**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 № 4

«Разработка моделей и контроллеров ASP.NET MVC приложения баз данных.»

ВАРИАНТ № 28

Выполнил: студент гр. ИТП-31

Трацевский И. С.

Принял:доцент

Асенчик О. Д.

Гомель 2023

**Цель работы:** ознакомиться с возможностями *ASP.NET Core MVC* и *Entity Framework Core* для разработки слоя доступа к данным, хранящимся в базе данных, и обработки запросов пользователя посредством контроллеров.

**Задание:** создать с использованием *ASP.NET Core MVC Web*-приложение, содержащее набор классов, моделирующих предметную область, и осуществляющих генерацию и заполнение тестовыми наборами записей базу данных. Разработать один компонент *middleware*, контроллеры и представления для выборки и отображения информации из не менее чем 3- таблиц базы данных с использованием механизма внедрение зависимостей.

Для выполнения задания необходимо создать:

• Классы, моделирующие не менее чем три таблицы базы данных согласно вашему варианту. Перечень таблиц предварительно согласовывается с преподавателем. Одна из таблиц обязательно должна находиться на стороне отношения «многие» связи с другой таблицей в схеме базы данных.

• Класс контекста данных.

• Другие классы, например, классы *View Model* и т.п. (при необходимости).

• Компонент *middleware*, вызываемый в классе *Startup*, для инициализации базы данных путем заполнения ее таблиц тестовым набором записей.

• Классы контроллеров (по одному на каждую таблицу базы данных) для обработки обращений пользователя, выборки данных из таблиц и вызова соответствующих представлений для отображения выбранных данных.

• Разработать представления для отображения данных из таблиц, выбранных контроллерами. Представления, работающими с таблицами, стоящими на стороне отношения «многие» в схеме базы данных, должны выводить вместо кодов внешних ключей смысловые значения из связанных таблиц, стоящих на стороне отношения «один».

• Используя предварительно созданный и сконфигурированный в классе *Startup* профиль кэширования, подключить кэширование вывода для страниц с использованием атрибута *ResponseCache* для соответствующих методов контроллера. Данные в кэше хранить неизменными в течение 2\**N*+240 секунд, где N- номер вашего варианта.

**Ход работы:** при выполнении данной лабораторной работы были внесены изменения в структуру уже созданной ранее базы данных, были добавлены связующие таблицы *CarsMechanics* и *CarsDrivers.* Данные изменения позволили улучшить качество связей между таблицами. На рисунке 1 продемонстрирована структура обновленной базы данных.

