**Отчёт по лабораторной работе №6**

Выполнил студент группы ИВТ(ВМК)-17-2:

Ковалёв Владимир Сергеевич

**Цель работы**

Создание приложения для работы с БД PostgreSQL

**Содержание работы**

* Описание принципа работы Entity Framework и NpqSQL для приложений на C#;
* Реализация приложения для работы с БД.

**Ход работы**

Предметная область:

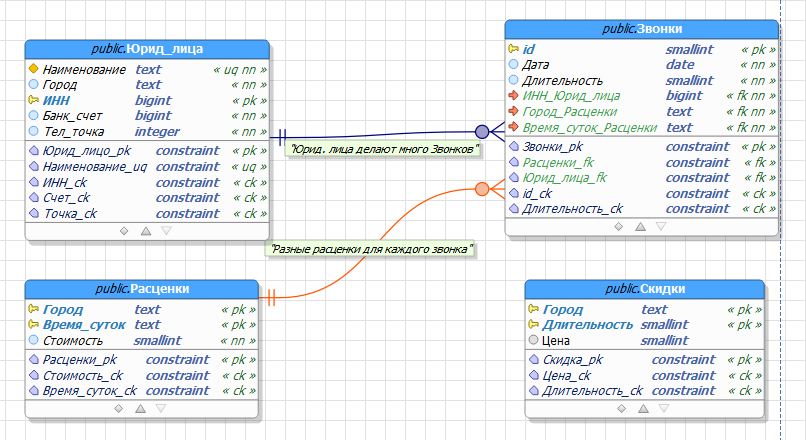
Вы работаете в коммерческой службе телефонной компании. Компания предоставляет абонентам телефонные линии для междугородних переговоров. Вашей задачей является отслеживание стоимости междугородних телефонных переговоров. Абонентами компании являются юридические лица, имеющие телефонную точку, ИНН, расчетный счет в банке. Стоимость переговоров зависит от города, в который осуществляется звонок, и времени суток (день, ночь). Каждый звонок абонента автоматически фиксируется в базе данных. При этом запоминаются город, дата, длительность разговора и время суток. Фирма вводит гибкую систему скидок. Так, стоимость минуты теперь уменьшается в зависимости от длительности разговора. Размер скидки для каждого города разный.

Рисунок 1. Диаграмма ER

|  |
| --- |
| Таблица 1. Юридические лица |
| Таблица 2. Расценки |
| Таблица 3. Звонки |
| Таблица 4. Скидки |

Для создания стороннего приложения для работы с БД PostgreSQL мы будем использовать язык программирования высокого уровня C# вместе с инструментом Entity Framework Core и Npqsql. Версия платформы .NET - .NET Core версии 3.x.

Entity Framework Core (EF Core) представляет собой объектно-ориентированную, легковесную и расширяемую технологию от компании Microsoft для доступа к данным. EF Core является ORM-инструментом (object-relational mapping - отображения данных на реальные объекты). То есть EF Core позволяет работать базами данных, но представляет собой более высокий уровень абстракции: EF Core позволяет абстрагироваться от самой базы данных и ее таблиц и работать с данными независимо от типа хранилища. Если на физическом уровне мы оперируем таблицами, индексами, первичными и внешними ключами, но на концептуальном уровне, который нам предлагает Entity Framework, мы уже работаем с объектами.

Отличительной чертой Entity Framework Core, как технологии ORM, является использование запросов LINQ для выборки данных из БД. С помощью LINQ мы можем создавать различные запросы на выборку объектов, в том числе связанных различными ассоциативными связями. А Entity Framework при выполнении запроса транслирует выражения LINQ в выражения, понятные для конкретной СУБД (как правило, в выражения SQL).

Entity Framework Core поддерживает множество различных систем баз данных. Таким образом, мы можем через EF Core работать с любой СУБД, если для нее имеется нужный провайдер. Для работы с PostgreSQL мы будем использовать провайдер Npqsql для EF Core. Также будем использовать язык интегрированных запросов Linq, встроенный в язык C#. Далее все методы и классы EF Core и Npgsql в тексте будут выделены жирным шрифтом.

