer);
public Customer GetById(int id) => _context.Customers.Find(id);
public void Update(Customer customer) => _context.Customers.Update(customer);
public void SaveChanges() => _context.SaveChanges();

public List<OrderDto> GetOrdersByCustomerId(int customerId)
{
```

```
    // Optimizasyon:
    // 1. Müşteriyle ilişkili siparişler `Where` ile filtreleniyor.
    // 2. `OrderByDescending` ile veritabanında sıralama yapılıyor.
    // 3. `Select` ile DTO'ya yansıtılıyor.
    // 4. `AsNoTracking` ile okuma performansı artırılıyor.
    return _context.Orders
        .AsNoTracking()
        .Where(o => o.CustomerId == customerId)
        .OrderByDescending(o => o.OrderDate)
        .Select(o => new OrderDto
        {
            OrderDate = o.OrderDate,
            TotalAmount = o.TotalAmount
        })
        .ToList();
}
```

} public class OrderRepository : IOrderRepository { private readonly ECommerceContext _context; public OrderRepository(ECommerceContext context) { _context = context; } public void Add(Order order) => _context.Orders.Add(order); }

4. Kullanım Örneği (Program.cs)

```csharp
using (var context = new ECommerceContext())
{
    context.Database.EnsureCreated();
    var customerRepo = new CustomerRepository(context);
    var orderRepo = new OrderRepository(context);

    // --- 4. Veri Ekleme ---
    if (!context.Customers.Any())
    {
        var customer1 = new Customer { FirstName = "Zeynep", LastName = "Güneş", Email = "zeynep@example.com" };
        var customer2 = new Customer { FirstName = "Ahmet", LastName = "Yıldız", Email = "ahmet@example.com" };
        customerRepo.Add(customer1);
        customerRepo.Add(customer2);
        context.SaveChanges();

        orderRepo.Add(new Order { OrderDate = DateTime.Now.AddDays(-10), TotalAmount = 150.50m, CustomerId = customer1.Id });
        orderRepo.Add(new Order { OrderDate = DateTime.Now.AddDays(-5), TotalAmount = 250.00m, CustomerId = customer1.Id });
        orderRepo.Add(new Order { OrderDate = DateTime.Now.AddDays(-2), TotalAmount = 99.90m, CustomerId = customer1.Id });
        orderRepo.Add(new Order { OrderDate = DateTime.Now.AddDays(-20), TotalAmount = 500.00m, CustomerId = customer2.Id });
        orderRepo.Add(new Order { OrderDate = DateTime.Now.AddDays(-1), TotalAmount = 75.25m, CustomerId = customer2.Id });
        context.SaveChanges();
        Console.WriteLine("Müşteri ve sipariş verileri eklendi.");
```

```
    }

    // --- 5. İlişkili Veri Çekme ---
    Console.WriteLine("\n--- ID'si 1 olan müşterinin siparişleri (yeniden eskiye) ---");
    var orders = customerRepo.GetOrdersByCustomerId(1);
    foreach (var order in orders)
    {
        Console.WriteLine($"Tarih: {order.OrderDate:dd.MM.yyyy}, Tutar: {order.TotalAmount:C}");
    }

    // --- 6. Veri Güncelleme ---
    Console.WriteLine("\n--- Müşteri e-posta güncelleme ---");
    var customerToUpdate = customerRepo.GetById(2);
    if(customerToUpdate != null)
    {
        Console.WriteLine($"Eski E-posta: {customerToUpdate.Email}");
        customerToUpdate.Email = "ahmet.yildiz@newdomain.com";
        customerRepo.Update(customerToUpdate);
        customerRepo.SaveChanges();
        Console.WriteLine($"Yeni E-posta: {customerToUpdate.Email}");
    }
}
```

Ödev 4: Film ve Kategori Sistemi

Bu ödevde, ilişkili iki tablodan veri çekip DTO'ya yansıtacak ve bir kategoriyi ilişkili filmleriyle birlikte sileceğiz.

1. Entity ve DbContext

```
// --- Entities ---
public class Category
{
    public int Id { get; set; }
    public string Name { get; set; }
    public ICollection<Movie> Movies { get; set; } = new List<Movie>();
}

public class Movie
{
    public int Id { get; set; }
    public string Title { get; set; }
    public string Director { get; set; }
    public int ReleaseYear { get; set; }
    public int CategoryId { get; set; }
    public Category Category { get; set; }
}