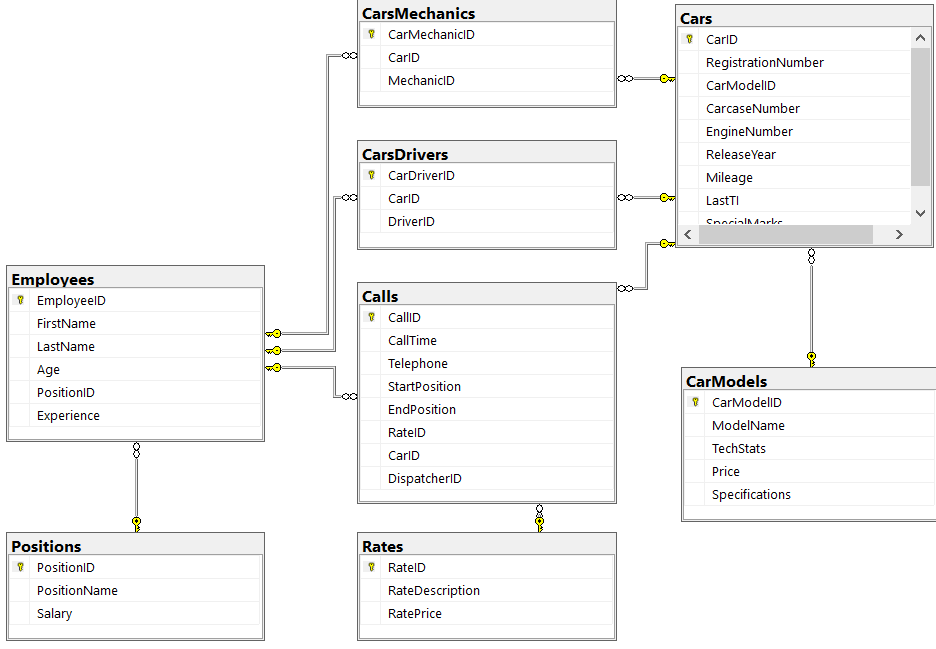


Рисунок 1 – Структура новой базы данных

Стоит отметить, что помимо создания дополнительных таблиц, были добавлены триггеры, которые не позволят записать на место механиков сотрудников с должностью водителя и т.д.

Далее при помощи *scaffolding* были сгенерированы классы моделей и контекста данных. Дальнейшее выполнение лабораторной работы заключалось в написании представлений и контроллеров, для работы с моделями. В программе были реализованы операции чтения, добавления и редактирования данных. И на рисунке 2 продемонстрирована страница, отображающая данные из базы данных. Для ускорения загрузки данных были разработаны сервисы кэширования. Поэтому при перезагрузке страницы данные загружаются намного быстрее в сравнении с первым запуском.

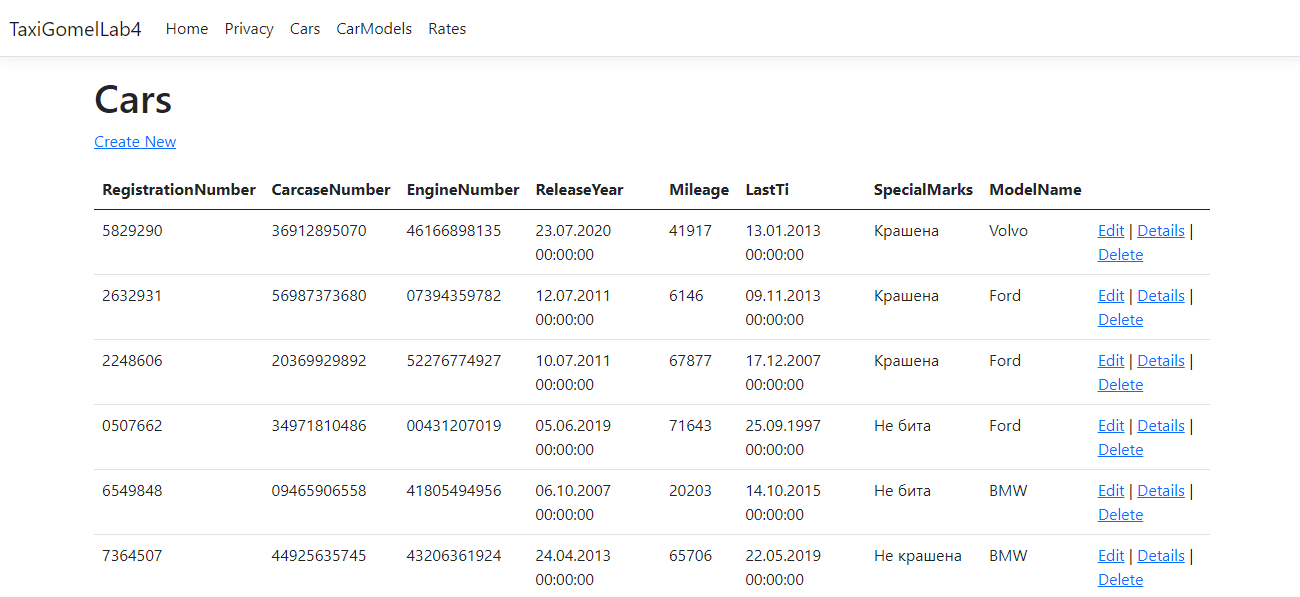


Рисунок 2 – Таблица машин

Для заполнения данных был разработан *middleware* генератор данных, в котором учтены особенности ограничений базы данных. В приложении А представлен генератор данных.