Сам EF Core, Npqsql и дополнение EF Core Tools можно скачать через менеджер пакетов NuGet. После скачивания в зависимостях проекта должны появиться следующие элементы:

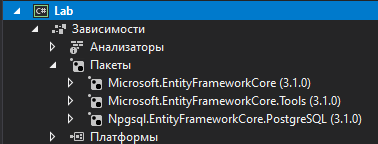


Рисунок 1. Зависимости проекта

Entity Framework предполагает три возможных способа взаимодействия с базой данных:

* **Database first**: Entity Framework создает набор классов, которые отражают модель конкретной базы данных
* **Model first**: сначала разработчик создает модель базы данных, по которой затем Entity Framework создает реальную базу данных на сервере.
* **Code first**: разработчик создает класс модели данных, которые будут храниться в бд, а затем Entity Framework по этой модели генерирует базу данных и ее таблицы

Для взаимодействия с БД мы будем использовать Database first способ. Для этого через консоль диспетчера пакетов (Средства / Диспетчер пакетов Nuget в меню Visual Studio) мы введём специальную команду, которая автоматически создаст специальные классы на основе уже созданных в PostgreSQL таблиц и связей:

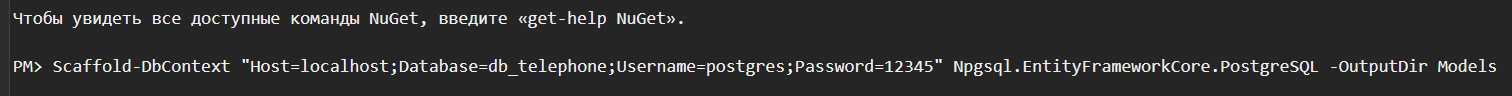


Рисунок 2. Команда Scaffold-DbContext

В кавычках мы указываем url-адрес СУБД, название базы данных, имя пользователя и пароль. Далее в данном случае указываем провайдера Npgsql для работы с PostgreSQL. Опционально можно указать, в какой папке создать классы.

После выполнения в папке Models появятся классы, описывающие таблицы в базе данных, а также создастся класс «TelephoneContext», наследуемый от **DbContext**. Данный класс определяет контекст данных, используемый для взаимодействия с базой данных. В нём будут описаны связи между таблицами из БД, набор объектов, которые хранятся в БД (**DbSet<T>**), а также информация для подключения к БД.

Покажем созданные классами (моделями), которые будут описывать данные, и соответствующие таблицы базы данных:

|  |  |
| --- | --- |
|  | public partial class Звонки  {  public short Id { get; set; }  public DateTime Дата { get; set; }  public short Длительность { get; set; }  public long Инн { get; set; }  public string Город { get; set; }  public string ВремяСуток { get; set; }  public virtual ЮридЛица ИннNavigation { get; set; }  public virtual Расценки Расценки { get; set; }  } |
|  | public partial class ЮридЛица  {  public ЮридЛица()  {  Звонки = new HashSet<Звонки>();  }  public string Наименование { get; set; }  public string Город { get; set; }  public long Инн { get; set; }  public long БанкСчет { get; set; }  public int ТелТочка { get; set; }  public virtual ICollection<Звонки> Звонки { get; set; }  } |
|  | public partial class Расценки  {  public Расценки()  {  Звонки = new HashSet<Звонки>();  }  public string Город { get; set; }  public string ВремяСуток { get; set; }  public short Стоимость { get; set; }  public virtual ICollection<Звонки> Звонки { get; set; }  } |
|  | public partial class Скидки  {  public string Город { get; set; }  public short Длительность { get; set; }  public short Цена { get; set; }  } |

Важно подметить, что для описания связи «один-ко-многим» в классах «ЮридЛица», «Расценки» создаются коллекции типа ICollection, а в классе «Звонки» - ссылки на вышеописанные классы.

