**..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 № 28

Выполнил:

студент гр. ИТП-31, Трацевский И. С.

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.

3.3. Разместить выполненный проект на *github*.

**Ход работы:** для работы с разработанной ранее базой данных необходимо иметь соответствующие классы, представляющие содержимое базы данных. Так как база данных была разработана в предыдущей лабораторной работе, то нет необходимости создавать классы таблиц вручную. Для автоматической генерации классов и контекста данных можно воспользоваться методом *Scaffolding*. Данная строка позволит подключится к нужно базе данных и создать все необходимые классы:

*Scaffold-DbContext "Server= TRATSEVSKIY\SQLEXPRESS01 ;Database=helloappdb;Trusted\_Connection=True;" Microsoft.EntityFrameworkCore.SqlServer*

Для любых операций с таблицами в базе данных необходим так называемый контекст данных. Данный класс обеспечивает связь программы и базы данных. Для использования этого класса, можно использовать блок *using(),* передав контекст в качестве параметра.

Далее можно создать функции, выполняющие *LINQ*-запросы. Первый запрос включает всего одну операцию: *select*. Для получения данных необходимо перечислить извлекаемые поля.

Второй запрос содержит условие для отбора данных. Данное условие задается при помощи слова *where.*

Для запроса к нескольким таблицам необходимо использовать оператор *join*, далее при помощи *on* указать по какому столбцу происходит связь, например по полю *id*.Таким же образом создается и следующий запрос на два поля из двух таблиц.

Фильтрация данных из нескольких таблиц выполняется тем же образом, как и для одной таблицы. Отличительной чертой для фильтрации данных нескольких таблиц служит требование указывать названия таблиц, из которых берутся столбцы.

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.

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.

В приложении А показаны результаты *LINQ* запросов в консоли. В приложении Б показаны классы базы данных, полученных при помощи метода *Scaffoldind*. В приложении В показаны сами *LINQ* запросы.

**Вывод:** в результате выполнения лабораторной работы было разработано консольное приложение с использование *Entity Framework*. При помощи *LINQ* запросов реализованы *CRUD* операции. Результат работы программы пользователь может увидеть в консоли.

