НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ

«КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ імені Ігоря Сікорського»

ФАКУЛЬТЕТ ПРИКЛАДНОЇ МАТЕМАТИКИ

Кафедра системного програмування та спеціалізованих комп'ютерних систем

Розрахунково-графічна робота

з дисципліни

«Бази даних і засоби управління»

Виконав: студент групи КВ-13

Шпилька І.В.

Перевірив: Петрашенко А.В.

Київ – 2023

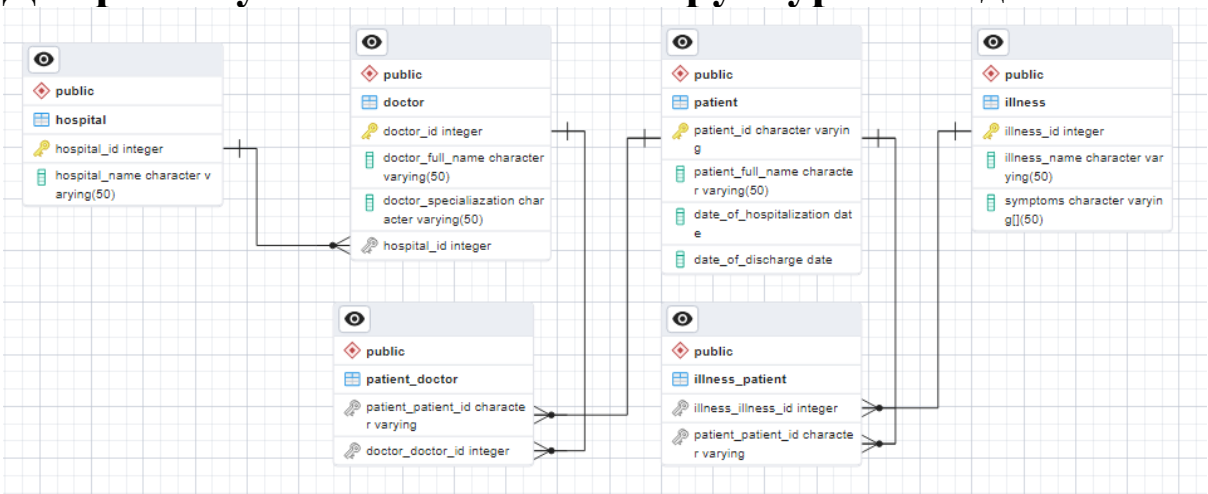
Створення додатку бази даних, орієнтованого на взаємодію з СУБД PostgreSQL

Метою роботи є здобуття вмінь програмування прикладних додатків баз даних PostgreSQL.

Загальне завдання роботи полягає у наступному:

1. Реалізувати функції перегляду, внесення, редагування та вилучення даних у таблицях бази даних, створених у лабораторній роботі №1, засобами консольного інтерфейсу.
2. Передбачити автоматичне пакетне генерування «рандомізованих» даних у базі.
3. Забезпечити реалізацію пошуку за декількома атрибутами з двох та більше сутностей одночасно: для числових атрибутів – у рамках діапазону, для рядкових – як шаблон функції LIKE оператора SELECT SQL, для логічного типу – значення True/False, для дат – у рамках діапазону дат.
4. Програмний код виконати згідно шаблону MVC (модель-подання-контролер).

Діаграма сутність-зв’язок та структура бази даних

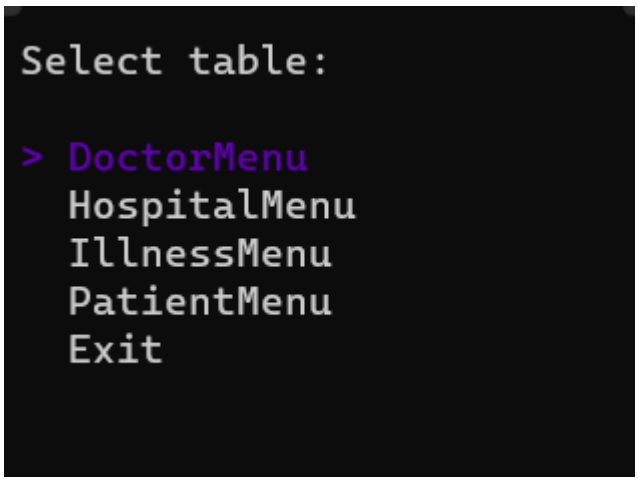


Використані технології

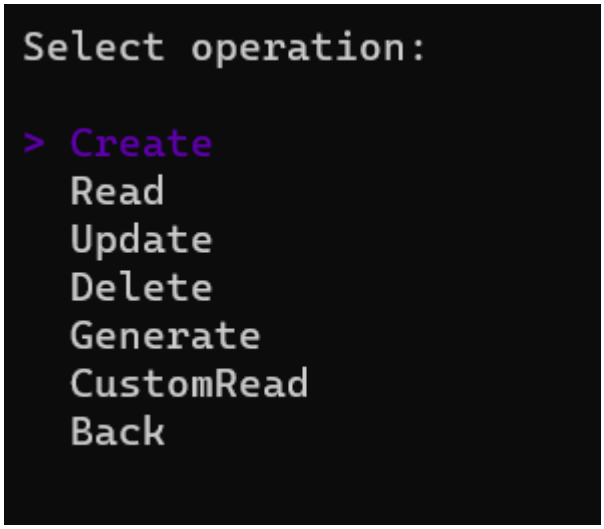
- .NET
  - ADO.NET
  - Spectre.Console

Схема меню користувача

Головне меню:



