Министерство науки и высшего образования Российской Федерации

Федеральное государственное бюджетное образовательное учреждение высшего образования

ТОМСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ СИСТЕМ УПРАВЛЕНИЯ И РАДИОЭЛЕКТРОНИКИ (ТУСУР)

Кафедра автоматизированных систем управления (АСУ)

**ПРОГРМАННОЕ ПОДКЛЮЧЕНИЕ К СЕРВЕРУ   
POSTGRESQL**

Отчёт о лабораторной работе № 4

по дисциплине «Базы данных»

Выполнил: студент гр. 431-3

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ Гурулёв А. В.

«\_\_\_»\_\_\_\_\_\_\_\_\_\_\_ 2023 г.

Проверил: ассистент каф. АСУ

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ Яблонский Я. В.

«\_\_\_»\_\_\_\_\_\_\_\_\_\_\_ 2023

г.Томск 2023

Оглавление

[1 Цель лабораторной работы 3](#_Toc151997704)

[2 Описание требований к отчёту 4](#_Toc151997705)

[3 Листинг 5](#_Toc151997706)

[4 Описание процесса реализации 22](#_Toc151997707)

[5 Выводы 26](#_Toc151997708)

# Цель лабораторной работы

Цели:

* научиться создавать программы, взаимодействующие с сервером СУБД.

Задачи:

* разработать программу, которая выполняет подключение к серверу СУБД PostgreSQL, заносит новые данные в одну из таблиц (по выбору студента), формирует на экране отчёт;

# Описание требований к отчёту

На рисунке 2.1 показано как должен выглядеть отчёт.

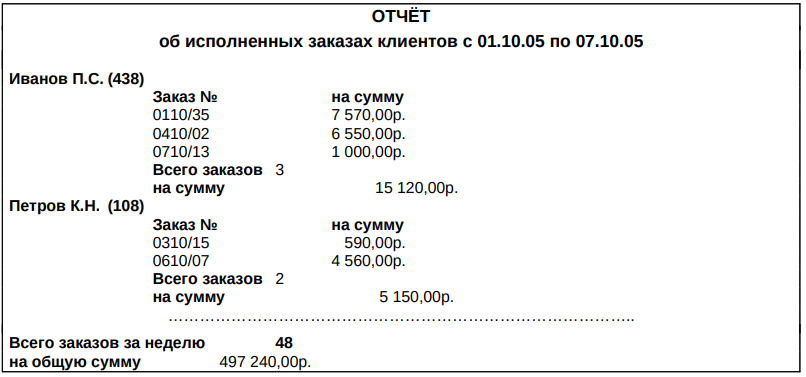


Рисунок . - шаблон отчёта на вывод

# Листинг

Program.cs:

using System;

using Microsoft.EntityFrameworkCore;

using Npgsql.EntityFrameworkCore.PostgreSQL;

using Lab4;

using Lab4.Db;

namespace Lab4

{

class Program

{

public static void Main(string[] args)

{

// Список атрибутов таблиц, для удобства в дальнейшем

string[][] attributes = { new string[] {"Art", "Name\_G", "Meas", "Price\_G" },

new string[] { "Code\_C", "FIO", "Addr\_C", "Tel" },

new string[] { "Code\_Z", "Code\_C", "Acc\_DT", "Addr\_D", "Del\_DT", "Price\_D" },

new string[] { "Code\_Z", "Art", "Qt" }

};

bool flag = true;

while (flag)

{

// Запрашиваем тип действия

byte mode;

Console.WriteLine("Выберите действие:\n\t1. Внести новые данные.\n\t2. Вывести отчёт за неделю.\n\tОстальное - выйти");

try

{

mode = Convert.ToByte(Console.ReadLine());

}

catch (Exception ex)

{

Console.WriteLine("Ошибка ввода");

flag = false;

break;

}

// Выполняем запрошенное действие

Console.WriteLine();

switch (mode)

{

case 1:

// Заправшиваем целевую таблицу

Console.WriteLine("Выберите тиблицу:\n\t1. GOODS.\n\t2. CLIENT.\n\t3. CLIENT\_ORDER\n\t4. ORD\_GD\n\tОстальное - вернуться");

try

{

mode = Convert.ToByte(Console.ReadLine());

}

catch

{

break;

}

Console.WriteLine();

// Утанавливаем количество полей в указанной таблице

byte count\_atttr = 0;

switch (mode)

