**..МИНИСТЕРСТВО ОБРАЗОВАНИЯ РЕСПУБЛИКИ БЕЛАРУСЬ**

**УЧРЕЖДЕНИЕ ОБРАЗОВАНИЯ**

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

Факультет автоматизированных и информационных систем

Кафедра «Информационные технологии»

дисциплина «Разработка приложений баз данных для информационных систем»

ОТЧЕТ ПО ЛАБОРАТОРНОЙ РАБОТЕ № 2

«Использование ENTITY FRAMEWORK и LINQ для работы с базами данных»

ВАРИАНТ № 15

Выполнил:

студент гр. ИТП-31, Козлов О. В.

Принял:

доцент 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:** 3.1. 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.

3.2. 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.

**Ход работы:** для начала необходимо сформировать исходные классы моделей и класс контекста данных, согласно теме варианта. Так как сама база данных и таблицы были разработаны в первой лабораторной работе, то можно воспользоваться методом Scaffolding. Данный метод позволит автоматически создать все необходимые классы моделей и сам класс контекста данных. На рисунке 1 показа команда, которая формирует все классы.

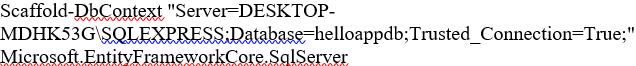


Рисунок 1 – Метод *Scaffolding*

Когда процесс формирования классов, можно приступить к работе с базой данных. Сама связь между базой данных и данными в программе обеспечивается при помощи класса контекста данных. Сделать это можно, например в блоке *using*, создав в нем объект контекста. На рисунке 2 показано начало блока, в котором будет происходить работа с базой данных.



Рисунок 2 – Работа с контекстом

Далее можно создать функции, выполняющие требуемые *LINQ*-запросы. Работа первого запроса выполняется при помощи простого *select* запроса. На рисунке 3 показан результат получения данных в консоли.

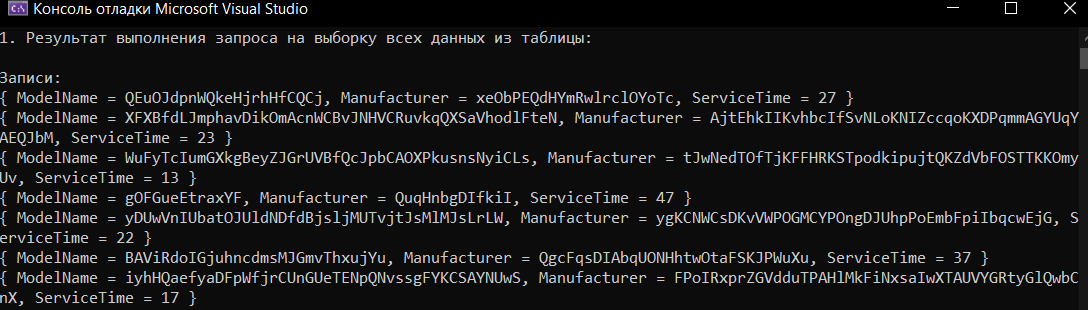


Рисунок 3 – Первый запрос

Второй запрос содержит условие для отбора данных. Данное условие задается при помощи слова *where.* На рисунке 4 показана работа запроса на получение данных о моделях счетчиков, чей срок службы больше 20.