***GitHub:*** https://github.com/IvanTratsevski/DBLABS/tree/main/lab2/TaxiGomel

**ПРИЛОЖЕНИЕ А**

**(обязательное)**

Результат работы запросов

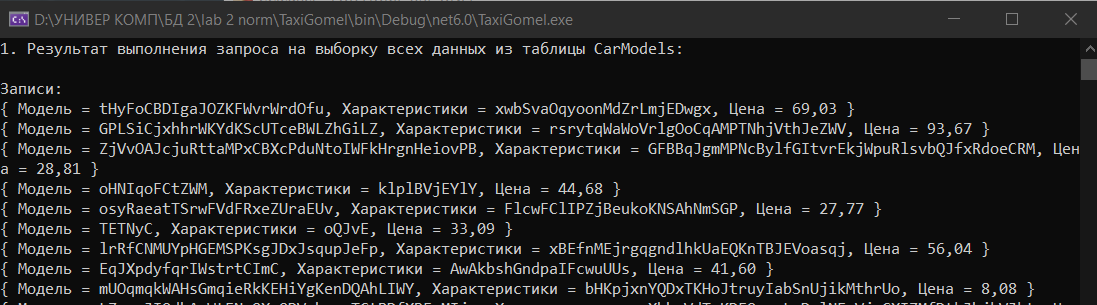


Рисунок А.1 – Выборка всех данных

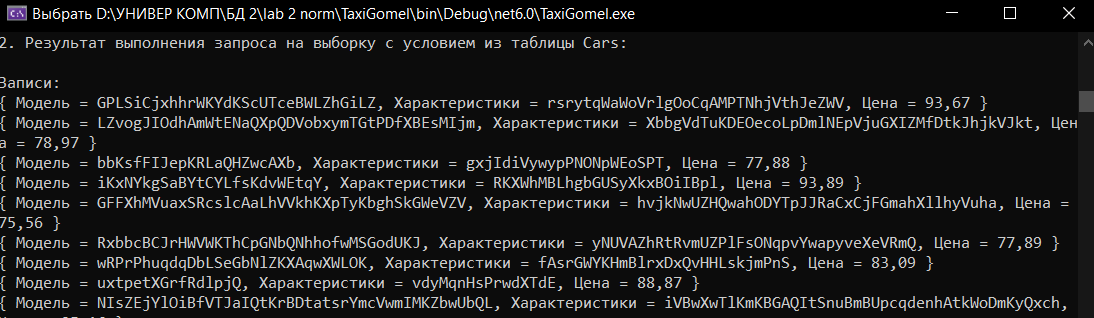


Рисунок А.2 – Выборка с условием

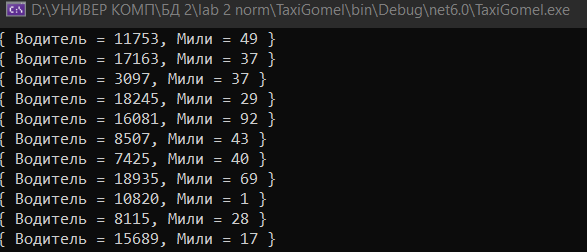


Рисунок А.3 – Выборка с группировкой

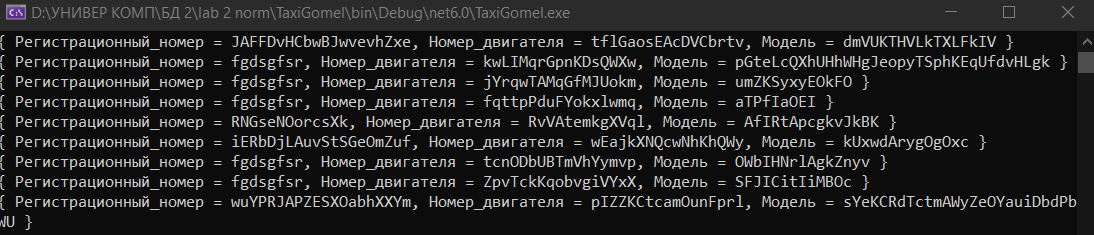
****

Рисунок А.4 – Запрос к нескольким таблицам

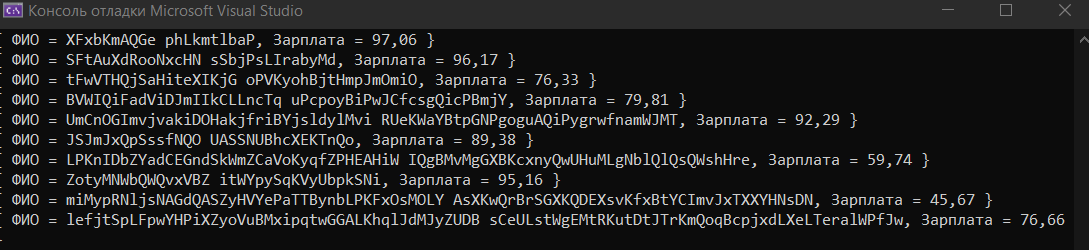
****

Рисунок А.5 – Запрос к двум таблицам с фильтрацией

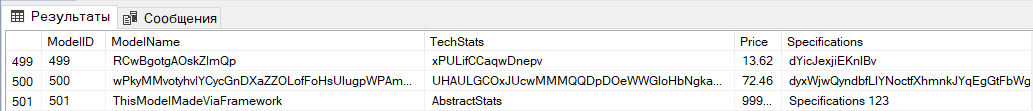


Рисунок А.6 – Создание записи в таблице

**ПРИЛОЖЕНИЕ Б**

**(обязательное)**

Классы моделей и контекста

public partial class TaxiGomelContext : DbContext

{

public TaxiGomelContext()