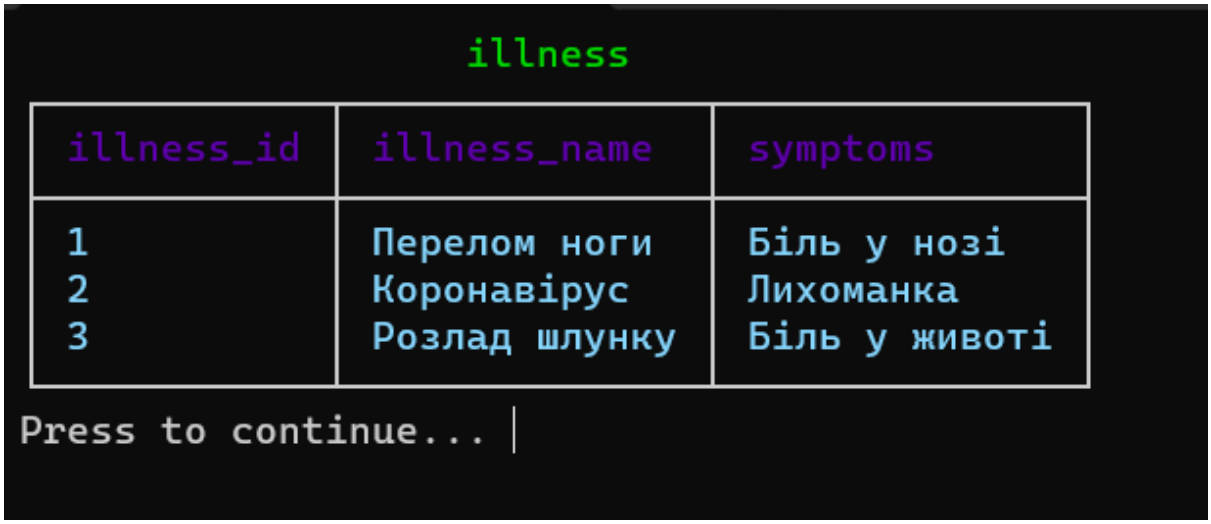
Меню з операціями CRUD



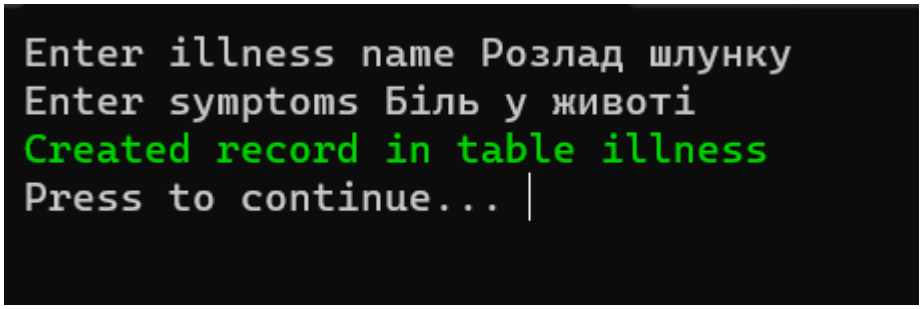
Тестування операцій CRUD

Для прикладу візьмемо таблицю illness

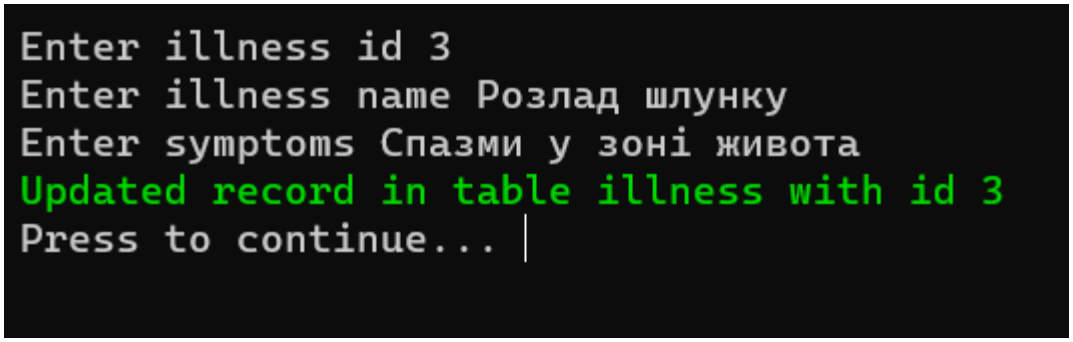
Read:

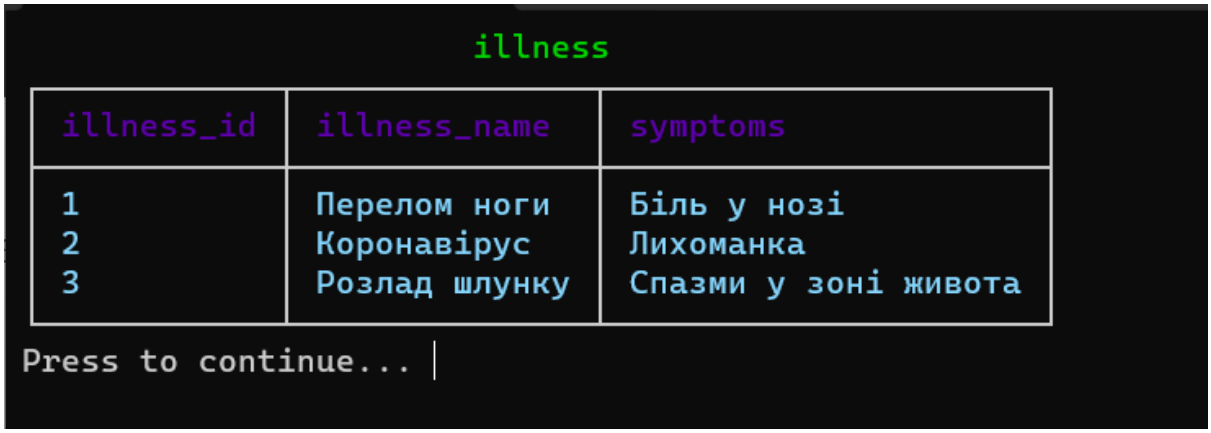


Create:

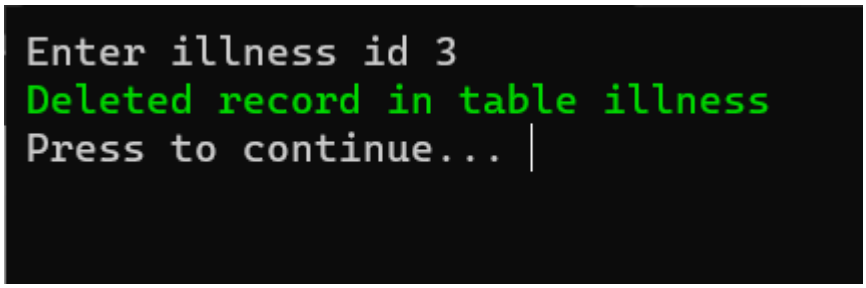


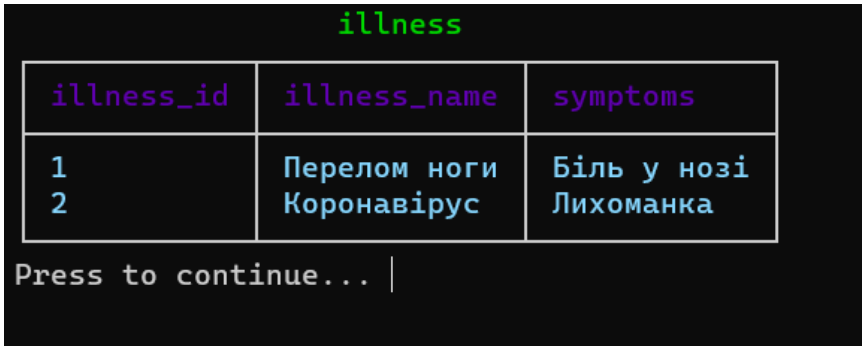
Update:





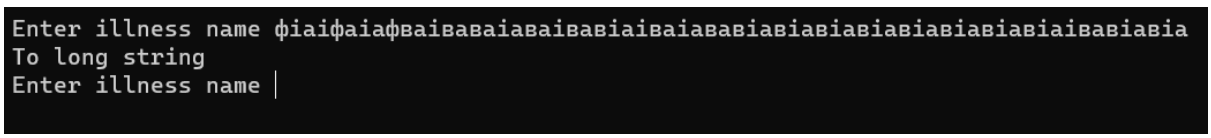
Delete:



