**МИНИСТЕРСТВО ОБРАЗОВАНИЯ РЕСПУБЛИКИ БЕЛАРУСЬ**

**УЧРЕЖДЕНИЕ ОБРАЗОВАНИЯ**

**ГОМЕЛЬСКИЙ ГОСУДАРСТВЕННЫЙ ТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ ИМЕНИ П. О. СУХОГО**

Факультет автоматизированных и информационных систем

Кафедра «Информационные технологии»

**ЛАБОРАТОРНАЯ РАБОТА №2**

по дисциплине: «Разработка приложений баз данных для информационных систем»

на тему: «Использование *ENTITY FRAMEWORK* и *LINQ* для работы с базами данных»

Выполнил: студент гр. ИТП-31

Иванов Д. А.

Принял: ректор

Асенчик О.Д.

Гомель 2023

**Цель работы:** ознакомиться с возможностями *ENTITY* *FRAMEWORK* и получить навыки написания *LINQ* запросов к объектам, связанным с таблицами базы данных СУБД *MS* *SQL* сервер.

**Задание:**

1.1. Создать с использованием .*NET* *Core* *Entity* *Framework* Core консольное приложение, содержащее набор классов, моделирующих предметную область соответствующей своему варианту и ранее созданную и заполненной тестовыми данными задания базой *MS* *SQL* *Server*. Для этого необходимо создать:

• Классы, моделирующие не менее чем три таблицы базы данных согласно вашему варианту.

• Класс контекста данных.

1.2. Выполнить, используя объекты *Entity* *Framework* *Core* и *LINQ*:

1. Выборку всех данных из таблицы, стоящей в схеме базы данных нас стороне отношения «один» – 1 шт.

2. Выборку данных из таблицы, стоящей в схеме базы данных нас стороне отношения «один», отфильтрованные по определенному условию, налагающему ограничения на одно или несколько полей – 1 шт.

3. Выборку данных, сгруппированных по любому из полей данных с выводом какого-либо итогового результата (*min*, *max*, *avg*, *сount* или др.) по выбранному полю из таблицы, стоящей в схеме базы данных нас стороне отношения «многие» – 1 шт.

4. Выборку данных из двух полей двух таблиц, связанных между собой отношением «один-ко-многим» – 1 шт.

5. Выборку данных из двух таблиц, связанных между собой отношением «один-ко-многим» и отфильтрованным по некоторому условию, налагающему ограничения на значения одного или нескольких полей – 1 шт.

6. Вставку данных в таблицы, стоящей на стороне отношения «Один» – 1 шт.

7. Вставку данных в таблицы, стоящей на стороне отношения «Многие» – 1 шт.:

8. Удаление данных из таблицы, стоящей на стороне отношения «Один» – 1 шт.

9. Удаление данных из таблицы, стоящей на стороне отношения «Многие» – 1 шт.

10. Обновление удовлетворяющих определенному условию записей в любой из таблиц базы данных – 1 шт.

1.3. Разместить выполненный проект на *github*.

**Ход работы**

В ходе выполнения лабораторной работы было создано консольное приложение на версии .*NET* 7.0. Далее для работы с базой данных при помощи *ENTITY FRAMEWORK* были установлены необходимые *NuGET* пакеты:

1. *Microsoft.EntityFrameworkCore.Design ­*­– Предоставляет интерфейсы для связи между базой данных и *C*#.
2. *Microsoft.EntityFrameworkCore.SqlServer* – Предоставляет функционал для связи ORM фреймворка с базой данных *MS SQL Server.*
3. *Microsoft.EntityFrameworkCore.Tools* – Предоставляет список команд для работы с миграциями, переносом созданной базы в виде объектов в *SQL* и переносу *SQL* базы данных в объекты *C*#.

После установки всех необходимых пакетов приложение готово для работы с базой данных через *ENTITY FRAMEWORK*. В основе *ENTITY FRAMEWORK* лежит технология *ORM* (*object-relational mapping* – отображения данных на реальные объекты) которая позволят связать таблицы баз данных с *C*# объектами. Для связи объектов по определенным правилам существует три подхода.

*Code Convention* для работы нет необходимости что-либо настраивать собственноручно. Например, при создании объекта модели указывается поле *Id* и логично что данное поле должно иметь свойство *PRIMARY KEY*. ENTITY FRAMEWORK уже из названия поля понимает это и при переносе этой модели в базу данных устанавливает этому столбцу значения *PRIMARY KEY.* Это подхода вполне достаточно для простых баз данных с несложными связями. Для более тонкой настройки существует другие два подхода: через атрибуты или *Fluent API*.