{

case 1:

case 2:

count\_atttr = 4;

break;

case 3:

count\_atttr = 6;

break;

case 4:

count\_atttr = 3;

break;

default:

break;

}

if (count\_atttr == 0)

break;

// Даем пользователю ввести значения

string[] input = new string[count\_atttr];

for (byte i = 0; i < count\_atttr; i++)

{

Console.WriteLine($"Введите значение атриубта {attributes[mode-1][i]}:");

input[i] = Console.ReadLine();

}

try

{

// Вносим значения

using (LabsContext labs = new LabsContext())

{

switch (mode)

{

case 1:

labs.Goods.Add(new Good

{

Art = input[0],

NameG = input[1],

Meas = input[2],

PriceG = Convert.ToDecimal(input[3].Replace('.', ','))

});

break;

case 2:

labs.Clients.Add(new Client

{

CodeC = Convert.ToInt32(input[0]),

Fio = input[1],

AddrC = input[2],

Tel = input[3]

});

break;

case 3:

labs.ClientOrders.Add(new ClientOrder

{

CodeZ = input[0],

CodeC = Convert.ToInt32(input[1]),

AccDt = DateOnly.Parse(input[2]),

AddrD = input[3],

DelDt = DateOnly.Parse(input[4]),

PriceD = Convert.ToDecimal(input[5])

});

break;

case 4:

labs.OrdGds.Add(new OrdGd

{

CodeZ = input[0],

Art = input[1],

Qt = Convert.ToInt32(input[2])

});

break;

}

labs.SaveChanges();

}

}

catch (Exception ex)

{

Console.WriteLine(ex.Message);

}

break;

case 2:

Console.WriteLine("Введите дату - начало отчёта");

DateOnly startDate;

try

{

startDate = DateOnly.Parse(Console.ReadLine());

}

catch

{

Console.WriteLine("Ошибка чтения даты");

flag = false;

break;

}

DateOnly endDate = startDate.AddDays(6);

//Фомриурем отчёт

using (LabsContext labs = new LabsContext())

{

var allTable = labs.ClientOrders.Where(x => (x.DelDt >= startDate) && (x.DelDt <= (endDate)))

.Join(labs.OrdGds, f => f.CodeZ, s => s.CodeZ, (f, s) => new { f.CodeC, f.CodeZ, f.PriceD, s.Art, s.Qt })

.Join(labs.Goods, f => f.Art, s => s.Art, (f, s) => new { f.CodeC, f.CodeZ, f.Qt, s.PriceG, f.PriceD })

.Join(labs.Clients, f => f.CodeC, s => s.CodeC, (f, s) => new { f.CodeC, s.Fio, f.CodeZ, f.Qt, f.PriceG, f.PriceD })

.Select(x => new { ClientCode = x.CodeC, FIO = x.Fio, OrderCode = x.CodeZ, OrderPrice = Convert.ToDecimal(x.Qt.Value) \* Convert.ToDecimal(x.PriceG.Value), DelPrice = x.PriceD })

.OrderBy(x => x.ClientCode)

.ThenBy(x => x.OrderCode);

var finalTable = allTable.Join(allTable.GroupBy(x => x.OrderCode).Select(x => new { code = x.Key, sum = x.Sum(y => y.OrderPrice) }),

f => f.OrderCode,

s => s.code,

(f, s) => new { f.ClientCode, f.FIO, f.OrderCode, OrderPrice = Convert.ToDecimal(s.sum) + Convert.ToDecimal(f.DelPrice.Value) })

.Distinct();

var tableInLst = finalTable.ToList();

// Локальные переменные для каждого клиента

int codeC = (int)tableInLst[0].ClientCode;

int \_count = 0;

decimal \_sum = 0;

// Для конца отчёта

int count = 0;

decimal sum = 0;

Console.WriteLine($"\t\t\t\t\tОТЧЁТ\n\t\tоб исполненных заказах клиентов с {startDate} по {endDate}");

foreach (var row in tableInLst)

{

if ((count == 0) && (\_count == 0))

{

Console.WriteLine($"\n{row.FIO} ({row.ClientCode})\n\tЗаказ #\t\tна сумму");

}

if ((row.ClientCode != codeC) && (\_count != 0))

{

Console.WriteLine($"\tВсего заказов\t{\_count}\n\tна сумму\t{\_sum}р");