**Github:** https://github.com/IvanTratsevski/TaxiGomelLab4

**Вывод:** результатом выполнения лабораторной работы стало *ASP.NET Core MVC Web*-приложение, содержащее набор классов, моделирующих предметную область, и осуществляющих генерацию и заполнение тестовыми наборами записей базу данных с помощью *Entity Framework*.

**ПРИЛОЖЕНИЕ А**

**(обязательное)**

Листинг *DbInitializer*

using System;

using System.Diagnostics;

using System.Diagnostics.Metrics;

using System.Drawing.Text;

using TaxiGomelLab4.Data;

using TaxiGomelLab4.Models;

namespace TaxiGomelLab4.Middleware

{

public class DbInitializer

{

public static void Initialize(TaxiGomelContext db)

{

db.Database.EnsureCreated();

int single\_link = 50;

int many\_link = 100;

Random random = new Random(1);

string[] car\_models = { "BMW", "Toyota", "Honda", "Lada", "Ford", "Volvo", "Skoda" };

string[] tech\_stats = { "Объем бака", "Время разгона", "Мощность", "Ход поршня", "Привод" };

string[] specifications = { "M5", "Granta", "M2", "Gran Coupe", "Aspire", "8" };

string[] first\_names\_m = { "Кирил", "Михаил", "Андрей", "Александр", "Иван", "Олег" };

string[] last\_names\_m = { "Нарыжкин", "Цаль", "Попов", "Сидоров", "Тихонов", "Гордей" };

string[] first\_names\_fm = { "Ольга", "Виктория", "Оксана", "Светалана", "Наталья" };

string[] last\_names\_fm = { "Нарыжкина", "Попова", "Сидорова", "Тихонова", "Мацкевич" };

string[] streets = { "Советская", "Героев-Подпольщиков", "Рябиновая", "Школьная", "Жукова" };

string[] special\_marks = { "Бита", "Крашена", "Не бита", "Не крашена" };

string[] rates = { "Эконом", "Комфорт", "Детский", "Экспресс", "Стандарт" };

string[] GetRandomFullName()

{

if (random.Next(0, 2) == 1)

{

string[] result ={ first\_names\_m[random.Next(0, first\_names\_m.Length)], last\_names\_m[random.Next(0, last\_names\_m.Length)] };

return result;

} else

{

string[] result = { first\_names\_fm[random.Next(0, first\_names\_fm.Length)], last\_names\_fm[random.Next(0, last\_names\_fm.Length)] };

return result;

}

}

string GetRandomNumbers(int length)

{

var random = new Random();

string s = string.Empty;

for (int i = 0; i < length; i++)

s = String.Concat(s, random.Next(10).ToString());

return s;

}

DateTime GetRandomDate()

{

DateTime start = new DateTime(1995, 1, 1);

int range = (DateTime.Today - start).Days;

return start.AddDays(random.Next(range));

}

string GetRandomAdress()

{

return streets[random.Next(0, streets.Length)] + random.Next(0, 70).ToString();

}

if (!db.CarModels.Any())

{

for (int i = 0; i < single\_link; i++)

{

db.CarModels.Add(new CarModel

{

ModelName = car\_models[random.Next(0, car\_models.Length)],

TechStats = tech\_stats[random.Next(0, tech\_stats.Length)],

Price = random.Next(800, 4000),

Specifications = specifications[random.Next(0, specifications.Length)]

});

}

db.SaveChanges();

}

if (!db.Positions.Any())

{

db.Positions.Add(new Position

{

PositionName = "Водитель",

Salary = 700

});

db.Positions.Add(new Position

{

PositionName = "Механик",

Salary = 1100

});

db.Positions.Add(new Position

{

PositionName = "Диспетчер",

Salary = 650

});

db.SaveChanges();