При установлении свойств через атрибуты пользователь может указать сколько угодно атрибутов для каждого свойства модели. Например, атрибут *Table* для класса модели указывает как будет называться модель в базе данных. Атрибут *Key* для свойства модели указывает что данное поле будет первичным ключом в базе данных. Это подход позволяет более тонко настаивать ограничения и связи в базе данных. Но его минусом является то, что если база дынных содержит много таблиц, то для настройки связей между таблицами необходимо каждый раз переключатся между ними. Из-за этого очень легко запутаться в созданной структуре. Намного удобнее, когда все настройки моделей хранятся в одном классе. С проблемой и призвана справится такая технология как *Fluent API*.

Перед тек как перейти в технологии *Fluent API* необходимо понять, что сами по себе в отдельность модели никак не могут быть связаны с базой данных. Для этого в *ENTITY FRAMEWORK* есть специальный класс *DbContext* который предоставляет специальное *API* для подключения к базе данных и связи моделей в *C*# с таблицами в *SQL*. Именно в этом классе и прописываются ограничения при помощи *Fluent API.* Это делаете при помощи паттерна строитель и метод расширения. Плюсом данного подхода настройки моделей является то, что вся конфигурации хранится в одном файле.

При работе с *ORM* существует два похода *Database-first* и *Code-first*. В первом походе сначала создается база данных, а потом при помощи *ENTITY FRAMEWORK* она переносится в *C*#. Другой подход – это тогда, когда программист уже создал все необходимые модели в *C*# и для того, чтобы не создавать такую же базу данных с нуля можно просто перенести все свои модели в базу данных при помощи *ENTITY FRAMWORK*.

Так как в предыдущей лабораторной работе уже была создана база данных то тут подойдет подход *Database-first.* Для переноса всех таблиц из базы данных в модели используется специальная команда *Scaffold-DbContext*. Которая в параметры принимает строку подключения к базе данных и провайдера, например *SQLServer.* После выполнения этой команды создаются классы моделей и класс контекста. Листинг этих классов указан в приложении А.

Далее для хранения запросов при помощи *LINQ* был создан класс *Queries*. Листинг данного класса указан в приложении А.

Первый запрос выполняет выборку данных из таблицы на стороне отношения один. Он достает данные из таблицы покупатели. Листинг данного метода *SelectOneRelation* указан в приложении А. Пример работы этого запроса указан на рисунке 1.

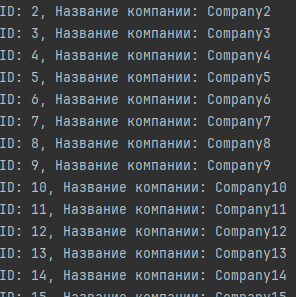


Рисунок 1 – Пример запроса на выборку данных из таблицы на стороне отношения один

Далее был реализован запрос на выборку данных из таблицы на стороне отношения один с последующей фильтрацией данных. Для этого был реализован метод *SelectOneRelationWithFilter* который в параметры принимает строку *filter* и выводит только те компании, которые начинаются с этого текста в своем названии. Листинг данного метода *SelectOneRelationWithFilter* указан в приложении А. Пример работы этого запроса указан на рисунке 2.

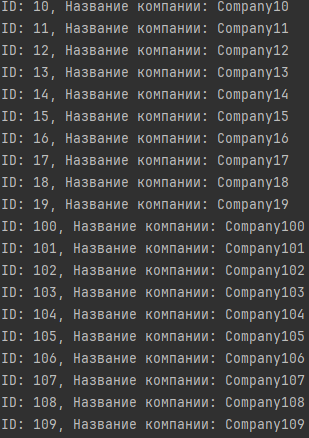


Рисунок 2 – Пример запроса на выборку данных из таблицы на стороне отношения один с фильтрацией данных

Далее был разработан запрос на выборку данных из таблицы стоящую на стороне отношения многие и агрегирующие полученный результат. В данном случае был разработан метод *SelectManyRelationWithAggregation* который выводит среднюю цену заказа для каждого клиента. Листинг данного метода *SelectManyRelationWithAggregation* указан в приложении А. Пример работы этого запроса указан на рисунке 3.