Console.WriteLine($"\n{row.FIO} ({row.ClientCode})\n\tЗаказ #\t\tна сумму");

count += \_count;

\_count = 0;

sum += \_sum;

\_sum = 0;

}

Console.WriteLine($"\t{row.OrderCode}\t{row.OrderPrice}р");

\_count++;

\_sum += row.OrderPrice;

}

Console.WriteLine($"\tВсего заказов\t{\_count}\n\tна сумму\t{\_sum}р");

count += \_count;

sum += \_sum;

Console.WriteLine($"\nВсего заказов за период\t{count}\nна общую сумму\t\t{sum}р\n");

}

break;

default:

flag = false;

break;

}

}

}

}

}

LabsContext.cs

using System;

using System.Collections.Generic;

using Microsoft.EntityFrameworkCore;

namespace Lab4.Db;

public partial class LabsContext : DbContext

{

public LabsContext()

{

}

public LabsContext(DbContextOptions<LabsContext> options)

: base(options)

{

}

public virtual DbSet<Client> Clients { get; set; }

public virtual DbSet<ClientOrder> ClientOrders { get; set; }

public virtual DbSet<Good> Goods { get; set; }

public virtual DbSet<OrdGd> OrdGds { 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.UseNpgsql("Host=localhost;Port=5432;Database=labs;Username=postgres;Password=002vTolnaS");

protected override void OnModelCreating(ModelBuilder modelBuilder)

{

modelBuilder.Entity<Client>(entity =>

{

entity.HasKey(e => e.CodeC).HasName("client\_pkey");

entity.ToTable("client");

entity.Property(e => e.CodeC).HasColumnName("code\_c");

entity.Property(e => e.AddrC)

.HasMaxLength(25)

.HasColumnName("addr\_c");

entity.Property(e => e.Fio)

.HasMaxLength(35)

.HasColumnName("fio");

entity.Property(e => e.Tel)

.HasMaxLength(38)

.HasColumnName("tel");

});

modelBuilder.Entity<ClientOrder>(entity =>

{

entity.HasKey(e => e.CodeZ).HasName("client\_order\_pkey");

entity.ToTable("client\_order");

entity.Property(e => e.CodeZ)

.HasMaxLength(9)

.HasColumnName("code\_z");

entity.Property(e => e.AccDt).HasColumnName("acc\_dt");

entity.Property(e => e.AddrD)

.HasMaxLength(25)

.HasColumnName("addr\_d");

entity.Property(e => e.CodeC).HasColumnName("code\_c");

entity.Property(e => e.DelDt).HasColumnName("del\_dt");

entity.Property(e => e.PriceD)

.HasColumnType("money")

.HasColumnName("price\_d");

entity.HasOne(d => d.CodeCNavigation).WithMany(p => p.ClientOrders)

.HasForeignKey(d => d.CodeC)

.HasConstraintName("client\_order\_code\_c\_fkey");

});

modelBuilder.Entity<Good>(entity =>

{

entity.HasKey(e => e.Art).HasName("goods\_pkey");

entity.ToTable("goods");

entity.Property(e => e.Art)

.HasMaxLength(12)

.IsFixedLength()

.HasColumnName("art");

entity.Property(e => e.Meas)

.HasMaxLength(3)

.HasColumnName("meas");

entity.Property(e => e.NameG)

.HasMaxLength(30)

.HasColumnName("name\_g");

entity.Property(e => e.PriceG)

.HasColumnType("money")

.HasColumnName("price\_g");

});

modelBuilder.Entity<OrdGd>(entity =>

{

entity.HasKey(e => new { e.CodeZ, e.Art }).HasName("ord\_gd\_pkey");

entity.ToTable("ord\_gd");

entity.Property(e => e.CodeZ)

.HasMaxLength(9)

.HasColumnName("code\_z");

entity.Property(e => e.Art)

.HasMaxLength(12)

.HasColumnName("art");

entity.Property(e => e.Qt).HasColumnName("qt");

entity.HasOne(d => d.ArtNavigation).WithMany(p => p.OrdGds)

.HasForeignKey(d => d.Art)

.OnDelete(DeleteBehavior.ClientSetNull)

.HasConstraintName("ord\_gd\_art\_fkey");

entity.HasOne(d => d.CodeZNavigation).WithMany(p => p.OrdGds)

.HasForeignKey(d => d.CodeZ)

.OnDelete(DeleteBehavior.ClientSetNull)

.HasConstraintName("ord\_gd\_code\_z\_fkey");