{

}

public TaxiGomelContext(DbContextOptions<TaxiGomelContext> options)

: base(options)

{

}

public virtual DbSet<Call> Calls { get; set; }

public virtual DbSet<Car> Cars { get; set; }

public virtual DbSet<CarModel> CarModels { get; set; }

public virtual DbSet<Employee> Employees { get; set; }

public virtual DbSet<Position> Positions { get; set; }

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

public virtual DbSet<View1> View1s { get; set; }

public virtual DbSet<View2> View2s { get; set; }

public virtual DbSet<View3> View3s { get; set; }

public virtual DbSet<ViewCarsAndDriver> ViewCarsAndDrivers { get; set; }

public virtual DbSet<ViewEmployeeAndPosition> ViewEmployeeAndPositions { 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=TaxiGomel;Trusted\_Connection=True; TrustServerCertificate=True;");

protected override void OnModelCreating(ModelBuilder modelBuilder)

{

modelBuilder.Entity<Call>(entity =>

{

entity.HasKey(e => e.CallId).HasName("PK\_\_Calls\_\_5180CF8AEE207507");

entity.Property(e => e.CallId).HasColumnName("CallID");

entity.Property(e => e.CallTime).HasColumnType("smalldatetime");

entity.Property(e => e.EndPosition).HasMaxLength(20);

entity.Property(e => e.StartPosition).HasMaxLength(20);

entity.Property(e => e.Telephone)

.HasMaxLength(12)

.IsUnicode(false);

entity.HasOne(d => d.Car).WithMany(p => p.Calls)

.HasForeignKey(d => d.CarId)

.OnDelete(DeleteBehavior.Cascade)

.HasConstraintName("FK\_Calls\_Cars");

entity.HasOne(d => d.Dispatcher).WithMany(p => p.Calls)

.HasForeignKey(d => d.DispatcherId)

.HasConstraintName("FK\_Calls\_Employees");

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

.HasForeignKey(d => d.RateId)

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

});

modelBuilder.Entity<Car>(entity =>

{

entity.HasKey(e => e.CarId).HasName("PK\_\_Cars\_\_68A0340E69AA6E02");

entity.Property(e => e.CarId).HasColumnName("CarID");

entity.Property(e => e.CarcaseNumber).HasMaxLength(17);

entity.Property(e => e.EngineNumber).HasMaxLength(17);

entity.Property(e => e.LastTi)

.HasColumnType("date")

.HasColumnName("LastTI");

entity.Property(e => e.RegistrationNumber).HasMaxLength(20);

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

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

entity.HasOne(d => d.Driver).WithMany(p => p.CarDrivers)

.HasForeignKey(d => d.DriverId)

.OnDelete(DeleteBehavior.Cascade)

.HasConstraintName("FK\_Cars\_EmployeesD");

entity.HasOne(d => d.Mechanic).WithMany(p => p.CarMechanics)

.HasForeignKey(d => d.MechanicId)

.HasConstraintName("FK\_Cars\_EmployeesM");

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

.HasForeignKey(d => d.ModelId)

.OnDelete(DeleteBehavior.Cascade)

.HasConstraintName("FK\_Cars\_CarModels");

});

modelBuilder.Entity<CarModel>(entity =>

{

entity.HasKey(e => e.ModelId).HasName("PK\_\_CarModel\_\_E8D7A1CCB9A702FD");

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

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

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

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

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

});

modelBuilder.Entity<Employee>(entity =>

{

entity.HasKey(e => e.EmployeeId).HasName("PK\_\_Employee\_\_7AD04FF136E2FF56");

entity.Property(e => e.EmployeeId).HasColumnName("EmployeeID");

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

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

entity.Property(e => e.PositionId).HasColumnName("PositionID");

entity.HasOne(d => d.Position).WithMany(p => p.Employees)

.HasForeignKey(d => d.PositionId)