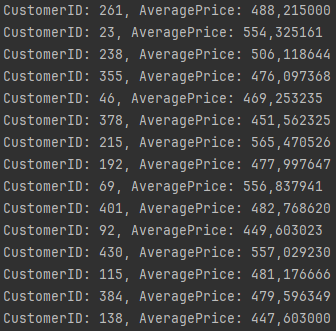


Рисунок 3 – Пример запроса на выборку данных из таблицы на стороне отношения многие с последующей агрегацией данных

Далее был разработан запрос на выборку данных, стоящую на стороне отношения один-многие с выводом данных из обоих таблиц. Для этого был реализован метод *SelectOneToManyRelationWithJoin* который выбирает данные из таблиц *Customers* и *Orders.* Для создания связи между двумя таблицами используется метод *Join* в который передается название навигационного свойства. В результате на экран будет выведено название компании покупателя из таблицы *Customers* и дата заказа из таблицы *Orders*. Листинг данного метода *SelectOneToManyRelationWithJoin* указан в приложении А. Пример работы этого запроса указан на рисунке 4.

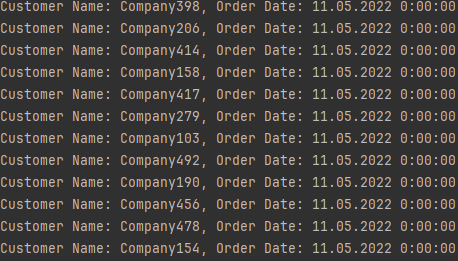


Рисунок 4 – Пример запроса на выборку данных из таблиц на стороне отношения один-многие

Далее был разработан запрос на выборку данных из таблиц, стоящих на стороне отношения один-многие с последующей фильтрацией данных. Для этого был разработан метод *SelectOneToManyRelationWithJoinAndFilter*. В данном методе выборка производится из таблиц *Customers*  и *Orders.* И фильтрация производится по полю *CustomerId* в таблице *Customers* и по полю *OrderDate* в таблице *Orders*. Листинг данного метода *SelectOneToManyRelationWithJoinAndFilter* указан в приложении А. Пример работы этого запроса указан на рисунке 5.

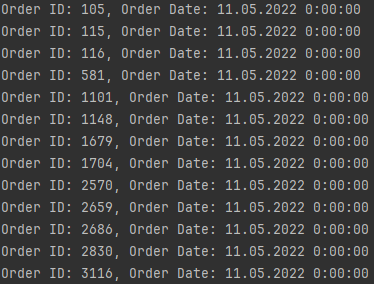


Рисунок 5 – Пример запроса на выборку данных из таблиц на стороне отношения один-многие с последующей фильтрацией данных

Далее был разработан запрос на добавления данных в таблицу, стоящую на стороне отношения один. Для этого был разработан метод *InsertOneRelation* который добавляет данные в таблицу *Customers*. Для добавления данных *SQL* базу в *ENTITY FRAMEWORK* достаточно добавить объект в нужный *DbSet* и у объекта контекста вызвать метод *SaveChanges*. Листинг данного метода *InsertOneRelation* указан в приложении А. Пример работы этого запроса указан на рисунке 6.



Рисунок 6 – Пример добавленных данных

Далее был разработан запрос на добавления данных в таблицу, стоящую на стороне отношения многие. Для этого был разработан метод *InsertManyRelation* который добавляет данные в таблицу *Orders*. Если объект хранит в себе ссылки на другие объекты базы данных, то при добавлении можно просто указать *Id* нужного элемента в базе данных или указать новый объект и если его нет в базе данных, то он автоматически создастся. В данном случае объект *InsuranceAgent* хранит в себе ссылки на таблицы *Contracts* и *AgentType*. Листинг данного метода *InsertManyRelation* указан в приложении А. Пример работы этого запроса указан на рисунке 8.



Рисунок 7 – Пример добавленных данных

Далее был реализован запрос для удаления данных из таблицы, стоящей на стороне отношения многие. Для этого был реализован метод *DeleteOneRelation* который удаляет данные из таблицы *Customers* и в качестве параметра принимает CustomerId*.* Для удаления данных достаточно вызвать метод *Remove* для объекта *DbSet* и у объекта класса контекста вызвать метод *SaveChanges.* Листинг данного метода *DeleteOneRelation* указан в приложении А. Пример работы этого запроса указан на рисунке 10.