}

if (!db.Rates.Any())

{

for (int i = 0; i < single\_link; i++)

{

db.Rates.Add(new Rate

{

RateDescription = rates[random.Next(0, rates.Length)],

RatePrice = random.Next(1, 6)

});

}

db.SaveChanges();

}

if (!db.Employees.Any())

{

string[] rndfullname\_driver = GetRandomFullName();

// создание трех содрудников чтобы гарантиовано иметь как минимум по однмоу сотруднику каждой специальности

//это необходимо т.к. в базе данных существую триггеры ограничения которые не позволят записать например механика на пост диспетчера

db.Employees.Add(new Employee

{

FirstName = rndfullname\_driver[0],

LastName = rndfullname\_driver[1],

Age = random.Next(18, 100),

PositionId = 1,

Experience = random.Next(0, 100)

});

string[] rndfullname\_mechanic = GetRandomFullName();

db.Employees.Add(new Employee

{

FirstName = rndfullname\_mechanic[0],

LastName = rndfullname\_mechanic[1],

Age = random.Next(18, 100),

PositionId = 2,

Experience = random.Next(0, 100)

});

string[] rndfullname\_dispatcher = GetRandomFullName();

db.Employees.Add(new Employee

{

FirstName = rndfullname\_dispatcher[0],

LastName = rndfullname\_dispatcher[1],

Age = random.Next(18, 100),

PositionId = 3,

Experience = random.Next(0, 100)

});

string[] rndfullname;

for (int i = 0; i < many\_link; i++)

{

rndfullname = GetRandomFullName();

db.Employees.Add(new Employee

{

FirstName = rndfullname[0],

LastName = rndfullname[1],

Age = random.Next(18, 100),

PositionId = random.Next(1, 4),

Experience = random.Next(0, 100)

});

}

db.SaveChanges();

}

if (!db.Cars.Any())

{

for (int i = 0; i < many\_link; i++)

{

db.Cars.Add(new Car

{

RegistrationNumber = GetRandomNumbers(7),

CarModelId = random.Next(1, single\_link),

CarcaseNumber = GetRandomNumbers(11),

EngineNumber = GetRandomNumbers(11),

ReleaseYear = GetRandomDate(),

Mileage = random.Next(0, 100000),

LastTi = GetRandomDate(),

SpecialMarks = special\_marks[random.Next(0, special\_marks.Length)]

});

}

db.SaveChanges();

}

//Я частично разобрался почему не добавлялись записи, дело было в том что при заполнении одной из табли CarsDrivers или CarsMechanics

//вторую таблицу невозможно заполнить, однако я так и не смог выяснить в чем конфликт

if (!db.CarDrivers.Any())

{

for (int i = 0; i < many\_link; i++)

{

db.CarDrivers.Add(new CarDriver

{

CarId = random.Next(1, many\_link),

DriverId = 1,

});

}

db.SaveChanges();

}

if (!db.Calls.Any())

{

//Пробелма состоит в том, что при попытке указать значени полей Call, они считываются как null

//т.е. я пытаюсь записать например в поле Telephone некое значение,

//но при отправке запроса выдается ошибка, что невозможно записать в поле Telephone значение null

//каким-то образом строка с данными не отправляется в запросе

//при чем DispatcherId отправляется корректно

string telephone = "1241424";

for (int i = 0; i < many\_link; i++)

{

db.Calls.Add(new Call

{

Telephone = telephone,

RateId=1,CarId=1,

DispatcherId=3

});

}

db.SaveChanges();

}

if (!db.CarMechanics.Any())

{

for (int i = 0; i < many\_link; i++)

{

db.CarMechanics.Add(new CarMechanic

{

CarId = random.Next(1, many\_link),

MechanicId = 2

});

}

db.SaveChanges();

}

}

}

}