.OnDelete(DeleteBehavior.Cascade)

.HasConstraintName("FK\_Employees\_Positions");

});

modelBuilder.Entity<Position>(entity =>

{

entity.HasKey(e => e.PositionId).HasName("PK\_\_Position\_\_60BB9A599819E018");

entity.Property(e => e.PositionId).HasColumnName("PositionID");

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

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

});

modelBuilder.Entity<Rate>(entity =>

{

entity.HasKey(e => e.RateId).HasName("PK\_\_Rates\_\_58A7CCBCD3C7663C");

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

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

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

});

modelBuilder.Entity<View1>(entity =>

{

entity

.HasNoKey()

.ToView("View\_1");

entity.Property(e => e.CarcaseNumber).HasMaxLength(17);

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

});

modelBuilder.Entity<View2>(entity =>

{

entity

.HasNoKey()

.ToView("View\_2");

entity.Property(e => e.RegistrationNumber).HasMaxLength(20);

});

modelBuilder.Entity<View3>(entity =>

{

entity

.HasNoKey()

.ToView("View\_3");

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

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

});

modelBuilder.Entity<ViewCarsAndDriver>(entity =>

{

entity

.HasNoKey()

.ToView("View\_CarsAndDrivers");

entity.Property(e => e.CarId).HasColumnName("CarID");

entity.Property(e => e.Driver).HasMaxLength(101);

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

entity.Property(e => e.RegistrationNumber).HasMaxLength(20);

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

});

modelBuilder.Entity<ViewEmployeeAndPosition>(entity =>

{

entity

.HasNoKey()

.ToView("View\_EmployeeAndPositions");

entity.Property(e => e.EmployeeId).HasColumnName("EmployeeID");

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

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

entity.Property(e => e.PositionId).HasColumnName("PositionID");

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

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

});

OnModelCreatingPartial(modelBuilder);

}

partial void OnModelCreatingPartial(ModelBuilder modelBuilder);

}

public partial class Call

{

public int CallId { get; set; }

public DateTime? CallTime { get; set; }

public string? Telephone { get; set; }

public string? StartPosition { get; set; }

public string? EndPosition { get; set; }

public int? RateId { get; set; }

public int? CarId { get; set; }

public int? DispatcherId { get; set; }

public virtual Car? Car { get; set; }

public virtual Employee? Dispatcher { get; set; }

public virtual Rate? Rate { get; set; }

}

public partial class Car

{

public int CarId { get; set; }

public string? RegistrationNumber { get; set; }

public int? ModelId { get; set; }

public string? CarcaseNumber { get; set; }

public string? EngineNumber { get; set; }

public DateTime? ReleaseYear { get; set; }

public int? Mileage { get; set; }

public int? DriverId { get; set; }

public DateTime? LastTi { get; set; }

public int? MechanicId { get; set; }

public string? SpecialMarks { get; set; }

public virtual ICollection<Call> Calls { get; set; } = new List<Call>();

public virtual Employee? Driver { get; set; }

public virtual Employee? Mechanic { get; set; }

public virtual CarModel? Model { get; set; }

}

public partial class CarModel

{

public int ModelId { get; set; }

public string? ModelName { get; set; }

public string? TechStats { get; set; }

public decimal? Price { get; set; }

public string? Specifications { get; set; }

public virtual ICollection<Car> Cars { get; set; } = new List<Car>();

}

public partial class Employee

{

public int EmployeeId { get; set; }

public string? FirstName { get; set; }

public string? LastName { get; set; }

public int? Age { get; set; }

public int? PositionId { get; set; }

public int? Experience { get; set; }

public virtual ICollection<Call> Calls { get; set; } = new List<Call>();

public virtual ICollection<Car> CarDrivers { get; set; } = new List<Car>();

public virtual ICollection<Car> CarMechanics { get; set; } = new List<Car>();

public virtual Position? Position { get; set; }

}

public partial class Position

{

public int PositionId { get; set; }

public string? PositionName { get; set; }

public decimal? Salary { get; set; }

public virtual ICollection<Employee> Employees { get; set; } = new List<Employee>();