Рисунок 10 – Пример удаления данных на стороне отношения один

Далее был реализован запрос на удаления данных из таблицы, стоящей на стороне отношения многие. Для этого был реализован метод *DeleteManyRelation* который удаляет данные в таблице *Orders* в качестве параметров метод принимает *OrderId*. Если этот класс содержит ссылки на другие объекты базы данных, то они удаляются согласно правилам, заданным при помощи атрибутов или *Fluent API.* Например, если удаления стоит каскадное, то удалятся сама запись в базе данных и все связанные с ней записи из других таблиц. Листинг данного метода *DeleteManyRelation* указан в приложении А. Пример работы этого запроса указан на рисунке 11.

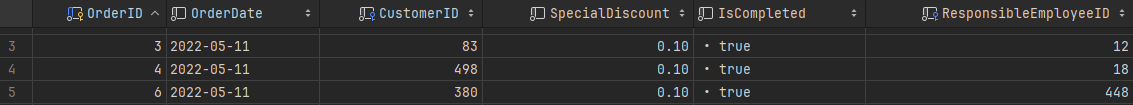


Рисунок 11 – Пример удаления данных на стороне отношения многие

Далее был разработан запрос на обновления данных в таблице. Для этого был разработан метод *UpdateWithCondition* который принимает в качестве параметра название компании, данные которой стоит изменить. Для сохранения изменений в базе данных достаточно изменить определенный параметр в *DbSet* и вызвать метод *SaveChanges.* Листинг данного метода *UpdateWithCondition* указан в приложении А. Пример работы этого запроса указан на рисунке 12.



Рисунок 12 – Пример обновления данных в базе данных

После выполнения лабораторной работы созданные проект был добавлен в локальный *git* репозиторий а потом перенесен в *GitHub* репозиторий своего аккаунта. Чтобы ознакомится с созданным проектом можно по ссылке [*https://github.com/Javaro3/lab2\_DB*](https://github.com/Javaro3/lab2_DB)*.*

**Вывод**: в ходе выполнения лабораторной работы была изучена такая технология как *ENTITY FRAMEWORK* которая позволяет работать с базой данный при помощи объектов *C*#. Также для работы с *ORM* была использована такая технология как *LINQ* которая позволяет удобно работать с данными. В ходе выполнения лабораторной работы база данных созданная в первой лабораторной работе была перенесена в *C*#. Были изучены основный функции *ENTITY FRAMEWORK* для агрегации, фильтрации и группировки данных при помощи *LINQ* запросов.

**ПРИЛОЖНИЕ А**

Листинг класса *Customer*

namespace lab2.Models;  
  
public partial class Customer  
{  
 public int CustomerId { get; set; }  
  
 public string CompanyName { get; set; } = null!;  
  
 public string RepresentativeLastName { get; set; } = null!;  
  
 public string RepresentativeFirstName { get; set; } = null!;  
  
 public string RepresentativeMiddleName { get; set; } = null!;  
  
 public string PhoneNumber { get; set; } = null!;  
  
 public string Address { get; set; } = null!;  
  
 public virtual ICollection<Order> Orders { get; set; } = new List<Order>();  
}

Листинг класса *Employee*

namespace lab2.Models;  
  
public partial class Employee  
{  
 public int EmployeeId { get; set; }  
  
 public string LastName { get; set; } = null!;  
  
 public string FirstName { get; set; } = null!;  
  
 public string MiddleName { get; set; } = null!;  
  
 public string Position { get; set; } = null!;  
  
 public string Education { get; set; } = null!;  
  
 public virtual ICollection<Invoice> Invoices { get; set; } = new List<Invoice>();  
  
 public virtual ICollection<Order> Orders { get; set; } = new List<Order>();  
}

Листинг класса *Furniture*

namespace lab2.Models;  
  
public partial class Furniture  
{  
 public int FurnitureId { get; set; }  
  
 public string FurnitureName { get; set; } = null!;  
  
 public string? Description { get; set; }  
  
 public string MaterialType { get; set; } = null!;  
  
 public decimal Price { get; set; }  
  
 public int QuantityOnHand { get; set; }  
  
 public virtual ICollection<OrderDetail> OrderDetails { get; set; } = new List<OrderDetail>();  
}