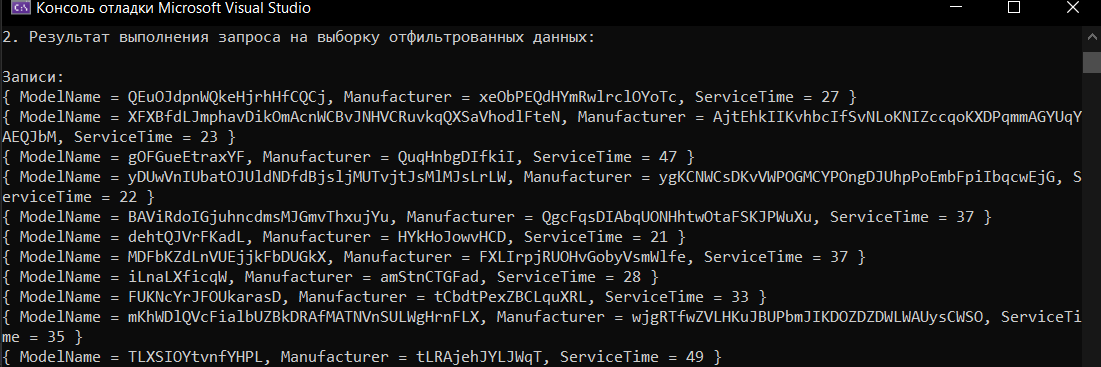


Рисунок 4 – Фильтрация данных

Для запроса к нескольким таблицам необходимо использовать оператор *join*, далее при помощи *on* указать по какому столбцу происходит связь. На рисунке 5 показана работа запроса к нескольким таблицам.

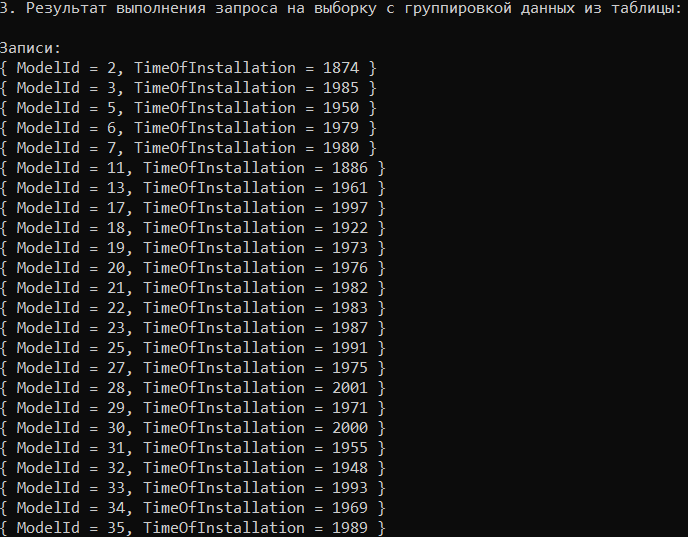


Рисунок 5 – Запрос к нескольким таблицам

Таким же образом создается и следующий запрос на два поля из двух таблиц. Результат работы запроса показан на рисунке 6.

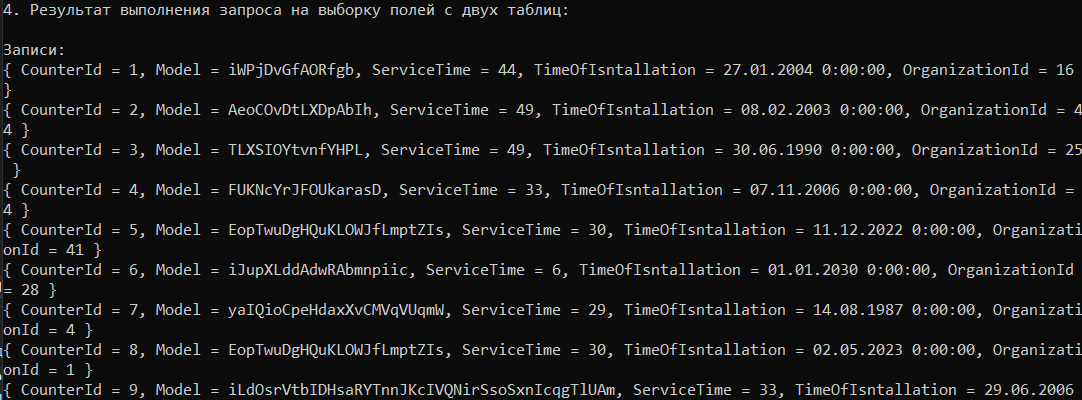


Рисунок 6 – Запрос к двум таблицам

Фильтрация данных из нескольких таблиц выполняется тем же образом, как и для одной таблицы. Единственным отличием является необходимость указания названия таблицы перед названием столбца. На рисунке 7 показан результат.