В классе «TelephoneContext» в перезаписанном методе **«OnModelCreating»** будут описаны такие свойства классов, как внешние ключи, первичные ключи и связи между классами. Далее приведён код с использованием EF Core и Linq:

**protected** **override** **void** OnModelCreating(ModelBuilder modelBuilder)

{

modelBuilder.Entity<Звонки>(entity =>

{

entity.Property(e => e.Id)

.HasColumnName("id")

.ValueGeneratedNever();

entity.Property(e => e.ВремяСуток)

.IsRequired()

.HasColumnName("Время\_суток");

entity.Property(e => e.Город).IsRequired();

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

entity.Property(e => e.Инн).HasColumnName("ИНН");

entity.HasOne(d => d.ИннNavigation)

.WithMany(p => p.Звонки)

.HasForeignKey(d => d.Инн)

.HasConstraintName("Юрид\_лица\_fk");

entity.HasOne(d => d.Расценки)

.WithMany(p => p.Звонки)

.HasForeignKey(d => new { d.Город, d.ВремяСуток })

.OnDelete(DeleteBehavior.ClientSetNull)

.HasConstraintName("Расценки\_fk");

});

modelBuilder.Entity<Расценки>(entity =>

{

entity.HasKey(e => new { e.Город, e.ВремяСуток })

.HasName("Расценки\_pk");

entity.Property(e => e.Город).HasDefaultValueSql("'Неизвестно'::text");

entity.Property(e => e.ВремяСуток)

.HasColumnName("Время\_суток")

.HasDefaultValueSql("'день'::text");

entity.Property(e => e.Стоимость).HasDefaultValueSql("10");

});

modelBuilder.Entity<Скидки>(entity =>

{

entity.HasKey(e => new { e.Город, e.Длительность })

.HasName("Скидка\_pk");

});

modelBuilder.Entity<ЮридЛица>(entity =>

{

entity.HasKey(e => e.Инн)

.HasName("Юрид\_лицо\_pk");

entity.ToTable("Юрид\_лица");

entity.HasIndex(e => e.Наименование)

.HasName("Название\_фирмы")

.IsUnique();

entity.Property(e => e.Инн)

.HasColumnName("ИНН")

.ValueGeneratedNever();

entity.Property(e => e.БанкСчет).HasColumnName("Банк\_счет");

entity.Property(e => e.Город).IsRequired();

entity.Property(e => e.Наименование).IsRequired();

entity.Property(e => e.ТелТочка).HasColumnName("Тел\_точка");

});

OnModelCreatingPartial(modelBuilder);

}

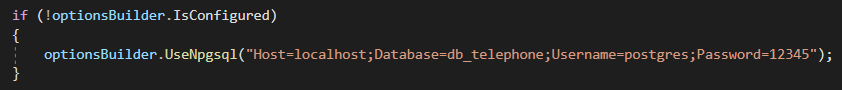


Рисунок 3. Подключение к базе данных

После создания классов и контекста для базы данных мы можем приступать к созданию приложения. Данное приложение будет консольным приложением. Пользователь для ввода, вывода, изменения и удаления данных будет вводить SQL-код запросов, и получать результаты.

В главном методе «Main» в классе «Program» создадим бесконечный цикл while, который при каждой своей итерации будет принимать запросы с консоли и выводить их результат на экран. Для выхода из цикла и из программы пользователю можно будет ввести команду exit или нажать сочетания клавиш Ctrl+C.

После принятия запроса в виде строки, создадим контекст «db» объекта «TelephoneContext», внутри которого мы будем работать с базой данных. Далее внутри контекста создаем контекст «command» для объекта через метод **«db.Database.GetDbConnection().CreateCommand()»**. Через поле **«commandText»** мы добавляем заранее введённую строку запроса. Подключаемся к БД через метод **«OpenConnection»** объекта **«db.Database»**.