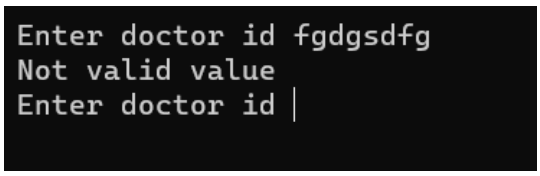


Валідація

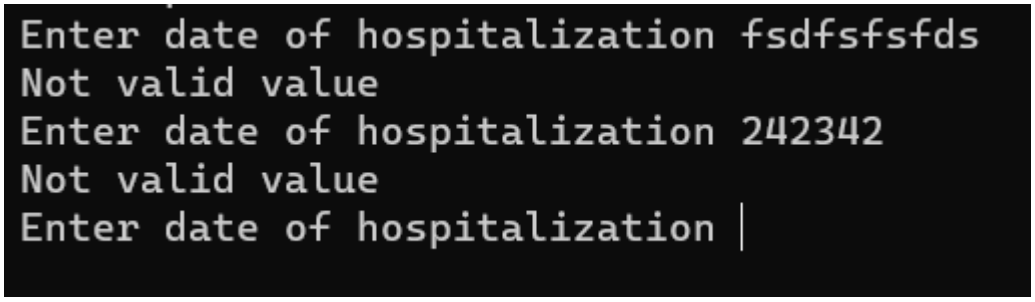
string:



int:

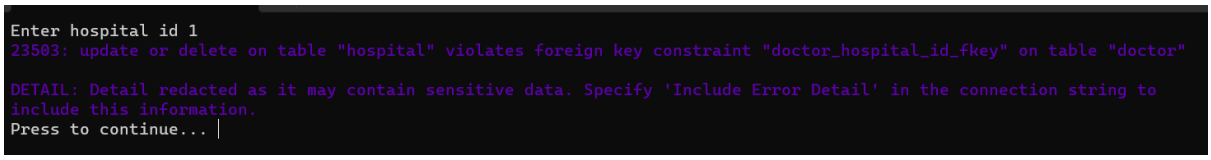


date:



Видалення батьківських таблиць

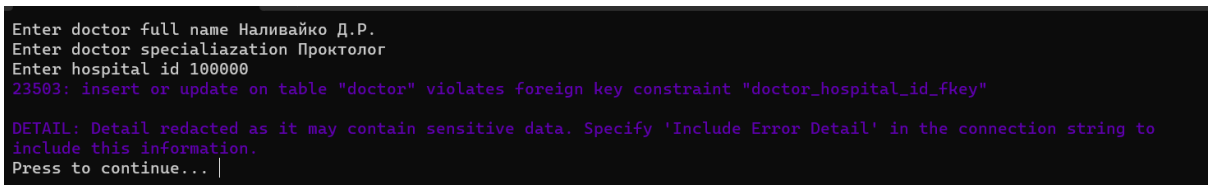
Спробуємо видалити запис із таблиці hospital



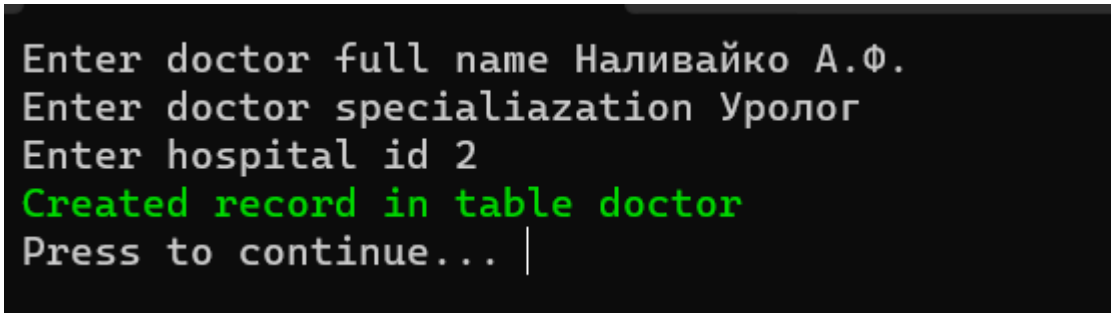
Як видно, нам не вдається виконати видалення за допомогою зовнішнього ключа. Програма працює належним чином.

Вставка дочірніх таблиць

Спробуємо створити запис у таблицю doctor з невірним значенням ключа hospital_id



Видно, що програма працює коректно. Давайте спробуємо вставити значення з вже існуючим ключем.



doctor

doctor_id	doctor_full_name	doctor_specialiazation	hospital_id
1	Кирилов П.З.	Хірург	1
2	Зілізничний П.І.	Терапевт	2
4	Наливайко А.Ф.	Уролог	2

Press to continue...