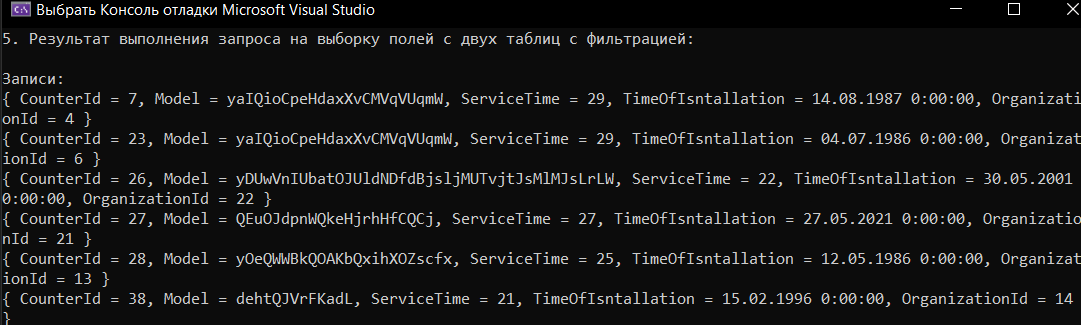


Рисунок 7 – Запрос двух таблиц с фильтрацией

Запросы на вставку данных в таблицы имеют схожую структуру, как для стороны «один», так и для «многие». Для добавления данных необходимо создать экземпляр класса записи соответствующей таблицы и передать в него все требуемые параметры. После методом *Add*() занести элемент в таблицу. На рисунке 8 показано добавление записи в таблицу.



Рисунок 8 – Новая запись в таблице организаций

Удаление записей включает в себя запрос требуемых записей для удаления по средством *select* и дальнейшее применение метода *RemoveRange()*, который удалит требуемые записи.

Обновление записей схоже с удаление, однако вместо метода *RemoveRange()* происходит переопределение значений.

Листинг сгенерированных классов представлен в приложении А. А в приложении Б приведен листинг всех *CRUD* опреаций.

**Вывод:** в результате выполнения лабораторной работы было разработано простое консольное приложение с использование *Entity Framework*. Разработанное приложение предоставляет пользователю возможность выполнения *CRUD* операций над созданной ранее базой данных.

*https://github.com/OLagsV/RPBDLabs/tree/main/lab2/OrganizationWaterSupplySol*

**ПРИЛОЖЕНИЕ А**

(обязательное)

***Классы моделей и контекста***

public partial class OrganizationsWaterSupplyContext : DbContext

{

public OrganizationsWaterSupplyContext()

{

}

public OrganizationsWaterSupplyContext(DbContextOptions<OrganizationsWaterSupplyContext> options)

: base(options)

{

}

public virtual DbSet<Counter> Counters { get; set; }

public virtual DbSet<CounterModel> CounterModels { get; set; }

public virtual DbSet<CountersCheck> CountersChecks { get; set; }

public virtual DbSet<CountersDatum> CountersData { get; set; }

public virtual DbSet<Organization> Organizations { get; set; }

public virtual DbSet<Rate> Rates { get; set; }

public virtual DbSet<RateOrg> RateOrgs { get; set; }

public virtual DbSet<ViewOrganizationsCounter> ViewOrganizationsCounters { get; set; }

protected override void OnConfiguring(DbContextOptionsBuilder optionsBuilder)

#warning To protect potentially sensitive information in your connection string, you should move it out of source code. You can avoid scaffolding the connection string by using the Name= syntax to read it from configuration - see https://go.microsoft.com/fwlink/?linkid=2131148. For more guidance on storing connection strings, see http://go.microsoft.com/fwlink/?LinkId=723263.

=> optionsBuilder.UseSqlServer("Server=TRATSEVSKIY\\SQLEXPRESS01;Database=OrganizationsWaterSupply;Trusted\_Connection=True;TrustServerCertificate=True");

protected override void OnModelCreating(ModelBuilder modelBuilder)

{

modelBuilder.Entity<Counter>(entity =>

{

entity.HasKey(e => e.RegistrationNumber).HasName("PK\_\_Counters\_\_E8864603567FE66E");

entity.Property(e => e.ModelId).HasColumnName("ModelID");

entity.Property(e => e.OrganizationId).HasColumnName("OrganizationID");

entity.Property(e => e.TimeOfInstallation).HasColumnType("date");

entity.HasOne(d => d.Model).WithMany(p => p.Counters)

.HasForeignKey(d => d.ModelId)

.OnDelete(DeleteBehavior.Cascade)

