**МИНОБРНАУКИ РОССИИ**

**САНКТ-ПЕТЕРБУРГСКИЙ ГОСУДАРСТВЕННЫЙ**

**ЭЛЕКТРОТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ**

**«ЛЭТИ» ИМ. В.И. УЛЬЯНОВА (ЛЕНИНА)**

**Кафедра Вычислительной Техники**

**ОТЧЕТ**

**по лабораторной работе №3**

**по дисциплине «Программирование в среде .NET»**

**ТЕМА: РАЗРАБОТКА СЛОЯ ДОСТУПА К ДАННЫМ ПРИЛОЖЕНИЯ**

|  |  |  |
| --- | --- | --- |
| Студентка гр. 6307 |  | Пякшина О.С. |
| Преподаватель |  | Пешехонов К.А. |

Санкт-Петербург

2020

**Цель работы**

Реализовать слой доступа к данным приложения.

**Задание**

Написать слой доступа к данным: Entity Framework Code First + MS SQL Server Developer Edition.

**Код программы**

Примеры кода программы даны для сущности «заказ», другие сущности («покупатель» и «книга») реализованы похожим образом.

1) Контекст для работы с базой данных

public partial class BookStoreContext : DbContext

{

public BookStoreContext()

{ }

public BookStoreContext(DbContextOptions<BookStoreContext> options) : base(options)

{ }

public virtual DbSet<Book> Book { get; set; }

public virtual DbSet<Client> Client { get; set; }

public virtual DbSet<Order> Order { get; set; }

protected override void OnConfiguring(DbContextOptionsBuilder optionsBuilder)

{ }

protected override void OnModelCreating(ModelBuilder modelBuilder)

{

modelBuilder.Entity<Book>(entity =>

{

entity.Property(c => c.Id).UseIdentityColumn().Metadata.SetBeforeSaveBehavior(PropertySaveBehavior.Ignore);

entity.Property(c => c.Name).IsRequired();

entity.Property(c => c.Author).IsRequired();

entity.Property(c => c.Price).IsRequired();

entity.Property(c => c.Year).IsRequired();

});

modelBuilder.Entity<Client>(entity =>

{

entity.Property(m => m.Id).UseIdentityColumn().Metadata.SetBeforeSaveBehavior(PropertySaveBehavior.Ignore);

entity.Property(m => m.FirstName).IsRequired();

entity.Property(m => m.LastName).IsRequired();

entity.Property(m => m.MiddleName).IsRequired();

});

modelBuilder.Entity<Order>(entity =>

{

entity.Property(s => s.Id).UseIdentityColumn().Metadata.SetBeforeSaveBehavior(PropertySaveBehavior.Ignore);

entity.Property(s => s.Date).IsRequired();

entity.Property(s => s.Time).IsRequired();

entity.HasOne(s => s.Book)

.WithMany(c => c.Order)

.HasForeignKey(s => s.BookId)

.HasConstraintName("Order\_Book");

entity.HasOne(s => s.Client)

.WithMany(m => m.Order).HasForeignKey(s => s.ClientId)

.HasConstraintName("Order\_Client");

});

this.OnModelCreatingPartial(modelBuilder);

}

partial void OnModelCreatingPartial(ModelBuilder modelBuilder);

}

2) Пример сущности (Order)

public partial class Order

{

[Key, DatabaseGenerated(DatabaseGeneratedOption.Identity)]

//идентификатор

public int Id { get; set; }

//Время покупки

public string Time { get; set; }

//Дата покупки

public string Date { get; set; }

public int? ClientId { get; set; }

public int? BookId { get; set; }

//Информация о книге

public virtual Book Book { get; set; }

//Информация о покупателе

public virtual Client Client { get; set; }

}

3) Пример интерфейса с декларацией методов для доступа к данным (IOrderDataAccess)

public interface IOrderDataAccess

{

Task<Order> InsertAsync(OrderUpdateModel order);

Task<IEnumerable<Order>> GetAsync();

Task<Order> GetAsync(IOrderIdentity orderId);

Task<Order> UpdateAsync(OrderUpdateModel order);

Task<Order> GetByAsync(IOrderContainer order);

}

4) Пример класса с реализацией методов для доступа к данным (OrderDataAccess)

public class OrderDataAccess : IOrderDataAccess

{

private BookStoreContext Context { get; }

private IMapper Mapper { get; }

public OrderDataAccess(BookStoreContext context, IMapper mapper)

{

this.Context = context;

Mapper = mapper;

}

public async Task<Order> InsertAsync(OrderUpdateModel order)

{

var result = await this.Context.AddAsync(this.Mapper.Map<DataAccess.Entities.Order>(order));

await this.Context.SaveChangesAsync();

return this.Mapper.Map<Order>(result.Entity);

}

public async Task<IEnumerable<Order>> GetAsync()

{

return this.Mapper.Map<IEnumerable<Order>>(await this.Context.Order.Include(x => x.Book).Include(x => x.Client).ToListAsync());

}

public async Task<Order> GetAsync(IOrderIdentity orderId)

{

var result = await this.Get(orderId);

return this.Mapper.Map<Order>(result);

}

private async Task<Store.DataAccess.Entities.Order> Get(IOrderIdentity orderId)

{

if (orderId == null)

throw new ArgumentNullException(nameof(orderId));

return await this.Context.Order.Include(x => x.Book).Include(x => x.Client).FirstOrDefaultAsync(x => x.Id == orderId.Id);

}

public async Task<Order> UpdateAsync(OrderUpdateModel order)

{

var existing = await this.Get(order);

var result = this.Mapper.Map(order, existing);

this.Context.Update(result);

await this.Context.SaveChangesAsync();

return this.Mapper.Map<Order>(result);

}

public async Task<Order> GetByAsync(IOrderContainer order)

{

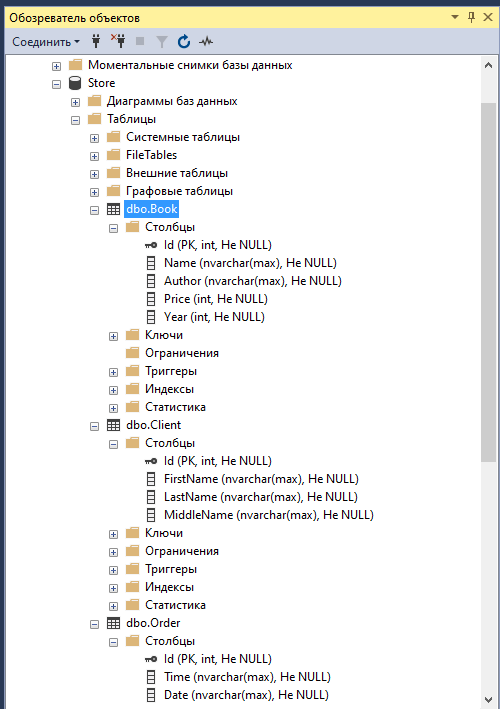
return order.OrderId.HasValue

? this.Mapper.Map<Order>(await this.Context.Order.FirstOrDefaultAsync(x => x.Id == order.OrderId))

: null;

}

}



**Выводы**

В процессе выполнения лабораторной работы был реализован доступа к данным приложения в среде .NET. Были получены навыки по разработке организации доступа приложения к внешней базе данных с помощью EntityFrameworkCore и MS SQL Server.