Также мы создадим обработчик исключений, чтобы в случае ошибки со стороны Npgsql вывести об этом сообщение и не прерывать работу программы.

Далее создаем другой контекст «reader» для объекта через метод **«command.ExecuteReader()»**, который выполнит данный введенный запрос. В случае запроса SELECT, мы читаем из объекта «reader» информацию об выделенных столбцах (через Linq) и последовательно читаем каждую строку из таблицы (с помощью метода **«reader.Read()»**). После получения данных мы выводим таблицу на экран.

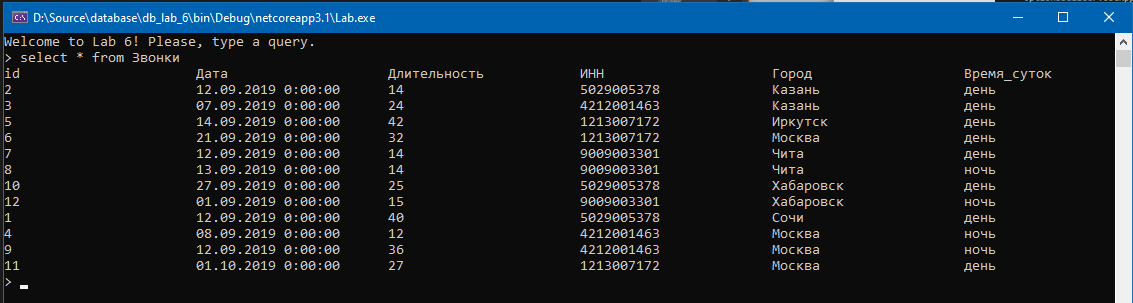


Рисунок 4. Результат работы запроса select

При других запросах в случае, если не произошла ошибка синтаксиса запроса или нарушение ограничения целостности, то программа после выполнения предложит ввести новый запрос. В случае ошибки программа выведет об этом сообщение:

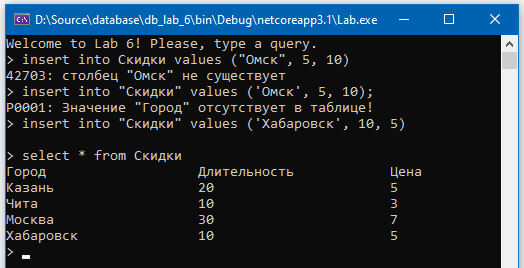


Рисунок 5. Результаты ввода различных запросов.

Полный код класса Program

**using** Microsoft.EntityFrameworkCore;

**using** Lab.Models;

**using** System;

**using** System.Data;

**using** System.Linq;

**using** System.Collections.Generic;

**namespace** Lab

{

**class** Program

{

**static** **void** Main(**string**[] args)

{

**string** input = **string**.Empty;

Console.WriteLine("Welcome to Lab 6! Please, type a query.");

**while** (**true**)

{

List<List<**string**>> table = new List<List<**string**>>();

Console.Write("> ");

input = Console.ReadLine();

**if** (input.ToLower() == "exit") **return**;

**using** (**var** db = new TelephoneContext())

{

**using** (**var** command = db.Database.GetDbConnection().CreateCommand())

{

command.CommandText = input;

**db.Database.OpenConnection();**

**try**

{

**using** (**var** reader = command.ExecuteReader())

{

**var** columns = Enumerable.Range(0, reader.FieldCount).**Select**(reader.GetName).ToList();

table.**Add**(columns);

**while** (reader.Read())

{

**var** rows = new List<**string**>();

**for** (**int** i = 0; i < reader.FieldCount; i++)

{

rows.**Add**(Convert.ToString(reader.GetValue(i)));

}

table.**Add**(rows);

}

}

}

**catch** (Npgsql.NpgsqlException error)

{

Console.WriteLine(error.Message);

**continue**;

}

}

**foreach** (**var** row **in** table)

{

**foreach** (**var** item **in** row)

{

Console.Write($"{item,-24}");

}

Console.WriteLine();

}

}

}

}

}

}