});

OnModelCreatingPartial(modelBuilder);

}

partial void OnModelCreatingPartial(ModelBuilder modelBuilder);

}

Client.cs:

using System;

using System.Collections.Generic;

namespace Lab4.Db;

public partial class Client

{

public int CodeC { get; set; }

public string? Fio { get; set; }

public string? AddrC { get; set; }

public string? Tel { get; set; }

public virtual ICollection<ClientOrder> ClientOrders { get; set; } = new List<ClientOrder>();

}

Good.cs:

using System;

using System.Collections.Generic;

namespace Lab4.Db;

public partial class Good

{

public string Art { get; set; } = null!;

public string? NameG { get; set; }

public string? Meas { get; set; }

public decimal? PriceG { get; set; }

public virtual ICollection<OrdGd> OrdGds { get; set; } = new List<OrdGd>();

}

ClientOreder.cs:

using System;

using System.Collections.Generic;

namespace Lab4.Db;

public partial class ClientOrder

{

public string CodeZ { get; set; } = null!;

public int? CodeC { get; set; }

public DateOnly? AccDt { get; set; }

public string? AddrD { get; set; }

public DateOnly? DelDt { get; set; }

public decimal? PriceD { get; set; }

public virtual Client? CodeCNavigation { get; set; }

public virtual ICollection<OrdGd> OrdGds { get; set; } = new List<OrdGd>();

}

OrdGd.cs:

using System;

using System.Collections.Generic;

namespace Lab4.Db;

public partial class OrdGd

{

public string CodeZ { get; set; } = null!;

public string Art { get; set; } = null!;

public int? Qt { get; set; }

public virtual Good ArtNavigation { get; set; } = null!;

public virtual ClientOrder CodeZNavigation { get; set; } = null!;

}

# Описание процесса реализации

Для работы с СУБД использовался язык C# и библиотека  
 EntityFramework. Для работы с PostgreSQL нужно установить пакеты, которые можно увидеть на рисунке 4.1.

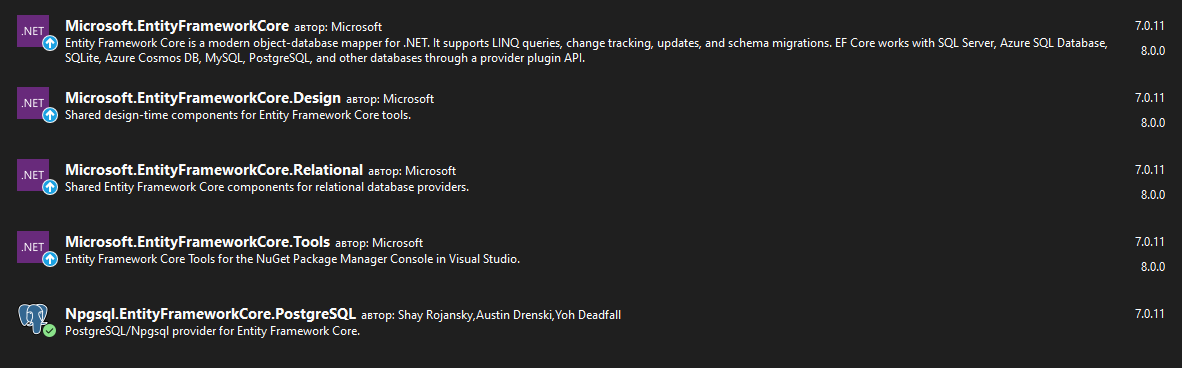


Рисунок . – установленные пакеты

После установки пакетов необходимо создать классы, описывающие базу данных, сделать это можно используя команду Scaffold-DbContext, для текущей базы нужно ввести следующее: *Scaffold-DbContext "Host=localhost; Port=5432; Database=labs; Username=postgres; Password=123456789"*, далее нужно указать поставщика данных, в текущем случае это Npgsql.EntityFrameworkCore.PostgreSQL.

После этого в проекте появятся файлы, описывающие базу данных. Их список можно увидеть на рисунке 4.2, а содержание в разделе 3.

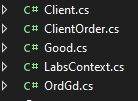


Рисунок . – файлы описания базы данных

Далее подключение к базе данных будет происходить при помощи класса LabsContext.cs, конкретно при помощи вызова функции внутри него UseNpsql({параметры подключения}).

Теперь можно приступать к разработке программы.