}

public partial class Rate

{

public int RateId { get; set; }

public string? RateDescription { get; set; }

public decimal? RatePrice { get; set; }

public virtual ICollection<Call> Calls { get; set; } = new List<Call>();

}

**ПРИЛОЖЕНИЕ В**

**(обязательное)**

*LINQ* запросы

namespace TaxiGomel

{

public class Program

{

public static void Main(string[] args)

{

using (TaxiGomelContext db = new TaxiGomelContext())

{

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(TaxiGomelContext db)

{

var carModelsLINQ = from cm in db.CarModels

select new

{

Модель = cm.ModelName,

Характеристики = cm.TechStats,

Цена = cm.Price,

};

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

Print(comment1, carModelsLINQ.ToList());

var carModelsSortLINQ = from cm in db.CarModels

where (cm.Price > 70)

select new

{

Модель = cm.ModelName,

Характеристики = cm.TechStats,

Цена = cm.Price,

};

string comment2 = "2. Результат выполнения запроса на выборку с условием из таблицы Cars: \r\n";

Print(comment2, carModelsSortLINQ.ToList());

var groupCatsLINQ = from c in db.Cars

group c.Mileage by c.DriverId into gr

select new

{

Водитель = gr.Key,

Мили = gr.Sum(),

};

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

Print(comment3, groupCatsLINQ.ToList());

var multiSelectLINQ = from c in db.Cars

join m in db.CarModels

on c.ModelId equals m.ModelId

select new

{

Регистрационный\_номер = c.RegistrationNumber,

Номер\_двигателя = c.EngineNumber,

Модель = m.ModelName,

};

string comment4 = "4. Результат выполнения запроса к нескольким таблицам (Cars, CarsModels): \r\n";

Print(comment4, multiSelectLINQ.ToList());

var salaryEmployeeLINQ = from e in db.Employees

join p in db.Positions

on e.PositionId equals p.PositionId

where (p.Salary > 30)

select new

{

ФИО = e.FirstName + " " + e.LastName,

Зарплата = p.Salary

};

string comment5 = "5. Результат выполнения запроса к нескольким таблицам с отбором (Employees, Positions): \r\n";

Print(comment5, salaryEmployeeLINQ.ToList());

};

static void Insert(TaxiGomelContext db)

{

CarModel newCarModel = new CarModel

{

ModelName = "ThisModelMadeViaFramework",

TechStats = "AbstractStats",

Price = 999,

Specifications = "Specifications 123"

};

db.CarModels.Add(newCarModel);

db.SaveChanges();

Car newCar = new Car

{

RegistrationNumber = "842FE425",

ModelId = newCarModel.ModelId,

CarcaseNumber = "7742GH32-23",

EngineNumber = "97VHH24-123F",

ReleaseYear = new DateTime(2014, 9, 10),

Mileage = 999,

DriverId = 18,

LastTi = new DateTime(2022,11,6),

MechanicId = 19,

SpecialMarks = "No marks",

};

db.Cars.Add(newCar);

db.SaveChanges();

}

static void Delete(TaxiGomelContext db)

{

int delPosition = 76;

var pos = db.Positions.Where(p => p.PositionId == delPosition);

var delEmployees = db.Employees

.Include("Position")

.Where(e => e.PositionId == delPosition);

db.Employees.RemoveRange(delEmployees);

db.SaveChanges();

db.Positions.RemoveRange(pos);

db.SaveChanges();

}

static void Update(TaxiGomelContext db)

{

var carsOutOfOrder = db.Cars.Where(c => (c.Mileage > 10000));

if (carsOutOfOrder != null)

{

foreach (var c in carsOutOfOrder)

{

c.SpecialMarks = "Old";

};

}

db.SaveChanges();

}

}

}

}