// --- DbContext ---
```

```
public class MovieContext : DbContext
{
    public DbSet<Category> Categories { get; set; }
    public DbSet<Movie> Movies { get; set; }
    protected override void OnConfiguring(DbContextOptionsBuilder
optionsBuilder) => optionsBuilder.UseSqlite("Data Source=movies.db");
}
```

2. DTO Sınıfı

```
public class MovieDetailDto
{
    public string Title { get; set; }
    public string Director { get; set; }
    public string CategoryName { get; set; }
}
```

3. Repository Arayüzleri ve Sınıfları

```
// --- Arayüzler ---
public interface ICategoryRepository
{
    void Add(Category category);
    void DeleteCategoryAndMovies(string categoryName);
    void SaveChanges();
}
public interface IMovieRepository
{
    void Add(Movie movie);
    List<MovieDetailDto> GetAllMovieDetails();
}

// --- Sınıflar ---
public class CategoryRepository : ICategoryRepository
{
    private readonly MovieContext _context;
    public CategoryRepository(MovieContext context) { _context = context; }

    public void Add(Category category) => _context.Categories.Add(category);
    public void SaveChanges() => _context.SaveChanges();

    public void DeleteCategoryAndMovies(string categoryName)
    {
        // Önce silinecek kategoriyi ve ilişkili filmleri bul.
        // Include, ilişkili filmlerin de yüklenmesini sağlar.
        var category = _context.Categories.Include(c => c.Movies)
                                          .FirstOrDefault(c => c.Name ==
categoryName);
        if (category != null)
        {
            // Önce ilişkili filmleri silmek daha güvenlidir,
```

```
            // ancak EF Core'da Cascade Delete ayarı varsa sadece kategoriyi
silmek yeterlidir.
            // Burada manuel olarak siliyoruz.
            _context.Movies.RemoveRange(category.Movies);
            _context.Categories.Remove(category);
        }
    }
}

public class MovieRepository : IMovieRepository
{
    private readonly MovieContext _context;
    public MovieRepository(MovieContext context) { _context = context; }

    public void Add(Movie movie) => _context.Movies.Add(movie);

    public List<MovieDetailDto> GetAllMovieDetails()
    {
        // Optimizasyon:
        // 1. Kategori adına erişmek için `Include` kullanılır.
        // 2. `Select` ile veriler doğrudan DTO'ya yansıtılır.
        // 3. `AsNoTracking` ile okuma performansı artırılır.
        return _context.Movies
            .AsNoTracking()
            .Include(m => m.Category)
            .Select(m => new MovieDetailDto
            {
                Title = m.Title,
                Director = m.Director,
                CategoryName = m.Category.Name
            })
            .ToList();
    }
}
```

4. Kullanım Örneği (Program.cs)

```
using (var context = new MovieContext())
{
    context.Database.EnsureCreated();
    var categoryRepo = new CategoryRepository(context);
    var movieRepo = new MovieRepository(context);

    // --- 4. Veri Ekleme ---
    if (!context.Categories.Any())
    {
        var drama = new Category { Name = "Drama" };
        var comedy = new Category { Name = "Comedy" };
        var scifi = new Category { Name = "Sci-Fi" };
        categoryRepo.Add(drama); categoryRepo.Add(comedy);
categoryRepo.Add(scifi);
        context.SaveChanges();
```

```
        movieRepo.Add(new Movie { Title = "The Godfather", Director = "F.F. Coppola", ReleaseYear = 1972, CategoryId = drama.Id });
        movieRepo.Add(new Movie { Title = "Babam ve Oğlum", Director = "Çağan Irmak", ReleaseYear = 2005, CategoryId = drama.Id });
        movieRepo.Add(new Movie { Title = "G.O.R.A", Director = "Cem Yılmaz", ReleaseYear = 2004, CategoryId = comedy.Id });
        movieRepo.Add(new Movie { Title = "The Matrix", Director = "Wachowskis", ReleaseYear = 1999, CategoryId = scifi.Id });
        context.SaveChanges();
        Console.WriteLine("Kategori ve film verileri eklendi.");
    }