.HasConstraintName("FK\_Counters\_CounterModels");

entity.HasOne(d => d.Organization).WithMany(p => p.Counters)

.HasForeignKey(d => d.OrganizationId)

.OnDelete(DeleteBehavior.Cascade)

.HasConstraintName("FK\_Counters\_Organizations");

});

modelBuilder.Entity<CounterModel>(entity =>

{

entity.HasKey(e => e.ModelId).HasName("PK\_\_CounterM\_\_E8D7A1CC12D7DCC6");

entity.Property(e => e.ModelId).HasColumnName("ModelID");

entity.Property(e => e.Manufacturer).HasMaxLength(50);

entity.Property(e => e.ModelName).HasMaxLength(50);

});

modelBuilder.Entity<CountersCheck>(entity =>

{

entity.HasKey(e => e.CountersCheckId).HasName("PK\_\_Counters\_\_49850B5DA52FAC99");

entity.Property(e => e.CountersCheckId).HasColumnName("CountersCheckID");

entity.Property(e => e.CheckDate).HasColumnType("date");

entity.Property(e => e.CheckResult).HasMaxLength(50);

entity.HasOne(d => d.RegistrationNumberNavigation).WithMany(p => p.CountersChecks)

.HasForeignKey(d => d.RegistrationNumber)

.OnDelete(DeleteBehavior.Cascade)

.HasConstraintName("FK\_CountersChecks\_Counters");

});

modelBuilder.Entity<CountersDatum>(entity =>

{

entity.HasKey(e => e.CountersDataId).HasName("PK\_\_Counters\_\_60A1F09B7549BC10");

entity.Property(e => e.CountersDataId).HasColumnName("CountersDataID");

entity.Property(e => e.DataCheckDate).HasColumnType("date");

entity.Property(e => e.RateId).HasColumnName("RateID");

entity.HasOne(d => d.Rate).WithMany(p => p.CountersData)

.HasForeignKey(d => d.RateId)

.HasConstraintName("FK\_CountersData\_RateOrg");

entity.HasOne(d => d.RegistrationNumberNavigation).WithMany(p => p.CountersData)

.HasForeignKey(d => d.RegistrationNumber)

.OnDelete(DeleteBehavior.Cascade)

.HasConstraintName("FK\_CountersData\_Counters");

});

modelBuilder.Entity<Organization>(entity =>

{

entity.HasKey(e => e.OrganizationId).HasName("PK\_\_Organiza\_\_CADB0B72595BF4CA");

entity.Property(e => e.OrganizationId).HasColumnName("OrganizationID");

entity.Property(e => e.Adress).HasMaxLength(50);

entity.Property(e => e.DirectorFullname).HasMaxLength(50);

entity.Property(e => e.DirectorPhone)

.HasMaxLength(11)

.IsUnicode(false);

entity.Property(e => e.OrgName).HasMaxLength(50);

entity.Property(e => e.OwnershipType).HasMaxLength(50);

entity.Property(e => e.ResponsibleFullname).HasMaxLength(50);

entity.Property(e => e.ResponsiblePhone)

.HasMaxLength(11)

.IsUnicode(false);

});

modelBuilder.Entity<Rate>(entity =>

{

entity.HasKey(e => e.RateId).HasName("PK\_\_Rates\_\_58A7CCBC0190846D");

entity.Property(e => e.RateId).HasColumnName("RateID");

entity.Property(e => e.Price).HasColumnType("decimal(18, 2)");

entity.Property(e => e.RateName).HasMaxLength(50);

});

modelBuilder.Entity<RateOrg>(entity =>

{

entity.HasKey(e => e.RateId).HasName("PK\_\_RateOrg\_\_58A7CCBCB3D0241A");

entity.ToTable("RateOrg");

entity.Property(e => e.RateId)

.ValueGeneratedNever()

.HasColumnName("RateID");

entity.Property(e => e.OrganizationId).HasColumnName("OrganizationID");

entity.HasOne(d => d.Organization).WithMany(p => p.RateOrgs)

.HasForeignKey(d => d.OrganizationId)

.OnDelete(DeleteBehavior.Cascade)

.HasConstraintName("FK\_RateOrg\_Organizations");

entity.HasOne(d => d.Rate).WithOne(p => p.RateOrg)

.HasForeignKey<RateOrg>(d => d.RateId)