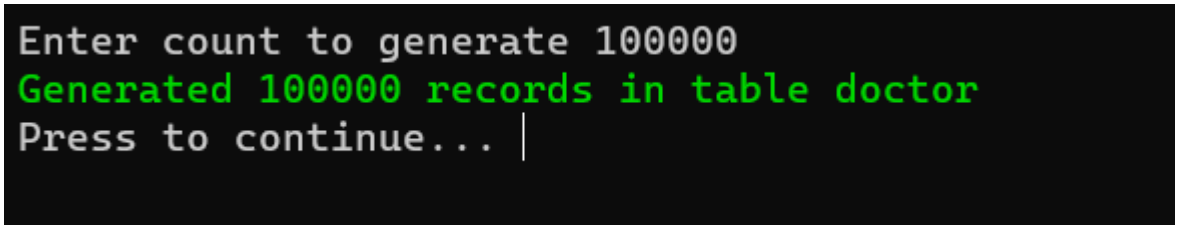
Генерація рандомізованих даних

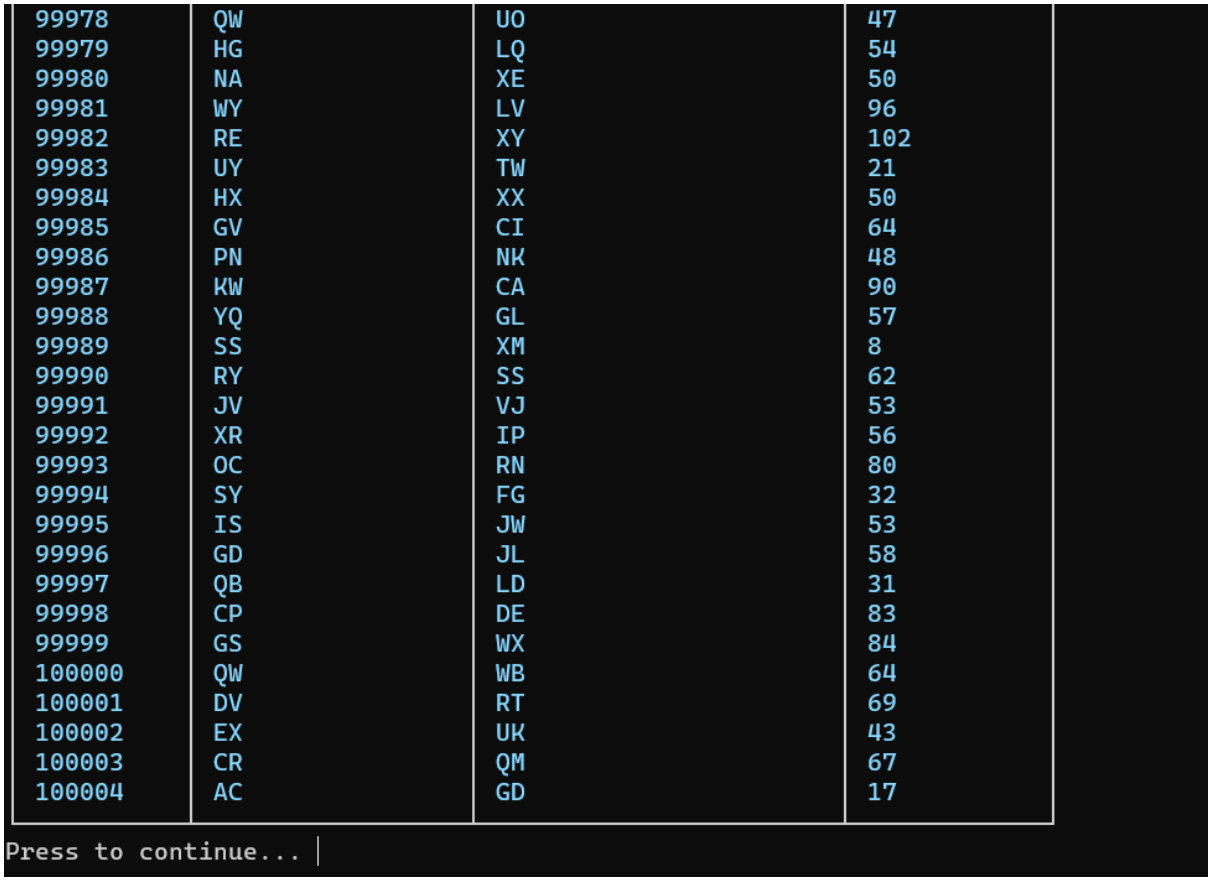
Створимо 100 записів для всіх таблиць для перевірки правильності складних запитів. Далі, для тестування генерації великої кількості даних, створимо 100000 записів для таблиці doctor. Виконаємо це за допомогою опції Generate.

```
Enter count to generate 100
Generated 100 records in table hospital
Press to continue...
```

92	CY
93	EY
94	UJ
95	DU
96	EL
97	FK
98	XN
99	EN
100	DW
101	WS
102	DW

Press to continue...





99978	QW	UO	47
99979	HG	LQ	54
99980	NA	XE	50
99981	WY	LV	96
99982	RE	XY	102
99983	UY	TW	21
99984	HX	XX	50
99985	GV	CI	64
99986	PN	NK	48
99987	KW	CA	90
99988	YQ	GL	57
99989	SS	XM	8
99990	RY	SS	62
99991	JV	VJ	53
99992	XR	IP	56
99993	OC	RN	80
99994	SY	FG	32
99995	IS	JW	53
99996	GD	JL	58
99997	QB	LD	31
99998	CP	DE	83
99999	GS	WX	84
100000	QW	WB	64
100001	DV	RT	69
100002	EX	UK	43
100003	CR	QM	67
100004	AC	GD	17

Запити для генерації даних:

doctor:

insert into doctor(doctor_full_name, doctor_specialiazation, hospital_id) values(chr(trunc(65 + random() * 25)::int) || chr(trunc(65 + random() * 25)::int), chr(trunc(65 + random() * 25)::int) || chr(trunc(65 + random() * 25)::int), (select f_id from trunc(1 + random() * (select max(hospital_id) from hospital)) as f_id inner join hospital on f_id = hospital.hospital_id)::int);

hospital:

insert into hospital(hospital_name) values(chr(trunc(65 + random() * 25)::int) || chr(trunc(65 + random() * 25)::int));

illness:

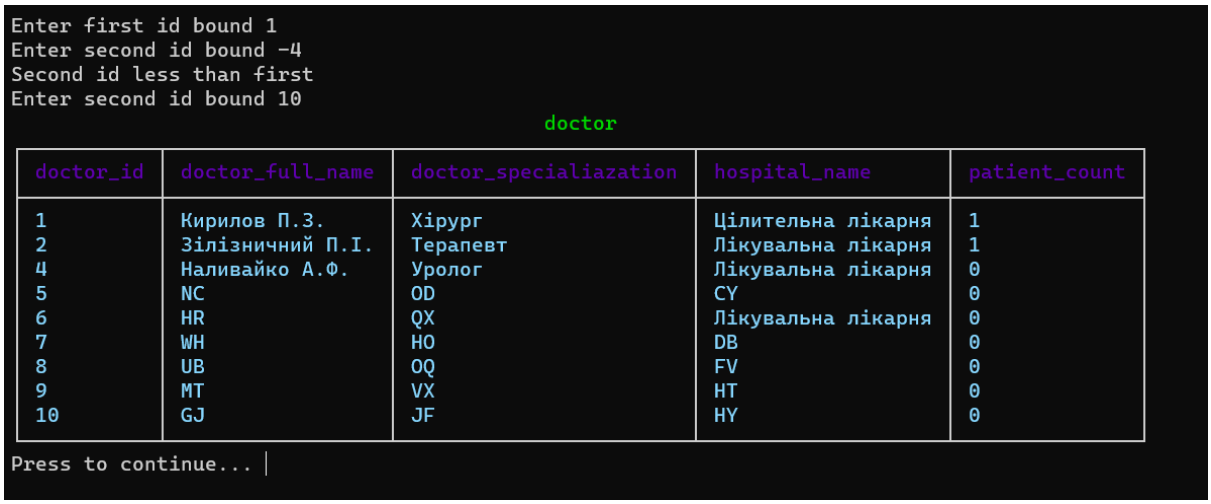
insert into illness(illness_name, symptoms) values(chr(trunc(65 + random() * 25)::int) || chr(trunc(65 + random() * 25)::int), chr(trunc(65 + random() * 25)::int) || chr(trunc(65 + random() * 25)::int));

patient:

insert into patient(patient_full_name, date_of_hospitalization, date_of_discharge) values(chr(trunc(65 + random() * 25)::int) || chr(trunc(65 + random() * 25)::int), date '1800-10-19' + (random() * (date '{2000-10-19}' - date '{1800-10-19}'))::int, date '1800-10-19' + (random() * (date '{2000-10-19}' - date '{1800-10-19}'))::int);