Листинг класса *Invoice*

namespace lab2.Models;  
  
public partial class Invoice  
{  
 public int InvoiceId { get; set; }  
  
 public string SupplierName { get; set; } = null!;  
  
 public DateTime DeliveryDate { get; set; }  
  
 public string MaterialType { get; set; } = null!;  
  
 public decimal Price { get; set; }  
  
 public decimal Weight { get; set; }  
  
 public int ReceivedByEmployeeId { get; set; }  
  
 public virtual Employee ReceivedByEmployee { get; set; } = null!;  
}

Листинг класса *Order*

namespace lab2.Models;  
  
public partial class Order  
{  
 public int OrderId { get; set; }  
  
 public DateTime OrderDate { get; set; }  
  
 public int CustomerId { get; set; }  
  
 public decimal SpecialDiscount { get; set; }  
  
 public bool IsCompleted { get; set; }  
  
 public int ResponsibleEmployeeId { get; set; }  
  
 public virtual Customer Customer { get; set; } = null!;  
  
 public virtual ICollection<OrderDetail> OrderDetails { get; set; } = new List<OrderDetail>();  
  
 public virtual Employee ResponsibleEmployee { get; set; } = null!;  
}

Листинг класса *OrderDetail*

namespace lab2.Models;  
  
public partial class OrderDetail  
{  
 public int OrderDetailId { get; set; }  
  
 public int OrderId { get; set; }  
  
 public int FurnitureId { get; set; }  
  
 public int Quantity { get; set; }  
  
 public virtual Furniture Furniture { get; set; } = null!;  
  
 public virtual Order Order { get; set; } = null!;  
}

Листинг класса *AcmeDataContext*

using System;  
using System.Collections.Generic;  
using lab2.Models;  
using Microsoft.EntityFrameworkCore;  
  
namespace lab2;  
  
public partial class AcmeDataContext : DbContext  
{  
 public AcmeDataContext()  
 {  
 }  
  
 public AcmeDataContext(DbContextOptions<AcmeDataContext> options)  
 : base(options)  
 {  
 }  
  
 public virtual DbSet<Customer> Customers { get; set; }  
  
 public virtual DbSet<CustomerTotalOrder> CustomerTotalOrders { get; set; }  
  
 public virtual DbSet<Employee> Employees { get; set; }  
  
 public virtual DbSet<EmployeeTotalCostWeight> EmployeeTotalCostWeights { get; set; }  
  
 public virtual DbSet<Furniture> Furnitures { get; set; }  
  
 public virtual DbSet<Invoice> Invoices { get; set; }  
  
 public virtual DbSet<Order> Orders { get; set; }  
  
 public virtual DbSet<OrderDetail> OrderDetails { get; set; }  
  
 public virtual DbSet<OrderDetailsView> OrderDetailsViews { get; set; }  
  
 protected override void OnConfiguring(DbContextOptionsBuilder optionsBuilder)  
 => optionsBuilder.UseSqlServer("Server=SAL\\SQLEXPRESS;Database=FurnitureFactory; Trusted\_Connection =True; TrustServerCertificate=True;");  
  
 protected override void OnModelCreating(ModelBuilder modelBuilder)  
 {  
 modelBuilder.Entity<Customer>(entity =>  
 {  
 entity.HasKey(e => e.CustomerId).HasName("PK\_\_Customer\_\_A4AE64B8AC00F230");  
  
 entity.Property(e => e.CustomerId).HasColumnName("CustomerID");  
 entity.Property(e => e.Address).HasMaxLength(255);  
 entity.Property(e => e.CompanyName).HasMaxLength(255);  
 entity.Property(e => e.PhoneNumber).HasMaxLength(15);  
 entity.Property(e => e.RepresentativeFirstName).HasMaxLength(255);  
 entity.Property(e => e.RepresentativeLastName).HasMaxLength(255);  
 entity.Property(e => e.RepresentativeMiddleName).HasMaxLength(255);  
 });  
  
 modelBuilder.Entity<CustomerTotalOrder>(entity =>  
 {  
 entity  
 .HasNoKey()  
 .ToView("CustomerTotalOrders");  
  
 entity.Property(e => e.Address).HasMaxLength(255);  
 entity.Property(e => e.CompanyName).HasMaxLength(255);  
 entity.Property(e => e.CustomerId).HasColumnName("CustomerID");  
 entity.Property(e => e.PhoneNumber).HasMaxLength(15);  
 entity.Property(e => e.RepresentativeFirstName).HasMaxLength(255);  
 entity.Property(e => e.RepresentativeLastName).HasMaxLength(255);  
 entity.Property(e => e.RepresentativeMiddleName).HasMaxLength(255);  
 entity.Property(e => e.TotalPurchases).HasColumnType("decimal(38, 4)");  
 });  
  
 modelBuilder.Entity<Employee>(entity =>  
 {  
 entity.HasKey(e => e.EmployeeId).HasName("PK\_\_Employee\_\_7AD04FF1BB42DE48");  
  