    // --- 5. Veri Listeleme ---
    Console.WriteLine("\n--- Tüm Filmlerin Detayları ---");
    var movies = movieRepo.GetAllMovieDetails();
    foreach (var movie in movies)
    {
        Console.WriteLine($"Film: {movie.Title}, Yönetmen: {movie.Director}, Kategori: {movie.CategoryName}");
    }

    // --- 6. Veri Silme ---
    Console.WriteLine("\n--- 'Comedy' kategorisi ve filmleri siliniyor ---");
    categoryRepo.DeleteCategoryAndMovies("Comedy");
    categoryRepo.SaveChanges();
    Console.WriteLine("Silme işlemi tamamlandı.");

    Console.WriteLine("\n--- Silme Sonrası Tüm Filmler ---");
    var remainingMovies = movieRepo.GetAllMovieDetails();
    foreach (var movie in remainingMovies)
    {
        Console.WriteLine($"Film: {movie.Title}, Kategori: {movie.CategoryName}");
    }
}
```

Ödev 5: Blog Sistemi (İleri Düzey)

Bu son ödevde, üç katmanlı bir ilişkiyi (Kategori -> Yazı -> Yorum) tek bir sorguda, hiyerarşik DTO'lar kullanarak ve performanstan ödün vermeden çekeceğiz.

1. Entity ve DbContext

```
// --- Entities ---
public class Category {
    public int Id { get; set; }
    public string Name { get; set; }
    public ICollection<Post> Posts { get; set; } = new List<Post>();
}
```

```
public class Post {
    public int Id { get; set; }
    public string Title { get; set; }
    public string Content { get; set; }
    public DateTime PublishedDate { get; set; }
    public int CategoryId { get; set; }
    public Category Category { get; set; }
    public ICollection<Comment> Comments { get; set; } = new List<Comment>();
}
public class Comment {
    public int Id { get; set; }
    public string AuthorName { get; set; }
    public string Message { get; set; }
    public int PostId { get; set; }
    public Post Post { get; set; }
}

// --- DbContext ---
public class BlogContext : DbContext
{
    public DbSet<Category> Categories { get; set; }
    public DbSet<Post> Posts { get; set; }
    public DbSet<Comment> Comments { get; set; }
    protected override void OnConfiguring(DbContextOptionsBuilder
optionsBuilder) => optionsBuilder.UseSqlite("Data Source=blog.db");
}
```

2. Hiyerarşik DTO Sınıfları

```
// --- DTOs ---
public class CommentDto {
    public string AuthorName { get; set; }
    public string Message { get; set; }
}

public class PostWithCommentsDto {
    public string Title { get; set; }
    public DateTime PublishedDate { get; set; }
    public List<CommentDto> Comments { get; set; }
}

public class CategoryWithPostsDto {
    public string CategoryName { get; set; }
    public List<PostWithCommentsDto> Posts { get; set; }
}
```

3. Repository Arayüzleri ve Sınıfları

```
// --- Arayüzler ---
public interface ICategoryRepository {
    void Add(Category category);
```

```
    CategoryWithPostsDto GetCategoryWithPostsAndComments(int categoryId);
}
public interface IPostRepository { void Add(Post post); }
public interface ICommentRepository {
    void Add(Comment comment);
    void Update(Comment comment);
    Comment GetById(int id);
    void SaveChanges();
}

// --- Sınıflar ---
public class CategoryRepository : ICategoryRepository
{
    private readonly BlogContext _context;
    public CategoryRepository(BlogContext context) { _context = context; }
    public void Add(Category category) => _context.Categories.Add(category);