Ілюстрації уведення пошукового запиту та результатів виконання запитів

doctor:



select

doctor_id,

doctor_full_name,

doctor_specialiazation,

hospital_name,

count(patient_id) as patient_count

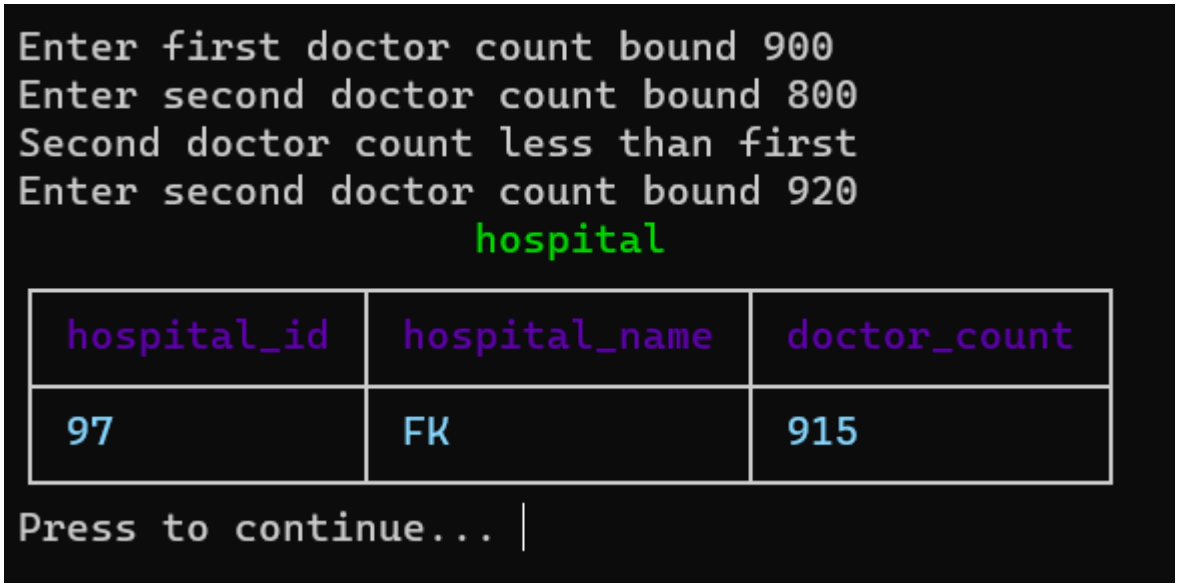
```
from
    doctor
full join hospital using(hospital_id)
full join patient_doctor using(doctor_id)
where
    doctor_id between {firstBound} and {secondBound}
group by
    doctor_id,
    doctor_full_name,
    doctor_specialiazation,
    hospital_name
order by doctor_id;
```

hospital:

```
Enter first doctor count bound 1
Enter second doctor count bound 3
                  hospital
```

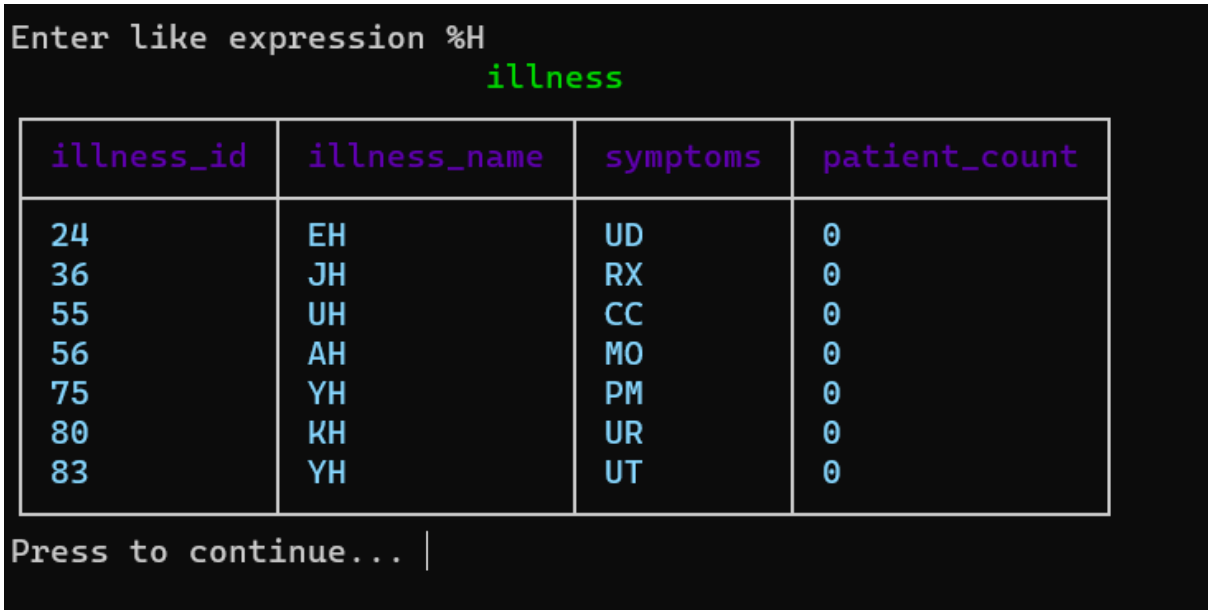
hospital_id	hospital_name	doctor_count
122	JW	1
168	EF	1

```
Press to continue...
```



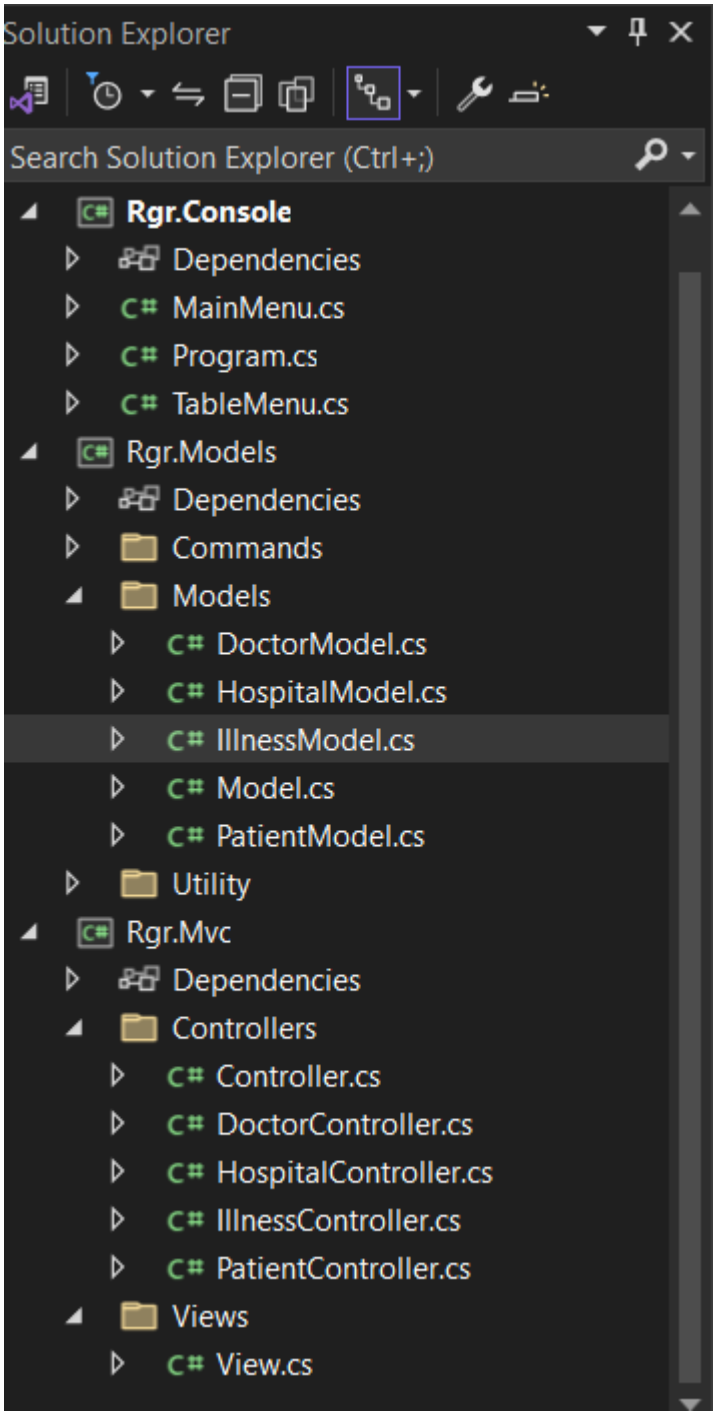

```
select
        hospital_id,
        hospital_name,
        count(doctor_id) as doctor_count
    from
        hospital
    full join doctor using(hospital_id)
    group by
        hospital_id,
        hospital_name
    having
       count(doctor_id) between {firstBound} and {secondBound}
    order by hospital_id;
```