.HasConstraintName("FK\_RateOrg\_Rates");

});

modelBuilder.Entity<ViewOrganizationsCounter>(entity =>

{

entity

.HasNoKey()

.ToView("View\_OrganizationsCounters");

entity.Property(e => e.Adress).HasMaxLength(50);

entity.Property(e => e.DirectorFullname).HasMaxLength(50);

entity.Property(e => e.DirectorPhone)

.HasMaxLength(11)

.IsUnicode(false);

entity.Property(e => e.Manufacturer).HasMaxLength(50);

entity.Property(e => e.ModelId).HasColumnName("ModelID");

entity.Property(e => e.ModelName).HasMaxLength(50);

entity.Property(e => e.OrgName).HasMaxLength(50);

entity.Property(e => e.OrganizationId).HasColumnName("OrganizationID");

entity.Property(e => e.OwnershipType).HasMaxLength(50);

entity.Property(e => e.ResponsibleFullname).HasMaxLength(50);

entity.Property(e => e.ResponsiblePhone)

.HasMaxLength(11)

.IsUnicode(false);

entity.Property(e => e.TimeOfInstallation).HasColumnType("date");

});

OnModelCreatingPartial(modelBuilder);

}

partial void OnModelCreatingPartial(ModelBuilder modelBuilder);

}

public partial class Organization

{

public int OrganizationId { get; set; }

public string? OrgName { get; set; }

public string? OwnershipType { get; set; }

public string? Adress { get; set; }

public string? DirectorFullname { get; set; }

public string? DirectorPhone { get; set; }

public string? ResponsibleFullname { get; set; }

public string? ResponsiblePhone { get; set; }

public virtual ICollection<Counter> Counters { get; set; } = new List<Counter>();

public virtual ICollection<RateOrg> RateOrgs { get; set; } = new List<RateOrg>();

}

public partial class CountersDatum

{

public int CountersDataId { get; set; }

public int? RegistrationNumber { get; set; }

public DateTime? DataCheckDate { get; set; }

public int? RateId { get; set; }

public int? Volume { get; set; }

public virtual RateOrg? Rate { get; set; }

public virtual Counter? RegistrationNumberNavigation { get; set; }

}

public partial class CountersCheck

{

public int CountersCheckId { get; set; }

public int? RegistrationNumber { get; set; }

public DateTime? CheckDate { get; set; }

public string? CheckResult { get; set; }

public virtual Counter? RegistrationNumberNavigation { get; set; }

}

public partial class CounterModel

{

public int ModelId { get; set; }

public string? ModelName { get; set; }

public string? Manufacturer { get; set; }

public int? ServiceTime { get; set; }

public virtual ICollection<Counter> Counters { get; set; } = new List<Counter>();

}

public partial class Counter

{

public int RegistrationNumber { get; set; }

public int? ModelId { get; set; }

public DateTime? TimeOfInstallation { get; set; }

public int? OrganizationId { get; set; }

public virtual ICollection<CountersCheck> CountersChecks { get; set; } = new List<CountersCheck>();

public virtual ICollection<CountersDatum> CountersData { get; set; } = new List<CountersDatum>();

public virtual CounterModel? Model { get; set; }

public virtual Organization? Organization { get; set; }

}

public partial class Rate

{

public int RateId { get; set; }

public string? RateName { get; set; }

public decimal? Price { get; set; }

public virtual RateOrg? RateOrg { get; set; }

}

public partial class RateOrg

{

public int RateId { get; set; }

public int? OrganizationId { get; set; }

public virtual ICollection<CountersDatum> CountersData { get; set; } = new List<CountersDatum>();

public virtual Organization? Organization { get; set; }

public virtual Rate Rate { get; set; } = null!;

}

**ПРИЛОЖЕНИЕ Б**

(обязательное)

***CRUD* операции**

public static void Main(string[] args)

{

using (OrganizationsWaterSupplyContext db = new OrganizationsWaterSupplyContext())

{

Select(db);

Insert(db);

Delete(db);

Update(db);

}

static void Print(string sqltext, IEnumerable items)

{

Console.WriteLine(sqltext);

Console.WriteLine("Записи: ");

foreach (var item in items)

{

Console.WriteLine(item.ToString());

}

Console.WriteLine();