 entity.Property(e => e.EmployeeId).HasColumnName("EmployeeID");  
 entity.Property(e => e.Education).HasMaxLength(255);  
 entity.Property(e => e.FirstName).HasMaxLength(255);  
 entity.Property(e => e.LastName).HasMaxLength(255);  
 entity.Property(e => e.MiddleName).HasMaxLength(255);  
 entity.Property(e => e.Position).HasMaxLength(100);  
 });  
  
 modelBuilder.Entity<EmployeeTotalCostWeight>(entity =>  
 {  
 entity  
 .HasNoKey()  
 .ToView("EmployeeTotalCostWeight");  
  
 entity.Property(e => e.EmployeeEducation).HasMaxLength(255);  
 entity.Property(e => e.EmployeeFirstName).HasMaxLength(255);  
 entity.Property(e => e.EmployeeId).HasColumnName("EmployeeID");  
 entity.Property(e => e.EmployeeLastName).HasMaxLength(255);  
 entity.Property(e => e.EmployeeMiddleName).HasMaxLength(255);  
 entity.Property(e => e.EmployeePosition).HasMaxLength(100);  
 entity.Property(e => e.TotalCost).HasColumnType("decimal(38, 2)");  
 entity.Property(e => e.TotalWeight).HasColumnType("decimal(38, 2)");  
 });  
  
 modelBuilder.Entity<Furniture>(entity =>  
 {  
 entity.HasKey(e => e.FurnitureId).HasName("PK\_\_Furnitur\_\_D4323505FFDAB4BF");  
  
 entity.ToTable("Furniture");  
  
 entity.Property(e => e.FurnitureId).HasColumnName("FurnitureID");  
 entity.Property(e => e.FurnitureName).HasMaxLength(255);  
 entity.Property(e => e.MaterialType).HasMaxLength(50);  
 entity.Property(e => e.Price).HasColumnType("decimal(10, 2)");  
 });  
  
 modelBuilder.Entity<Invoice>(entity =>  
 {  
 entity.HasKey(e => e.InvoiceId).HasName("PK\_\_Invoices\_\_D796AAD5F186281F");  
  
 entity.Property(e => e.InvoiceId).HasColumnName("InvoiceID");  
 entity.Property(e => e.DeliveryDate).HasColumnType("date");  
 entity.Property(e => e.MaterialType).HasMaxLength(50);  
 entity.Property(e => e.Price).HasColumnType("decimal(10, 2)");  
 entity.Property(e => e.ReceivedByEmployeeId).HasColumnName("ReceivedByEmployeeID");  
 entity.Property(e => e.SupplierName).HasMaxLength(255);  
 entity.Property(e => e.Weight).HasColumnType("decimal(10, 2)");  
  
 entity.HasOne(d => d.ReceivedByEmployee).WithMany(p => p.Invoices)  
 .HasForeignKey(d => d.ReceivedByEmployeeId)  
 .OnDelete(DeleteBehavior.**ClientSetNull**)  
 .HasConstraintName("FK\_\_Invoices\_\_Receiv\_\_44FF419A");  
 });  
  
 modelBuilder.Entity<Order>(entity =>  
 {  
 entity.HasKey(e => e.OrderId).HasName("PK\_\_Orders\_\_C3905BAF2EEEDECA");  
  
 entity.Property(e => e.OrderId).HasColumnName("OrderID");  
 entity.Property(e => e.CustomerId).HasColumnName("CustomerID");  
 entity.Property(e => e.OrderDate).HasColumnType("date");  
 entity.Property(e => e.ResponsibleEmployeeId).HasColumnName("ResponsibleEmployeeID");  
 entity.Property(e => e.SpecialDiscount).HasColumnType("decimal(5, 2)");  
  
 entity.HasOne(d => d.Customer).WithMany(p => p.Orders)  
 .HasForeignKey(d => d.CustomerId)  
 .OnDelete(DeleteBehavior.**ClientSetNull**)  
 .HasConstraintName("FK\_\_Orders\_\_Customer\_\_3D5E1FD2");  
  
 entity.HasOne(d => d.ResponsibleEmployee).WithMany(p => p.Orders)  
 .HasForeignKey(d => d.ResponsibleEmployeeId)  
 .OnDelete(DeleteBehavior.**ClientSetNull**)  
 .HasConstraintName("FK\_\_Orders\_\_Responsi\_\_3E52440B");  
 });  
  
 modelBuilder.Entity<OrderDetail>(entity =>  
 {  
 entity.HasKey(e => e.OrderDetailId).HasName("PK\_\_OrderDet\_\_D3B9D30CC07AF555");  
  