illness:




```
select
        illness_id,
        illness_name,
    symptoms,
        count(patient_id) as patient_count
from
        illness
full join illness_patient using(illness_id)
where
    illness_name like '{likeExp}'
group by
        illness_id,
        illness_name,
    symptoms
order by illness_id;
```

Ілюстрації програмного коду модуля “Model”



Command.cs

```
using Npgsql;
using System.Data;

namespace Rgr.Models.Commands;

public abstract class Command
{
```

```
    protected readonly NpgsqlConnection _connection;
    protected readonly NpgsqlCommand _command;

    public Command(NpgsqlConnection connection)
    {
        _connection = connection;
        _command = _connection.CreateCommand();
        _command.CommandType = CommandType.Text;
    }
}
```

CreateCommand.cs

```
using Npgsql;

namespace Rgr.Models.Commands;

public class CreateCommand(NpgsqlConnection connection) : Command(connection)
{
    public void Execute(
        string tableName,
        IEnumerable<string> tableColumns,
        IEnumerable<object> values)
    {
        _connection.Open();

        try
        {
            List<string> valueStrings = new List<string>();

            for (int i = 0; i < values.Count(); i++)
            {
                _command.Parameters.AddWithValue($"@param{i}", values.ToList()[i]);
                valueStrings.Add($"@param{i}");
            }

            string valuesString = Utility.Utility.AggregateWithCommas(valueStrings);
            string tableColumnsString = Utility.Utility.AggregateWithCommas(tableColumns);

            _command.CommandText = $"insert into {tableName}({tableColumnsString})
values({valuesString});";

            _command.ExecuteNonQuery();
        }
        finally { _connection.Close(); }
    }
}
```

CustomCommand.cs

```
using Npgsql;

namespace Rgr.Models.Commands;

public class CustomCommand : Command
{
    public List<string> ColumnNames { get; private set; }

    public CustomCommand(NpgsqlConnection connection) : base(connection)
    {
    }

    public List<List<object>> Execute(string query, List<string> columnNames)
    {
        List<List<object>> objects = new List<List<object>>();

        ColumnNames = columnNames;

        _connection.Open();

        try
        {
            _command.CommandText = query;

            using (NpgsqlDataReader reader = _command.ExecuteReader())
            {
                while (reader.Read())
                {
                    objects.Add(new List<object>());

                    for (int i = 0; i < reader.FieldCount; i++)
                    {
                        objects.Last().Add(reader.GetValue(i));
                    }
                }
            }
        }
        finally { _connection.Close(); }

        return objects;
    }
}
```

DeleteCommand.cs

```
using Npgsql;

namespace Rgr.Models.Commands;

public class DeleteCommand(NpgsqlConnection connection) : Command(connection)
```

```
{
    public void Execute(
        string tableName,
        string primaryKeyColumnString,
        int id)
    {
        _connection.Open();

        try
        {
            _command.CommandText = $"delete from {tableName} where {primaryKeyColumnString} =
{id};";

            _command.ExecuteNonQuery();
        }
        finally { _connection.Close(); }
    }
}
```

GenerateCommand.cs

```
using Npgsql;
using System.Text;

namespace Rgr.Models.Commands;

public class GenerateCommand : Command
{
    public static string String() =>
        "chr(trunc(65 + random() * 25)::int) || chr(trunc(65 + random() * 25)::int)";

    public static string DateOnly() =>
        "date '1800-10-19' + (random() * (date '{2000-10-19}' - date '{1800-10-19}'))::int";

    public static string ForeignKey(string tableName, string keyColumnName) =>
        $"(select f_id from trunc(1 + random() * (select max({keyColumnName}) from {tableName})) as
f_id inner join {tableName} on f_id = {tableName}.{keyColumnName})::int";

    public GenerateCommand(NpgsqlConnection connection) : base(connection)
    {
    }

    public void Execute(
        string tableName,
        IEnumerable<string> tableColumns,
        IEnumerable<object> values)
    {
        _connection.Open();

        try
```

```
        {
            string valuesString = Utility.Utility.AggregateWithCommas(values);
            string tableColumnsString = Utility.Utility.AggregateWithCommas(tableColumns);

            _command.CommandText = $"insert into {tableName}({tableColumnsString})
values({valuesString});";

            _command.ExecuteNonQuery();
        }
        finally { _connection.Close(); }
    }
}
```

ReadCommand.cs

```
using Npgsql;

namespace Rgr.Models.Commands;

public class ReadCommand(NpgsqlConnection connection) : Command(connection)
{
    public List<List<object>> Execute(string tableName, string keyColumnName)
    {
        List<List<object>> objects = new List<List<object>>();

        _connection.Open();

        try
        {
            _command.CommandText = $"select * from {tableName} order by {keyColumnName};";

            using (NpgsqlDataReader reader = _command.ExecuteReader())
            {
                while (reader.Read())
                {
                    objects.Add(new List<object>());

                    for (int i = 0; i < reader.FieldCount; i++)
                    {
                        objects.Last().Add(reader.GetValue(i));
                    }
                }
            }
        }
        finally { _connection.Close(); }

        return objects;
    }
}
```

UpdateCommand.cs

```
using Npgsql;

namespace Rgr.Models.Commands;

public class UpdateCommand(NpgsqlConnection connection) : Command(connection)
{
   public void Execute(
      string tableName,
      IEnumerable<string> tableColumns,
      IEnumerable<object> values,
      string primaryKeyColumnString,
      int id)
   {
      _connection.Open();

      try
      {
         List<string> valueStrings = new List<string>();

         for (int i = 0; i < values.Count(); i++)
         {
            _command.Parameters.AddWithValue($"@param{i}", values.ToList()[i]);
            valueStrings.Add($"@param{i}");
         }

         string setString = Utility.Utility.ConvertToSetString(tableColumns, valueStrings);

         _command.CommandText = $"update {tableName} set {setString} where
{primaryKeyColumnString} = {id};";

         _command.ExecuteNonQuery();
      }
      finally {  _connection.Close(); }
   }
}
```