Сначала создается массив attributes, где перепишем атрибуты таблиц, для удобного их вывода в дальнейшем. Далее идет цикл, чтобы программа не завершалась без желания пользователя. После чего запрашивается тип действия пользователя (внести данные или получить отчёт).

В случае выбора первого типа – у пользователя запрашивается таблица, в которую он хочет внести данные. После чего ему выводится и запрашивается список атрибутов.

Далее, при помощи функции Add(), указанные значения вносятся в таблицу, после чего при помощи функции SaveChanges() внесенные изменения сохраняются.

Формирование запросов схоже с SQL, но вместо команд используются функции, которые имеет класс таблицы. К примеру, метод join выполняет те же действия, что и команда JOIN в SQL, но принимает аргументы по-другому. Первым аргументом идет таблица, с которой будет объединение, вторым аргументом выступает колонка первой таблицы, по которой она будет объединяться с второй таблицей, третьим аргументом соответствующая колонка для второй таблицы, четвертым аргументом идет что-то вроде SELECT, перечисление атрибутов, которые будет содержать итоговая таблица.

Таким образом строка

labs.ClientOrders.Join(labs.OrdGds, f => f.CodeZ, s => s.CodeZ, (f, s) => new { f.CodeC, f.CodeZ, f.PriceD, s.Art, s.Qt })

Является аналогом

SELECT CodeZ, CodeC, PriceD, Art, Qt FROM TABLE ClientOrders JOIN OrdGds ON ClientOrders.CodeZ = OrdGds.CodeZ;

Если выбрали второй тип – формируется таблица allTable, которая содержит записи о заказе с датой выполнения в указанном промежутке, с данными о списке товаров в заказе, стоимости за все единицы товара, стоимости доставки, коде клиента, фио клиента, по которому записи упорядочиваются в первую очередь, коде заказа, по которому записи упорядочиваются во вторую очередь. Далее стоимость всех типов товара в одном заказе суммируются, к результату добавляется стоимость доставки и получается атрибут с общей стоимостью заказа. В итоге получается таблица с кодом клиента, фио клиента, кодом заказа, стоимостью заказа. Далее все указанные данные выводятся в требуемом виде.

На рисунке 4.3 изображен запрос типа действия пользователя.

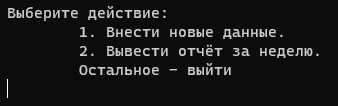


Рисунок .- запрос типа действия пользователя

На рисунке 4.4 изображен процесс ввода данных.

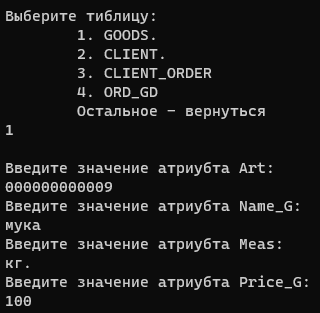


Рисунок . - ввод данных в таблицу goods

На рисунке 4.5 изображен выводимый отчёт.

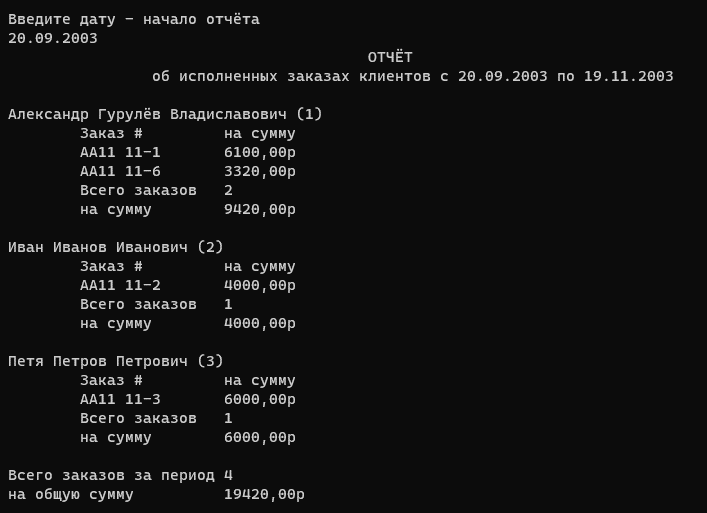


Рисунок . - отчёт

# Выводы

В ходе лабораторной работы:

* получены навыки работы с базами данных при разработке программ;
* получены навыки работы с технологией EntityFramework для .NET