    public CategoryWithPostsDto GetCategoryWithPostsAndComments(int categoryId)
    {
        // Optimizasyon:
        // 1. `Include` ve `ThenInclude` ile ilişkili tüm verileri (Posts ve
Comments) tek bir sorguda yüklüyoruz.
        //    Bu, "N+1" problemini önler.
        // 2. `Where` ile ana kategoriyi veritabanında filtreliyoruz.
        // 3. `Select` ile tüm bu veriyi hiyerarşik DTO yapımıza yansıtıyoruz
(projeksiyon).
        // 4. `AsNoTracking` bu derin okuma operasyonunda performansı ciddi
şekilde artırır.
        return _context.Categories
            .AsNoTracking()
            .Where(c => c.Id == categoryId)
            .Include(c => c.Posts)
                .ThenInclude(p => p.Comments)
            .Select(c => new CategoryWithPostsDto
            {
                CategoryName = c.Name,
                Posts = c.Posts.Select(p => new PostWithCommentsDto
                {
                    Title = p.Title,
                    PublishedDate = p.PublishedDate,
                    Comments = p.Comments.Select(co => new CommentDto
                    {
                        AuthorName = co.AuthorName,
                        Message = co.Message
                    }).ToList()
                }).ToList()
            })
            .FirstOrDefault();
    }
}

public class PostRepository : IPostRepository
{
```

```
    private readonly BlogContext _context;
    public PostRepository(BlogContext context) { _context = context; }
    public void Add(Post post) => _context.Posts.Add(post);
}

public class CommentRepository : ICommentRepository
{
    private readonly BlogContext _context;
    public CommentRepository(BlogContext context) { _context = context; }
    public void Add(Comment comment) => _context.Comments.Add(comment);
    public Comment GetById(int id) => _context.Comments.Find(id);
    public void Update(Comment comment) => _context.Comments.Update(comment);
    public void SaveChanges() => _context.SaveChanges();
}
```

4. Kullanım Örneği (Program.cs)

```
using (var context = new BlogContext())
{
    context.Database.EnsureCreated();
    var catRepo = new CategoryRepository(context);
    var postRepo = new PostRepository(context);
    var commentRepo = new CommentRepository(context);

    // --- 4. Veri Ekleme ---
    if (!context.Categories.Any())
    {
        var tech = new Category { Name = "Teknoloji" };
        var travel = new Category { Name = "Gezi" };
        catRepo.Add(tech); catRepo.Add(travel);
        context.SaveChanges();

        var post1 = new Post { Title = "EF Core Optimizasyonu", Content = "...",
PublishedDate = DateTime.Now.AddDays(-5), CategoryId = tech.Id };
        var post2 = new Post { Title = "Ege'de Bir Hafta", Content = "...",
PublishedDate = DateTime.Now.AddDays(-10), CategoryId = travel.Id };
        postRepo.Add(post1); postRepo.Add(post2);
        context.SaveChanges();

        commentRepo.Add(new Comment { AuthorName = "Ali", Message = "Harika bir
yazı!", PostId = post1.Id });
        commentRepo.Add(new Comment { AuthorName = "Zeynep", Message = "Çok
bilgilendirici.", PostId = post1.Id });
        commentRepo.Add(new Comment { AuthorName = "Murat", Message = "Ben de
gitmek istiyorum!", PostId = post2.Id });
        commentRepo.SaveChanges();
        Console.WriteLine("Blog verileri eklendi.");
    }

    // --- 5. İleri Düzey İlişkili Veri Çekme ---
    Console.WriteLine("\n--- 'Teknoloji' Kategorisindeki Yazılar ve Yorumları
---");
```

```
    var techCategoryDetails = catRepo.GetCategoryWithPostsAndComments(1); //
ID'si 1 olan Teknoloji kategorisi
    if(techCategoryDetails != null)
    {
        Console.WriteLine($"Kategori: {techCategoryDetails.CategoryName}");
        foreach (var post in techCategoryDetails.Posts)
        {
            Console.WriteLine($"  - Yazı: {post.Title}
({post.PublishedDate:dd.MM.yyyy})");
            foreach (var comment in post.Comments)
            {
                Console.WriteLine($"    - Yorum: '{comment.Message}' (Yazan:
{comment.AuthorName})");
            }
        }
    }

    // --- 6. Veri Güncelleme ---
    Console.WriteLine("\n--- Yorum güncelleme ---");
    var commentToUpdate = commentRepo.GetById(1);
    if(commentToUpdate != null)
    {
        Console.WriteLine($"Eski Yorum: {commentToUpdate.Message}");
        commentToUpdate.Message = "Harika bir yazı! Çok teşekkürler.";
        commentRepo.Update(commentToUpdate);
        commentRepo.SaveChanges();
        Console.WriteLine($"Yeni Yorum: {commentToUpdate.Message}");
    }
}
```