Model.cs

```
using Npgsql;
using Rgr.Models.Commands;

namespace Rgr.Models.Models;

public abstract class Model
{
   private readonly NpgsqlConnection _connection;

   protected readonly CreateCommand _createCommand;
```

```
    protected readonly ReadCommand _readCommand;

    protected readonly UpdateCommand _updateCommand;

    protected readonly DeleteCommand _deleteCommand;

    protected readonly GenerateCommand _generateCommand;

    protected readonly CustomCommand _customCommand;

    protected readonly string _tableName;

    protected readonly string _keyColumnName;

    protected readonly List<string> _columnNames;

    public IReadOnlyCollection<string> ColumnNames { get; }

    public string KeyColumnName { get => _keyColumnName; }

    public string TableName { get => _tableName; }

    protected List<string> _generateValues;

    protected Model(NpgsqlConnection connection, string tableName, string keyColumnName,
List<string> columnNames, List<string> generateValues)
    {
        _connection = connection;
        _tableName = tableName;

        _createCommand = new CreateCommand(_connection);
        _readCommand = new ReadCommand(_connection);
        _updateCommand = new UpdateCommand(_connection);
        _deleteCommand = new DeleteCommand(_connection);
        _generateCommand = new GenerateCommand(_connection);
        _customCommand = new CustomCommand(_connection);
        _columnNames = columnNames;

        ColumnNames = _columnNames;
        _keyColumnName = keyColumnName;
        _generateValues = generateValues;
    }

    public void Create(
        IEnumerable<object> values)
    {
        _createCommand.Execute(_tableName, _columnNames, values);
    }

    public List<List<object>> Read()
```

```
    {
        return _readCommand.Execute(_tableName, _keyColumnName);
    }

    public void Update(
        IEnumerable<object> values,
        int id)
    {
        _updateCommand.Execute(_tableName,
            _columnNames,
            values,
            _keyColumnName,
            id);
    }

    public void Delete(
        int id)
    {
        _deleteCommand.Execute(_tableName, _keyColumnName, id);
    }

    public void Generate()
    {
        _generateCommand.Execute(_tableName, _columnNames, _generateValues);
    }
}
```

DoctorModel.cs

```
using Npgsql;
using Rgr.Models.Commands;

namespace Rgr.Models.Models;

public sealed class DoctorModel : Model
{
    public DoctorModel(NpgsqlConnection connection) : base(
        connection,
        "doctor",
        "doctor_id",
        [
            "doctor_full_name",
            "doctor_specialiazation",
            "hospital_id"
        ],
        [
            GenerateCommand.String(),
            GenerateCommand.String(),
            GenerateCommand.ForeignKey("hospital", "hospital_id")
        ])
```

```
    {
    }

    public List<List<object>> ReadAllInfo(int firstBound, int secondBound, out List<string>
columnName)
    {
        var result = _customCommand.Execute(
            $@"select
                        doctor_id,
                        doctor_full_name,
                        doctor_specialiazation,
                        hospital_name,
                        count(patient_id) as patient_count
            from
                        doctor
            full join hospital using(hospital_id)
            full join patient_doctor using(doctor_id)
            where
                        doctor_id between {firstBound} and {secondBound}
            group by
                        doctor_id,
                        doctor_full_name,
                        doctor_specialiazation,
                        hospital_name
            order by doctor_id;",
            [
                "doctor_id",
                "doctor_full_name",
                "doctor_specialiazation",
                "hospital_name",
                "patient_count"
            ]);
        columnName = _customCommand.ColumnNames;
        return result;
    }
}
```

HospitalModel.cs

```
using Npgsql;
using Rgr.Models.Commands;

namespace Rgr.Models.Models;

public sealed class HospitalModel : Model
{
    public HospitalModel(NpgsqlConnection connection) : base(
        connection,
        "hospital",
        "hospital_id",
```

```
        [
            "hospital_name"
        ],
        [
            GenerateCommand.String(),
        ])
    {
    }

    public List<List<object>> ReadAllInfo(int firstBound, int secondBound, out List<string>
columnName)
    {
        var result = _customCommand.Execute(
            $@"select
                        hospital_id,
                        hospital_name,
                        count(doctor_id) as doctor_count
            from
                        hospital
            full join doctor using(hospital_id)
            group by
                        hospital_id,
                        hospital_name
            having
                count(doctor_id) between {firstBound} and {secondBound}
            order by hospital_id;",
            [
                "hospital_id",
                "hospital_name",
                "doctor_count",
            ]);
        columnName = _customCommand.ColumnNames;
        return result;
    }
}
```

IllnesModel.cs

```
using Npgsql;
using Rgr.Models.Commands;

namespace Rgr.Models.Models;

public sealed class IllnessModel : Model
{
    public IllnessModel(NpgsqlConnection connection) : base(
        connection,
        "illness",
        "illness_id",
        [
```

```
            "illness_name",
            "symptoms"
        ],
        [
            GenerateCommand.String(),
            GenerateCommand.String(),
        ])
    {
    }

    public List<List<object>> ReadFullInfo(string likeExp, out List<string> columnName)
    {
        var result = _customCommand.Execute(
            $@"select
                        illness_id,
                        illness_name,
                symptoms,
                        count(patient_id) as patient_count
            from
                        illness
            full join illness_patient using(illness_id)
            where
                illness_name like '{likeExp}'
            group by
                        illness_id,
                        illness_name,
                symptoms
            order by illness_id;",
            [
                "illness_id",
                "illness_name",
                "symptoms",
                "patient_count"
            ]);
        columnName = _customCommand.ColumnNames;
        return result;
    }
}
```

PatienModel.cs

```
using Npgsql;
using Rgr.Models.Commands;

namespace Rgr.Models.Models;

public sealed class PatientModel : Model
{
    public PatientModel(NpgsqlConnection connection) : base(
        connection,
```

```
        "patient",
        "patient_id",
        [
            "patient_full_name",
            "date_of_hospitalization",
            "date_of_discharge"
        ],
        [
            GenerateCommand.String(),
            GenerateCommand.DateOnly(),
            GenerateCommand.DateOnly(),
        ])
    {
    }

    public List<List<object>> ReadFullInfo(string likeExp, out List<string> columnName)
    {
        var result = _customCommand.Execute(
            $@"select
                        illness_id,
                        illness_name,
                symptoms,
                        count(patient_id) as patient_count
            from
                        illness
            full join illness_patient using(illness_id)
            where
                illness_name like '{likeExp}'
            group by
                        illness_id,
                        illness_name,
                symptoms
            order by illness_id;",
            [
                "illness_id",
                "illness_name",
                "symptoms",
                "patient_count"
            ]);
        columnName = _customCommand.ColumnNames;
        return result;
    }
}
```