Console.ReadKey();

}

static void Select(OrganizationsWaterSupplyContext db)

{

var linq1 = from cm in db.CounterModels

select new

{

ModelName = cm.ModelName,

Manufacturer = cm.Manufacturer,

ServiceTime = cm.ServiceTime,

};

string comment1 = "1. Результат выполнения запроса на выборку всех данных из таблицы: \r\n";

Print(comment1, linq1.ToList());

var linq2 = from cm in db.CounterModels

where (cm.ServiceTime > 20)

select new

{

ModelName = cm.ModelName,

Manufacturer = cm.Manufacturer,

ServiceTime = cm.ServiceTime,

};

string comment2 = "2. Результат выполнения запроса на выборку отфильтрованных данных: \r\n";

Print(comment2, linq2.ToList());

var linq3 = from c in db.Counters

join cm in db.CounterModels

on c.ModelId equals cm.ModelId

group c.RegistrationNumber by cm.ServiceTime into gr

select new

{

ModelId = gr.Key,

TimeOfInstallation = gr.Max()

};

string comment3 = "3. Результат выполнения запроса на выборку с группировкой данных из таблицы: \r\n";

Print(comment3, linq3.ToList());

var linq4 = from c in db.Counters

join cm in db.CounterModels

on c.ModelId equals cm.ModelId

select new

{

CounterId = c.RegistrationNumber,

Model = cm.ModelName,

ServiceTime = cm.ServiceTime,

TimeOfIsntallation = c.TimeOfInstallation,

OrganizationId = c.OrganizationId

};

string comment4 = "4. Результат выполнения запроса на выборку полей с двух таблиц: \r\n";

Print(comment4, linq4.ToList());

var linq5 = from c in db.Counters

join cm in db.CounterModels

on c.ModelId equals cm.ModelId

where (cm.ServiceTime > 20 && cm.ServiceTime < 30)

select new

{

CounterId = c.RegistrationNumber,

Model = cm.ModelName,

ServiceTime = cm.ServiceTime,

TimeOfIsntallation = c.TimeOfInstallation,

OrganizationId = c.OrganizationId

};

string comment5 = "5. Результат выполнения запроса на выборку полей с двух таблиц с фильтрацией: \r\n";

Print(comment5, linq5.ToList());

};

static void Insert(OrganizationsWaterSupplyContext db)

{

CounterModel newModel = new CounterModel

{

ModelName = "ModelExample",

Manufacturer = "ManufacturerExample",

ServiceTime = 40

};

Organization newOrg = new Organization

{

OrgName = "OrgExample",

OwnershipType = "Ownership",

Adress = "Adress",

DirectorFullname = "Director",

DirectorPhone = "+375123456",

ResponsibleFullname = "Responsible",

ResponsiblePhone = "+3754412341"

};

db.CounterModels.Add(newModel);

db.Organizations.Add(newOrg);

db.SaveChanges();

Counter newCounter = new Counter

{

ModelId = newModel.ModelId,

TimeOfInstallation = new DateTime(2020,12,24),

OrganizationId = newOrg.OrganizationId

};

db.Counters.Add(newCounter);

db.SaveChanges();

}

static void Delete(OrganizationsWaterSupplyContext db)

{

string namemanufact = "ManufacturerToDelete";

var manufact = db.CounterModels.Where(cm => cm.Manufacturer == namemanufact);

string nameorg = "OrgToDelete";

var org = db.Organizations

.Where(c => c.OrgName == nameorg);

var someCounters = db.Counters

.Include("Organization")

.Include("Model")

.Where(o => ((o.Organization.OrgName == nameorg)) && (o.Model.Manufacturer == namemanufact));

db.Counters.RemoveRange(someCounters);

db.SaveChanges();

db.CounterModels.RemoveRange(manufact);

db.Organizations.RemoveRange(org);

db.SaveChanges();

}

static void Update(OrganizationsWaterSupplyContext db)

{

var oldCounter = db.Counters.Where(c => (c.RegistrationNumber == 6));

//обновление

if (oldCounter != null)

{

foreach (var c in oldCounter)

{

Counter counter = (Counter)c;

counter.TimeOfInstallation = new DateTime(2030, 01, 01);

};

}

db.SaveChanges();

}

}