 entity.Property(e => e.OrderDetailId).HasColumnName("OrderDetailID");  
 entity.Property(e => e.FurnitureId).HasColumnName("FurnitureID");  
 entity.Property(e => e.OrderId).HasColumnName("OrderID");  
  
 entity.HasOne(d => d.Furniture).WithMany(p => p.OrderDetails)  
 .HasForeignKey(d => d.FurnitureId)  
 .OnDelete(DeleteBehavior.**ClientSetNull**)  
 .HasConstraintName("FK\_\_OrderDeta\_\_Furni\_\_4222D4EF");  
  
 entity.HasOne(d => d.Order).WithMany(p => p.OrderDetails)  
 .HasForeignKey(d => d.OrderId)  
 .OnDelete(DeleteBehavior.**ClientSetNull**)  
 .HasConstraintName("FK\_\_OrderDeta\_\_Order\_\_412EB0B6");  
 });  
  
 modelBuilder.Entity<OrderDetailsView>(entity =>  
 {  
 entity  
 .HasNoKey()  
 .ToView("OrderDetailsView");  
  
 entity.Property(e => e.Address).HasMaxLength(255);  
 entity.Property(e => e.CompanyName).HasMaxLength(255);  
 entity.Property(e => e.CustomerId).HasColumnName("CustomerID");  
 entity.Property(e => e.FurnitureId).HasColumnName("FurnitureID");  
 entity.Property(e => e.FurnitureMaterial).HasMaxLength(50);  
 entity.Property(e => e.FurnitureName).HasMaxLength(255);  
 entity.Property(e => e.OrderDate).HasColumnType("date");  
 entity.Property(e => e.OrderId).HasColumnName("OrderID");  
 entity.Property(e => e.PhoneNumber).HasMaxLength(15);  
 entity.Property(e => e.RepresentativeFirstName).HasMaxLength(255);  
 entity.Property(e => e.RepresentativeLastName).HasMaxLength(255);  
 entity.Property(e => e.RepresentativeMiddleName).HasMaxLength(255);  
 entity.Property(e => e.TotalPrice).HasColumnType("decimal(28, 4)");  
 });  
  
 OnModelCreatingPartial(modelBuilder);  
 }  
  
 partial void OnModelCreatingPartial(ModelBuilder modelBuilder);  
}

Листинг класса *Program*

using lab2.Models;  
using Microsoft.EntityFrameworkCore;  
  
namespace lab2;  
  
public static class Program  
{  
 private static void Main()  
 {  
 using var context = new AcmeDataContext();  
 SelectOneRelation(context);  
 SelectOneRelationWithFilter(context, "Company1");  
 SelectManyRelationWithAggregation(context);  
 SelectOneToManyRelationWithJoin(context);  
 SelectOneToManyRelationWithJoinAndFilter(context, 3, new DateTime(2022, 5, 10));  
 InsertOneRelation(context);  
 InsertManyRelation(context);  
 DeleteOneRelation(context, 1);  
 DeleteManyRelation(context, 5);  
 UpdateWithCondition(context, "Company10");  
 }  
  
 private static void SelectOneRelation(AcmeDataContext context)  
 {  
 var customers = context.Customers.ToList();  
  
 foreach (var customer in customers)  
 {  
 Console.WriteLine($"ID: {customer.CustomerId}, Название компании: {customer.CompanyName}");  
 }  
 }  
  
 private static void SelectOneRelationWithFilter(AcmeDataContext context, string filterString)  
 {  
 var filteredCustomers = context.Customers  
 .Where(customer => customer.CompanyName.StartsWith(filterString))  
 .ToList();  
  
 foreach (var customer in filteredCustomers)  
 {  
 Console.WriteLine($"ID: {customer.CustomerId}, Название компании: {customer.CompanyName}");  
 }  
 }  
   
 private static void SelectManyRelationWithAggregation(AcmeDataContext context)  
 {  
 var orderAverages = context.OrderDetails  
 .Join(context.Furnitures,  
 orderDetail => orderDetail.FurnitureId,  
 furniture => furniture.FurnitureId,  
 (orderDetail, furniture) => new { orderDetail, furniture })  
 .GroupBy(joinResult => joinResult.orderDetail.Order.CustomerId)  
 .Select(group => new  
 {  
 CustomerID = group.Key,  
 AveragePrice = group.Average(joinResult => joinResult.furniture.Price)  
 })  
 .ToList();  
  