Controller.cs

```
using Rgr.Models.Models;
using Rgr.Mvc.Views;
using Spectre.Console;
using System.Collections.Generic;
```

```
namespace Rgr.Mvc.Controllers;

public abstract class Controller
{
    protected readonly Model _model;

    protected List<object> _values;

    protected int _id;

    public Controller(Model model)
    {
        _model = model;
        _values = new List<object>();
    }

    public void Read()
    {
        try
        {
            View.PrintTable(_model.TableName, _model.KeyColumnName, _model.ColumnNames.ToList(),
_model.Read());
        }
        catch (Exception ex)
        {
            View.PrintError(ex);
        }
    }

    public void Create()
    {
        try
        {
            _model.Create(_values);
            View.PrintCreate(_model.TableName);
        }
        catch (Exception ex)
        {
            View.PrintError(ex);
        }
    }

    public void Update()
    {
        try
        {
            _model.Update(_values, _id);
            View.PrintUpdate(_model.TableName, _id);
        }
```

```
        catch (Exception ex)
        {
            View.PrintError(ex);
        }
    }

    public void Delete()
    {
        try
        {
            _model.Delete(_id);
            View.PrintDelete(_model.TableName);
        }
        catch (Exception ex)
        {
            View.PrintError(ex);
        }
    }

    public void Generate()
    {
        int count = AnsiConsole.Prompt(
                new TextPrompt<int>("Enter count to generate")
                .ValidationErrorMessage("Not valid value"));

        try
        {
            for (int i = 0; i < count; i++)
                _model.Generate();

            View.PrintGenerate(_model.TableName, count);
        }
        catch (Exception ex)
        {
            View.PrintError(ex);
        }
    }
}
```

MainMenu.cs

```
namespace Rgr.Console;

public enum MainMenu
{
    DoctorMenu,
    HospitalMenu,
    IllnessMenu,
    PatientMenu,
    Exit
```

```
}
```

TableMenu.cs

```
namespace Rgr.Console;

public enum TableMenu
{
    Create,
    Read,
    Update,
    Delete,
    Generate,
    CustomRead,
    Back
}
```

Program.cs

```
using Npgsql;
using Rgr.Console;
using Rgr.Models.Models;
using Rgr.Mvc.Controllers;
using Spectre.Console;
using System.Text;

string connectionString = "host=localhost;port=5433;database=Lab1;user
id=postgres;password=pass12345";
var connection = new NpgsqlConnection(connectionString);

Console.OutputEncoding = Encoding.UTF8;
Console.InputEncoding = Encoding.UTF8;

var currentScene = runMainMenu();

while (currentScene != MainMenu.Exit)
{
    switch (runMainMenu())
    {
        case MainMenu.DoctorMenu:
            {
                DoctorController doctorController = new DoctorController(
                        new DoctorModel(connection));
                switch (runTableMenuWithCustom())
                {
                    case TableMenu.Read:
                        {
                            doctorController.Read();
                        }
                        break;
```

```
            case TableMenu.Create:
                {
                    doctorController.Create();
                }
                break;
            case TableMenu.Update:
                {
                    doctorController.Update();
                }
                break;
            case TableMenu.Delete:
                {
                    doctorController.Delete();
                }
                break;
            case TableMenu.Generate:
                {
                    doctorController.Generate();
                }
                break;
            case TableMenu.Back:
                {
                    currentScene = MainMenu.DoctorMenu;
                }
                break;
            default:
                {
                    doctorController.ReadFullInfo();
                }
                break;
        }
    }
    break;
case MainMenu.HospitalMenu:
    {
        HospitalController hospitalController = new HospitalController(
                    new HospitalModel(connection));
        switch (runTableMenuWithCustom())
        {
            case TableMenu.Read:
                {
                    hospitalController.Read();
                }
                break;
            case TableMenu.Create:
                {
                    hospitalController.Create();
                }
                break;
            case TableMenu.Update:
```

```
            {
                hospitalController.Update();
            }
            break;
        case TableMenu.Delete:
            {
                hospitalController.Delete();
            }
            break;
        case TableMenu.Generate:
            {
                hospitalController.Generate();
            }
            break;
        case TableMenu.Back:
            {
                currentScene = MainMenu.HospitalMenu;
            }
            break;
        default:
            {
                hospitalController.ReadFullInfo();
            }
            break;
    }
  }
  break;
case MainMenu.IllnessMenu:
  {
    IllnessController illnessController = new IllnessController(
                new IllnessModel(connection));
    switch (runTableMenuWithCustom())
    {
        case TableMenu.Read:
            {
                illnessController.Read();
            }
            break;
        case TableMenu.Create:
            {
                illnessController.Create();
            }
            break;
        case TableMenu.Update:
            {
                illnessController.Update();
            }
            break;
        case TableMenu.Delete:
            {
```

```
                illnessController.Delete();
            }
            break;
        case TableMenu.Generate:
            {
                illnessController.Generate();
            }
            break;
        case TableMenu.Back:
            {
                currentScene = MainMenu.IllnessMenu;
            }
            break;
        default:
            {
                illnessController.ReadFullInfo();
            }
            break;
    }
  }
  break;
case MainMenu.PatientMenu:
  {
    PatientController patientController = new PatientController(
            new PatientModel(connection));
    switch (runTableMenu())
    {
        case TableMenu.Read:
            {
                patientController.Read();
            }
            break;
        case TableMenu.Create:
            {
                patientController.Create();
            }
            break;
        case TableMenu.Update:
            {
                patientController.Update();
            }
            break;
        case TableMenu.Delete:
            {
                patientController.Delete();
            }
            break;
        case TableMenu.Generate:
            {
                patientController.Generate();
```

```
                }
                break;
            case TableMenu.Back:
                {
                    currentScene = MainMenu.IllnessMenu;
                }
                break;
        }
      }
      break;
    case MainMenu.Exit:
      return;
  }
}

TableMenu runTableMenu()
{
  return AnsiConsole.Prompt(
    new SelectionPrompt<TableMenu>()
    {
      DisabledStyle = new(Color.SkyBlue1)
    }
    .HighlightStyle(new(Color.Purple4_1))
    .Title("Select operation:")
    .AddChoices( [
      TableMenu.Create,
      TableMenu.Read,
      TableMenu.Update,
      TableMenu.Delete,
      TableMenu.Generate,
      TableMenu.Back
    ])
    );
}

TableMenu runTableMenuWithCustom()
{
  return AnsiConsole.Prompt(
    new SelectionPrompt<TableMenu>()
    {
      DisabledStyle = new(Color.SkyBlue1)
    }
    .HighlightStyle(new(Color.Purple4_1))
    .Title("Select operation:")
    .AddChoices([
      TableMenu.Create,
      TableMenu.Read,
      TableMenu.Update,
      TableMenu.Delete,
      TableMenu.Generate,
```

```
            TableMenu.CustomRead,
            TableMenu.Back
        ]));
}

MainMenu runMainMenu()
{
    return AnsiConsole.Prompt(
        new SelectionPrompt<MainMenu>()
        {
            DisabledStyle = new(Color.SkyBlue1)
        }
        .HighlightStyle(new(Color.Purple4_1))
        .Title("Select table:")
        .AddChoices( [
            MainMenu.DoctorMenu,
            MainMenu.HospitalMenu,
            MainMenu.IllnessMenu,
            MainMenu.PatientMenu,
            MainMenu.Exit
        ]));
}

return;
```

Контакти:

1. Github: https://github.com/IhorShpilka/BD_labs
2. Telegram: https://t.me/ihorshpilka