 foreach (var result in orderAverages)  
 {  
 Console.WriteLine($"CustomerID: {result.CustomerID}, AveragePrice: {result.AveragePrice}");  
 }  
 }  
   
 private static void SelectOneToManyRelationWithJoin(AcmeDataContext context)  
 {  
 var customerOrders = context.Customers  
 .Join(  
 context.Orders,  
 customer => customer.CustomerId,  
 order => order.CustomerId,  
 (customer, order) => new  
 {  
 CustomerName = customer.CompanyName,  
 OrderDate = order.OrderDate  
 })  
 .ToList();  
  
 foreach (var entry in customerOrders)  
 {  
 Console.WriteLine($"Customer Name: {entry.CustomerName}, Order Date: {entry.OrderDate}");  
 }  
 }  
   
 private static void SelectOneToManyRelationWithJoinAndFilter(AcmeDataContext context, int targetCustomerId, DateTime targetDate)  
 {  
 var customerOrders = context.Customers  
 .Where(customer => customer.CustomerId == targetCustomerId)  
 .SelectMany(customer => customer.Orders)  
 .Where(order => order.OrderDate >= targetDate)  
 .ToList();  
  
 foreach (var order in customerOrders)  
 {  
 Console.WriteLine($"Order ID: {order.OrderId}, Order Date: {order.OrderDate}");  
 }  
 }  
   
 private static void InsertOneRelation(AcmeDataContext context)  
 {  
 var newCustomer = new Customer  
 {  
 CompanyName = "Новая Компания",  
 RepresentativeLastName = "Фамилия",  
 RepresentativeFirstName = "Имя",  
 RepresentativeMiddleName = "Отчество",  
 PhoneNumber = "123-456-7890",  
 Address = "Адрес"  
 };  
  
 context.Customers.Add(newCustomer);  
 context.SaveChanges();   
 }  
   
 private static void InsertManyRelation(AcmeDataContext context)  
 {  
 var newOrder = new Order  
 {  
 OrderDate = DateTime.Now,  
 CustomerId = 10,  
 SpecialDiscount = (decimal)0.1,   
 IsCompleted = false,  
 ResponsibleEmployeeId = 20   
 };  
  
 var orderDetail1 = new OrderDetail  
 {  
 FurnitureId = 30,   
 Quantity = 2   
 };  
  
 var orderDetail2 = new OrderDetail  
 {  
 FurnitureId = 50,   
 Quantity = 1   
 };  
  
 newOrder.OrderDetails.Add(orderDetail1);  
 newOrder.OrderDetails.Add(orderDetail2);  
  
 context.Orders.Add(newOrder);  
  
 context.SaveChanges();  
 }  
   
 private static void DeleteOneRelation(AcmeDataContext context, int customerId)  
 {  
 var customerToDelete = context.Customers  
 .Include(c => c.Orders)  
 .ThenInclude(o => o.OrderDetails) *// Включите связанные записи OrderDetails* .SingleOrDefault(c => c.CustomerId == customerId);  
  
 if (customerToDelete == null) return;  
 context.OrderDetails.RemoveRange(  
 customerToDelete.Orders.SelectMany(o => o.OrderDetails));  
  
 context.Orders.RemoveRange(customerToDelete.Orders);  
  
 context.Customers.Remove(customerToDelete);  
  
 context.SaveChanges();  
 }  
   
 private static void DeleteManyRelation(AcmeDataContext context, int orderId)  
 {  
 var orderToDelete = context.Orders  
 .Include(o => o.OrderDetails)  
 .SingleOrDefault(o => o.OrderId == orderId);  
  
 if (orderToDelete == null) return;  
 context.OrderDetails.RemoveRange(orderToDelete.OrderDetails);  
   
 context.Orders.Remove(orderToDelete);  
  
 context.SaveChanges();  
 }  
   
 private static void UpdateWithCondition(AcmeDataContext context, string targetCompanyName)  
 {  
 var customersToUpdate = context.Customers  
 .Where(customer => customer.CompanyName == targetCompanyName)  
 .ToList();  
  
 customersToUpdate.ForEach(customer =>  
 {  
 customer.RepresentativeLastName = "Новая Фамилия";   
 customer.RepresentativeFirstName = "Новое Имя";  
 customer.RepresentativeMiddleName = "Новое Отчество";  
 customer.PhoneNumber = "Новый Телефон";  
 customer.Address = "Новый Адрес";  
 });  
  
 context.SaveChanges();  
 }  
}